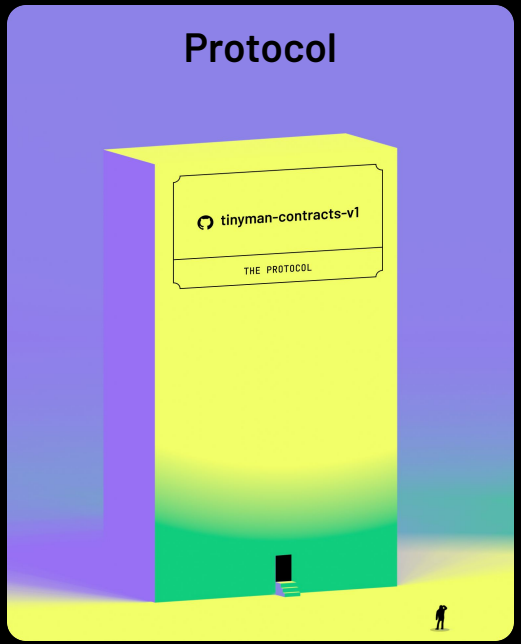
tinyman

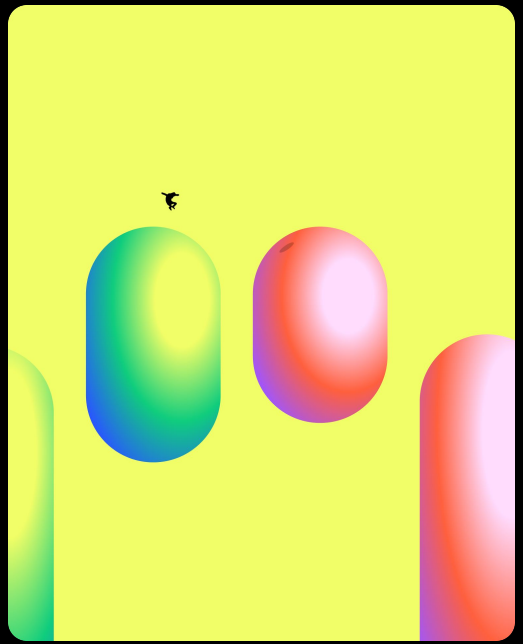
Tinyman

Algorand Developer Office Hours
October 5th 2021

Tinyman is a permissionless trustless decentralized trading protocol on Algorand

Tinyman





Tinyman’s Friends


TINYMAN
BALANCE
A 8 ASSET-A

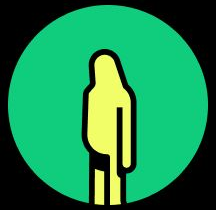

TINYMAN
BALANCE
A 8 ASSET-A


1.00
B
ASSET-B
TINYMAN
BALANCE
A
8 ASSET-A


ACTIVE QUEST
FIND 1.00 ASSET-B
TINYMAN
BALANCE
A
8 ASSET-A

ACTIVE QUEST

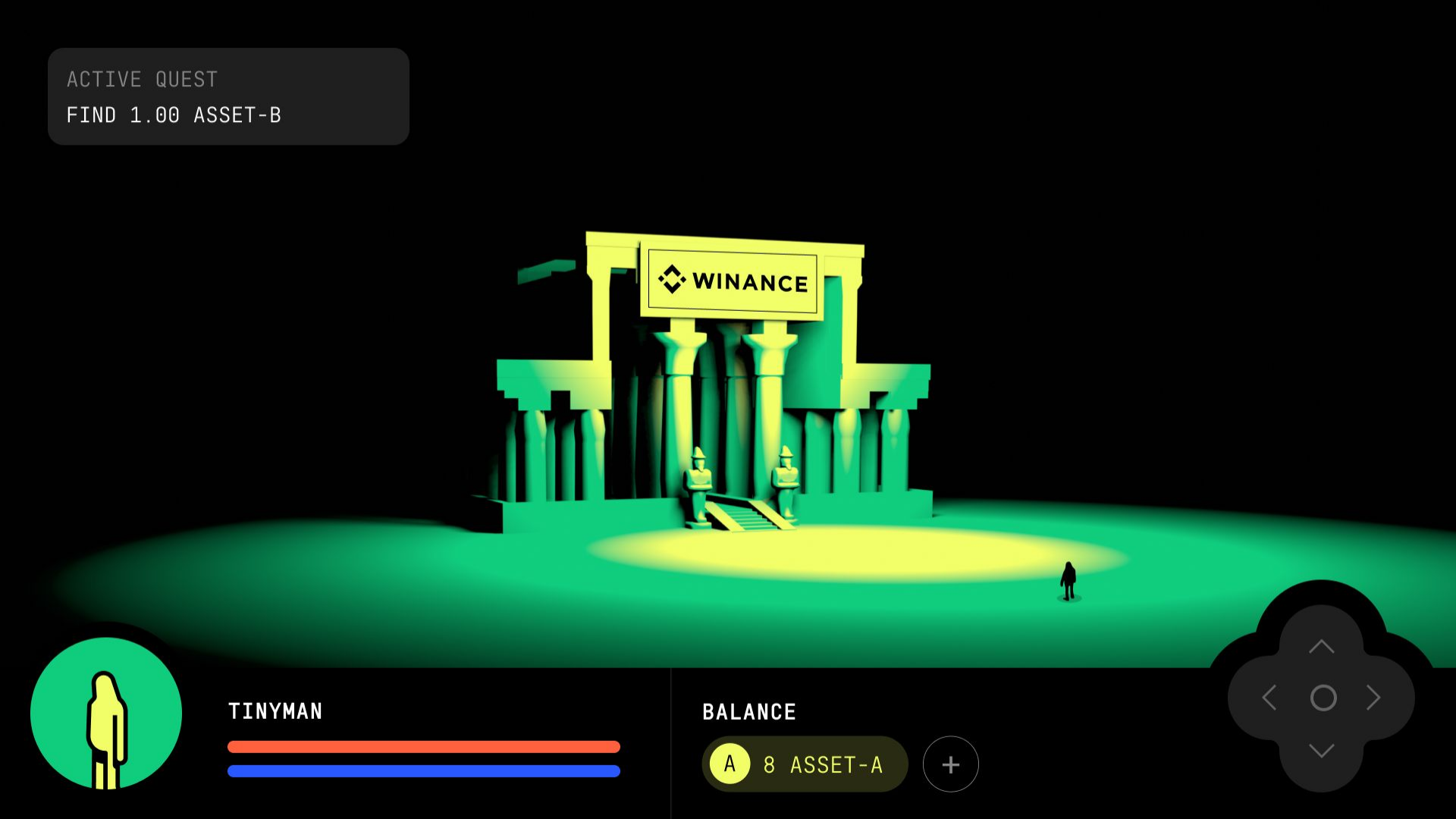
FIND 1.00 B-COIN

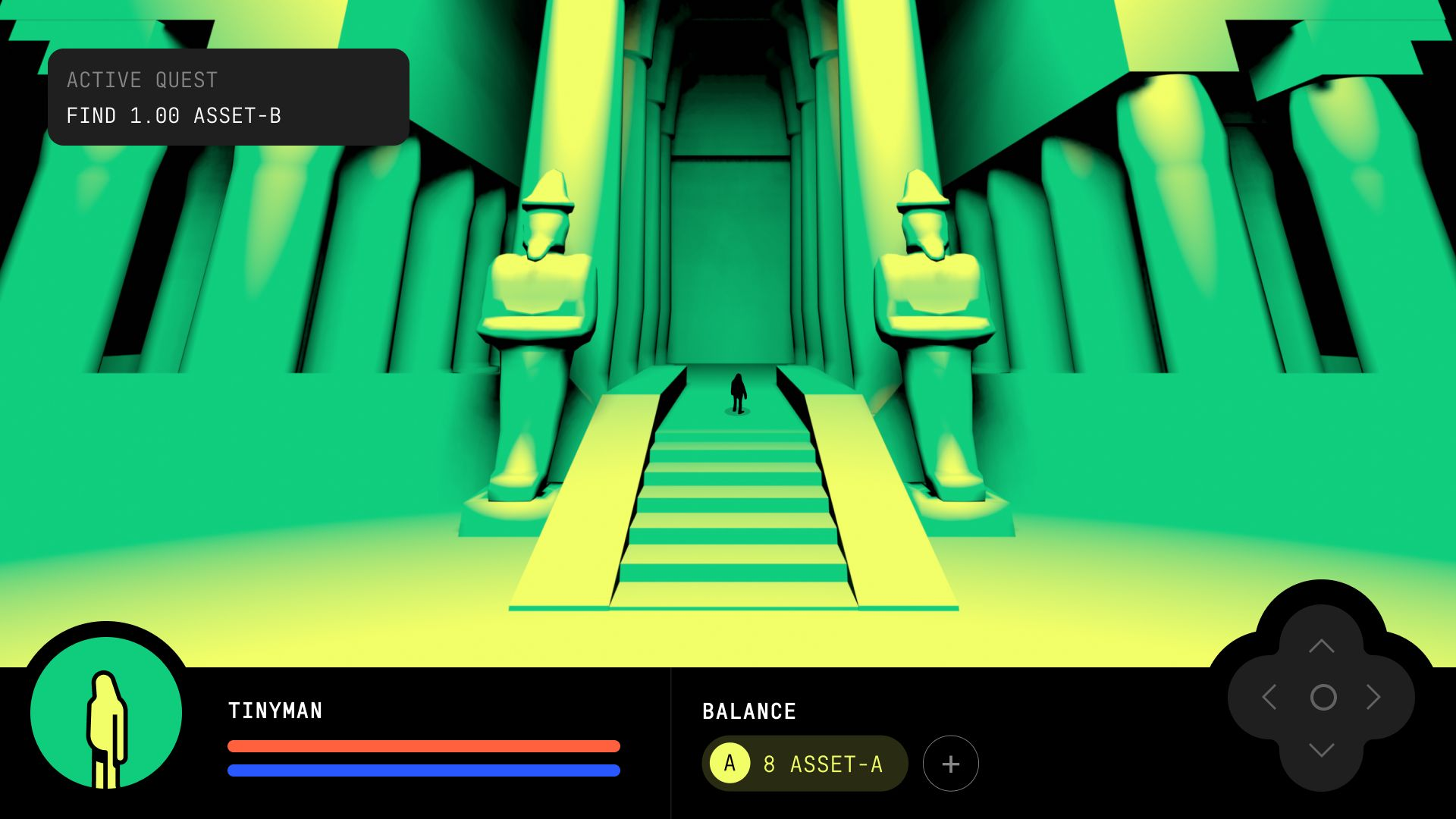


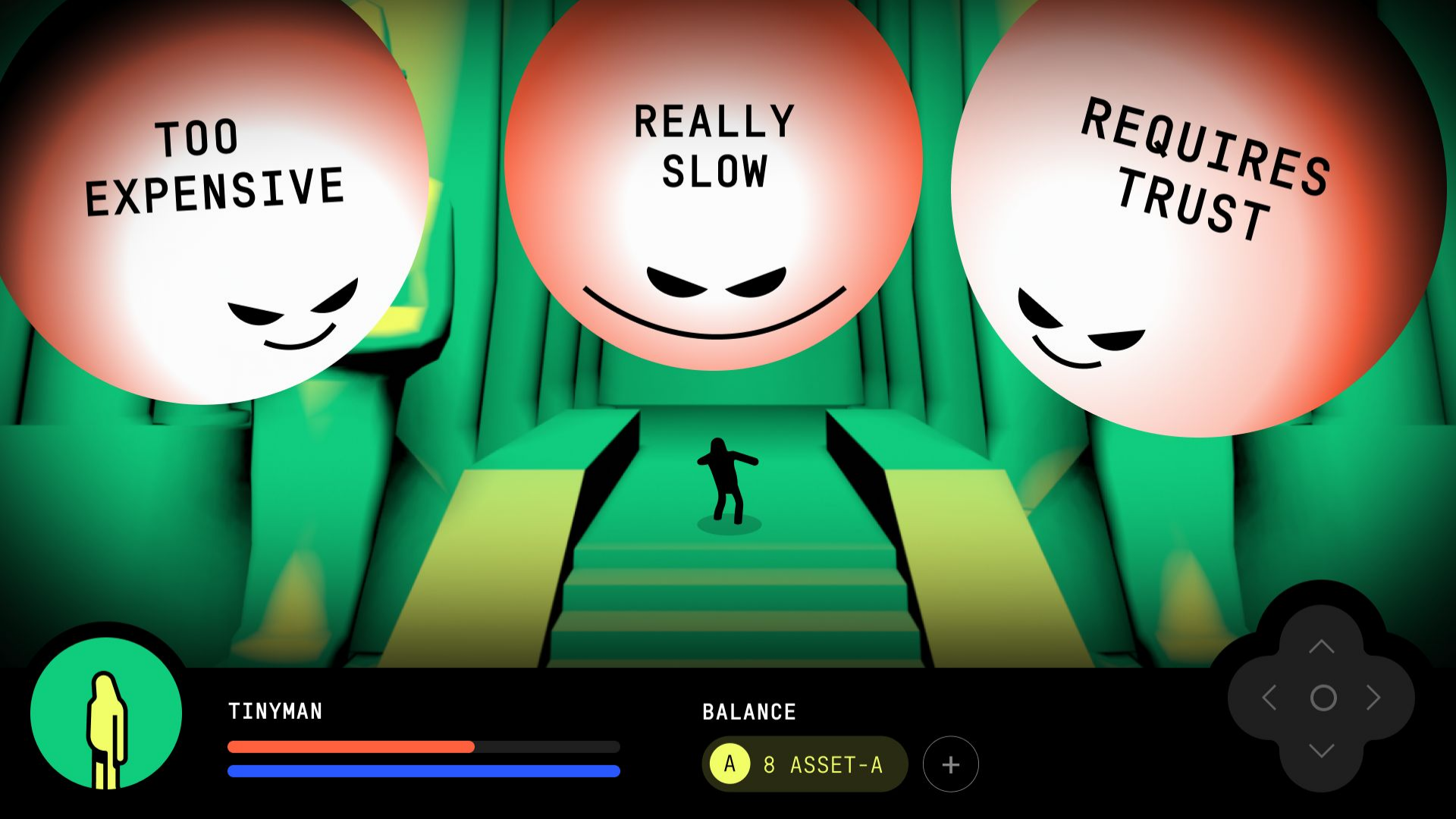
TINYMAN

BALANCE

A 8 ASSET-A


ACTIVE QUEST
FIND 1.00 ASSET-B
WINANCE
TINYMAN
BALANCE
A
8 ASSET-A


ACTIVE QUEST
FIND 1.00 ASSET-B
TINYMAN
BALANCE
A
8 ASSET-A


TOO EXPENSIVE
REALLY SLOW
REQUIRES TRUST
TINYMAN
BALANCE
A 8 ASSET-A

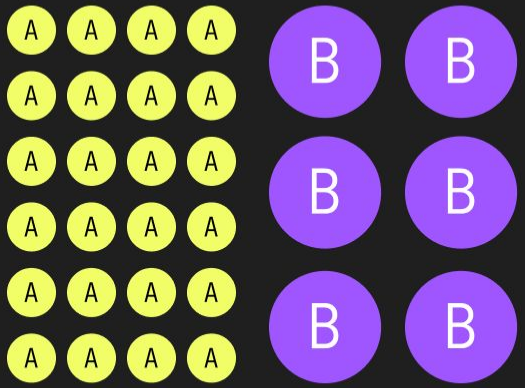

ACTIVE QUEST
FIND 1.00 ASSET-B
TINYMAN
BALANCE
A
8 ASSET-A

ACTIVE QUEST

FIND 1.00 ASSET-B

DECENTRALIZED EXCHANGE

A-B POOL

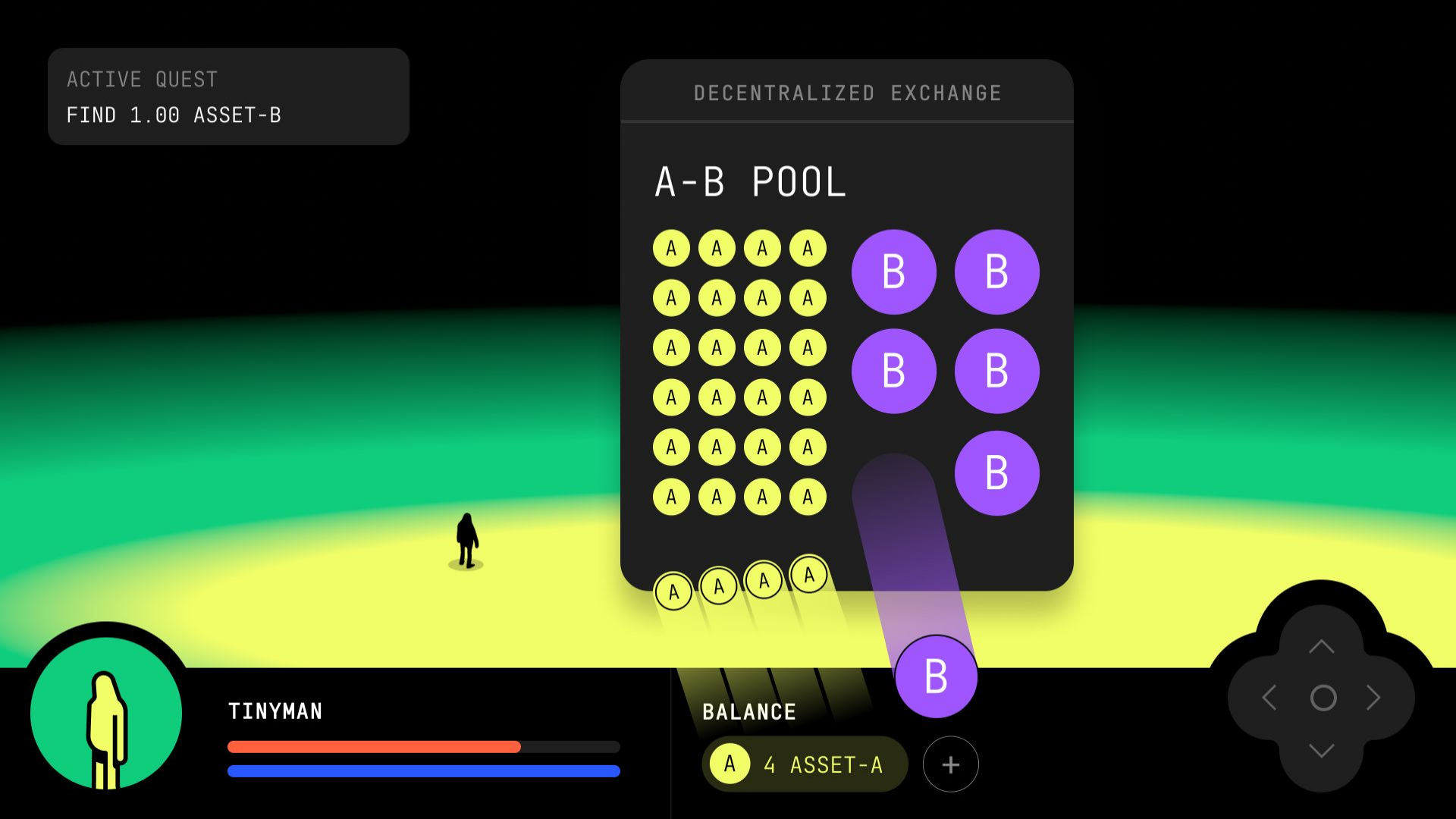


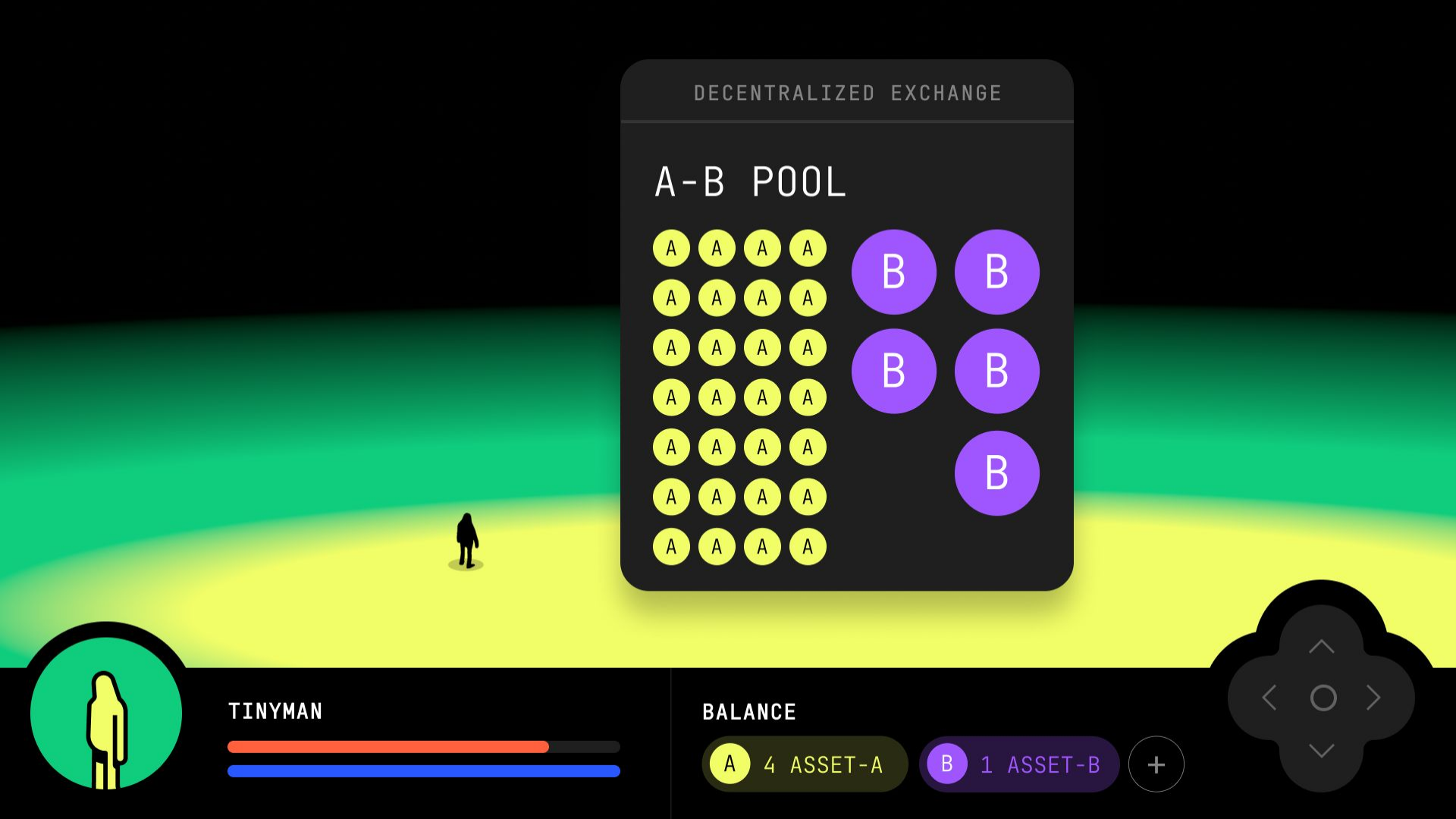
TINYMAN

BALANCE

A 8 ASSET-A

+


ACTIVE QUEST
FIND 1.00 ASSET-B
DECENTRALIZED EXCHANGE
A-B POOL
TINYMAN
BALANCE
4 ASSET-A


DECENTRALIZED EXCHANGE
A-B POOL
A A A A
A A A A
A A A A
A A A A
A A A A
A A A A
A A A A
B B
B B
B
TINYMAN
BALANCE
A 4 ASSET-A
B 1 ASSET-B
+


