https://github.com/loda-lang/loda-rust

ARCathon 2023 “notXORdinary”

Genetic algorithm + Logistic regression = score 8

- Genetic algorithm: 5 tasks solved by the existing solutions.
- Genetic algorithm: 1 task solved by mutating the existing solutions.
- Logistic regression: 2 tasks solved. **Most interesting and what I explain.**

Simon Strandgaard **neoneye@gmail.com** **2023 nov 20, version 1**

I use logistic regression for predicting the pixel colors.

There are 11 classifications: 1 class per color, and 1 class for padding.

The number of table columns is around 1700 floats, 64bit. Most of the columns are one-hot encoded, either being 1.0 or 0.0. Changing to 32bit and the number of solved tasks drops, so 64bit helps solving tasks.

Logistic regression fit

This table generated from **train** pairs.

The number of table rows correspond to the number of pixels, across the input images only for the train pairs.

Each row is a classification: either the output color or the padding color.

The logistic regression fit function is set to max 50 iterations.

Logistic regression predict

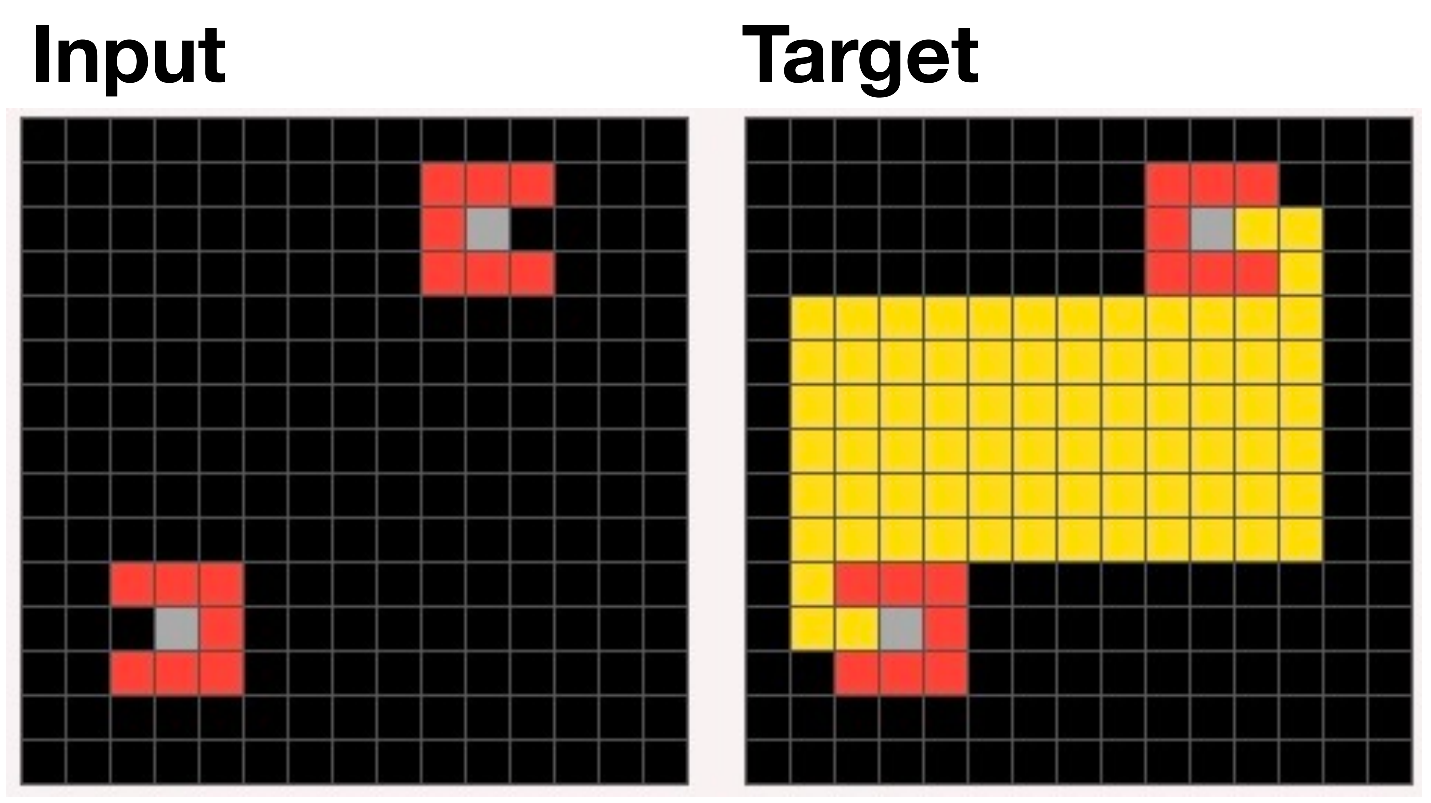
This table generated from **test** pairs.

The number of table rows correspond to the number of pixels, across the input images only for the test pairs.

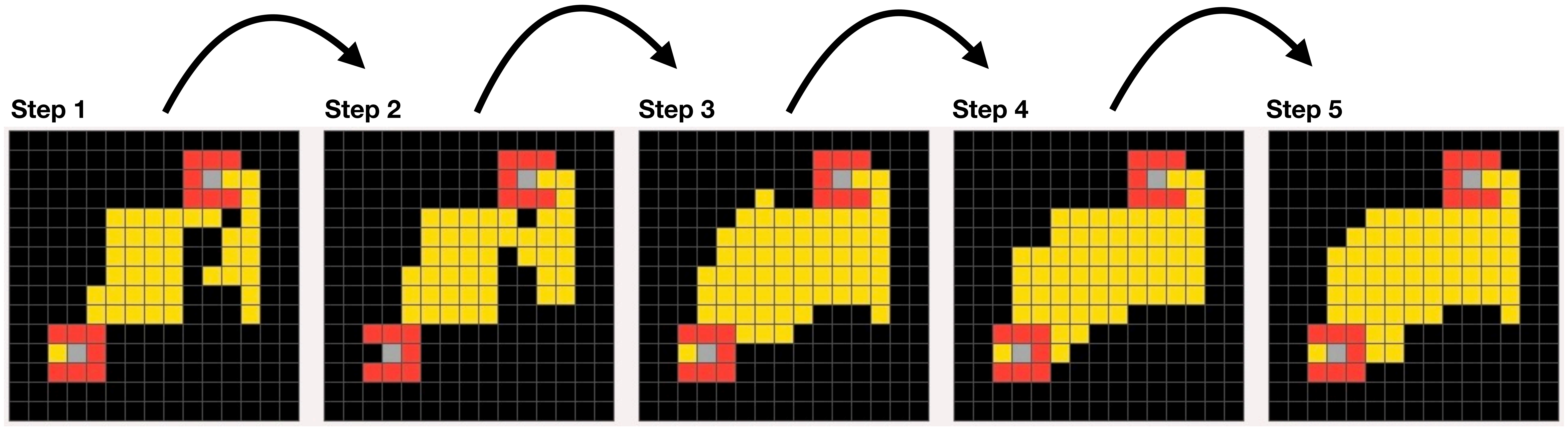
Predicts a classification for each row.

5 steps of refinement

692cd3b6



Predictions from previous step goes into next step.



initial guess is bad

final prediction is better

Fake terrible predictions for train pairs

Sending predictions from the previous step to the next step is not possible with **Logistic regression fit**.

The **train** pairs have an output image and I can send this as a prediction to the next step. However sending the correct pixel worsens the ability to solve tasks. It’s better to start with junk, and then send slightly cleaner data and in last step send the correct output.

In step 1, there is no previous step, so this field is set to zero.

In step 2, take 10% of the correct output pixels, take 10% of original input pixels and take 80% noise to the next step.

In step 3, take 30% of the correct output pixels, take 10% of original input pixels and take 60% noise to the next step.

In step 4, take 50% of the correct output pixels, take 5% of original input pixels and take 45% noise to the next step.

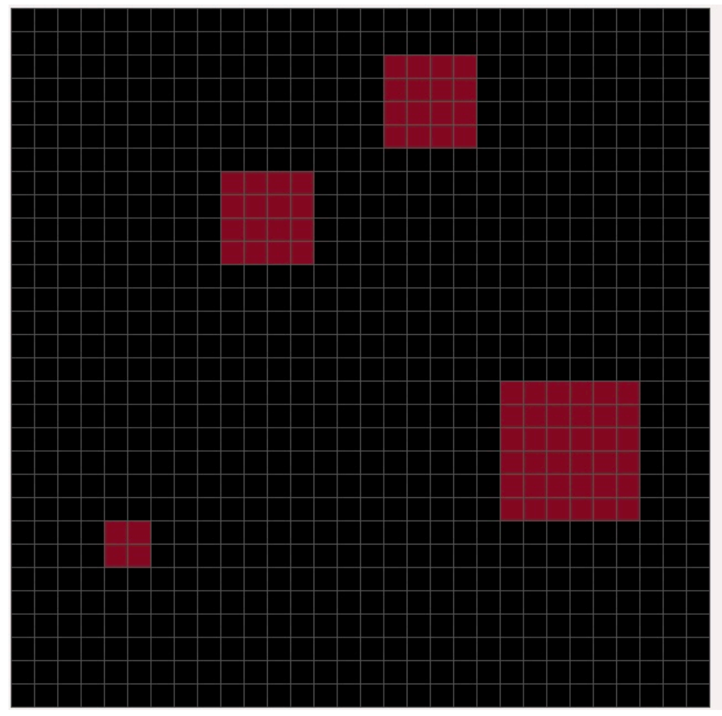
In step 5, take 80% of the correct output pixels, take 3% of original input pixels and take 17% noise to the next step.

db93a21d

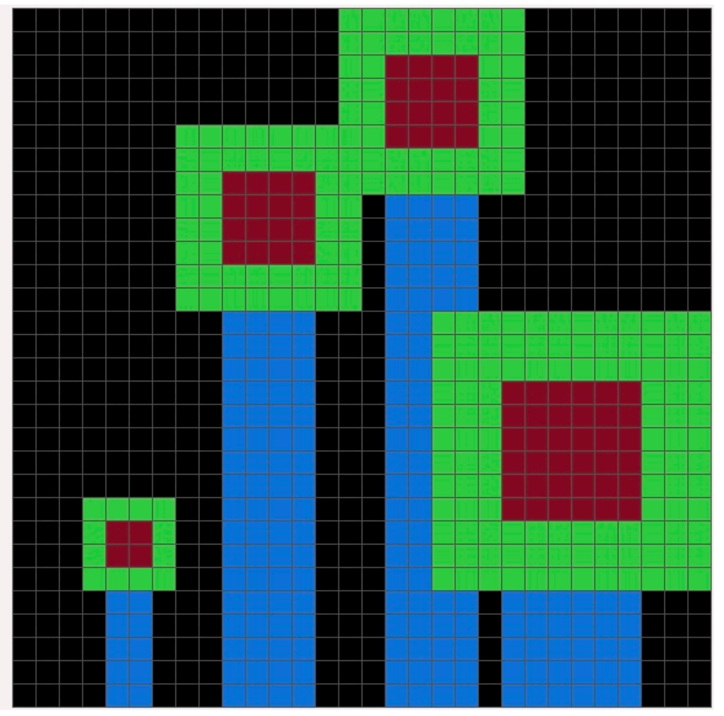
The 1700 columns for logistic regression.

