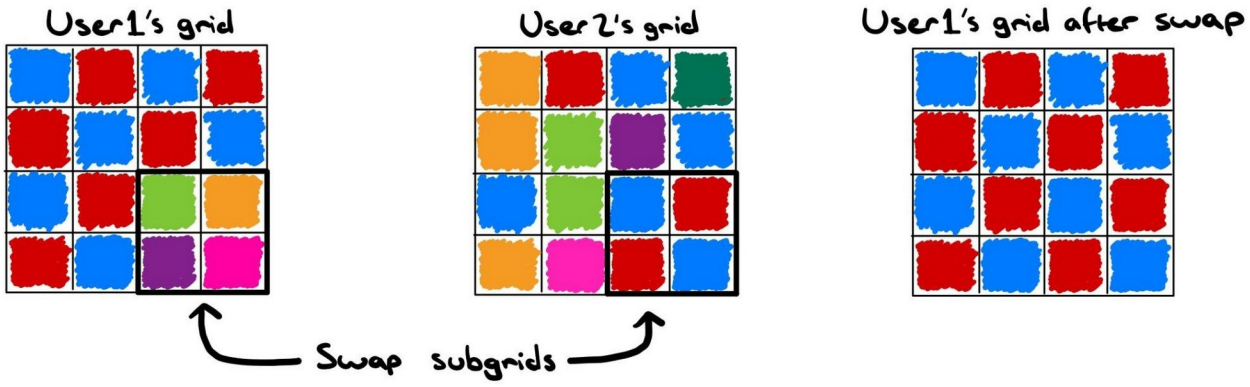
ColoredGrid dApp Concept

Executive summary

Users collect 4x4 grids (simply called grids), and each grid has nine 2x2 grids (called subgrids). Each slot on the grid can contain one of 10 colours and the objective is to exchange/create/modify grids with the objective of accumulating grids with distinct patterns/designs. Grids are created by paying a flat fee (currently planned to be 0.001 eth) and in return a randomly generated grid is given to the user. The randomly generated grids do not often have any pattern/design to start with. A main feature is the ability to exchange subgrids with other users, with the objective of finding grids that belong to others where their subgrid may create a pattern/design if they were swapped.

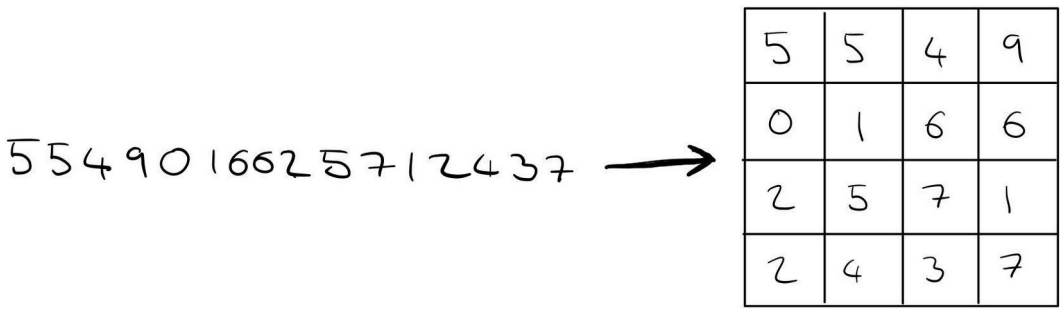


Four grids can be put together in a ‘set’ making an 8x8 grid allowing for users to create more complex patterns/designs. This does not permanently affect the grids and is more akin to a ‘collection’. The overall concept is that the goal for users is to accumulate distinct designs/patterns of grids to create sets. This is done through exchanging grids/subgrids with other users and randomly generating new grids.

Techincal summary

Grid

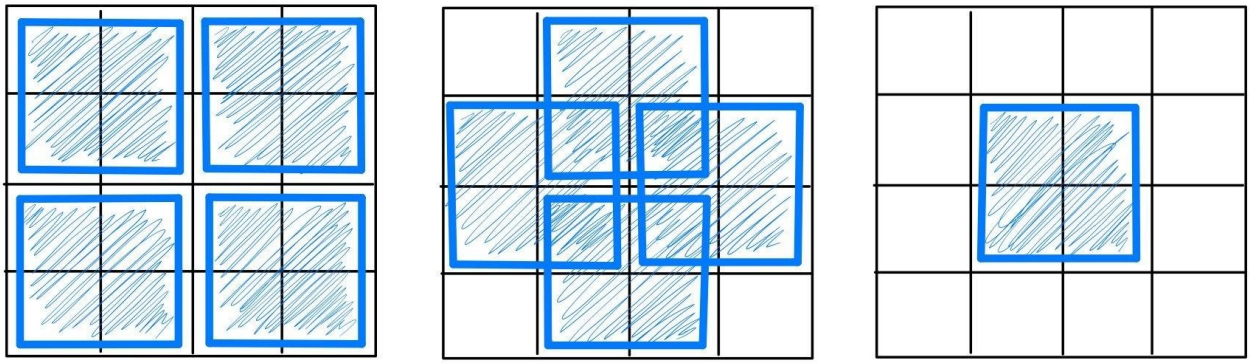
The grid is visually represented as a 4x4 grid but in data it is stored as an integer which is 16 digits long. Each digit in the integer represents a colour, so there are 10 color choices. When a grid is created, an integer of length 16 is randomly created. From now of the contents of grids will be referenced its numerical value, not the color. Shown below is an example of an integer and how it relates to a visual grid.



