Introduction to Mathematical and Computational Thinking: A New Gen-Ed Math Course

Betty Love, Victor Winter, Michael Matthews, Michelle Friend
University of Nebraska - Omaha

Background

- University of Nebraska - Omaha
- One gen-ed math course: College Algebra
- 1600 students/year enrolled
- 2% (32 students) went on to take calculus

New math gen-ed course

- Introduction to Mathematical and Computational Thinking
- Goals:
  - Engage students
  - Change attitudes about math and STEM
  - Provide many opportunities for students to be creative
  - Teach mathematical ***thinking***, not calculating
  - Use IBL to promote deeper learning
- Offered for the first time in Spring 2018

- *Mathematical thinking* is more than being able to do arithmetic or solve algebra problems… Mathematical thinking is a whole way of looking at things, of *stripping them down* to their numerical, structural, or logical *essentials*, and of *analyzing the underlying patterns*. — Keith Devlin, *Mathematics: The Science of Patterns*

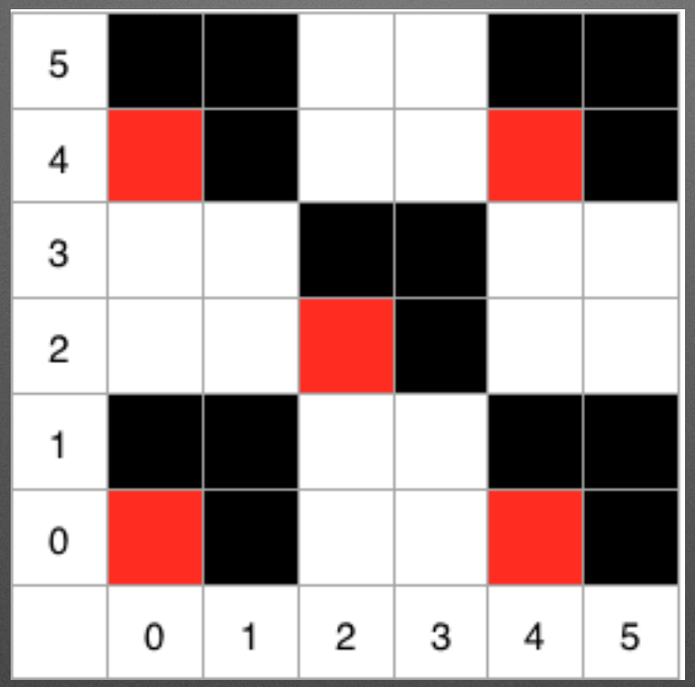
- Computational thinking involves
  - decomposition - *breaking down* a complex problem or system into smaller, more manageable parts
  - *pattern recognition* – looking for similarities among and within problems
  - abstraction – focusing on the *important information only*, ignoring irrelevant detail
  - algorithms - developing a step-by-step solution to the problem, or the rules to follow to solve the problem

Patterns

- “Mathematicians of all kinds now see their work as the study of patterns - real or imagined, visual or mental, arising from the natural world or from within the human mind.” — Keith Devlin, *Mathematics: The Science of Patterns*

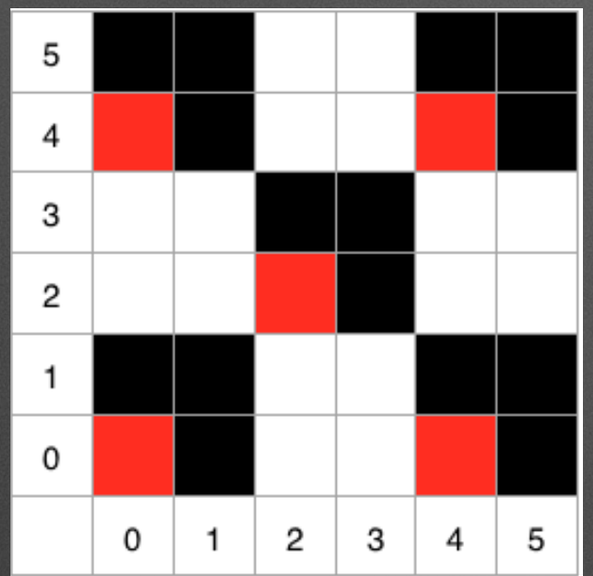
- To leverage the power of the computer, one must be able to create a small program that, when executed, produces a large number of computational steps.

- And in order for this to occur, there must exist a pattern in the computational sequence that can be described by the program.

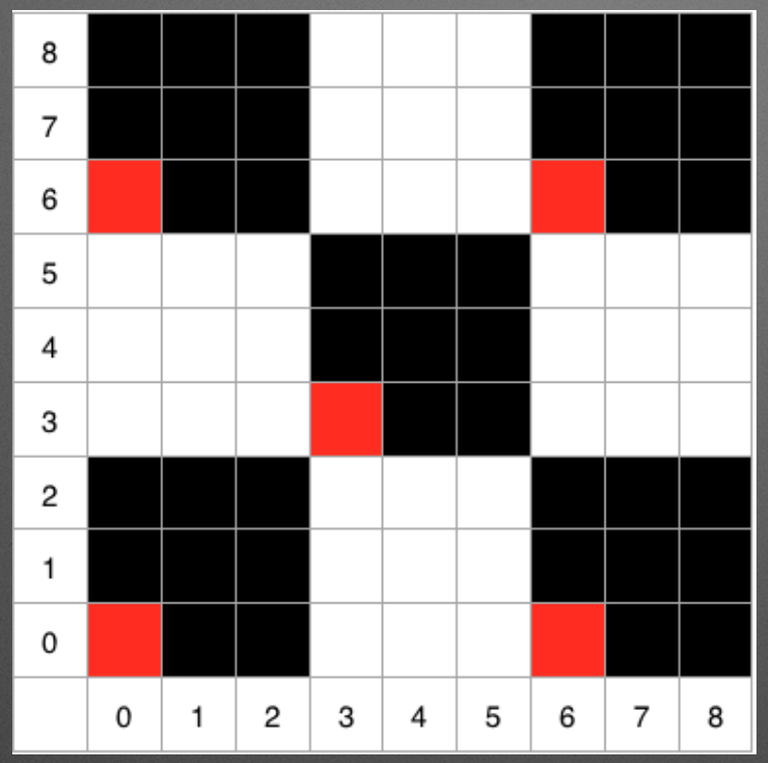

5
4
3
2
1
0
0 1 2 3 4 5

5						
4						
3						
2						
1						
0						
	0	1	2	3	4	5

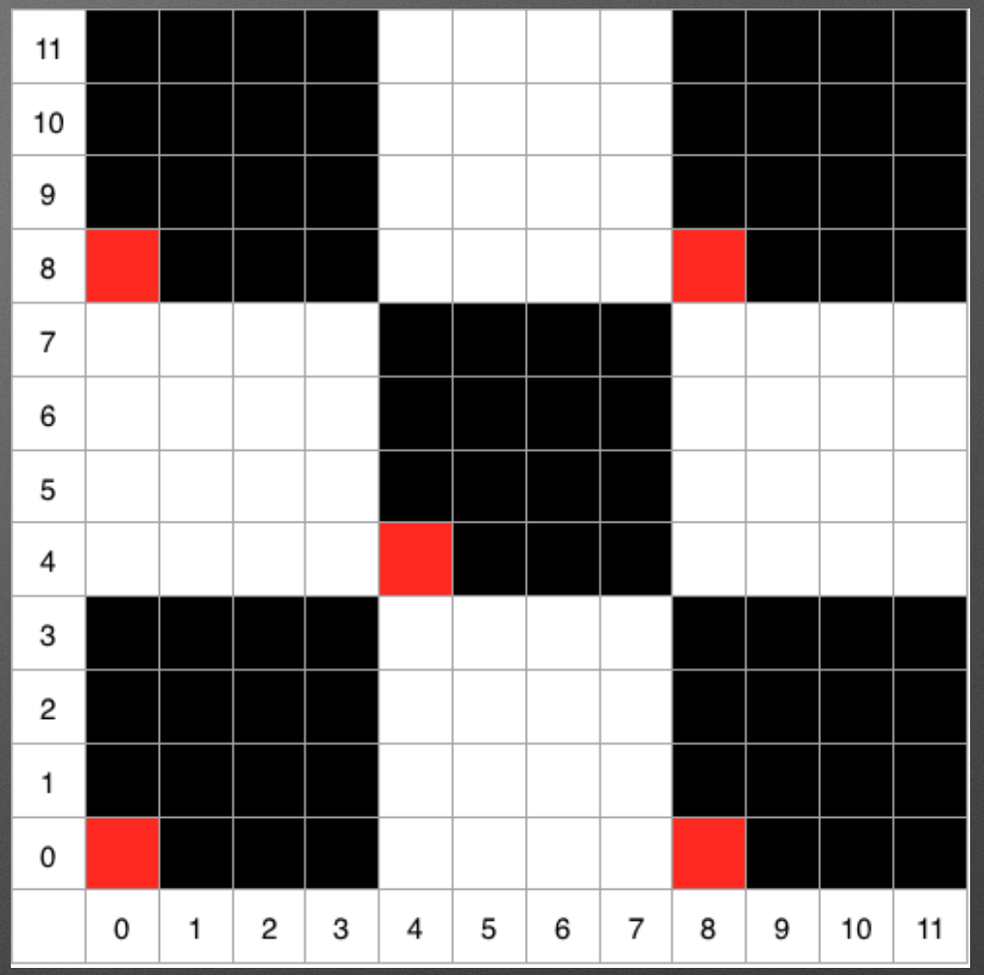
8									
7									
6									
5									
4									
3									
2									
1									
0									
	0	1	2	3	4	5	6	7	8



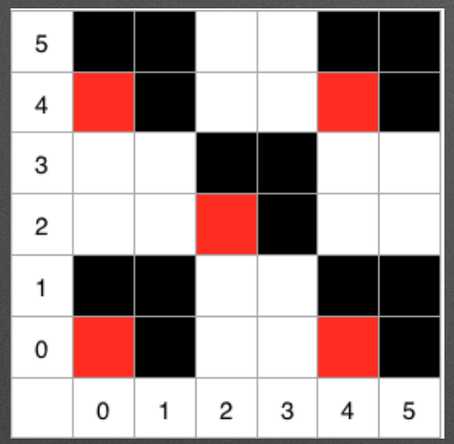
5						
4						
3						
2						
1						
0						
	0	1	2	3	4	5



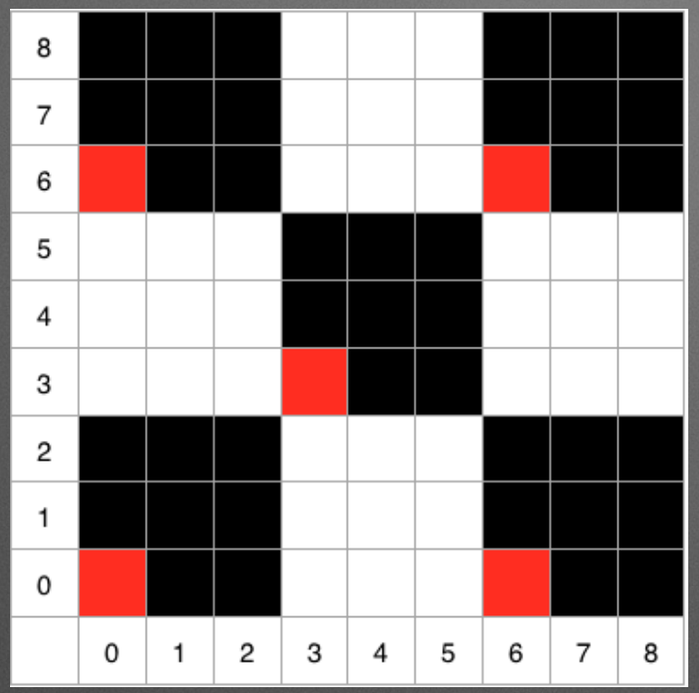
8									
7									
6									
5									
4									
3									
2									
1									
0									
	0	1	2	3	4	5	6	7	8