One of the columns is the **pair-id** encoded as an integer. It’s not one-hot encoded. And it works surprisingly well for learning concepts across multiple pairs.

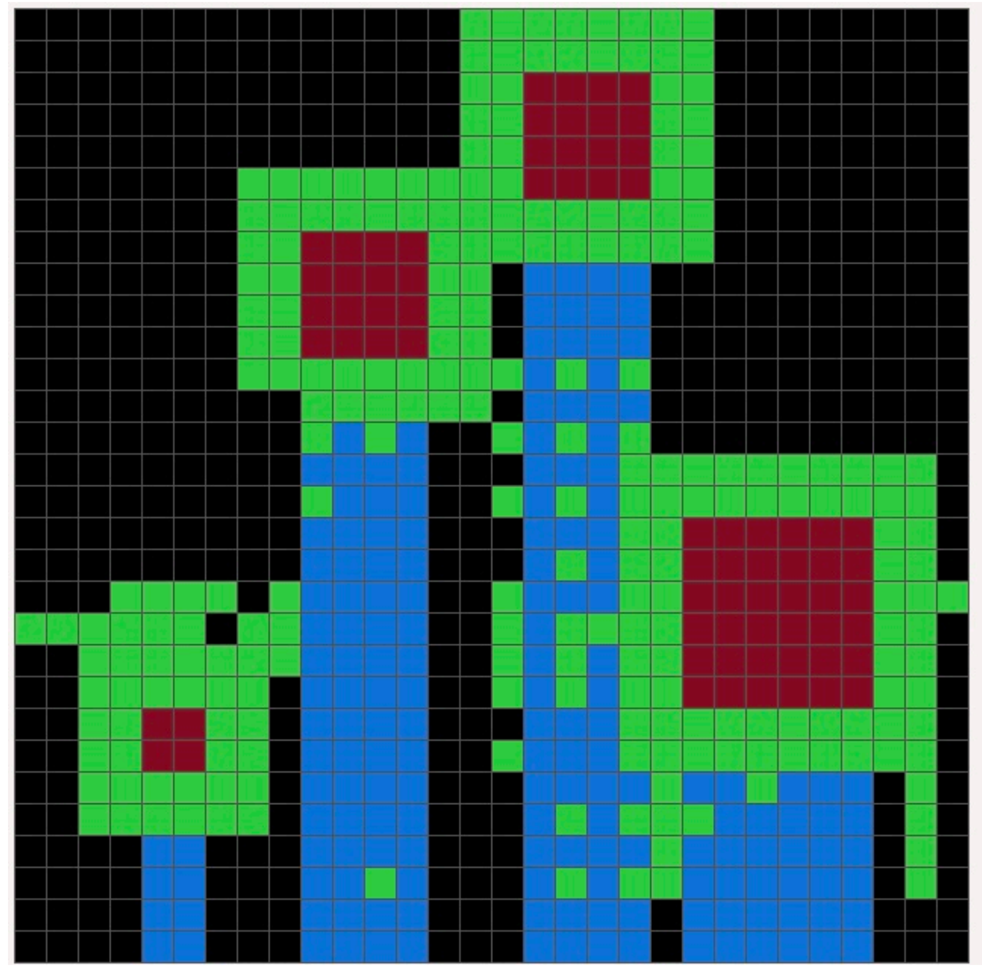
Input