QUEST COMPLETED
FIND 1.00 ASSET-B
TINYMAN
BALANCE
A 4 ASSET-A
B 1 ASSET-B

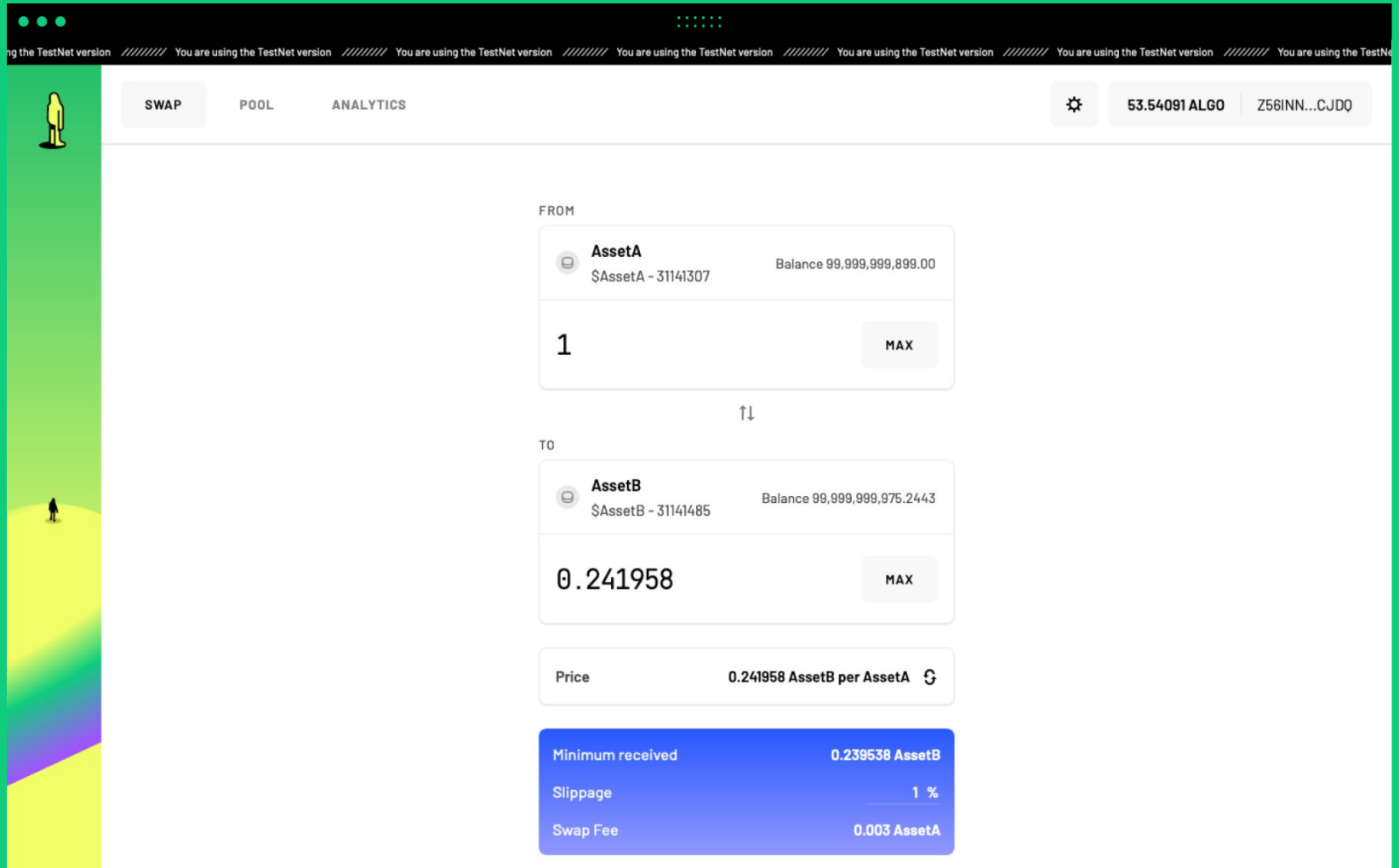
YOU HAVE A NEW ITEM
IN YOUR INVENTORY!

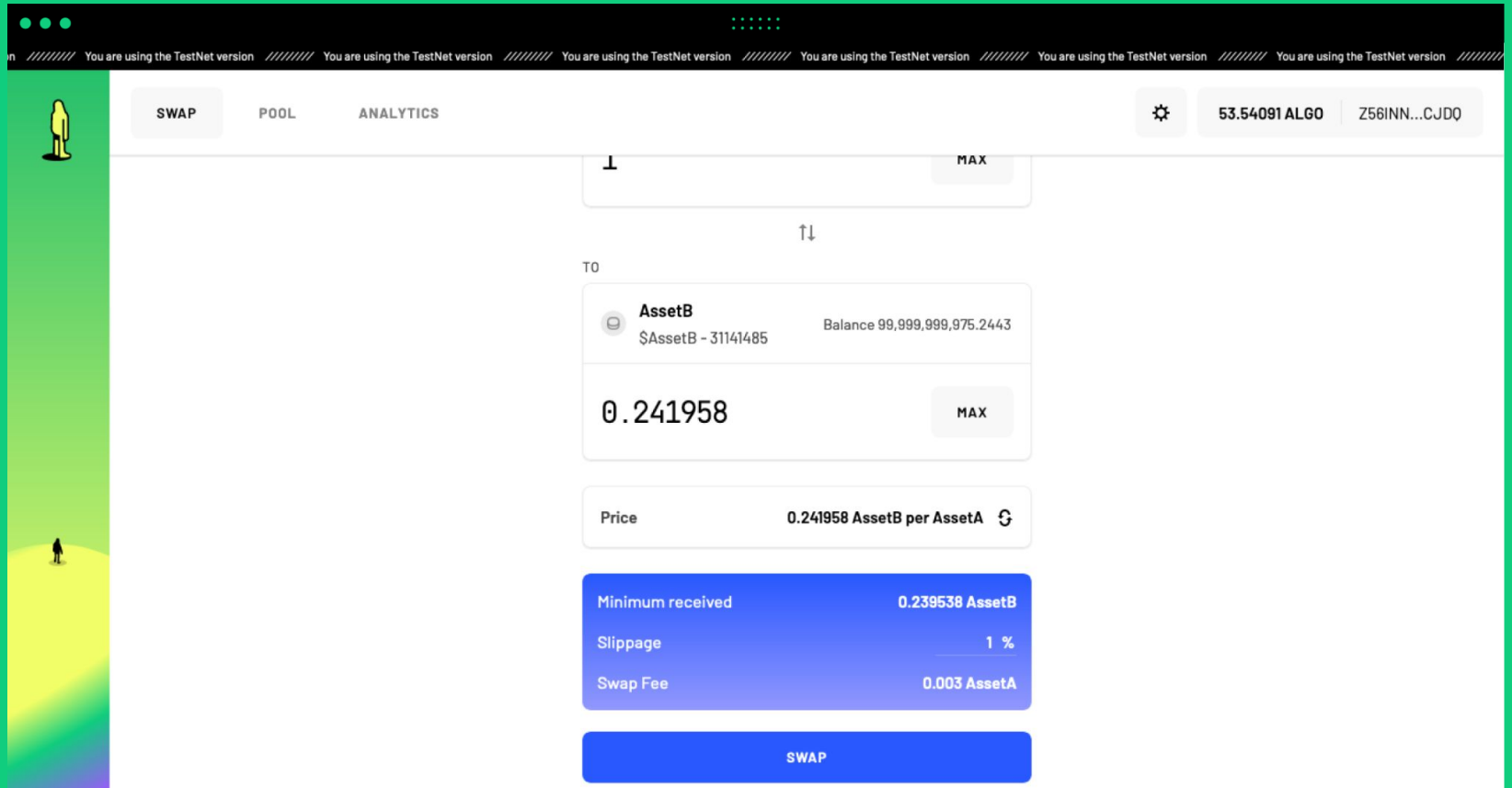
TINYMAN

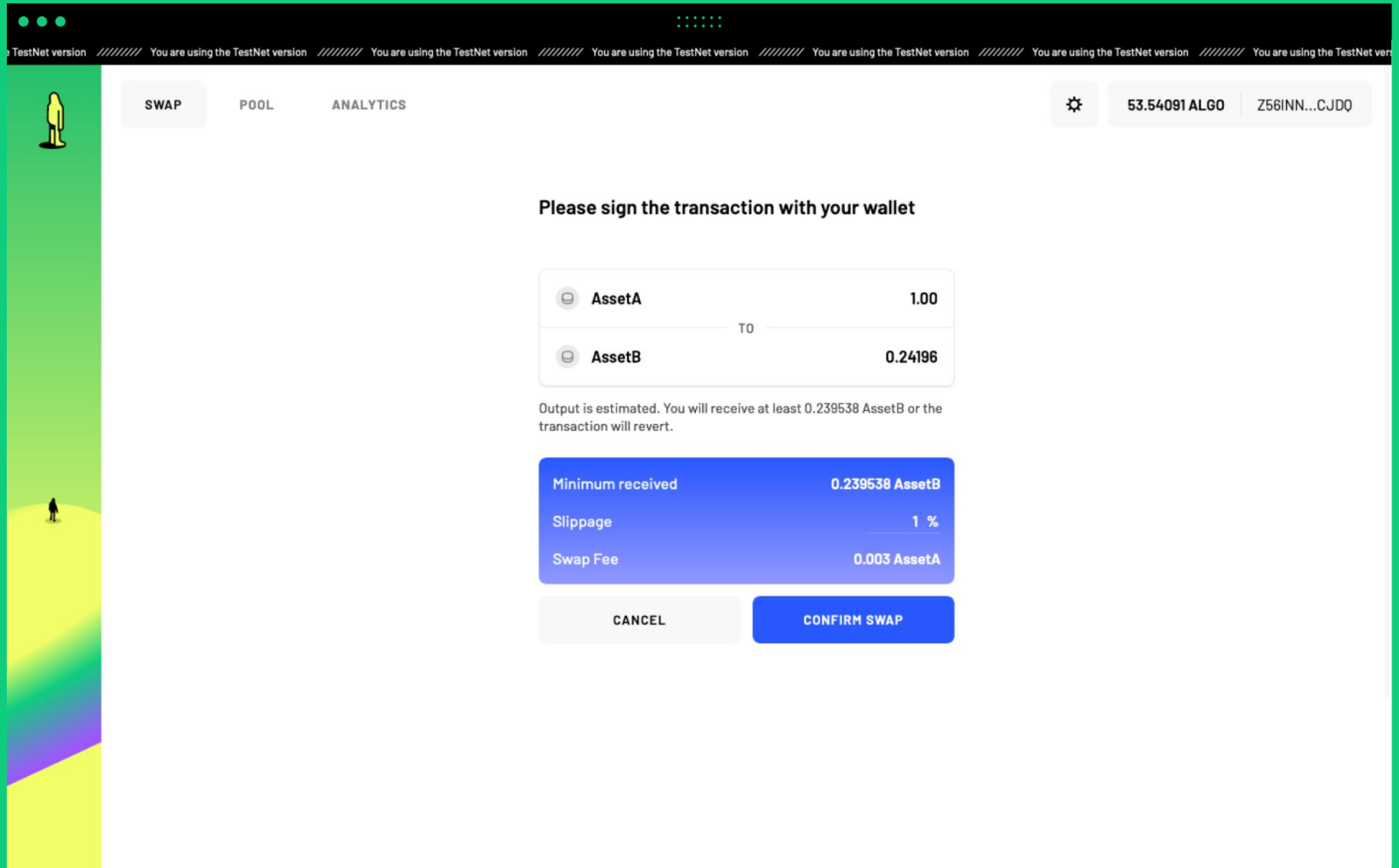
BALANCE

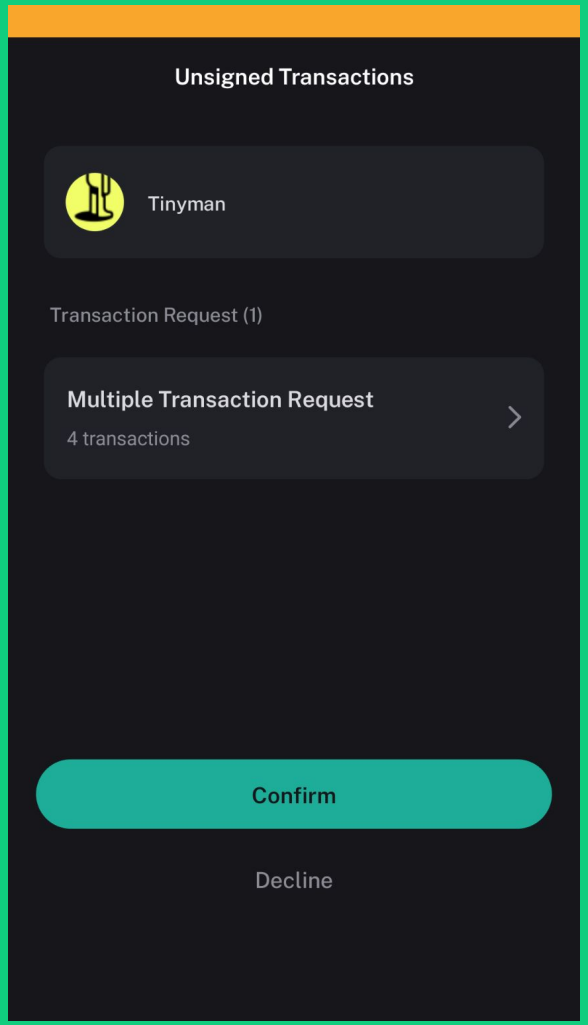
A 4 ASSET-A

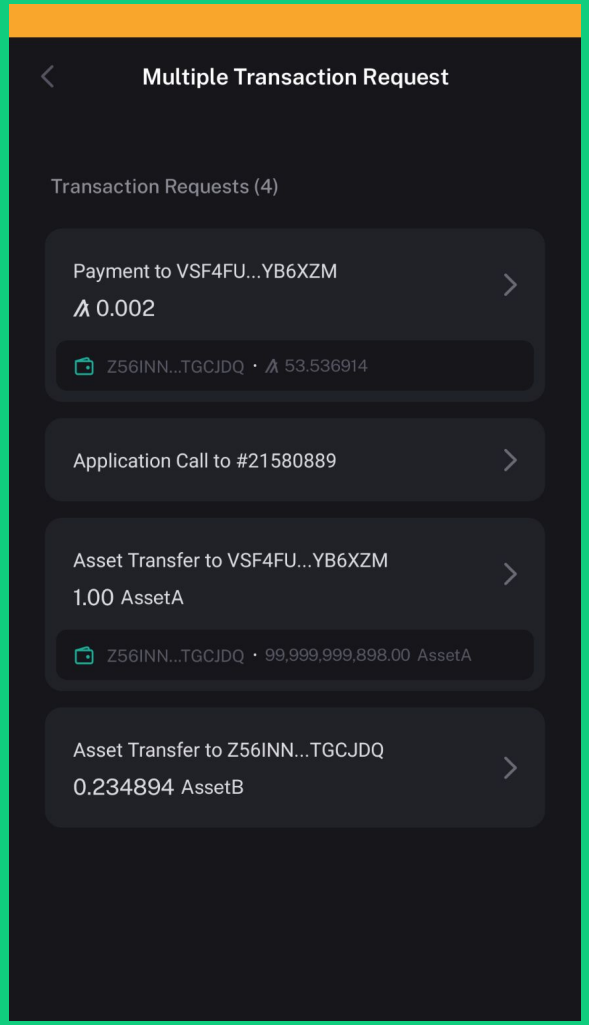
+


You are using the TestNet version
SWAP
POOL
ANALYTICS
53.54091 ALGO
Z56INN...CJD0
FROM
AssetA
$AssetA - 31141307
Balance 99,999,999,899.00
1
MAX
TO
AssetB
$AssetB - 31141485
Balance 99,999,999,975.2443
0.241958
MAX
Price
0.241958 AssetB per AssetA
Minimum received
0.239538 AssetB
Slippage
1 %
Swap Fee
0.003 AssetA


You are using the TestNet version
You are using the TestNet version
You are using the TestNet version
You are using the TestNet version
You are using the TestNet version
You are using the TestNet version
SWAP
POOL
ANALYTICS
53.54091 ALGO
Z56INN...CJD0
MAX
TO
AssetB
$AssetB - 31141485
Balance 99,999,999,975.2443
0.241958
MAX
Price
0.241958 AssetB per AssetA
Minimum received
0.239538 AssetB
Slippage
1 %
Swap Fee
0.003 AssetA
SWAP


TestNet version
You are using the TestNet version
SWAP
POOL
ANALYTICS
53.54091 ALGO
Z56INN...CJD0
Please sign the transaction with your wallet
AssetA
1.00
TO
AssetB
0.24196
Output is estimated. You will receive at least 0.239538 AssetB or the transaction will revert.
Minimum received
0.239538 AssetB
Slippage
1 %
Swap Fee
0.003 AssetA
CANCEL
CONFIRM SWAP


Unsigned Transactions
Tinyman
Transaction Request (1)
Multiple Transaction Request
4 transactions
Confirm
Decline


Multiple Transaction Request
Transaction Requests (4)
Payment to VSF4FU...YB6XZM
0.002
Z56INN...TGCJDQ · 53.536914
Application Call to #21580889
Asset Transfer to VSF4FU...YB6XZM
1.00 AssetA
Z56INN...TGCJDQ · 99,999,999,898.00 AssetA
Asset Transfer to Z56INN...TGCJDQ
0.234894 AssetB

SWAP POOL ANALYTICS

53.53691 ALGO Z56INN...CJD0

AssetA / AssetB swap completed

AssetA	1.00
TO	
AssetB	0.24196

The amounts shown reflect the final swap values including excess amounts which must be redeemed separately.

Transaction ID

5HHWX62S...ZMD2B2HQ

View on AlgoExplorer ↗

You have some excess AssetB tokens that you can claim now or later from the Accounts page.

REDEEM 0.004886 AssetB

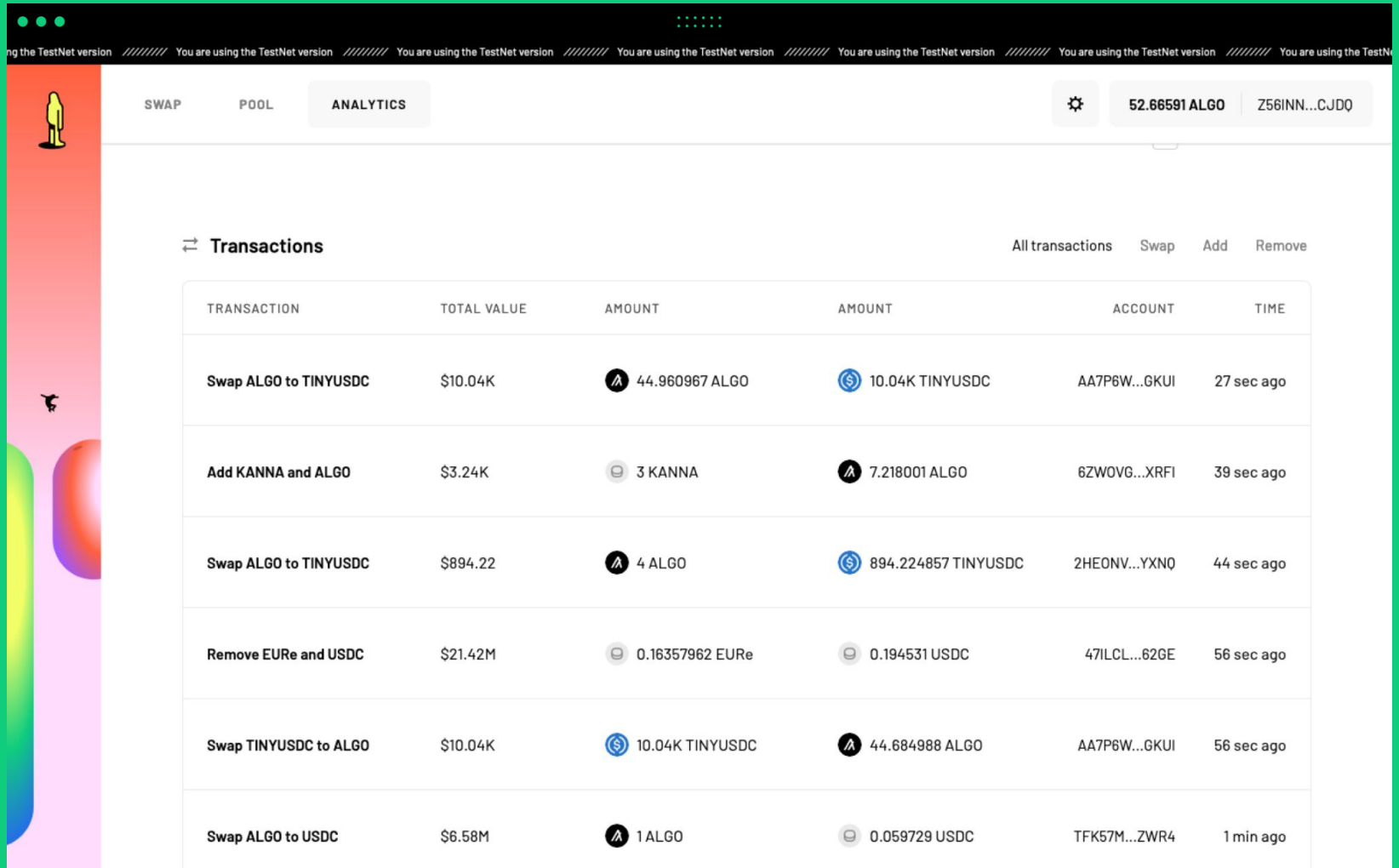
TINYMAN ANALYTICS

Total Liquidity	Volume (24h)	$ALGO Price (24h)
$ 4,703,740,968,521,806	$ 473,493,552,054.93	$ 224.77 16.52%

Search a name or paste address

Assets

TOKEN	LIQUIDITY	VOLUME (24H)	PRICE	24H CHANGE
Algorand $ALGO	$130.68M	$22.08M	≈ $224.77	16.52% ↗
USDC	$2.47T	$472.59B	≈ $110,105,283.42	-


You are using the TestNet version
SWAP
POOL
ANALYTICS
52.66591 ALGO
Z56INN...CJDQ
Transactions
All transactions
Swap
Add
Remove
TRANSACTION
TOTAL VALUE
AMOUNT
AMOUNT
ACCOUNT
TIME
Swap ALGO to TINYUSDC
$10.04K
44.960967 ALGO
10.04K TINYUSDC
AA7P6W...GKUI
27 sec ago
Add KANNA and ALGO
$3.24K
3 KANNA
7.218001 ALGO
6ZW0VG...XRFI
39 sec ago
Swap ALGO to TINYUSDC
$894.22
4 ALGO
894.224857 TINYUSDC
2HE0NV...YXNQ
44 sec ago
Remove EURe and USDC
$21.42M
0.16357962 EURe
0.194531 USDC
47ILCL...62GE
56 sec ago
Swap TINYUSDC to ALGO
$10.04K
10.04K TINYUSDC
44.684988 ALGO
AA7P6W...GKUI
56 sec ago
Swap ALGO to USDC
$6.58M
1 ALGO
0.059729 USDC
TFK57M...ZWR4
1 min ago

» Why Algorand?