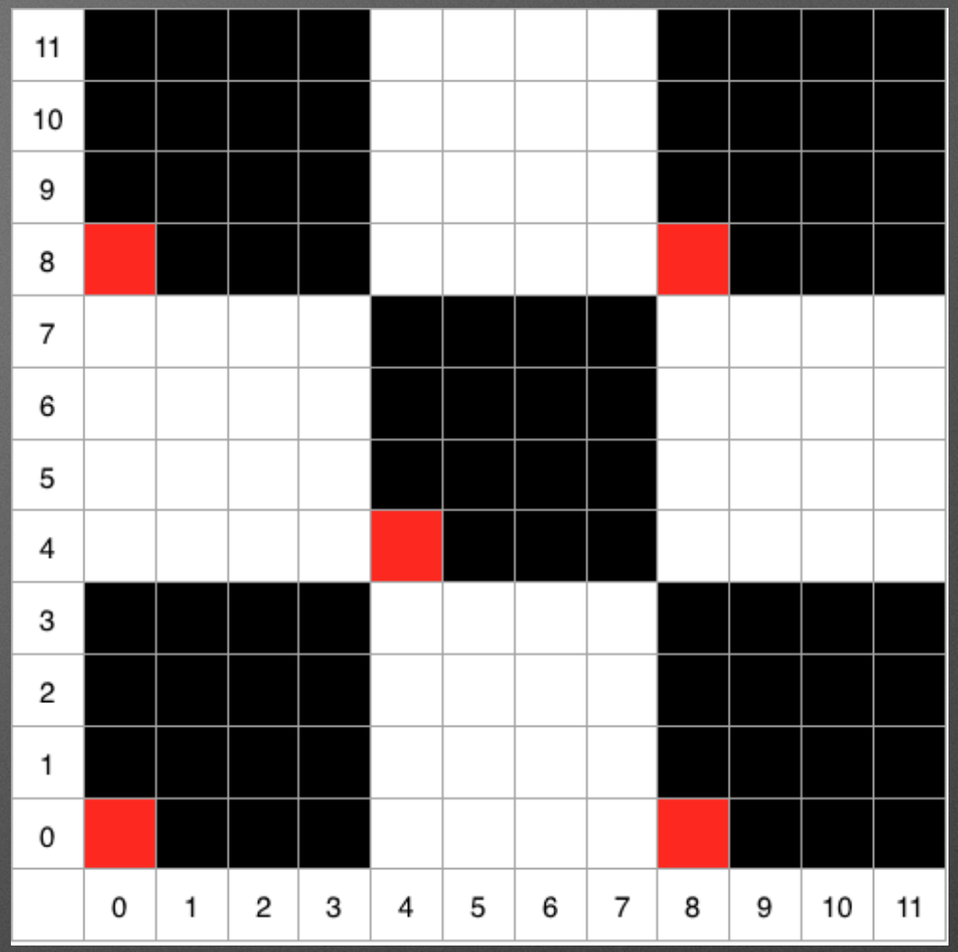
11												
10												
9												
8												
7												
6												
5												
4												
3												
2												
1												
0												
	0	1	2	3	4	5	6	7	8	9	10	11



2 x 2



3 x 3

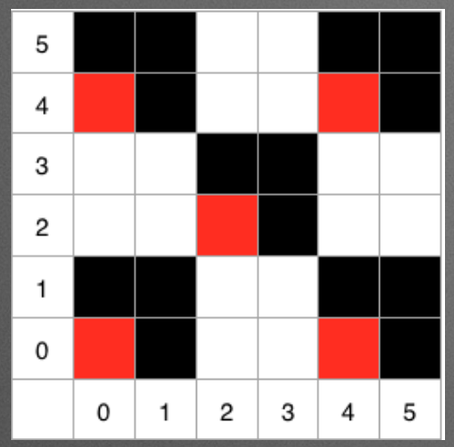


4 x 4

What are the coordinates of the red cells in the n x n case?

Let’s focus on x-coordinates.

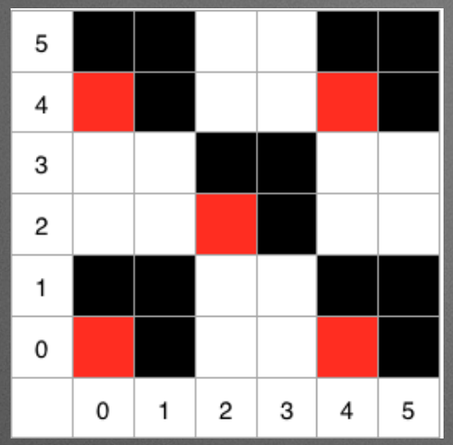
2 x 2



Square	2x2
LL	0
LR	4
M	2
UL	0
UR	4

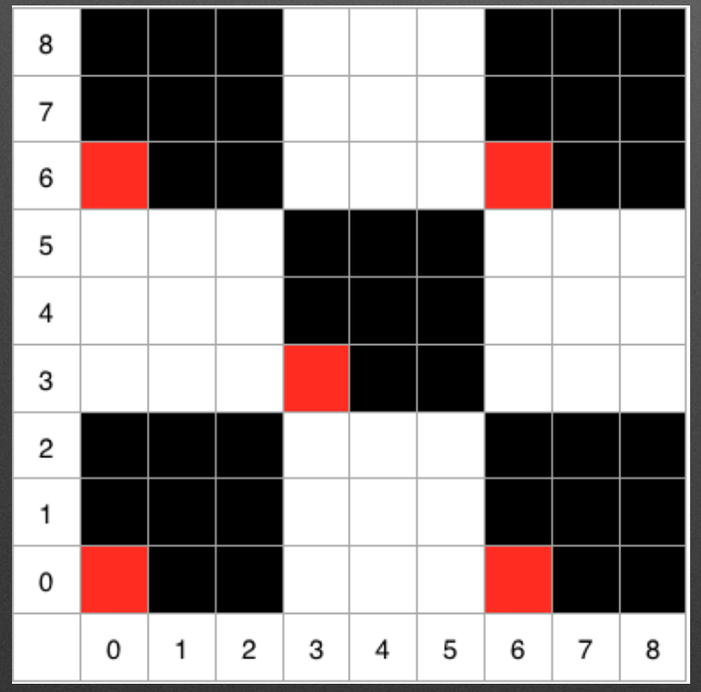
Let’s focus on x-coordinates.

2 x 2



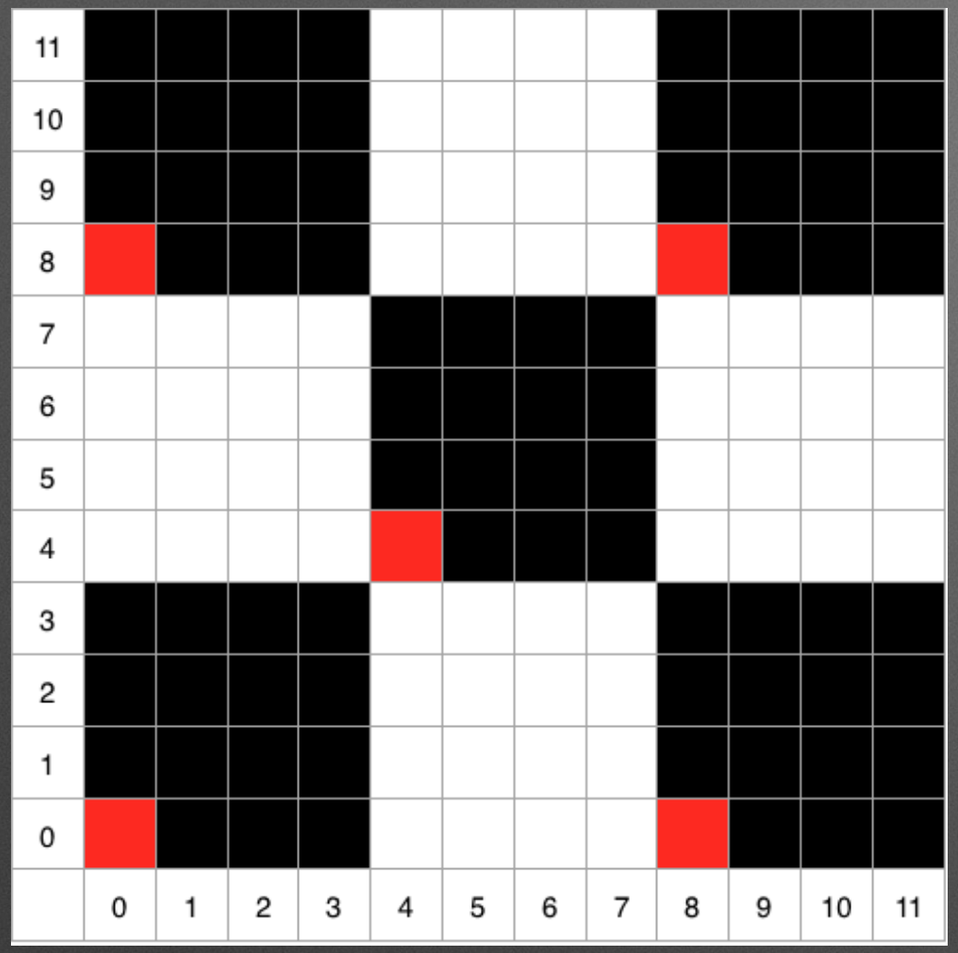
Square	2x2
LL	0
LR	4
M	2
UL	0
UR	4

3 x 3



Square	3x3
LL	0
LR	6
M	3
UL	0
UR	6

Let’s focus on x-coordinates.



Square	2x2	3x3	4x4	5x5
LL	0	0	0	0
LR	4	6	8	10
M	2	3	4	5
UL	0	0	0	0
UR	4	6	8	10

See any patterns?

Let’s focus on x-coordinates.

Square	2x2	3x3	4x4	5x5
LL	0	0	0	0
LR	4	6	8	10
M	2	3	4	5
UL	0	0	0	0
UR	4	6	8	10

Square	2x2	3x3	4x4	5x5
LL	0	0	0	0
LR	2 x 2	2 x 3	2 x 4	2 x 5
M	1 x 2	1 x 3	1 x 4	1 x 5
UL	0	0	0	0
UR	2 x 2	2 x 3	2 x 4	2 x 5

Let’s focus on x-coordinates.