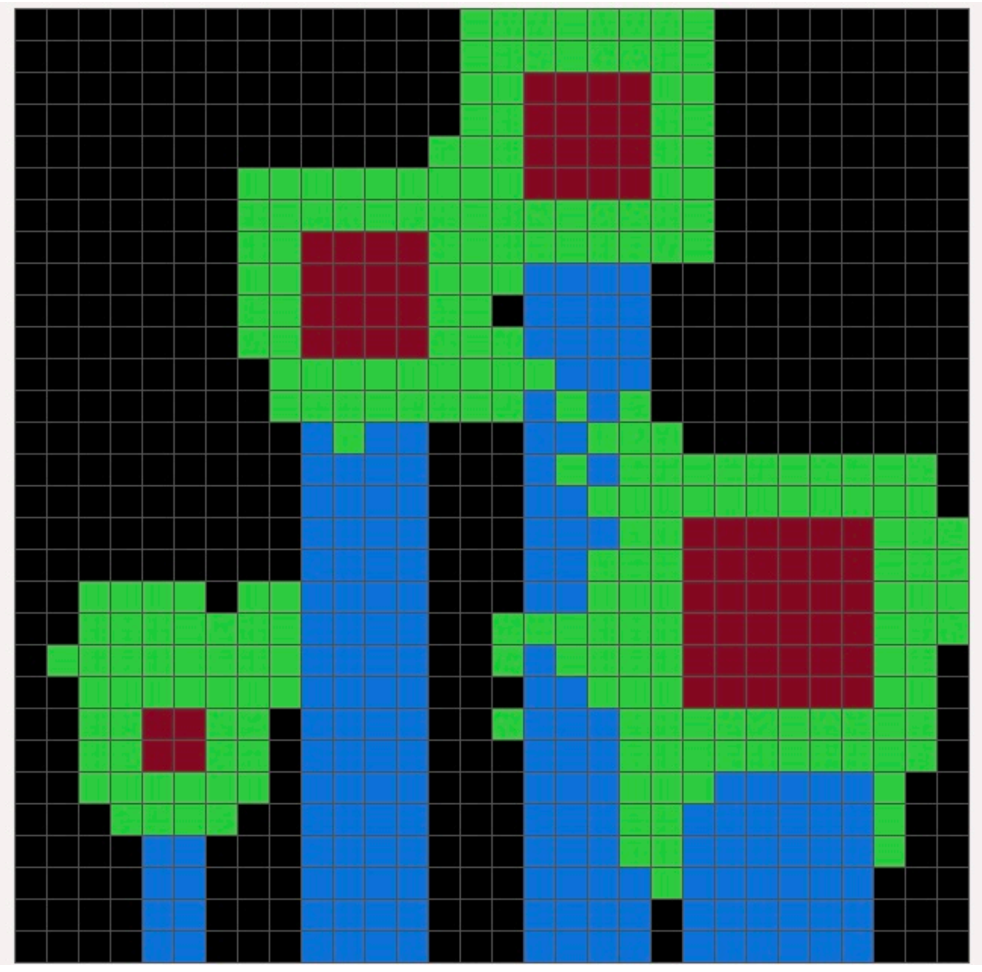
Target



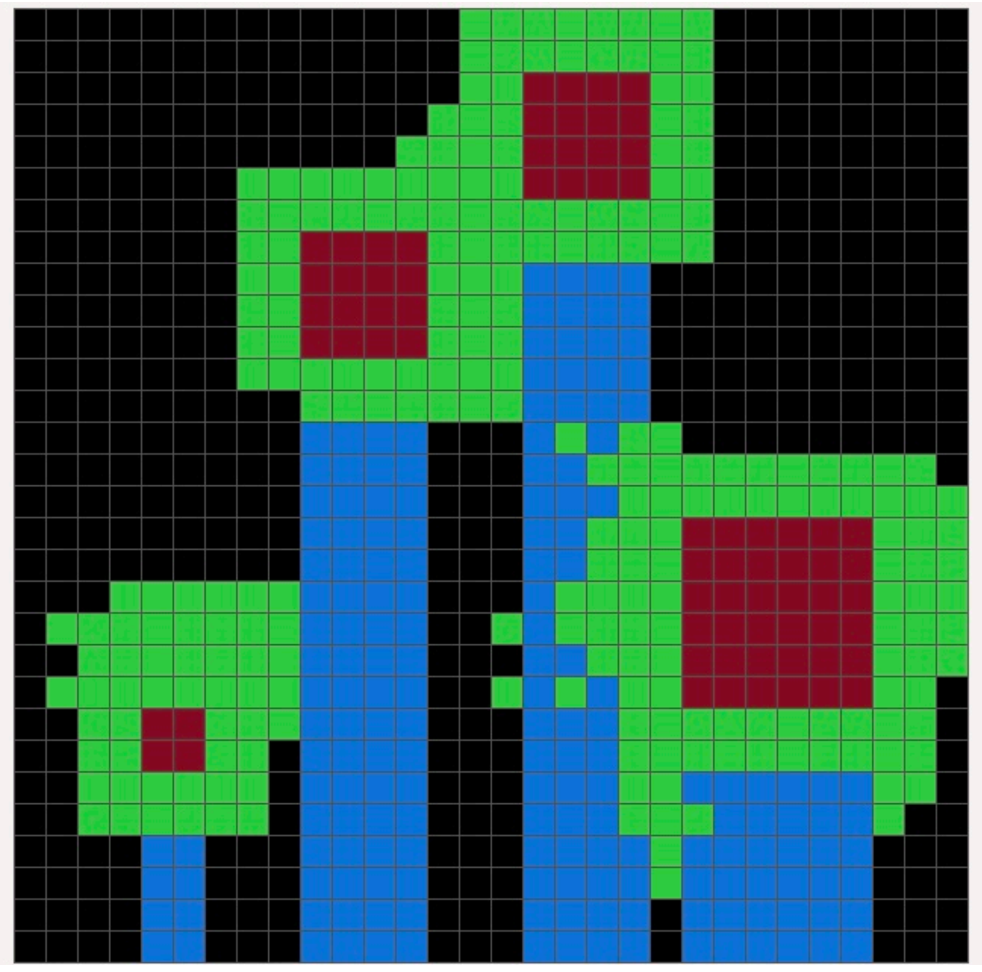
Step 1



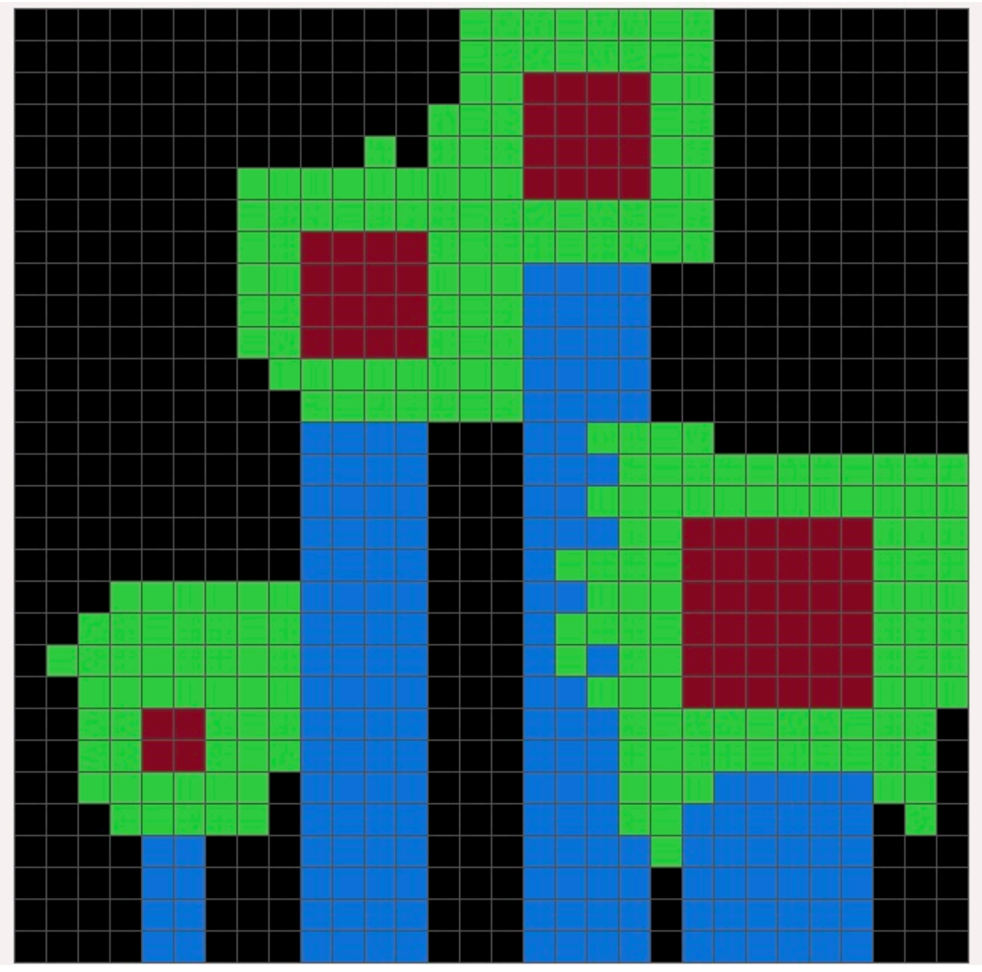
Step 2



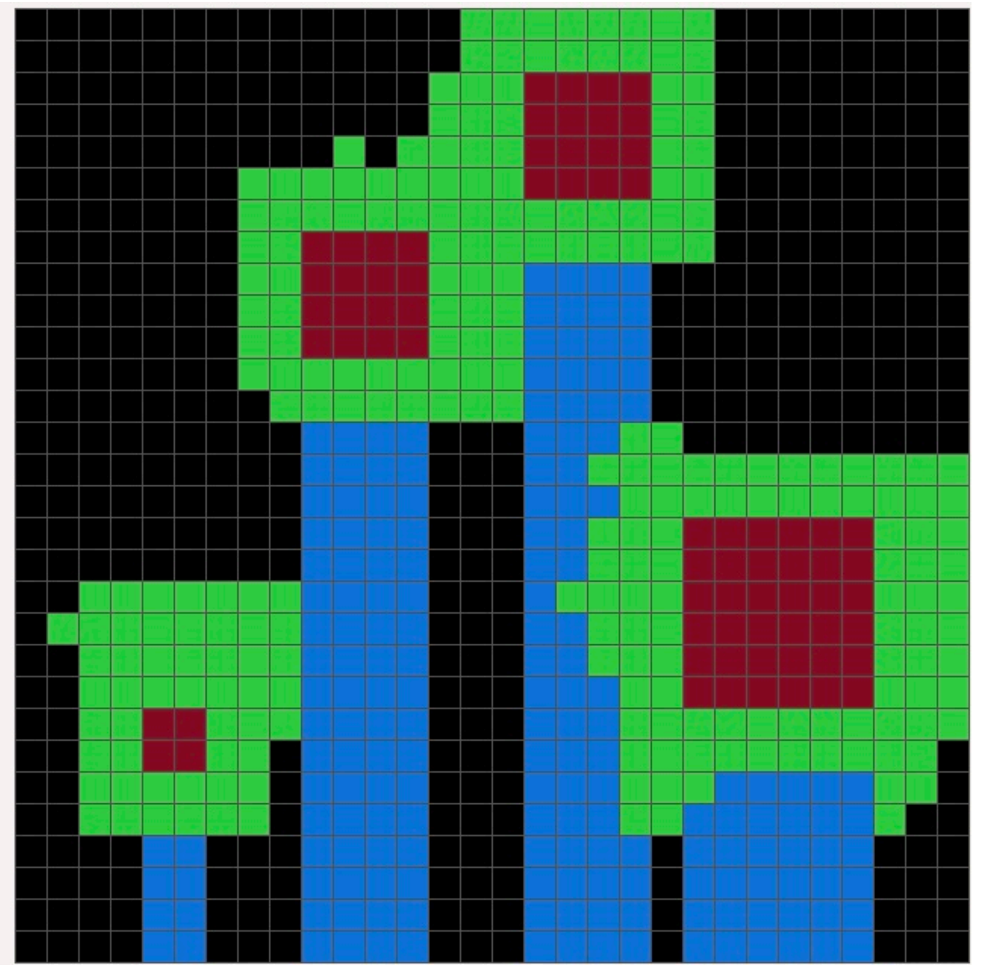
Step 3



Step 4

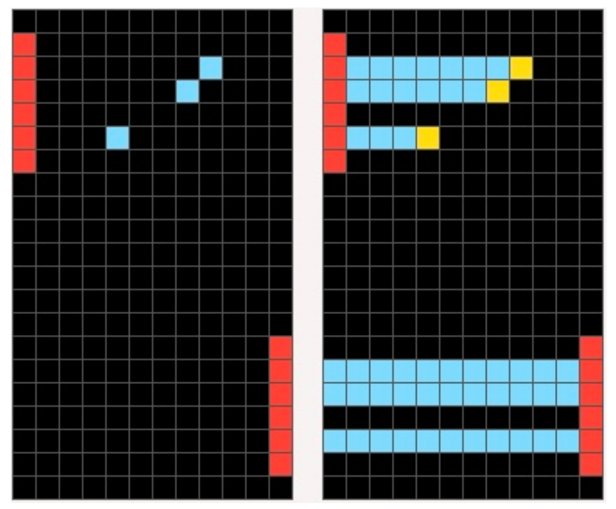


