

Labeled Reconfiguration by Compaction



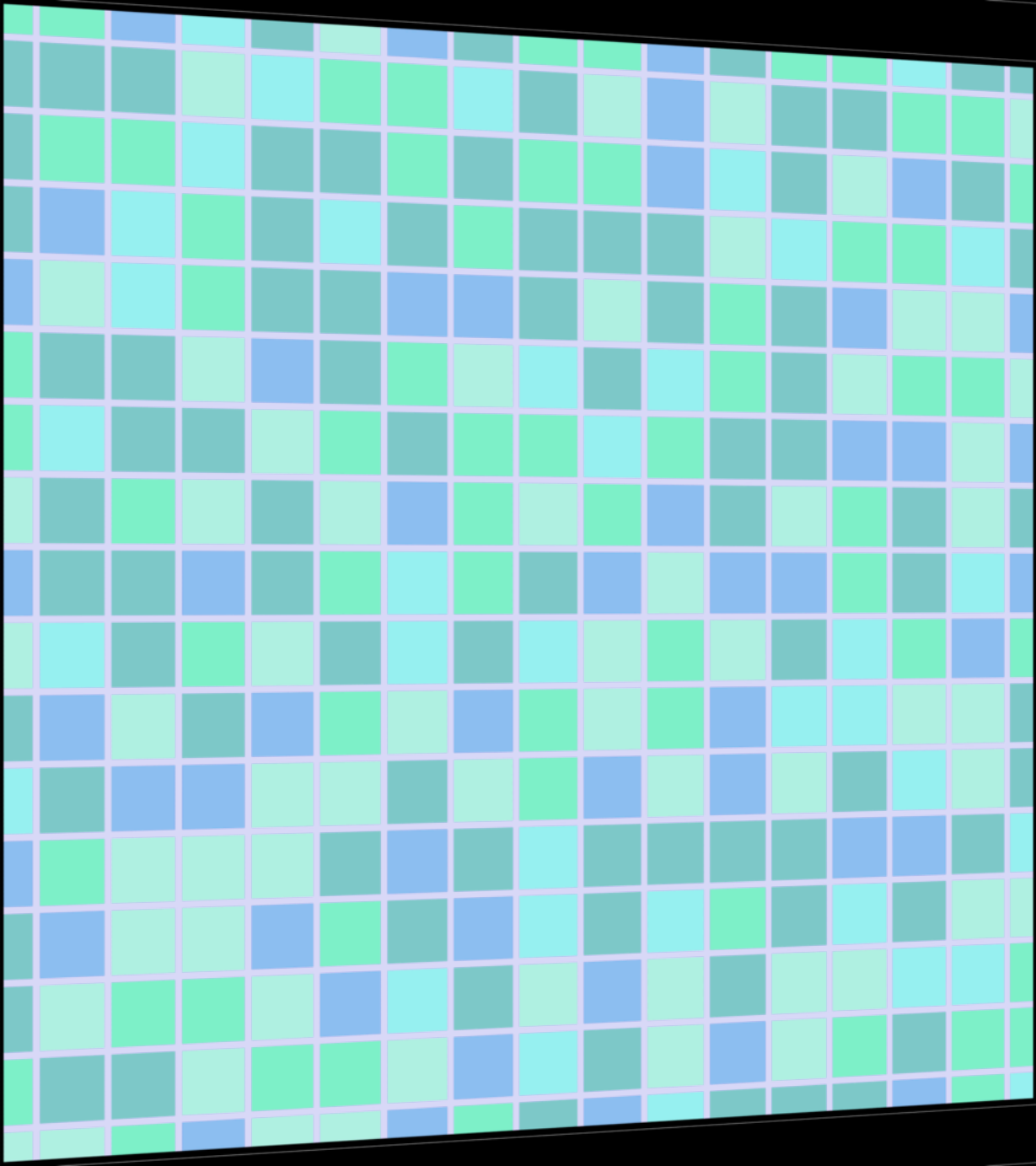
Maarten Löffler

Based on joint work with

Hugo Akitaya
Greg Aloupis
Anika Rounds
Giovanni Viglietta

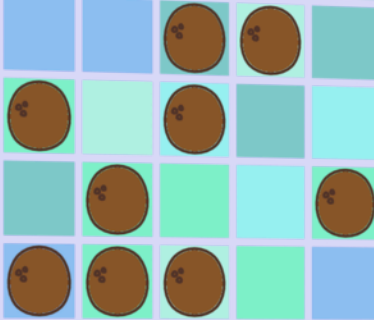


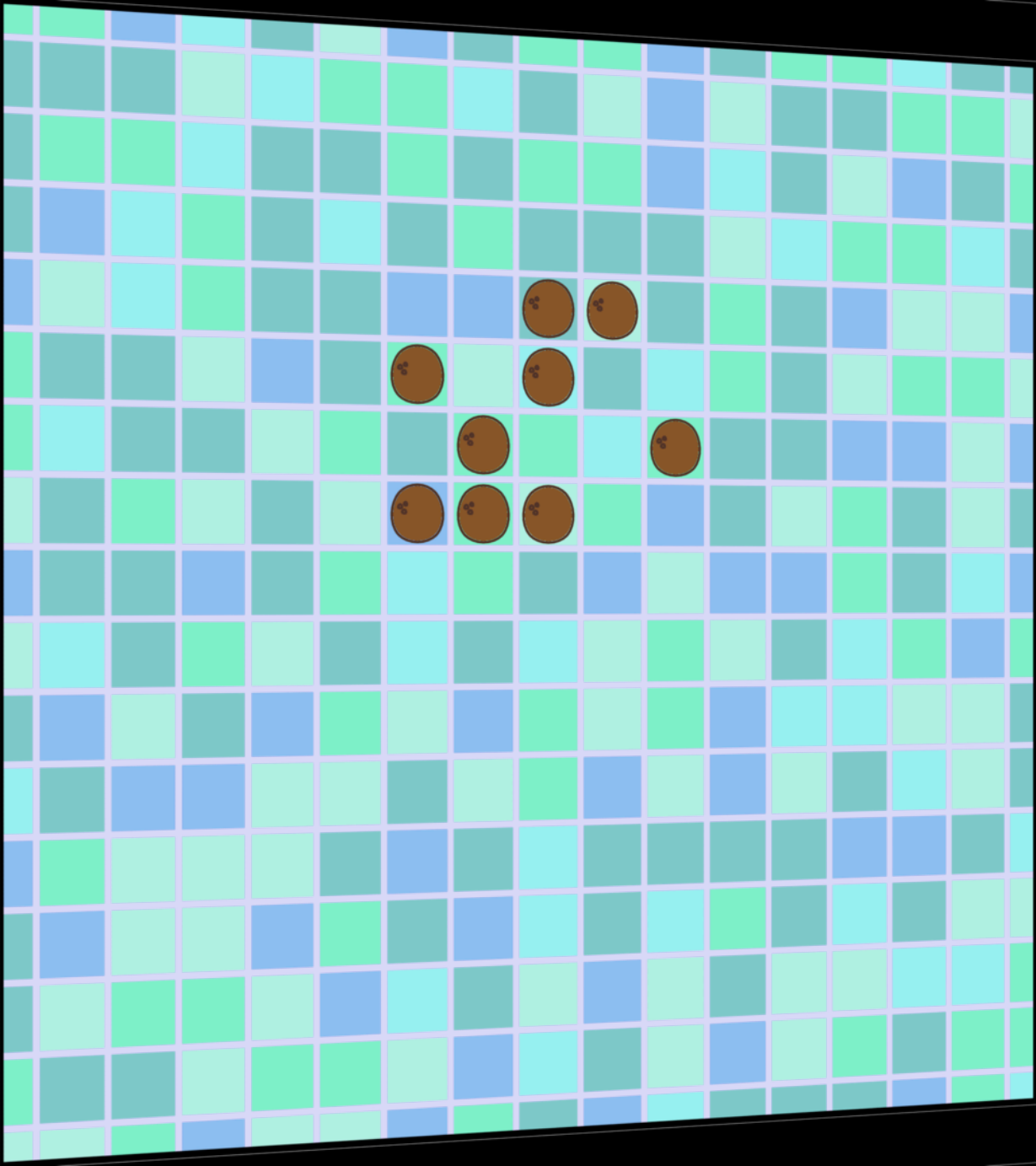




Let P be a set of n
coconuts in a pool.

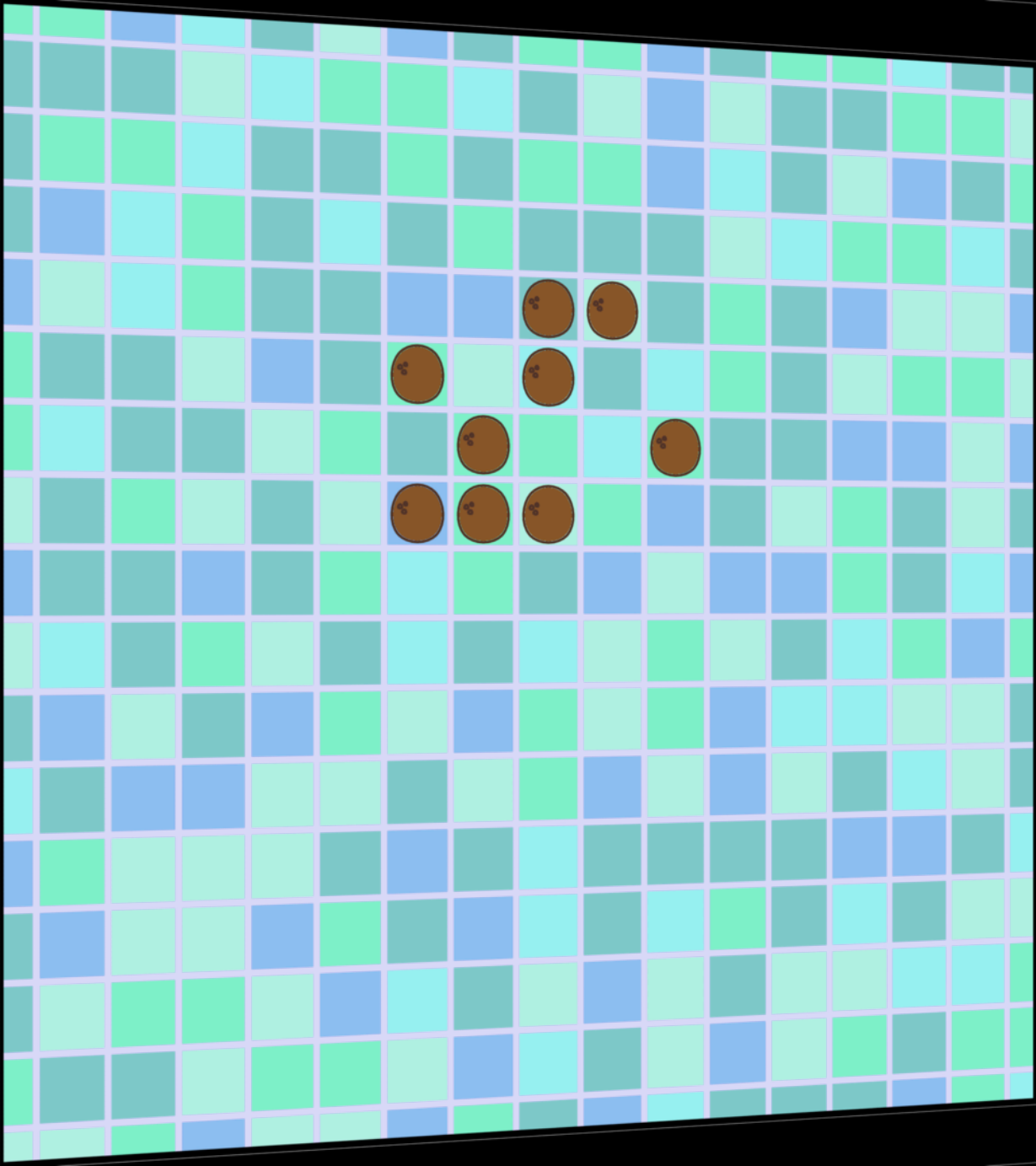
Let P be a set of n
coconuts in a pool.





Let P be a set of n
coconuts in a pool.

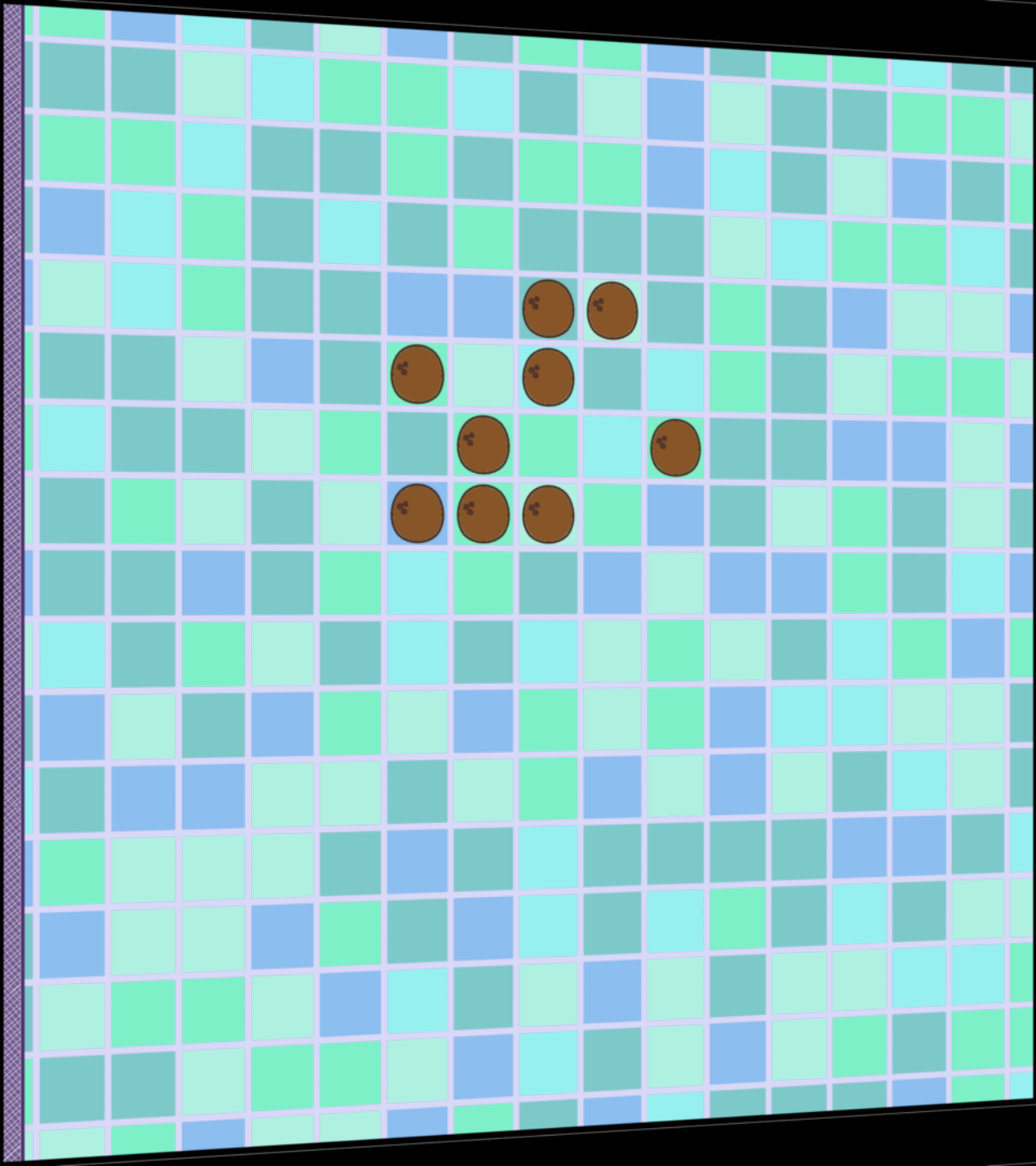
We assume the coconuts are
aligned to a grid.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

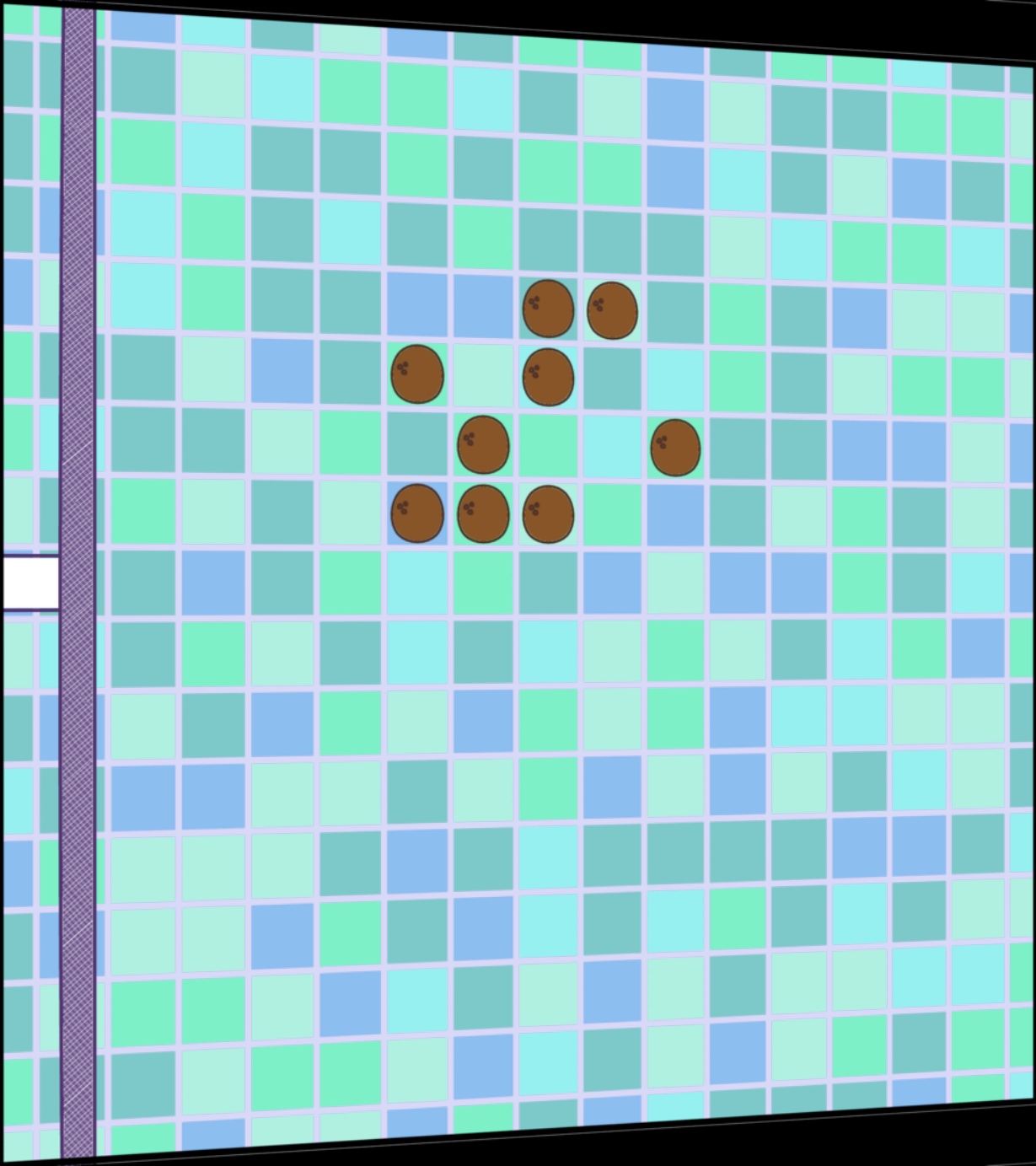
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

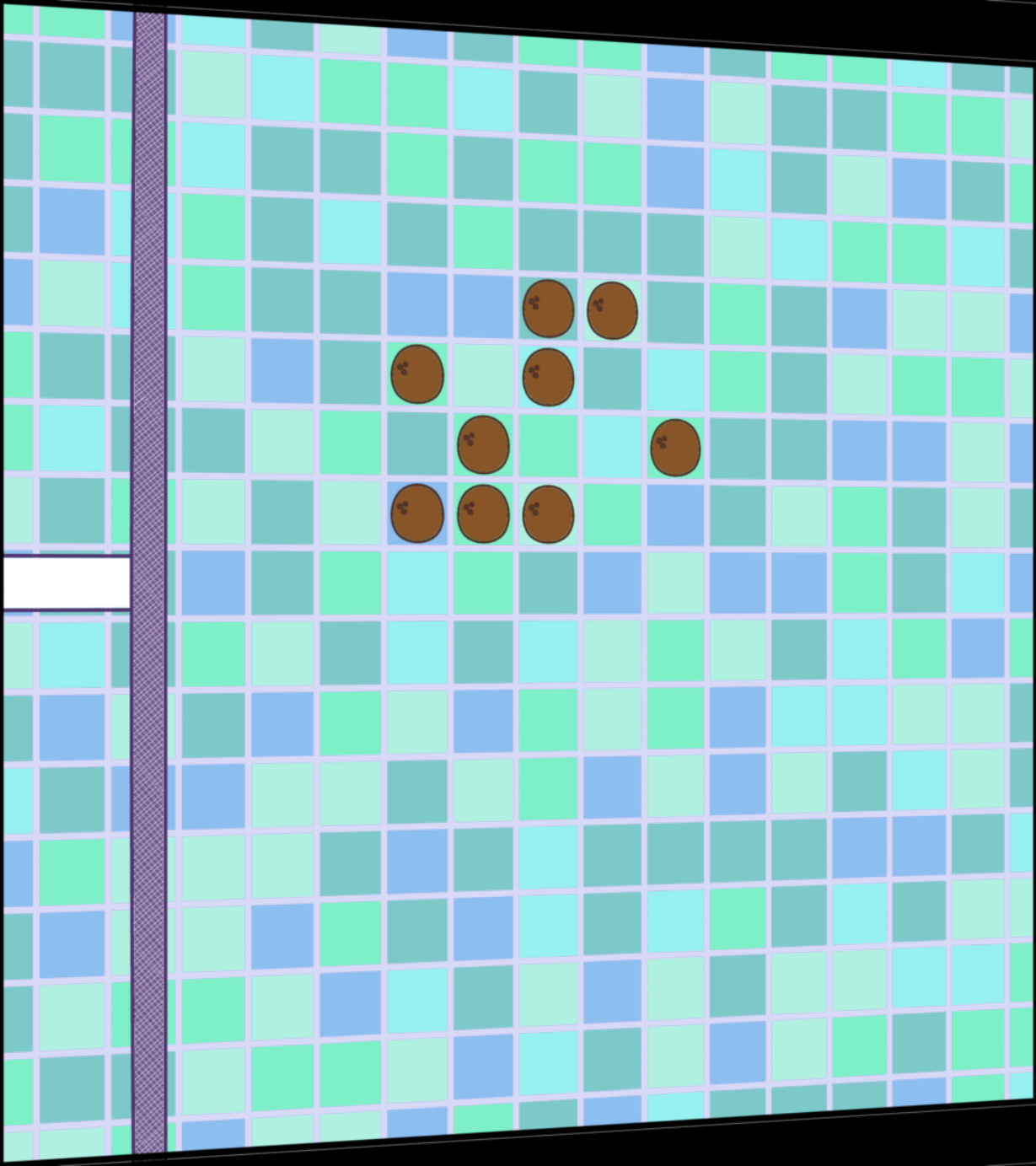
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

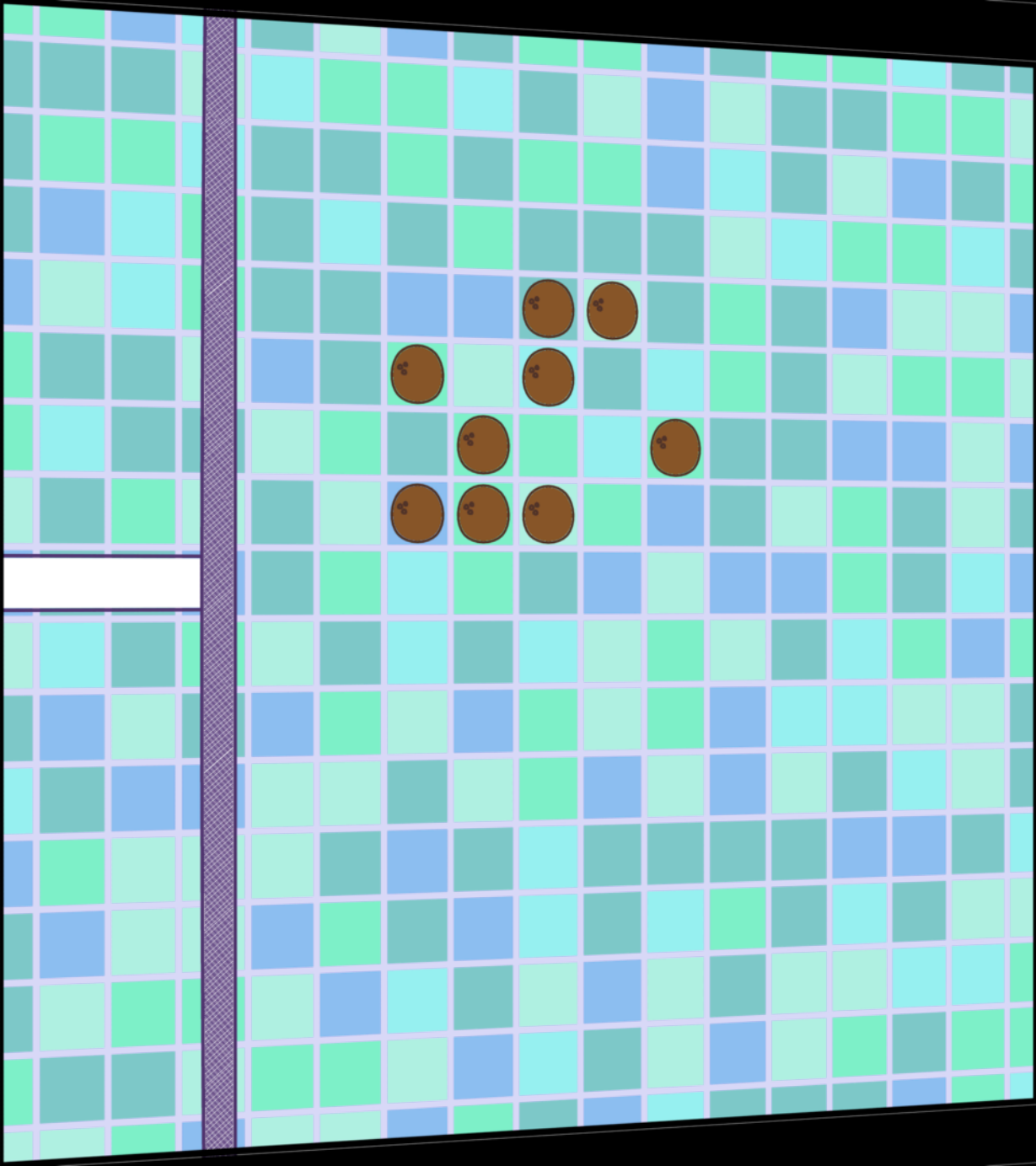
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

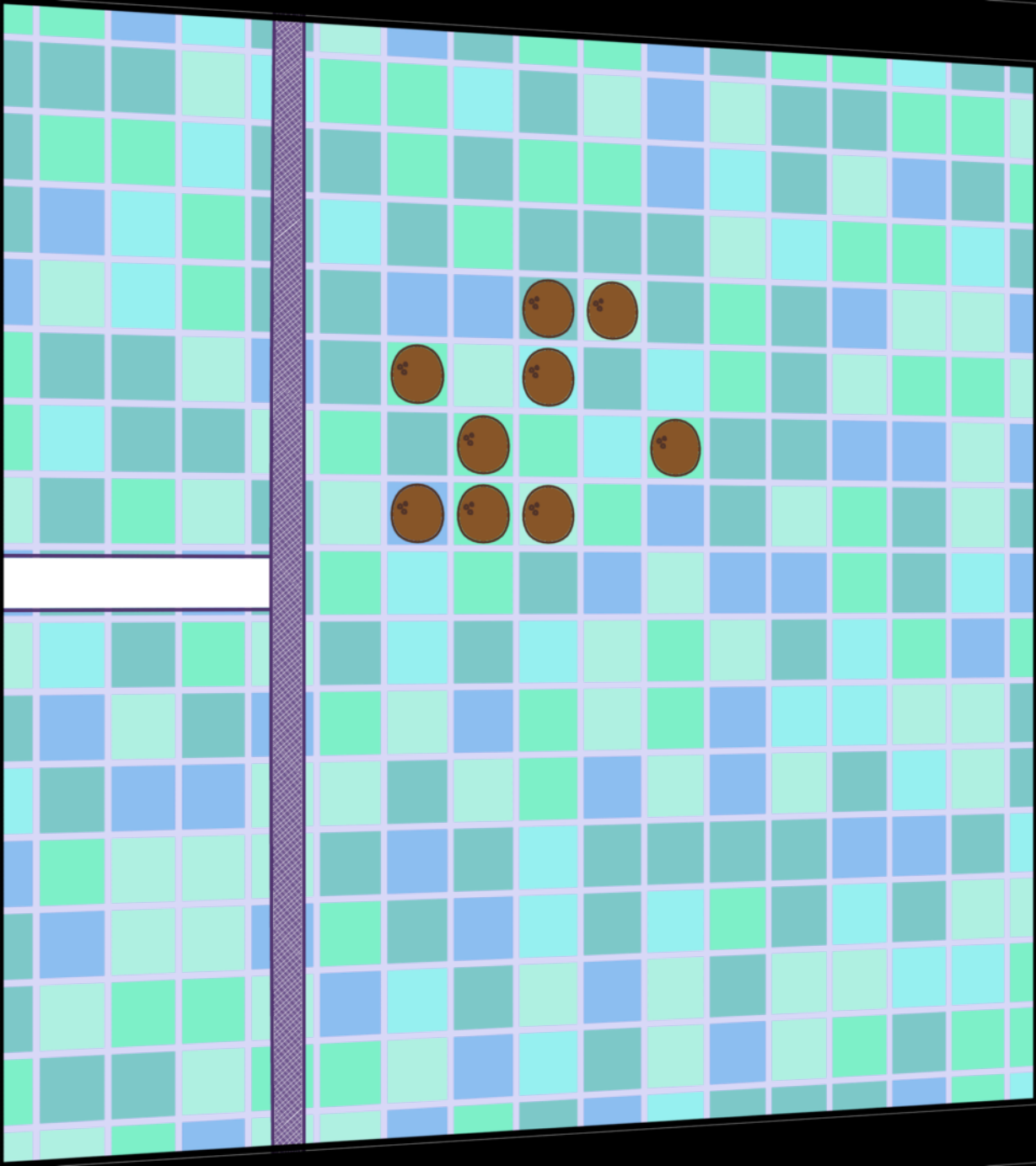
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

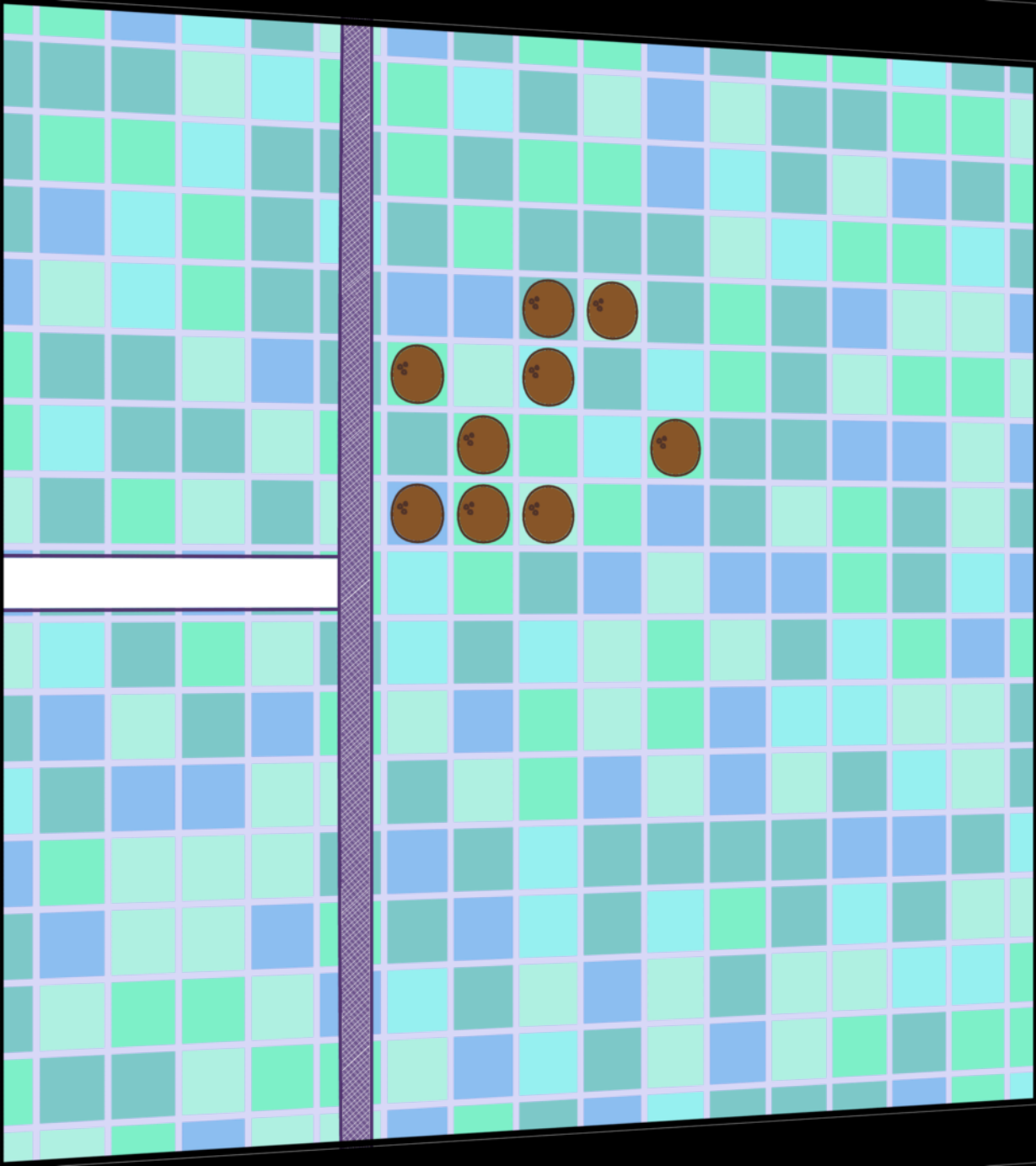
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

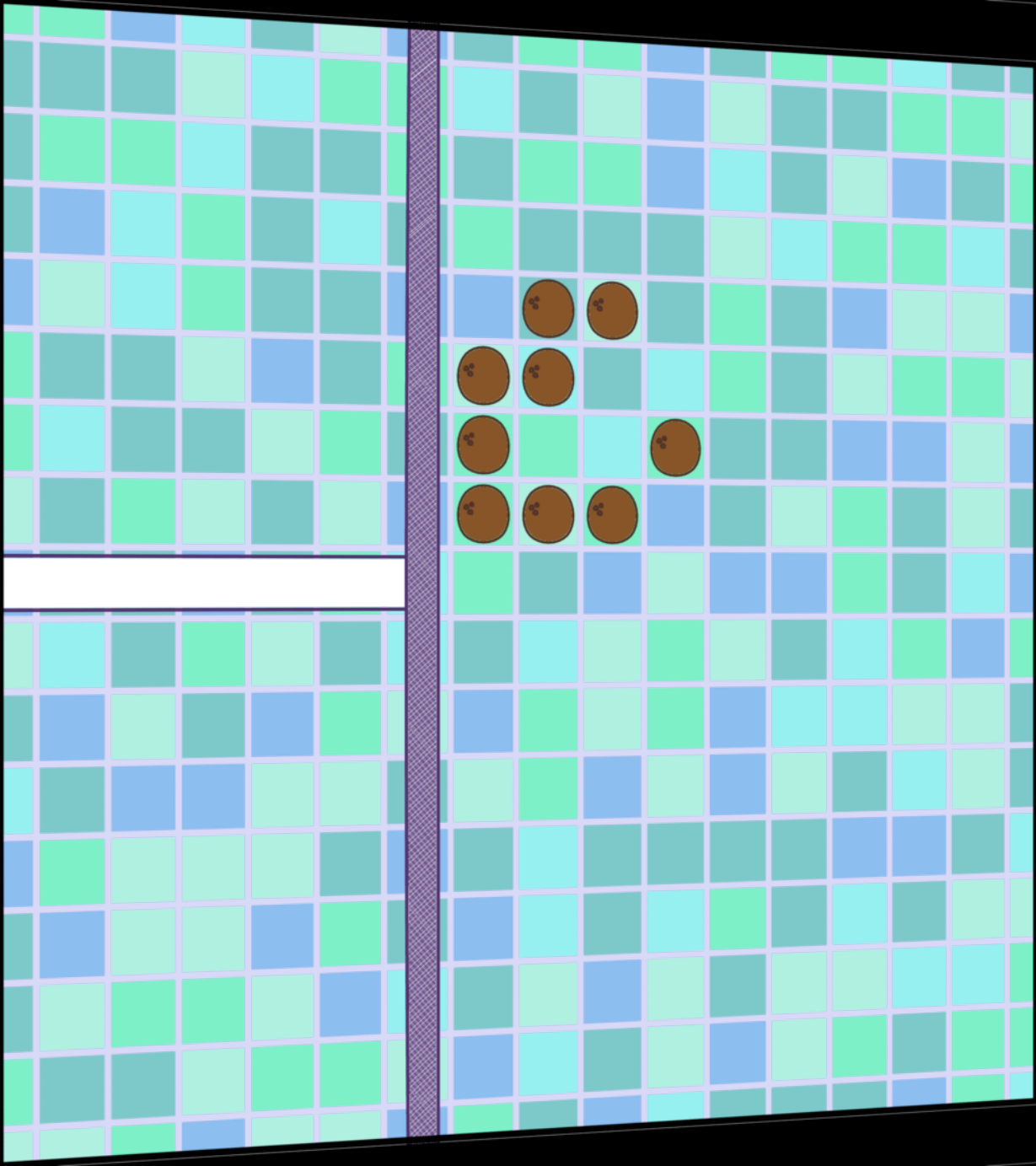
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