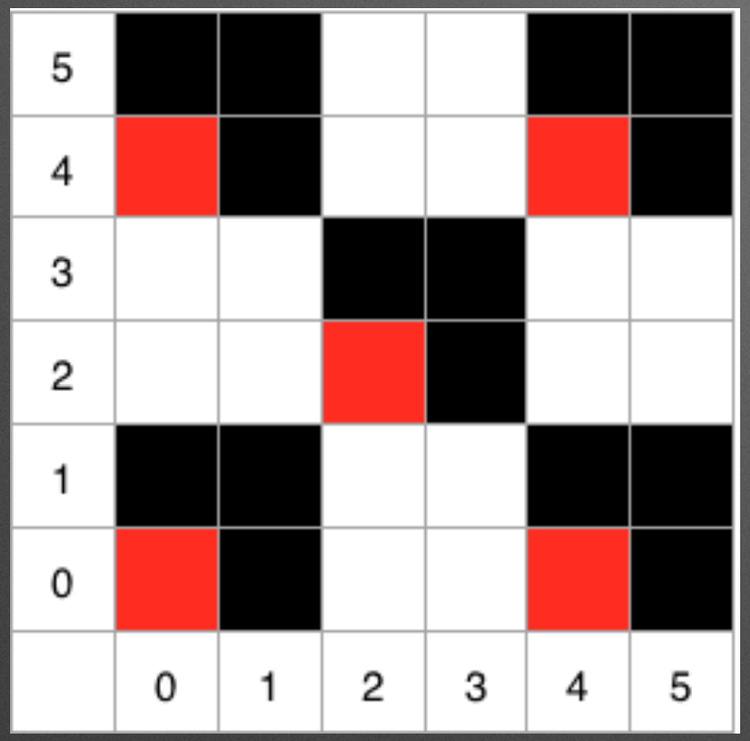
Square	2x2	3x3	4x4	5x5
LL	0	0	0	0
LR	4	6	8	10
M	2	3	4	5
UL	0	0	0	0
UR	4	6	8	10

Square	2x2	3x3	4x4	5x5	n x n
LL	0	0	0	0	0
LR	2 x 2	2 x 3	2 x 4	2 x 5	2 x n
M	1 x 2	1 x 3	1 x 4	1 x 5	1 x n
UL	0	0	0	0	0
UR	2 x 2	2 x 3	2 x 4	2 x 5	2 x n

Coordinates of red squares for n x n case:

- LL: (0, 0)
- LR: (2*n, 0)
- M: (1*n, 1*n)
- UL: (0, 2*n)
- UR: (2*n, 2*n)

How do I know if I’m right???

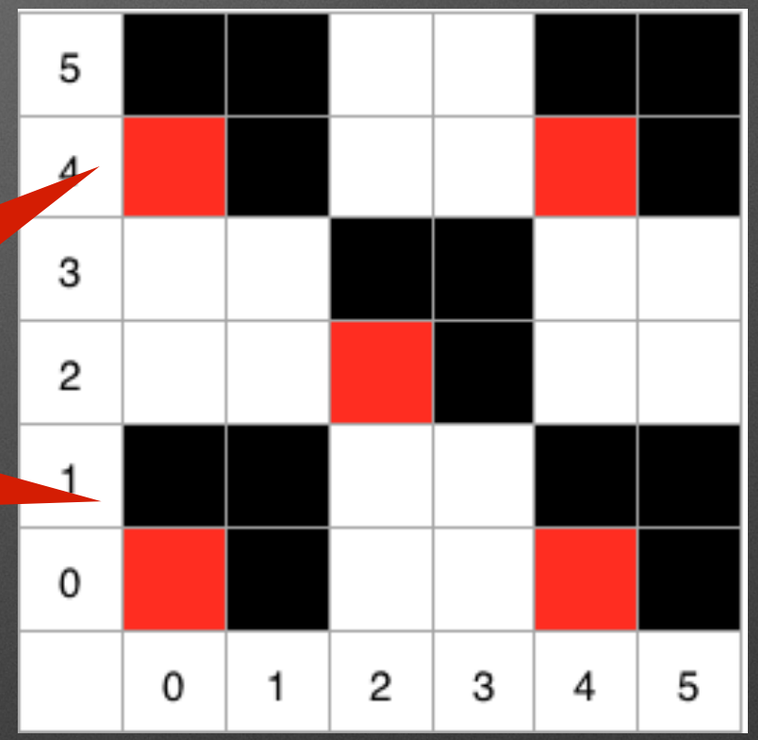



```
open Level_3;

fun oneDottedBlackSquare (x,y) =
    (
        put2D (2,2) BLACK (x,y);
        put2D (1,1) RED   (x,y)
    );


build2D (8,8);

oneDottedBlackSquare (0,0);
oneDottedBlackSquare (4,0);
oneDottedBlackSquare (2,2);
oneDottedBlackSquare (0,4);
oneDottedBlackSquare (4,4);

show2D "my artifact";
```

Creates one 2 x 2 square at (x,y)

Call it five times to create five squares.

```
1   open Level_3;
2
3   fun oneDottedBlackSquare (x,y) =
4       (
5           put2D (2,2) BLACK (x,y);
6           put2D (1,1) RED   (x,y)
7       );
8
9
10  build2D (8,8);
11
12  oneDottedBlackSquare (0,0);
13  oneDottedBlackSquare (4,0);
14  oneDottedBlackSquare (2,2);
15  oneDottedBlackSquare (0,4);
16  oneDottedBlackSquare (4,4);
17
18  show2D "my artifact";
```



Creates one n x n dotted square at (x,y)

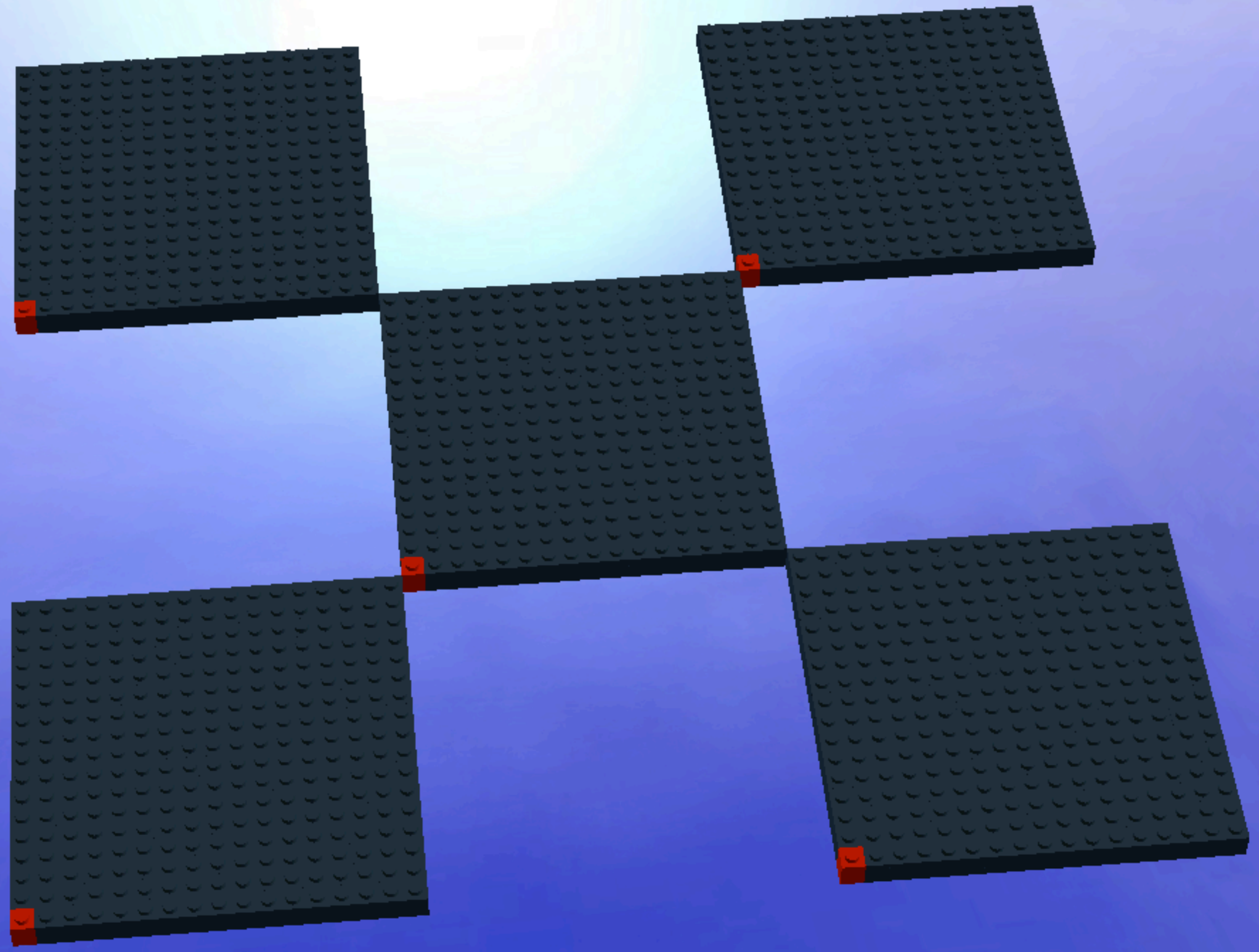
Call it five times to create five squares.

```
1   open Level_3;
2
3   fun oneDottedBlackSquare (n,x,y) =
4       (
5           putZD (n,n) BLACK (x,y);
6           putZD (1,1) RED   (x,y)
7       );
8
9   fun fiveDottedBlackSquares (n) =
10      (
11          oneDottedBlackSquare (n,  0  ,    0);
12          oneDottedBlackSquare (n,  2*n,    0);
13          oneDottedBlackSquare (n,  1*n,  1*n);
14          oneDottedBlackSquare (n,  0  ,  2*n);
15          oneDottedBlackSquare (n,  2*n,  2*n)
16      );
17
18  build2D (100,100);
19
20  fiveDottedBlackSquares (17);
21
22  show2D "my artifact";
```

Create artifact corresponding to n = 17

Square size

Coordinates - where to put squares



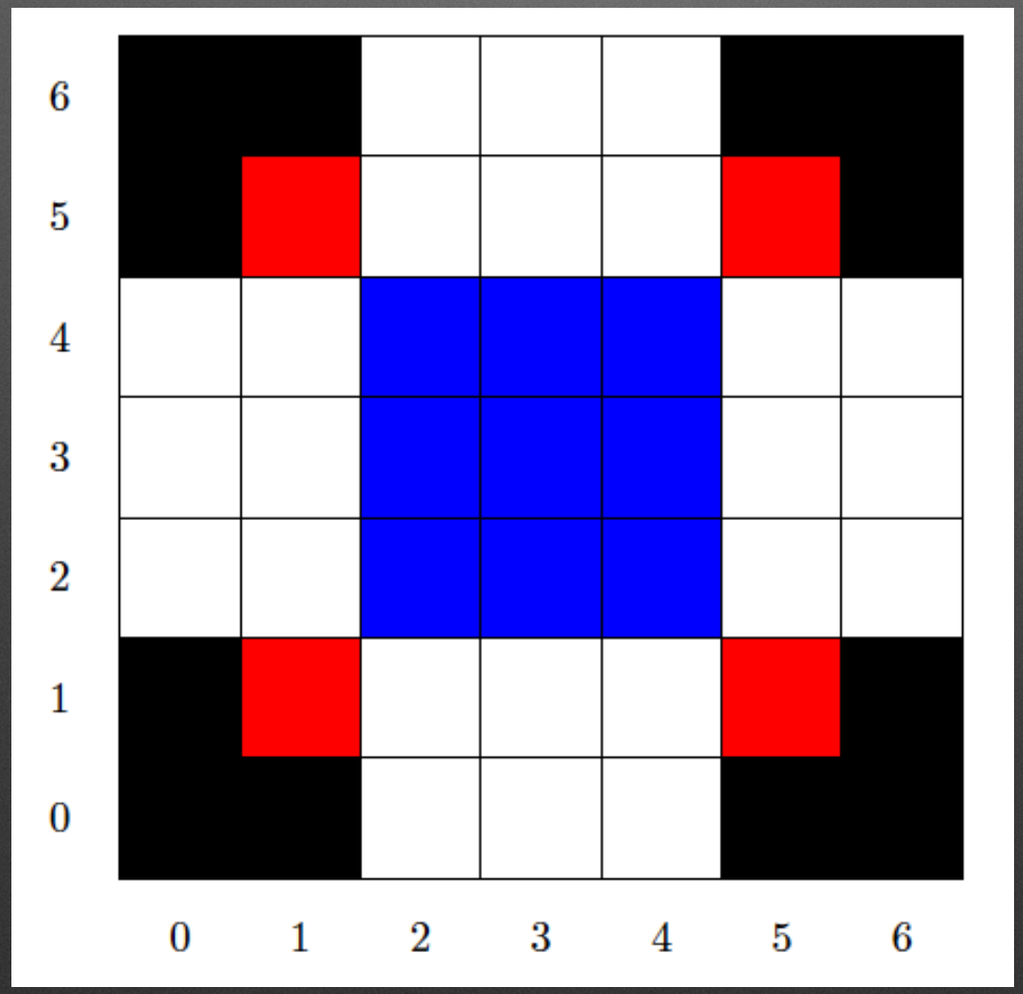
Program output viewed in LEGO Digital Designer