Fast

Cheap

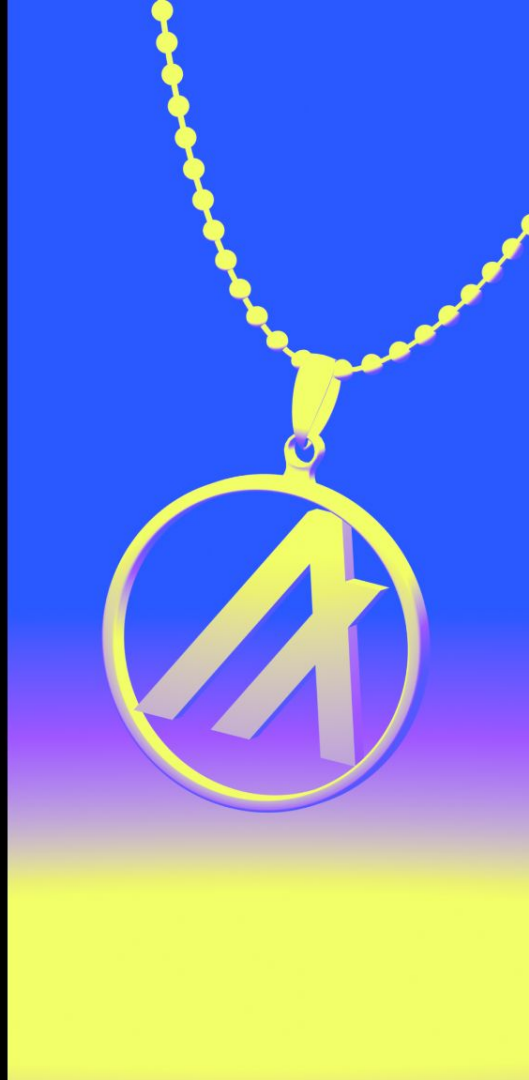
Secure

Reliable

Greeen

No “front running”

Low possibility of MEV (Miner Extractable Value)



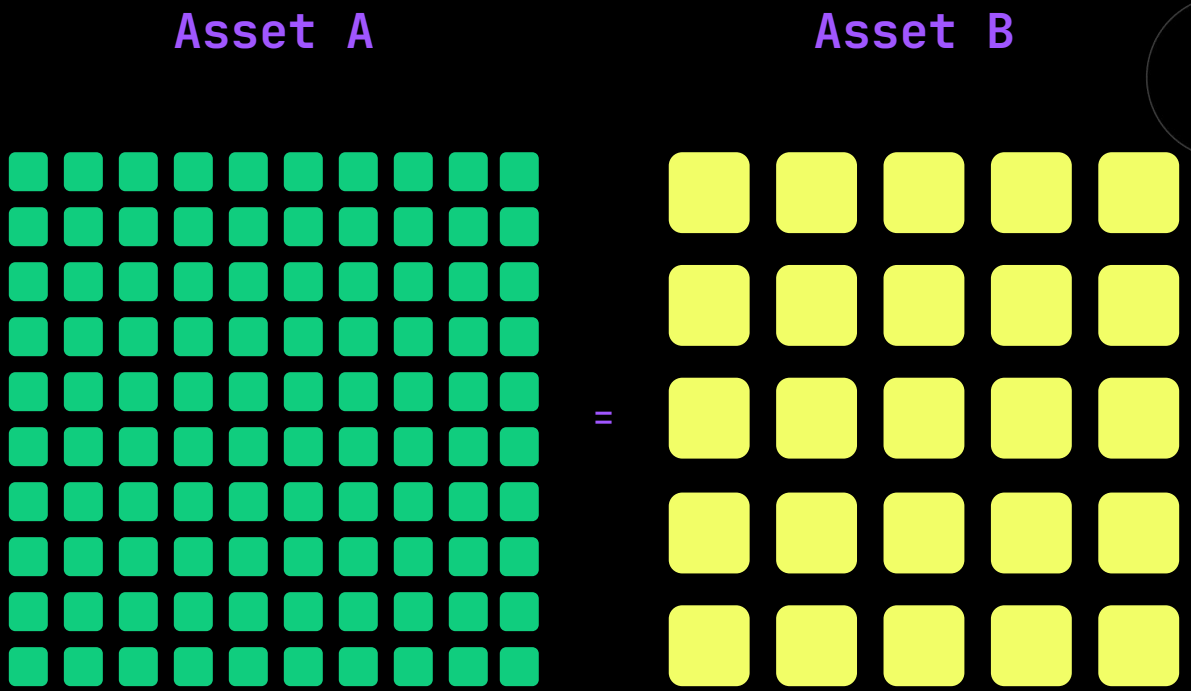
Basics

AMM DEX

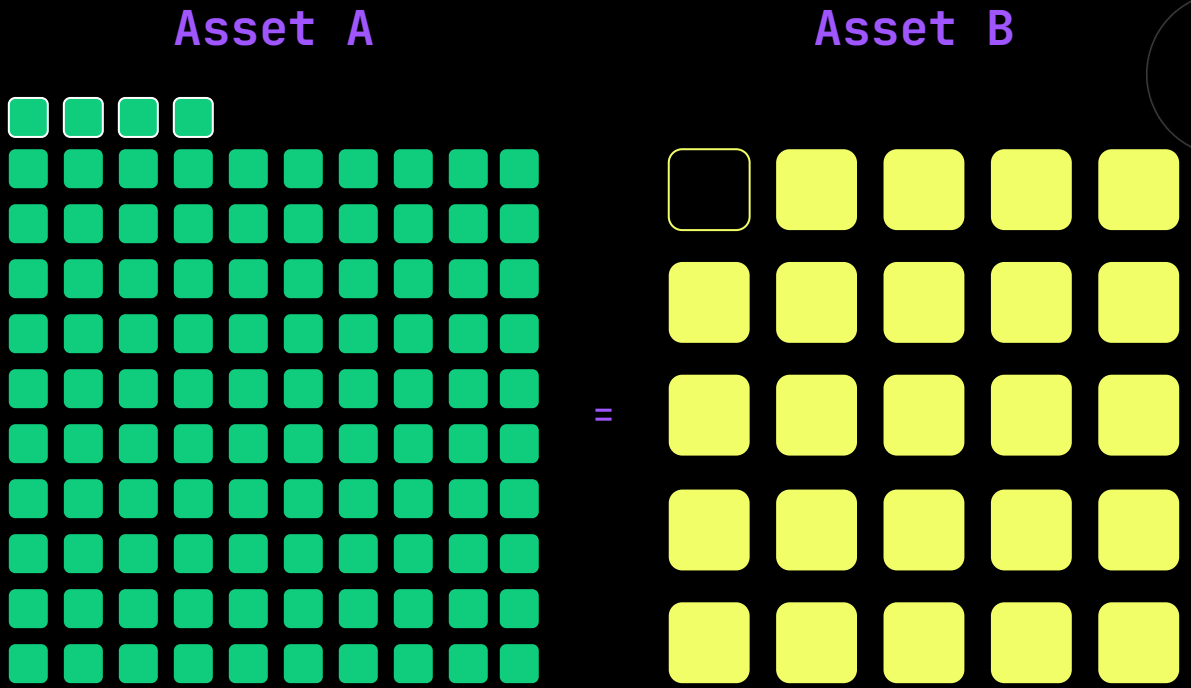
AMM - Automated Market Maker

DEX - Decentralised Exchange

Pool



Pool



» AMM

Automated Market Maker

Constant Product Market Maker

$$x * y = k$$

```
(asset_1_reserves * asset_2_reserves) == (new_asset_1_reserves * new_asset_2_reserves)
```

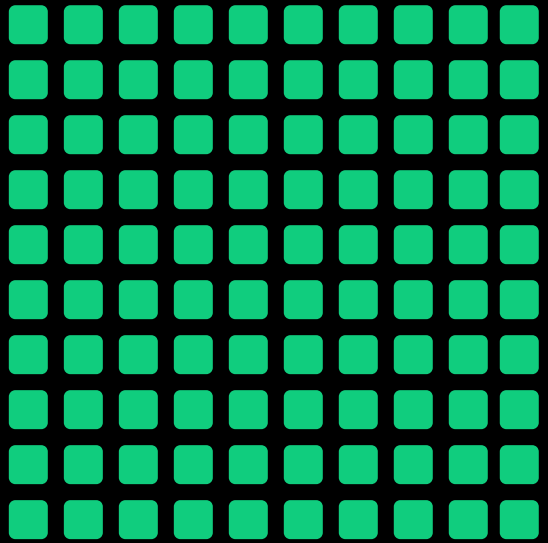
The product of the reserves of asset 1 and asset2 must remain constant