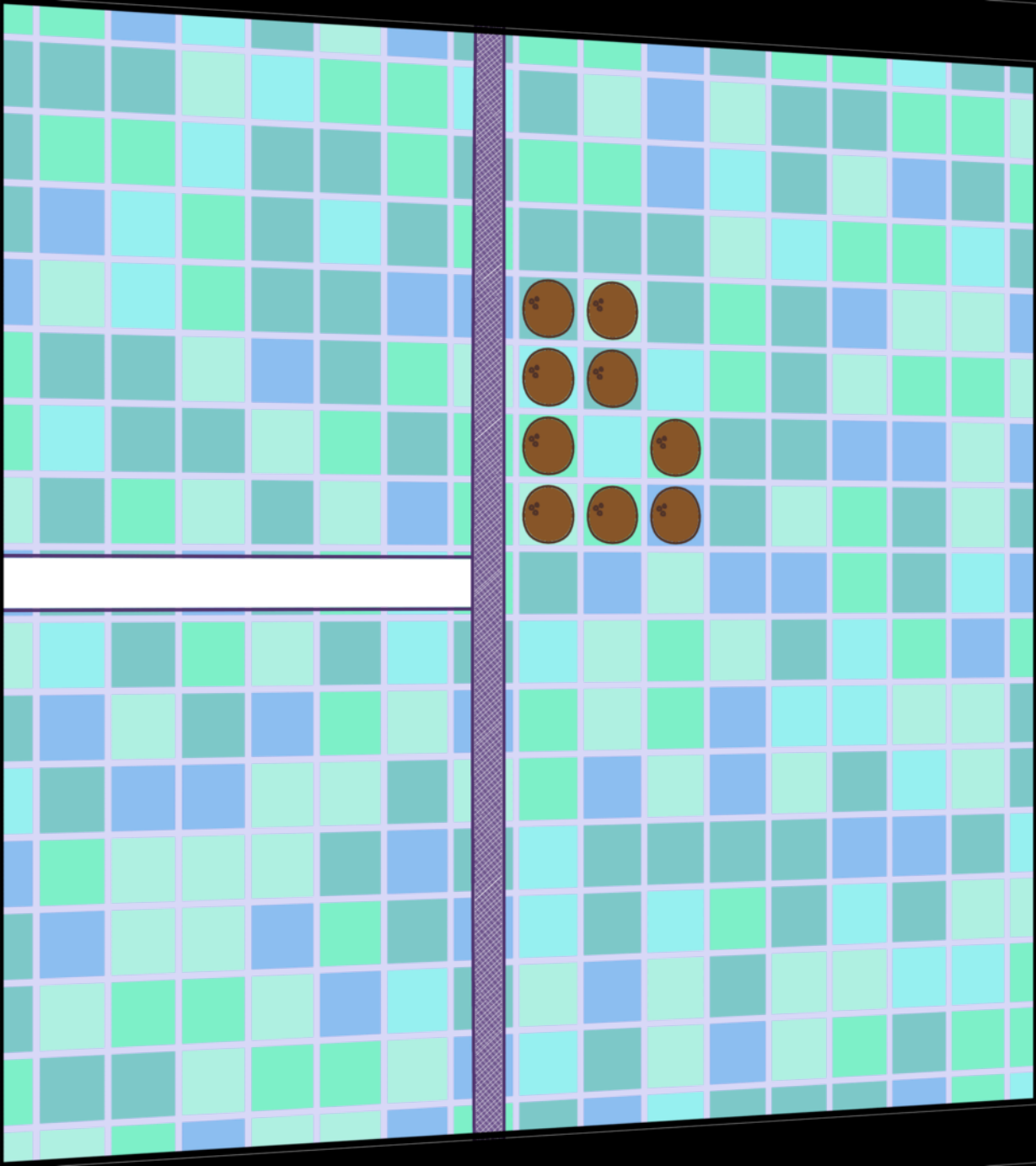
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

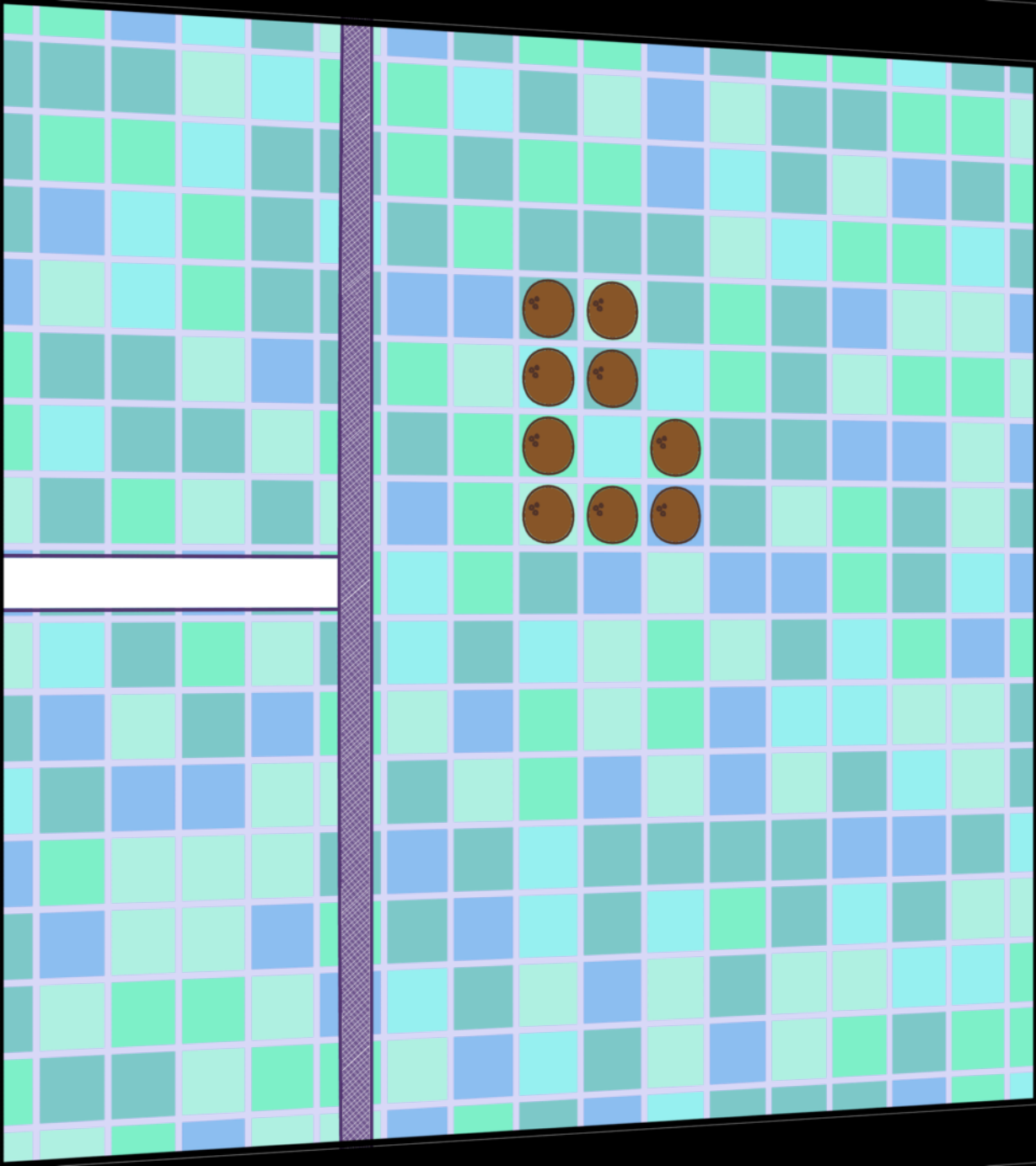
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

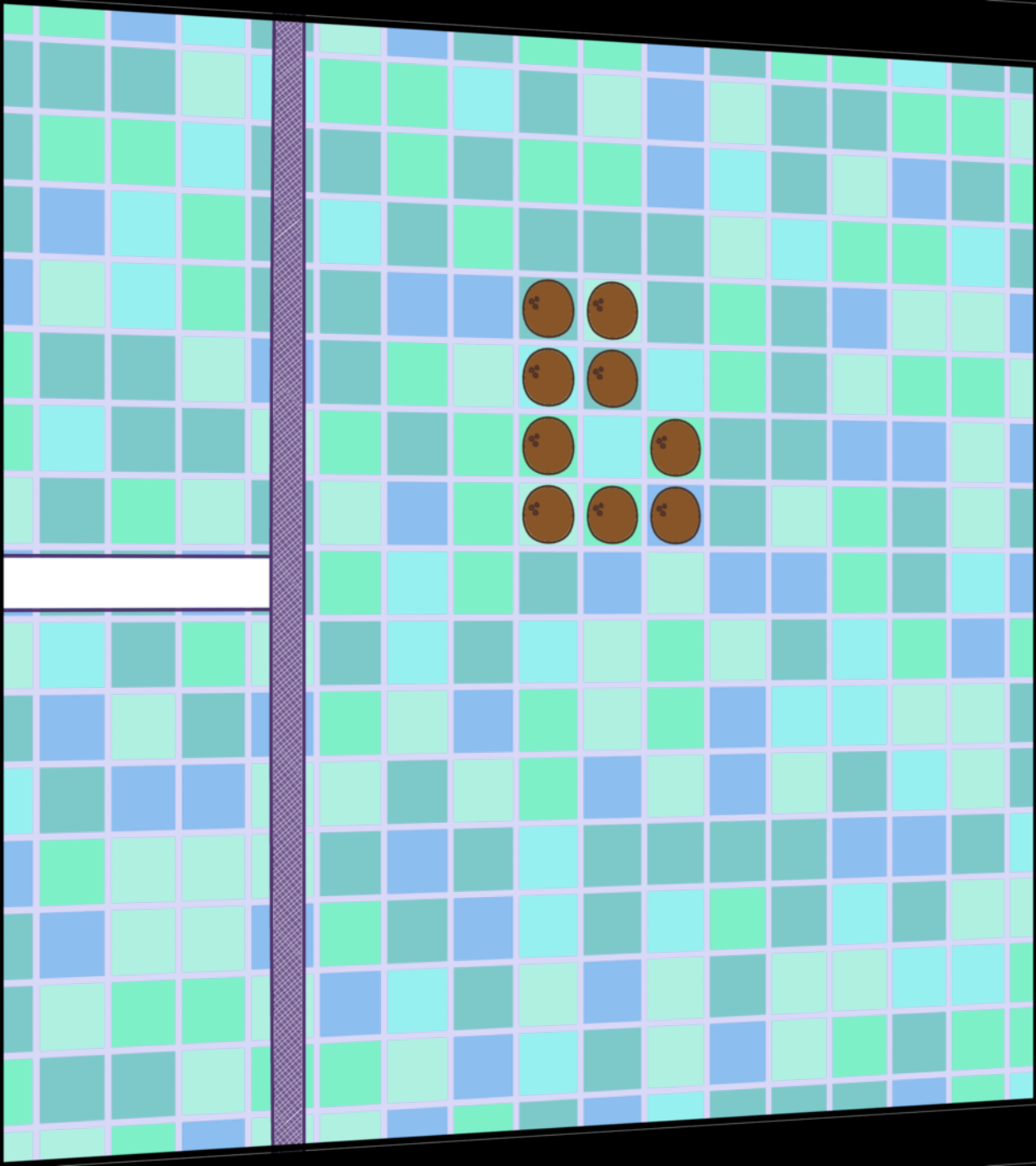
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

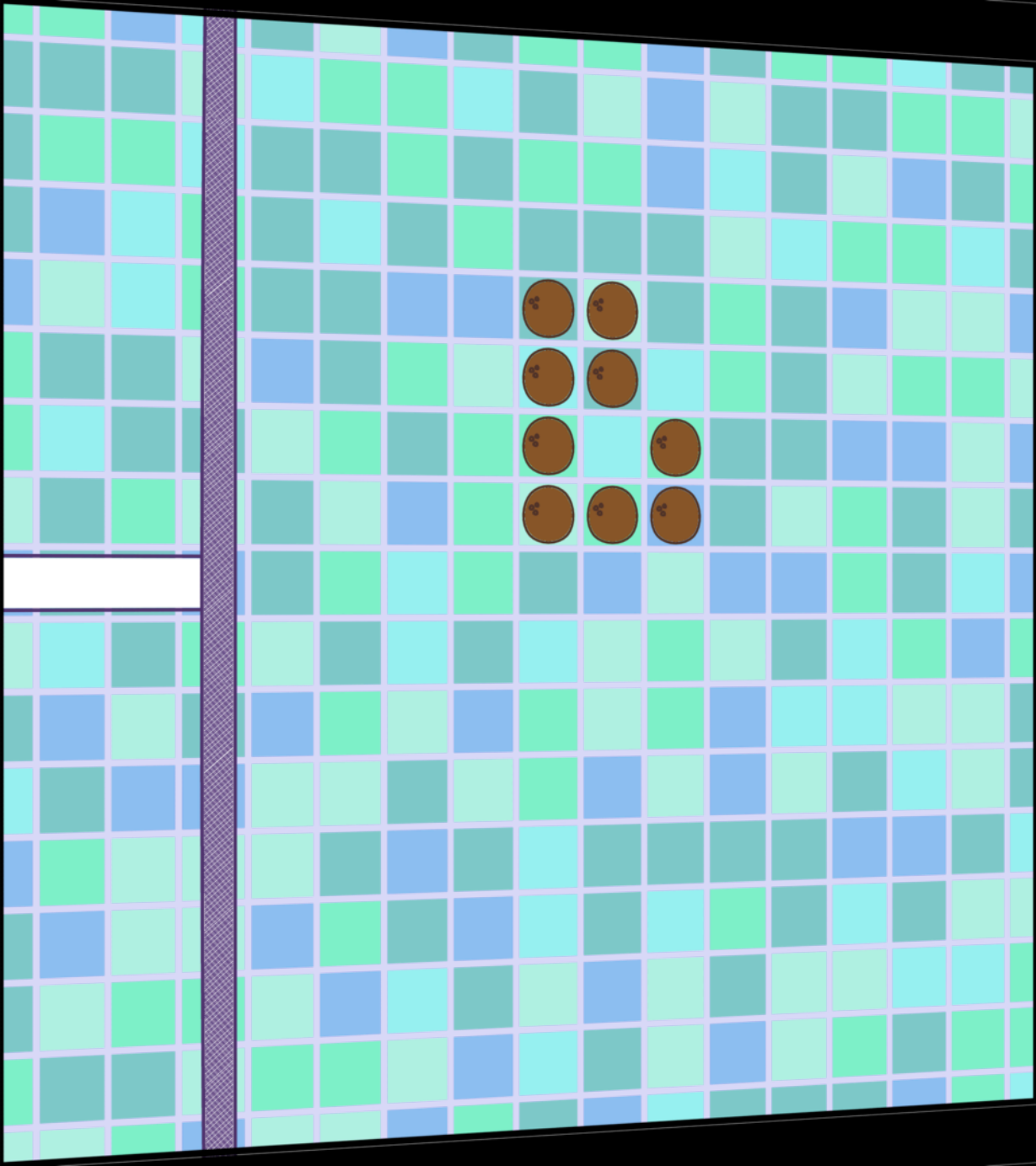
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

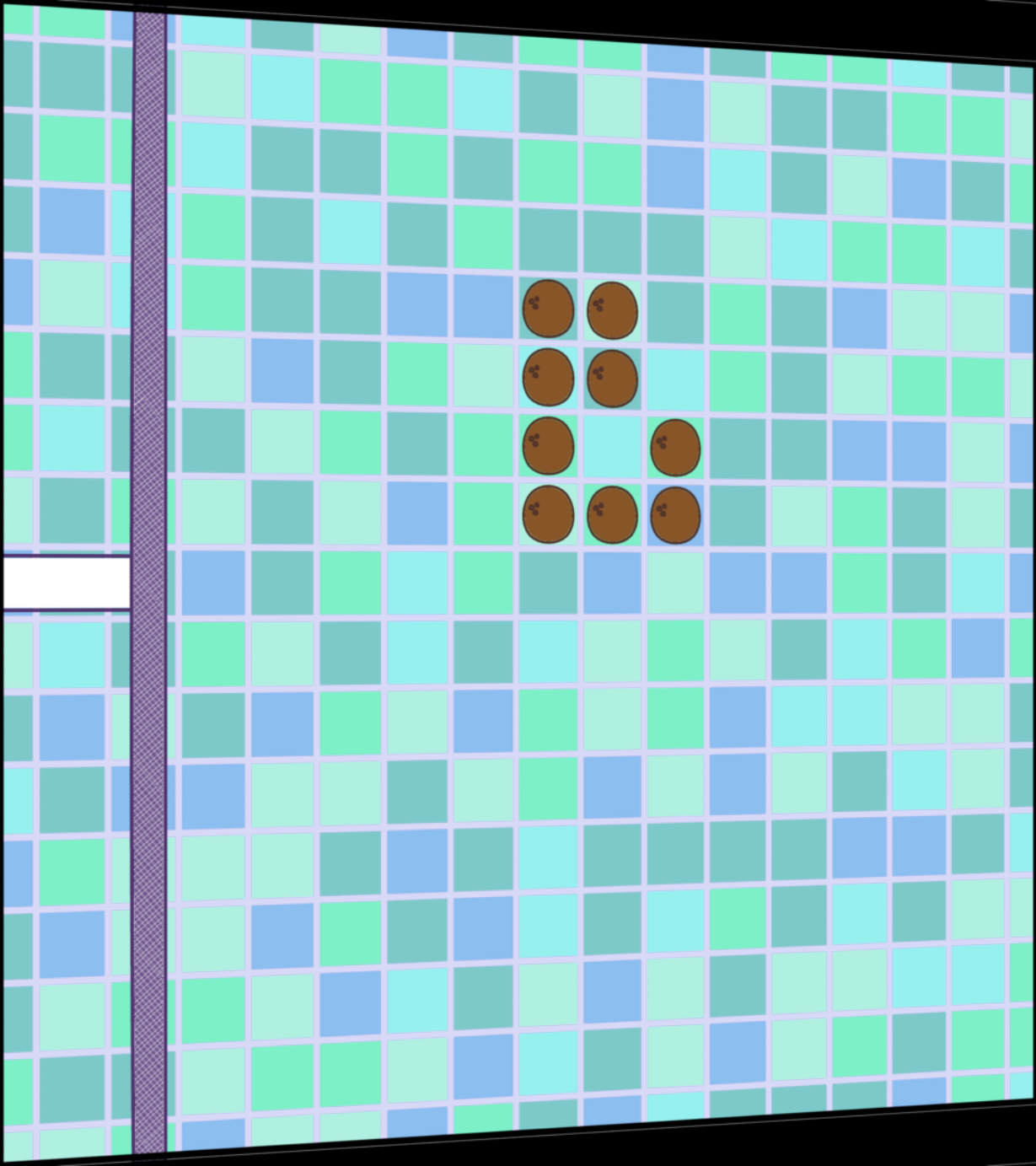
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

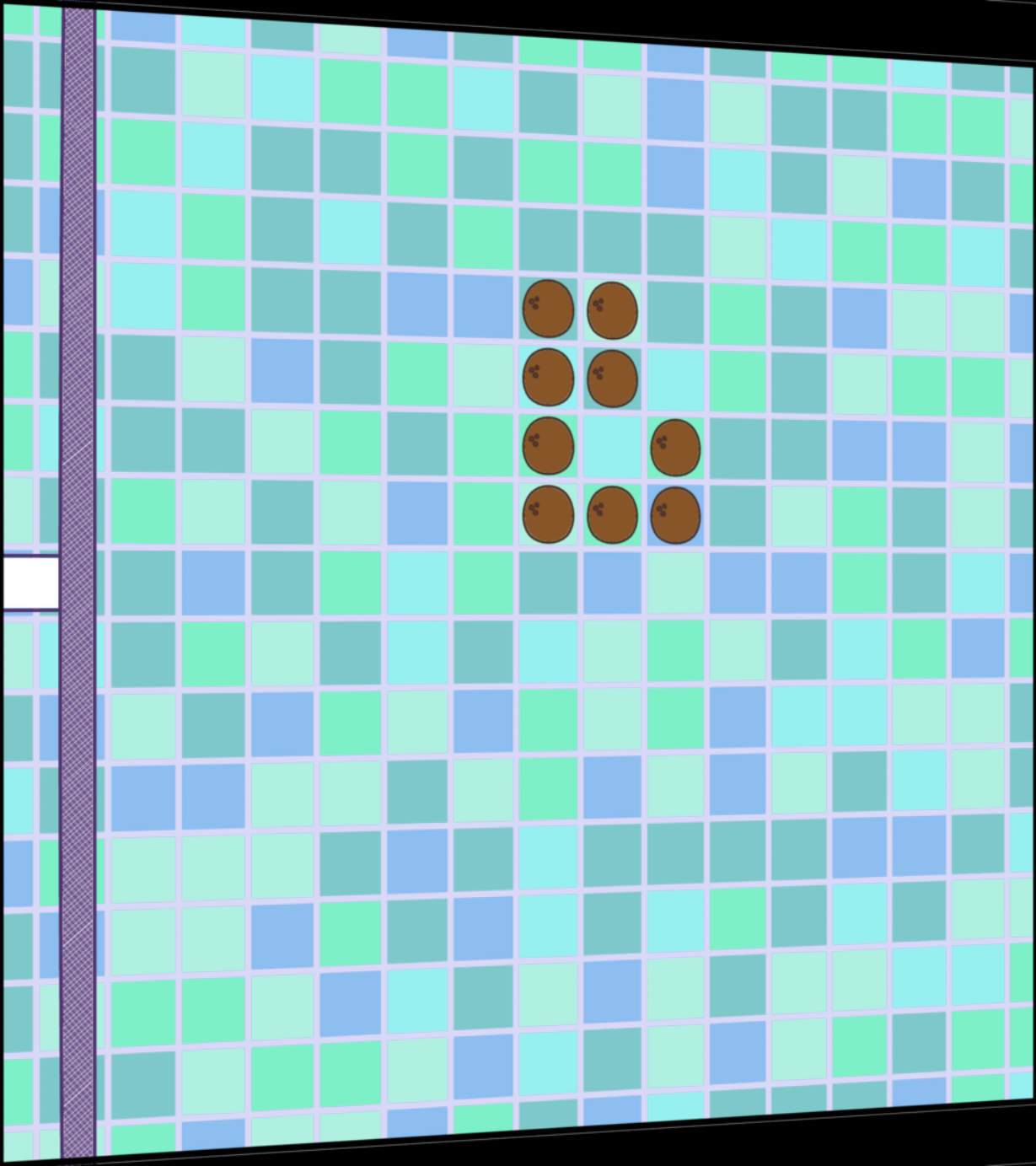
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

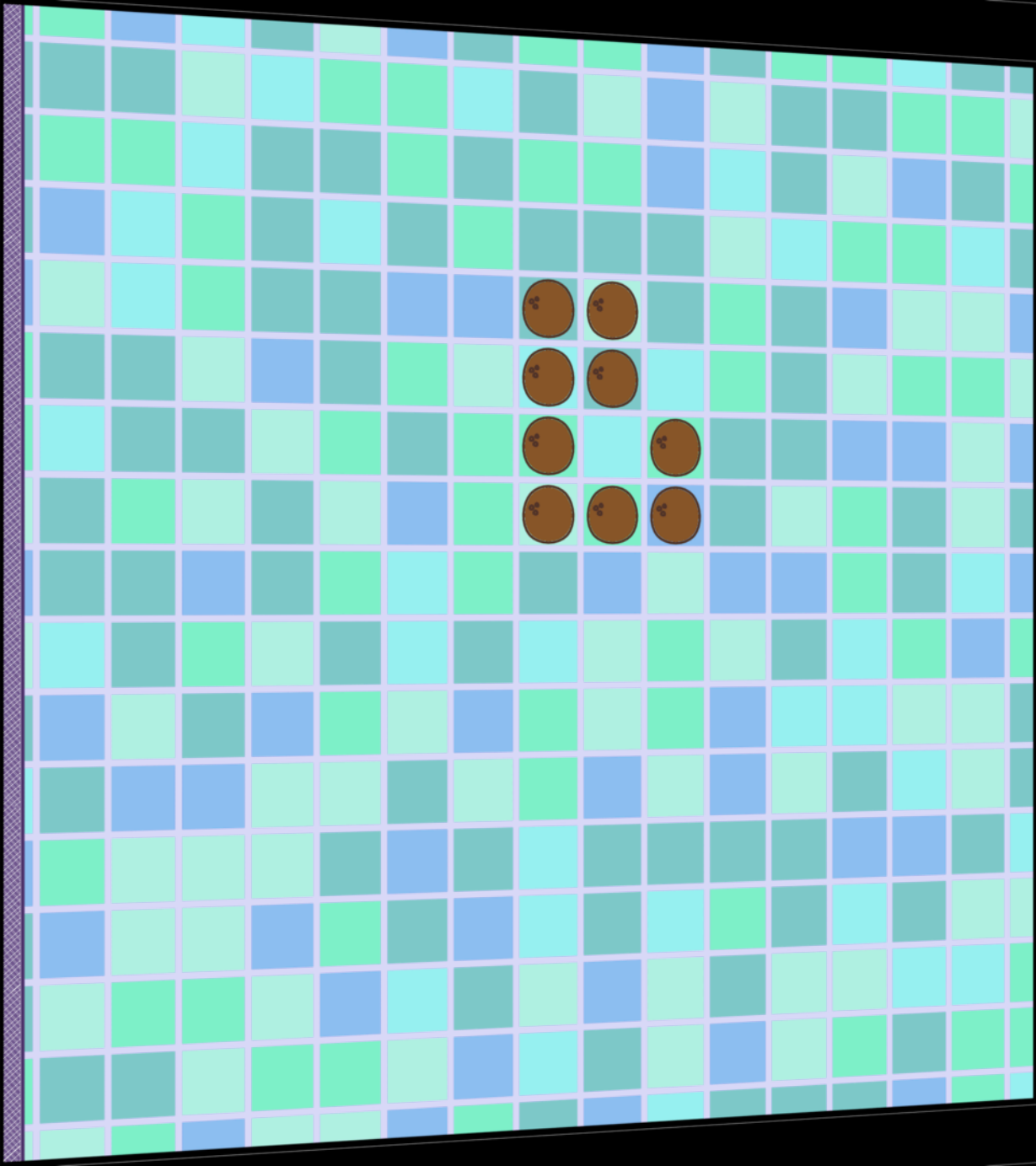
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

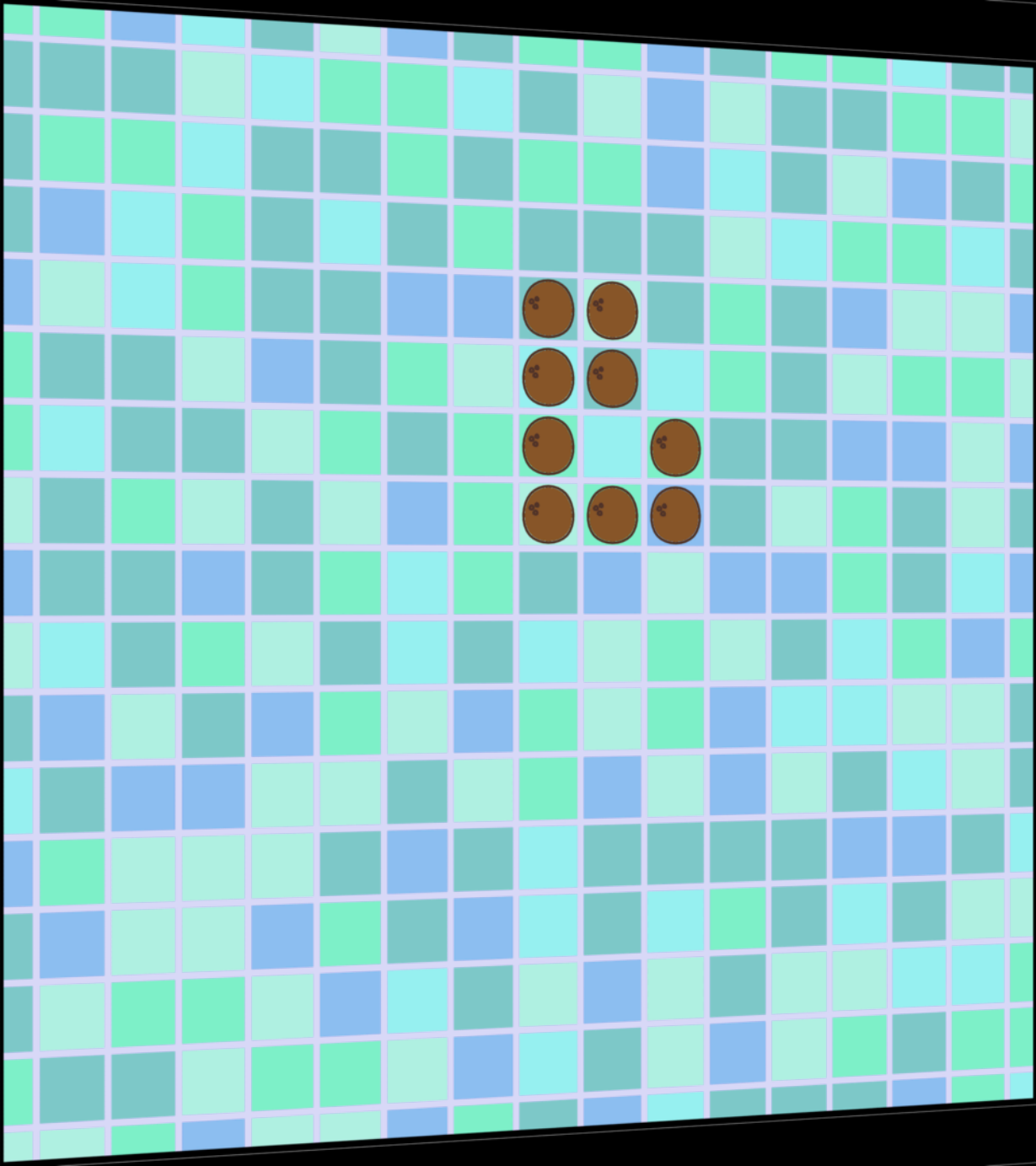
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

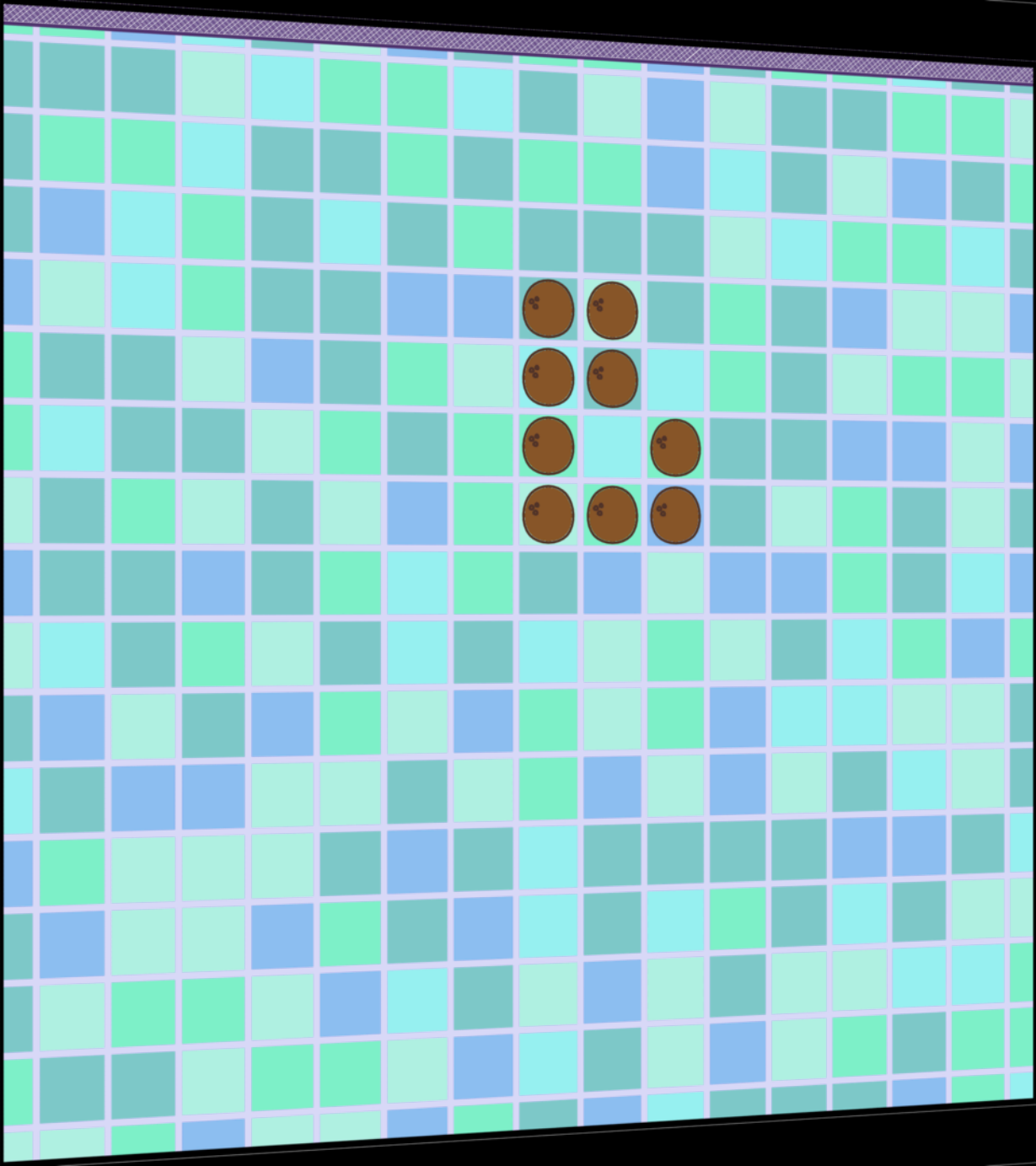
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

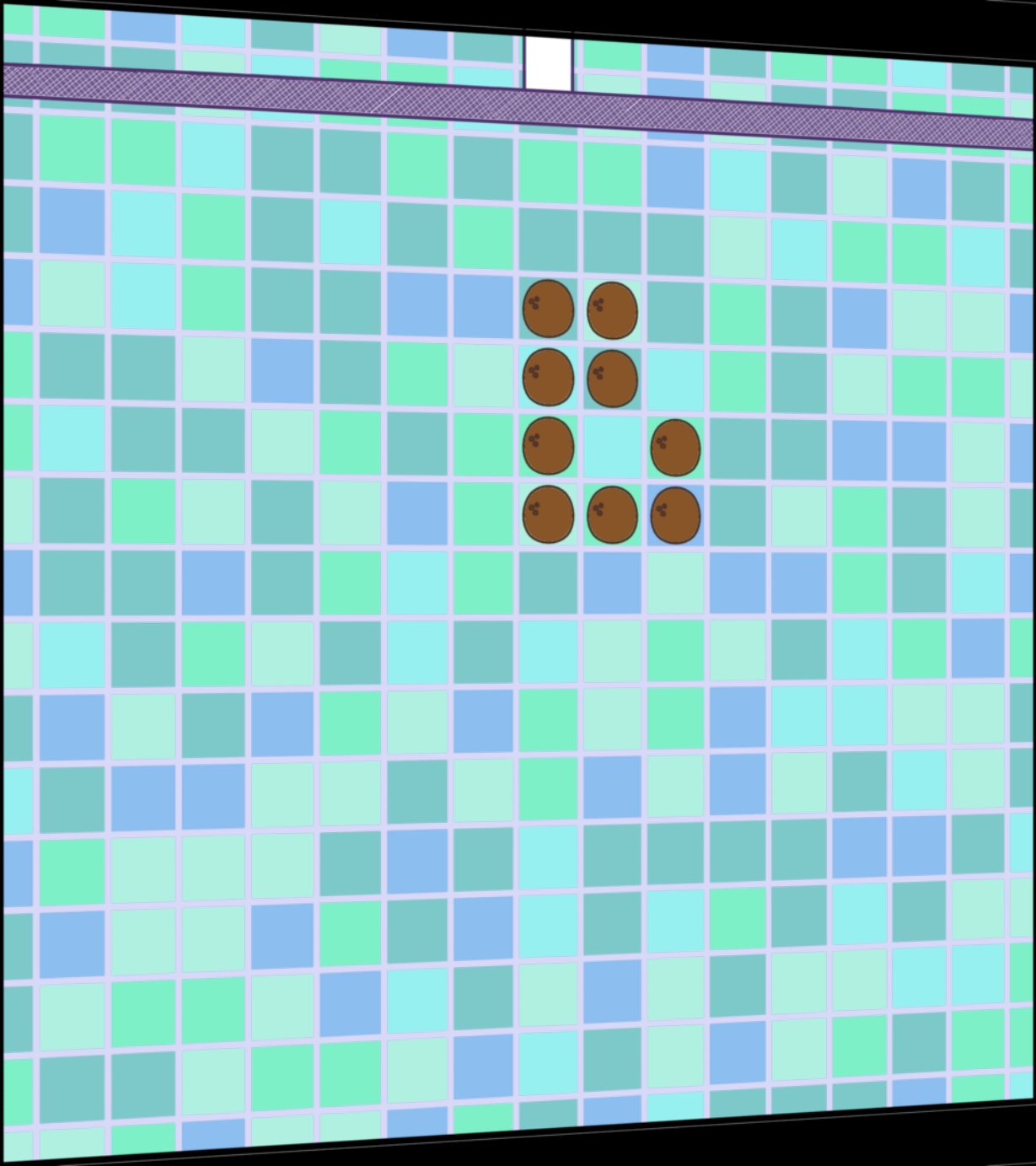
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let T be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

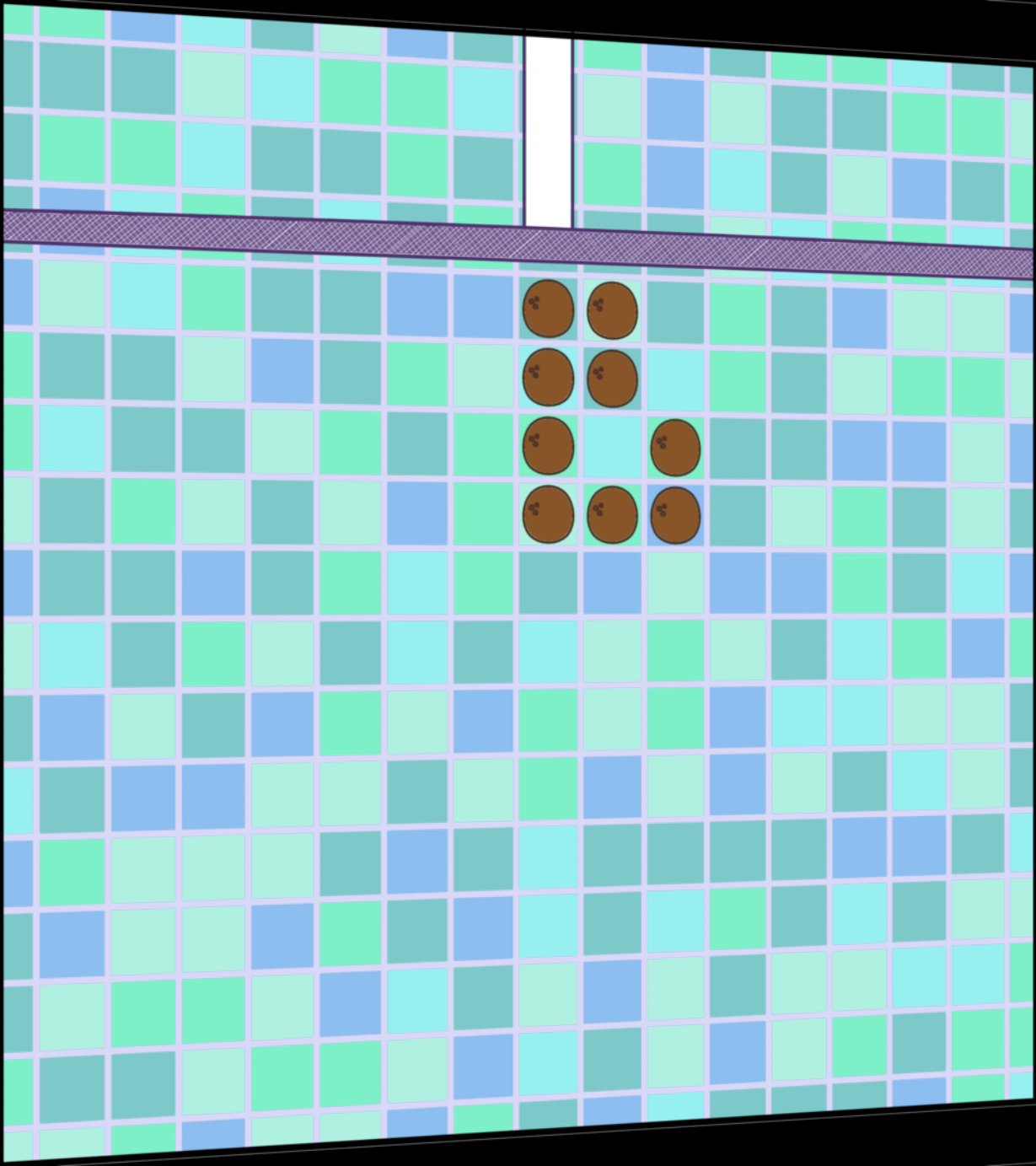
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

A diagram of a pool represented as a grid of colored squares (shades of blue and green). A purple barrier runs horizontally across the top. In the center of the grid, there are seven brown circular icons representing coconuts, arranged in two columns of four and one single icon to the right of the second row.