Endless options for making problems of varying degrees of difficulty

- Offset artifact from the origin

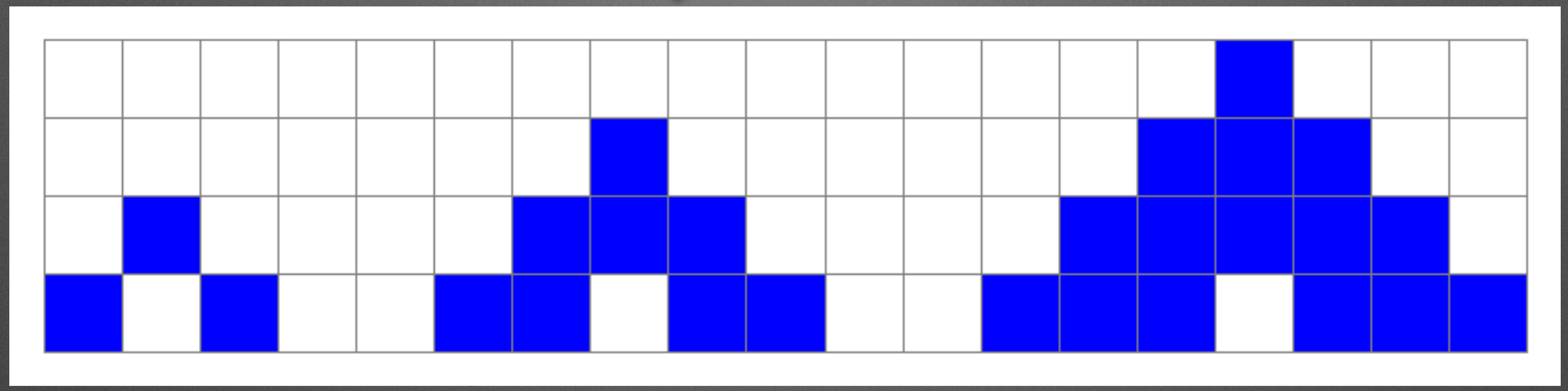
Make harder (or easier) problems

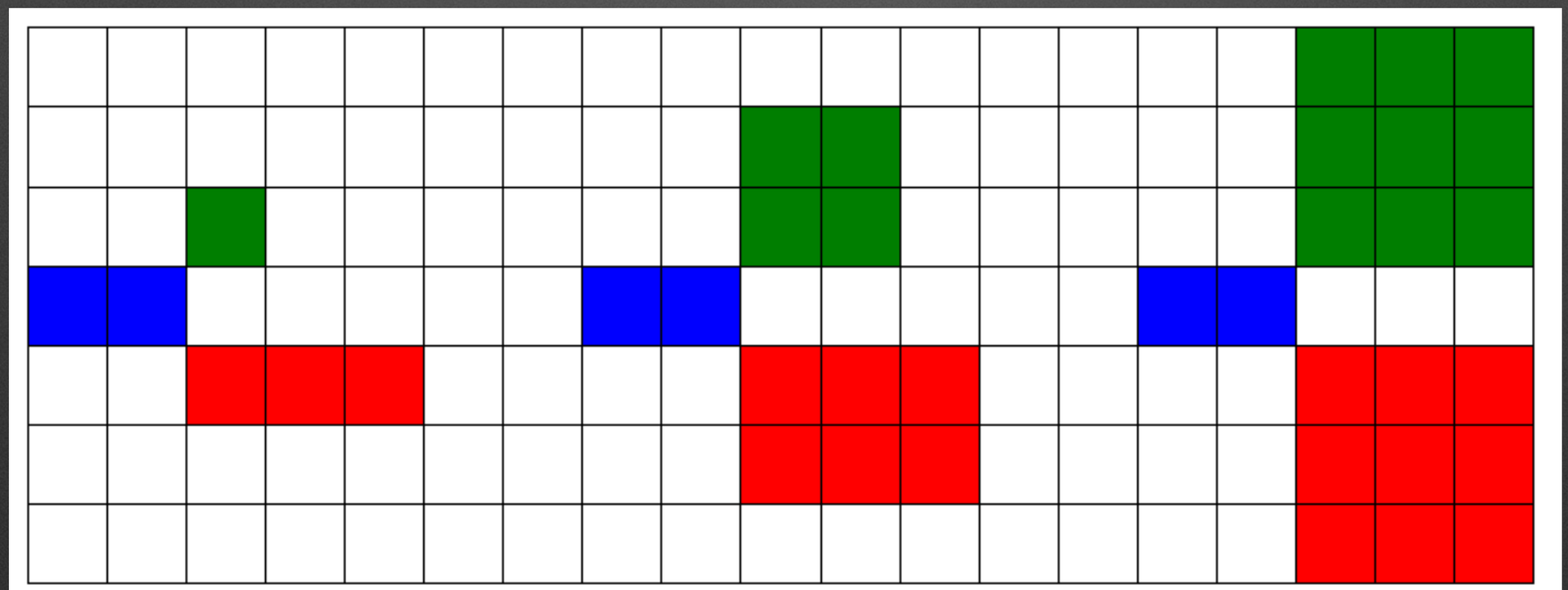
- Offset artifact from the origin
- Introduce more variables and more complexity