Step 5



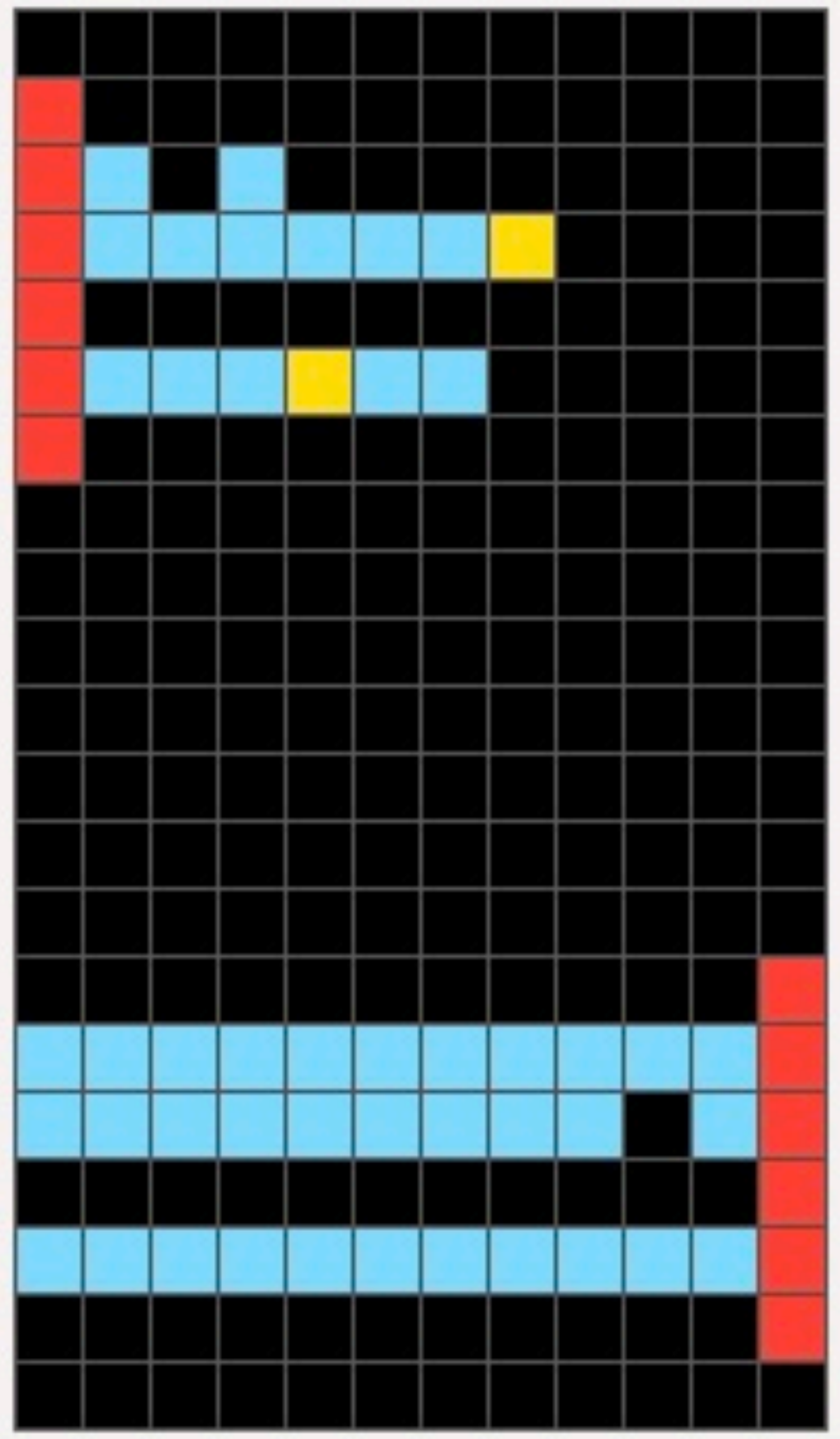
673ef223

Input Target

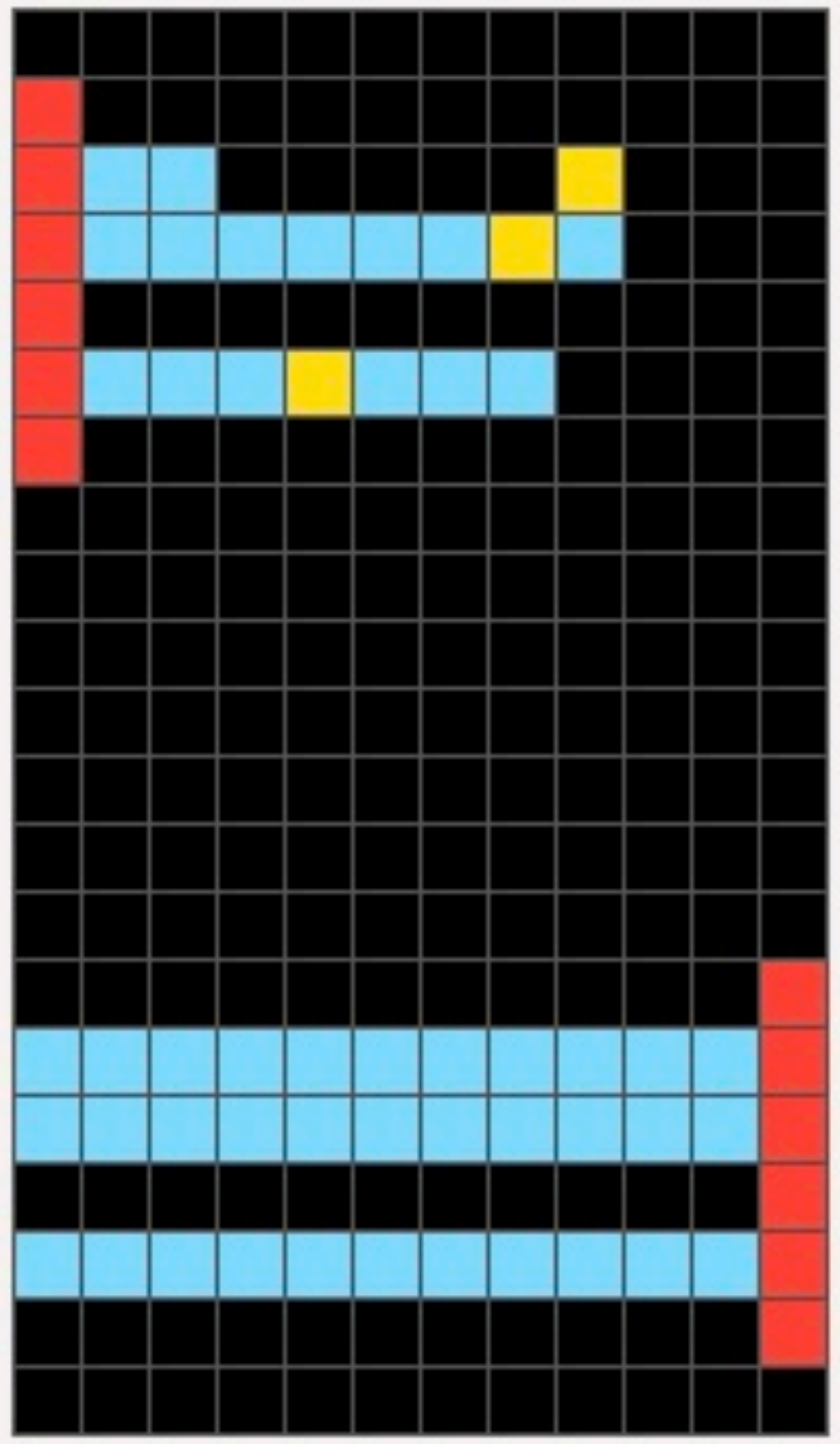


Sometimes it gets close at solving a task

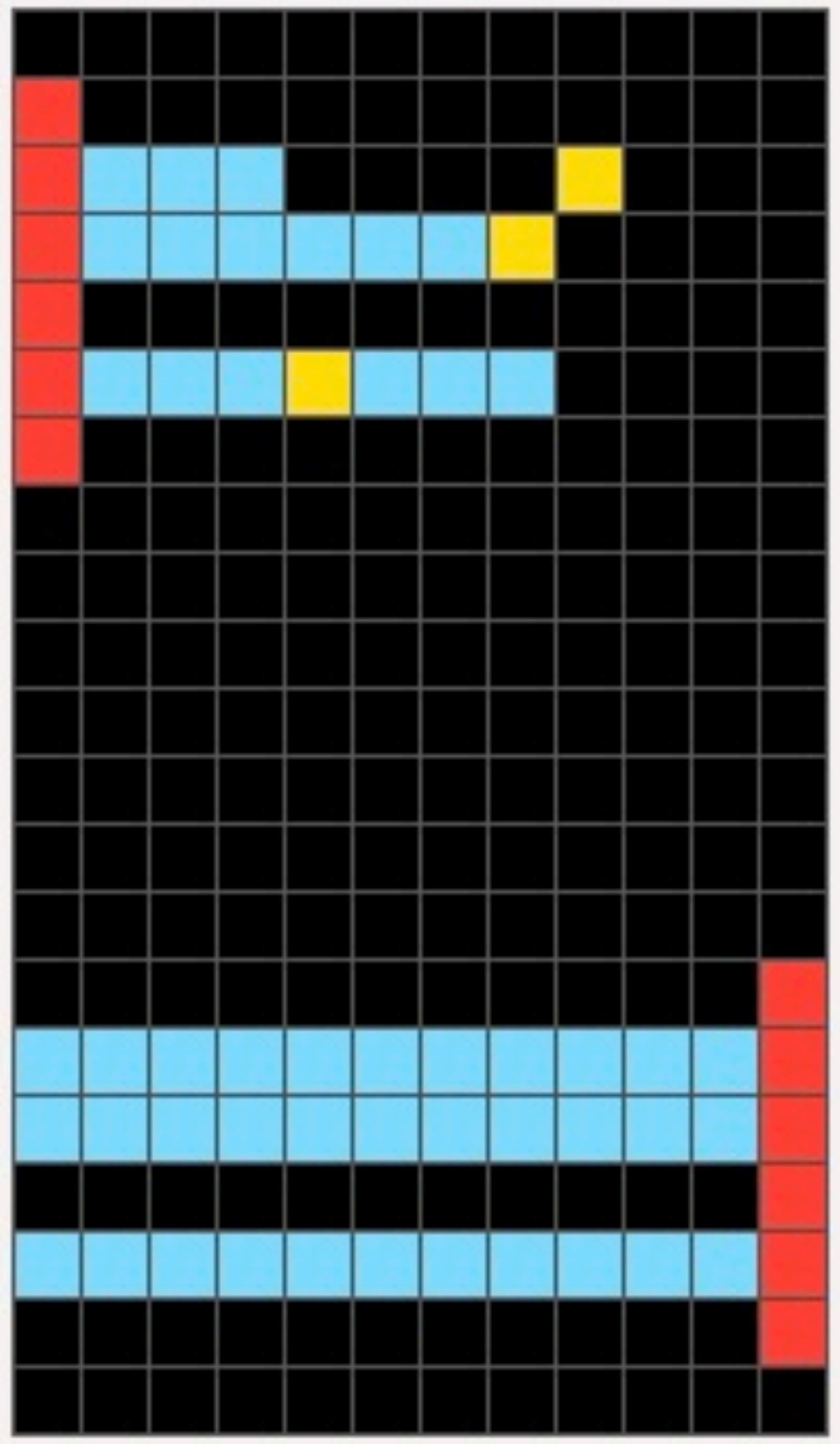
Step 1



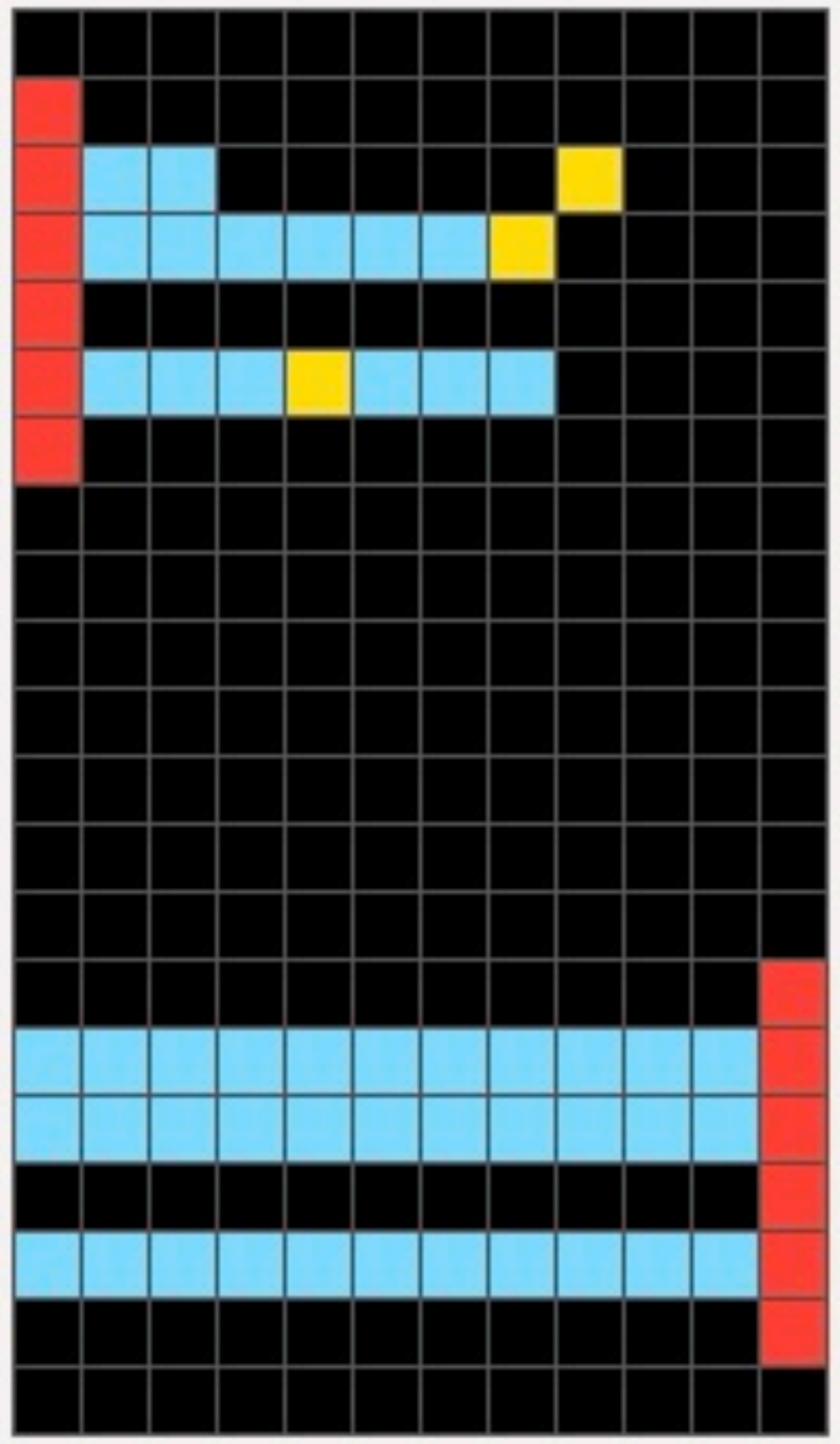