Let P be a set of n
coconuts in a pool.

We assume the coconuts are
aligned to a grid.

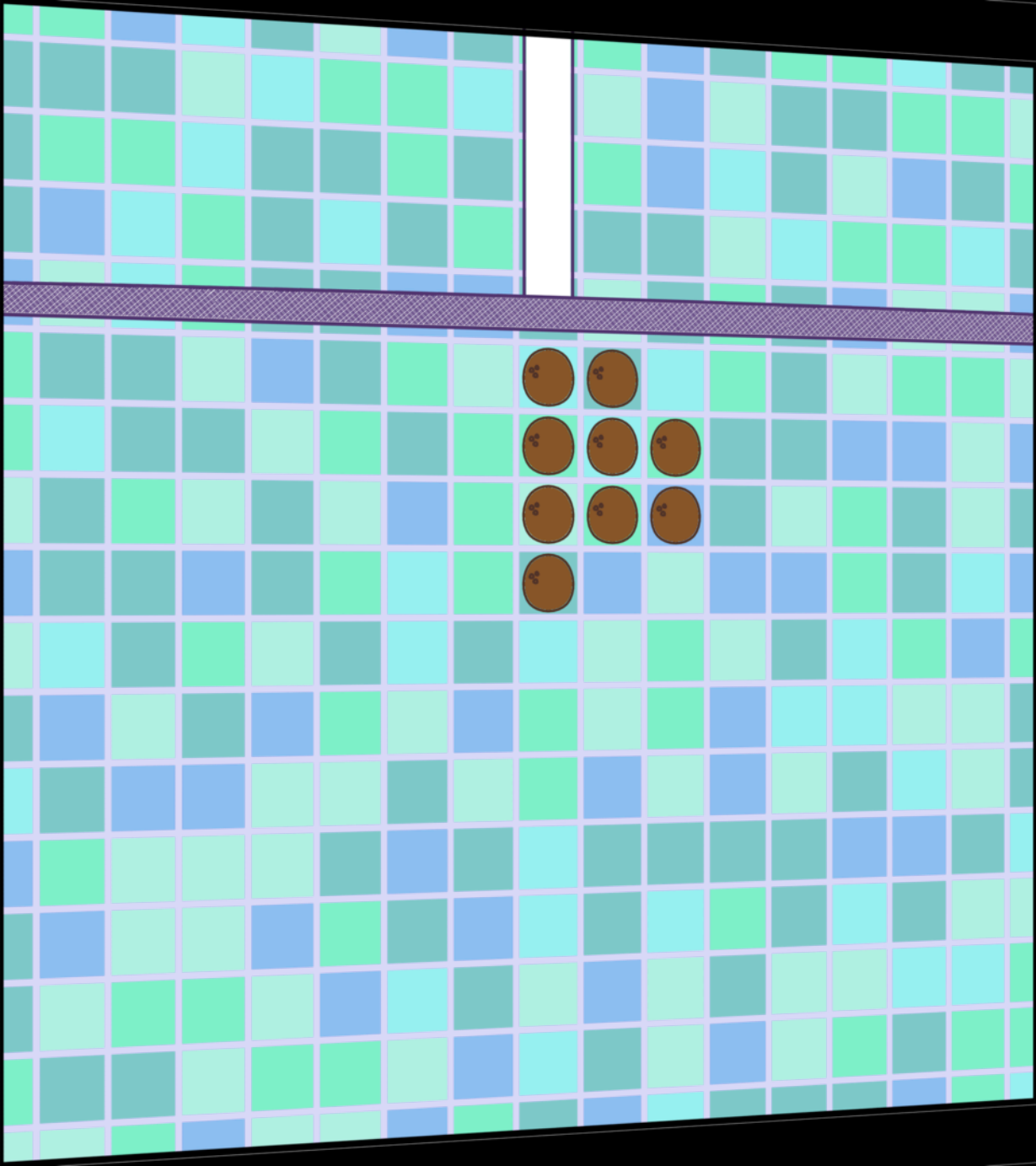
Assume further that we
have a giant *coconut*
pusher, which can be used
to push coconuts.

A 2D grid-based environment with a purple barrier at the top. The grid is composed of squares in various shades of blue and green. Seven brown circular icons representing coconuts are arranged on the grid. A purple oval highlights the text on the right.

Let P be a set of n
coconuts in a pool.

We assume the coconuts are
aligned to a grid.

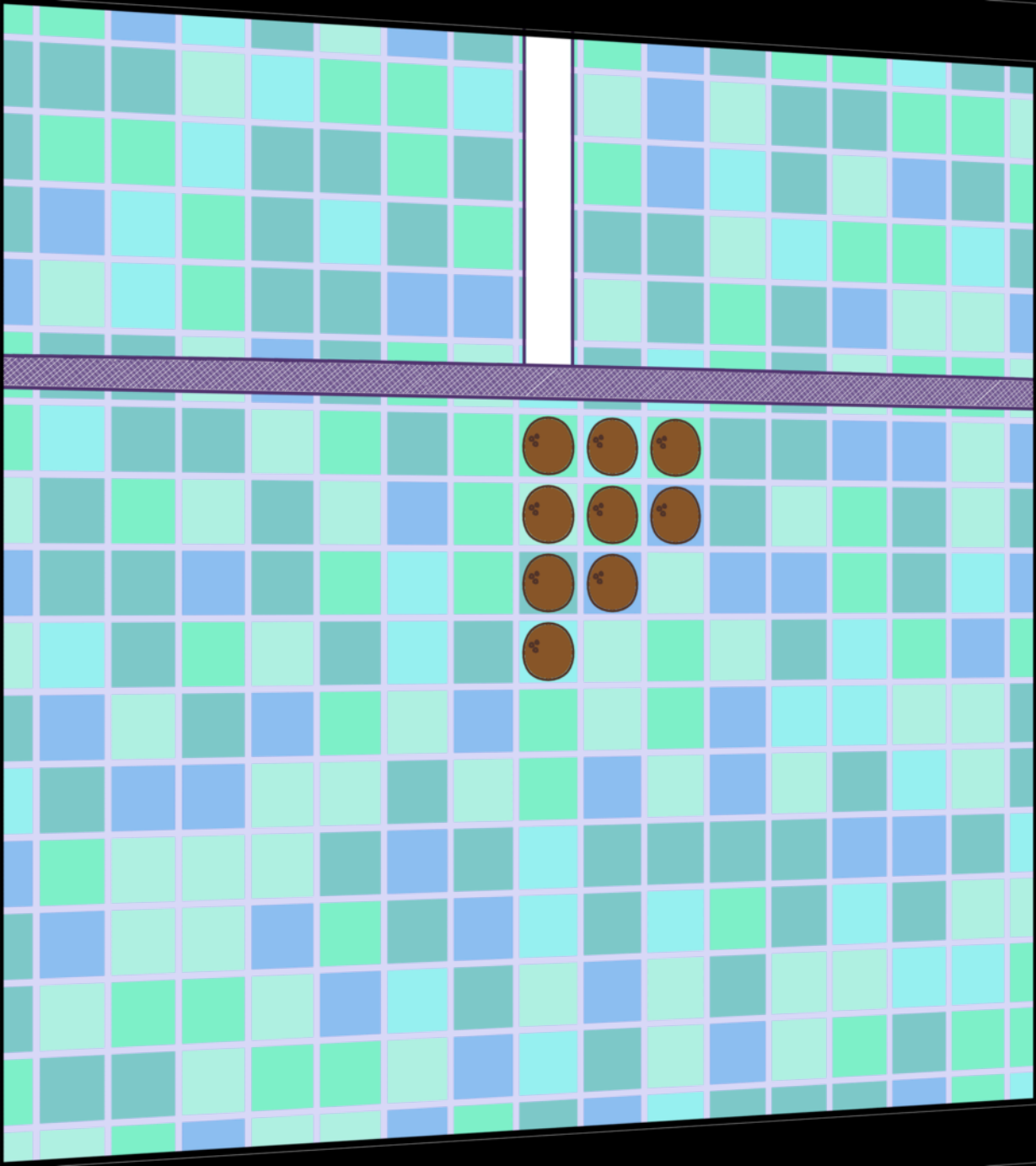
Assume further that we
have a giant *coconut*
pusher, which can be used
to push coconuts.



Let P be a set of n
coconuts in a pool.

~~We assume the~~ coconuts are
aligned to a grid.

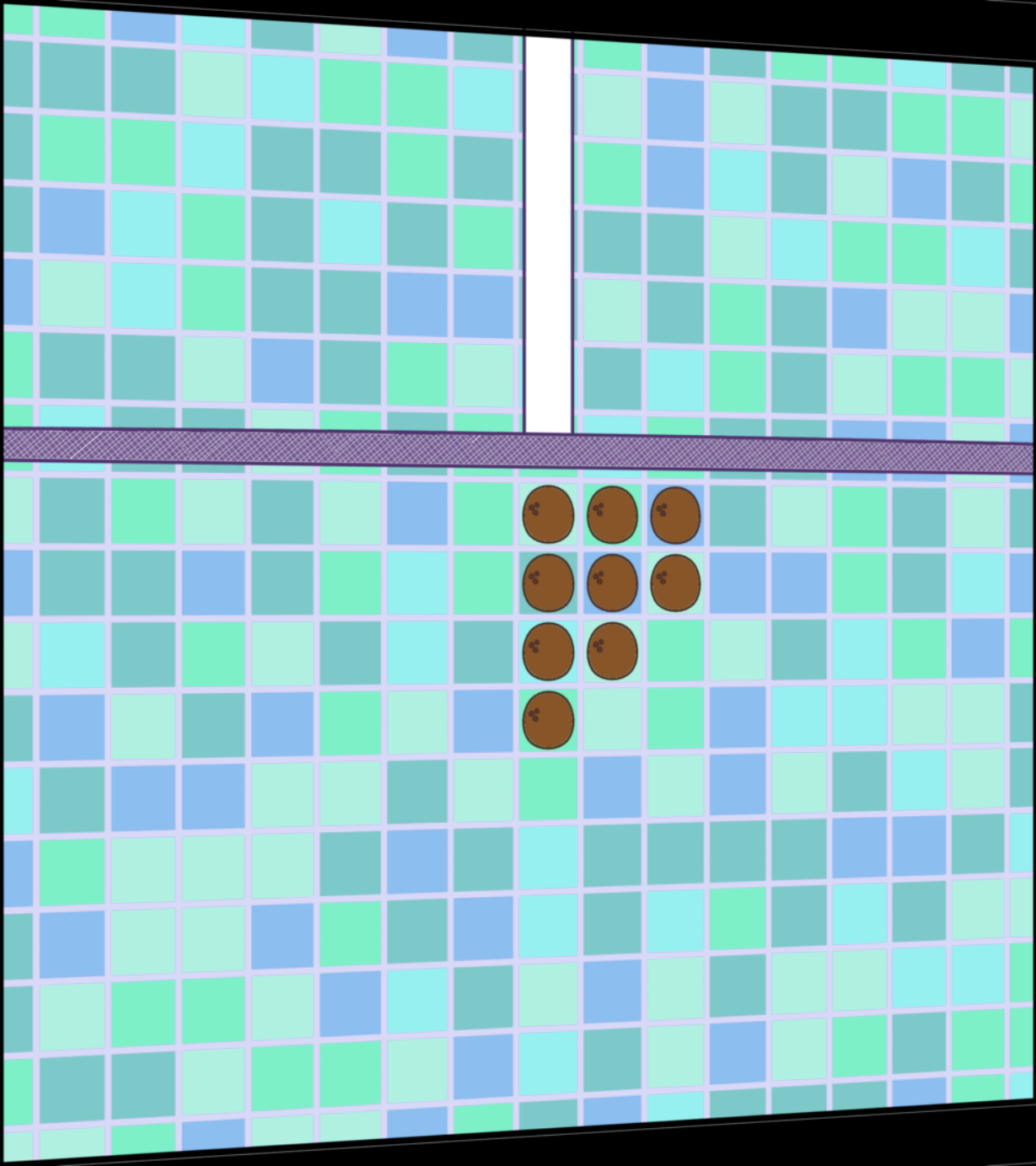
Assume further that we
have a giant *coconut*
pusher, which can be used
to push coconuts.

A large grid of blue and green squares covers the left side of the image. A horizontal purple bar with a textured pattern runs across the middle. On the left side of this bar, there are ten brown circles arranged in a grid: three in the top row, three in the second row, two in the third row, and one in the fourth row.

Let P be a set of n
coconuts in a pool.

We assume the coconuts are
aligned to a grid.

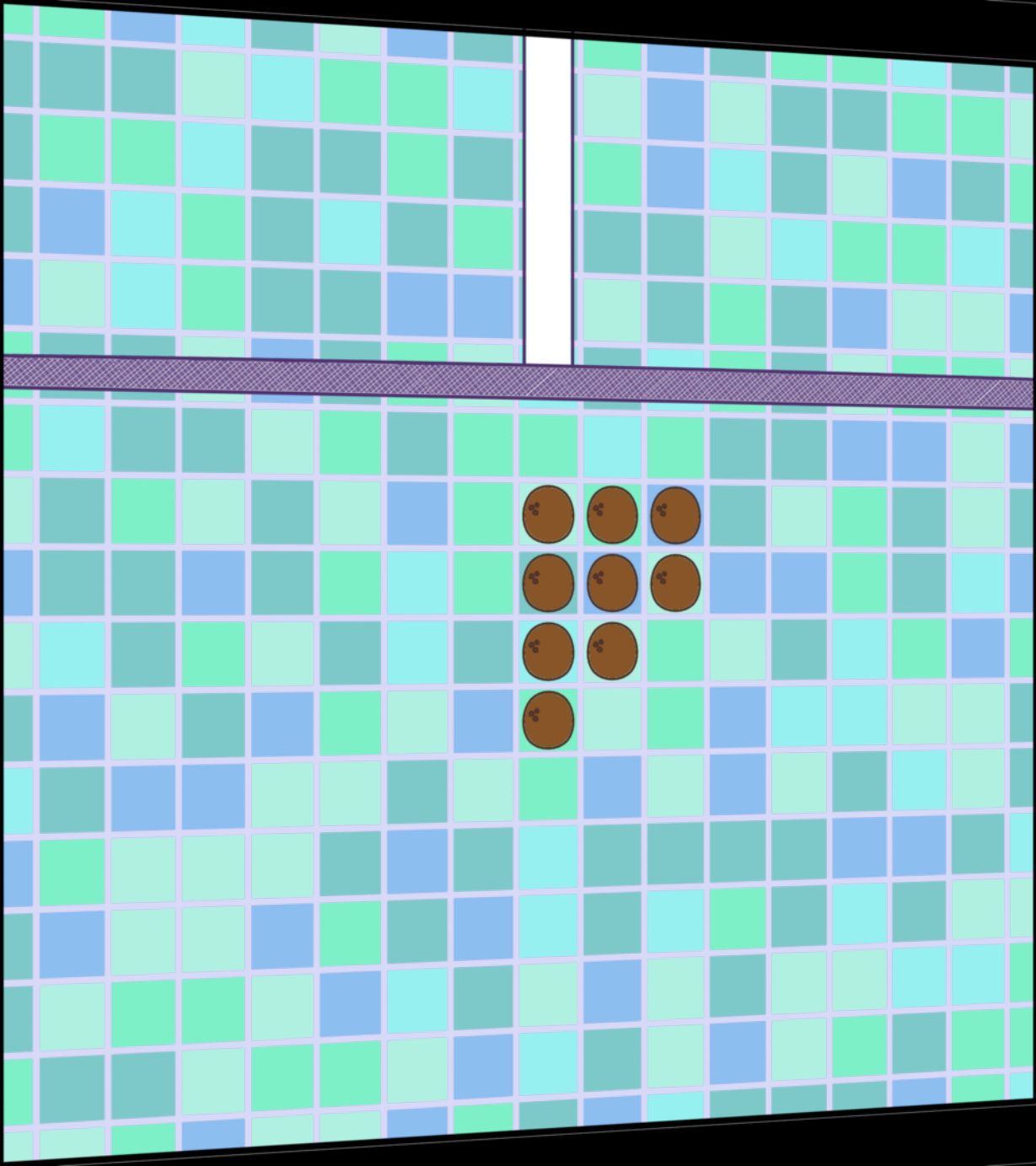
Assume further that we
have a giant *coconut*
pusher, which can be used
to push coconuts.



Let P be a set of n
coconuts in a pool.

We assume the coconuts are
aligned to a grid.

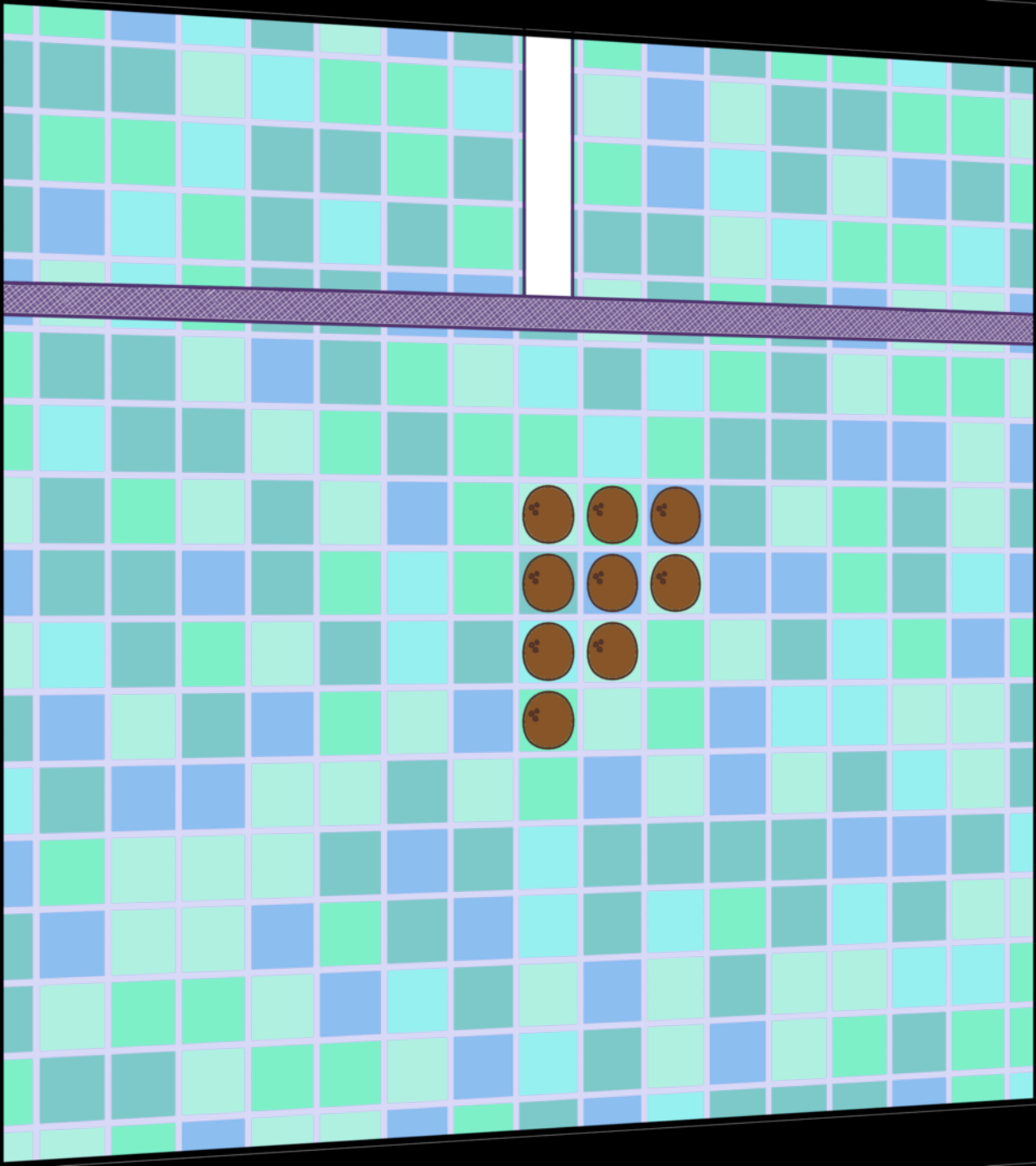
Assume further that we
have a giant *coconut
pusher*, which can be used
to push coconuts.

A grid of blue and green squares, with a purple bar across the middle. In the lower-left quadrant, there are seven brown circles arranged in a 3x3 grid with the bottom-right cell empty.

Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n
coconuts in a pool.

~~We assume the~~ coconuts are
aligned to a grid.

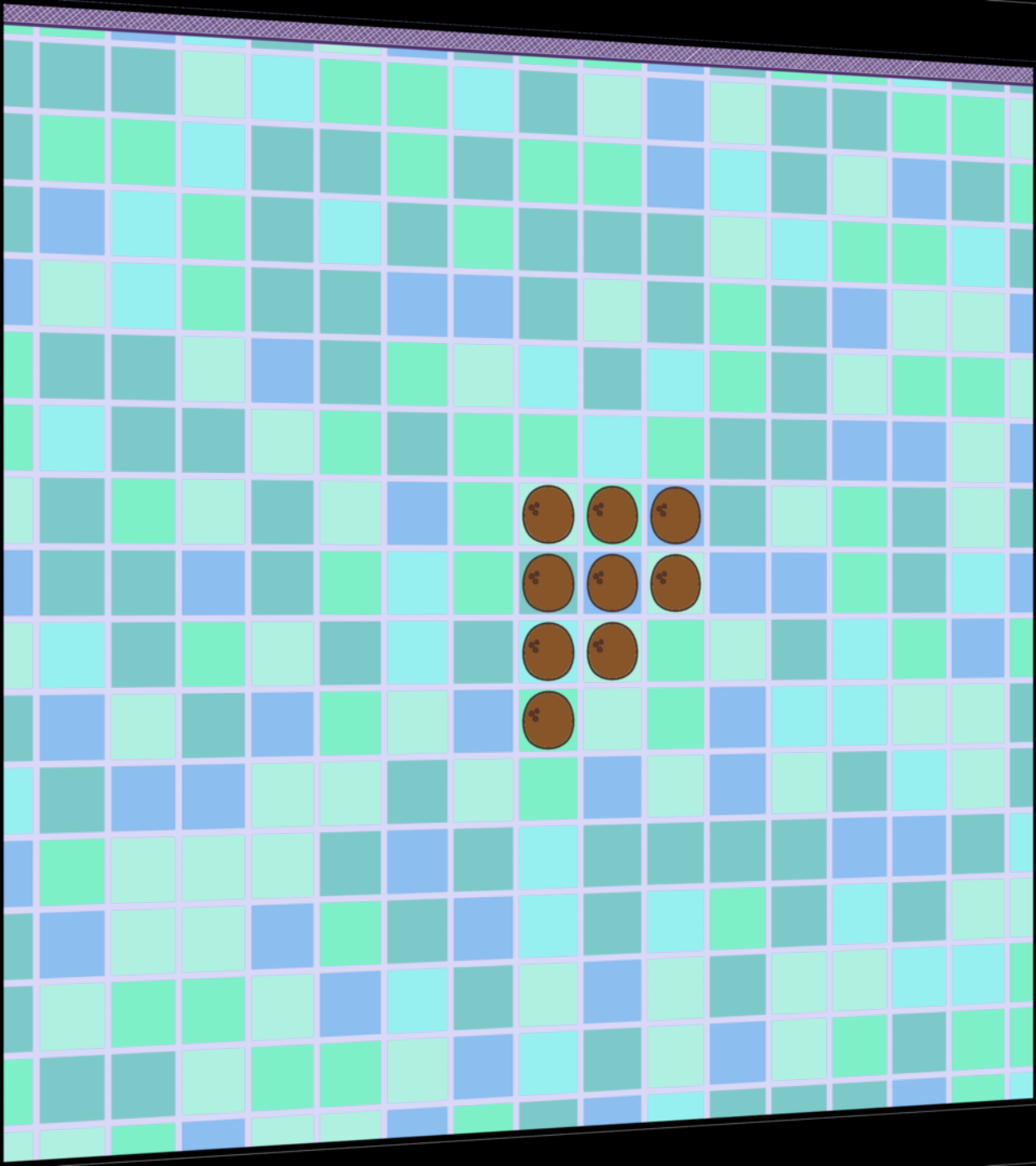
Assume further that we
have a giant *coconut*
pusher, which can be used
to push coconuts.

A diagram of a pool represented as a grid of colored squares in shades of blue and green. A purple horizontal bar is positioned near the top. In the lower-left area of the grid, there are 10 brown circles representing coconuts, arranged in a 4x3 grid with the last cell empty.

Let P be a set of n
coconuts in a pool.

We assume the coconuts are
aligned to a grid.

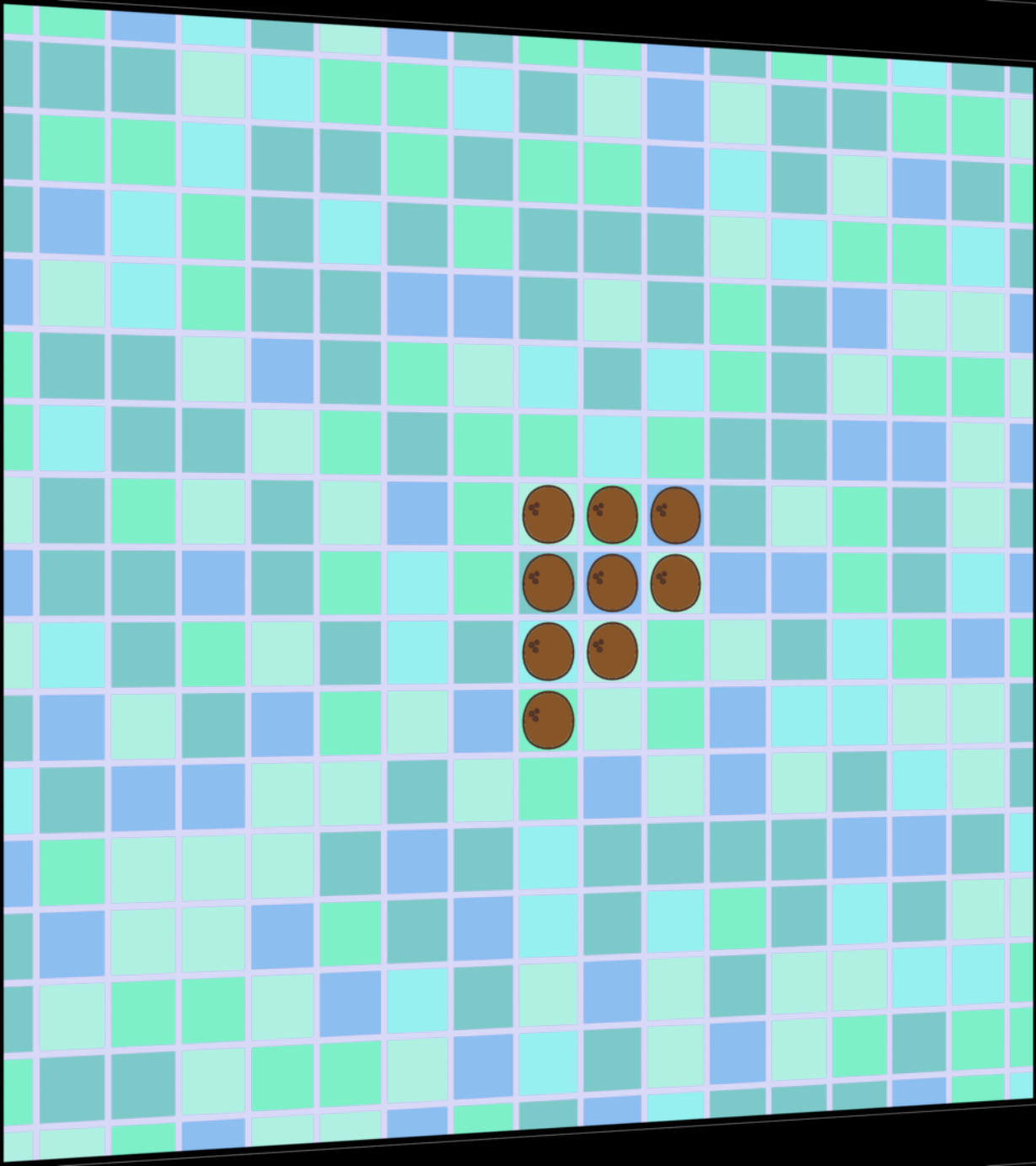
Assume further that we
have a giant *coconut*
pusher, which can be used
to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

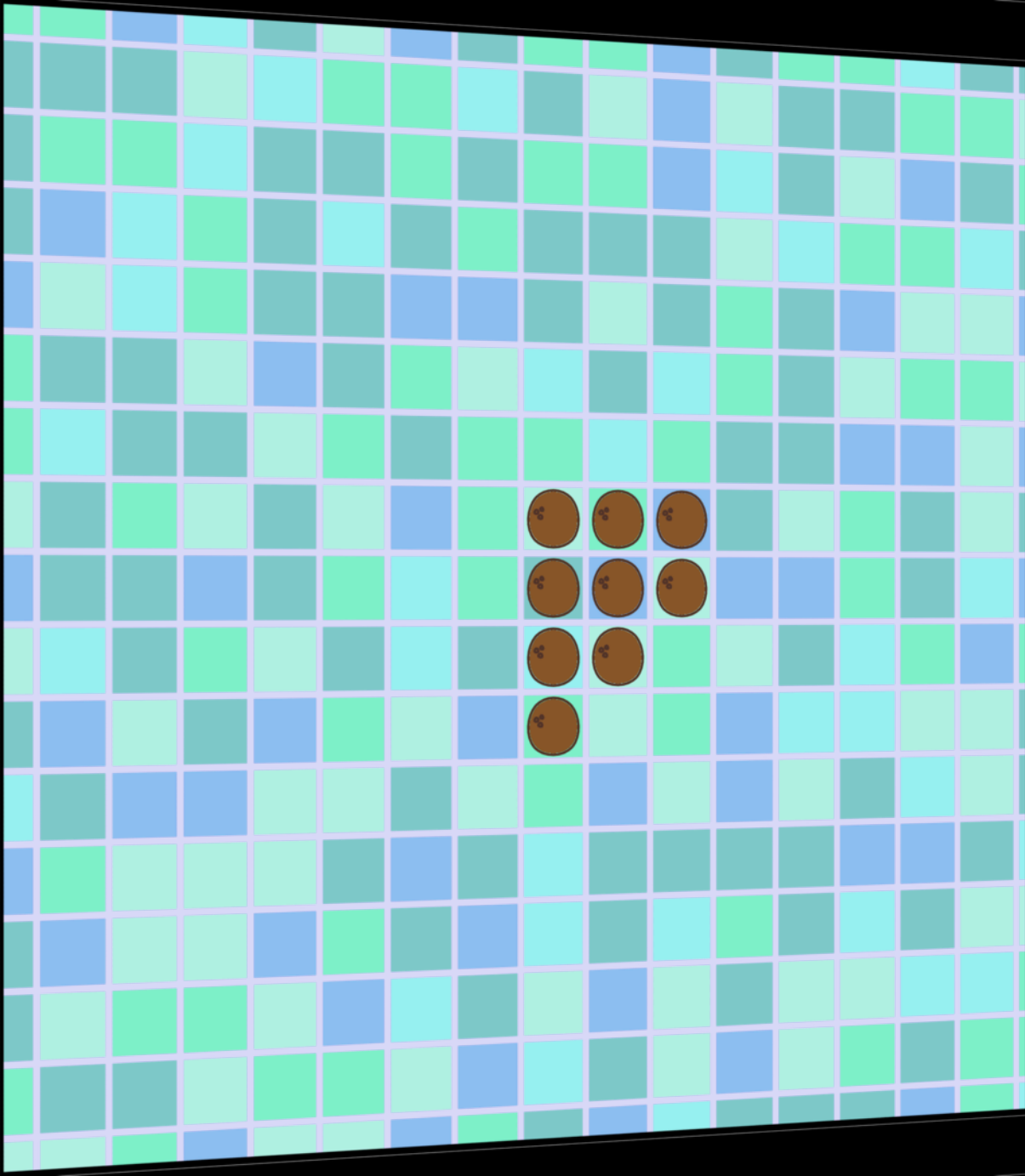
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