Swaps must not cause the pool to lose value

» Pool

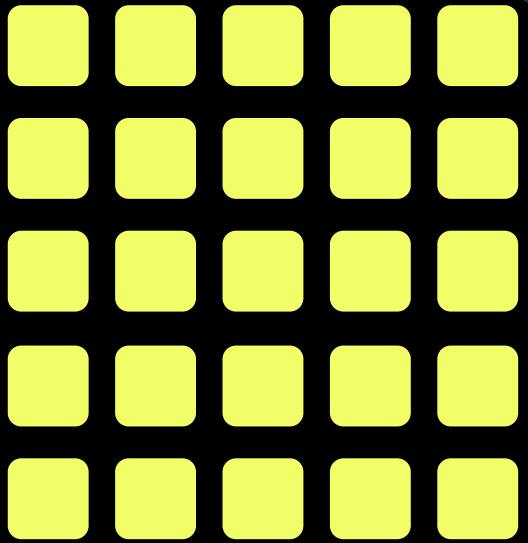
1A : 0.25B

Asset A



100

Asset B

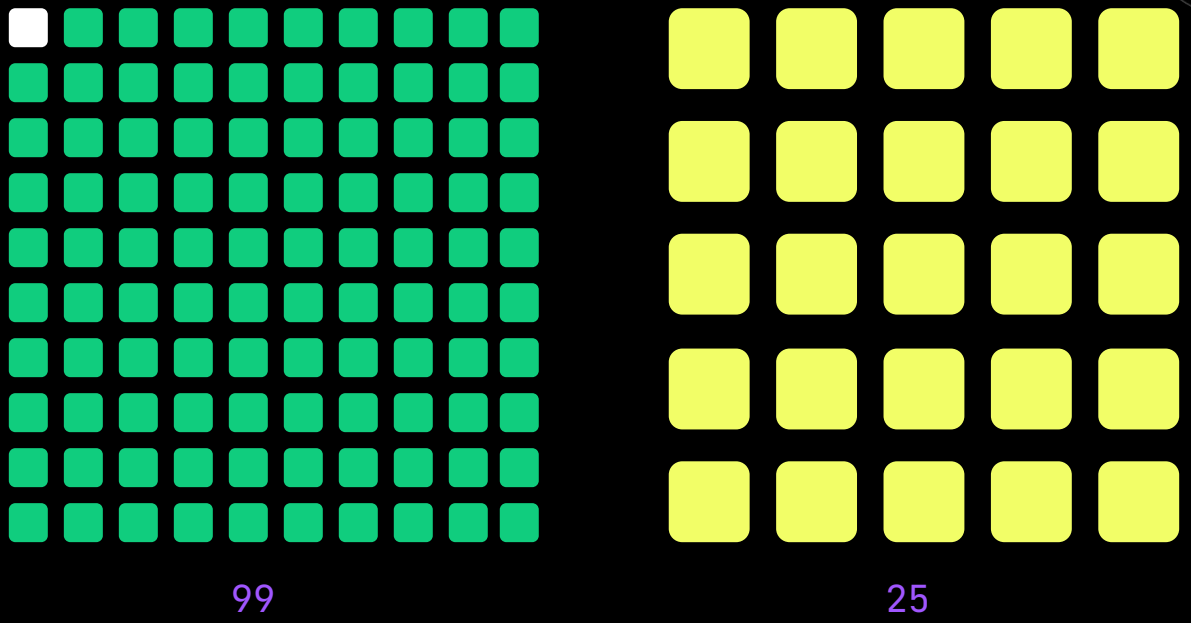


25

Product = 100 x 25 = 2500

» Pool

Buy 1A
Sell ?B

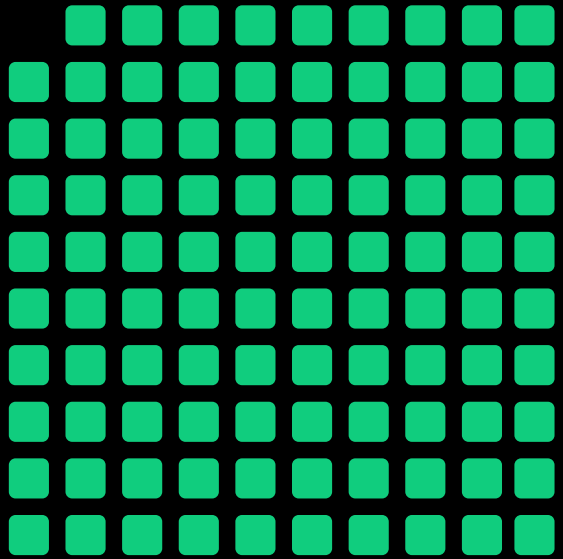


Product = 99 x 25 = 2475

» Pool

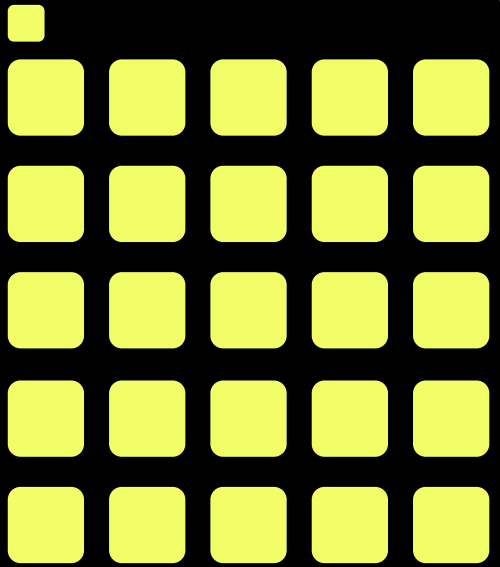
Buy 1A
Sell 0.25B
Rate: 1A:0.25B

Asset A



99

Asset B

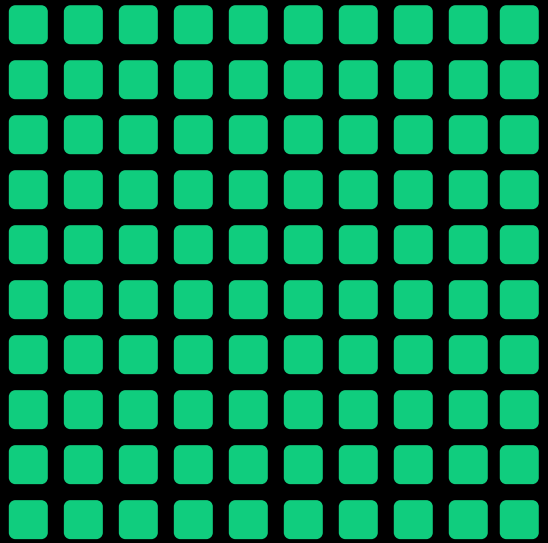


25.25

Product = 99 x 25.25 = 2500

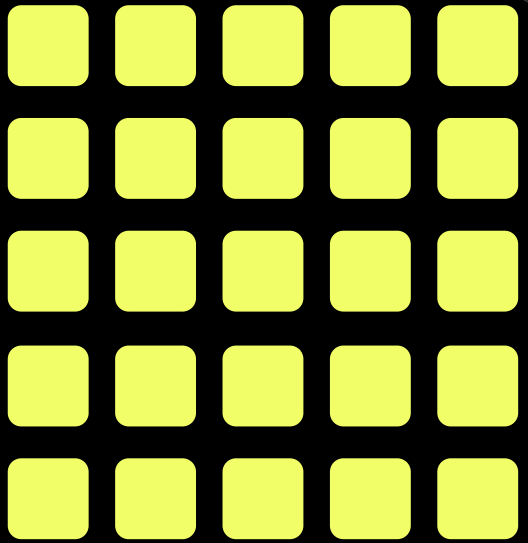
Pool

Asset A



100

Asset B

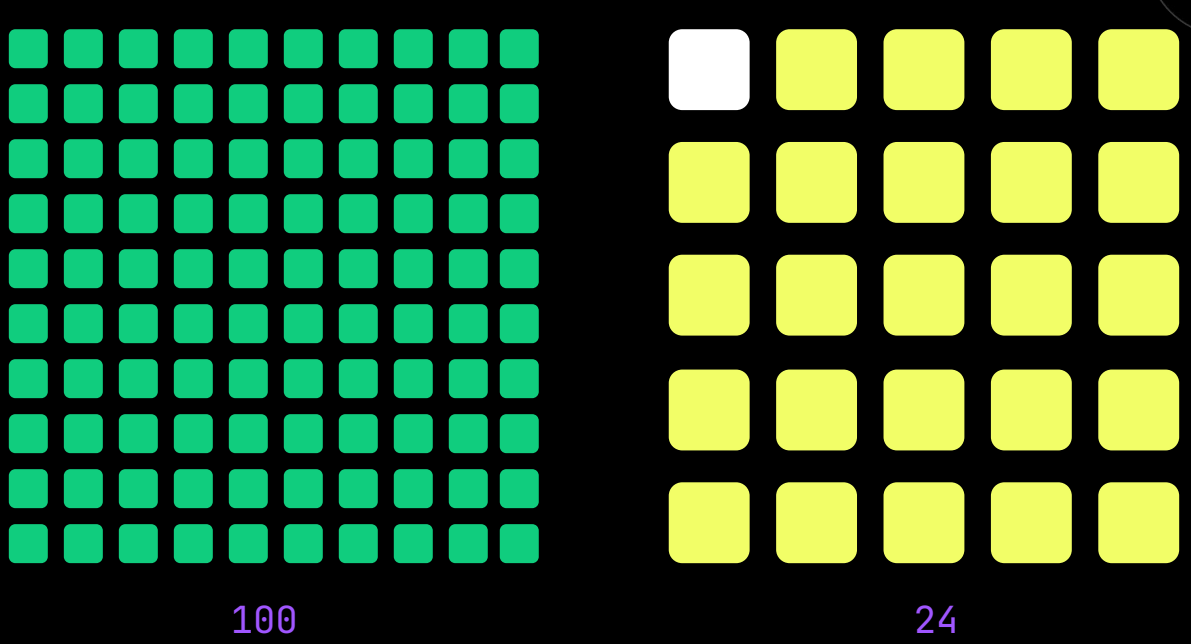


25

Product = 100 x 25 = 2500

» Pool

Buy 1B
Sell ?A

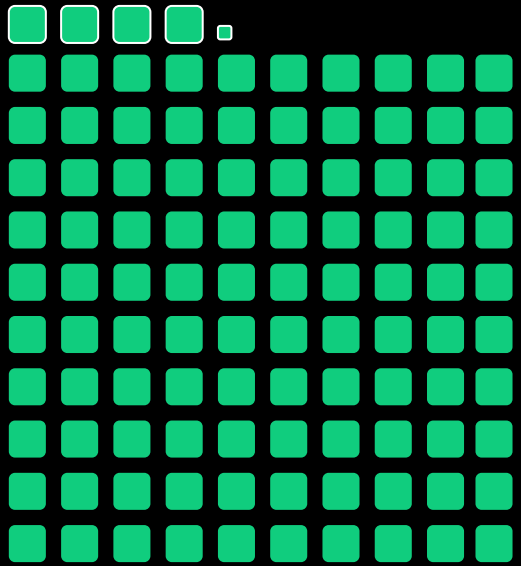


Product = 100 x 24 = 2400

Pool

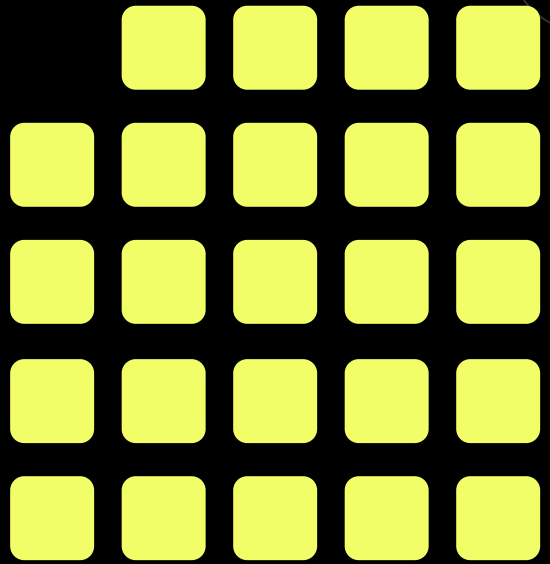
Buy 1B
Sell 4.2A
Rate: 1:0.23B

Asset A



104.2

Asset B



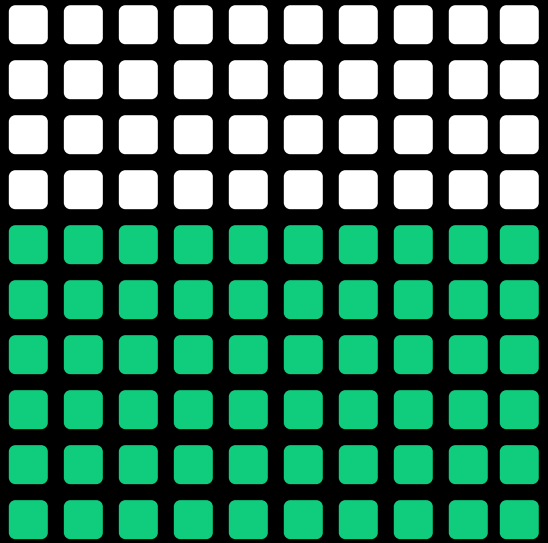
24

Product = 104.2 x 24 = 2500

» Pool

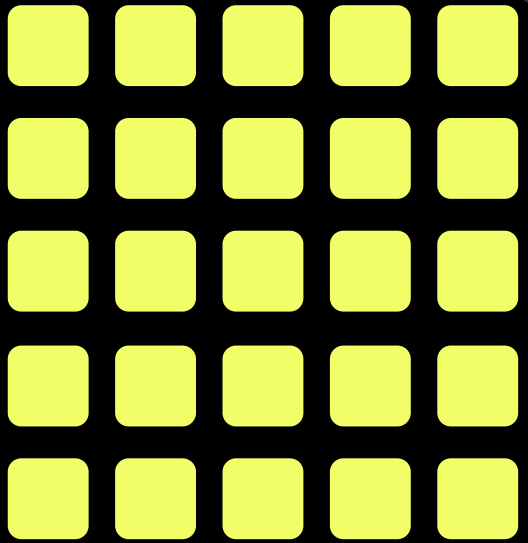
Buy 40A
Sell ?B

Asset A



60

Asset B

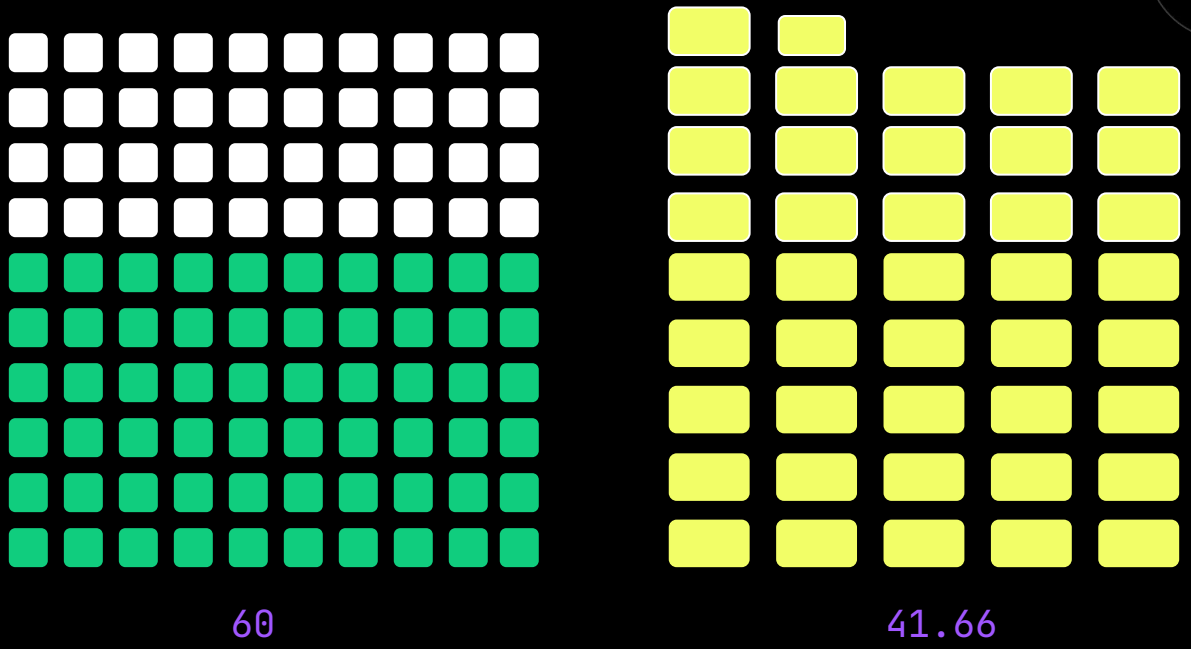


25

Product = 60 x 25 = 1500

» Pool

Buy 40A
Sell 16.66B
Rate: 1B:2.5A

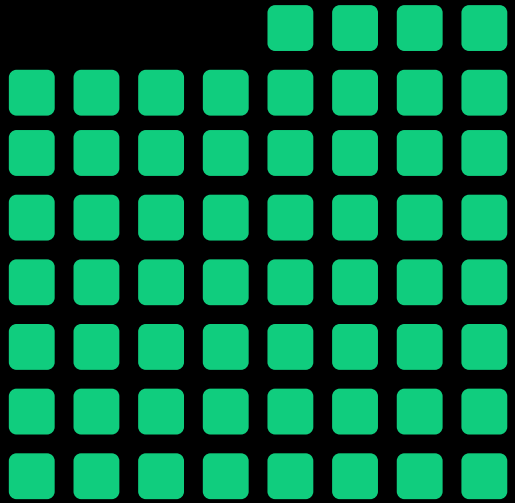


Product = 60 x 41.66 = 2500

» Pool

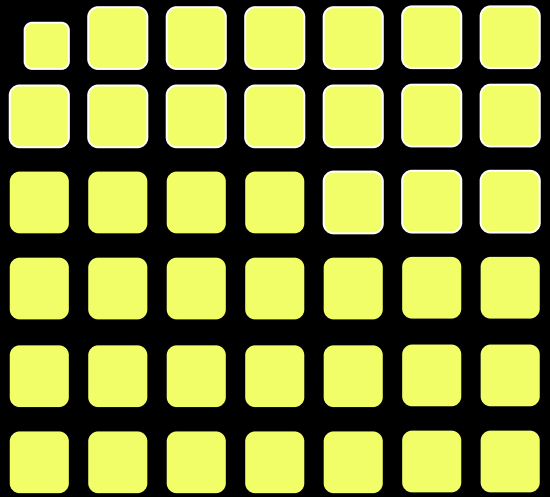
Buy 40A
Sell 16.66B
Rate: 1B:2.5A

Asset A



60

Asset B

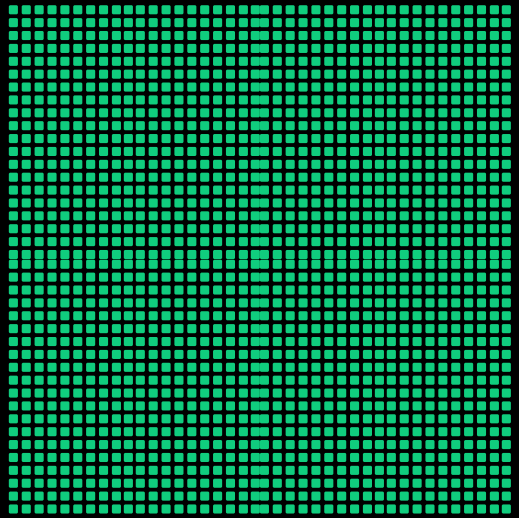


41.66

Product = 60 x 41.66 = 2500

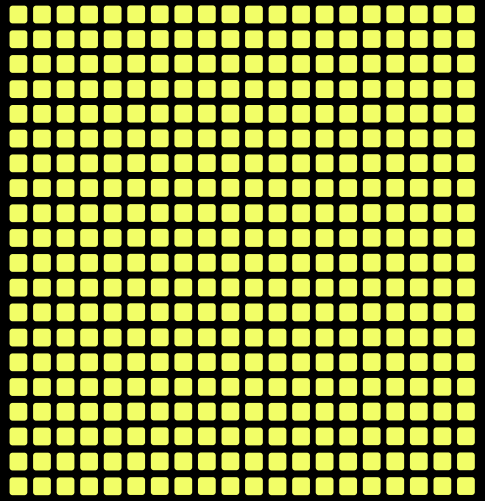
» Pool

Asset A



10000

Asset B

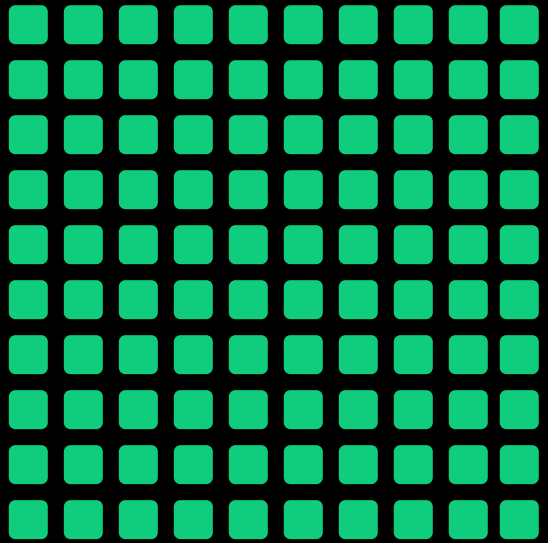


2500

Product = 10000 x 2500 = 25000000

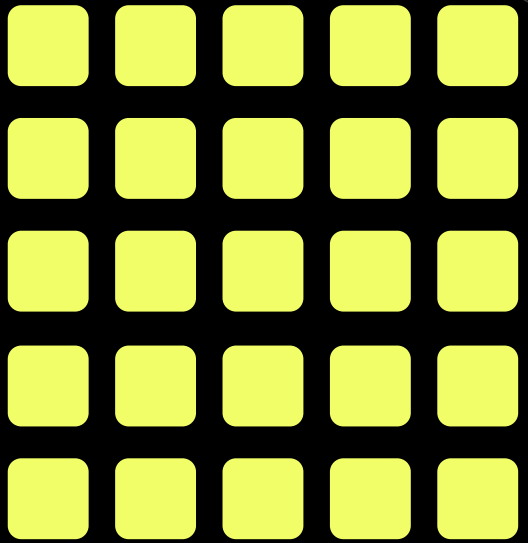
Poolers

Asset A

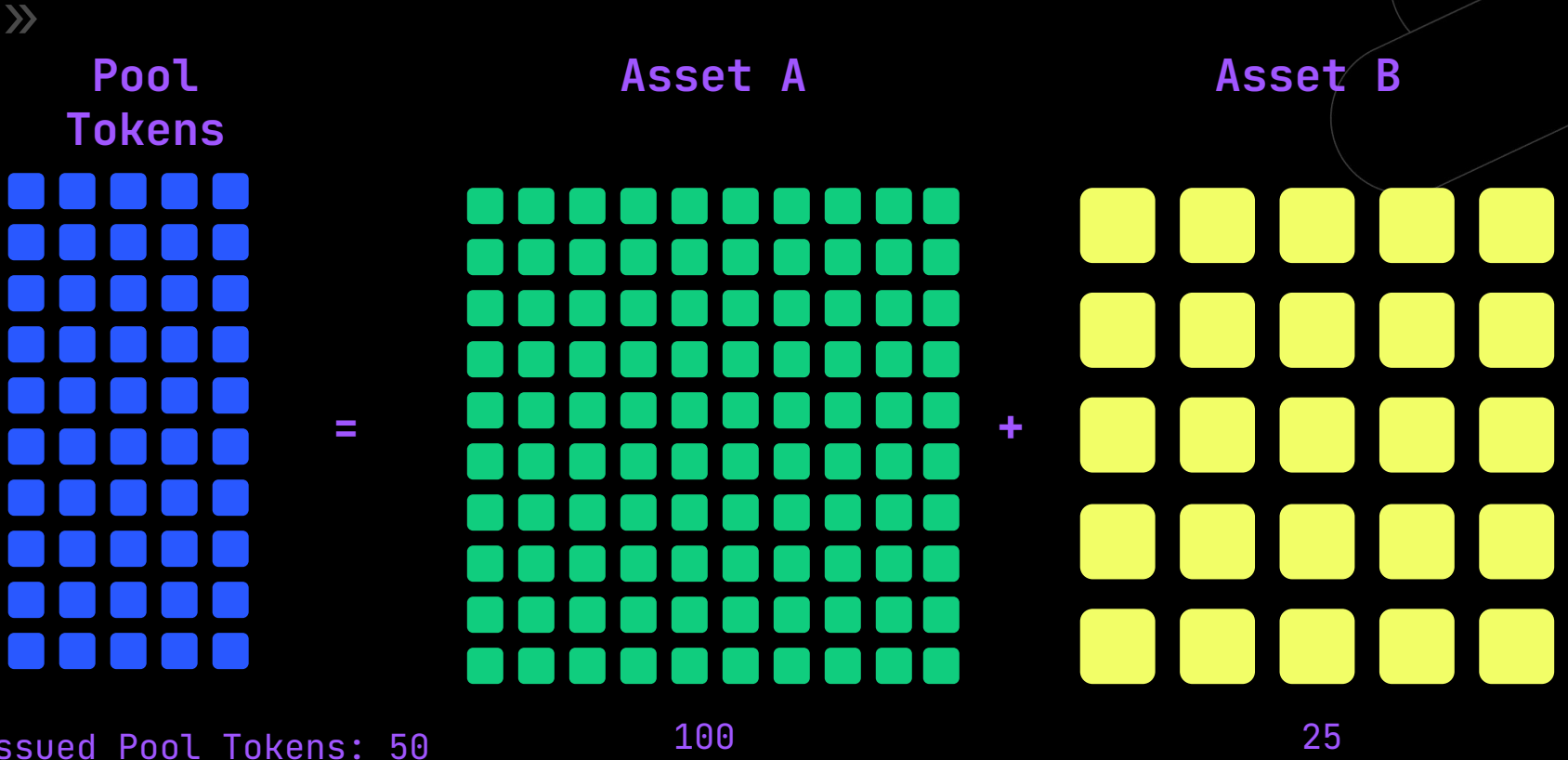


100

Asset B

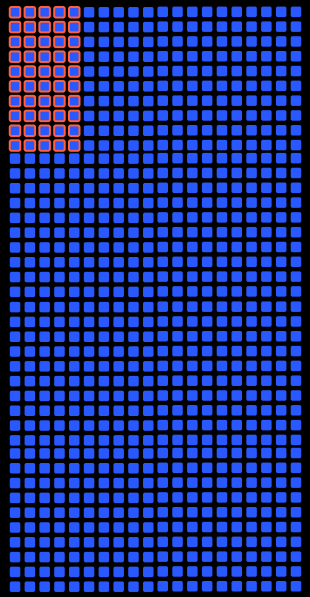


25



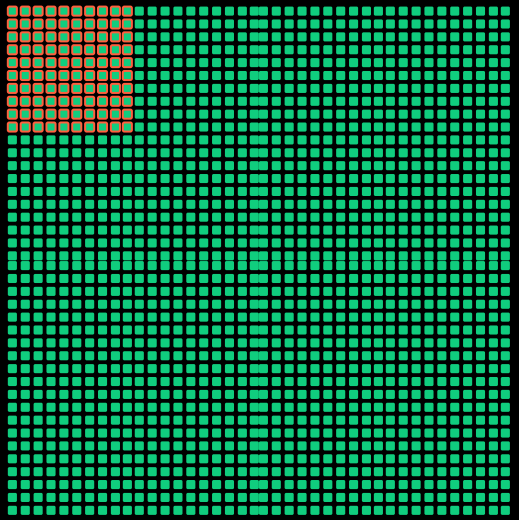
Issued Pool Tokens: 50
Held Pool Tokens: 50
Share of Pool: 100%

Pool Tokens

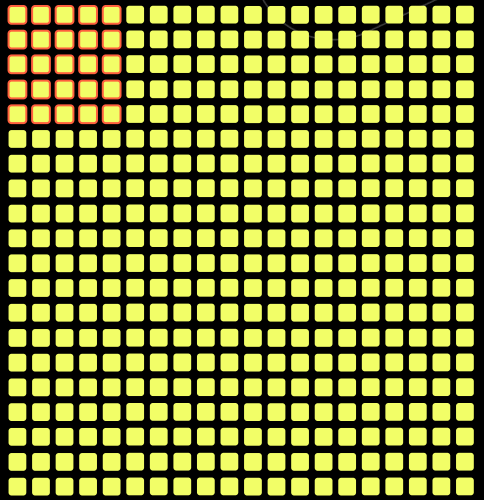


=

Asset A

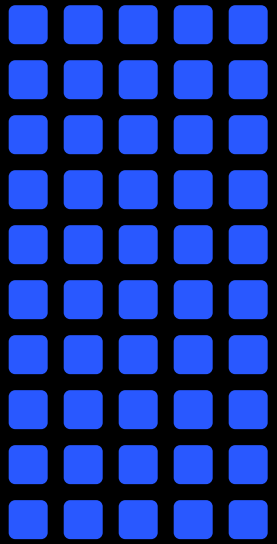


Asset B



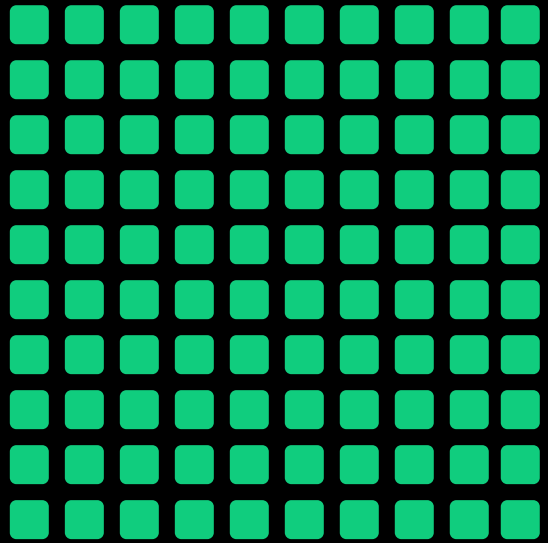
Issued Pool Tokens: 800
Held Pool Tokens: 50
Share of Pool: 6.25%

Pool Tokens



=

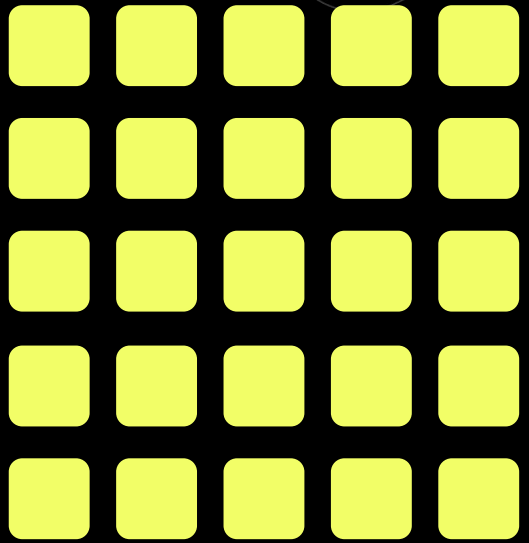
Asset A



100

+

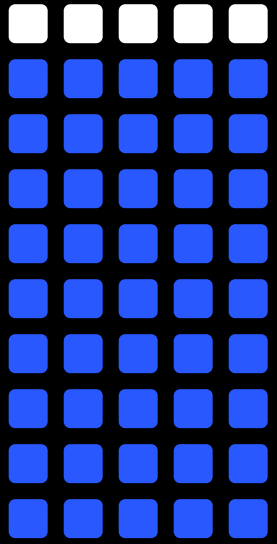
Asset B



25

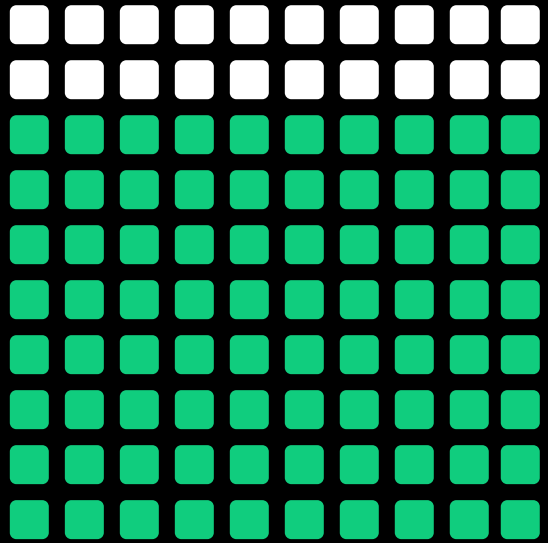
Held Pool Tokens: 50

Pool Tokens



=

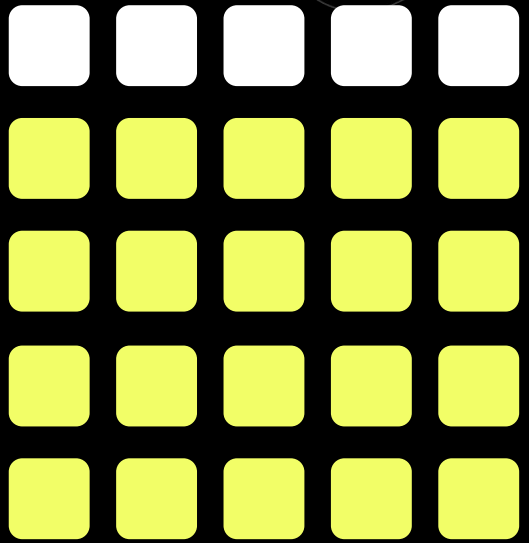
Asset A



80

+

Asset B



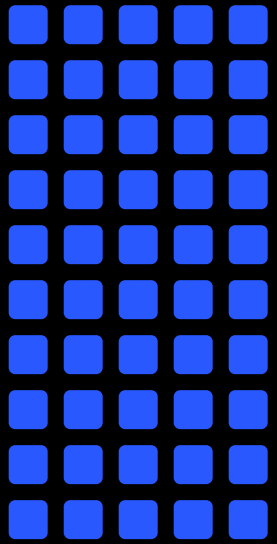
20

Held Pool Tokens: 45

Swap Fees

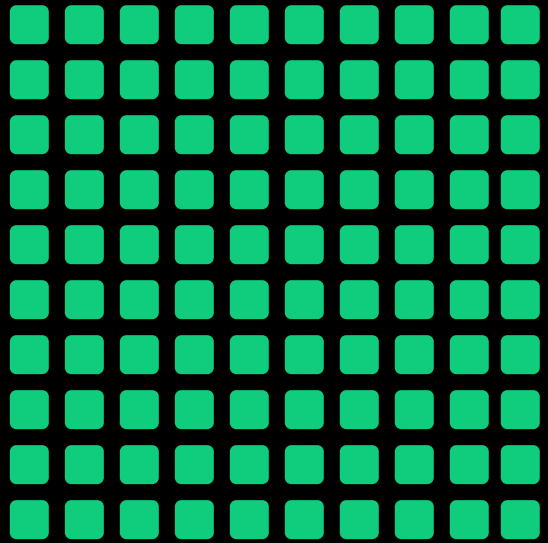
“By adding liquidity you'll earn 0.25% of all trades on this pair proportional to your share of the pool. Fees are added to the pool, accrue in real time and can be claimed by withdrawing your liquidity.”