A 16 digit integer has been chosed to store the grid data instead of a 2D array due to gas and storage considerations. Slightly more complicated math is needed to access and make changes to the grid data but I believe that this is a reasonable tradeoff.

Subgrids

A subgrid is a 2x2 area that exists within a grid. There are 9 subgrids in every grid. Shown below is a summary of all 9 possible subgrids.

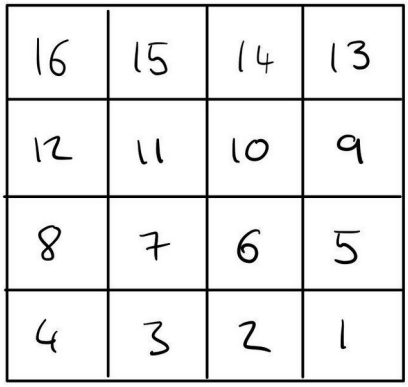


Subgrids have been chosen as a way to modify grids instead of individual pixels since it is too easy to simply find another grid which happens to contain a pixel with a particular color in a particular location. Making users trade with subgrids adds more scarcity to the elibible grids that a user would exchange subgrids with.

Accessing subgrid data

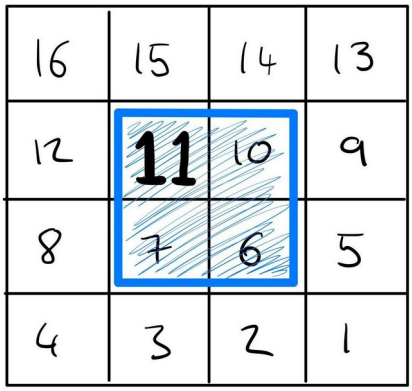
Since the grid data is stored as an integer instead of an array, we can't simply use gridData[x][y] to access data. To address this, there is a formula that can be used to access the contents of a subgrid.

Before this fomula can be shown, we need to be able to identify each subgrid. As we know from the summary on subgrids, there are 9 subgrids to a grid. We can uniquely identify each subgrid by the position of its top-leftmost value. We use the grid style to the right. Just like the description about the grids, the most significant digit starts at the beginning, and the least significant an the end. The only different is that on this grid, we show how many digits are between this current point in the integer and the least significant digit.



EG: 5549016625712437 is 16 digits long. 49016625712437 is 14 digits long. 2437 is 4 digits long.

So if we refer to the third diagram with the single subgrid from the subgrid section, we would be able to identify that subgrid by the value 11. This can be applied to every subgrid, and from that we can know all valid subgrids by their identity value.



The valid subgrids are:

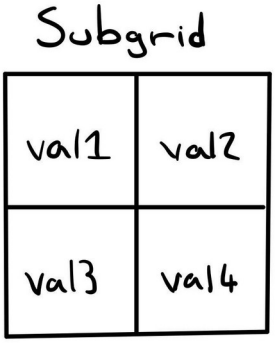
16, 15, 14, 12, 11, 10, 8, 7, 6

Knowing identifiers for every subgrid, now we can simply use this formula to identify the contents of a subgrid.

NOTE: The division in this formula is floor division and the ‘%’ operator is modulus.

g = grid data (the 16 digit integer)

i = subgrid identifier for the subgrid that we are checking



val1 = $(g / 10^{i-1}) \% 10$

val2 = $(g / 10^{i-2}) \% 10$

val3 = $(g / 10^{i-5}) \% 10$

val4 = $(g / 10^{i-6}) \% 10$

Modify a grid by changing a subgrid

When an exchange of subgrids between two users occurs, the grid data integer value must be changed to reflect this. The logic for changing grid data based on a new subgrid is as follows.

NOTE: subgridTopUpdate is the gridData after val1 and val2 have changed, but not val3 and val4.

g = grid data (the 16 digit integer)

i = subgrid identifier for the subgrid we are replacing

nt = new top values in the subgrid. This is equal to ((val1 * 10) + val2)

nb = new bottom values in the subgrid. This is equal to ((val3 * 10) + val4)

subgridTopUpdate = $((g / 10^{i}) * 10^{i}) + (nt * 10^{i-2}) + (g \% 10^{i-2})$

subgridUpdateDone = $((subgridTopUpdate / 10^{i-4}) + (nb * 10^{i-6}) + (subgridTopUpdate \% 10^{i-6})$

Sets

A set is simply an 8x8 grid which is comprised of 4 grids. The purpose of sets is to allow users to emperiment with more complex patterns/designs. Sets simply reference to four individual grids, and does not make any changes to any data. A grid can only belong to one set at a given time.

Send/Exchange

Send:
A party can send an item to another party without the second party having to offer anything
Items that can be sent:

- Sets
- Grids

Exchange:
A party sends an item to another party and the second party must offer the same type of item back.

Items that can be exchanged:

- Sets
- Grids
- Subgrids

The reason that subgrids can only be exchanged is to avoid having subgrids that don’t belong to a grid. Note that when exchanging subgrids, they must both have the same subgrid identifier. Users can offer ether along with an exchange to incentivise the user to accept the trade. This is because there may be times when a user needs a subgrid, but the exchange will only benefit one user wheni t comes to how the grids will look after the exchange. If a recipient of an exchange offer likes the trade idea but wants more ether, they are able to send counteroffers back to the user

Events

Events will be emitted to track activity from users and to know when to make changes to the front-end user interface. These are the following events:

- Generate new grid
- Send set/grid
- Exchange offered (counteroffers can trigger this too)
- Exchange accepted
- Exchange declined
- Exchange withdrawn
- Set created
- Set deleted