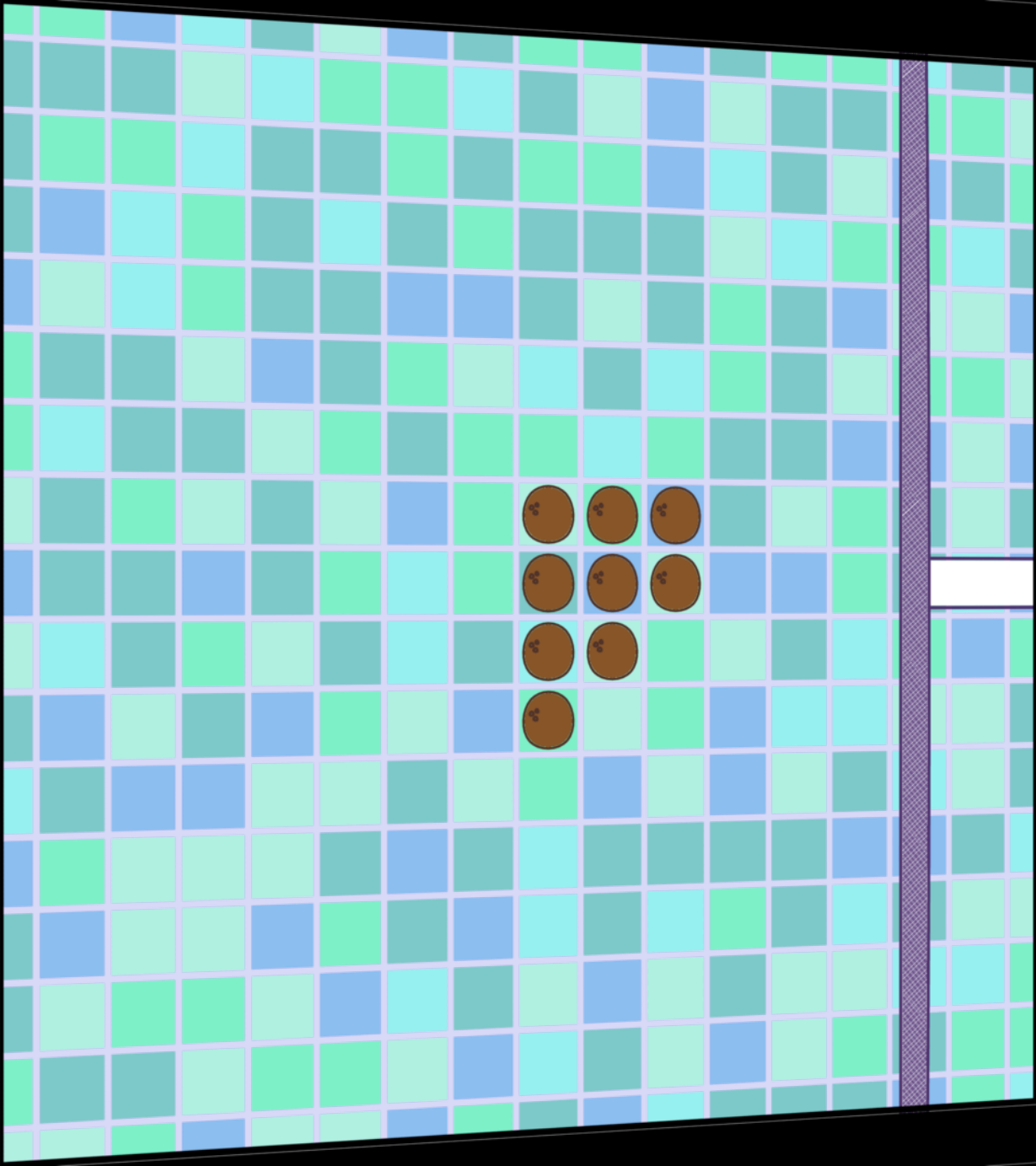


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

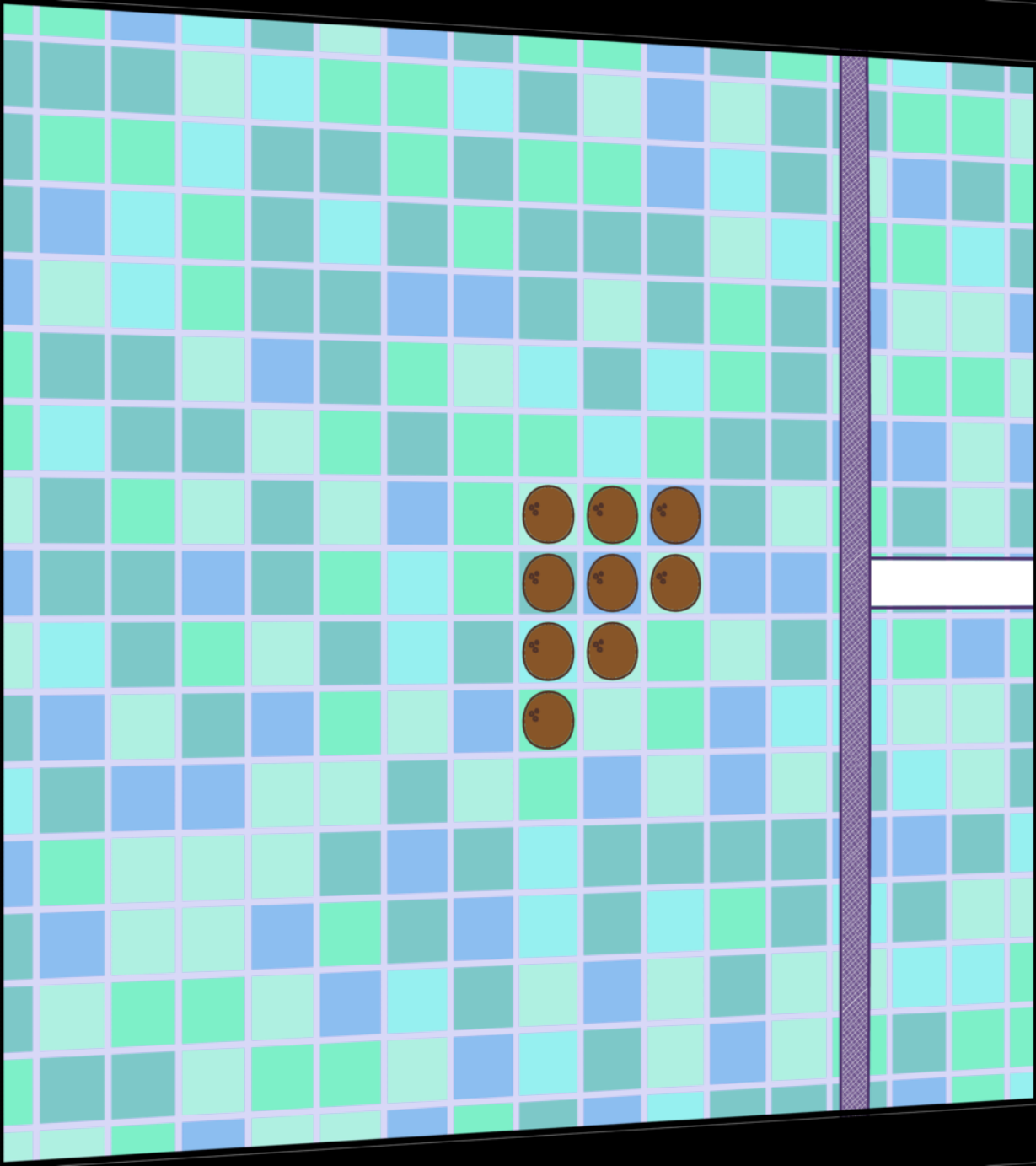


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

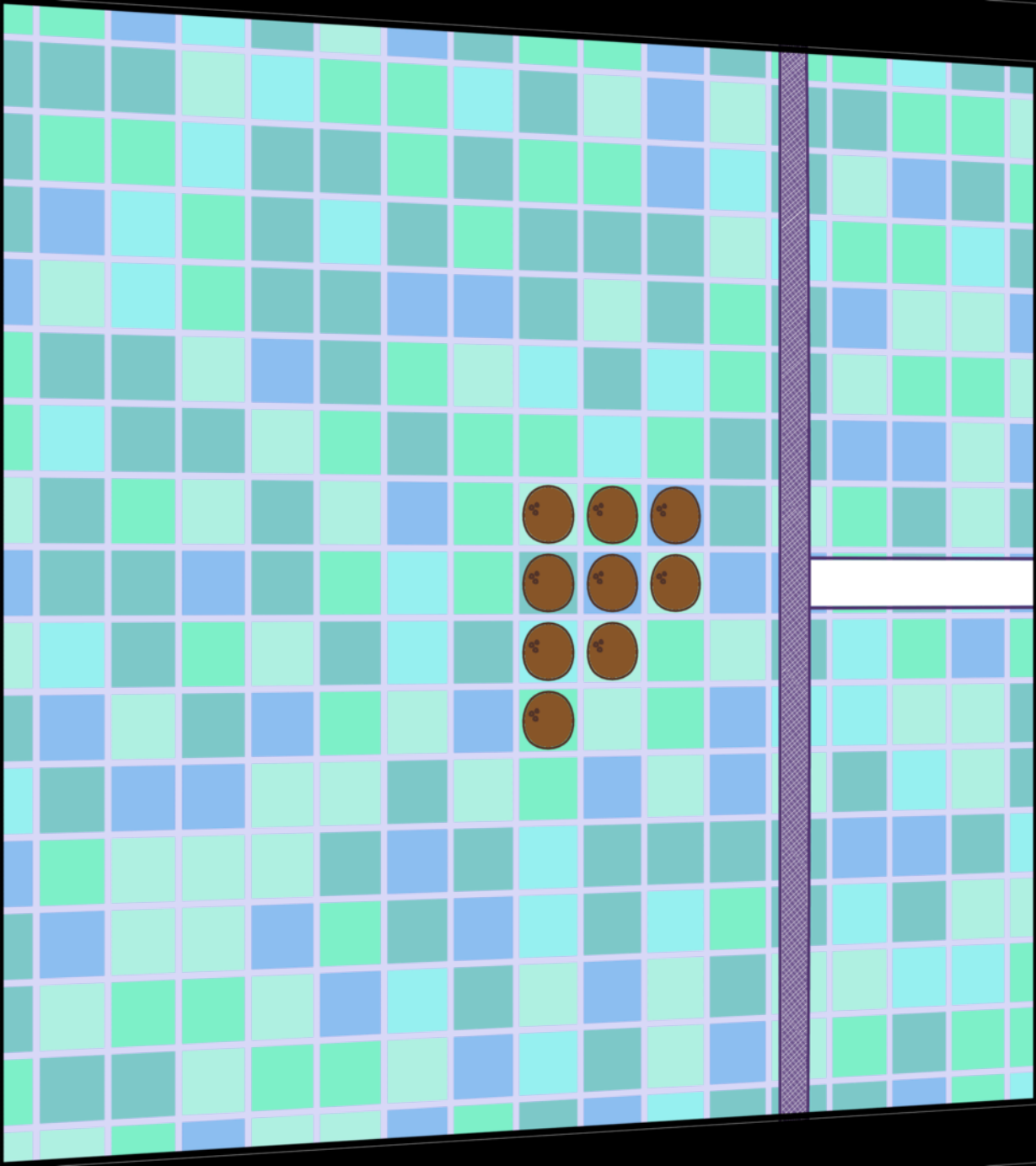


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

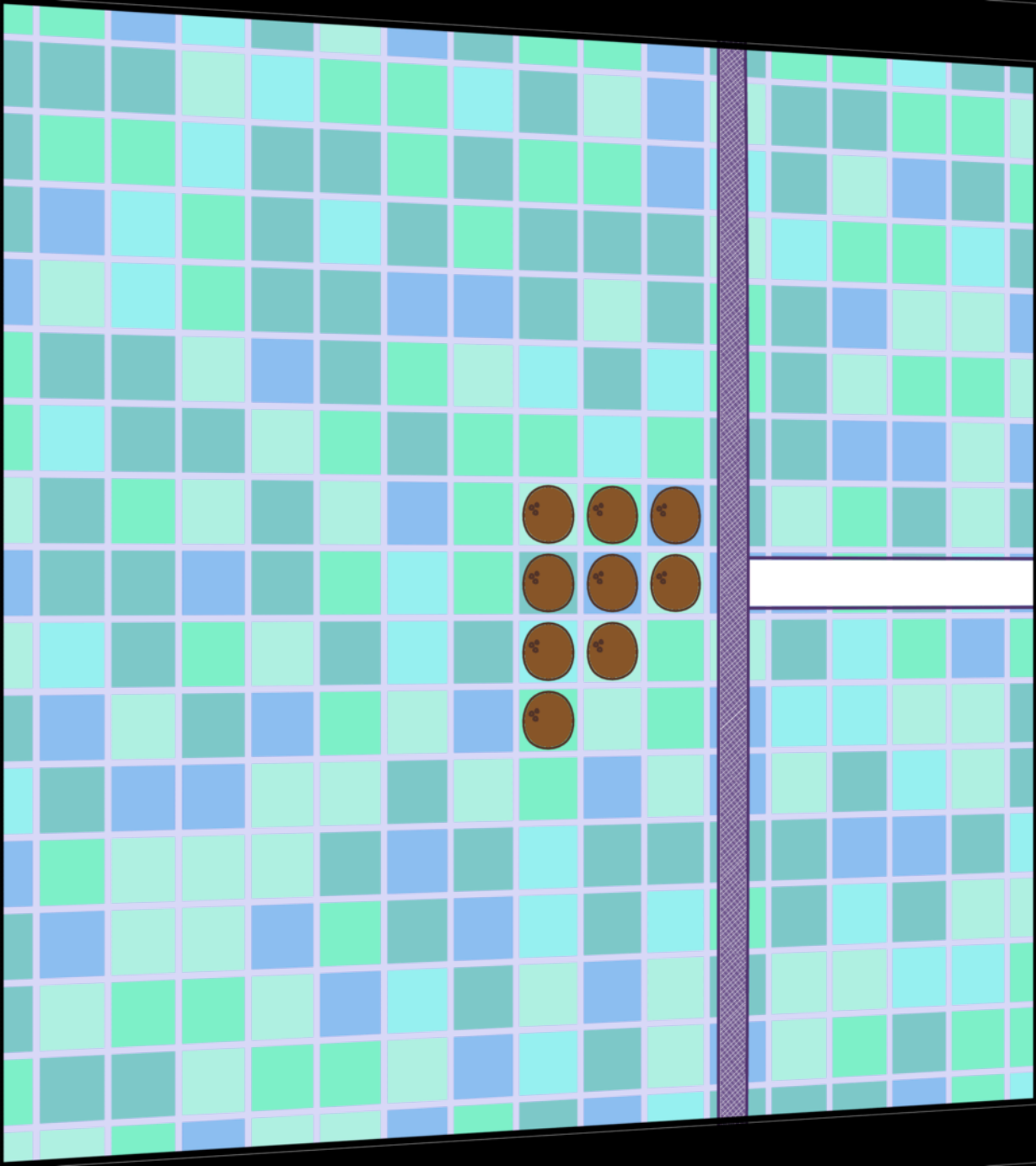


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

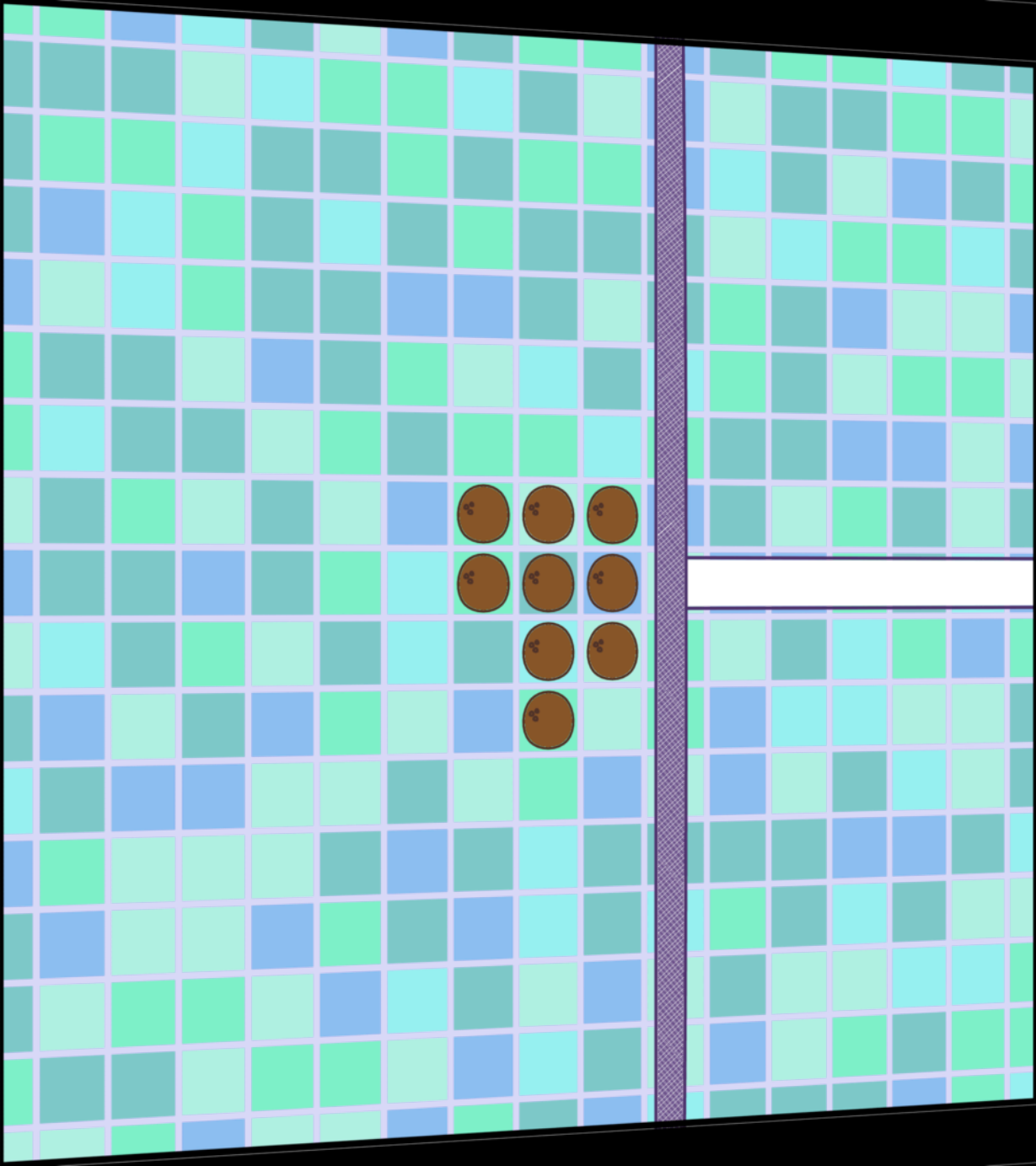


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

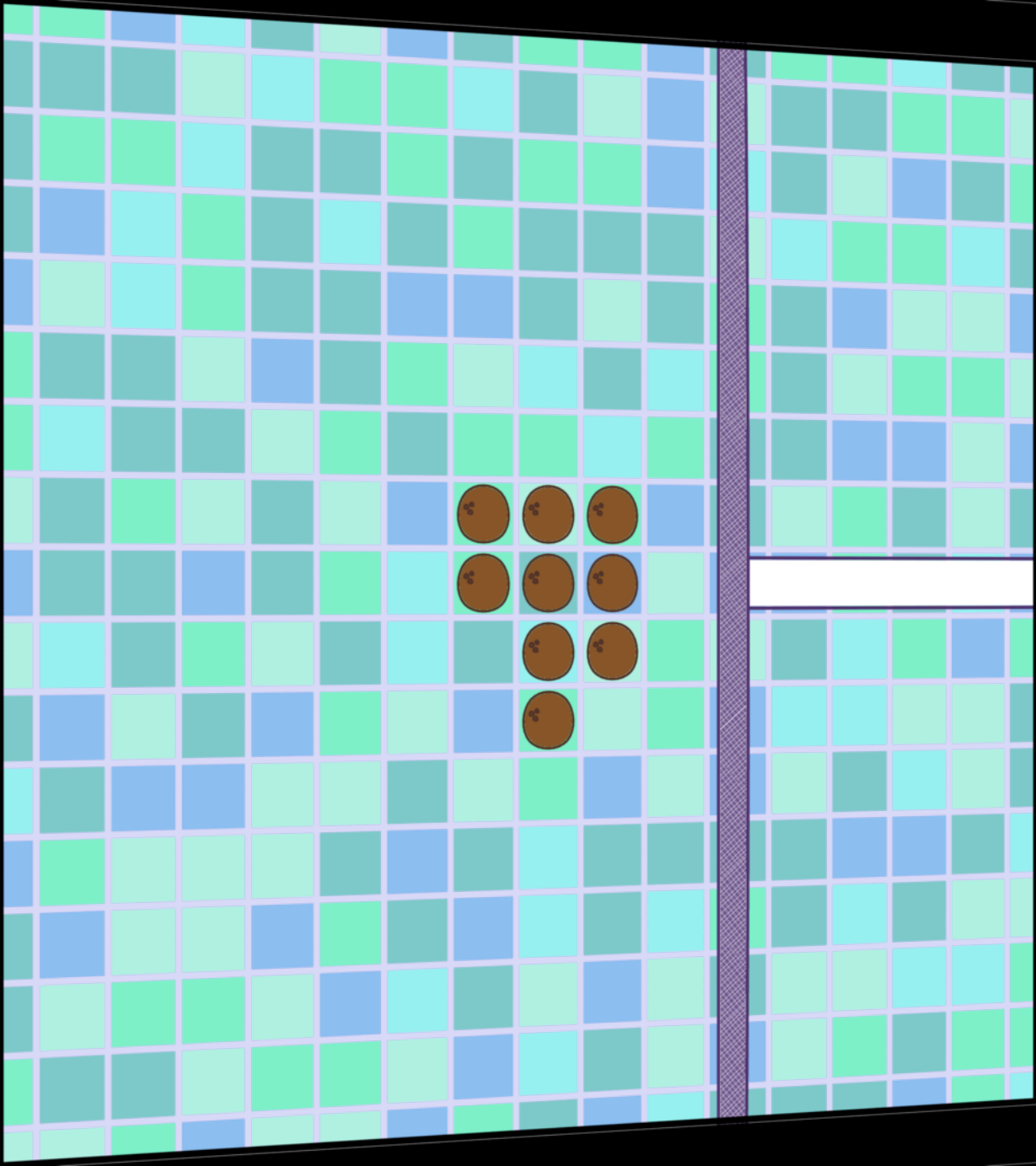
pusher, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

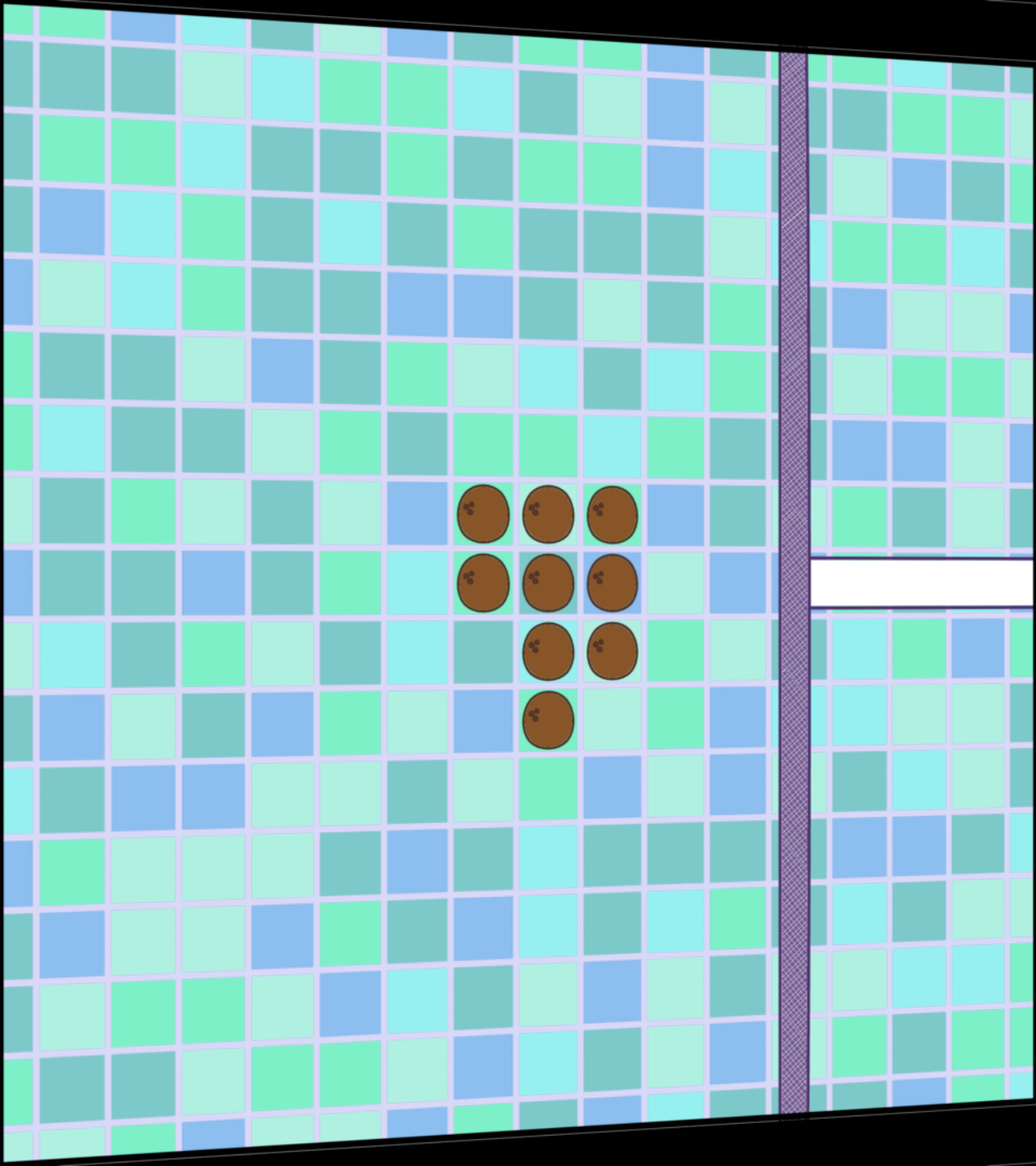


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

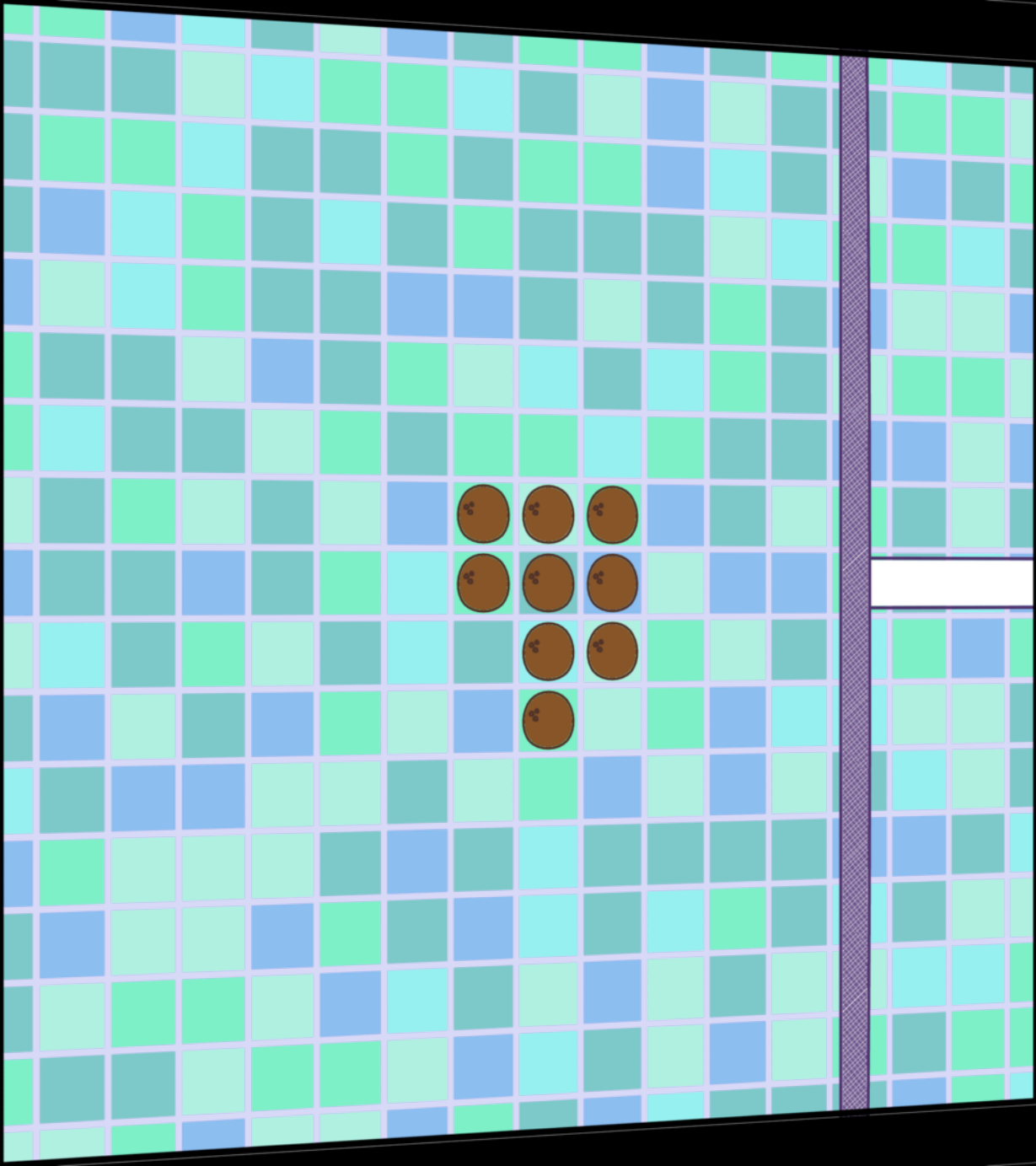


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

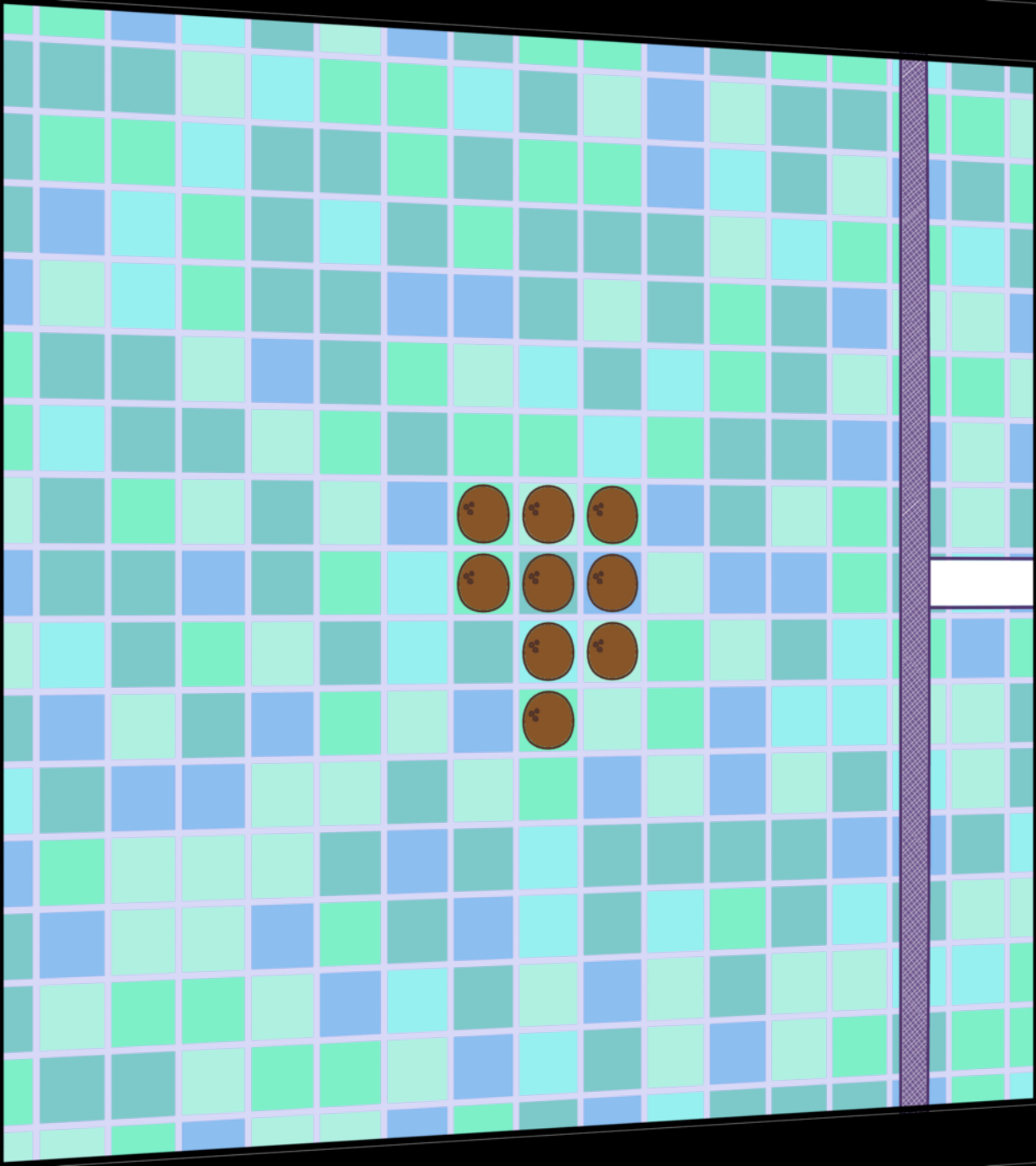


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

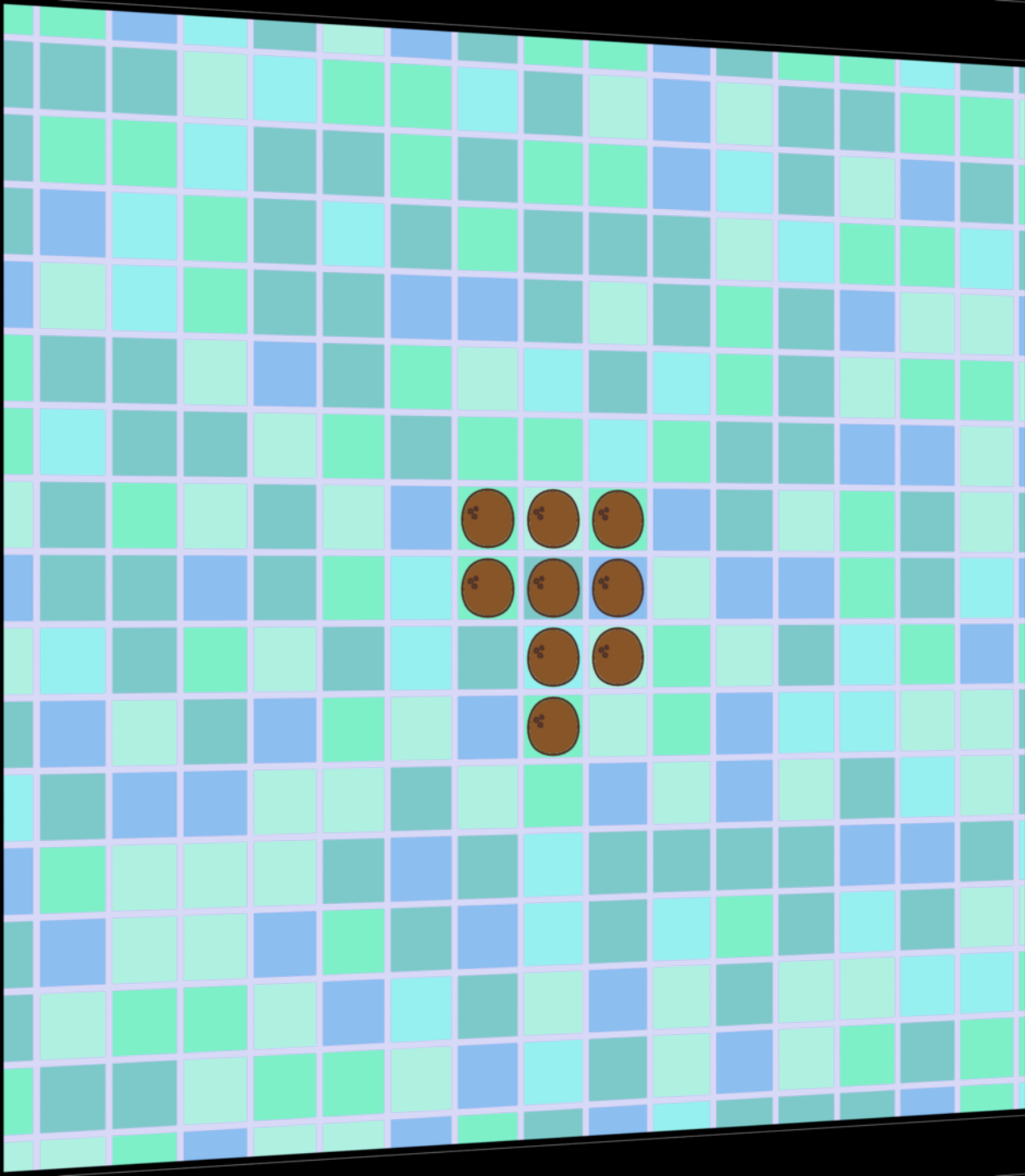


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

pusher, which can be used to push coconuts.

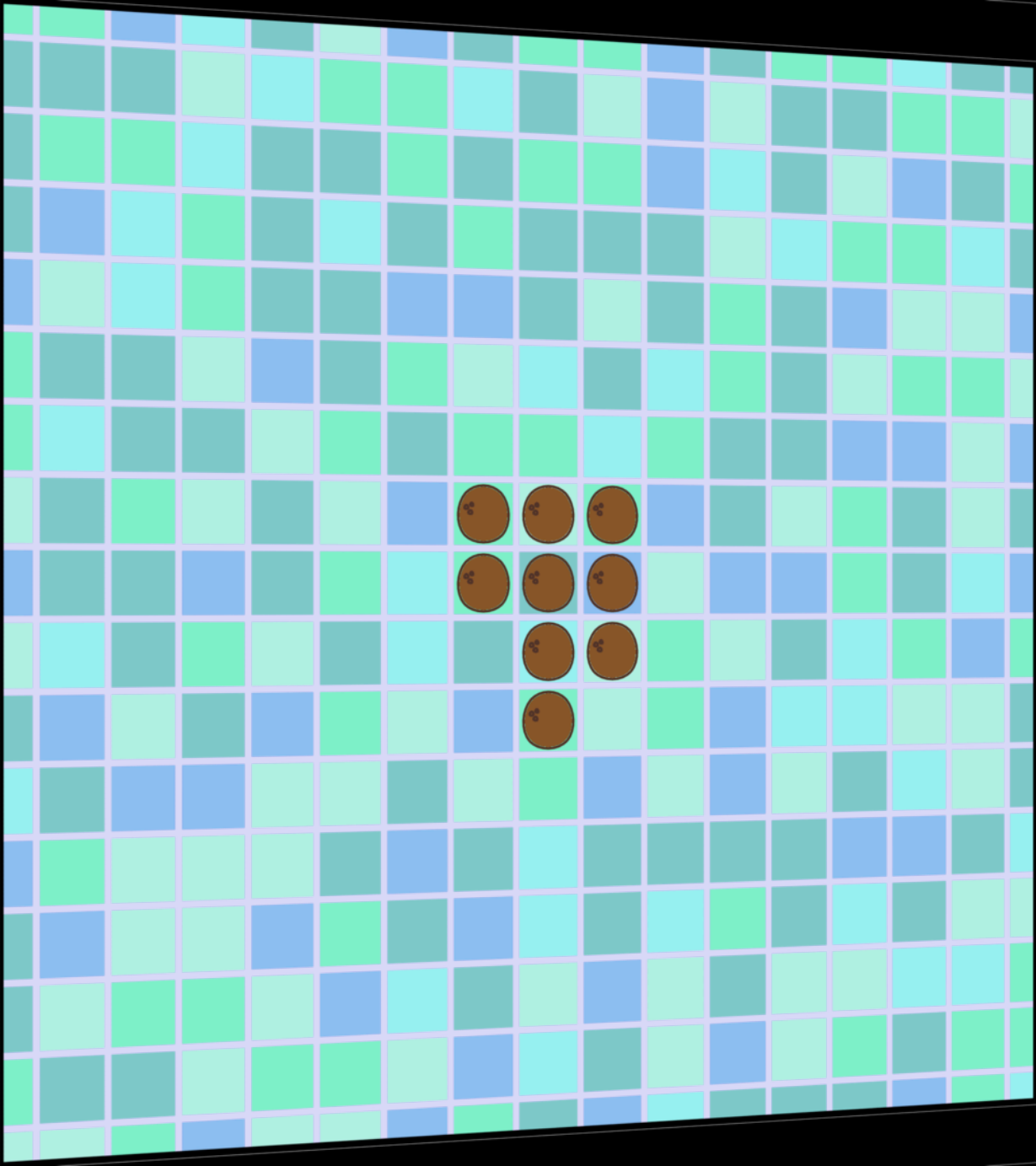


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut*

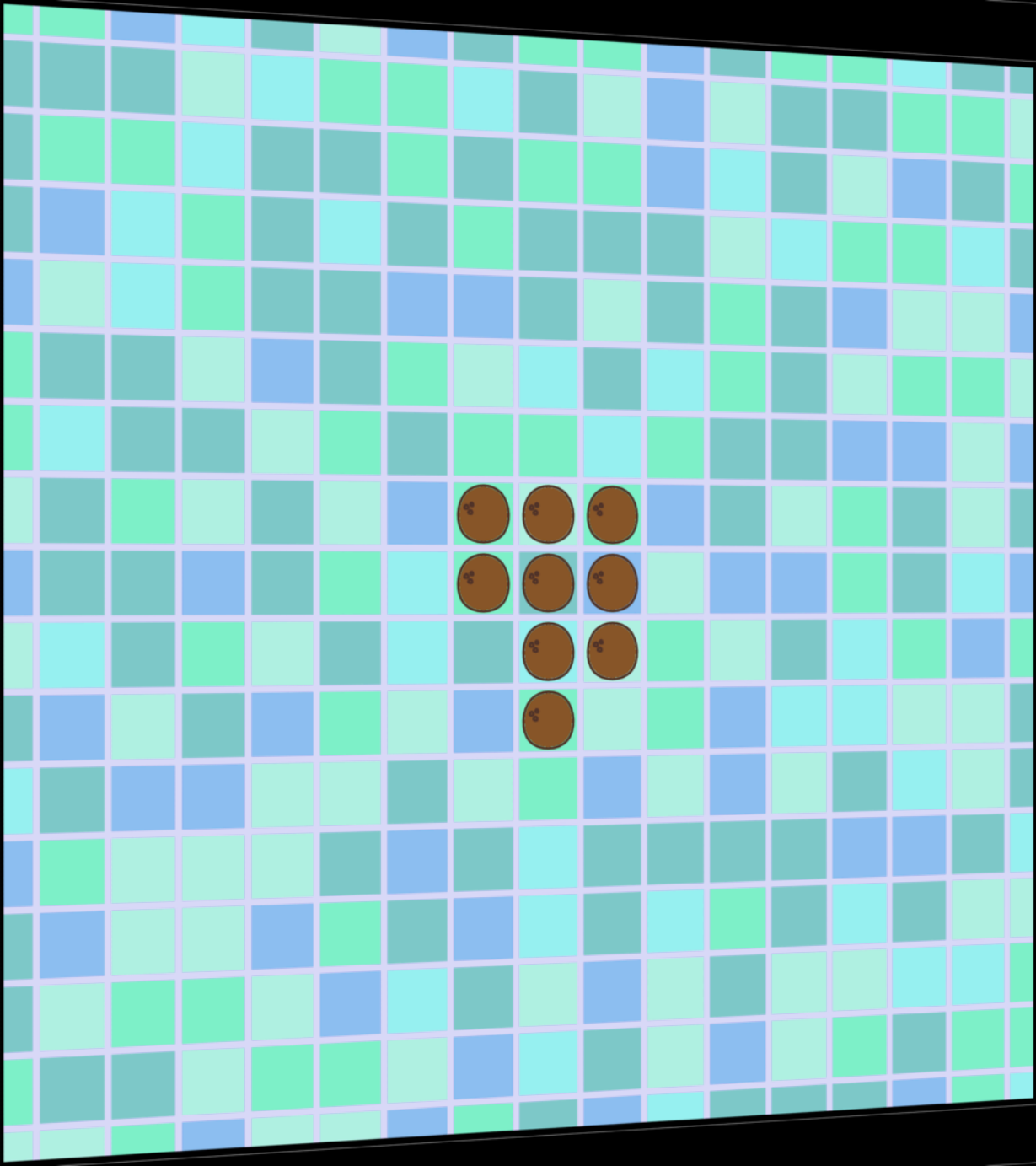
pusher, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

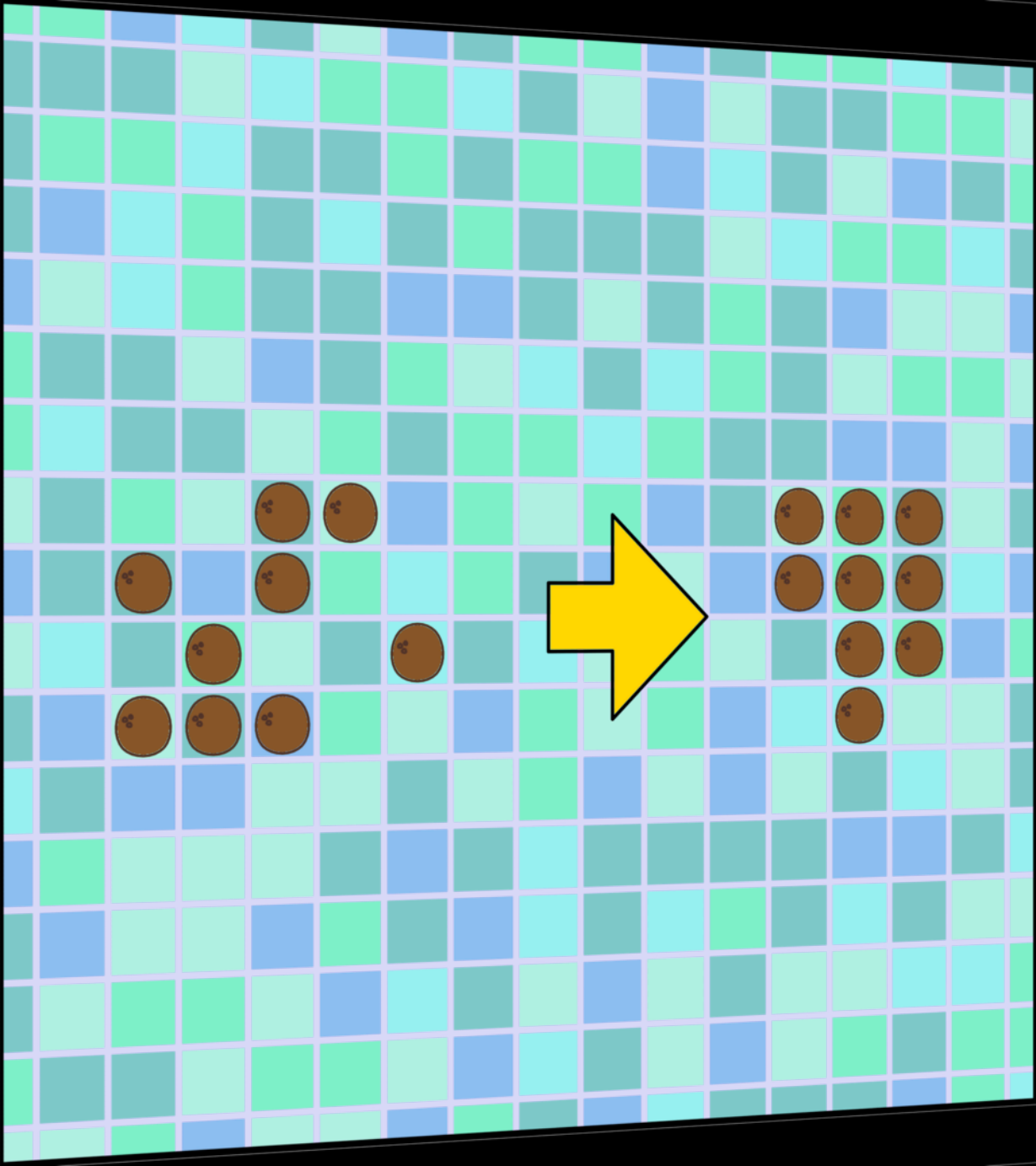


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

Any sequence of such coconut pushes leads to a *reconfiguration* of the coconuts.



