Sudoku application example

This is a toy example I played with to see how a simple application could work within the taiga execution model. It doesn't contain a "correct solution" but rather a history of ideas with (currently) two versions of the sudoku app design. A simplified version of this app can be found in the taiga repository (link to be added)

Roles

- dealer (D) - creates the challenge, sends the reward
- participant a.k.a solver - provides a [partial] solution to the puzzle. Each participant in the correct solution chain is rewarded. Every participant is a solver, which means they don't need to create intents

Idea

There is a sudoku application, notes of which contain the current puzzle state. Sudoku AppVP checks the puzzle state transitions and the reward size. A new puzzle note can be created either by a dealer (new puzzle) or if a state note was previously spent (update of the current puzzle state).

Steps

1. Dealer spends the reward note, creates the current puzzle note and creates an intent note:
2. the intent note can be spent, when a note with a full solution to the puzzle is presented
3. every solver in the chain supposed to claim only an amount of reward proportional to their contribution. Here the rule can be different (e.g. everyone could be rewarded equally) but I just chose this one
4. the intent note can be spent, when a note with a full solution to the puzzle is presented
5. every solver in the chain supposed to claim only an amount of reward proportional to their contribution. Here the rule can be different (e.g. everyone could be rewarded equally) but I just chose this one
6. Every solver spends the current puzzle state note and creates a new puzzle state note.
7. The sudoku appVP checks the state transition and the solver's reward size.
8. Once the note with full solution (with 0 value) is created, the dealer's intent is satisfied and the transaction can be finalized.

[

4136×1875 742 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/d38a9cff486ba1380240442bcc140b6c2e440059.png)

Partial tx

created notes

spent notes

VPs checked

balance

Initial ptx

a puzzle note S_1, an intent note I

reward

R tokenVP, Sudoku AppVP, Intent App VP, dealer's intentVP

R - [1]S_1 - [1]I

Alice ptx

new puzzle note S_2, Alice reward

old puzzle note S_1

Sudoku AppVP, R tokenVP

4R/6 - [1]I - [1]S_2

Bob ptx

new puzzle note S_3, Bob reward

old puzzle note S_2, intent note

R tokenVP, Sudoku AppVP, Intent App VP, dealer’s intentVP

[0] S_3

Some details

Intents

As I remember the sudoku app was supposed to be an example of usage of intents, but as all of the users are also solvers, there is no need for them to use intents, so only dealer has an intent. Dealer’s intent makes sure that the transaction isn’t finalized until the puzzle is solved.

Avoid using sudoku app & use more intents instead

I considered using intents instead of the sudoku app, but it didn’t make much sense as even if we move the appVP logic to intentVP, the notes with the state still need a type and making them intent notes seems confusing and unnecessary.

Mistake correction

In this simple version of the app, mistakes correction isn’t allowed (and mistakes are not allowed), and users can only follow one chain of solution. There can only be one current state note. Not perfect, but simple.

Sudoku application v2

This is an updated version of the sudoku application design above aiming to solve the problem of cheating. The previous design didn’t have a proper mechanism to prevent it, but hopefully, this one does.

The problem and the solution

The design of the sudoku application is based on having a single valid state the users would update until eventually the final puzzle solution is achieved. The problem of the previous design was that there was no way to commit to the current state, so nothing would actually prevent malicious participants from updating the old state, stealing the results of other participants and getting their reward.

There is a natural way to commit to a state - publish it on the blockchain. The new design uses this approach, forcing the participants to update the latest published state.

The new design

There is a sudoku application, unspent notes of which contain the current puzzle state. Sudoku app checks the puzzle state transitions. Intent app is responsible for the reward distribution.

Roles

- dealer (D) - creates the challenge, sends the rewards
- participant a.k.a solver a.k.a user - provides a [partial] solution to the puzzle. Each participant in the correct solution chain is rewarded. Every participant is a solver, which means they don’t need to create intents

Setup

- N - the sudoku number, e.g. for 9 \times 9 sudoku puzzle N = 9

- values of empty cells are equal to zero, non-empty cells have values 1..N

Steps

[

3335×1681 163 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/d4a248cf9961b168ff4771814ab7a9035205f5a8.png)

1. Init

: Dealer creates the new puzzle note[1]

1. Reward

: dealer creates a reward intent, offering the leftover reward amount.

1. Solving round

: solvers propose their [partial] solutions by spending the current state and creating a new one and send them to the intent gossip network. One of the solutions gets accepted, the new state is published on the blockchain, the reward is sent to the solver.

[

2820×1292 361 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/10aa07d8ff543bf7ccfc9bfc2742e0249a5c3c16.png)

Partial tx

created notes

spent notes

VPs checked

balance (accumulated)

Dealer ptx

[0]Sudoku-init-state, [1]Intent

[R]Reward

Reward appVP, Reward Dealer's userVP, Sudoku appVP, Intent appVP, Dealer's intent userVP (4-5)

[R] - [0]Sudoku - [1]Intent

Alice ptx

[0]Sudoku-new-state, [R - R1]Reward, [R1]Reward

[0]Sudoku-init-state, [1]Intent

Sudoku appVP, Reward appVP, Reward Dealer's userVP, Reward Alice's userVP, Intent appVP, Intent Dealer's userVP(4-6)

0

1. Repeat

: If the solution is not final, repeat steps 2-3.

[

2818×1162 351 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/8f738ef7e114a58e54c529824df3944503e9d804.png)

Partial tx

created notes

spent notes

VPs checked

balance (accumulated)

Dealer ptx

[0]Sudoku-updated-state, [1]Intent

[R - R1]Reward

Reward appVP, Reward Dealer's userVP, Sudoku appVP, Intent appVP, Dealer's intent userVP(4-5)

[R - R1] - [0]Sudoku - [1]Intent

Bob ptx

[0]Sudoku-final-state, [R - R1]Reward

[0]Sudoku-updated-state, [1]Intent

Sudoku appVP, Reward appVP, Intent appVP, Intent Dealer's userVP, Reward Dealer's userVP, Reward Bob's userVP (4-6)

0

Reward distribution

- tc
- the total amount of cells in the puzzle. For - 9\times 9

puzzle tc = 81

- ic
- initial amount of non-zero puzzle cells
- ec
- the current amount of empty cells
- R
- total reward
- r = \frac{R}{ic}
- the reward per one cell

Each round the dealer creates an intent, offering rewards of size 1 * r, 2 * r, ... , ec * r

for setting the value of 1, 2, ... , ec

cells respectively. The correctness of the values is checked by the Sudoku appVP

(not against the existing solution, just for not violating the rules).

The dealer spends the reward they have left and when the state is updated, the solver receives the proportional reward and sends the leftover reward back to the dealer. When the puzzle is solved, nothing is sent back to the dealer.

The Sudoku application

The new proposed state is valid if it is correct and extends the current puzzle state. Sudoku application notes have zero value, but the solver can't spend them without satisfying the VP of the Sudoku app (so the state must be valid) and the intent app (so the reward must be distributed correctly).

A new puzzle note can be created either by a dealer (new puzzle) or if a state note was previously spent (update of the current puzzle state).

1. Note that the first state doesn’t have to be published on the blockchain↩