Pool Tokens



=

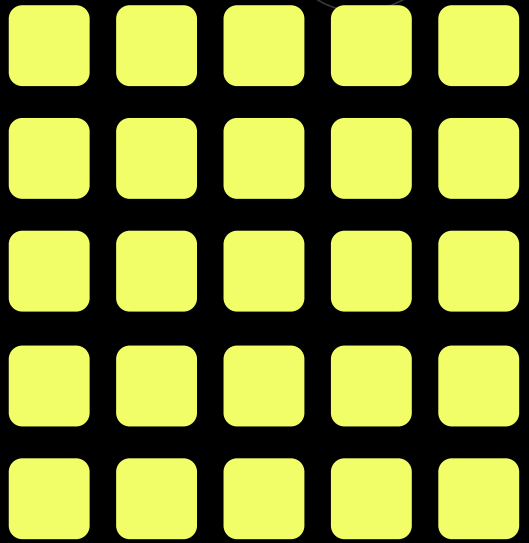
Asset A



100

+

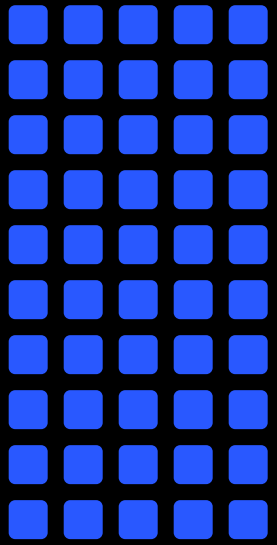
Asset B



25

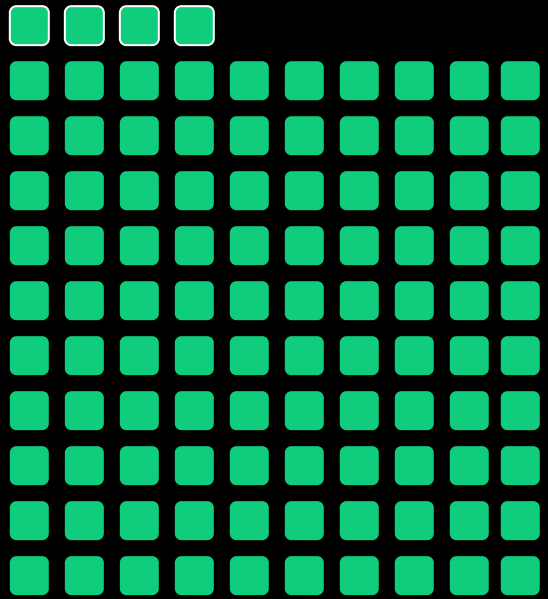
Held Pool Tokens: 50

Pool Tokens



=

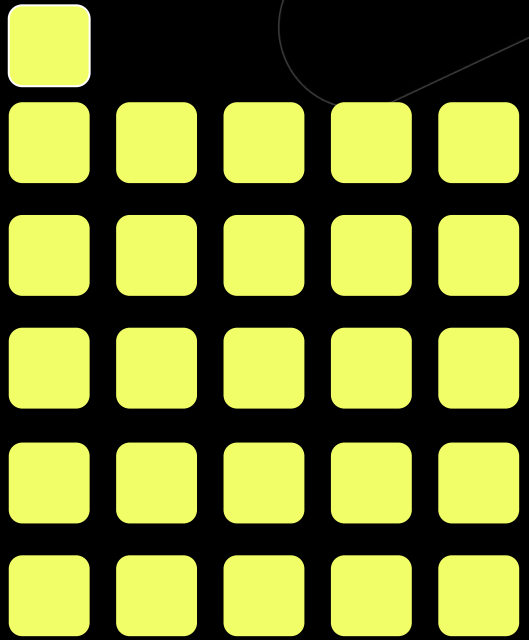
Asset A



104

+

Asset B



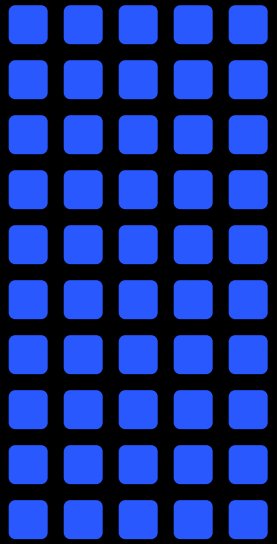
26

Held Pool Tokens: 50

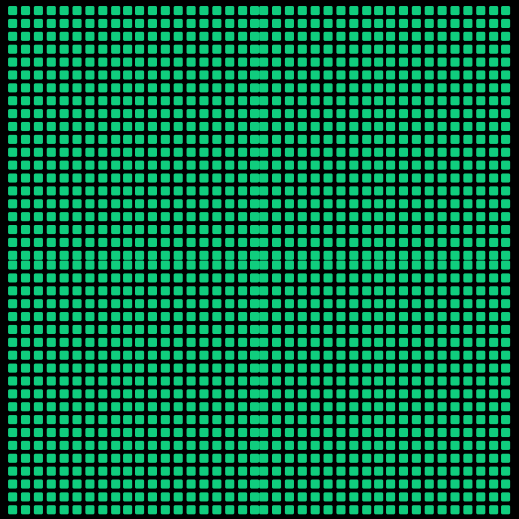
Pool Tokens

Asset A

Asset B



=



+

Held Pool Tokens: 50

10000

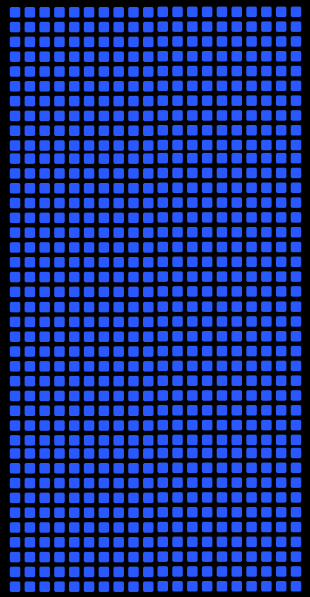
0.1

AMM on Algorand

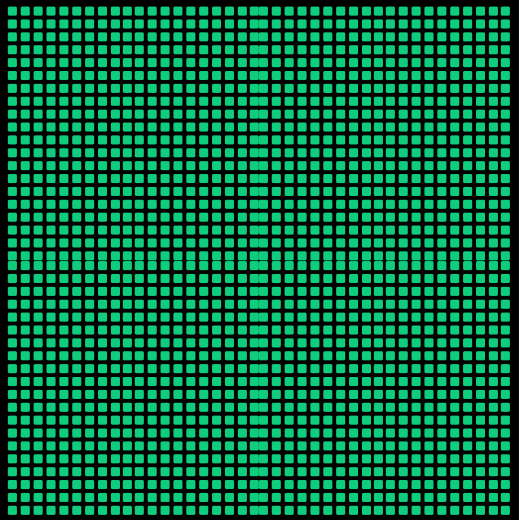
AMM on Algorand

(April 2021, TEAL3)

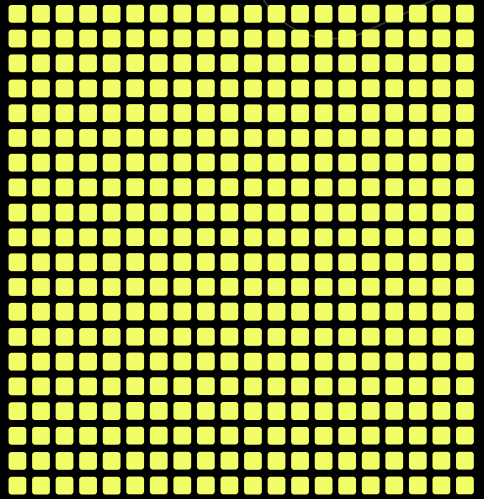
Pool Tokens



Asset A



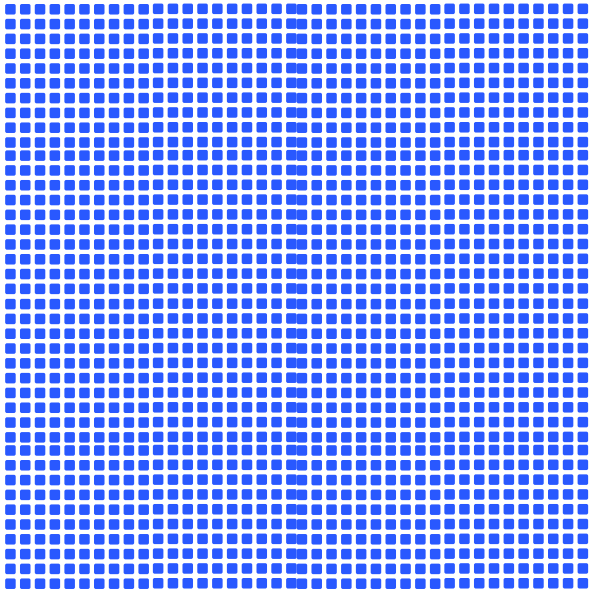
Asset B



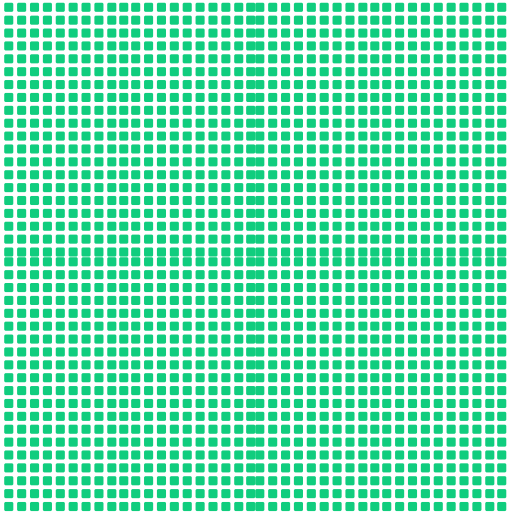
Contract Account

Pool Token (ASA)

UnMinted Supply



Asset 1 (ASA)



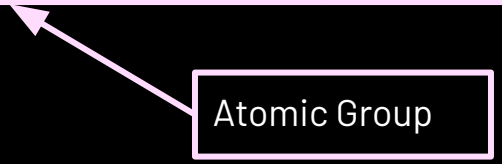
Asset 2 (ASA)

» Protocol - Swap

	Transaction Type	Signer (Sender)	Receiver
0	Asset Transfer (Asset 1)	Swapper	Pool
1	Asset Transfer (Asset 2)	Pool (LogicSig)	Swapper

» Protocol - Swap

	Transaction Type	Signer (Sender)	Receiver
0	Asset Transfer (Asset 1)	Swapper	Pool
1	Asset Transfer (Asset 2)	Pool (LogicSig)	Swapper



» Pool Contract Responsibilities

- Enforce $x * y = k$
- Only allow minting of Pool Token in exchange for correct amount of assets 1 & 2
- Only allow withdrawal of Assets 1 & 2 in exchange for correct amount of Pool Token

» Pool Contract (Logic Signature)

```
// enforce x * y = k

asset1_reserves = asset_holding_get(1, AssetBalance)

asset2_reserves = asset_holding_get(2, AssetBalance)

k = asset1_reserves * asset2_reserves

new_k = (asset1_reserves + gtxn[0].Amount) * (asset2_reserves - gtxn[1].Amount)

assert(new_k == k)
```

Pseudocode!

» Pool Contract (Logic Signature)

```
// enforce x * y = k

asset1_reserves = asset_holding_get(1, AssetBalance)

asset2_reserves = asset_holding_get(2, AssetBalance)

k = asset1_reserves * asset2_reserves

new_k = (asset1_reserves + gtxn[0].Amount) * (asset2_reserves - gtxn[1].Amount)

assert(new_k == k)
```

Stateless contract! Cannot check asset balances

Pseudocode!

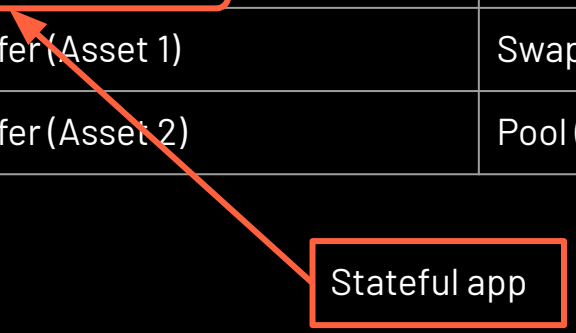
» Protocol - Swap

	Transaction Type	Signer (Sender)	Receiver
0	Asset Transfer (Asset 1)	Swapper	Pool
1	Asset Transfer (Asset 2)	Pool (LogicSig)	Swapper

Stateless! Cannot check asset balances

» Protocol - Swap

	Transaction Type	Signer (Sender)	Receiver
0	Application Call (“swap”)	Pool (LogicSig)	
1	Asset Transfer (Asset 1)	Swapper	Pool
2	Asset Transfer (Asset 2)	Pool (LogicSig)	Swapper



» Protocol - Swap

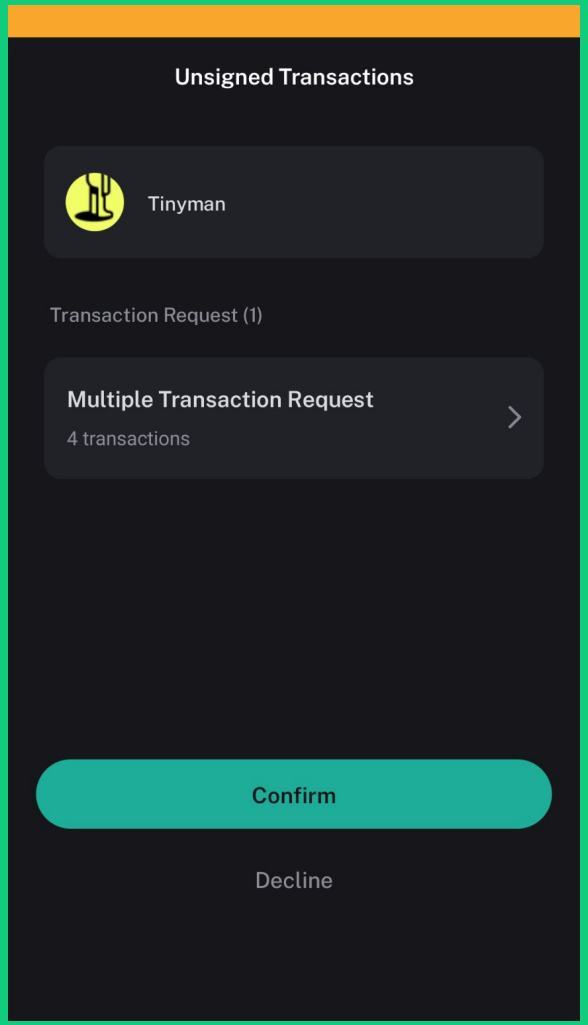
	Transaction Type	Signer (Sender)	Receiver	Transaction Fee
0	Application Call (“swap”)	Pool (LogicSig)		0.001
1	Asset Transfer (Asset 1)	Swapper	Pool	0.001
2	Asset Transfer (Asset 2)	Pool (LogicSig)	Swapper	0.001

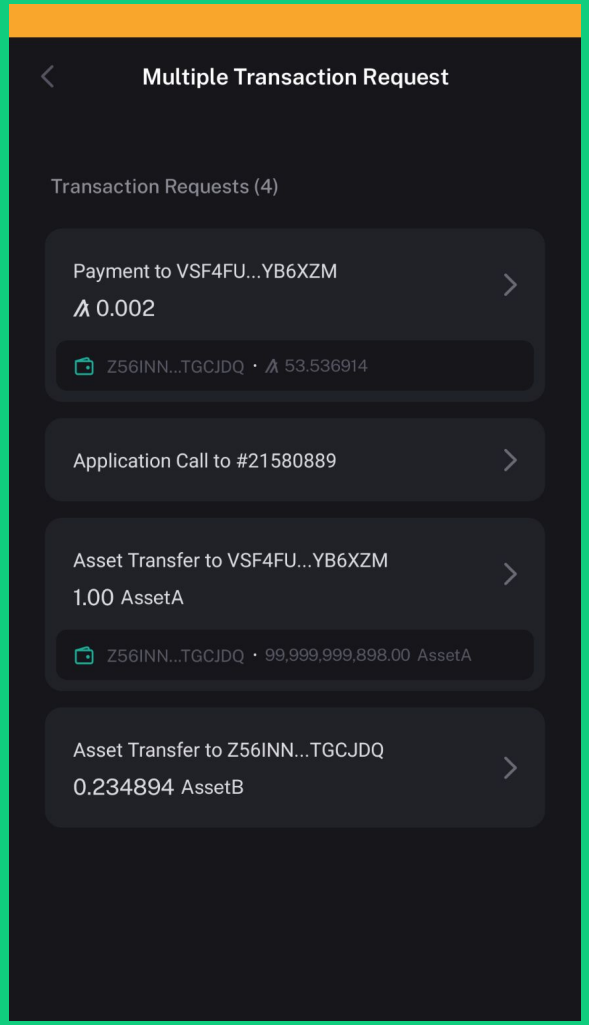
» Protocol - Swap

	Transaction Type	Signer (Sender)	Receiver	Transaction Fee
0	Application Call (“swap”)	Pool (LogicSig)		0.001
1	Asset Transfer (Asset 1)	Swapper	Pool	0.001
2	Asset Transfer (Asset 2)	Pool (LogicSig)	Swapper	0.001

» Protocol - Swap

	Transaction Type	Signer (Sender)	Receiver
0	Payment	Swapper	Pool
1	Application Call (“swap”)	Pool (LogicSig)	
2	Asset Transfer (Asset 1)	Swapper	Pool
3	Asset Transfer (Asset 2)	Pool (LogicSig)	Swapper


Unsigned Transactions
Tinyman
Transaction Request (1)
Multiple Transaction Request
4 transactions
Confirm
Decline


Multiple Transaction Request
Transaction Requests (4)
Payment to VSF4FU...YB6XZM
0.002
Z56INN...TGCJDQ · 53.536914
Application Call to #21580889
Asset Transfer to VSF4FU...YB6XZM
1.00 AssetA
Z56INN...TGCJDQ · 99,999,999,898.00 AssetA
Asset Transfer to Z56INN...TGCJDQ
0.234894 AssetB

» Pool Logic Signature

```
// ensure gtxn 1 is ApplicationCall to Validator App
assert(gtxn[1].Sender == txn.Sender)
assert(gtxn[1].TypeEnum == ApplicationCall)
assert(gtxn[1].ApplicationID == VALIDATOR_APP_ID)

// ensure gtxn 0 amount covers all fees
// ensure Pool is not paying the fee
assert(gtxn 0 Sender != txn Sender)
// ensure Pool is receiving the fee
assert(gtxn[0].Receiver == txn Sender)
// ensure fee amount is sufficient
assert(gtxn[0].Amount >= fee_total)
```

Pseudocode!

» Validator App - Swapping

```
// so simple :)

k = asset1_reserves * asset2_reserves

new_k = (asset1_reserves + sell_amount) * (asset2_reserves - buy_amount)

assert(new_k == k)
```

Pseudocode!

This works perfectly

This works perfectly

... in demos on Testnet!

This would cause huge frustration on Mainnet

with concurrent users!

This would cause huge frustration on Mainnet

with concurrent users!

Why?

Slippage

» Swapping - Slippage

1. Client looks up pool reserves from account and generates quote
2. User accepts quote and signs transactions
3. Client submits transactions
4. Transactions get executed by Algorand Node
5. User receives confirmation

» Swapping - Slippage

1. Client looks up pool reserves from account and generates quote
2. User accepts quote and signs transactions
3. Client submits transactions
4. Transactions get executed by Algorand Node
5. User receives confirmation

Between steps 1 & 4 multiple rounds may pass and the pool reserves may change due to other swaps/mints/burns, thus invalidating the quote.

Even within the same round multiple users may try to swap with the same pool.

» Swapping - Slippage

1. Client looks up pool reserves from account and generates quote
2. User accepts quote and signs transactions
3. Client submits transactions
4. Transactions get executed by Algorand Node
5. User receives confirmation

Between steps 1 & 4 multiple rounds may pass and the pool reserves may change due to other swaps/mints/burns, thus invalidating the quote.

Even within the same round multiple users may try to swap with the same pool.

The exchange rate has now **slipped** but the system has no tolerance for slippage because the user has signed transactions expecting a precise amount in return for their input.

A contract can only approve or deny transactions, not create or modify transactions!

Slippage Tolerance

» Swapping - Slippage Tolerance

Swapper: “I want to buy *100A* for *25B* but I’m will to accept *99A* at minimum. But I’d really prefer *100A*.”

i.e 1% slippage tolerance

1. Swapper signs transactions for the minimum amount they are willing to receive.
2. The contract calculates the the expected output amount at the time of execution.
3. The contract stores the difference as *excess* in local state (i.e. change/IOU).
4. The user *redeems* their *excess* from the pool after the swap completes.

» Swapping with Tolerance

```
k = asset1_reserves * asset2_reserves

asset_in_amount = gtxn[2].Amount

asset_out_amount = gtxn[3].Amount

calculated_amount_out = asset2_reserves - (k / (asset1_reserves + asset_in_amount))

// Calculate excess amount

excess = calculated_amount_out - asset_out_amount

excess_asset_2_amount += excess_asset_out

// Store excess amount in swapper's local state

app_local_put(1, "excess_2", excess_asset_2_amount)

put outstanding_asset_out_amount += excess_asset_out

// Store outstanding amount in pool's local state

app_local_put(0, "outstanding_2", excess_asset_2_amount)
```

Pseudocode!

» Protocol - Redeem

	Transaction Type	**Sender (Signer)**	**Receiver**
0	Payment	Swapper	Pool
1	Application Call (“redeem”)	Pool (LogicSig)	
2	Asset Transfer	Pool (LogicSig)	Swapper

TO

AssetB 0.23271

The amounts shown reflect the final swap values including excess amounts which must be redeemed separately.

Transaction ID

CBP6ZIHH...ZK5KTLPQ

View on AlgoExplorer ↗

You have some excess AssetB tokens that you can claim now or later from the Accounts page.

REDEEM 0.009585 AssetB

GO BACK TO SWAP

TO

AssetB 0.23271

The amounts shown reflect the final swap values including excess amounts which must be redeemed separately.

Transaction ID

CBP6ZIHH...ZK5KTLPQ

View on AlgoExplorer ↗

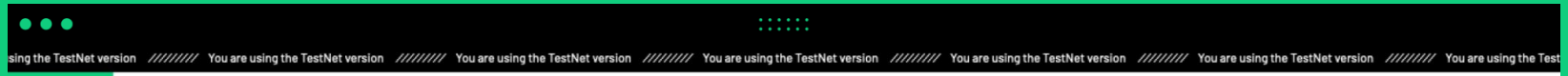
You have some excess AssetB tokens that you can claim now or later from the Accounts page.

0.009585 AssetB

GO BACK TO SWAP

Sending the transaction

Redeeming AssetB



TO

AssetB 0.23271

The amounts shown reflect the final swap values including excess amounts which must be redeemed separately.

Transaction ID

CBP6ZIHH...ZK5KTLPQ

View on AlgoExplorer ↗

You have some excess AssetB tokens that you can claim now or later from the Accounts page.