Let P be a set of n coconuts in a pool.

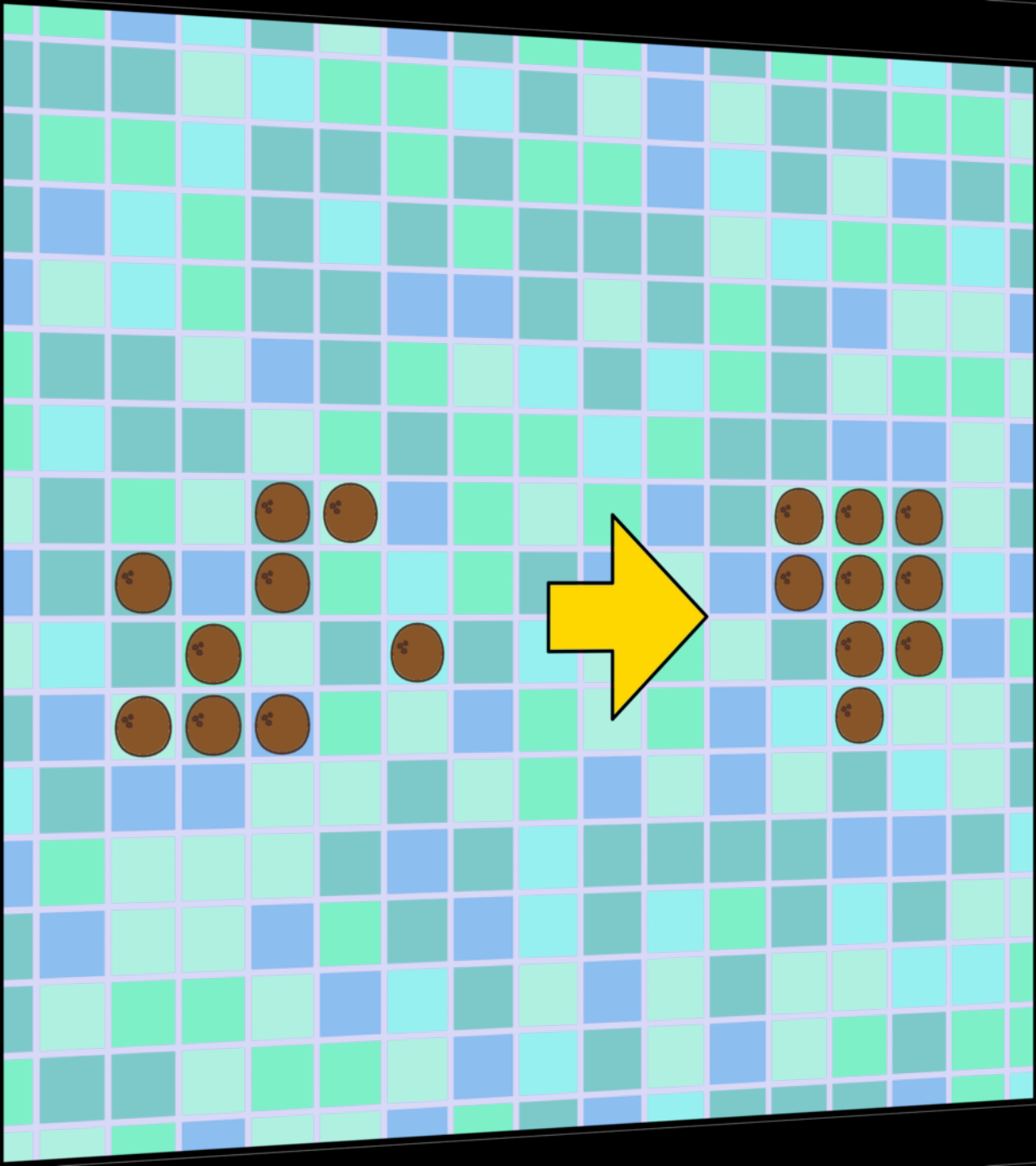
We assume the coconuts are aligned to a grid.

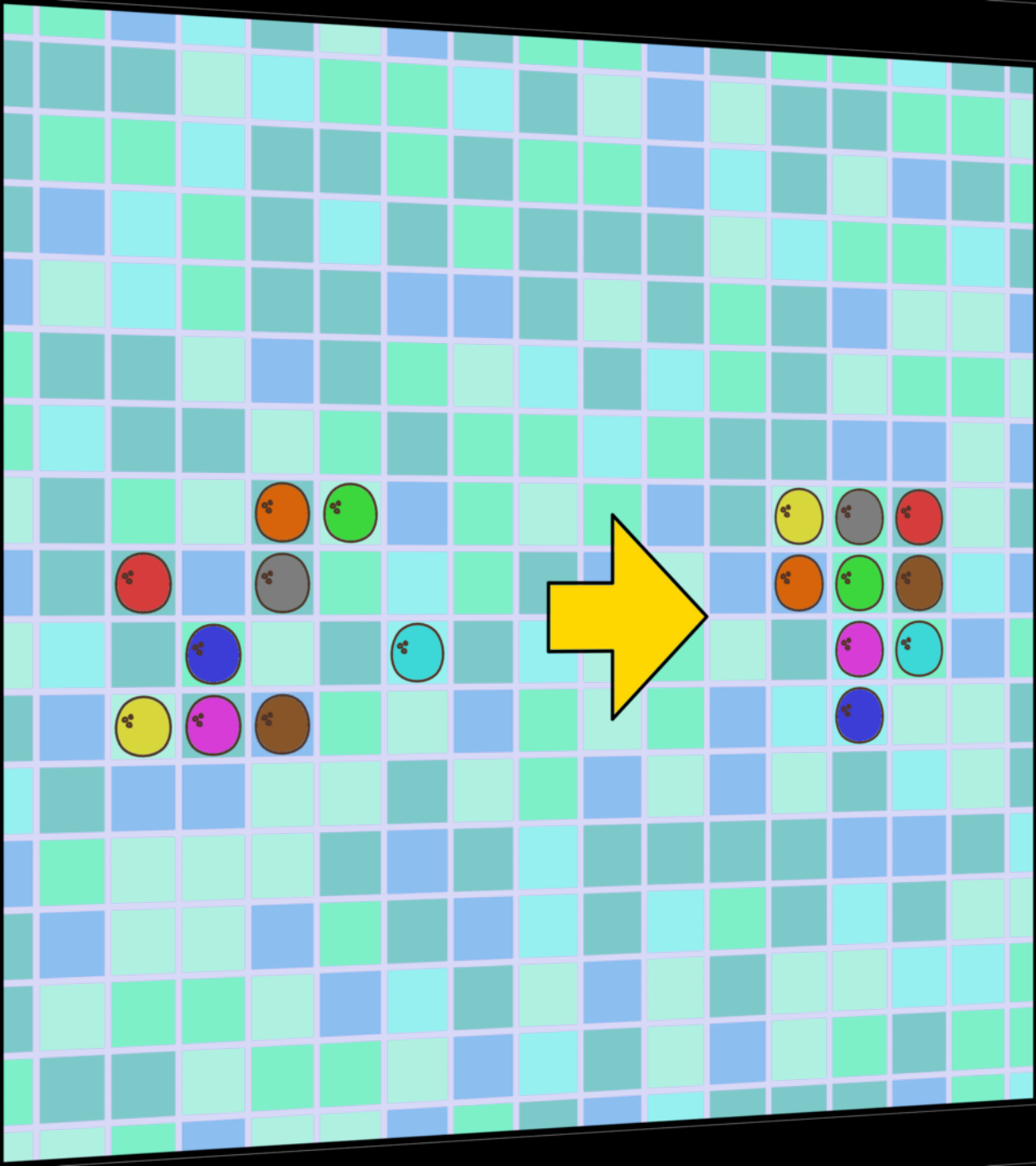
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

Any sequence of such coconut pushes leads to a *reconfiguration* of the coconuts.

Problem 1. (*unlabeled
coconut reconfiguration*)

Given a starting
configuration P and a
goal configuration P' , is
there a sequence of pushes
that transforms P into P' ?



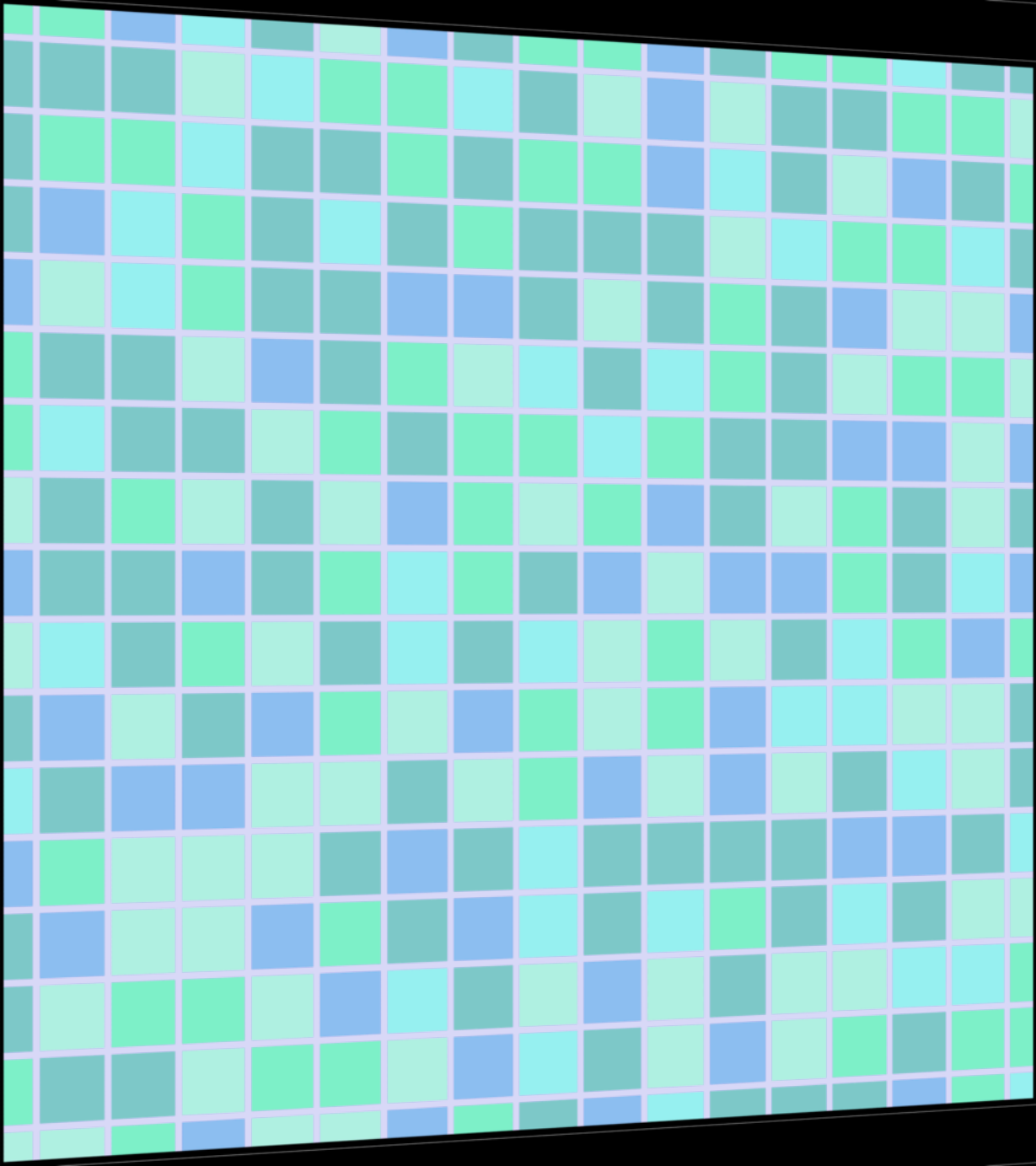


Problem 1. (*unlabeled coconut reconfiguration*)

Given a starting configuration P and a goal configuration P' , is there a sequence of pushes that transforms P into P' ?

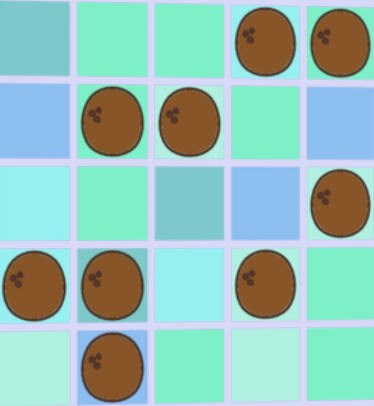
Problem 2. (*labeled coconut reconfiguration*)

Same thing, with labels.

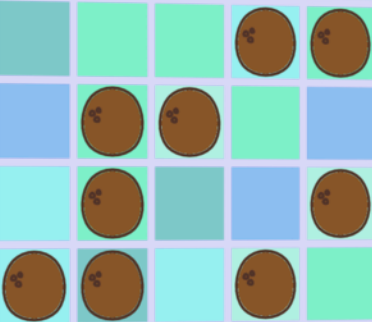


*Can we always push
our coconuts into tidy
rectangles?*

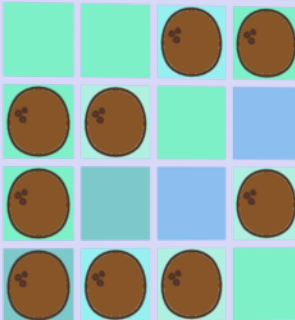
*Can we always push
our coconuts into tidy
rectangles?*



Can we always push
our coconuts into tidy
rectangles?



*Can we always push
our coconuts into tidy
rectangles?*



Can we always push
our coconuts into tidy
rectangles?



*Can we always push
our coconuts into tidy
rectangles?*



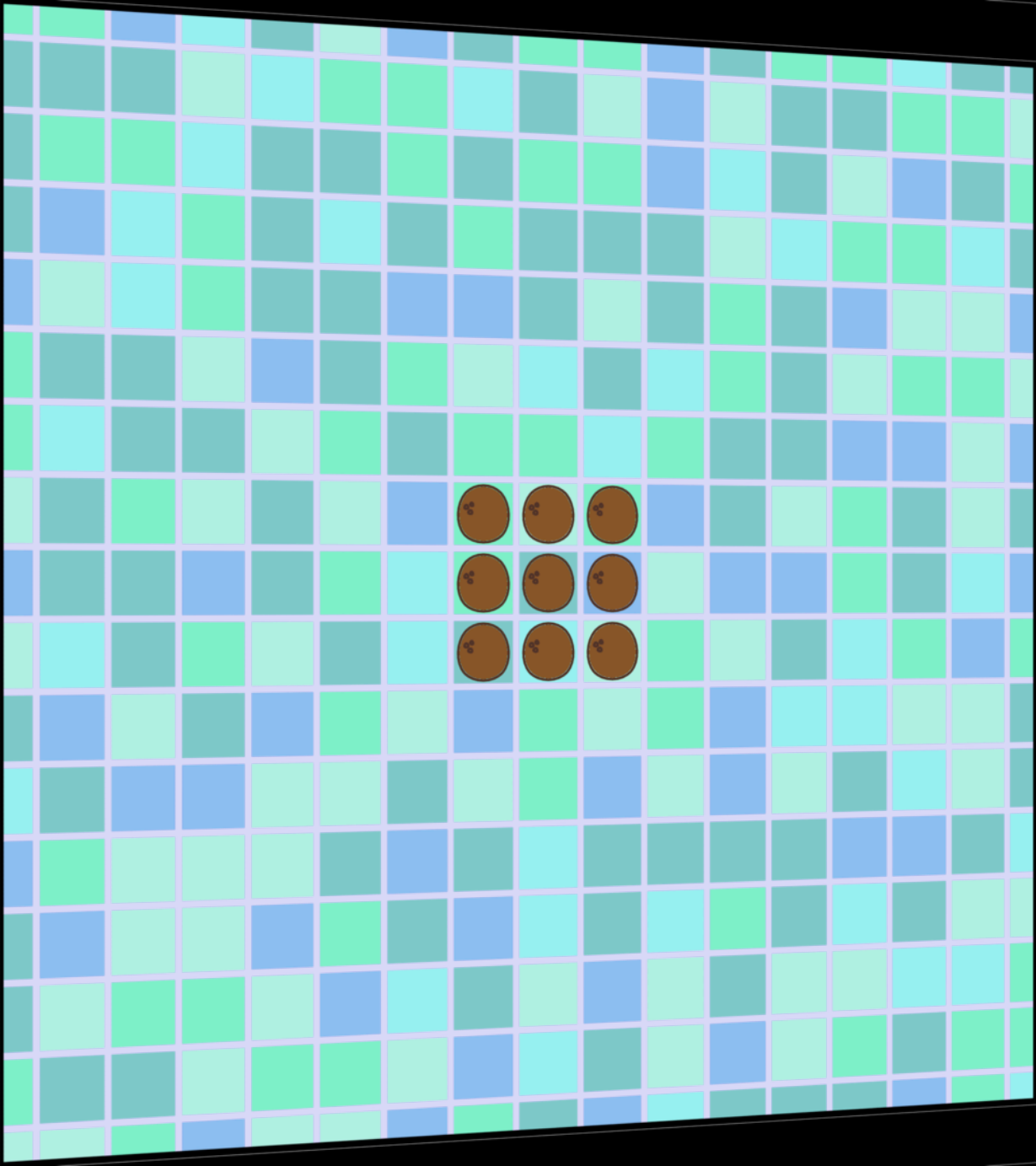
*Can we always push
our coconuts into tidy
rectangles?*



Can we always push
our coconuts into tidy
rectangles?

Obviously, n coconuts can
only be pushed into a $a \times b$
rectangle if $ab = n!$

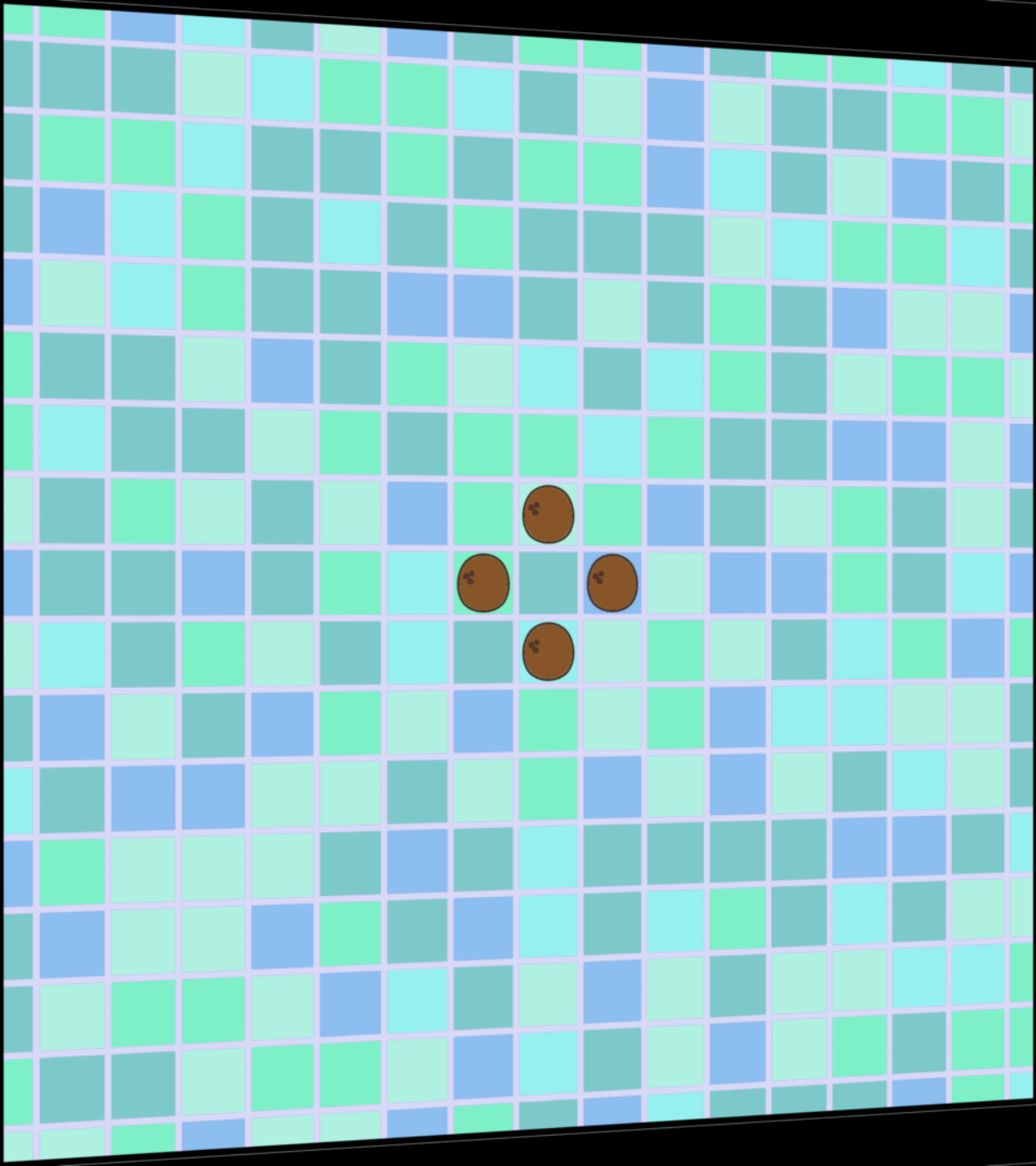


A large grid of colorful squares in shades of blue, green, and cyan, arranged in a pattern that resembles a distorted grid. In the center of the grid, there is a 3x3 arrangement of brown circles, each with a small white 'a' on it, representing coconuts.

Can we always push
our coconuts into tidy
rectangles?

Obviously, n coconuts can
only be pushed into a $a \times b$
rectangle if $ab = n!$

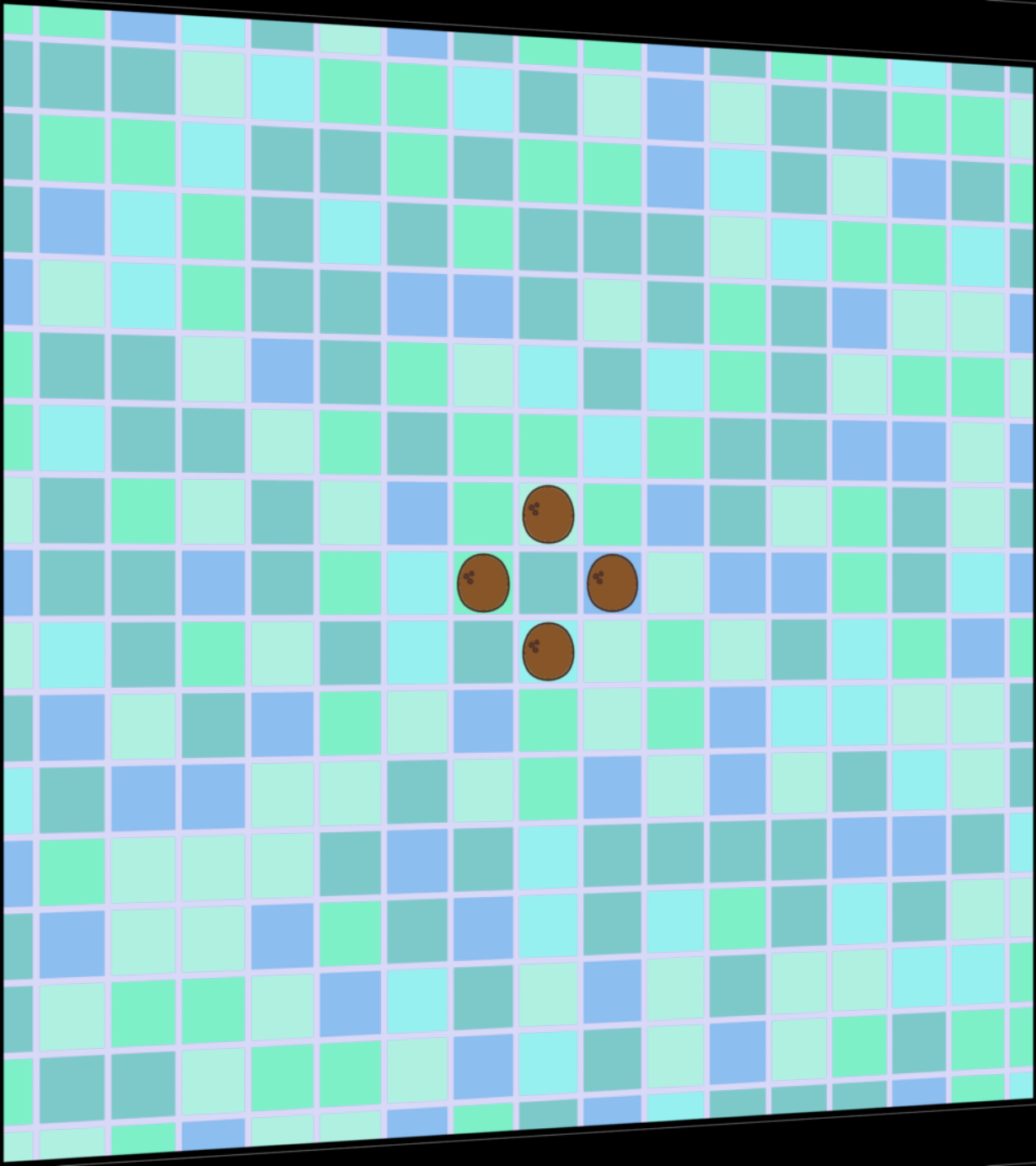
But is it always possible?

A large grid of colored squares in shades of blue, green, and cyan, arranged in a pattern that suggests a tiling or packing problem. Four brown circles, representing coconuts, are placed on the grid. They are located at approximately (row, column) coordinates (4, 5), (5, 4), (5, 6), and (6, 5) if the top-left square is (1,1).

Can we always push
our coconuts into tidy
rectangles?

Obviously, n coconuts can
only be pushed into a $a \times b$
rectangle if $ab = n!$

But is it always possible?

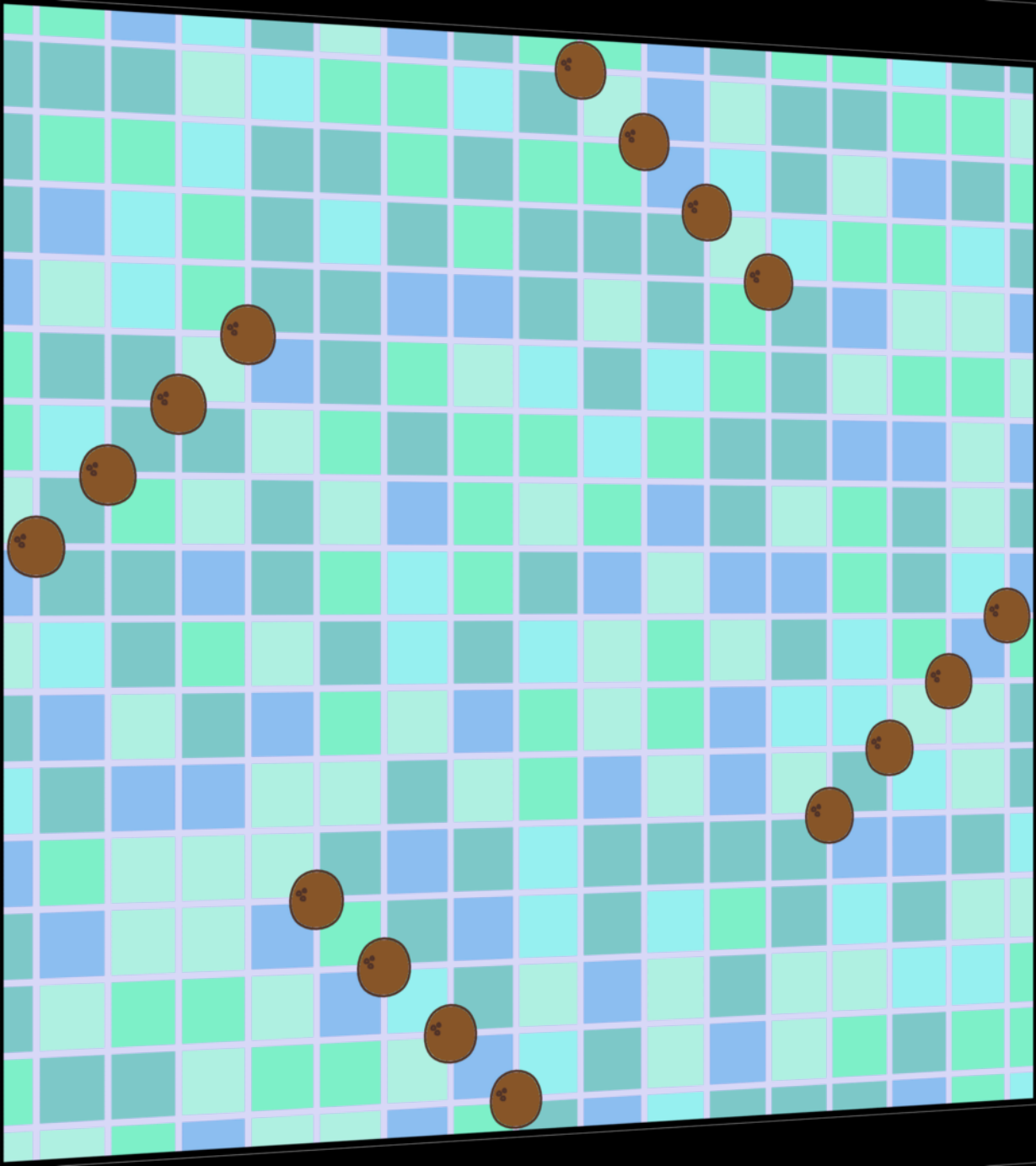
A large grid of colored squares in shades of blue, green, and cyan, arranged in a pattern that suggests a 2D coordinate system. Four brown circles, representing coconuts, are placed on the grid. They are located at approximately (row, column) coordinates (1, 1), (1, 2), (2, 1), and (2, 2) relative to a local 2x2 sub-grid.

Can we always push
our coconuts into tidy
rectangles?

Obviously, n coconuts can
only be pushed into a $a \times b$
rectangle if $ab = n!$

But is it always possible?

No!

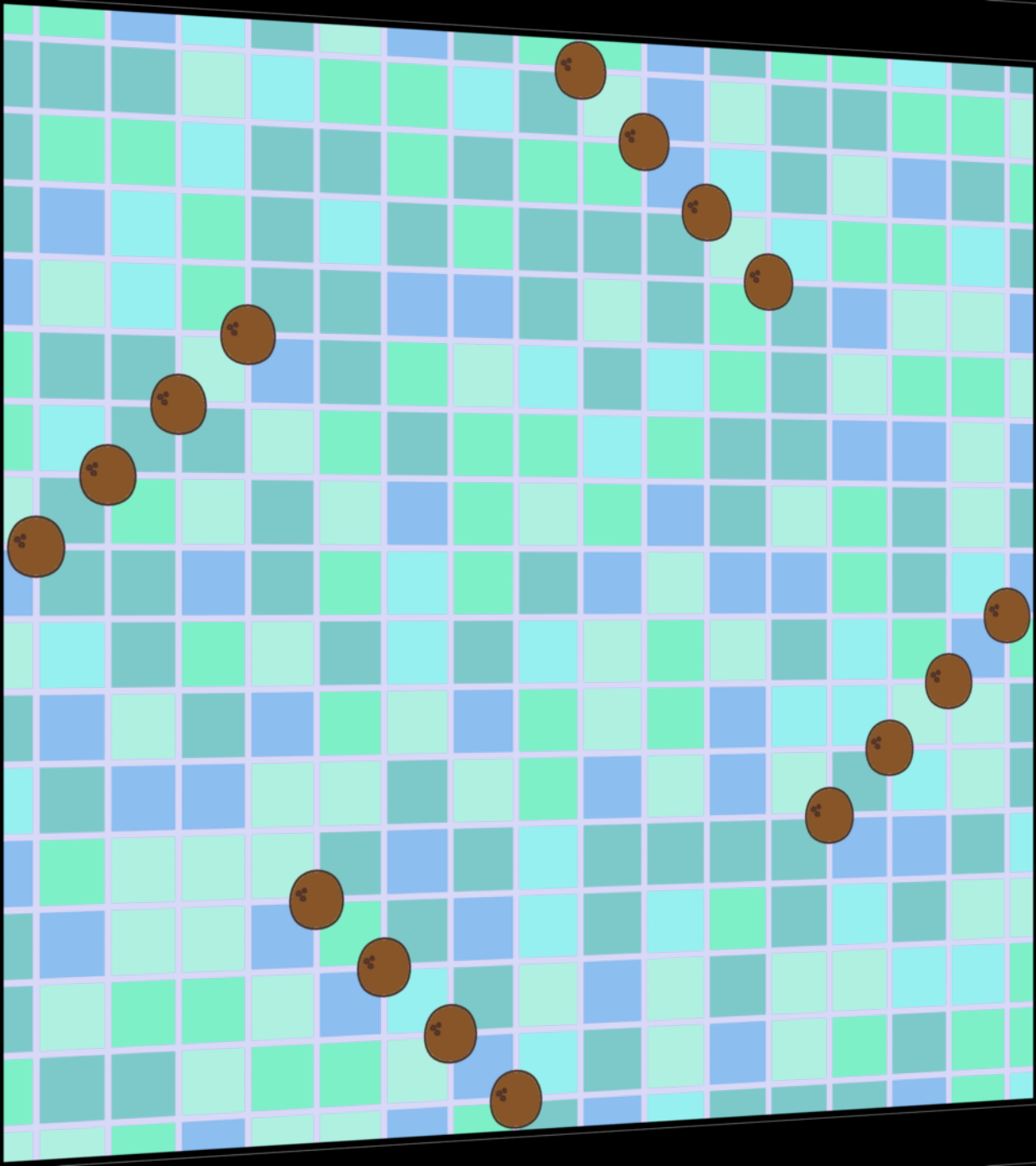


Can we always push
our coconuts into tidy
rectangles?

Obviously, n coconuts can
only be pushed into a $a \times b$
rectangle if $ab = n!$

But is it always possible?

No!