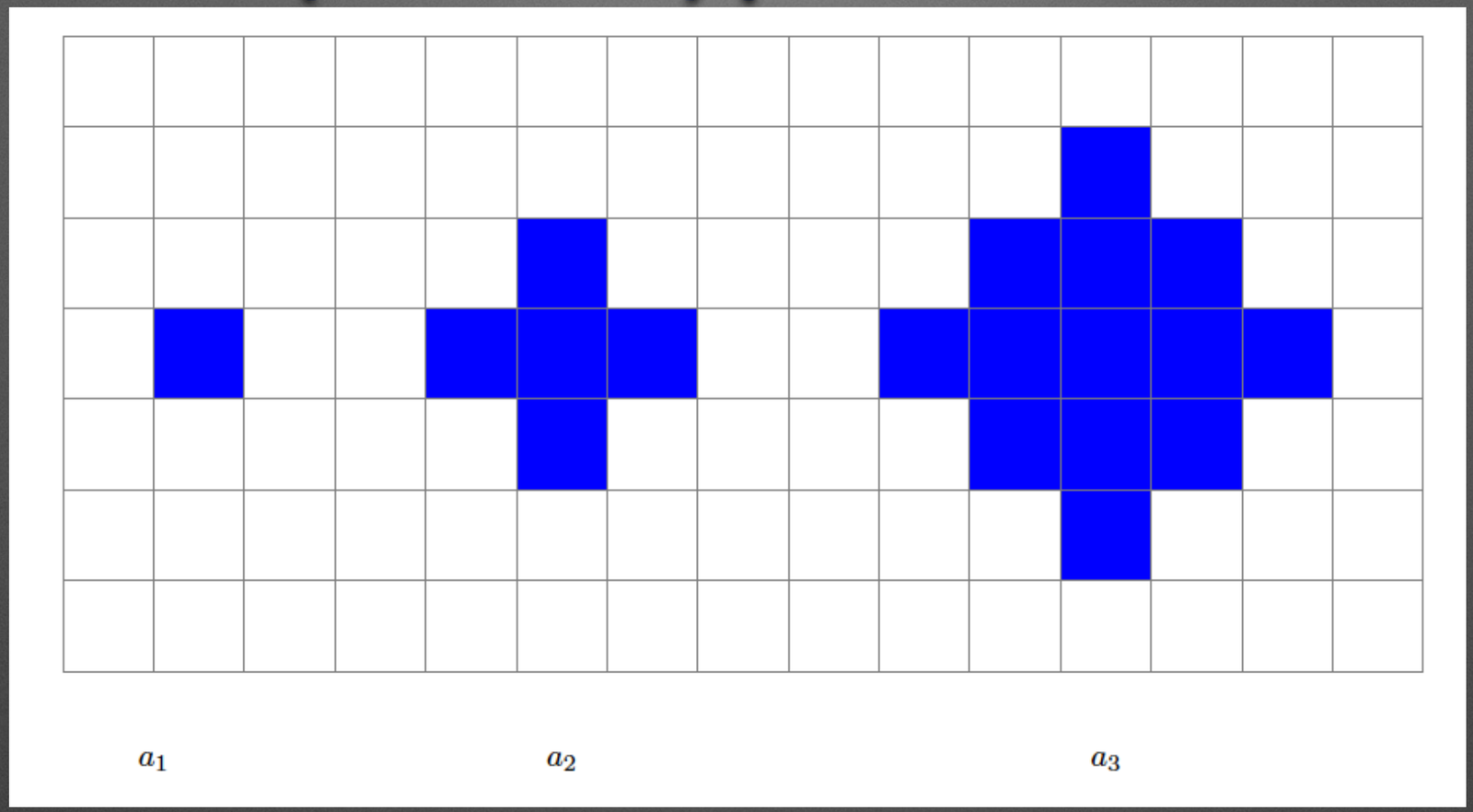
Make harder (or easier) problems

- Grow artifacts in multiple directions



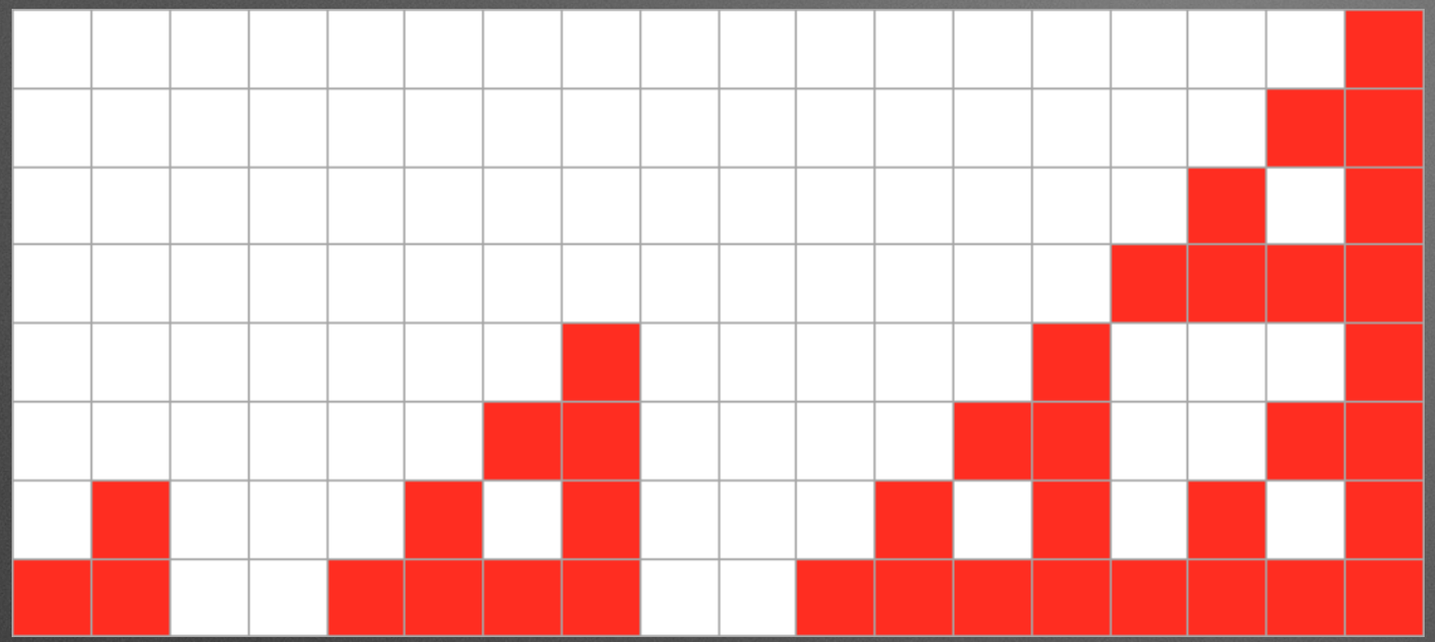


Make harder (or easier) problems

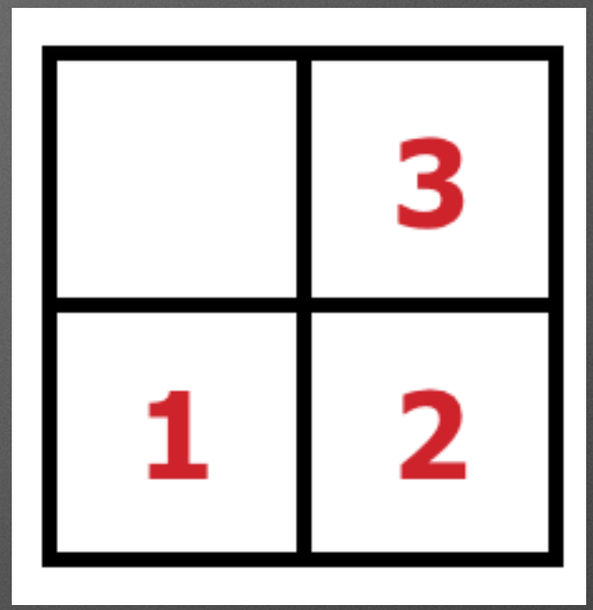


- How many blocks in a_{20}?
- Find *i* such that a_i has 113 blocks.
- Develop an expression for computing the number of blocks in the n^{th} element of this pattern.
- Write a Bricklayer program to generate the n^{th} artifact centered at location (x,y).