REDEEMED 0.009585 AssetB

GO BACK TO SWAP

Success

Redeeming AssetB

QSYBB6UE...W4E3M4HA ↗

Poolers

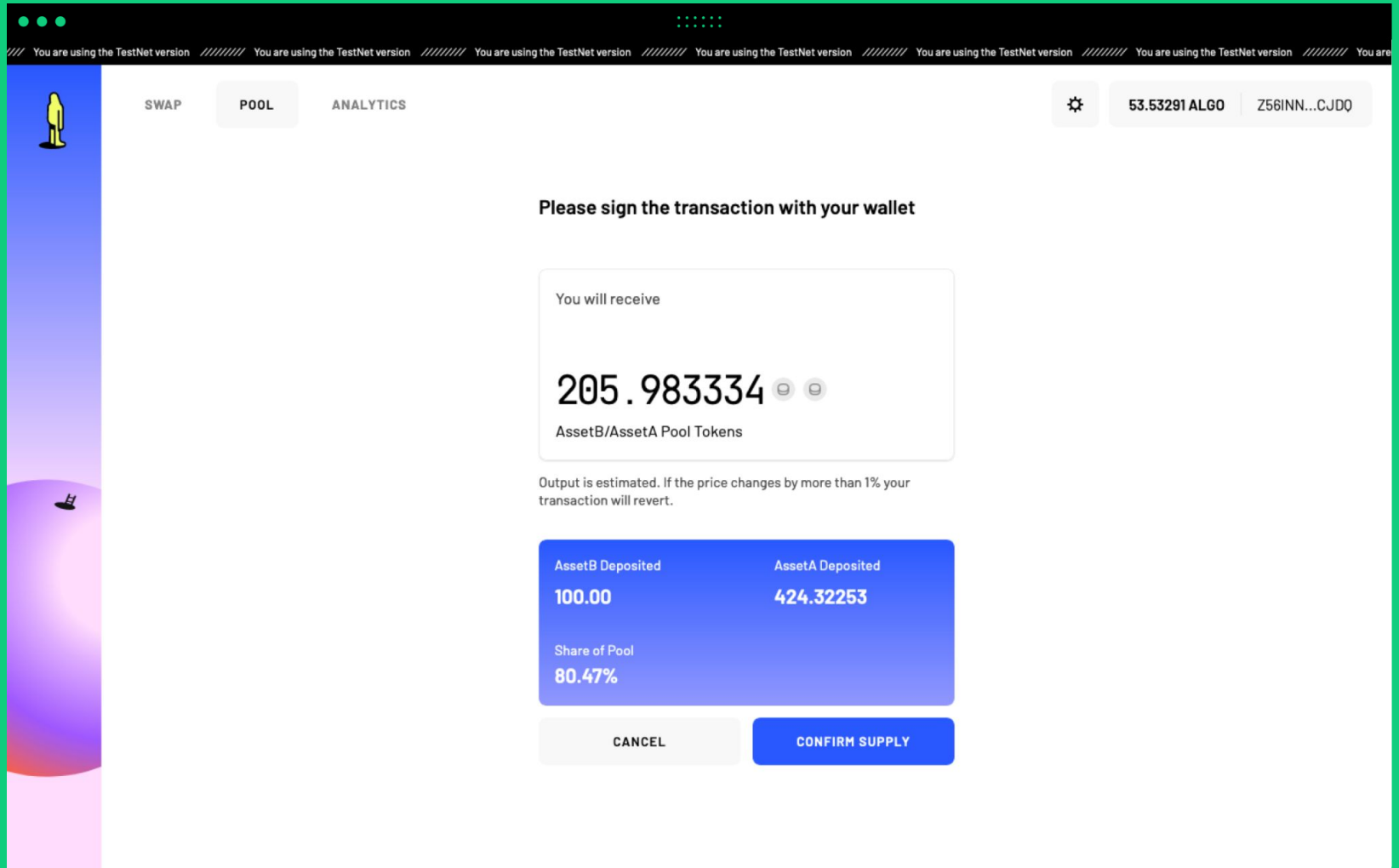
» Protocol - Mint (Add Liquidity)

	Transaction Type	**Signer (Sender)**	**Receiver**
0	Payment	Pooler	Pool
1	Application Call (“mint”)	Pool (LogicSig)	
2	Asset Transfer (Asset 1)	Pooler	Pool
3	Asset Transfer (Asset 2)	Pooler	Pool
4	Asset Transfer (Pool Token)	Pool (LogicSig)	Pooler

» Validator App - Mint (Add Liquidity)

```
if gtxna[1].ApplicationArgs[0] == “mint”:

    // Calculate expected pool token out amount
    calculated_pool_token_out = Min(
     asset1_amount * issued_pool_tokens / asset1_reserves,
     asset2_amount * issued_pool_tokens / asset2_reserves
    )

    excess_pool_token = calculated_pool_token_out - pool_token_amount

    // record outstanding liquidity in Pool account state
    outstanding_pool_token_amount += excess_pool_token
    app_local_put(0, "outstanding_pool_token_amount", outstanding_pool_token_amount)

    // record excess liquidity in Pooler account state
    excess_pool_token_amount += excess_pool_token
    app_local_put(1, "excess_pool_token_amount", excess_asset_2_amount)
```

Pseudocode!


You are using the TestNet version
SWAP
POOL
ANALYTICS
53.53291 ALGO
Z56INN...CJD0
Please sign the transaction with your wallet
You will receive
205.983334
AssetB/AssetA Pool Tokens
Output is estimated. If the price changes by more than 1% your transaction will revert.
AssetB Deposited
100.00
AssetA Deposited
424.32253
Share of Pool
80.47%
CANCEL
CONFIRM SUPPLY

» Protocol - Bootstrap

	Transaction Type	**Signer (Sender)**	**Receiver**
0	Payment	User	Pool
1	Application Call (“bootstrap”)	Pool (LogicSig)	
2	Asset Creation (Pool Token)	Pool (LogicSig)	
3	Asset Optin (Asset 1)	Pool (LogicSig)	
4	Asset Optin (Asset 2)	Pool (LogicSig)	

Liquidity provider rewards

Liquidity providers earn a 0.25% fee on all trades proportional to their share of the pool. Fees are added to the pool, accrue in real time and can be claimed by withdrawing your liquidity.

Learn more about providing liquidity

Your liquidity

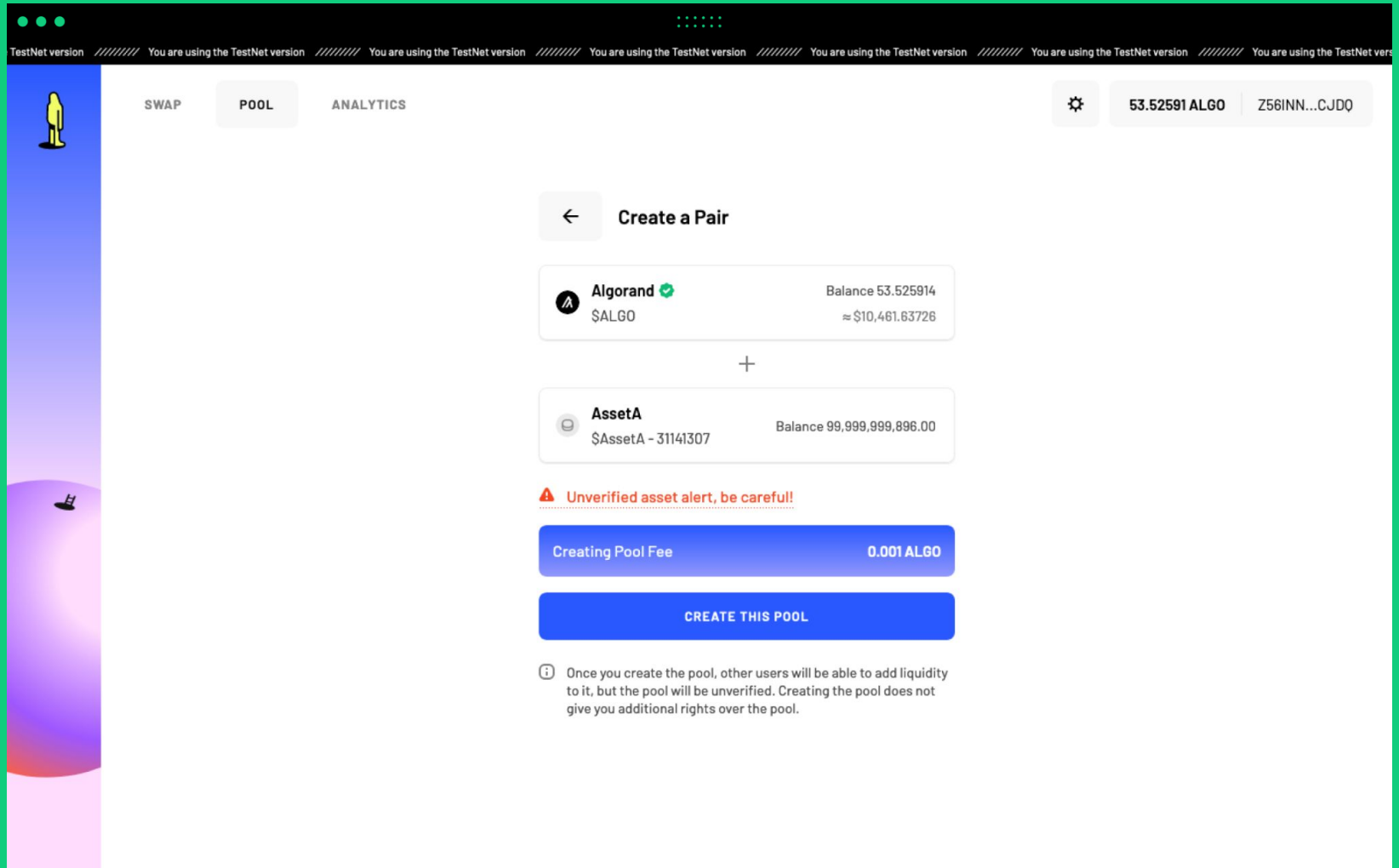
CREATE A PAIR ADD LIQUIDITY

AssetB / AssetA 1 AssetB = 4.325893 AssetA

100% Share of Pool	24.040561 AssetB	103.996898 AssetA	MANAGE

TINYUSDC / USDH 1 TINYUSDC = 0.000002 USDH

33.47%	6.690.915395 TINYUSDC	0.016 USDH


TestNet version
You are using the TestNet version
SWAP
POOL
ANALYTICS
53.52591 ALGO
Z56INN...CJDQ
Create a Pair
Algorand
$ALGO
Balance 53.525914
≈ $10,461.63726
AssetA
$AssetA - 31141307
Balance 99,999,999,896.00
Unverified asset alert, be careful!
Creating Pool Fee
0.001 ALGO
CREATE THIS POOL
Once you create the pool, other users will be able to add liquidity to it, but the pool will be unverified. Creating the pool does not give you additional rights over the pool.

Add Liquidity

Algorand $ALGO

Balance 52.665914
≈ $10,293.52723

100
≈ $19,544.95128

MAX

AssetA $AssetA - 31141307

Balance 99,999,999,896.00

1,000

MAX

Unverified asset alert, be careful!

Prices and pool share

0.1	10	100%
ALGO per AssetA	AssetA per ALGO	Share of pool

The first 0.001 pool tokens are locked in the pool as a

SWAP | POOL | ANALYTICS

52.66591 ALGO | Z56INN...CJDQ

Unverified asset alert, be careful!

Prices and pool share

0.1	10	100%
ALGO per AssetA	AssetA per ALGO	Share of pool

The first 0.001 pool tokens are locked in the pool as a protection mechanism. These amounts will be permanently locked.

0.003162 ALGO | 0.000316 AssetA

To add liquidity, you need to first opt-in your account to the liquidity token of this pool. You only need to do this once per account to validate that it has access to the liquidity token.

OPT IN | ADD LIQUIDITY

By adding liquidity you'll earn 0.25% of all trades on this pair proportional to your share of the pool. Fees are added to the pool, accrue in real time and can be claimed by withdrawing your liquidity.

» Protocol - Burn (Remove Liquidity)

	Transaction Type	**Sender (Signer)**	**Receiver**
0	Payment	Pooler	Pool
1	Application Call (“burn”)	Pool (LogicSig)	
2	Asset Transfer (Asset 1)	Pool (LogicSig)	Pooler
3	Asset Transfer (Asset 2)	Pool (LogicSig)	Pooler
4	Asset Transfer (Pool Token)	Pooler	Pool

» Validator App - Burn (Remove Liquidity)

```
if gtxna[1].ApplicationArgs[0] == “burn”:

    burn_amount = gtxn[4].AssetAmount
    asset1_amount = gtxn[2].AssetAmount
    asset2_amount = gtxn[3].AssetAmount

    calculated_asset1_out = asset1_reserves * (burn_amount / issued_pool_tokens)
    calculated_asset2_out = asset2_reserves * (burn_amount / issued_pool_tokens)

    excess_asset_1 = calculated_asset1_out - asset1_out
    excess_asset_2 = calculated_asset2_out - asset2_out
    // if the excess amount is negative an error will occur

    // record outstanding assets in Pool account state
    outstanding_asset1_amount +=  excess_asset_1
    app_local_put(0, "outstanding_1", outstanding_asset1_amount)
    outstanding_asset2_amount +=  excess_asset_2
    app_local_put(0, "outstanding_2", outstanding_asset1_amount)

    // record excess assets in Pooler account state
    excess_asset1_amount += excess_asset_1
    app_local_put(1, "excess_1", excess_asset_2_amount)
    excess_asset2_amount += excess_asset_2
    app_local_put(1, "excess_2", excess_asset_2_amount)
```

Pseudocode!

» Pool Logic Signature

```
if gtxna[1].ApplicationArgs[0] == “bootstrap”:
      bootstrap:
      // ensure correct asset ids are included as args to the bootstrap call
      assert(gtxna[1].ApplicationArgs[1] == TMPL_ASSET_ID_1)
      assert(gtxna[1].ApplicationArgs[2] == TMPL_ASSET_ID_2)
      ...
```

`TMPL_*` variables are replaced in bytecode dynamically by clients when generating LogicSig for specific asset pair. This makes each LogicSig contract deterministically unique.

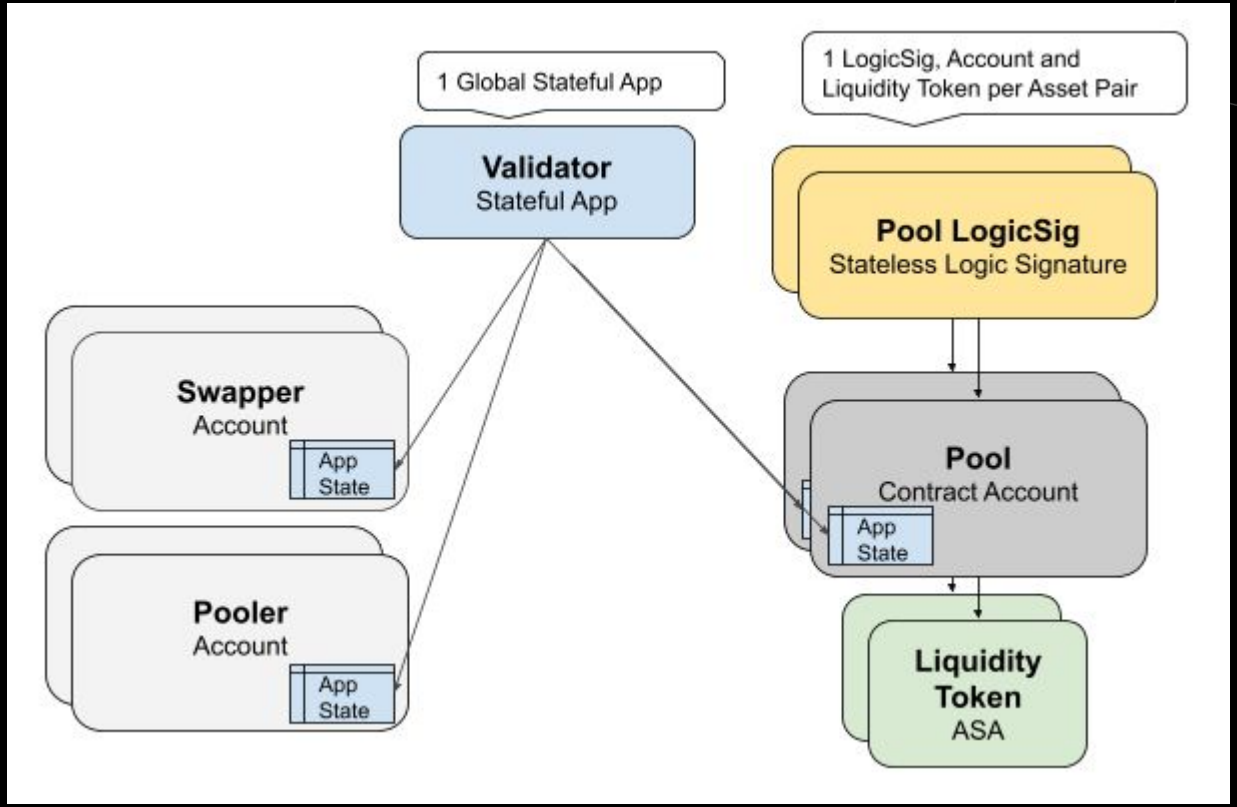
Unique contract ⟶ unique contract account address

Pseudocode!

» Protocol - Common Structure

	Transaction Type	**Sender (Signer)**	**Receiver**
0	Payment	*User*	Pool
1	Application Call (“*operation*”)	Pool (LogicSig)	
2			
3			
4			

» Protocol - Architecture



“Tinyman is a **permissionless trustless** decentralized trading protocol on Algorand”

» Immutable Code

No updates

No deletion

```
// Deny Update, Delete, CloseOut

txn OnCompletion
int UpdateApplication
==

txn OnCompletion
int DeleteApplication
==

||

txn OnCompletion
int CloseOut
==

||

bnz fail
```

https://github.com/tinymanorg/tinyman-contracts-v1/blob/main/contracts/validator_approval.teal#L51:L63

TEAL

» Permissionless Assets

Pool Token:

No clawback

No freeze

```
// ensure no asset freeze address is set
gtxn 2 ConfigAssetFreeze
global ZeroAddress
==
assert

// ensure no asset clawback address is set
gtxn 2 ConfigAssetClawback
global ZeroAddress
==
assert
```

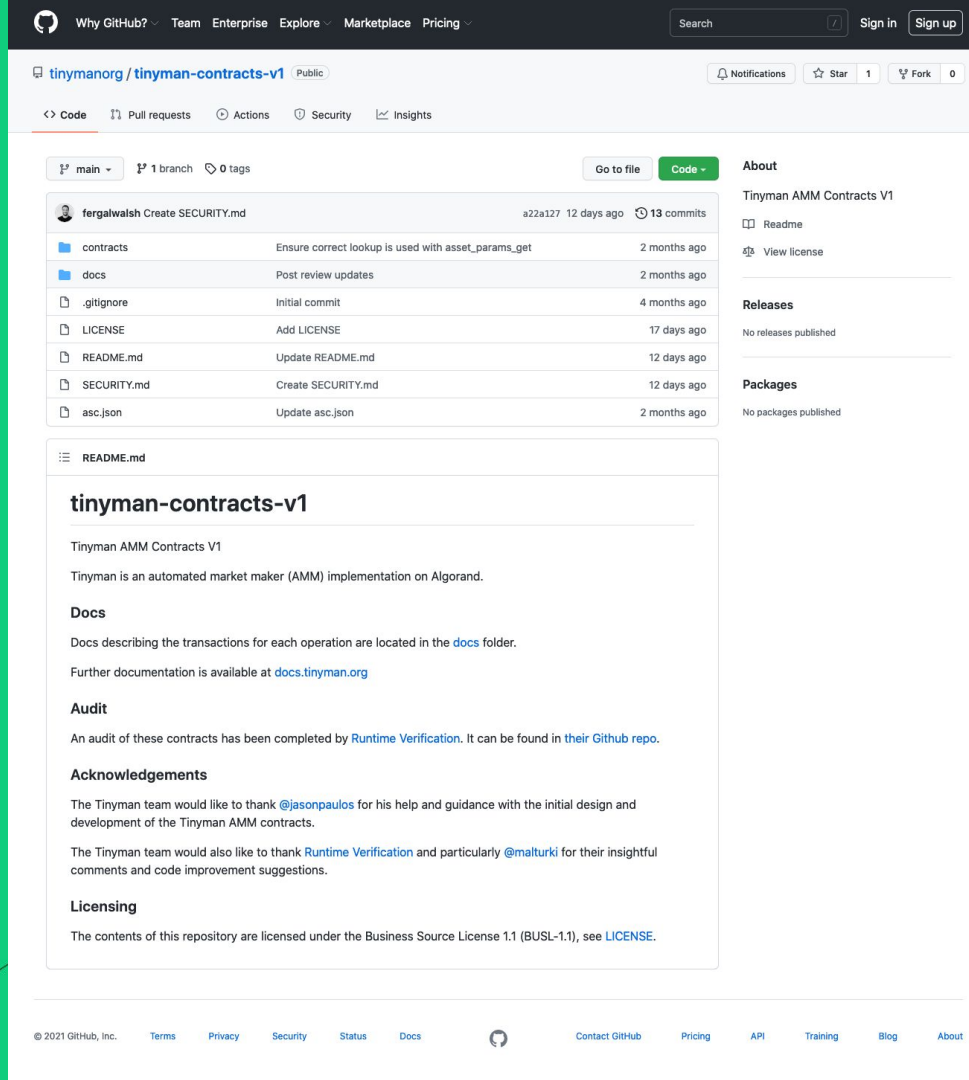
https://github.com/tinymanorg/tinyman-contracts-v1/blob/main/contracts/pool_logicsig.teal.tmpl#L215:L225

TEAL

Fully commented code is public on Github:

https://github.com/tinymanorg/tinyman-contracts-v1

Open Source Contracts



https://runtimeverification.com/blog/runtime-verification-audits-tinyman

Audit

Runtime Verification Audits Tinyman

Posted on September 22, 2021 by Silvia Barredo
Posted in Audits

Runtime Verification is thrilled to announce Tinyman's audit completion. Tinyman is a new project under development that focuses on bringing a decentralized trading protocol to the Algorand ecosystem and its community.

Tinyman's Audit Scope

Before Tinyman's launch, their team has decided to audit the project's code with Runtime Verification to identify any issues that could cause the system to malfunction or be exploited.

Tinyman's protocol is built with Algorand's Layer-1 smart contract language TEAL and aims to serve as a decentralized trading platform, similar to the Uniswap protocol, for the Algorand community. The protocol is built using two core smart contracts:

- The Pool Logic Signature Template (a stateless TEAL smart contract template), from which contract accounts representing liquidity pools are created.
- The Validator Application (a stateful TEAL smart contract), which implements the protocol's logic and maintains the state of the system on the blockchain. The contract was created by the core team and is immutable.

Runtime Verification performed an audit on the two contracts just mentioned above and the system's high-level documentation. The focus was reviewing the high-level business logic (protocol design) of Tinyman's system based on the provided documentation and reviewing the low-level implementation of the system in TEAL. In addition, the audit highlighted some informative findings that could improve the performance and efficiency of the implementation and optimize its code size.

Methodology

Runtime Verification team lead Musab Alturki conducted Tinyman's audit and published a detailed report on August 4th, 2021.

The first step consisted of rigorously reasoning about the business logic of the contract and validating security-critical properties to ensure the absence of loopholes in the logic. We also reviewed past public audit reports of Uniswap v1 and v2 and checked if the list of known issues could be applied to Tinyman, and in case they did, to check the code to make sure there is no space for any vulnerability. This step was crucial to conduct as Tinyman's system is based on Uniswap's AMM (Automated Market Maker) design.

The second step consisted of reviewing the contract source code to detect any unexpected (and possibly exploitable) behaviors. We applied an approach that consisted of constructing different high-level representations of the TEAL codebase to systematically check consistency between the logic and the low-level TEAL implementation.

The third step consisted of reviewing the TEAL guidelines published by Algorand to check for known issues and reviewing a list of known Ethereum security vulnerabilities and attack vectors to check whether they apply to TEAL smart contracts and, if they do, to check whether the code is vulnerable to them.

Results

The audit identified and highlighted some critical issues along with a number of informative findings (see the report for details). The Tinyman team properly addressed all the issues and concerns raised during the audit, and incorporated all the necessary changes in the smart contracts.

Finally, in a concluding phase of the audit, we further reviewed the security impact of the changes made based on the issues raised in the first phase and investigated their effects on other parts of the contracts to ensure that no new issues or vulnerabilities were introduced in the process.