Step 2



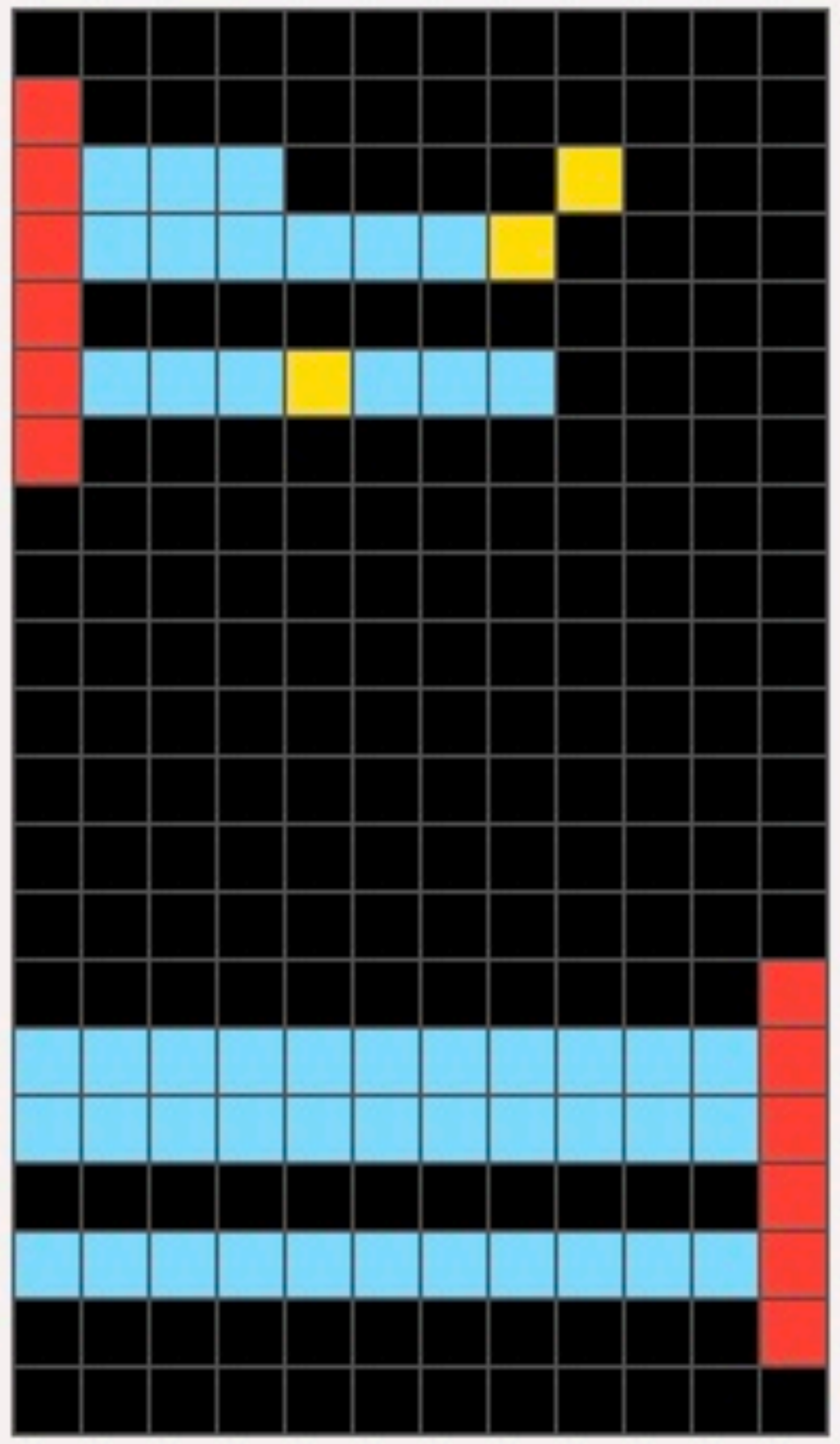
Step 3



Step 4

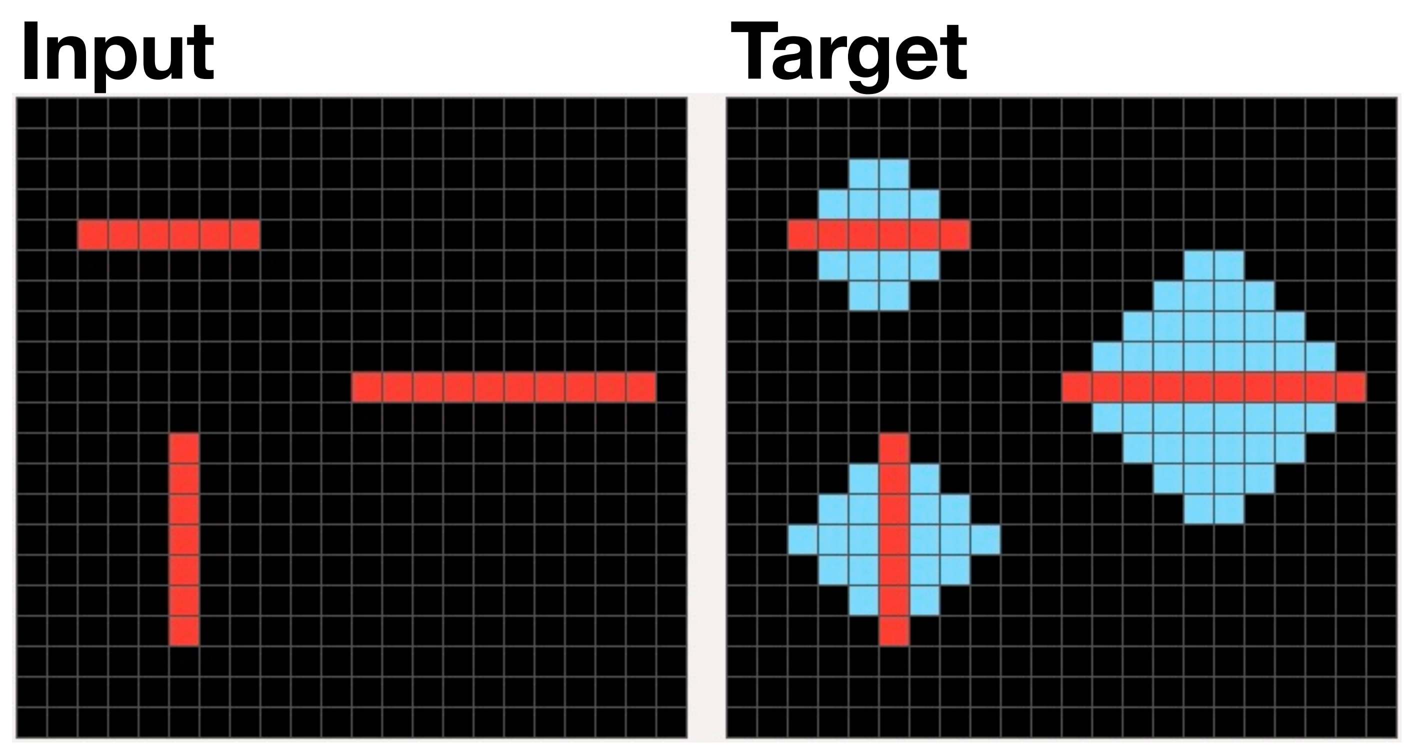


Step 5



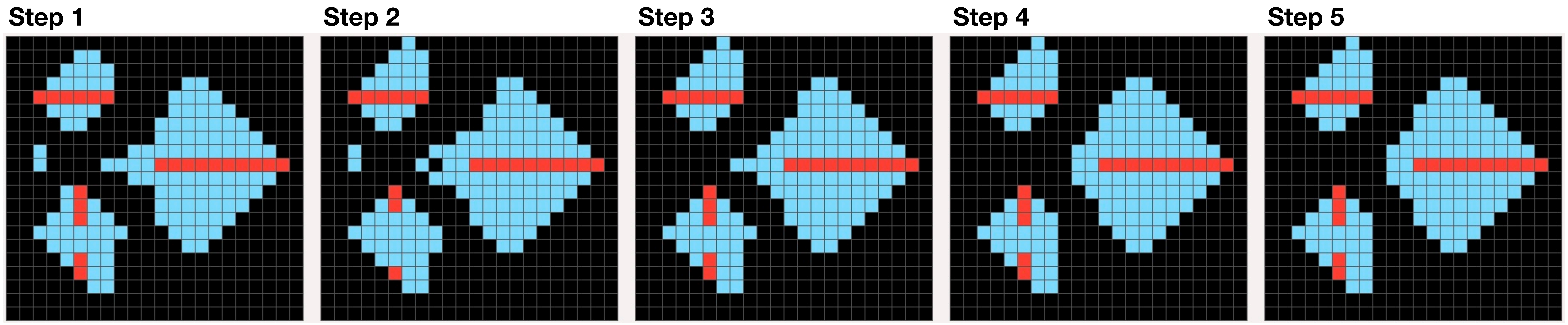
Inspecting what the training pairs have in common