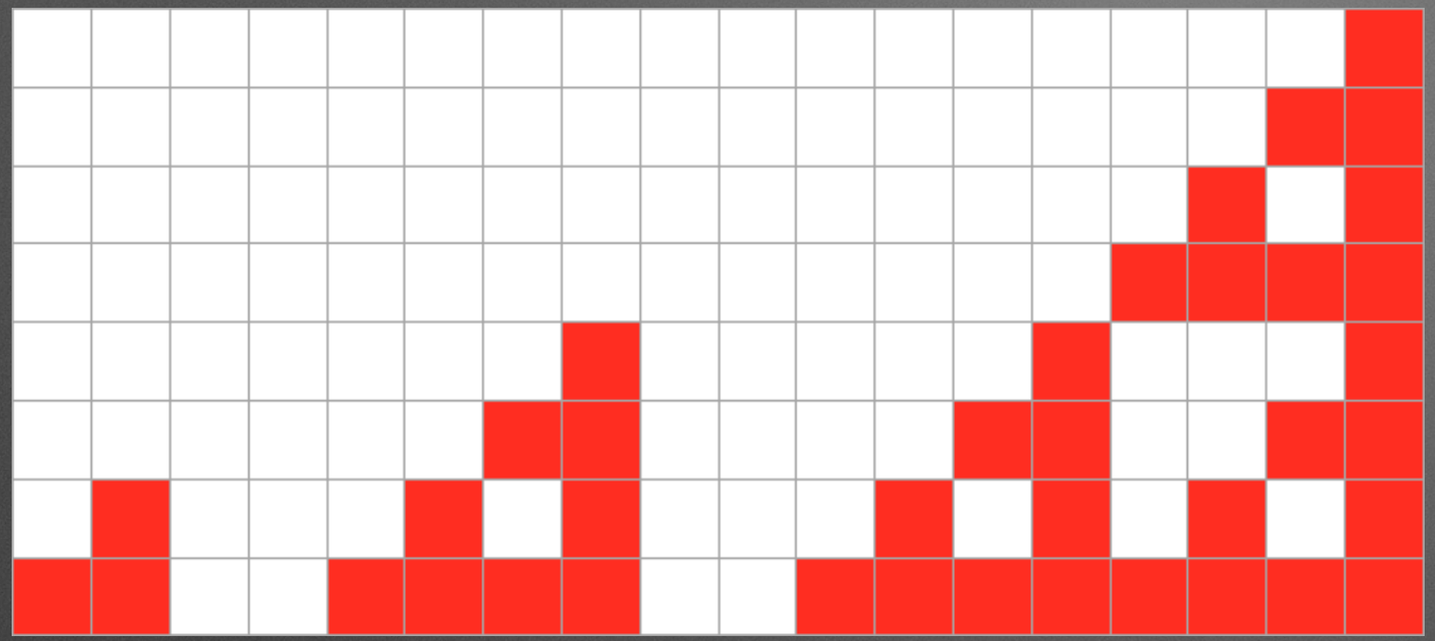
Geometric progressions

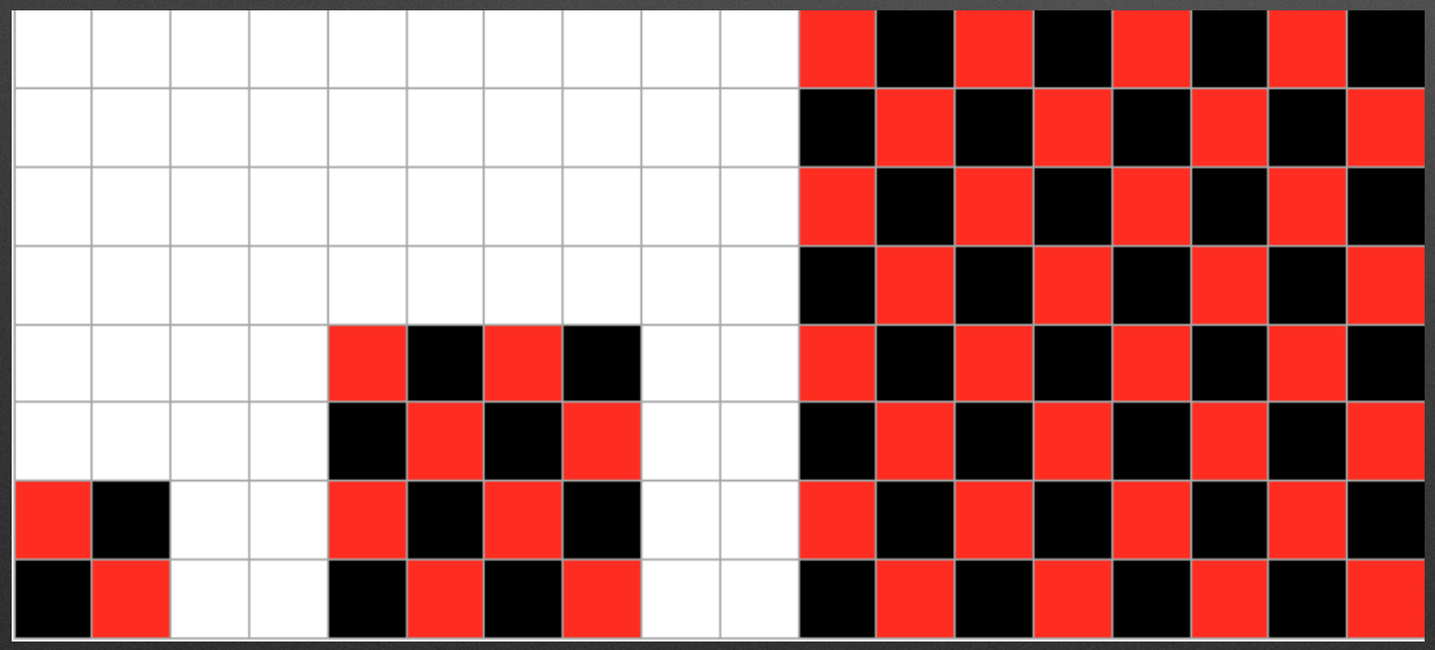


Stamping Pattern

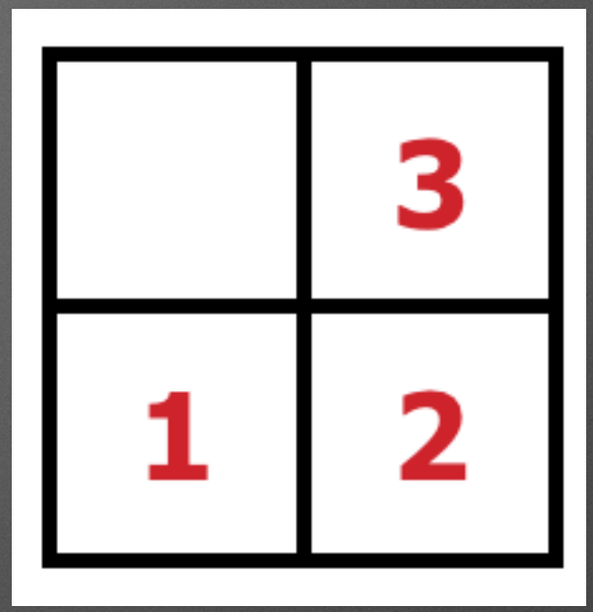


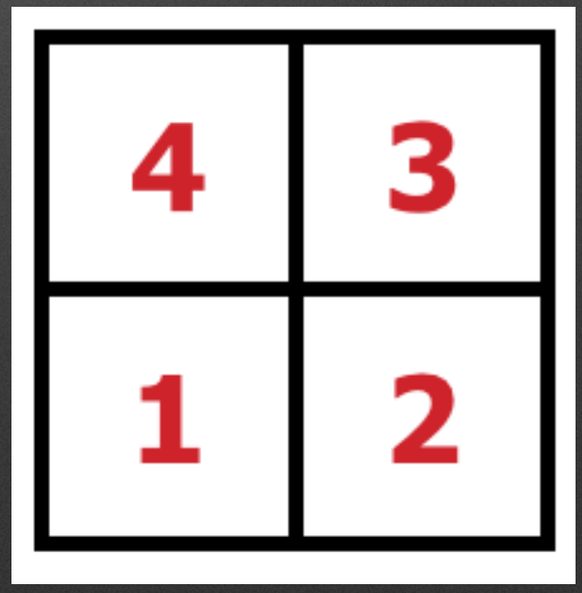
Geometric progressions

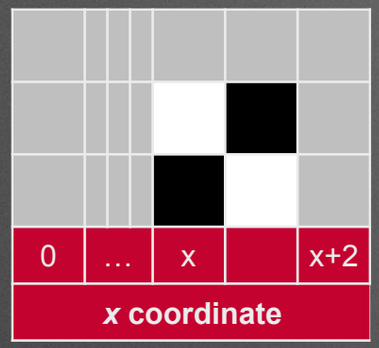




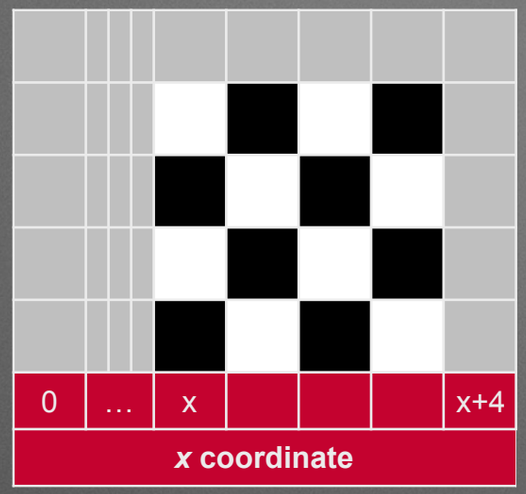
Stamping Pattern



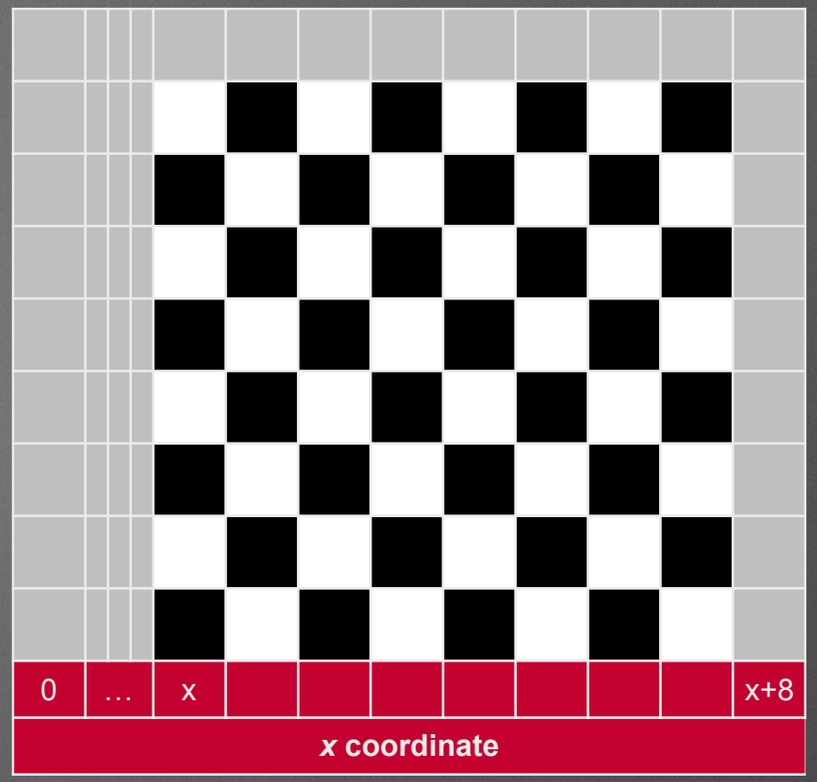




Step 0



Step 1



Step 2

Artifact positions along the *x*-axis

Initial Tile Size →	2x2	3x3	4x4	5x5		2x2	3x2	4x2	5x2		2x2	3x3	4x4	5x5
Step 0	x + 2	x + 3	x + 4	x + 5		x + 2∗1	x + 3∗1	x + 4∗1	x + 5∗1		x + 2∗2^0	x + 3∗2^0	x + 4∗2^0	x + 5∗2^0
Step 1	x + 4	x + 6	x + 8	x + 10		x + 2∗2	x + 3∗2	x + 4∗2	x + 5∗2		x + 2∗2^1	x + 3∗2^1	x + 4∗2^1	x +5∗2^1
Step 2	x + 8	x + 12	x + 16	x + 20		x + 2∗4	x + 3∗4	x + 4∗4	x + 5∗4		x + 2∗2^2	x + 3∗2^2	x + 4∗2^2	x + 5∗2^2

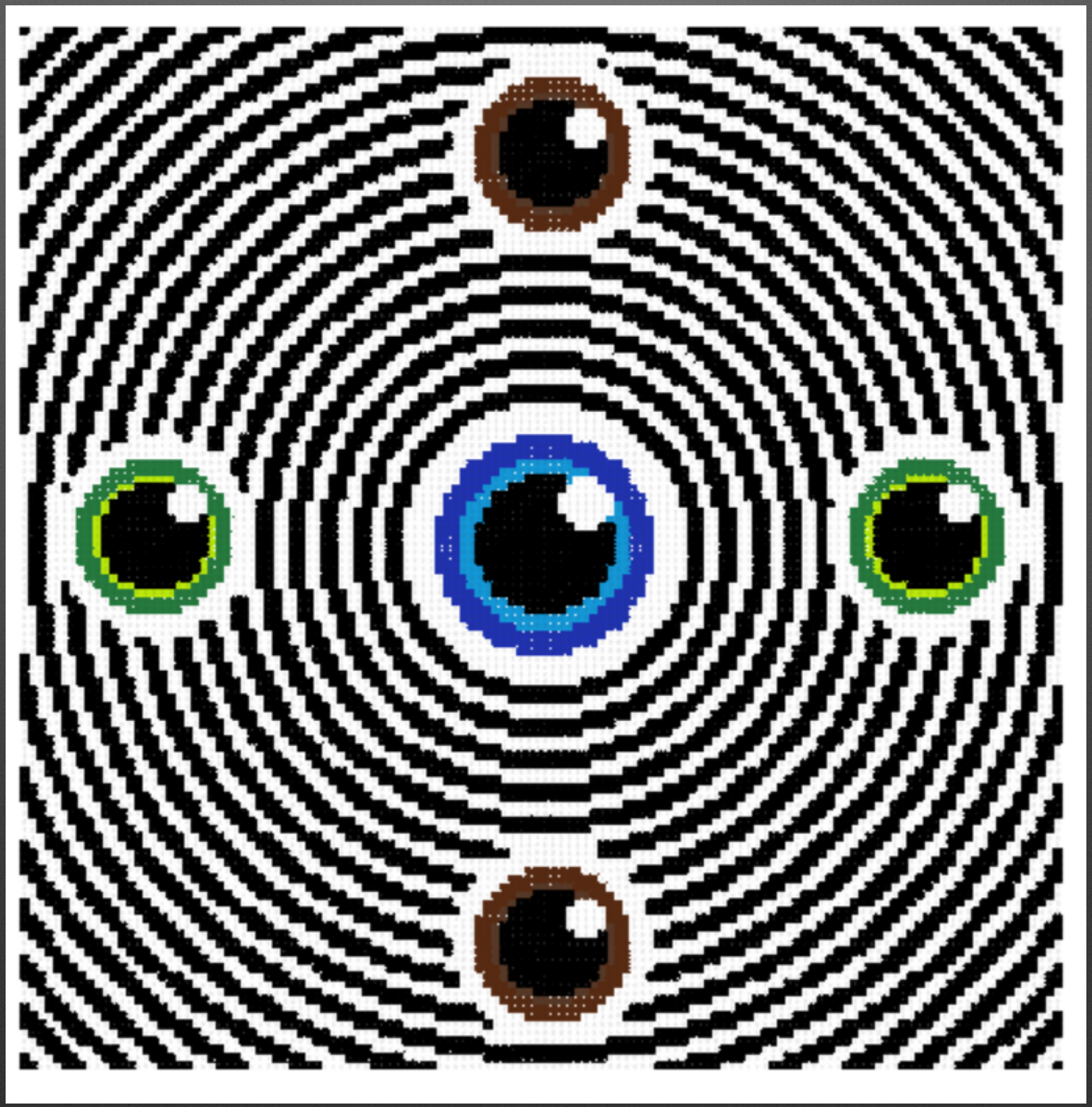
	Initial Tile Size: m x m
Step n	x + *m*∗2^n