We enjoyed working with the Tinyman team and wish them the best of luck with their project.

About Tinyman

Tinyman is a re-imagined decentralized trading protocol which utilizes the fast and secure framework of the Algorand blockchain, creating an open and safe marketplace for traders, liquidity providers, and developers.

About Runtime Verification

Runtime Verification is a technology startup based in Champaign-Urbana, Illinois. The company uses formal methods to perform security audits on virtual machines and smart contracts on public blockchains. It also provides software testing, verification services and products to improve the safety, reliability, and correctness of software systems in the blockchain field.

← From 0 to K Tutorial

» Contracts - AVM 1.0+ / TEAL5+

Future design possibilities:

- Stateful Contract Account
- Inline slippage tolerance support with inner transactions
- Code reuse with functions
- Fee pooling

Web App


SWAP
POOL
ANALYTICS
FROM
Balance 4,000.00
≈ $4,600.23
Algorand
ALGO
0.00
TO
Select a token, or paste the ID
CONNECT TO A WALLET

Web App

JS AlgoSDK

Tinyman JS SDK

React

MyAlgoConnect

AlgoSigner

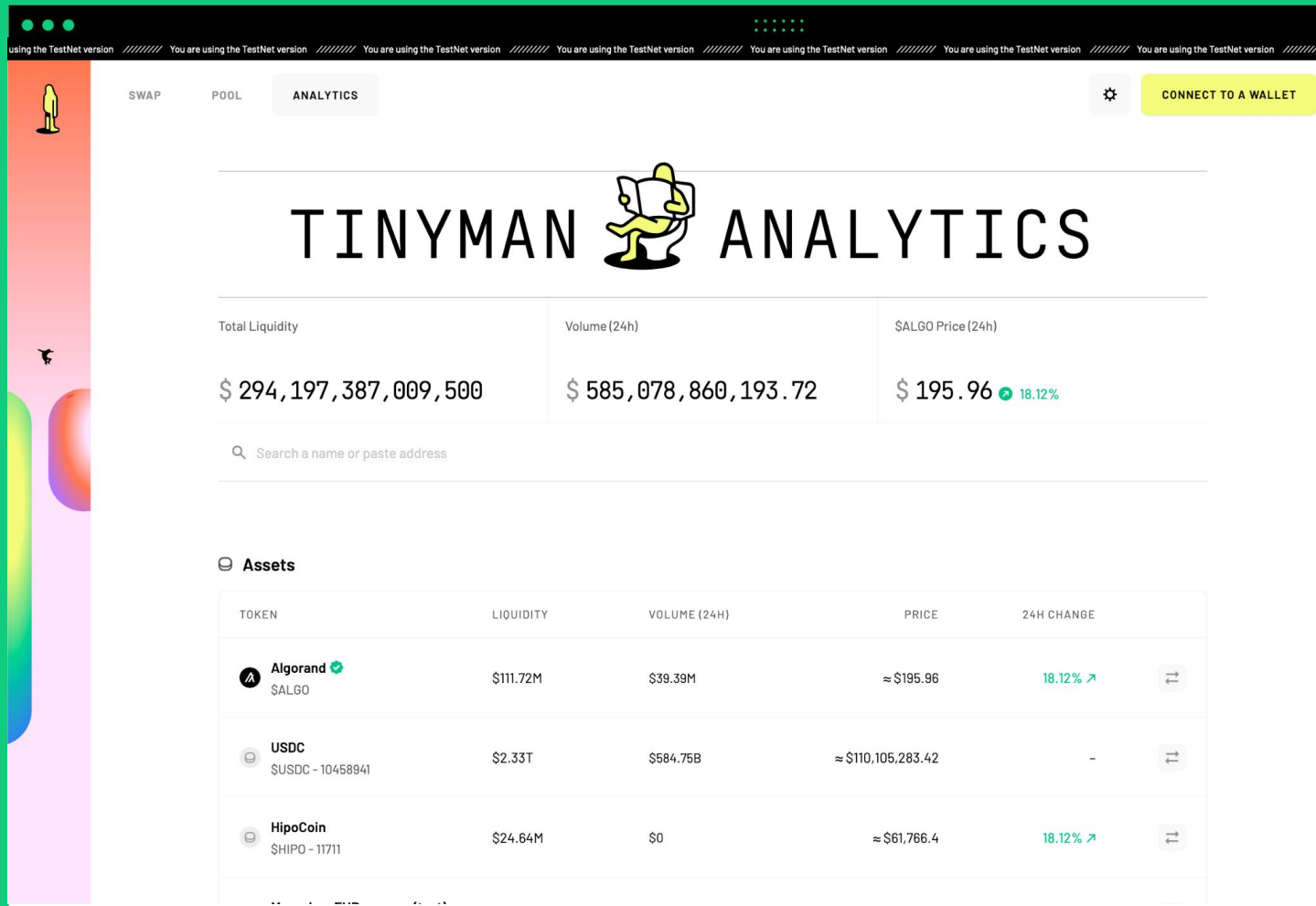
WalletConnect

Analytics API

Tinyman Analytics

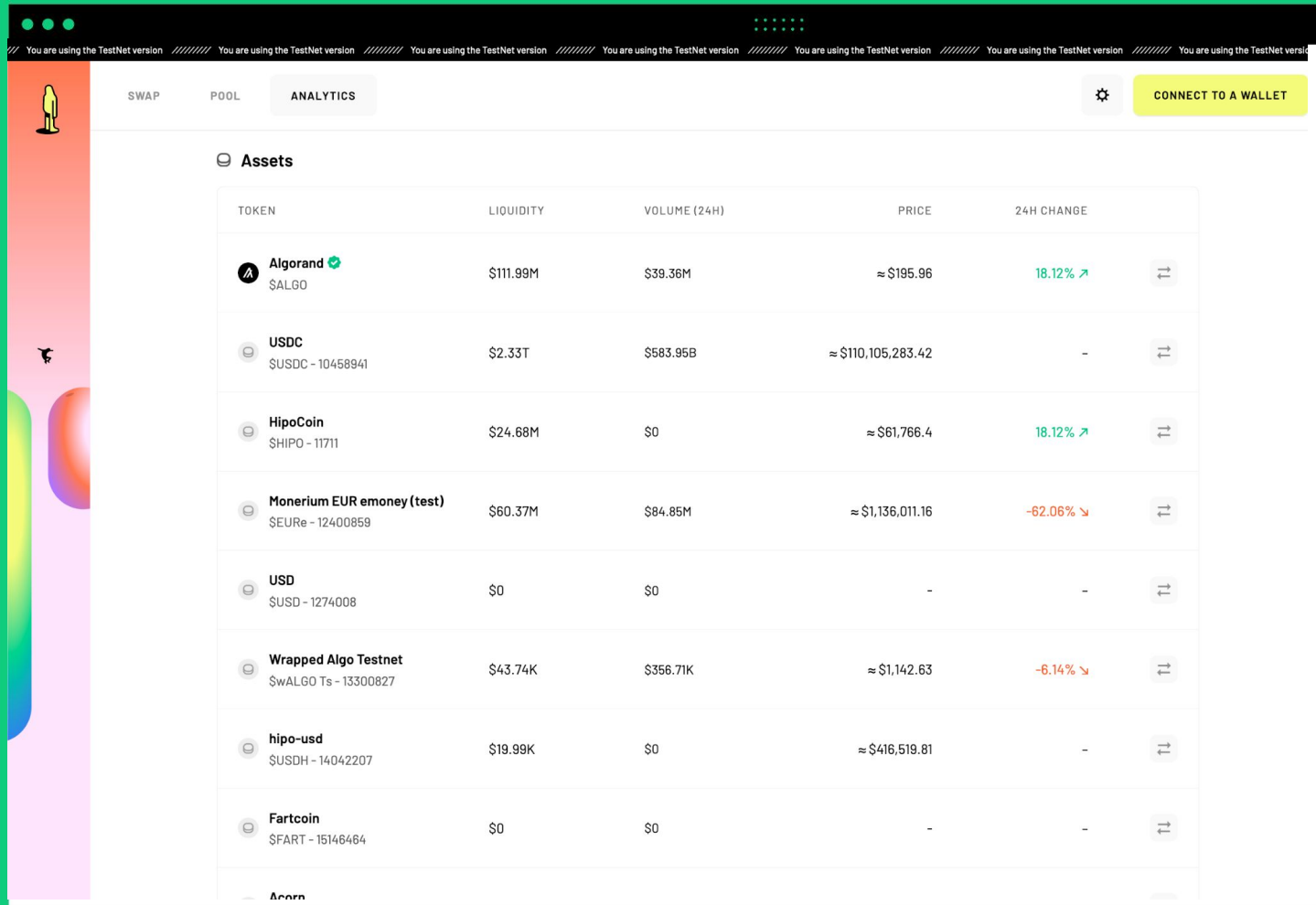
Tinyman Analytics

i. General Statistics



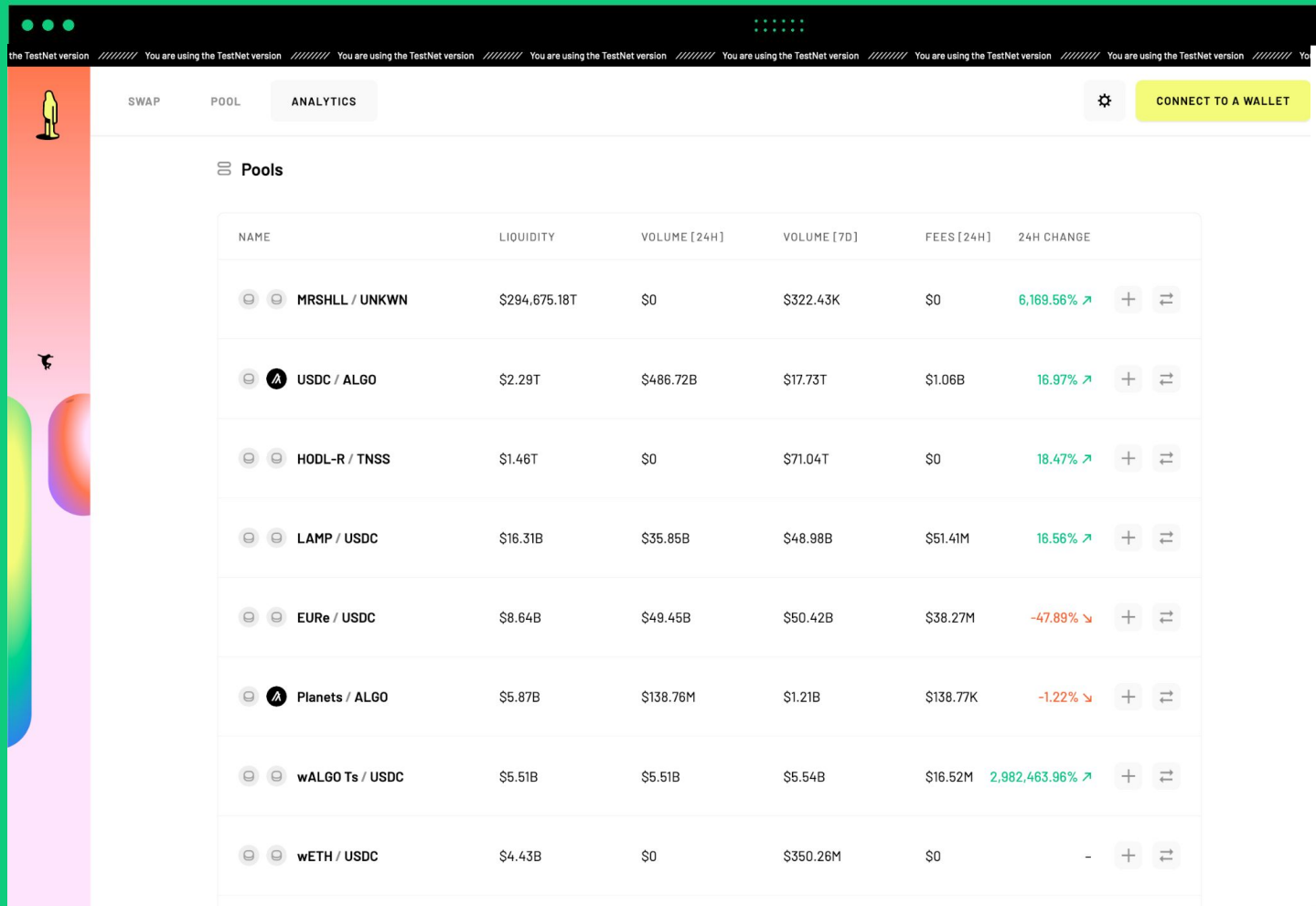
Tinyman Analytics

ii. Asset Statistics



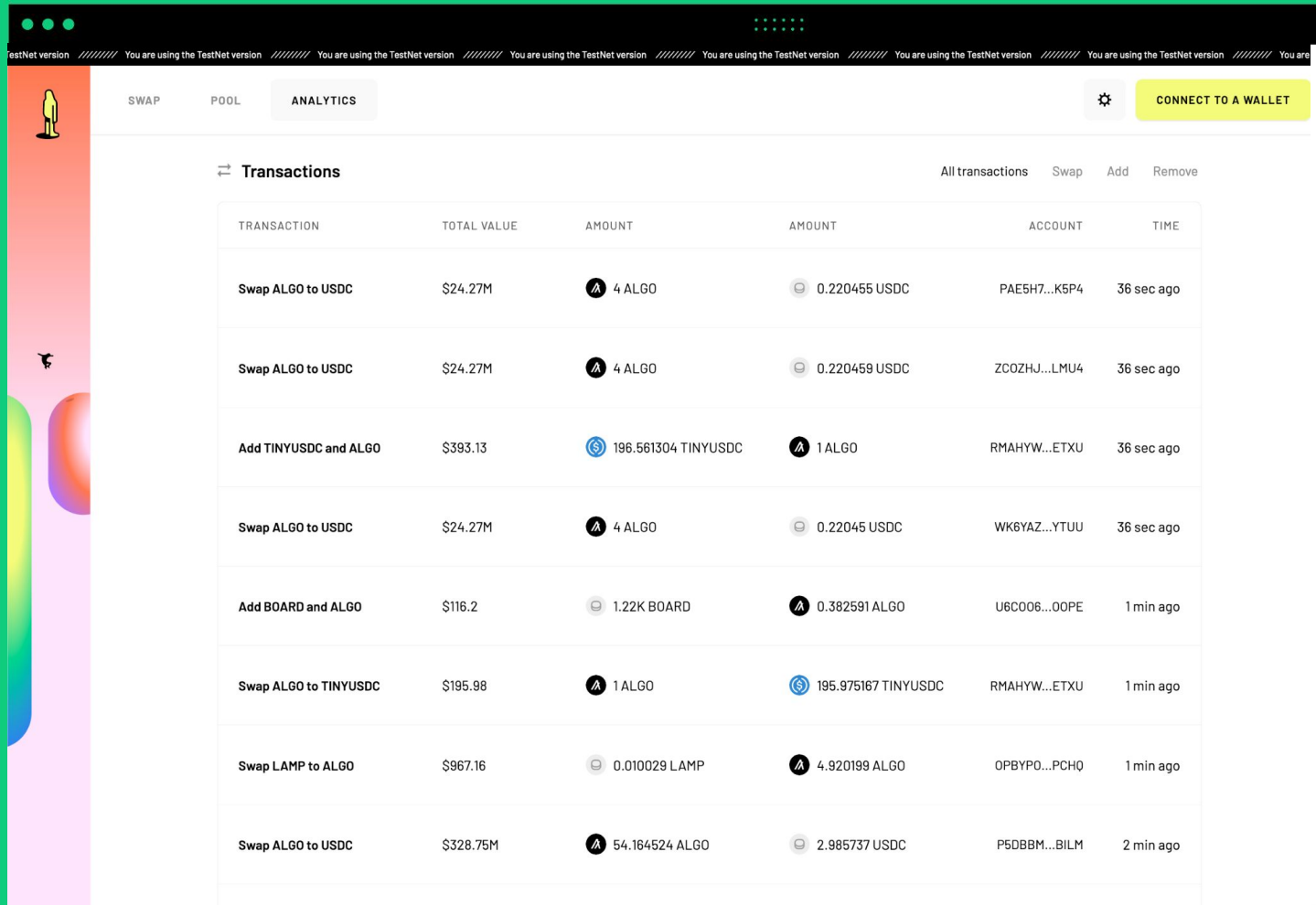
Tinyman Analytics

iii. Pool Statistics



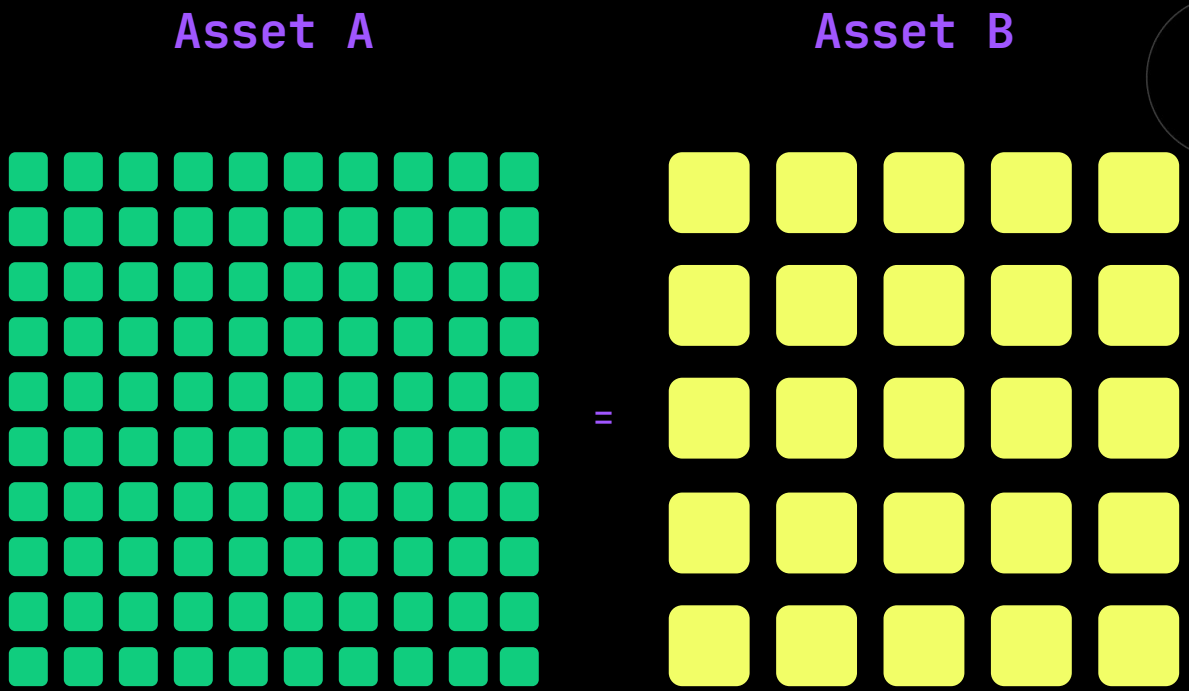
Tinyman Analytics

iv. Transaction Statistics

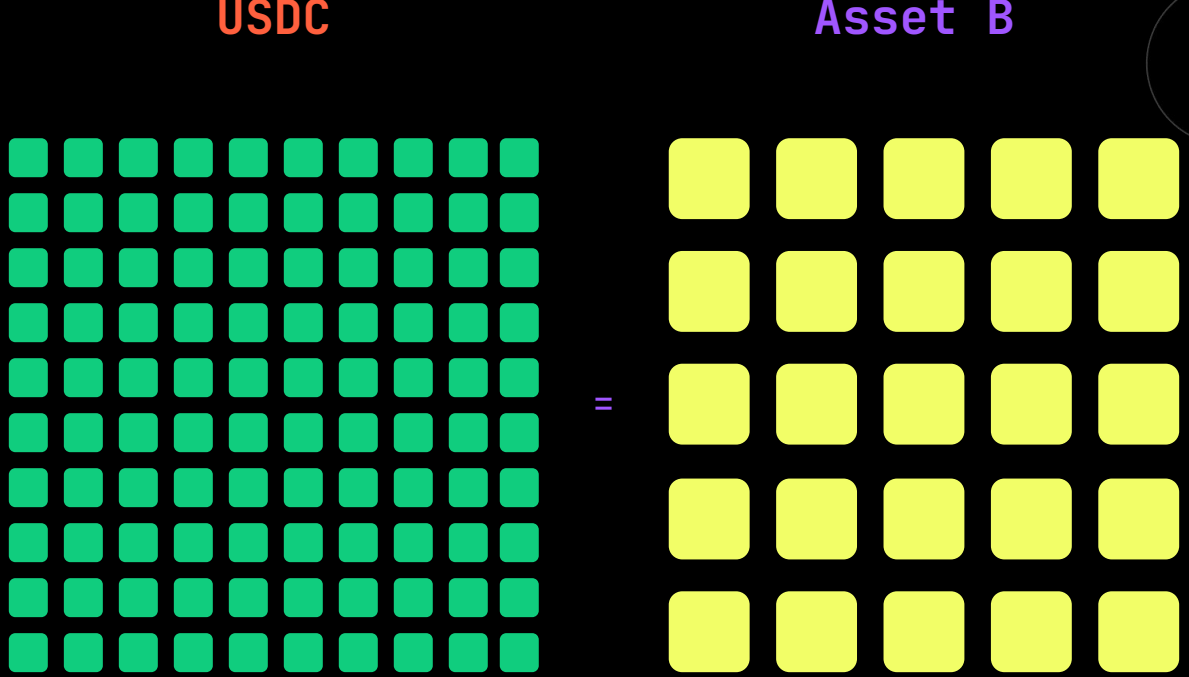



USD?

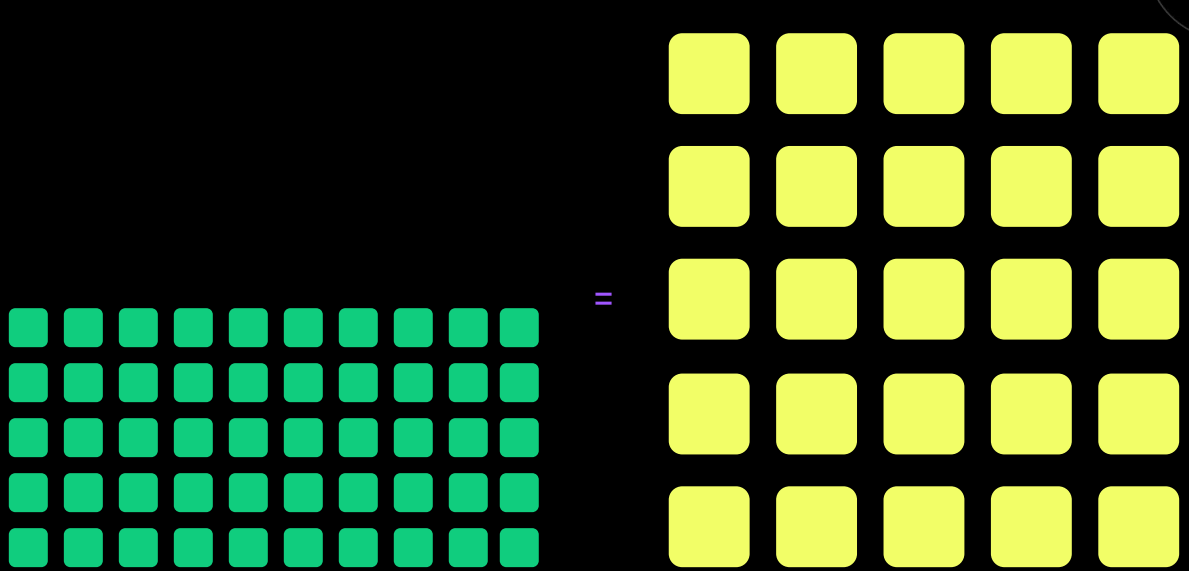
» USD?