c97c0139



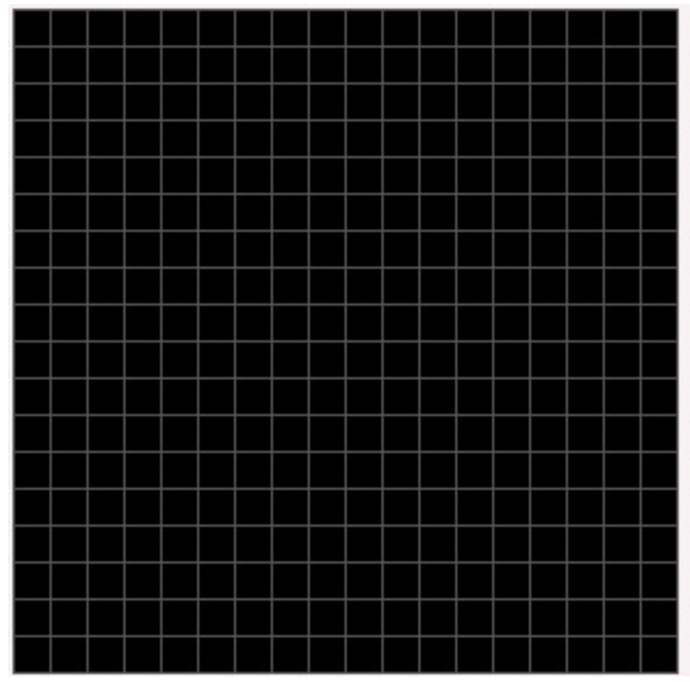
```
InputImageIsOutputImageWithNoChangesToPixelsWithColor { color: 0 }
InputImageIsOutputImageWithNoChangesToPixelsWithColor { color: 2 }
InputLeastPopularColorIsOutputLeastPopularColor
InputMostPopularColorIsOutputMostPopularColor
OutputImageIsInputImageWithChangesLimitedToPixelsWithColor { color: 0 }
OutputImageIsInputImageWithChangesLimitedToPixelsWithMostPopularColorOfTheInputImage
OutputImageIsInputImageWithNoChangesToPixelsWithColor { color: 2 }
OutputImagePreserveInputImageCorner { corner: BottomLeft }
OutputImagePreserveInputImageCorner { corner: BottomRight }
OutputImagePreserveInputImageCorner { corner: TopLeft }
OutputImagePreserveInputImageCorner { corner: TopRight }
OutputImagePreserveInputImageEdge { edge: Bottom }
OutputImagePreserveInputImageEdge { edge: Left }
OutputImagePreserveInputImageEdge { edge: Right }
OutputImagePreserveInputImageEdge { edge: Top }
OutputImageUniqueColorCount { count: 3 }
OutputPropertyIsEqualToInputProperty { output: OutputHeight, input: Height }
OutputPropertyIsEqualToInputProperty { output: OutputHeight, input: WidthMinus2 }
OutputPropertyIsEqualToInputProperty { output: OutputWidth, input: HeightPlus2 }
OutputPropertyIsEqualToInputProperty { output: OutputWidth, input: Width }
OutputPropertyIsInputPropertyMultipliedByInputSize { output: OutputHeight, input:
NumberOfClustersWithMostPopularIntersectionColor }
OutputPropertyIsInputPropertyMultipliedByInputSize { output: OutputHeight, input:
UniqueColorCountMinus1 }
OutputPropertyIsInputPropertyMultipliedByInputSize { output: OutputWidth, input:
NumberOfClustersWithMostPopularIntersectionColor }
OutputPropertyIsInputPropertyMultipliedByInputSize { output: OutputWidth, input:
UniqueColorCountMinus1 }
```

It has difficulties with diagonal structures.