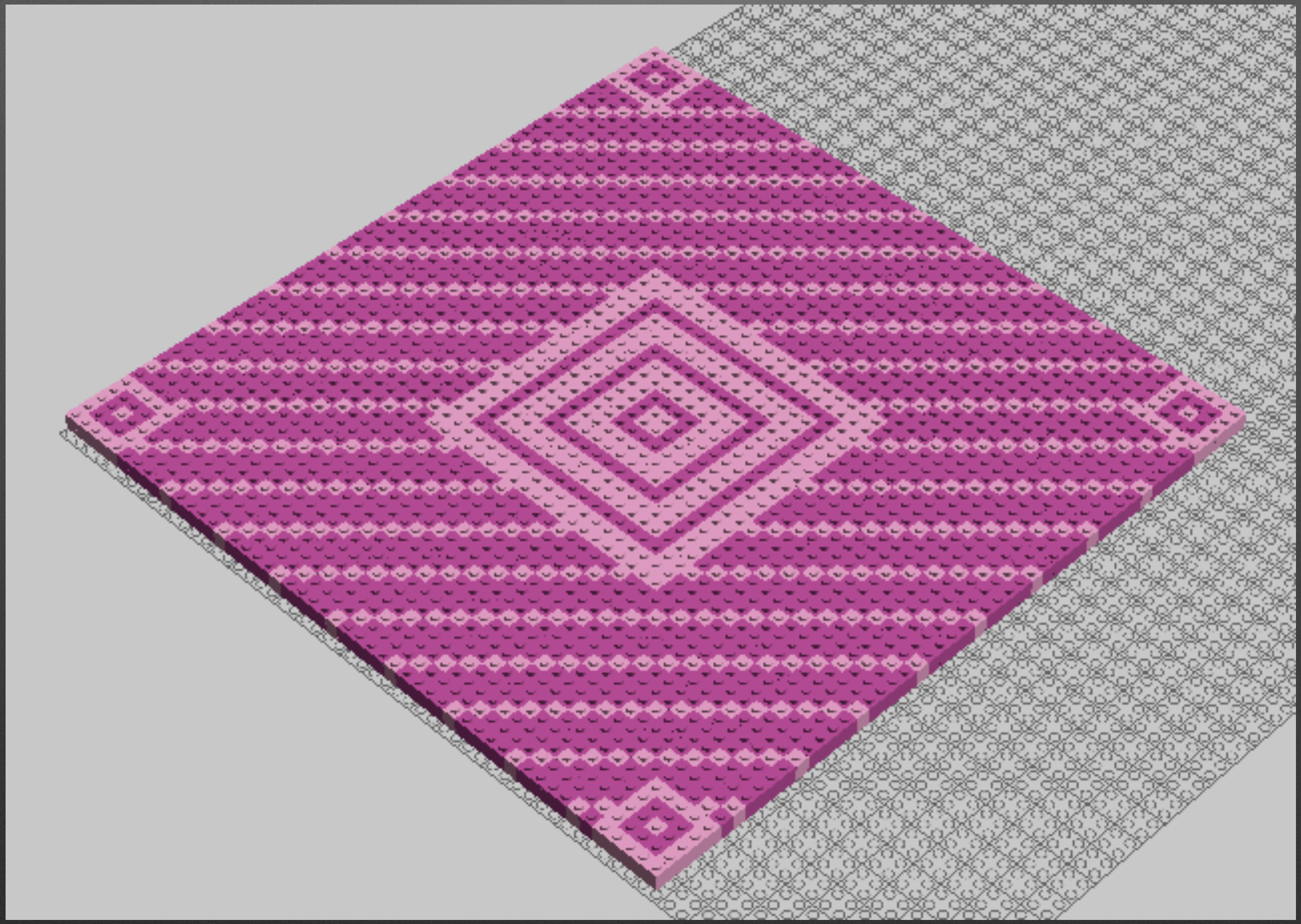
Generalize

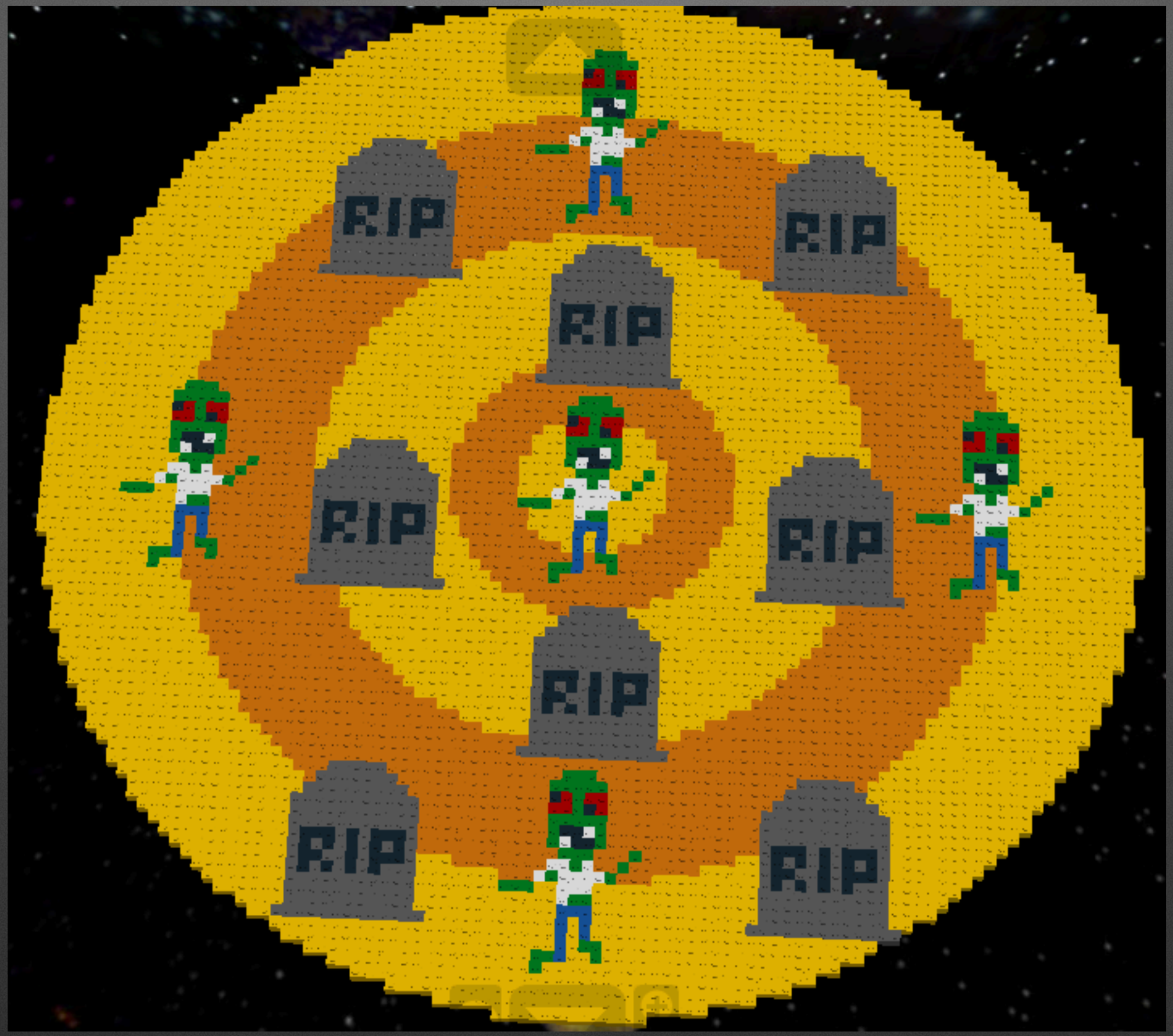
Thinking process to determine artifact position in the general case

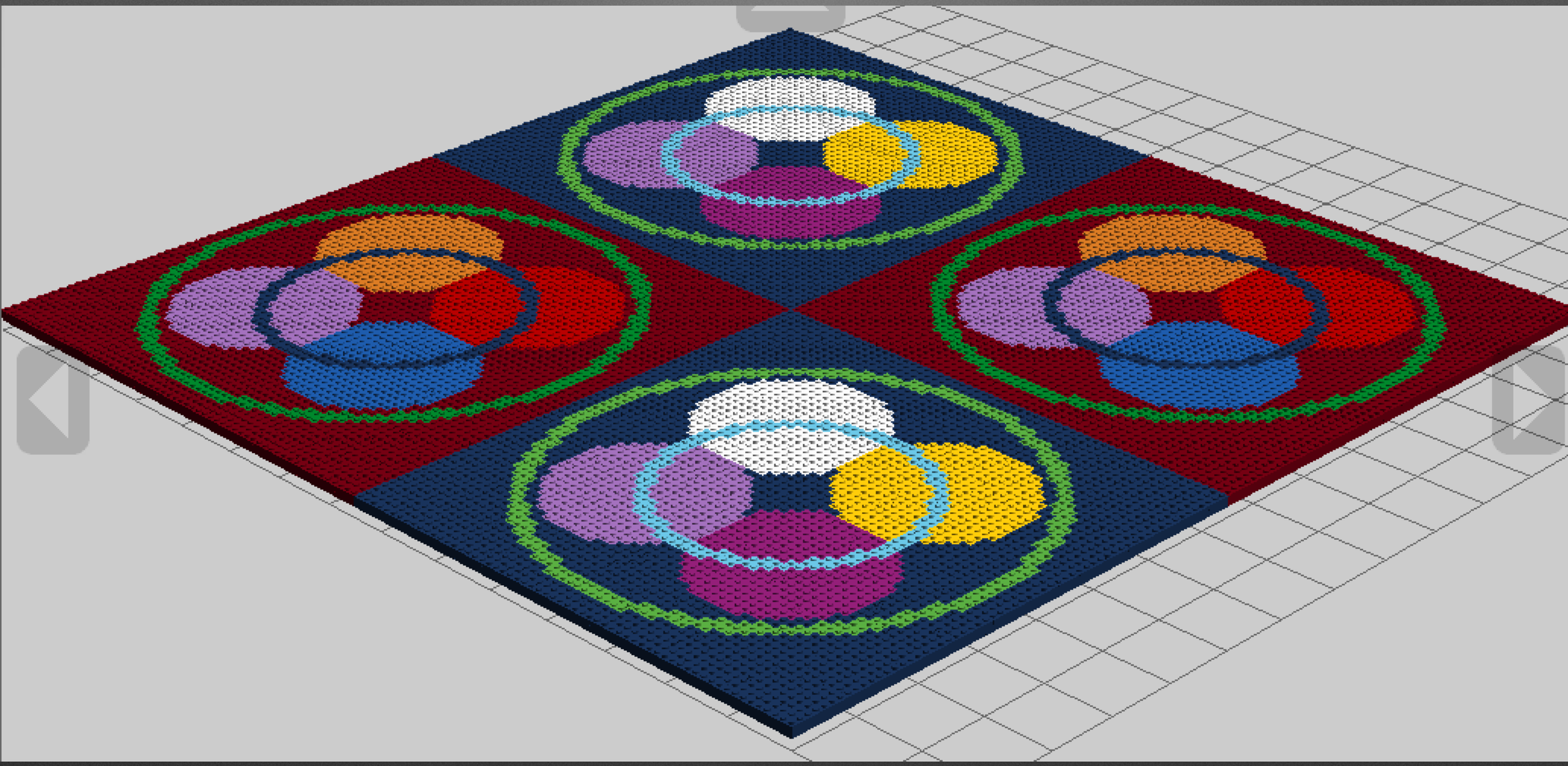
Opportunities for Creativity

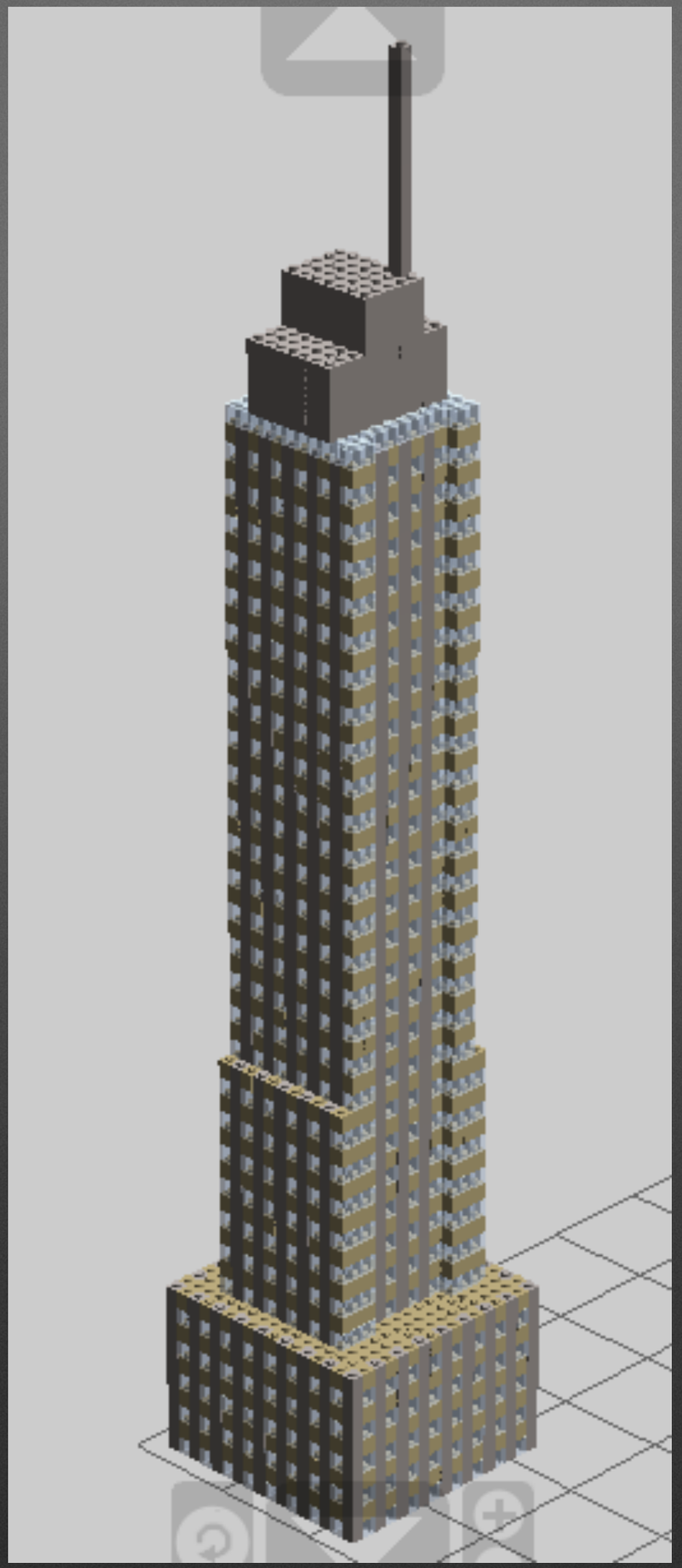
Art Shows!