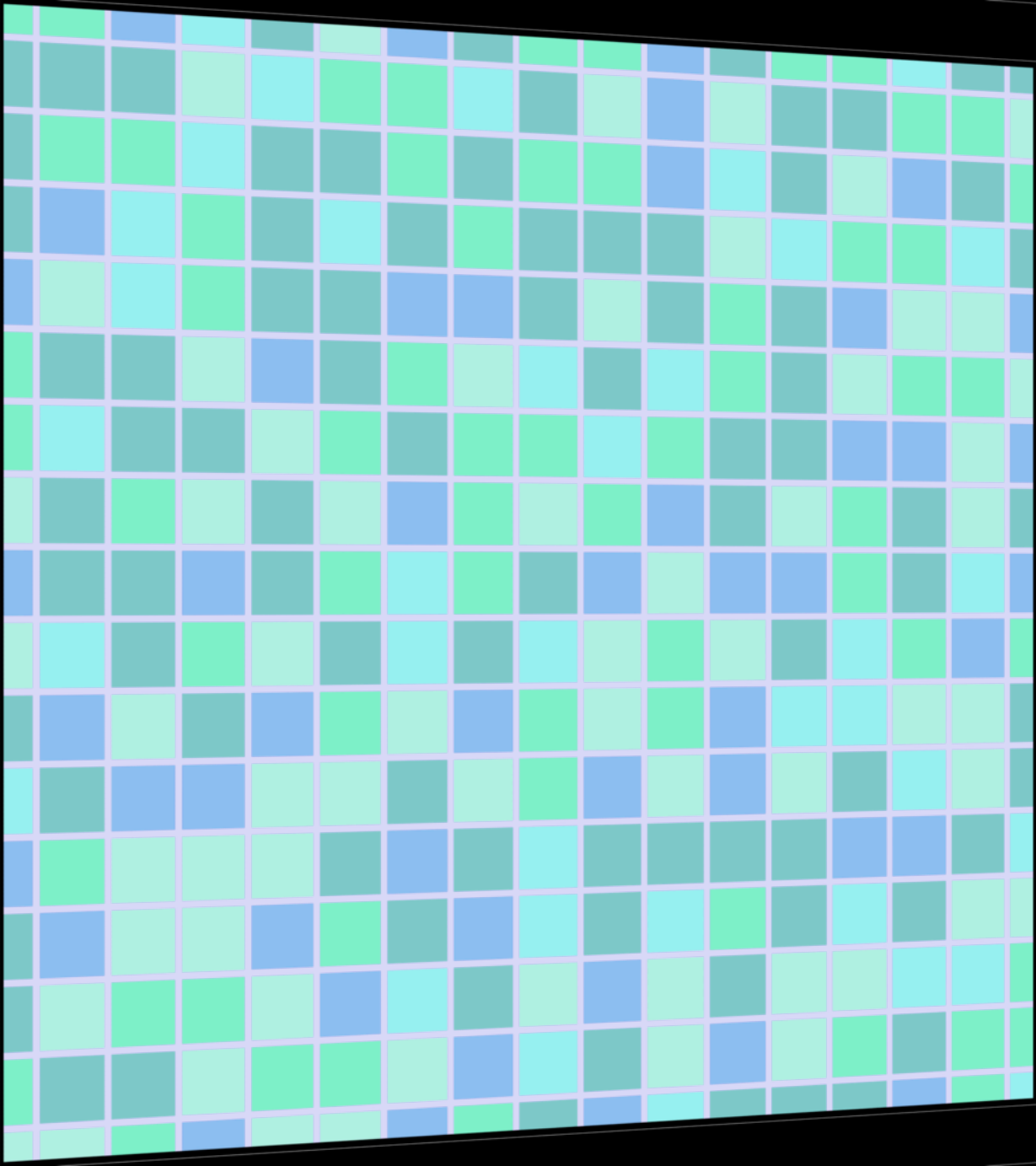
Can we always push
our coconuts into tidy
rectangles?

Obviously, n coconuts can
only be pushed into a $a \times b$
rectangle if $ab = n!$

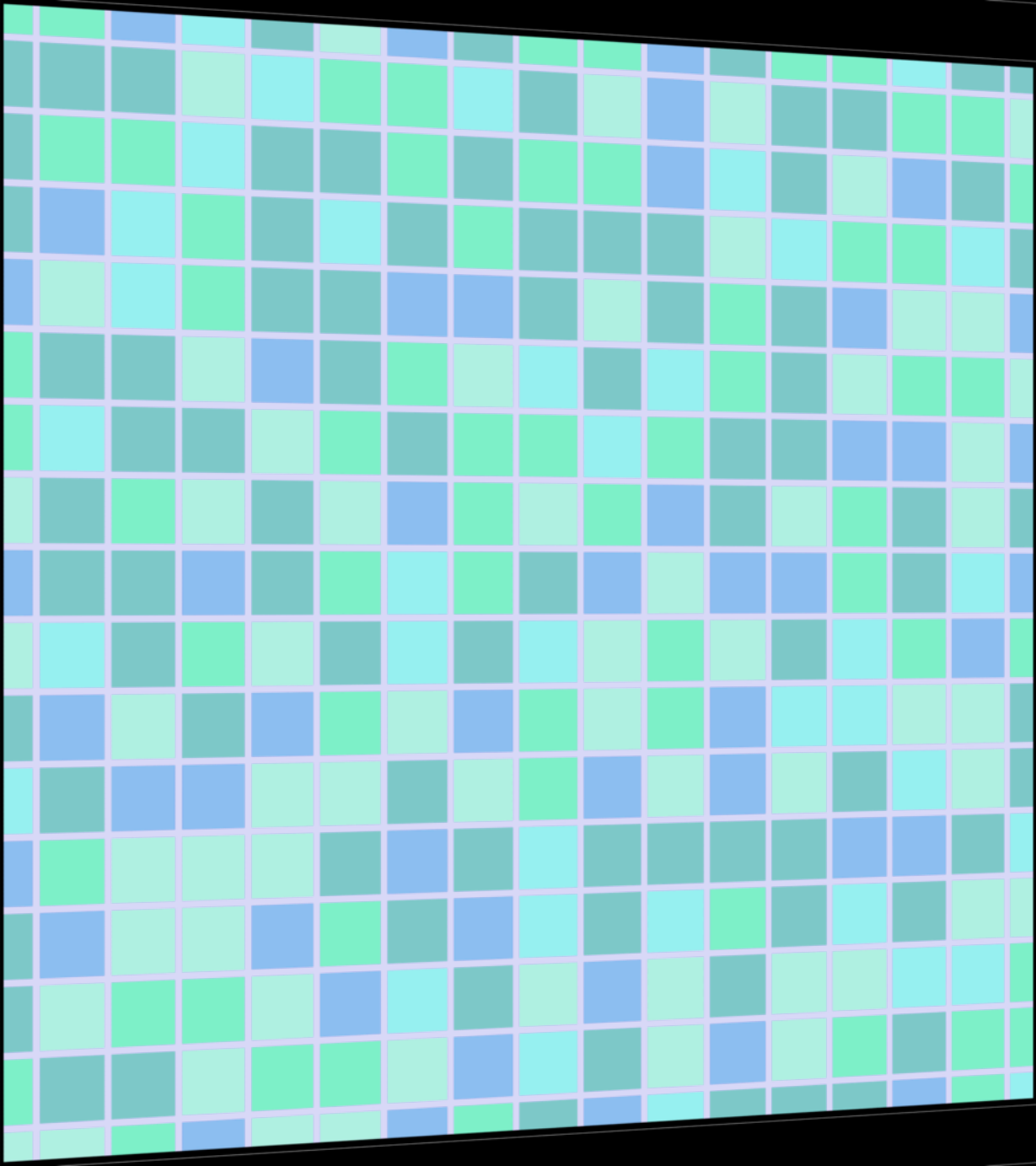
But is it always possible?

No!

...and some aspect ratios
are not even possible if we
start with at most one
coconut per row and column.

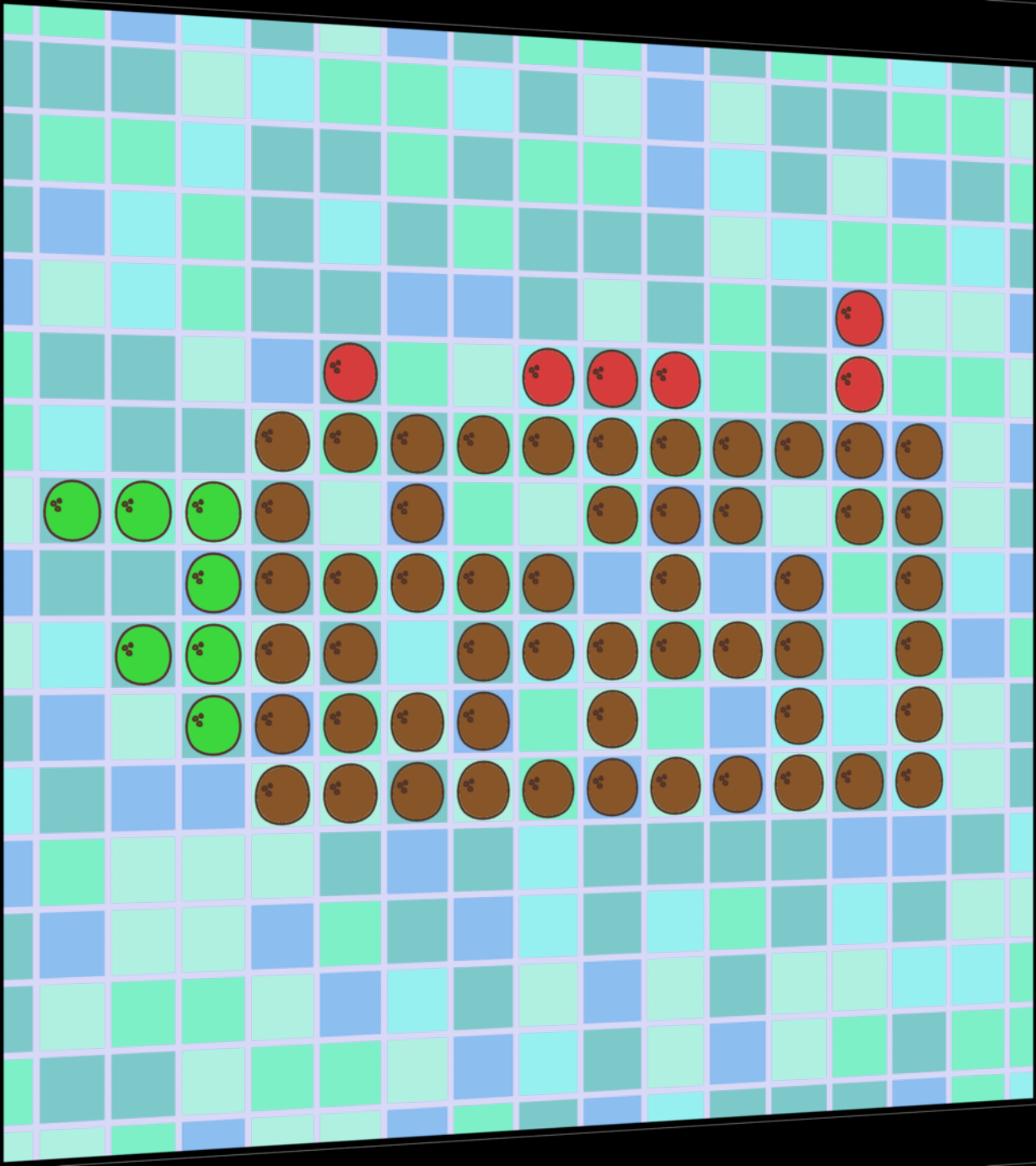


Deciding whether n
coconuts can be pushed
into an $a \times b$ rectangle is
NP-complete.



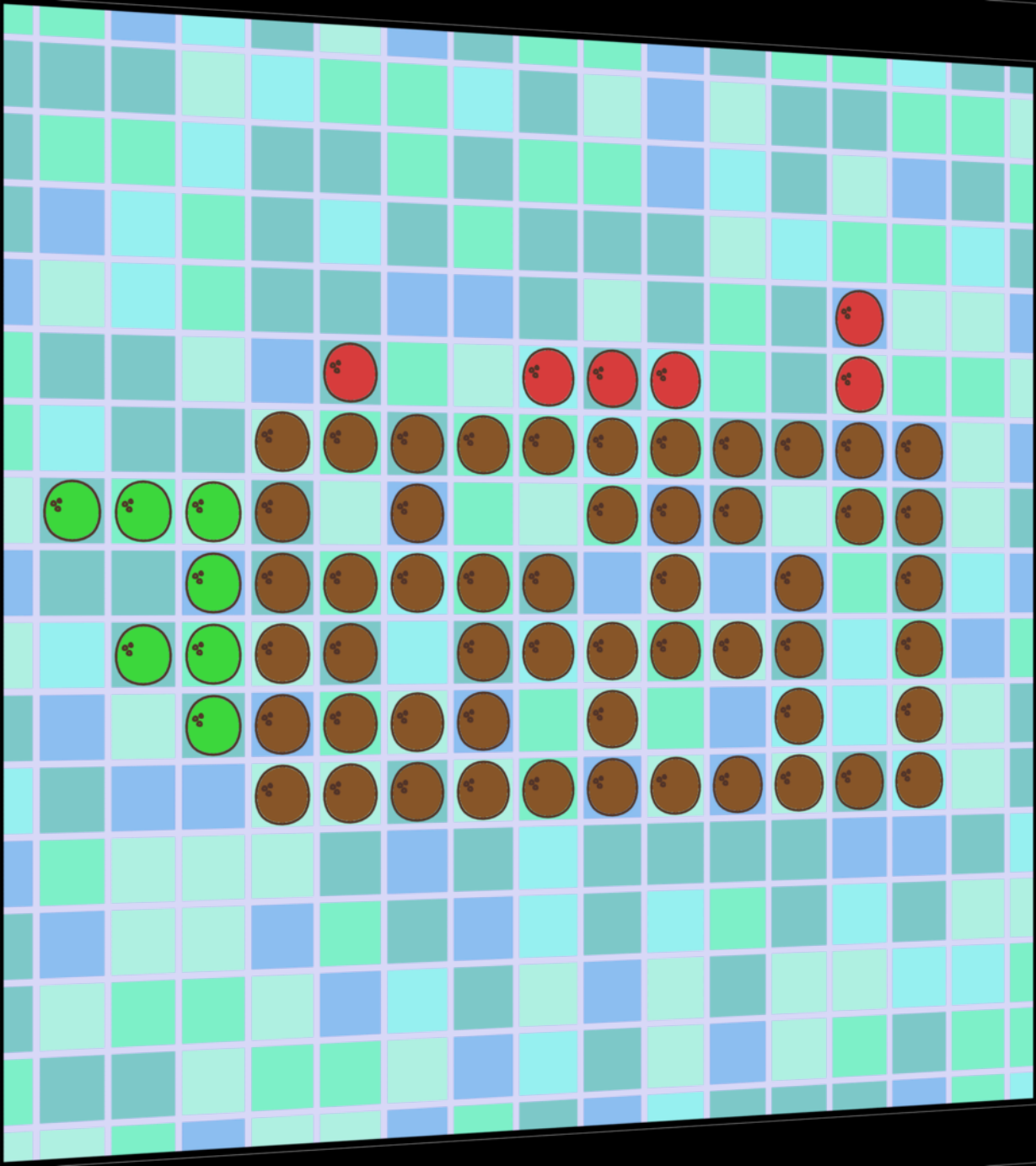
Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

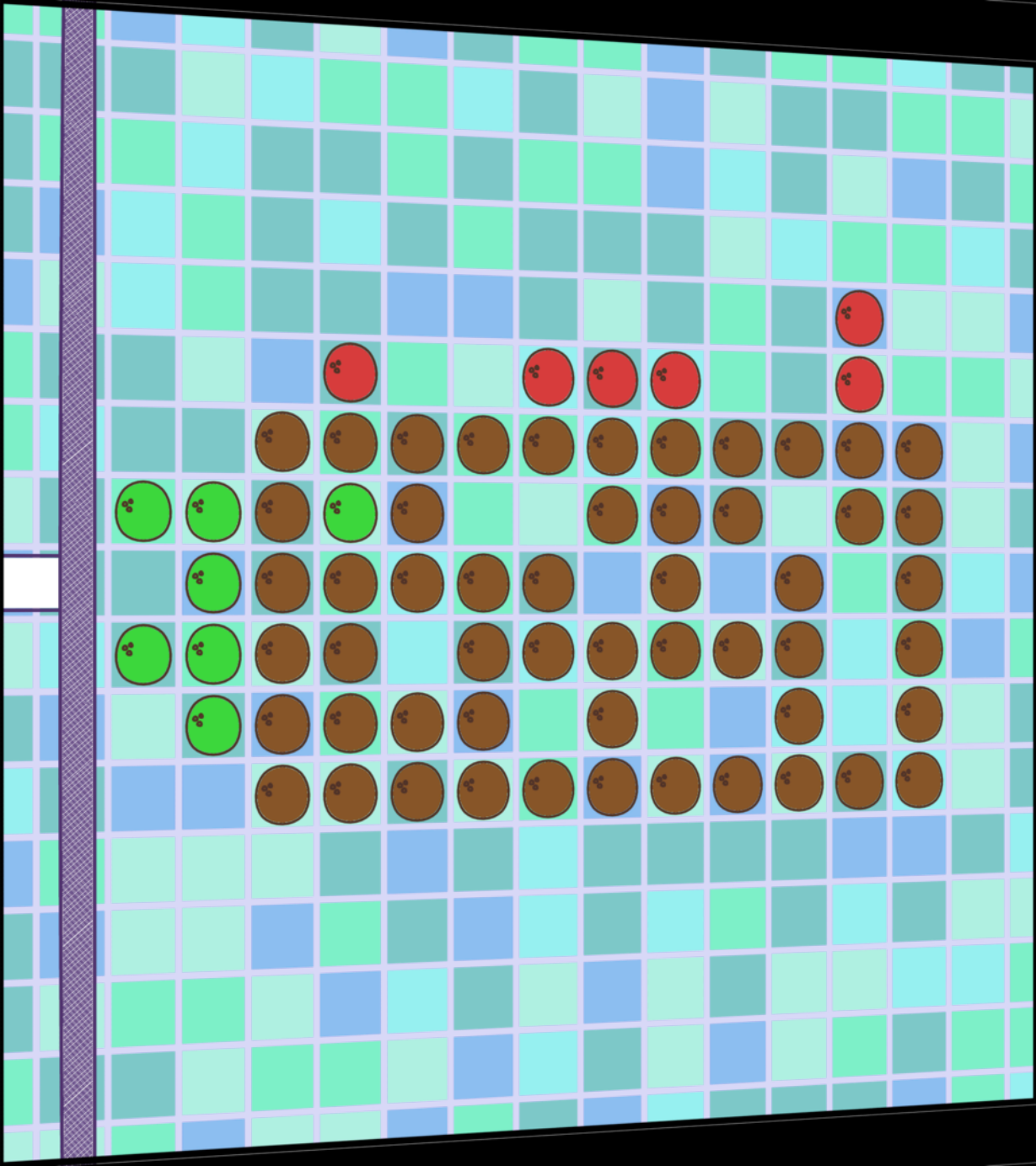
General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

The question then becomes in which order to push to make everything fit.

Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

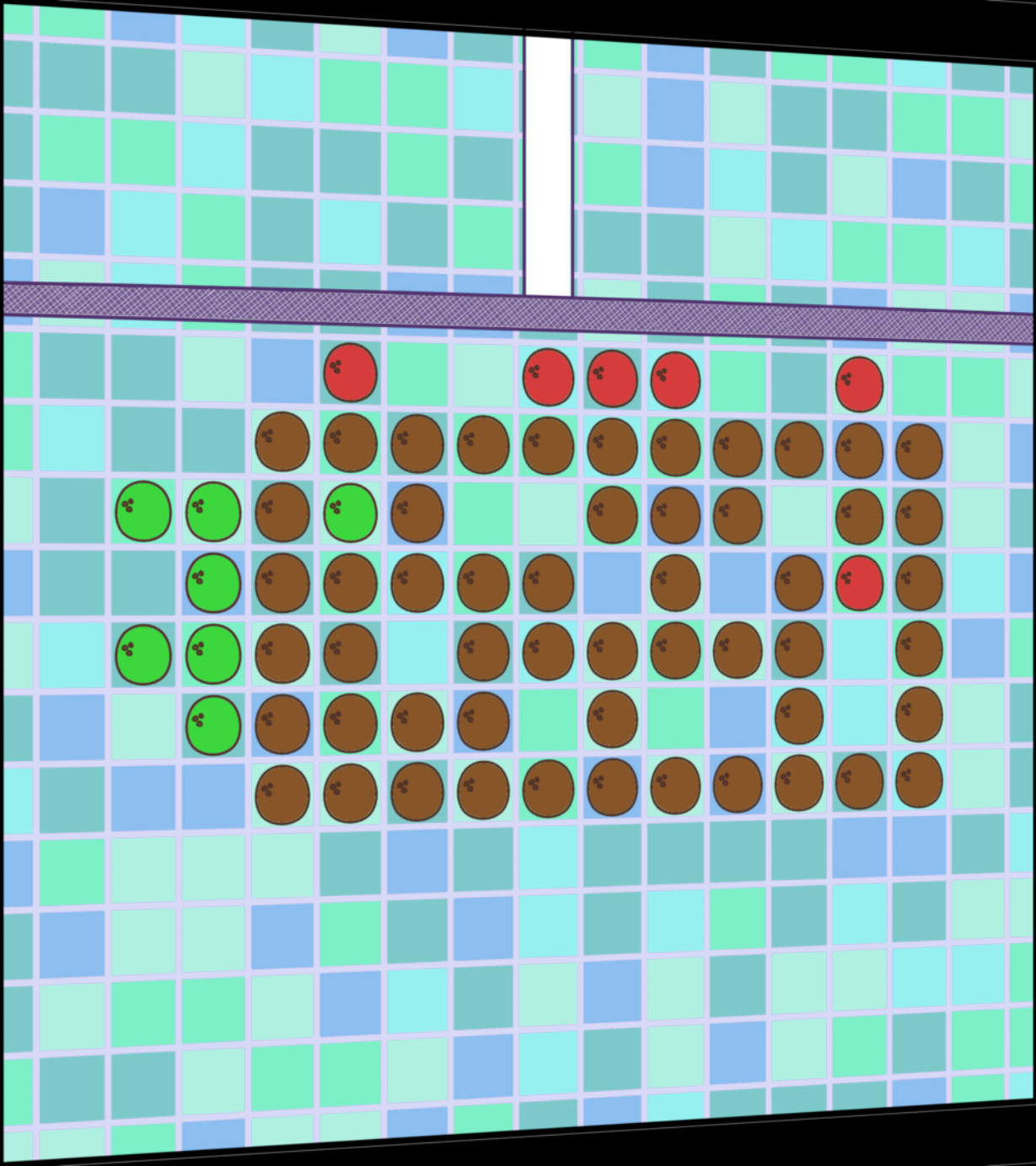
The question then becomes in which order to push to make everything fit.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

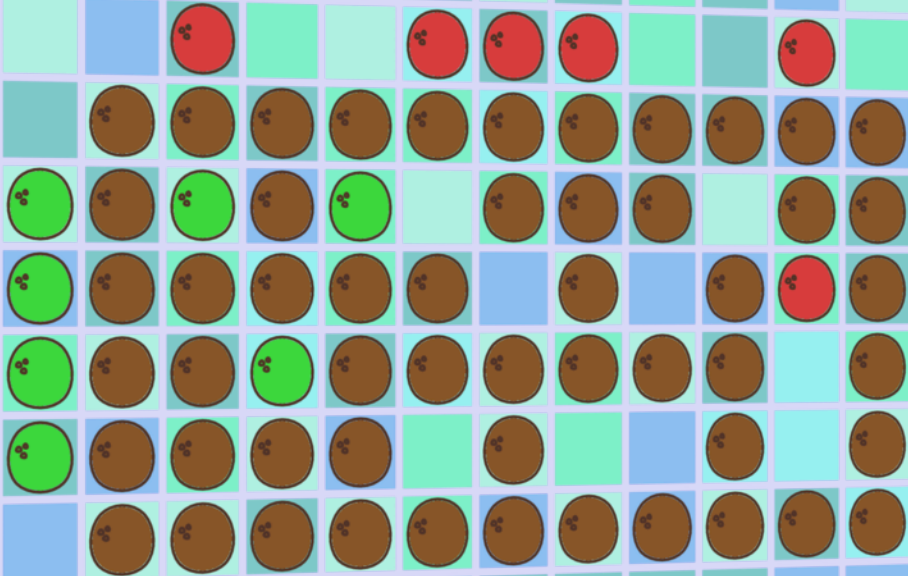
The question then becomes in which order to push to make everything fit.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

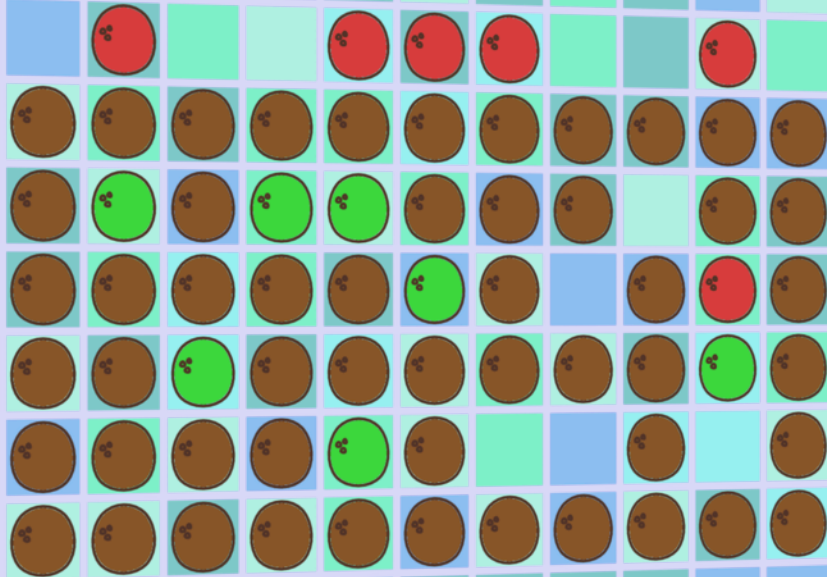
The question then becomes in which order to push to make everything fit.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

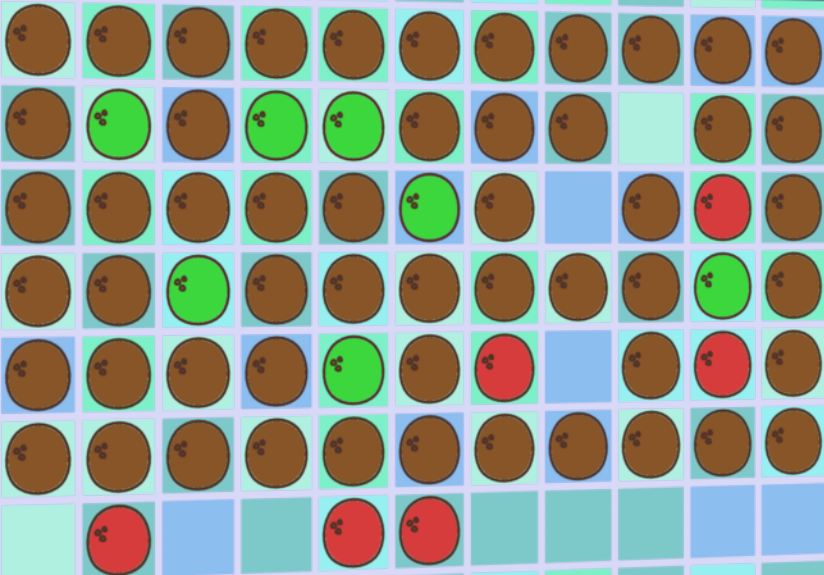
The question then becomes in which order to push to make everything fit.

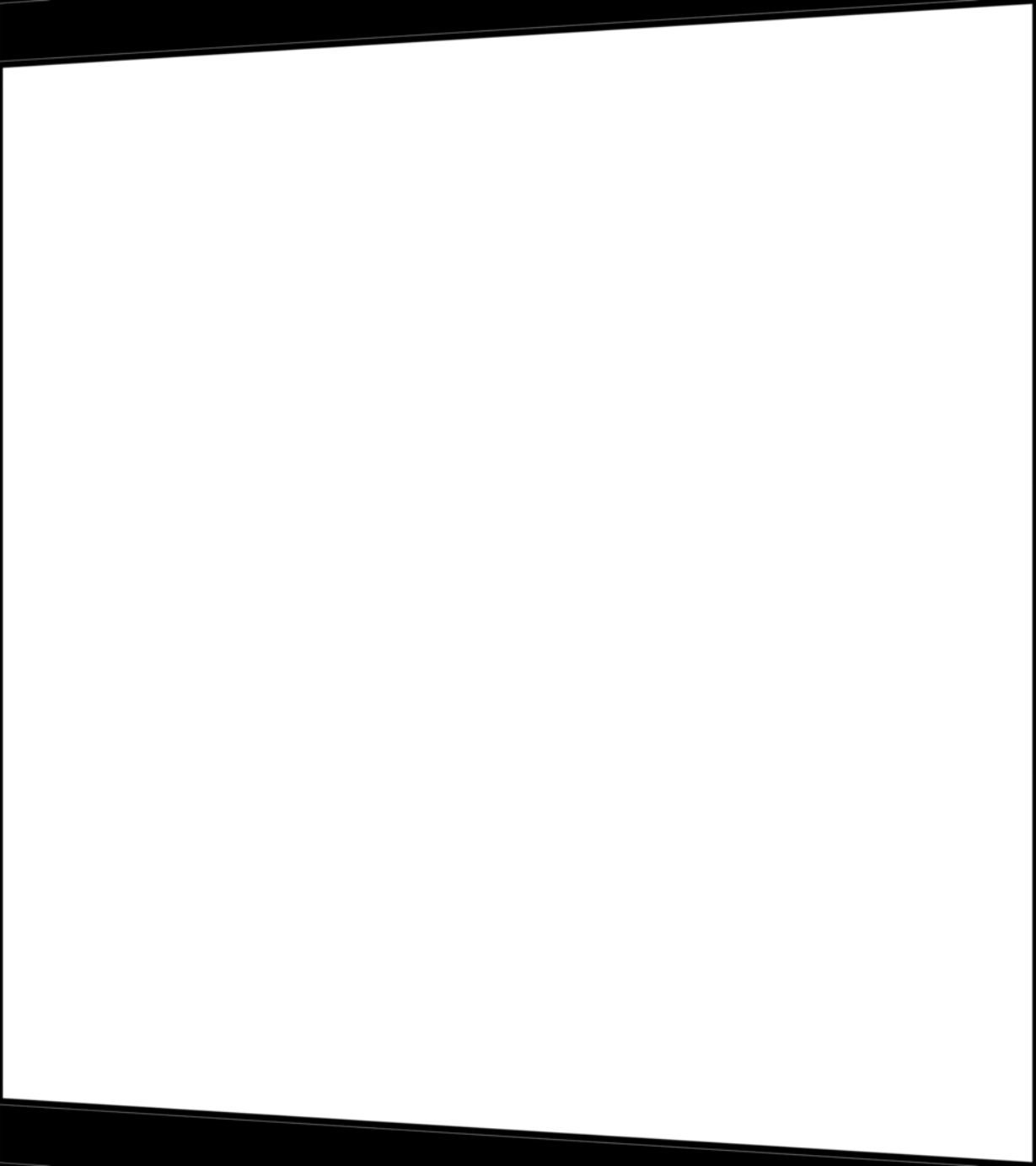
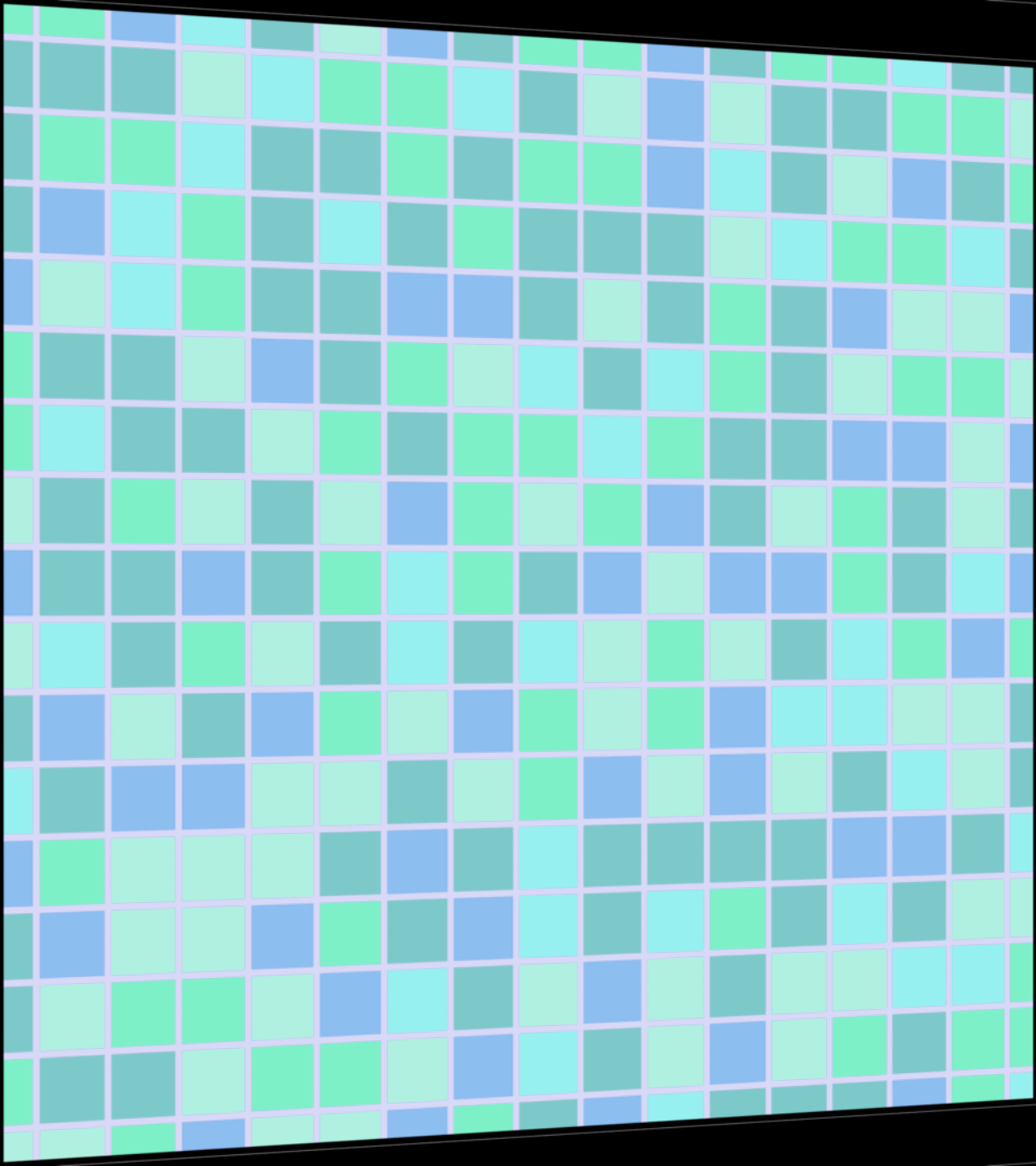


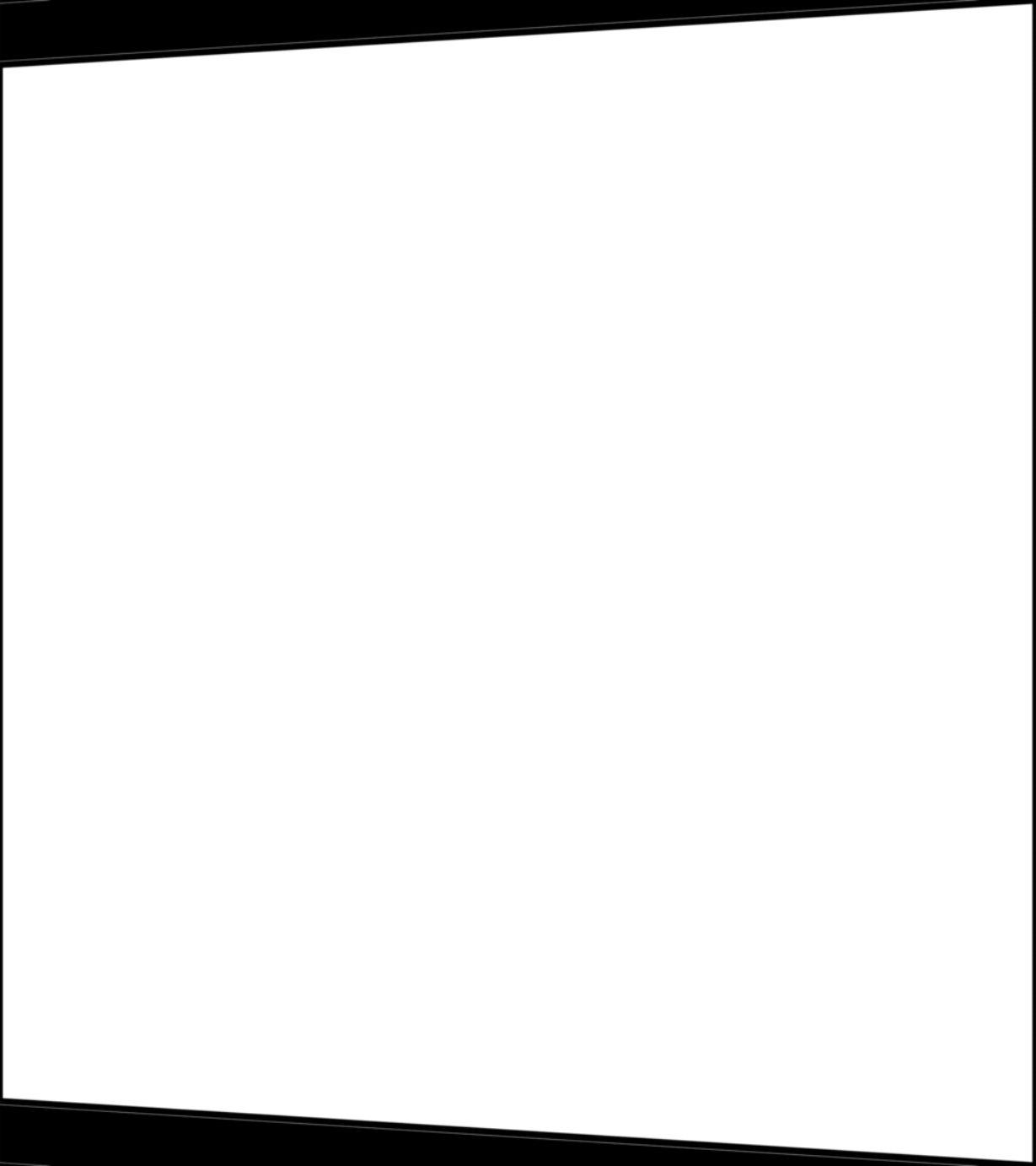
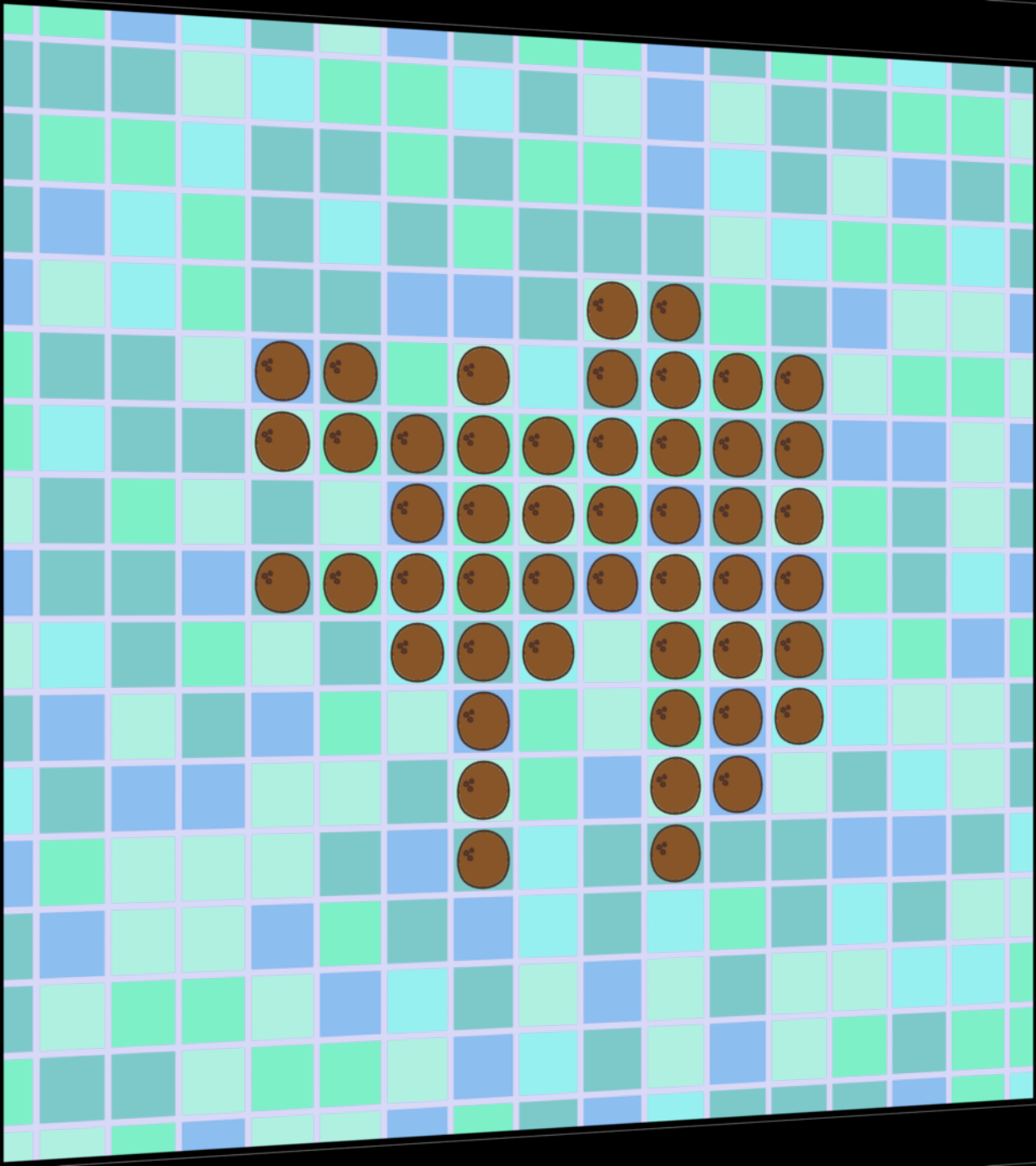
Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