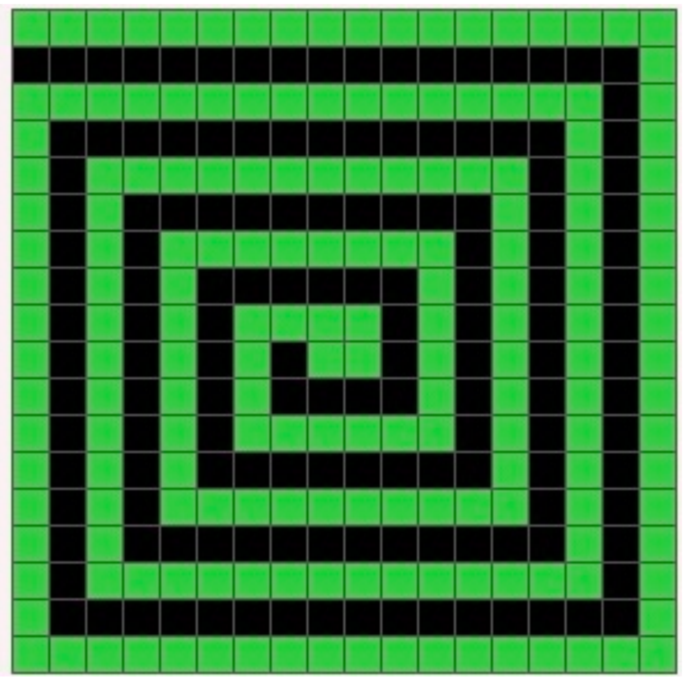


28e73c20

Input



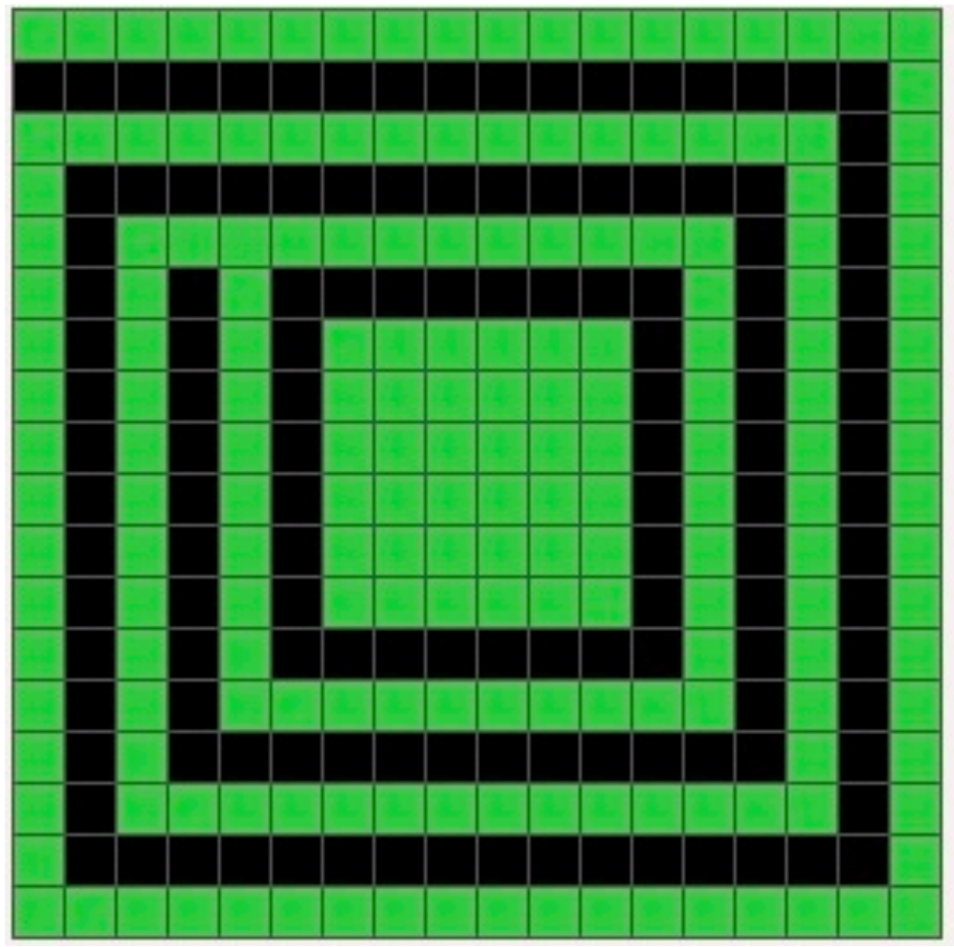
Target



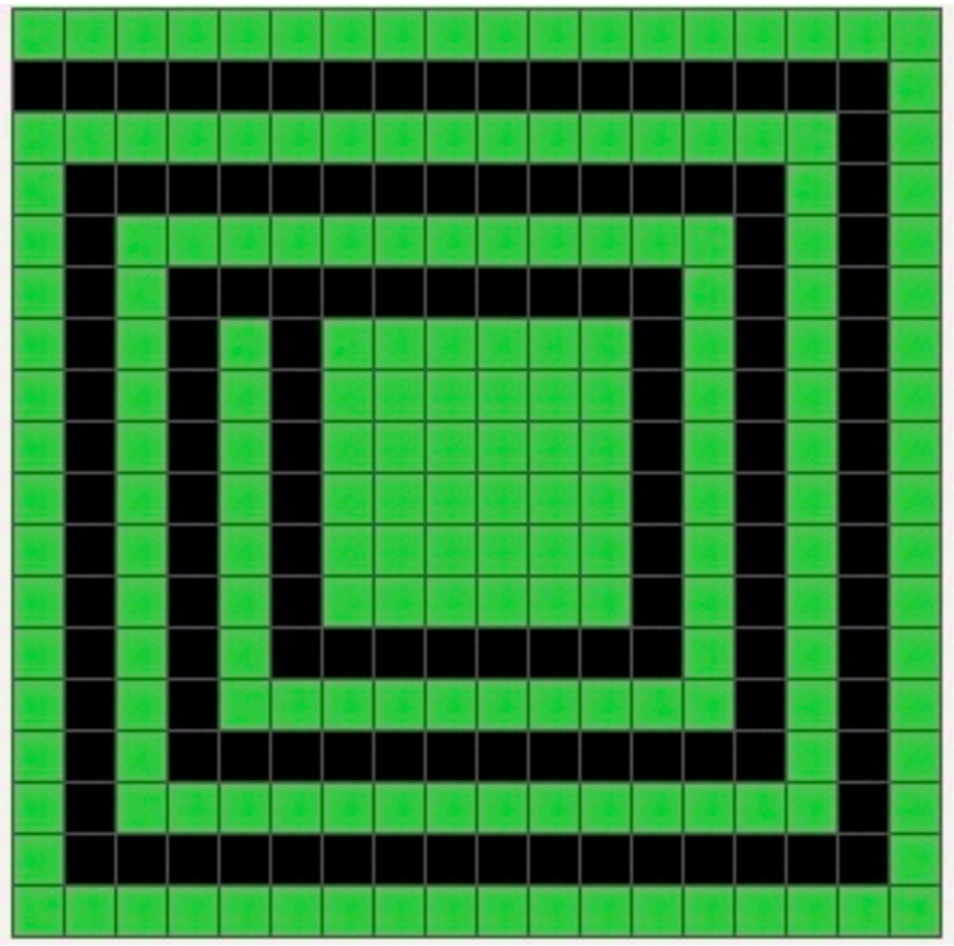
Not yet able to solve the spiral.

Maybe more steps can help.