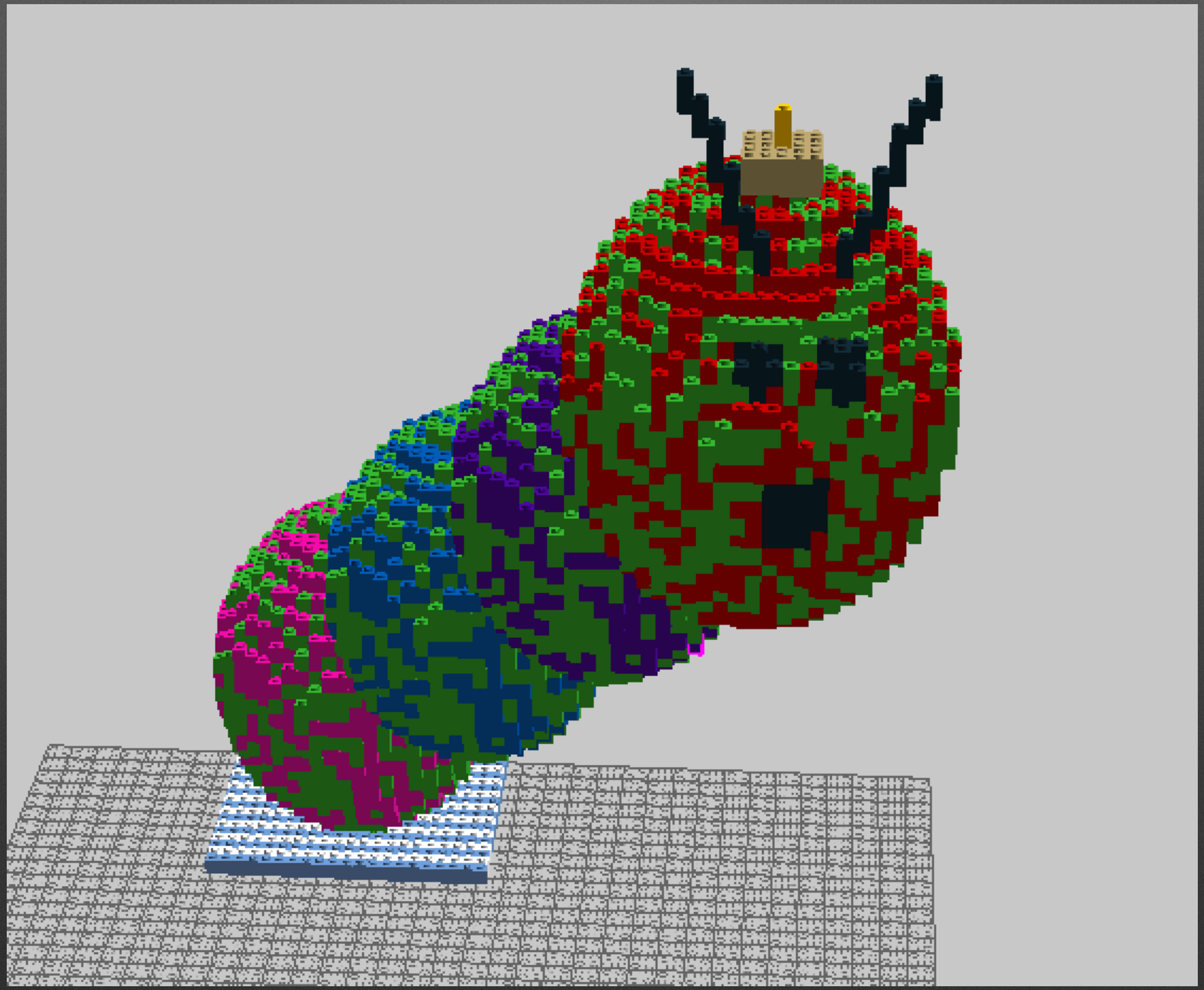


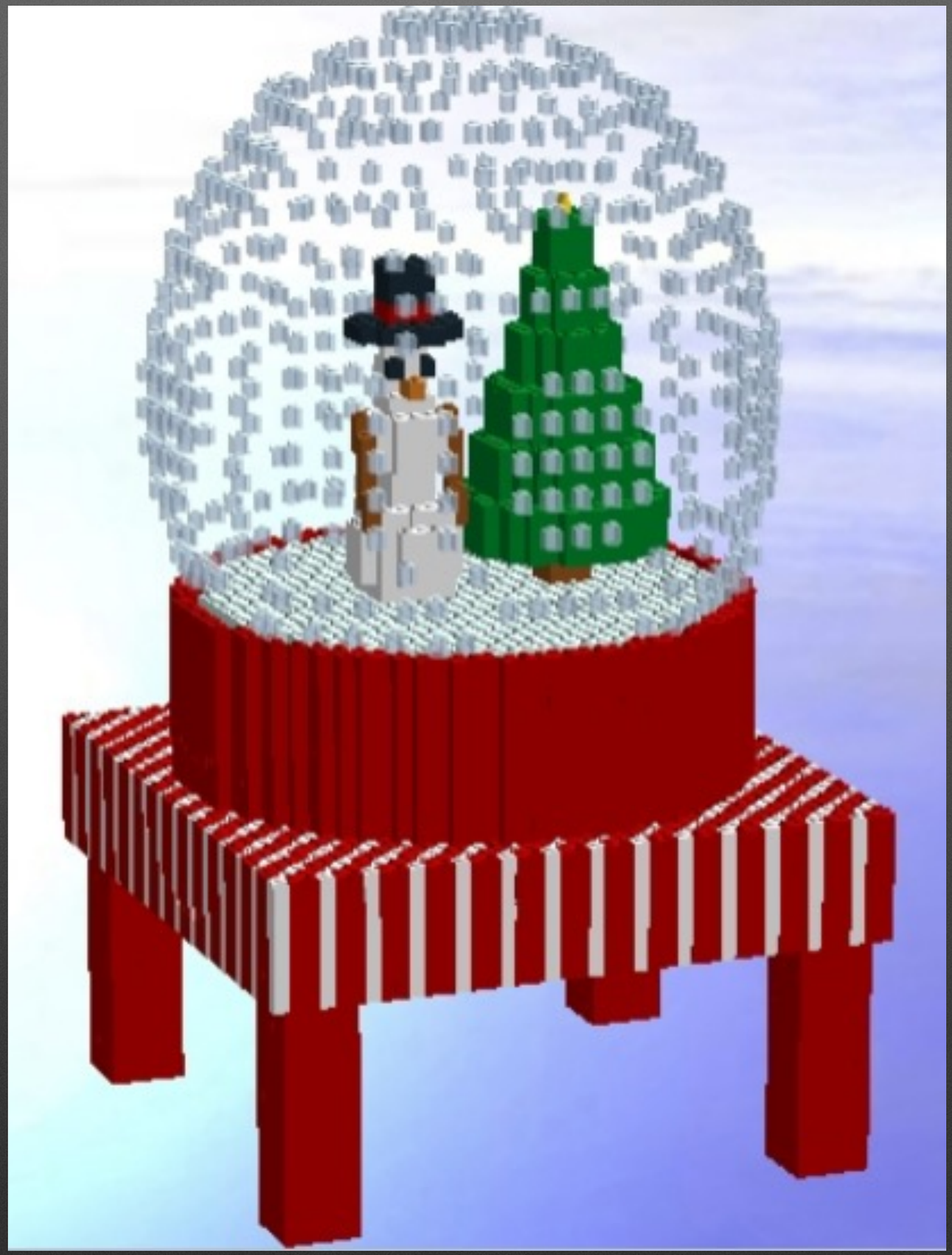


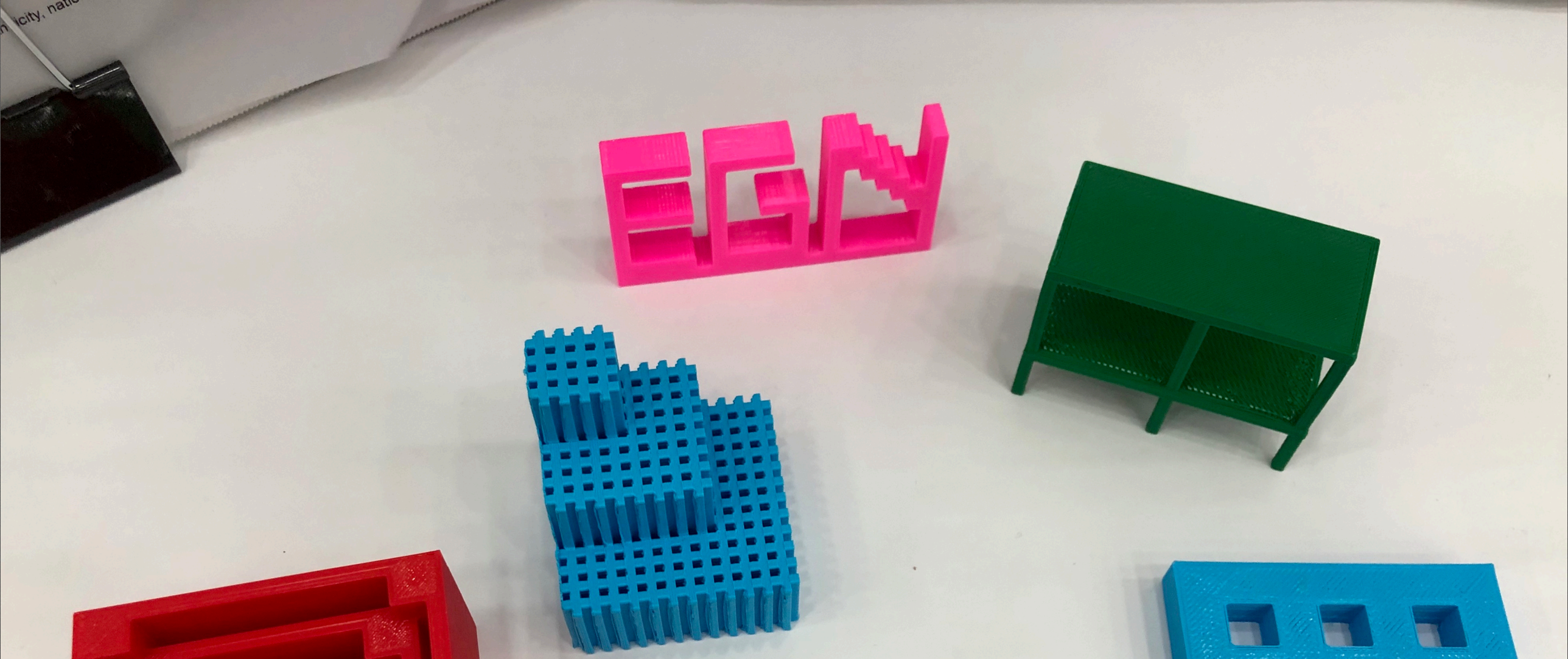

RIP
RIP
RIP
RIP
RIP
RIP
RIP











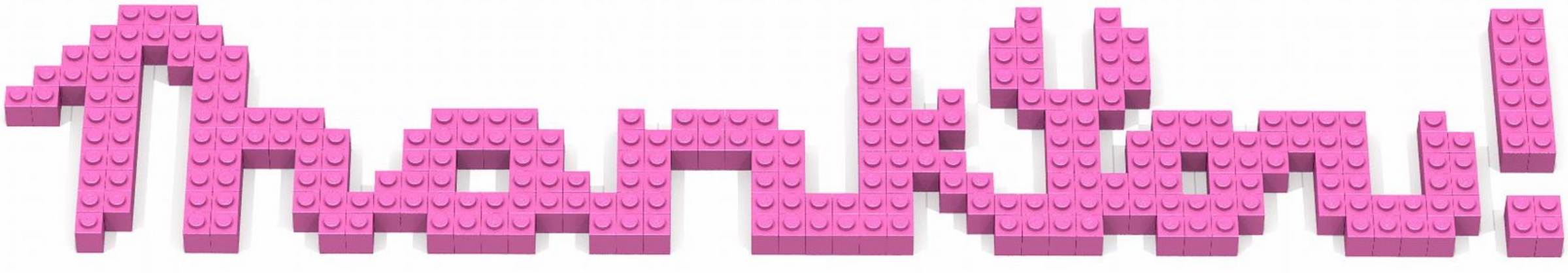
This project is supported by:

Grant # 1712080

We would love to talk to you about implementing our course at your school!

bricklayer.org

All Bricklayer materials and software are free.


Thank You!