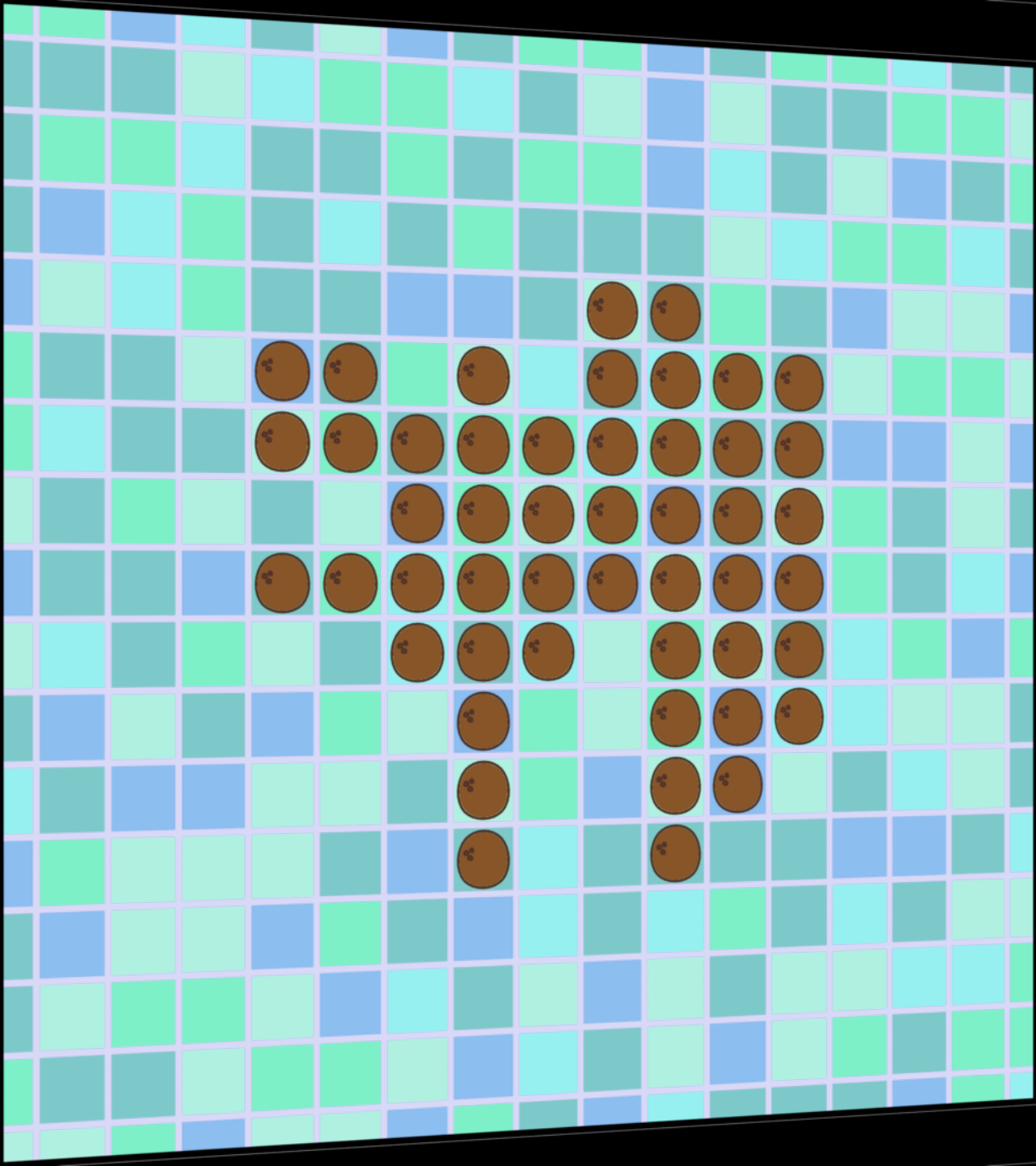
General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

The question then becomes in which order to push to make everything fit.

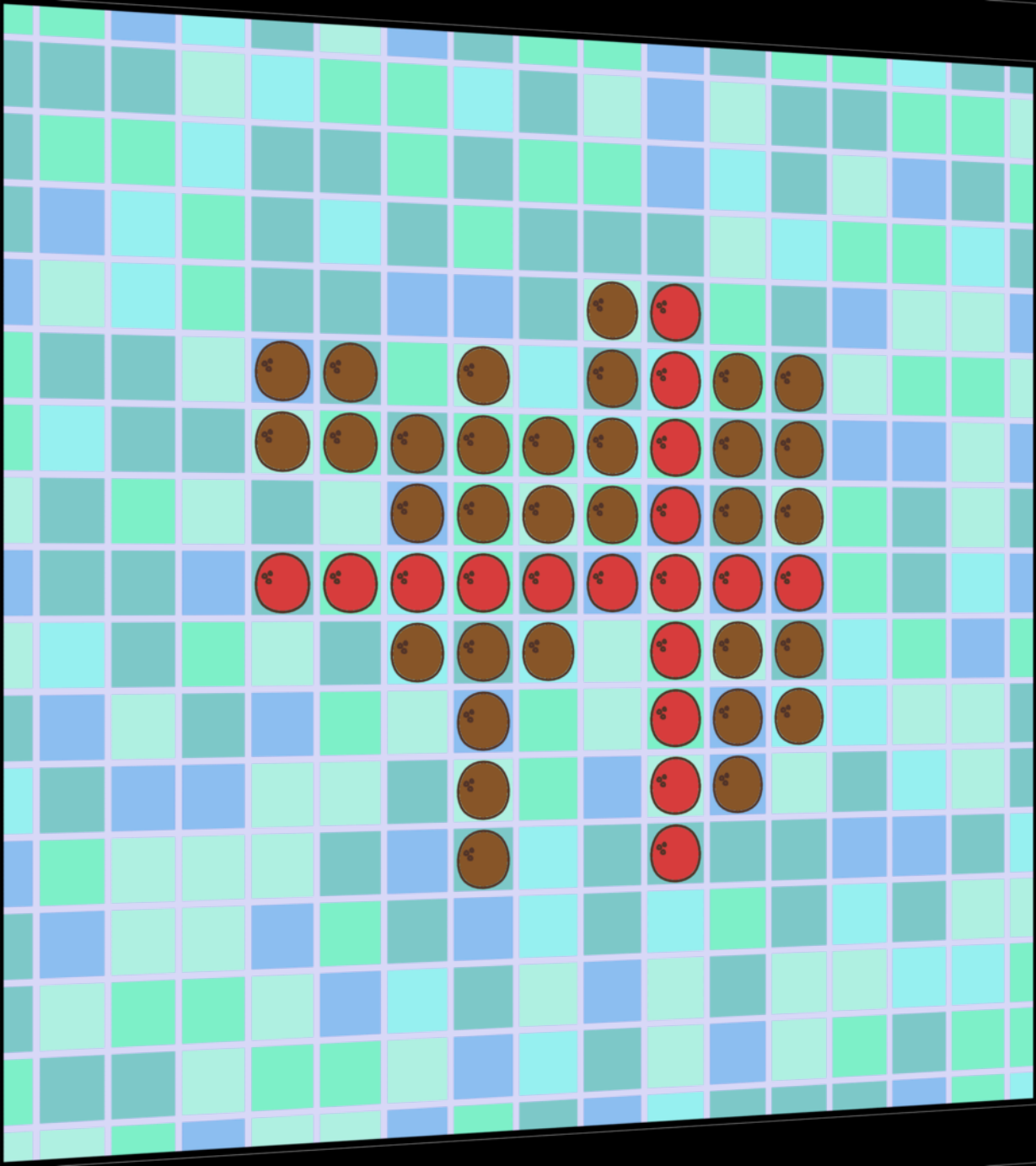




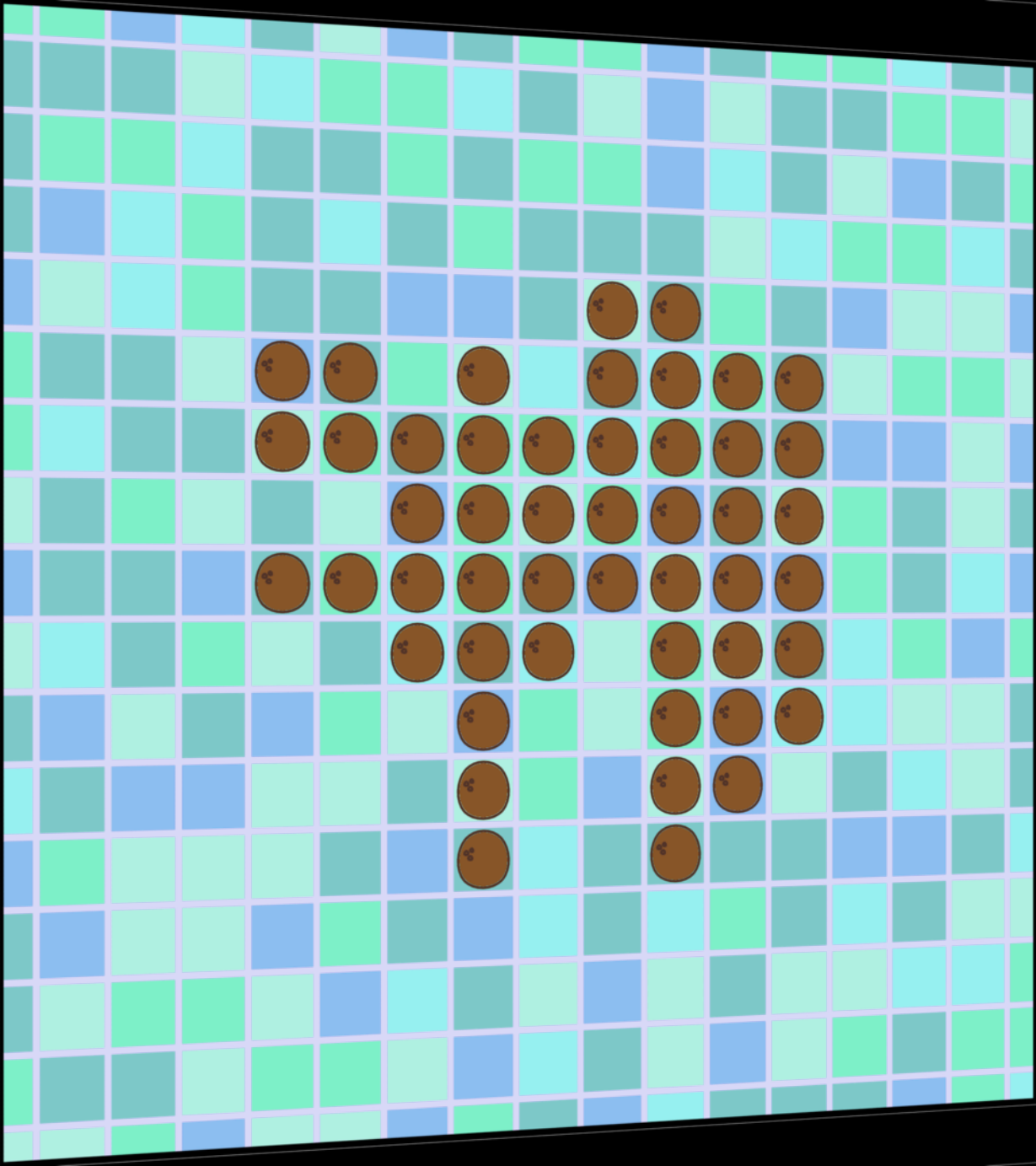




When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

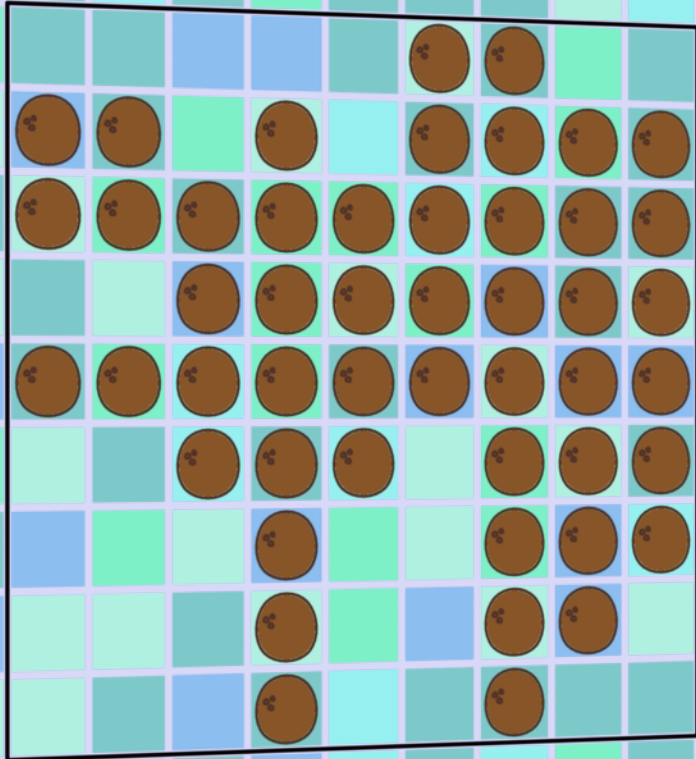


When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

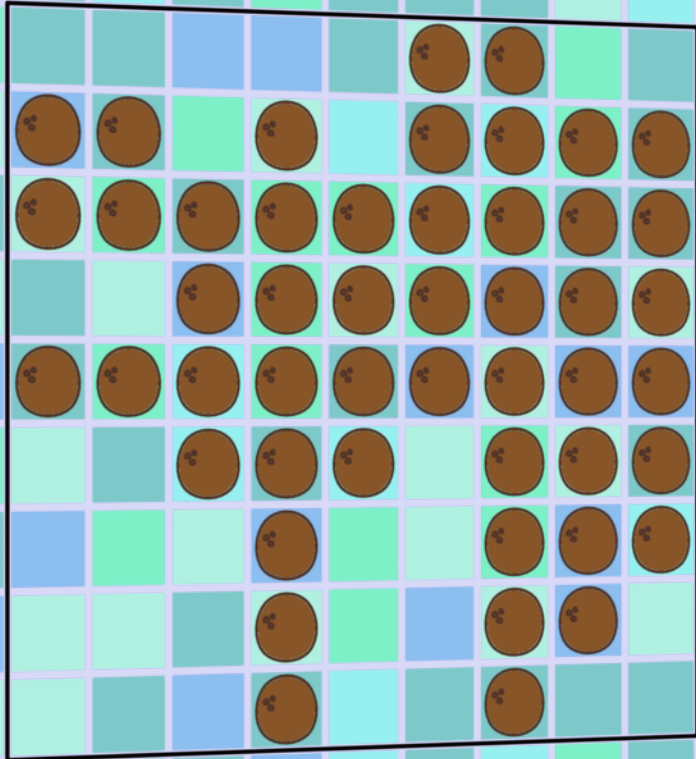


When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

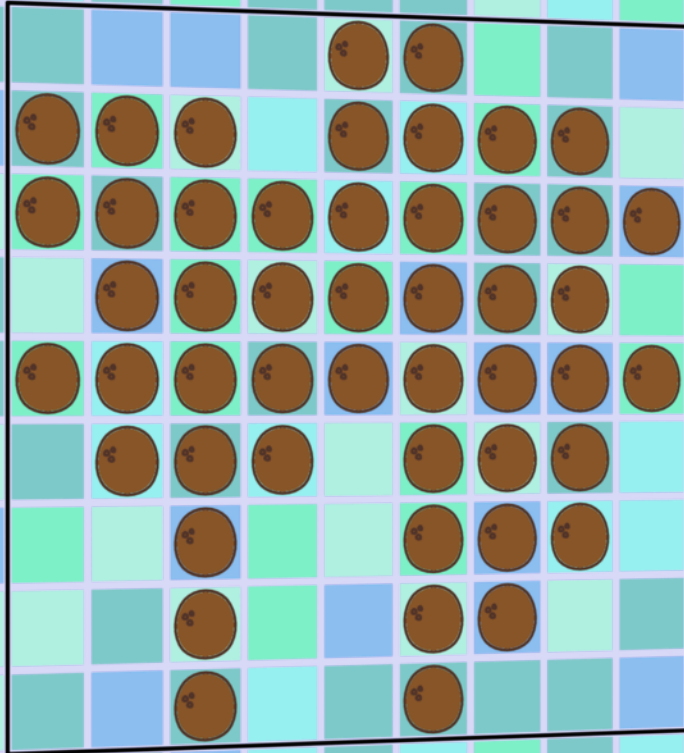
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



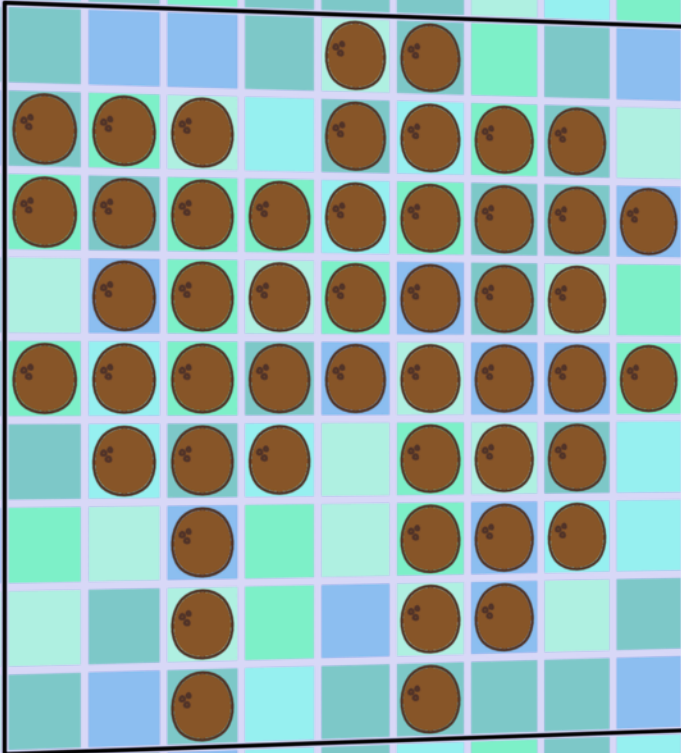
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

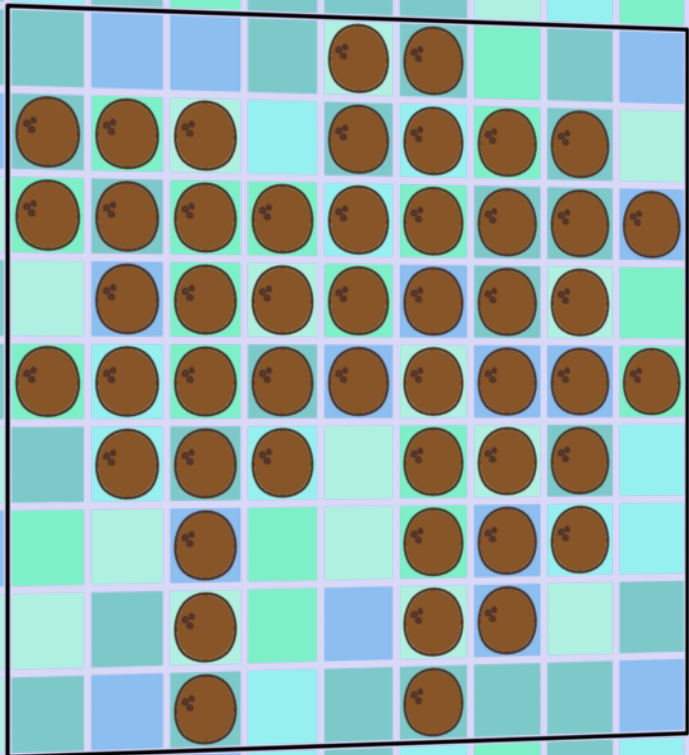


When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



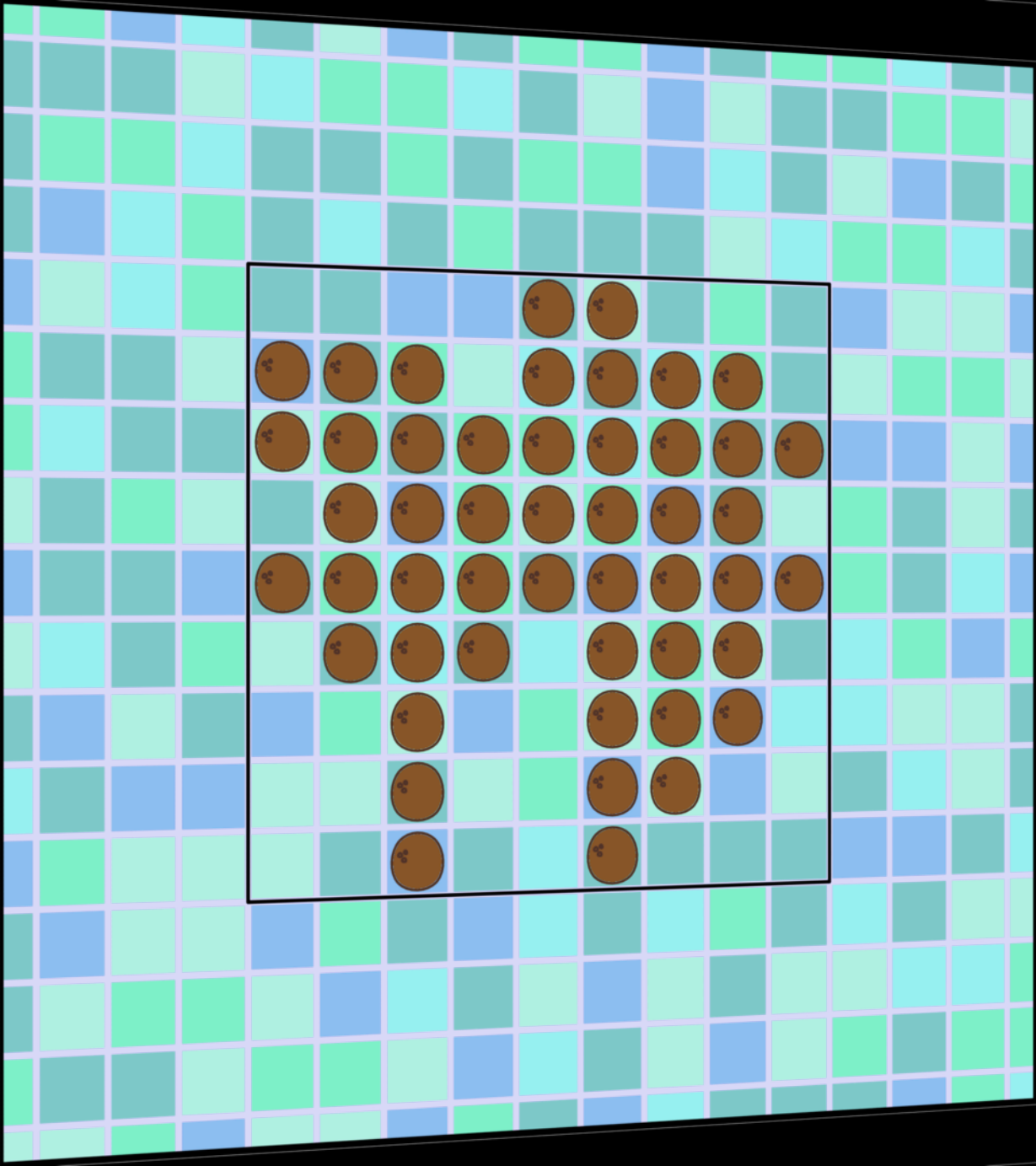
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.





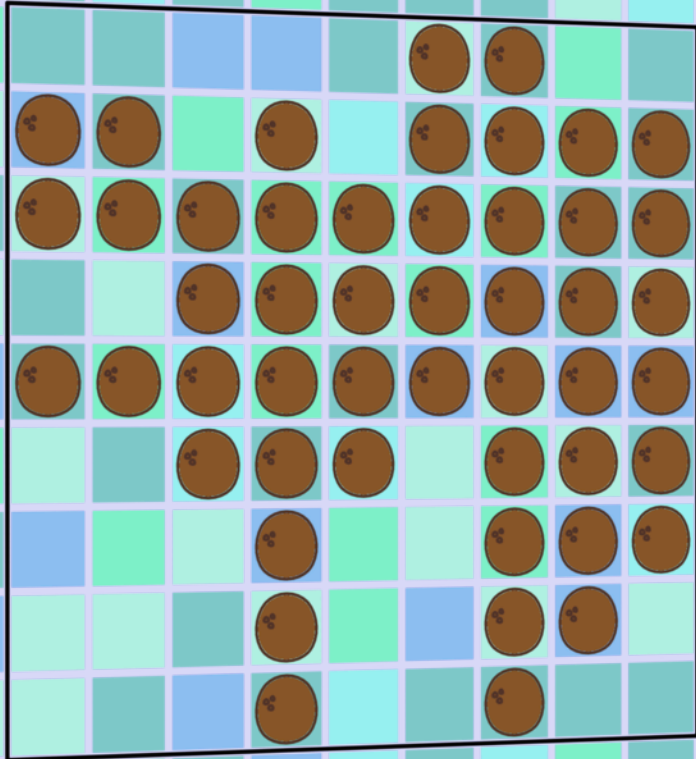
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.



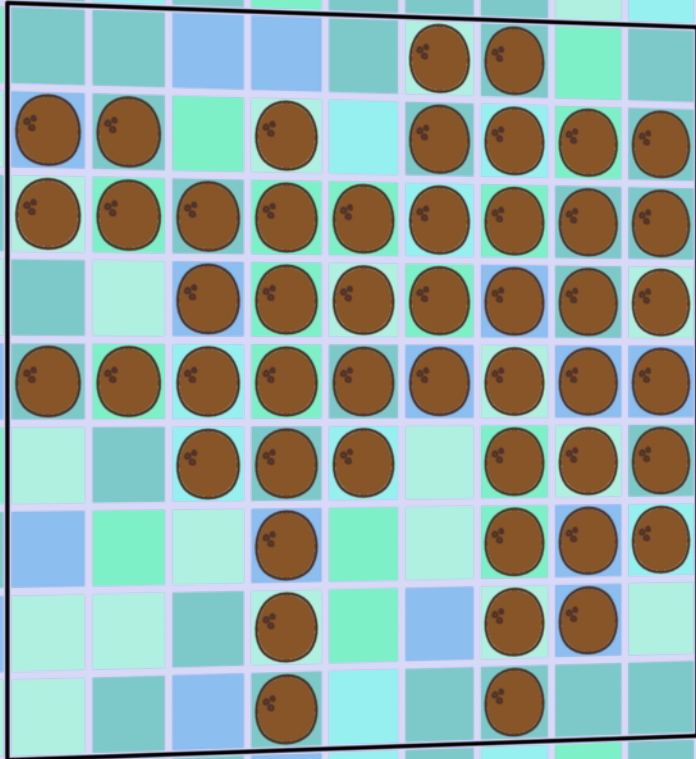
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

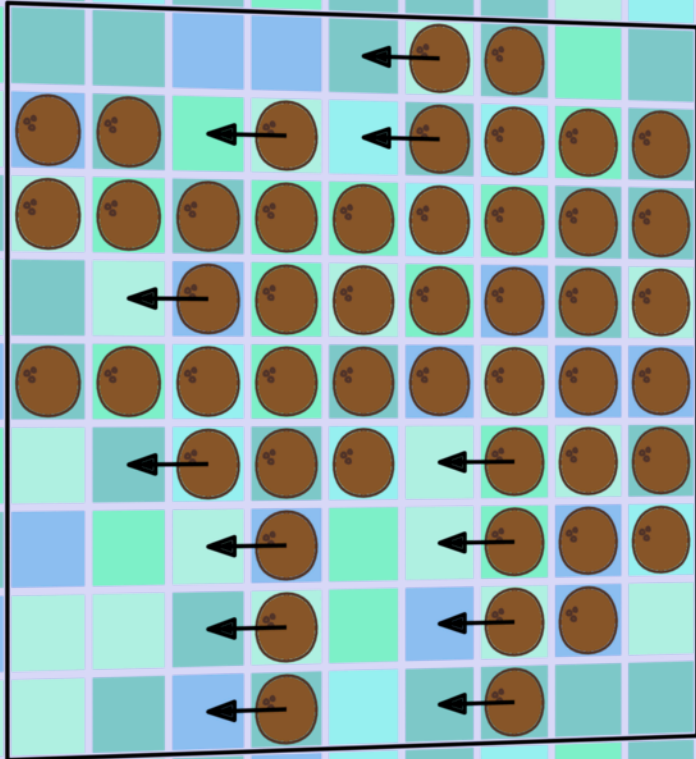
Now, suppose we keep the view centered on the box.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

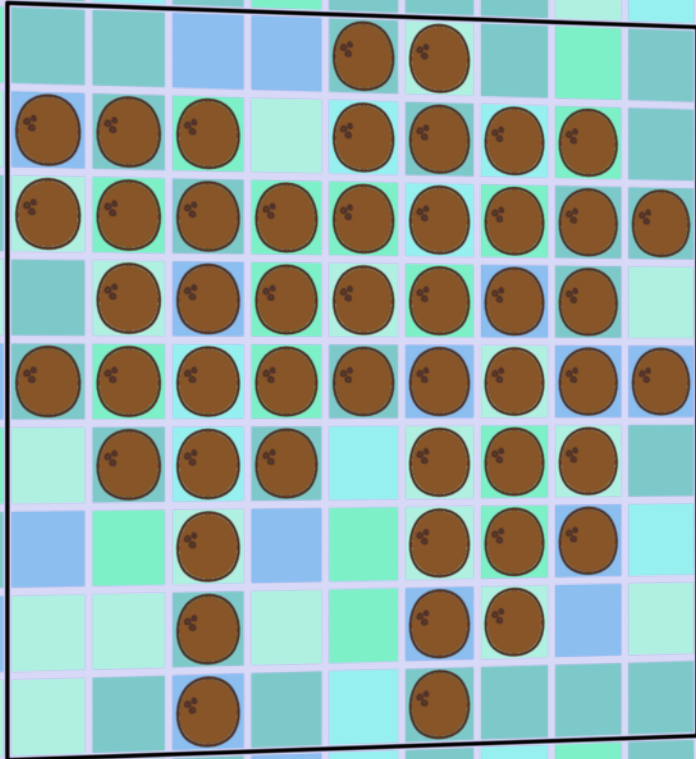
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

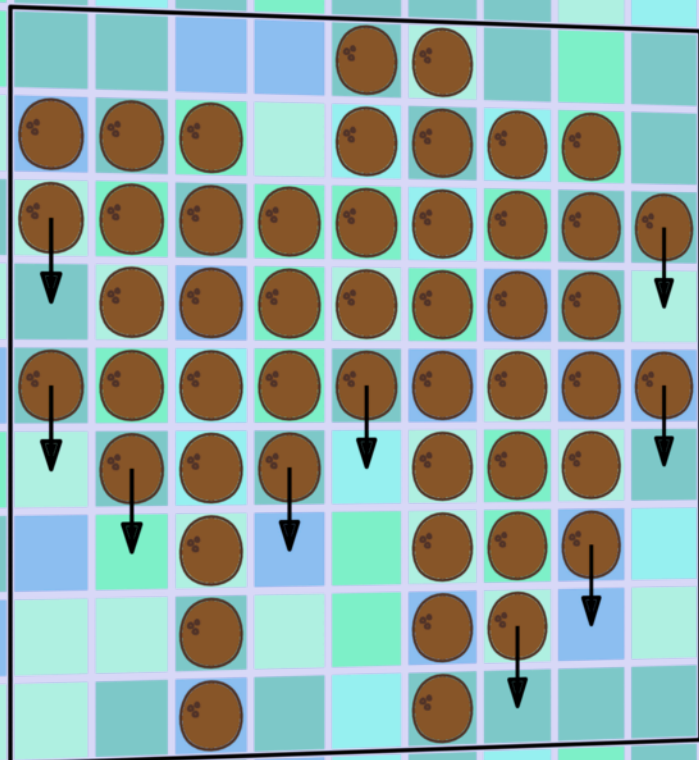
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

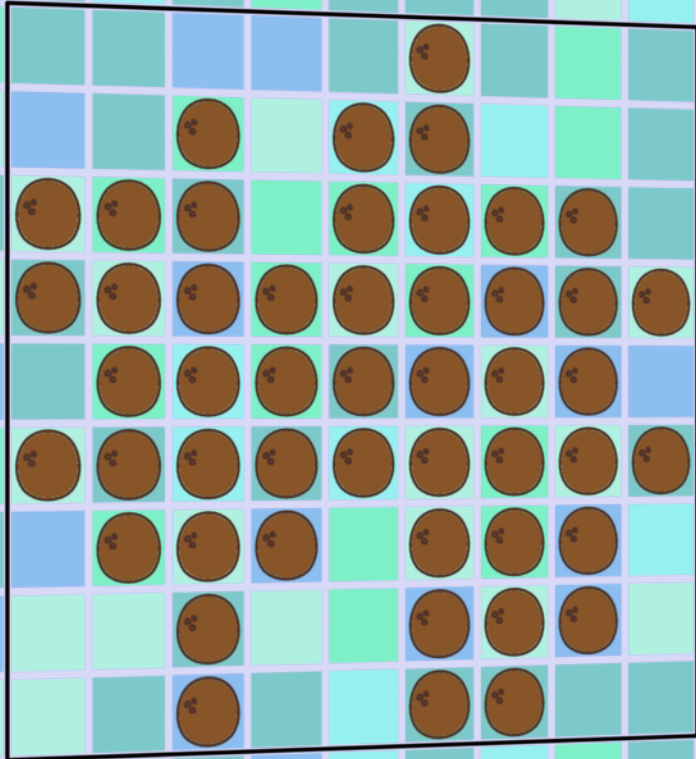
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

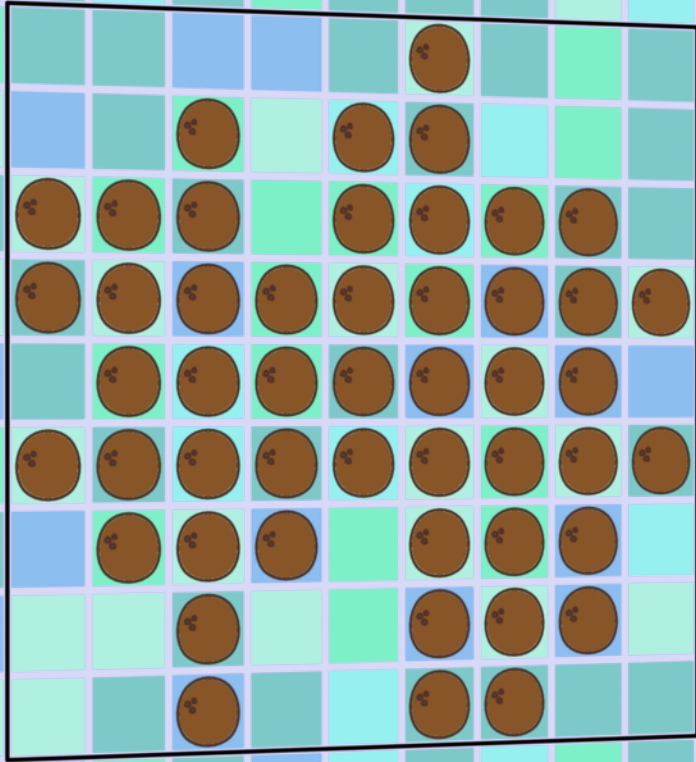
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



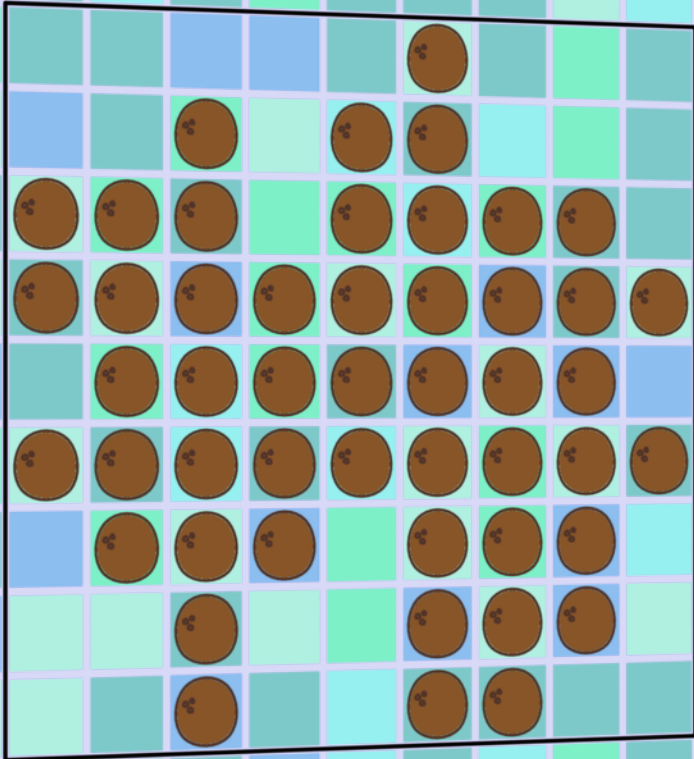
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

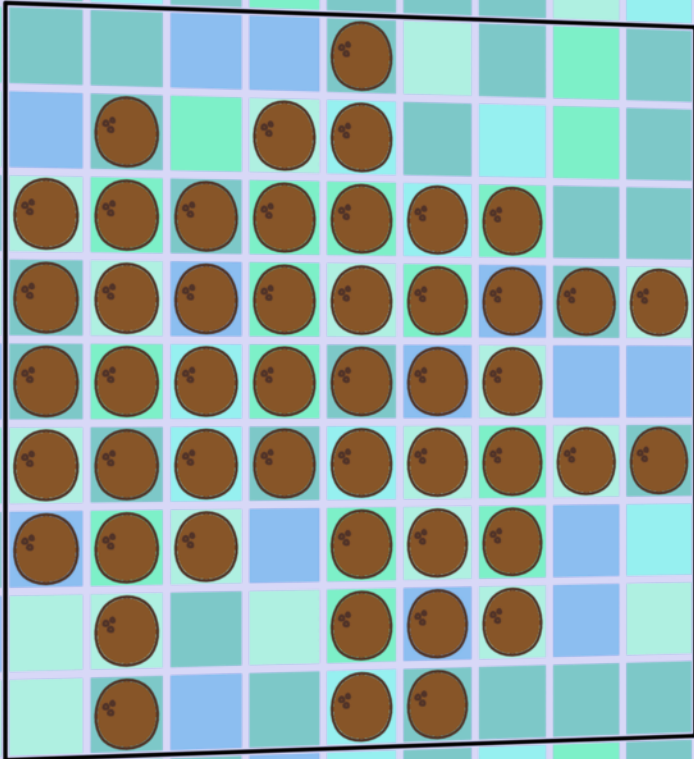
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.