» USD?



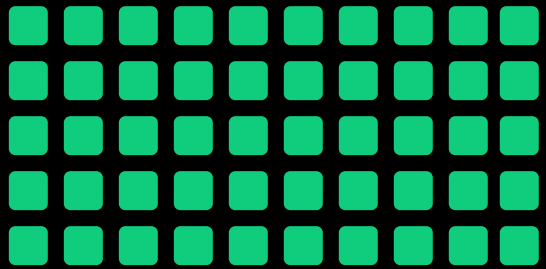
» USD?



» USD?

1Algo
= 2 USDC

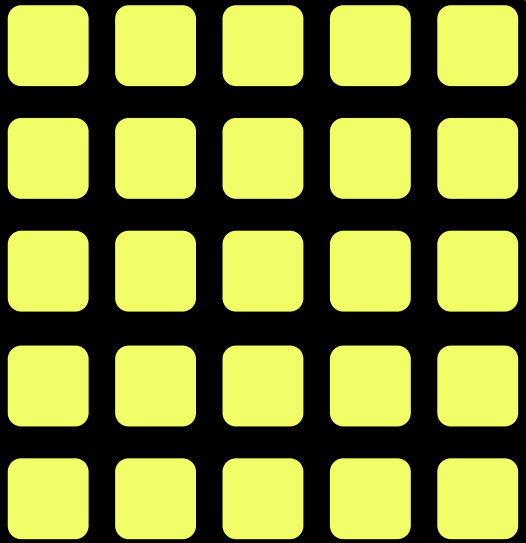
USDC



50

=

Algo



25

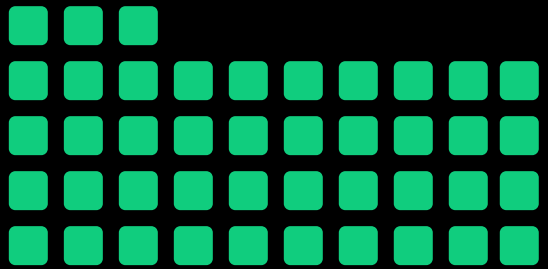
50 USDC = 25 ALGO

An asset without USDC pool?

» USD?

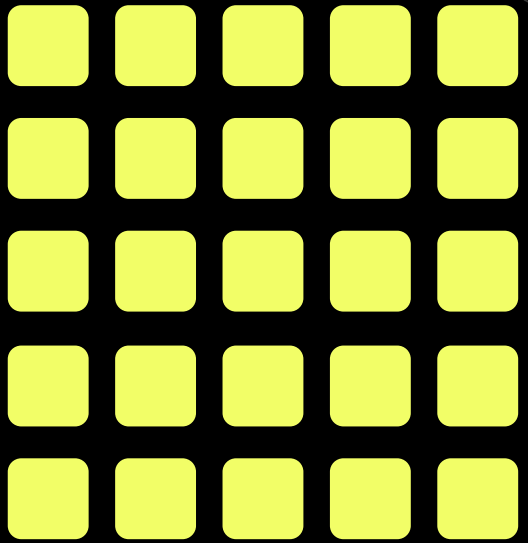
1Algo
=
1.72
TinyEuro

TinyEuro



43

Algo



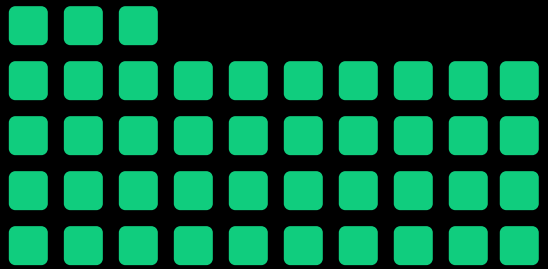
25

43 TinyEuro = 25 Algo

» USD?

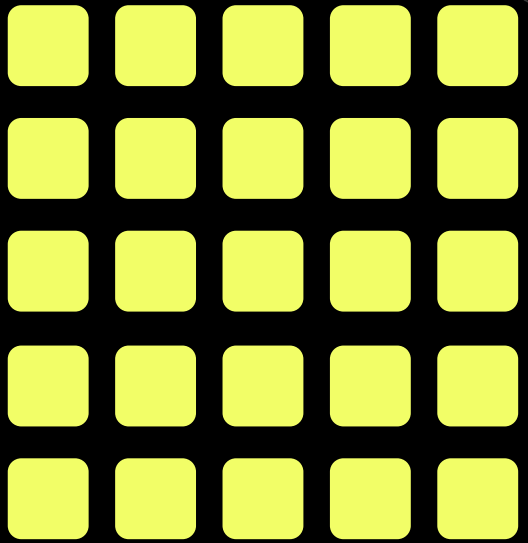
1Algo
=
1.72
TinyEuro
=
2 USDC

TinyEuro



43

Algo



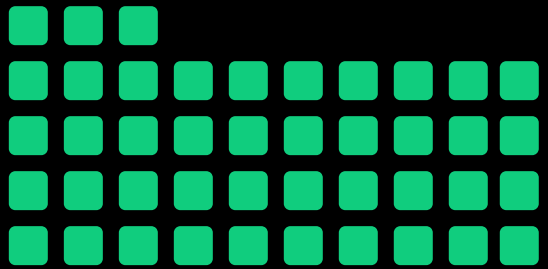
25

43 TinyEuro = 25 Algo

» USD?

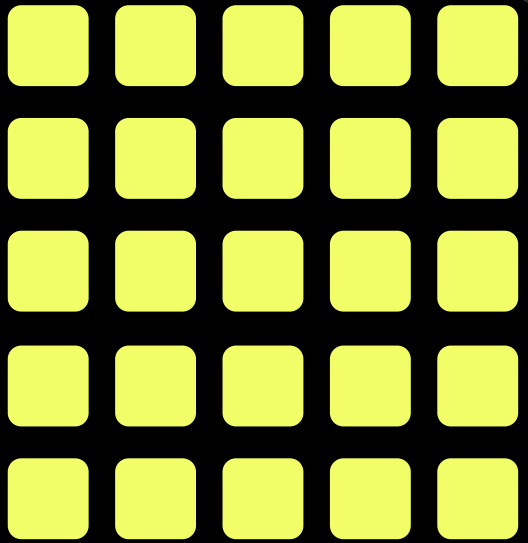
1Algo
=
1.72
TinyEuro
=
2 USDC

TinyEuro



43

Algo



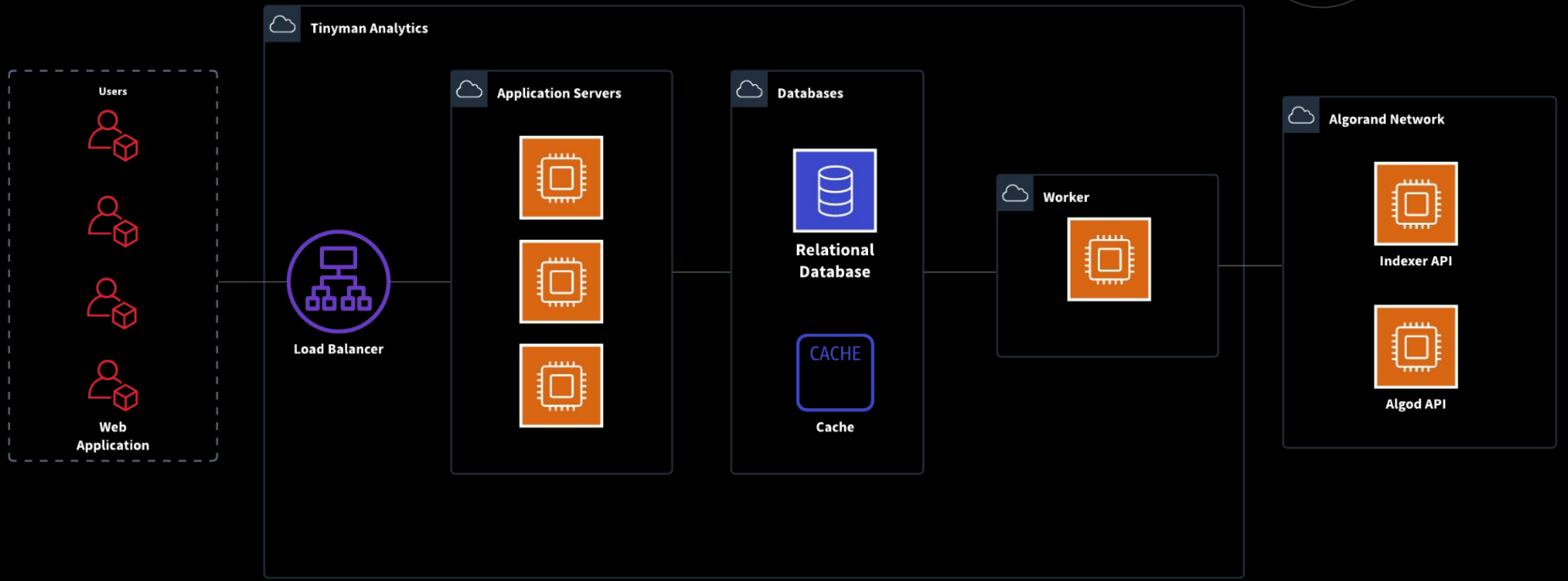
25

43 TinyEuro = 25 Algo

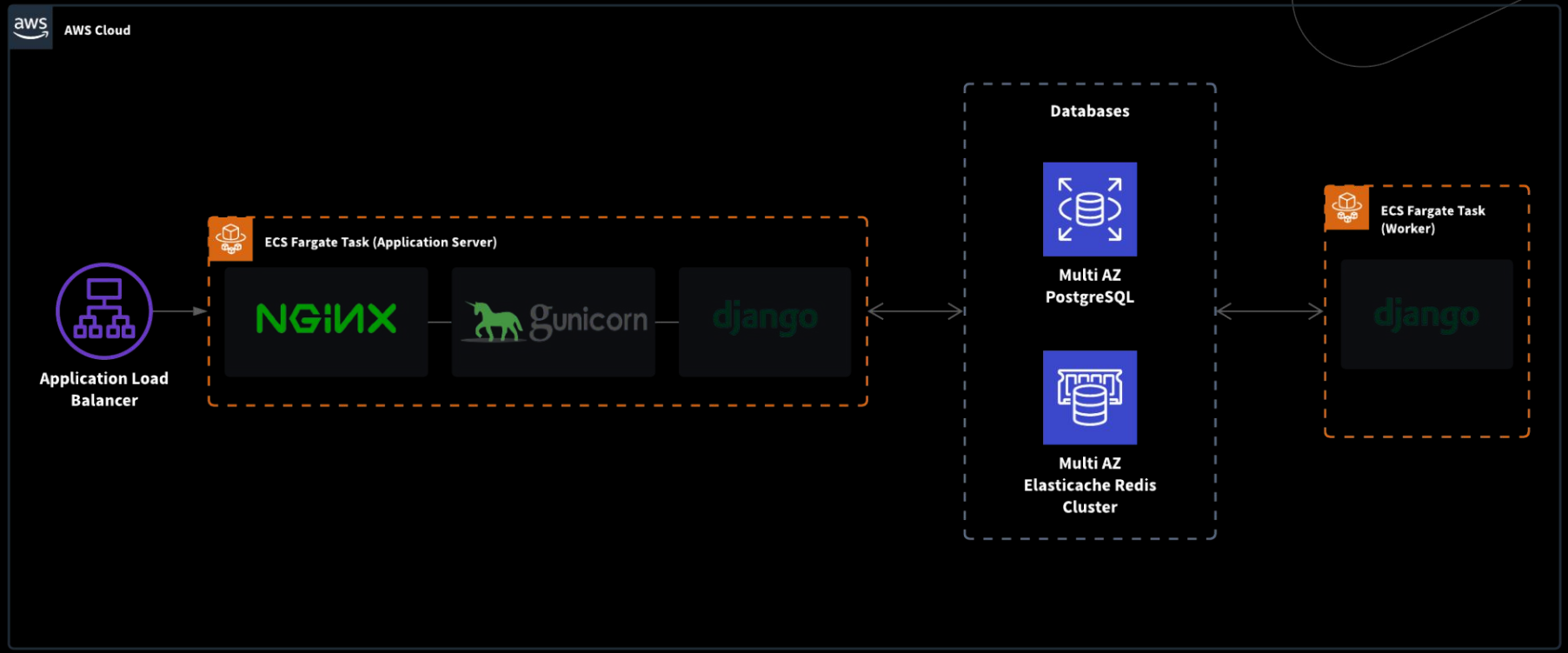
USD Values

- accepts USDC as reference currency
- uses just Tinyman pools not external API
- calculates shortest path to USDC pool directly or indirectly

Infrastructure



Running Infrastructure



How worker works?

- Following blocks
  - Bootstrap
  - Mint
  - Burn
  - Swap
  - Redeem
- Periodically recording
  - Pool
  - Asset price

How worker works?

Uses

- tinyman-py-sdk (open sourced)
- py-algorand-sdk

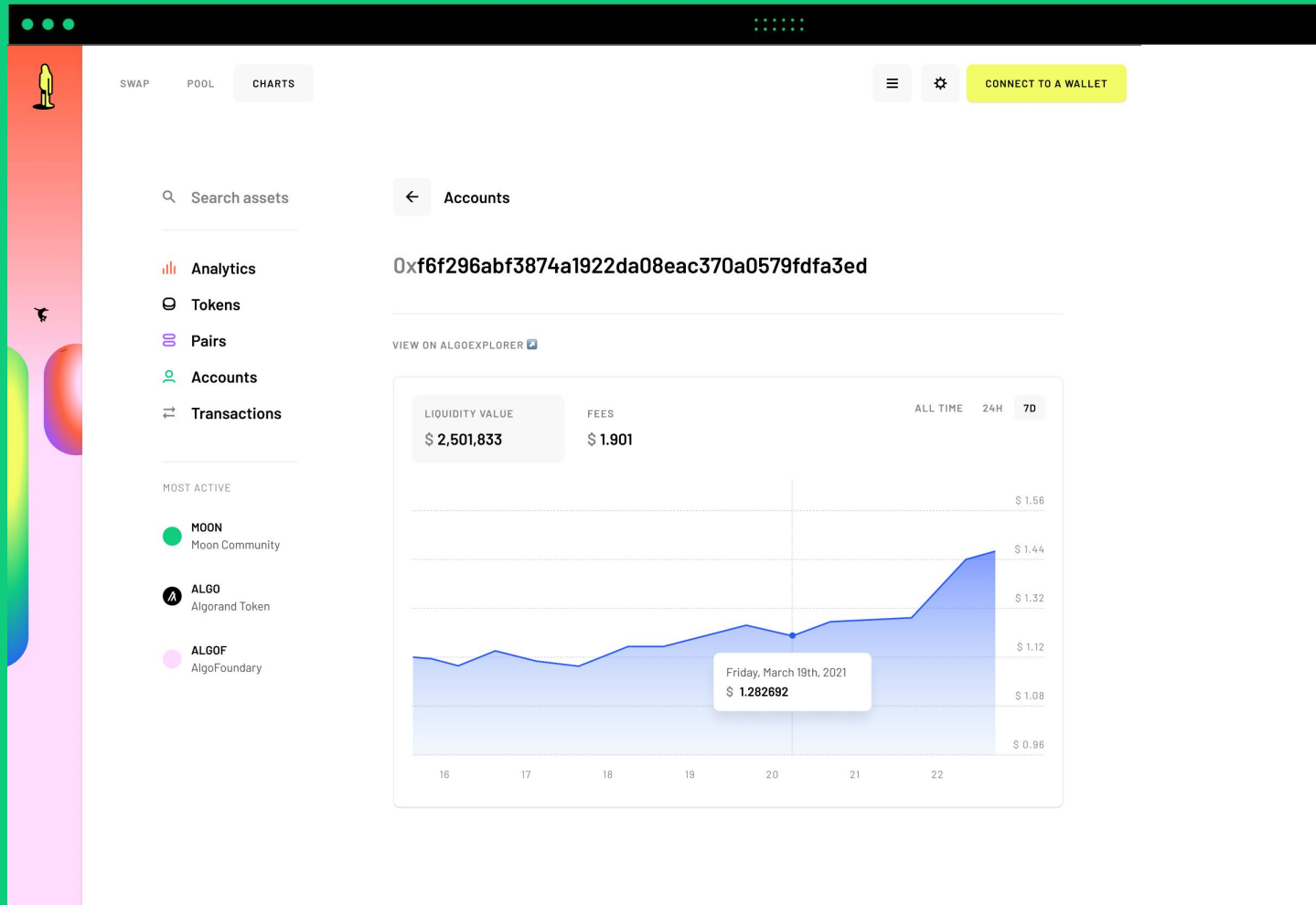
internally.

Coming soon to frontend

- Account detail analytics
- Pool detail analytics
- Asset detail analytics
- Time series data

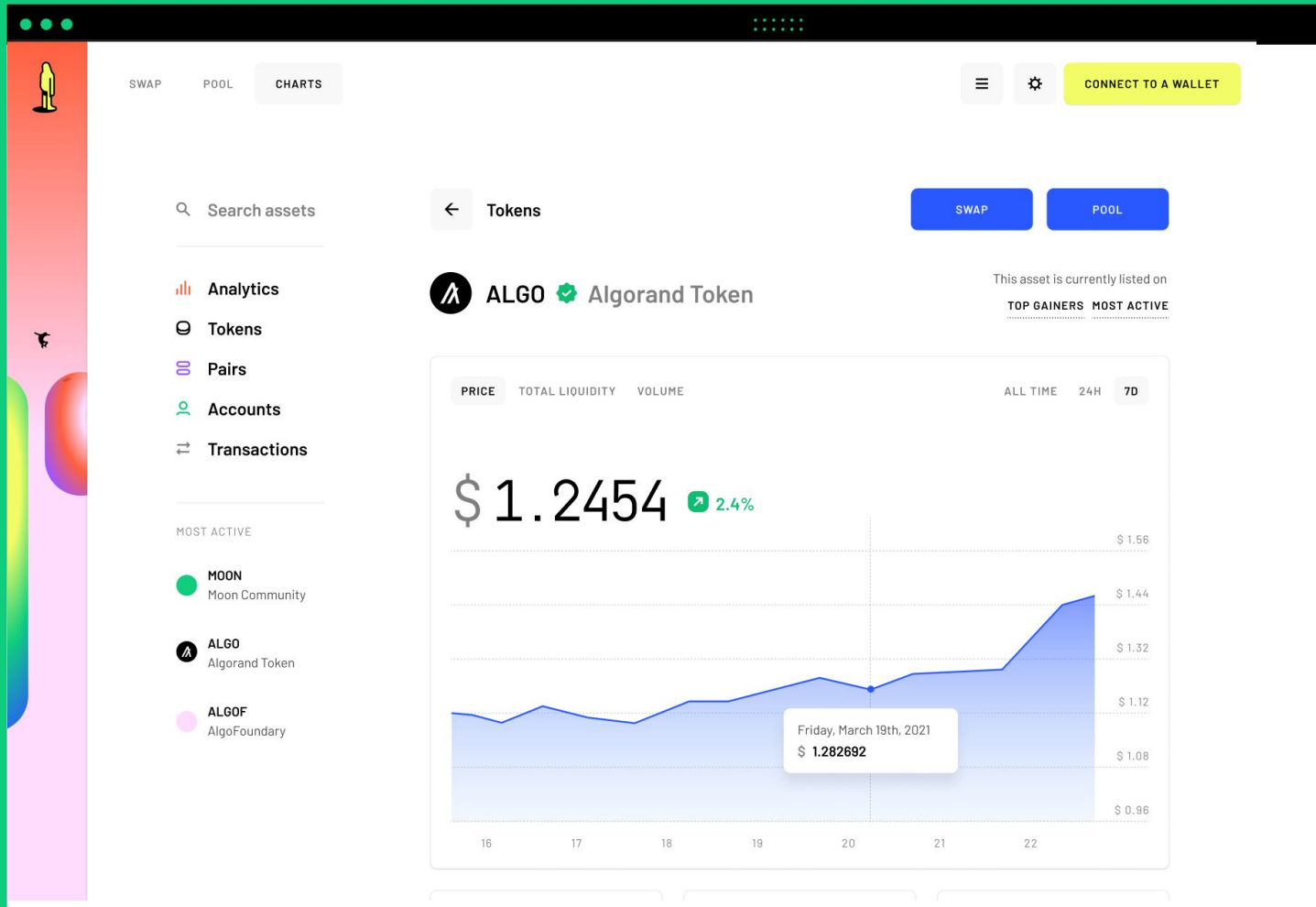
Tinyman Analytics

i. Account Detail Statistics



Tinyman Analytics

ii. Asset Detail Statistics



Future Plans on Analytics

- Open sourcing
- Restful API Reference Documentation
- Analytics SDK

» SDKs

Official SDKs created by the development team:

Python - https://github.com/tinymanorg/tinyman-py-sdk

Javascript/Typescript - In use by web app. Will be published in coming months.

Community created SDKs:

.NET - https://github.com/geoffodonnell/dotnet-tinyman-sdk.
Thanks @geoffodonnell!

SDKs

```python
from tinyman.v1.client import TinymanTestnetClient

client = TinymanTestnetClient()

# Fetch our two assets of interest
TINYUSDC = client.fetch_asset(21582668)
ALGO = client.fetch_asset(0)

# Fetch the pool we will work with
pool = client.fetch_pool(TINYUSDC, ALGO)

# Get a quote for a swap of 1 ALGO to TINYUSDC with 1% slippage tolerance
quote = pool.fetch_fixed_input_swap_quote(ALGO(1_000_000), slippage=0.01)
print(quote)
print(f'TINYUSDC per ALGO: {quote.price}')
print(f'TINYUSDC per ALGO (worst case): {quote.price_with_slippage}')
```

» Tinyman Oracle

Spot Price

TWAP - Time Weighted Average Price

Available from all pools. Can be read by other contracts to get asset price data on chain.

Roadmap

Public Testnet ✓ `testnet.tinyman.org`

Mainnet Launch

Analytics Improvements

Web UI improvements

AMM V2

Gradual continuous decentralisation

Gradual continuous open-sourcing

» Thanks

We'd like to thank all those who supported Tinyman on Testnet and provided valuable feedback.

Thanks to the community members for asking questions.

Thanks to those community members who help out others by answering questions.

tinyman.org

docs.tinyman.org

Twitter: @tinymanorg

Telegram: tinymanofficial

Github: github.com/tinymanorg/

Soon..