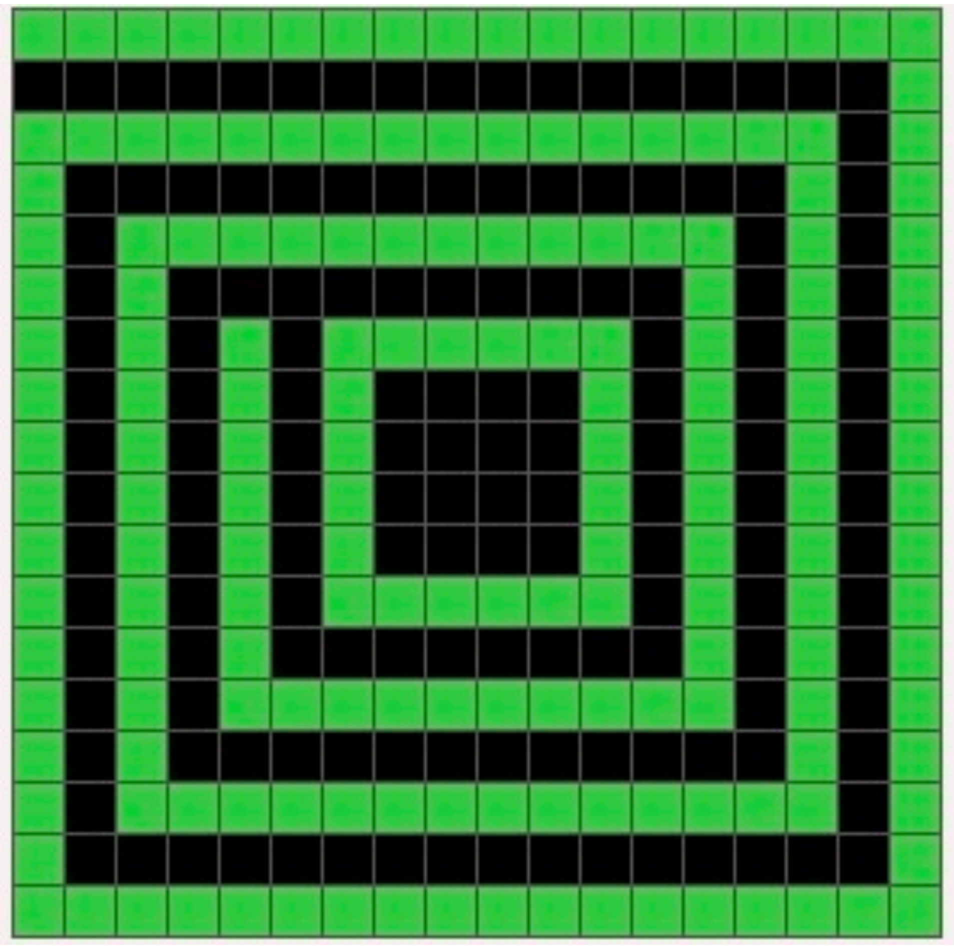
Step 1



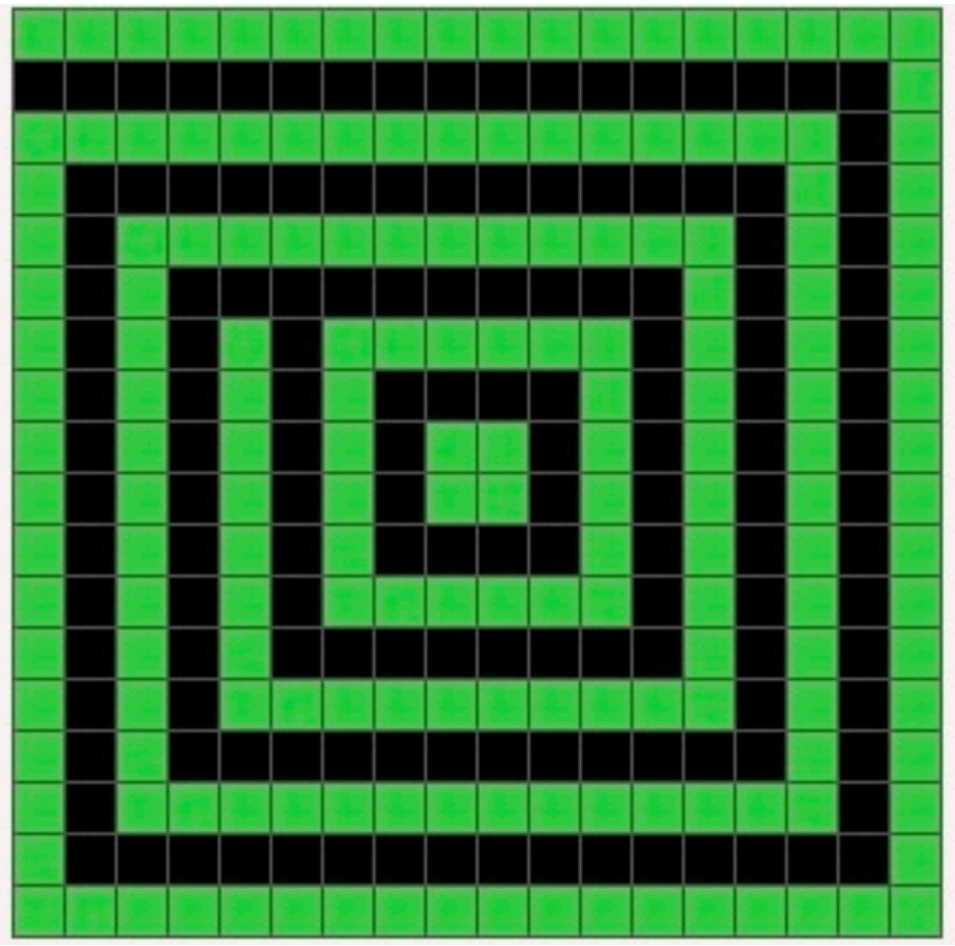
Step 2



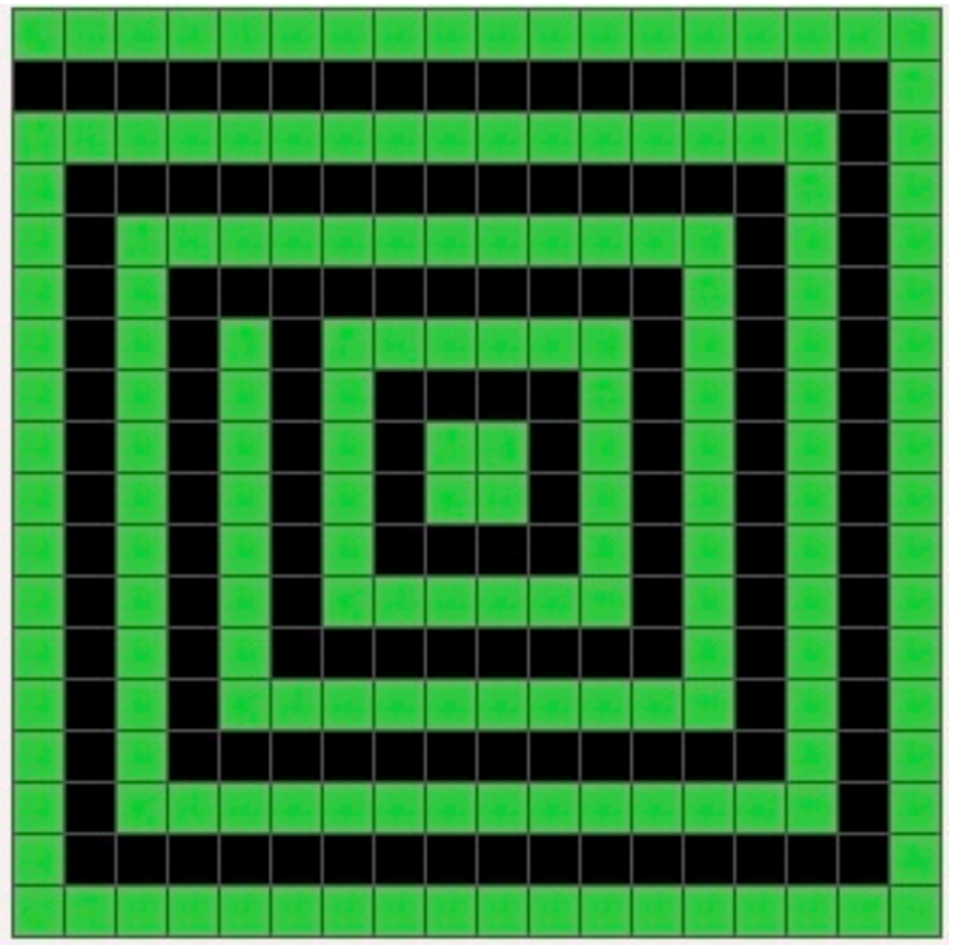
Step 3



Step 4



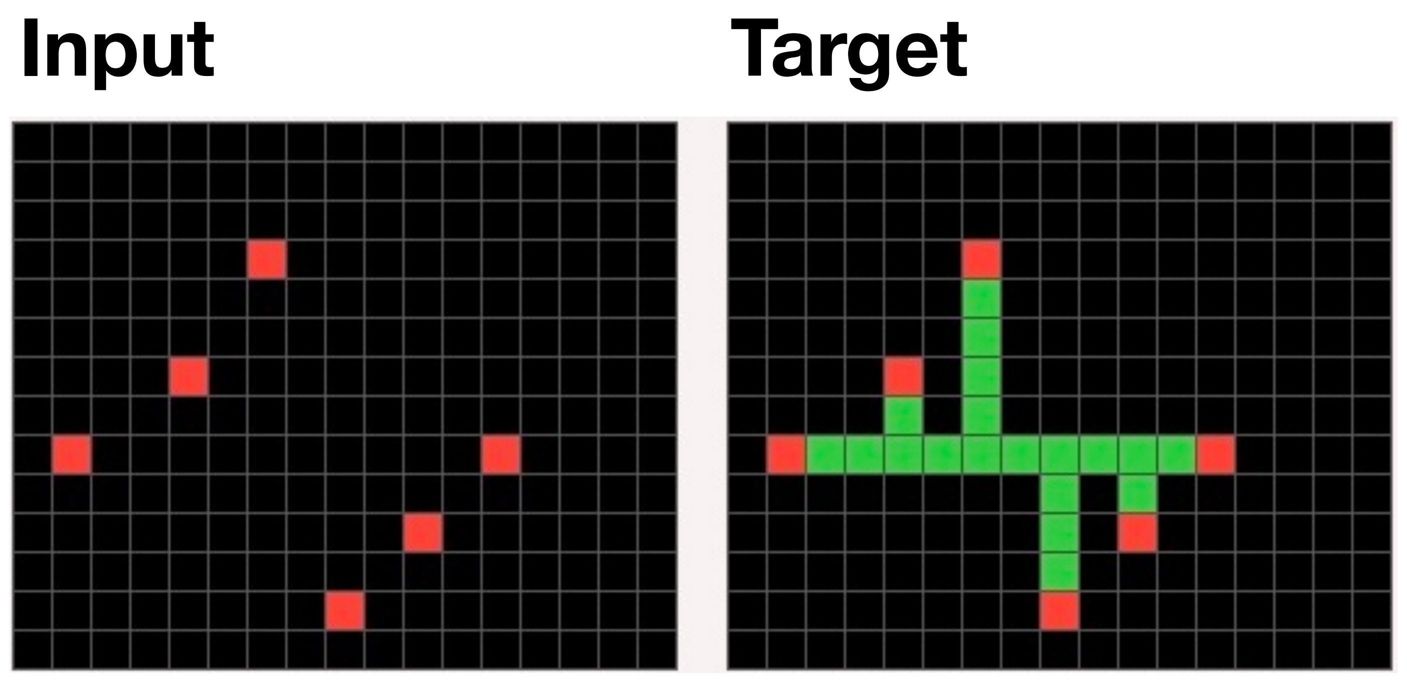
Step 5

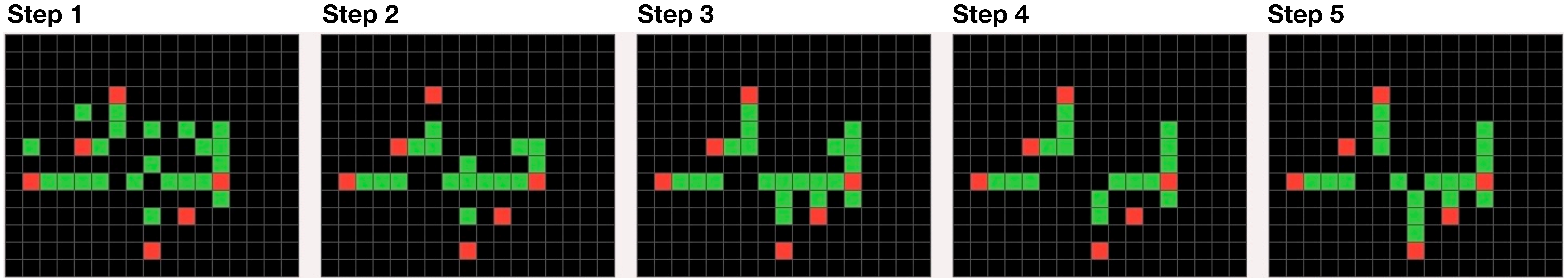


Weakness: No memory

Since it's logistic regression, it has no memory of what objects it has seen in the past. So it cannot recognize familiar objects.

bf89d739





Weakness: Colors are often wrong

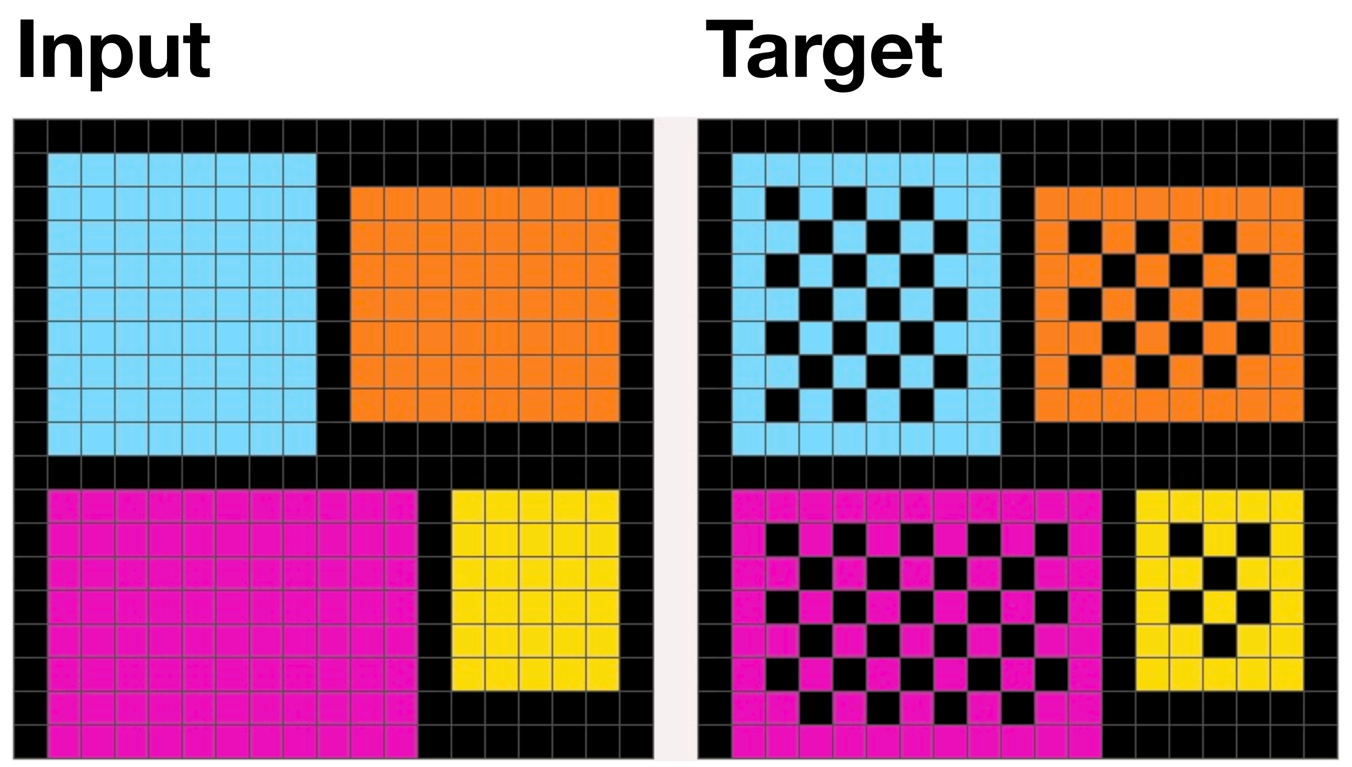
I have tried one-hot encoded colors, with mixed results.

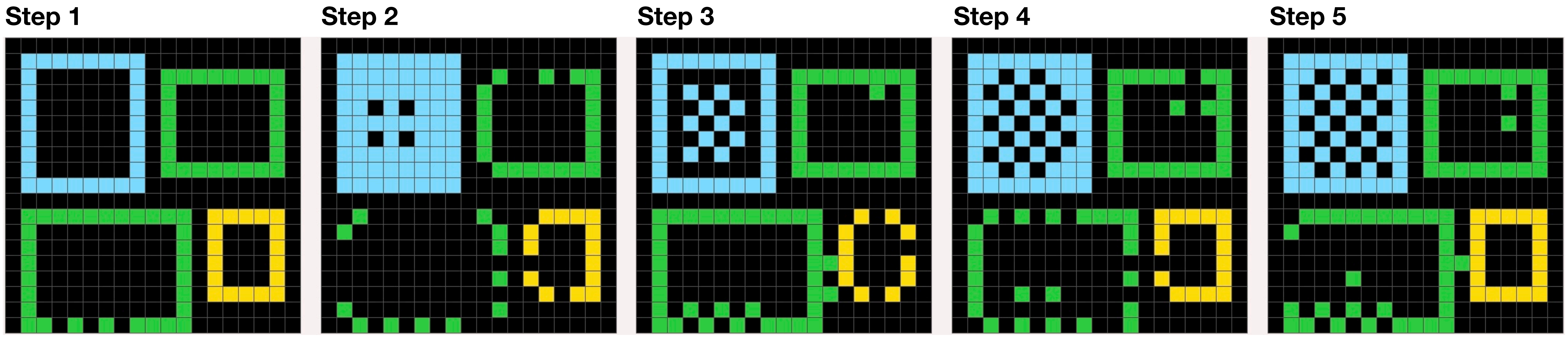
The colors are encoded as two 64bit floats. It's x, y coordinates to the unit circle

radians = color * TAU / 11 + color_obfuscation

The color_obfuscation offsets gets changed for each step.

ba9d41b8


Input
Target


Step 1
Step 2
Step 3
Step 4
Step 5