Same game, different viewpoint.

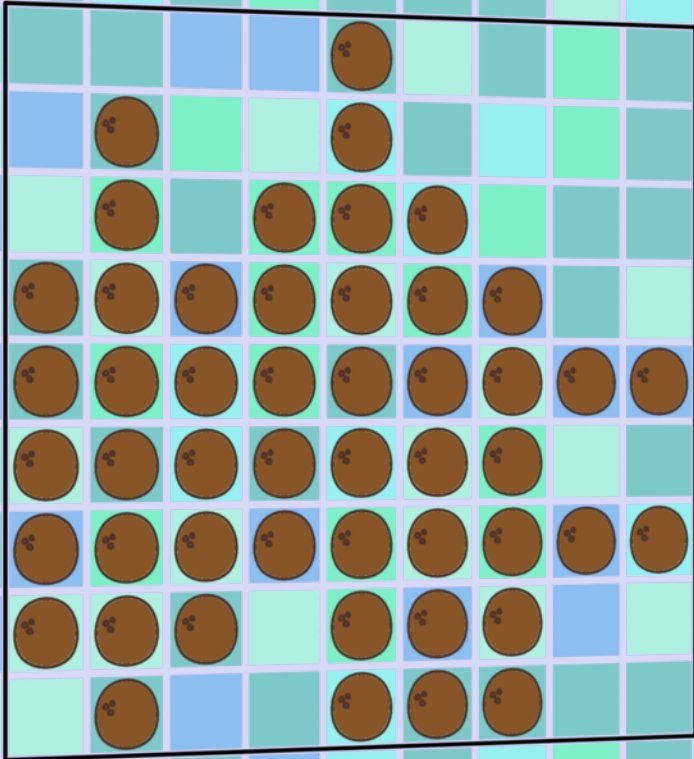
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



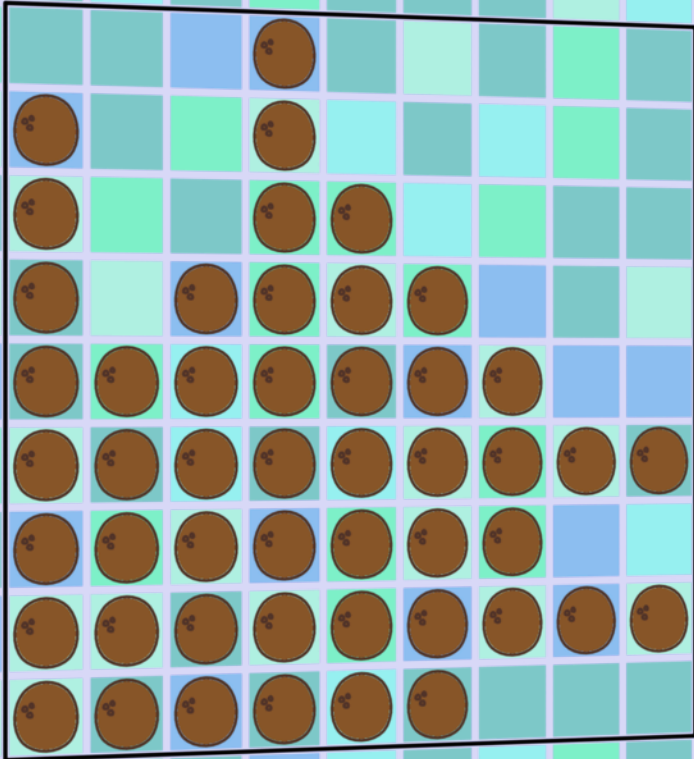
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



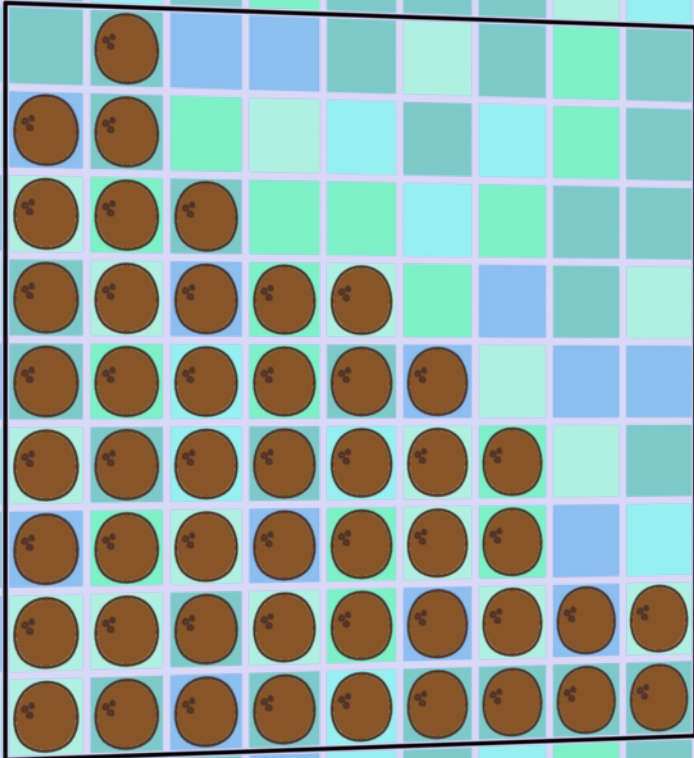
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



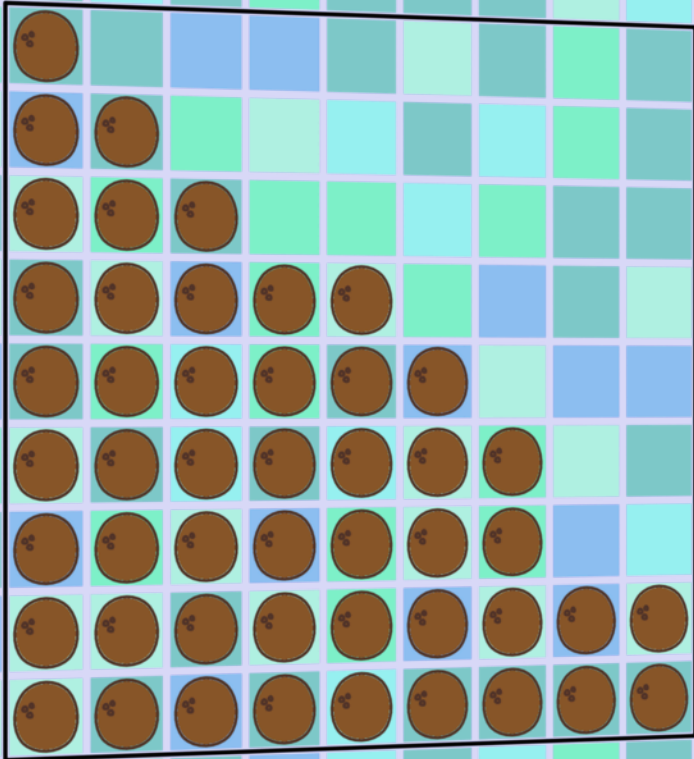
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



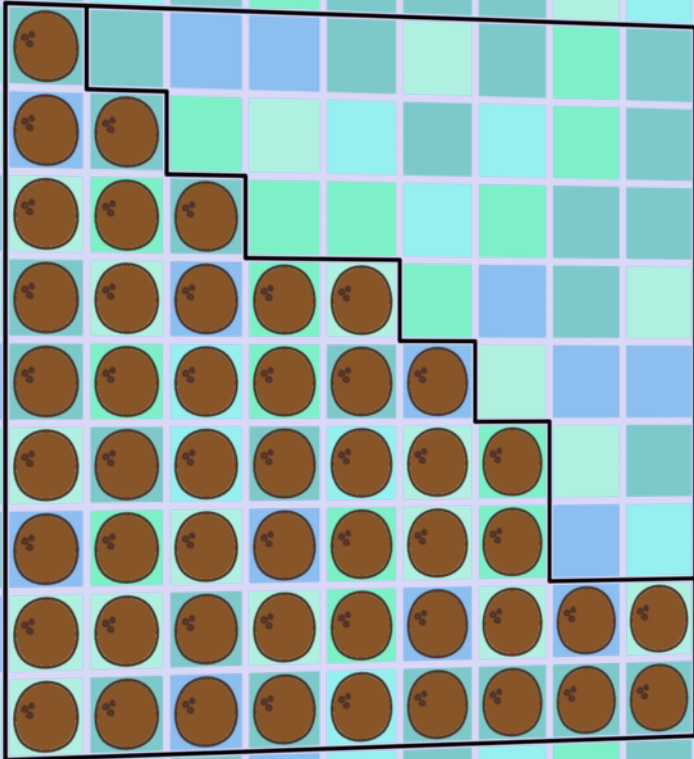
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.

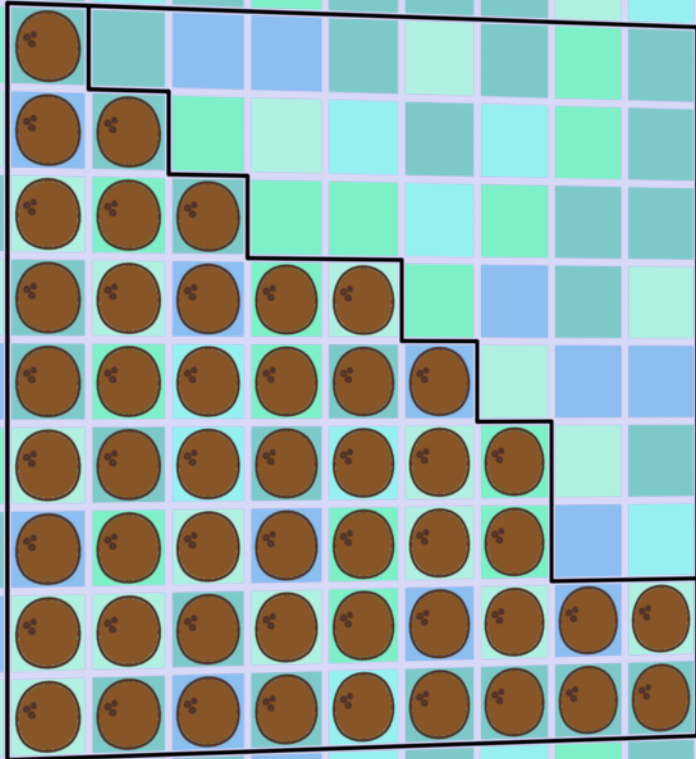


After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.

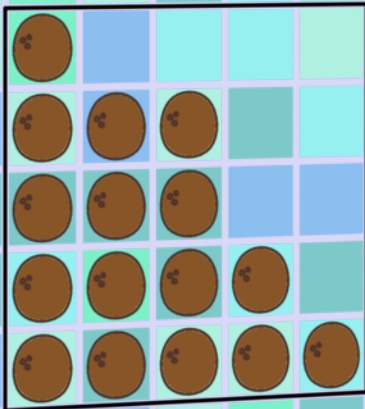
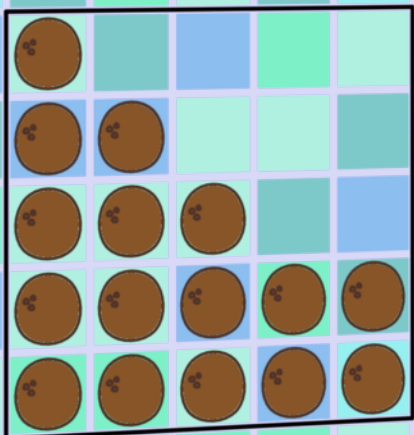
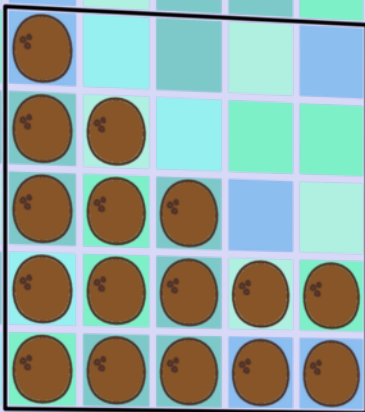
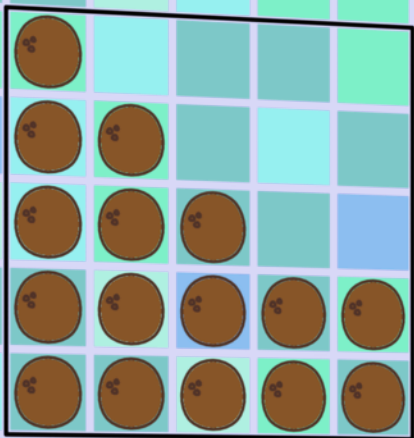


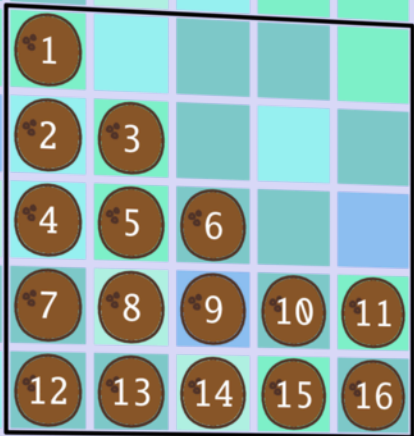


After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.

What can we say about reconfiguring canonical configurations?

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.



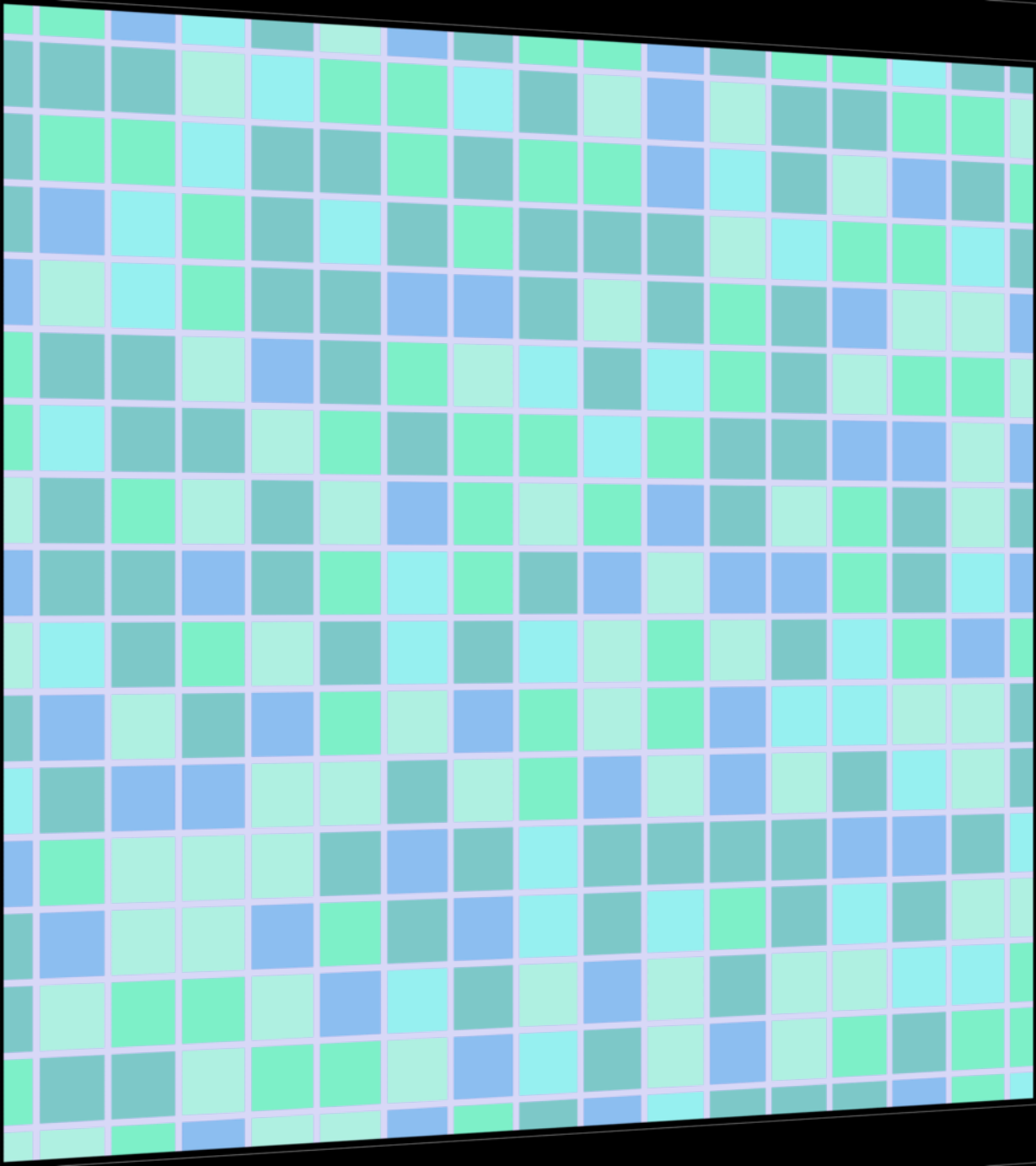


Theorem 2. Every realizable permutation of a canonical configuration is even.

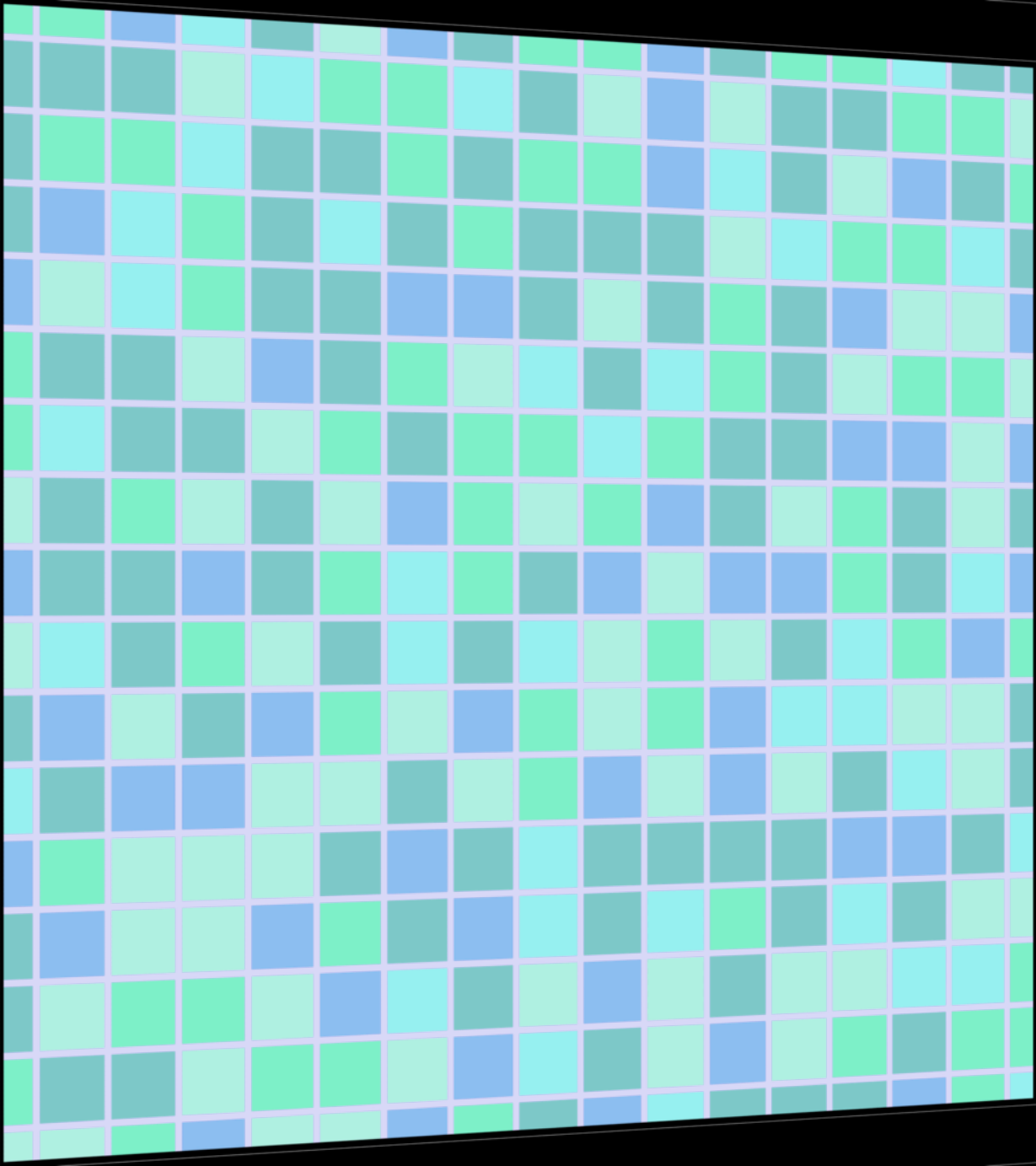
Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape:



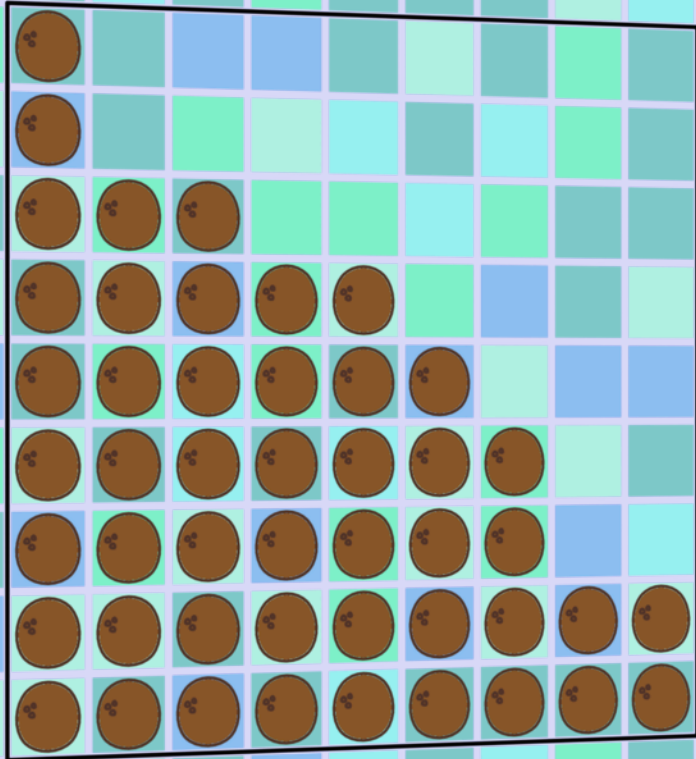


Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.



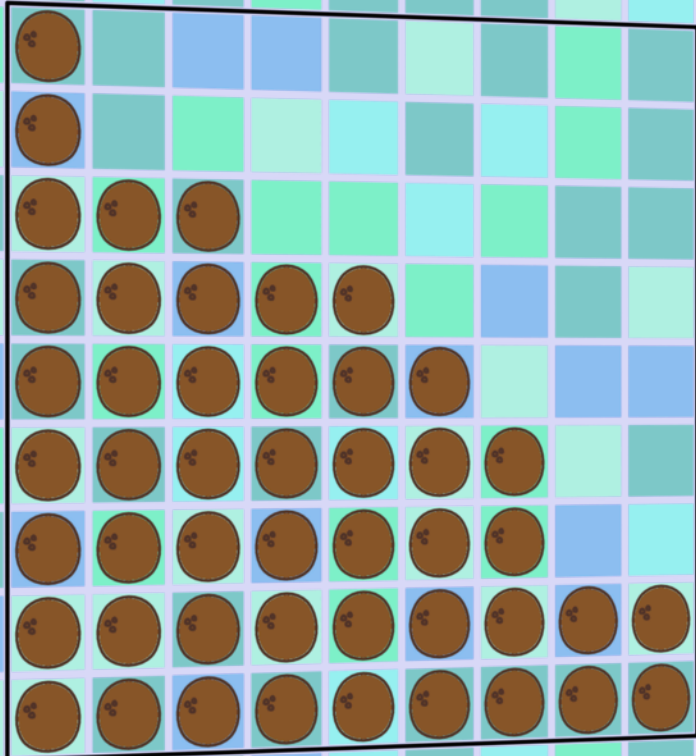
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

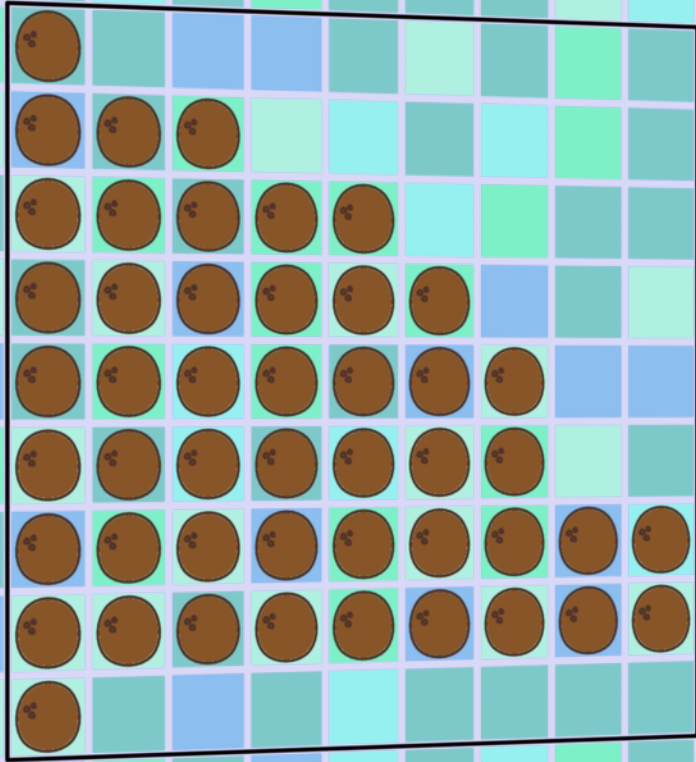
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

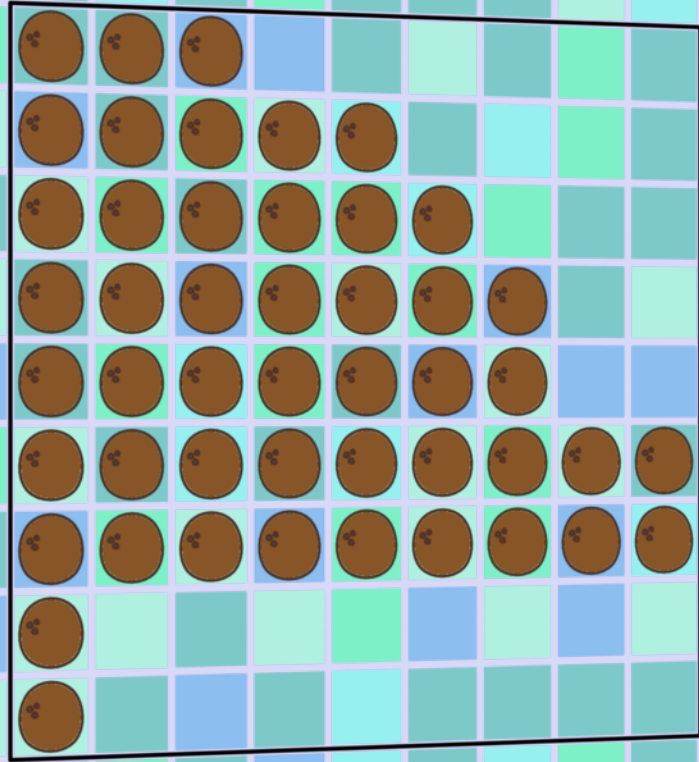
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

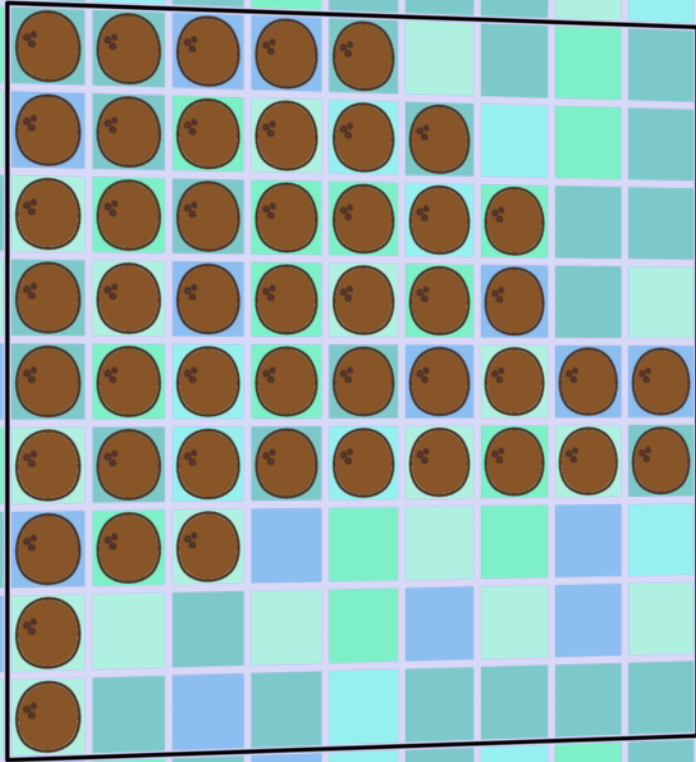
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

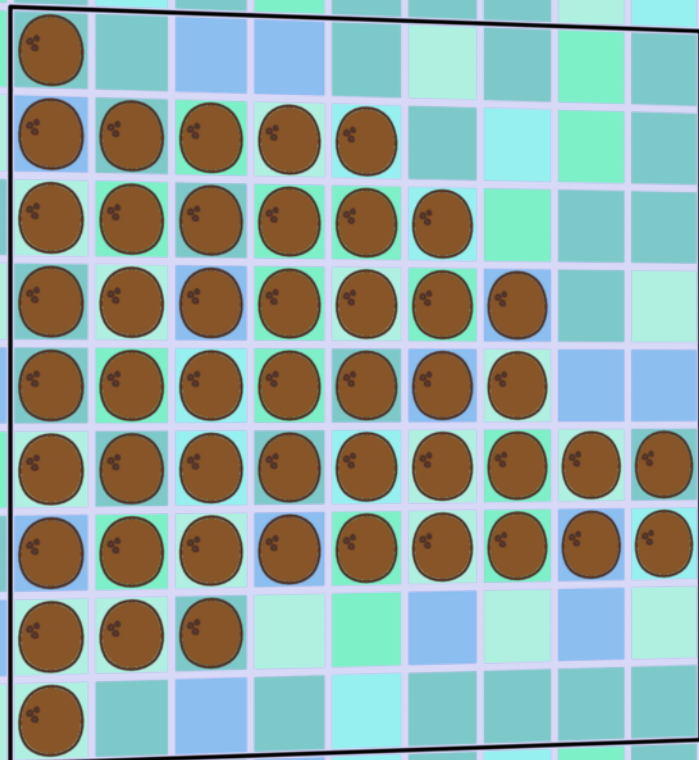
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

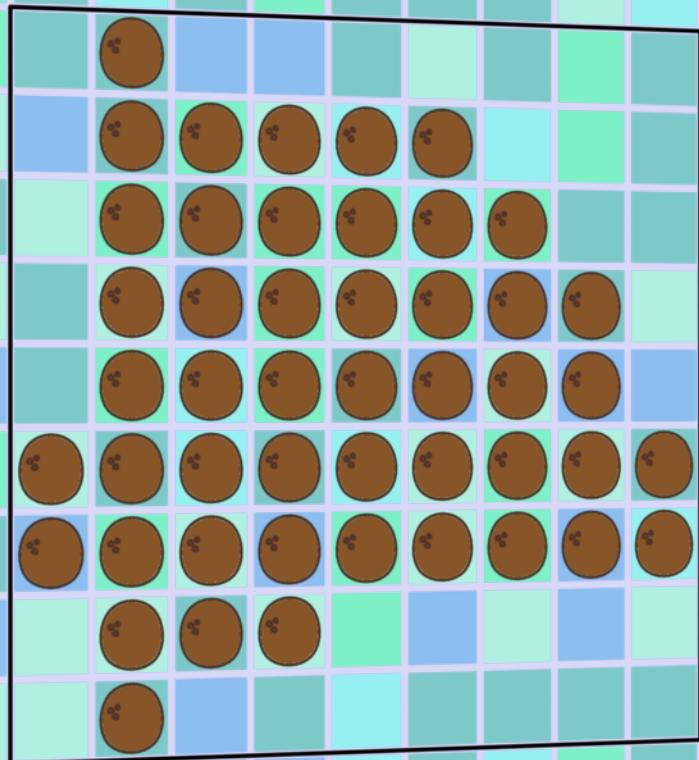
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

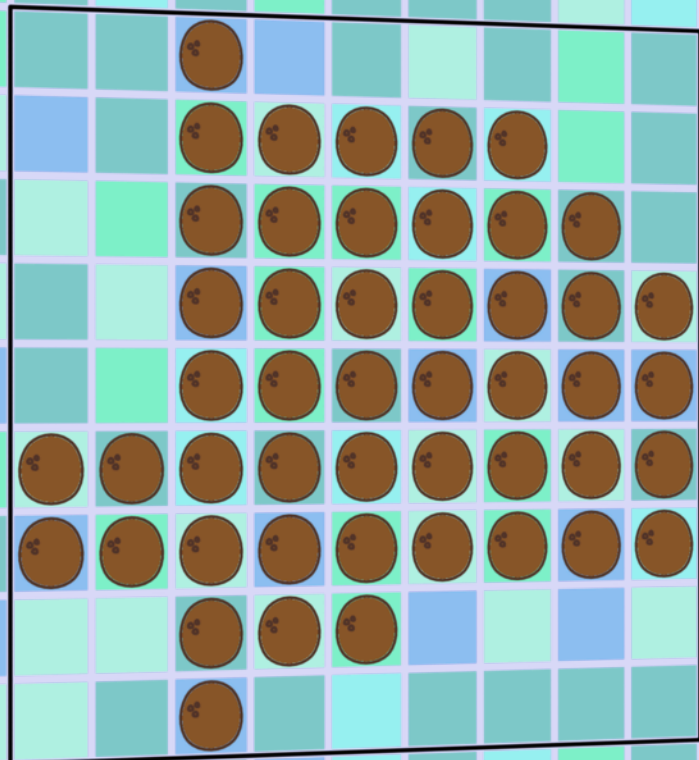
Proof idea. The number of rows and columns of given lengths are invariant.



2 9 7 7 6 6 5 4 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

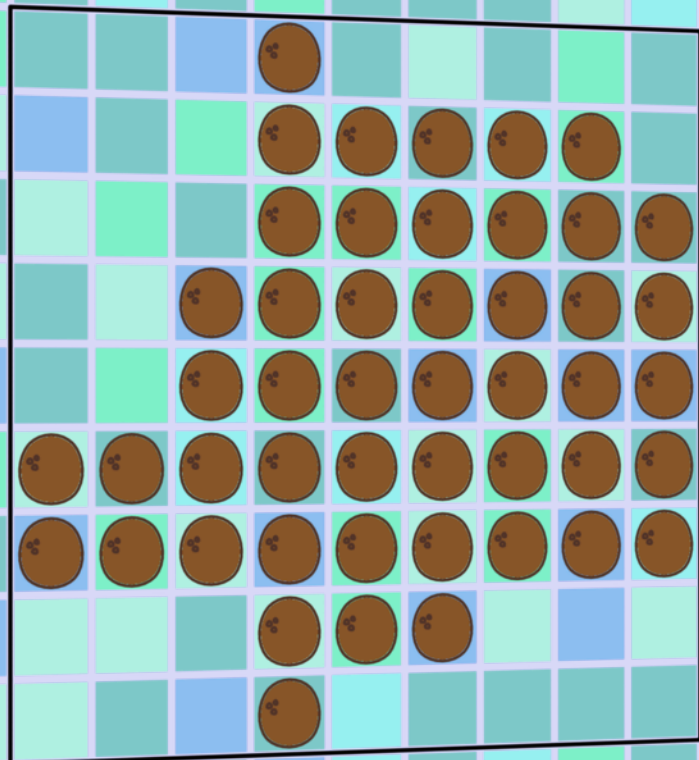
Proof idea. The number of rows and columns of given lengths are invariant.



2 2 9 7 7 6 6 5 4

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

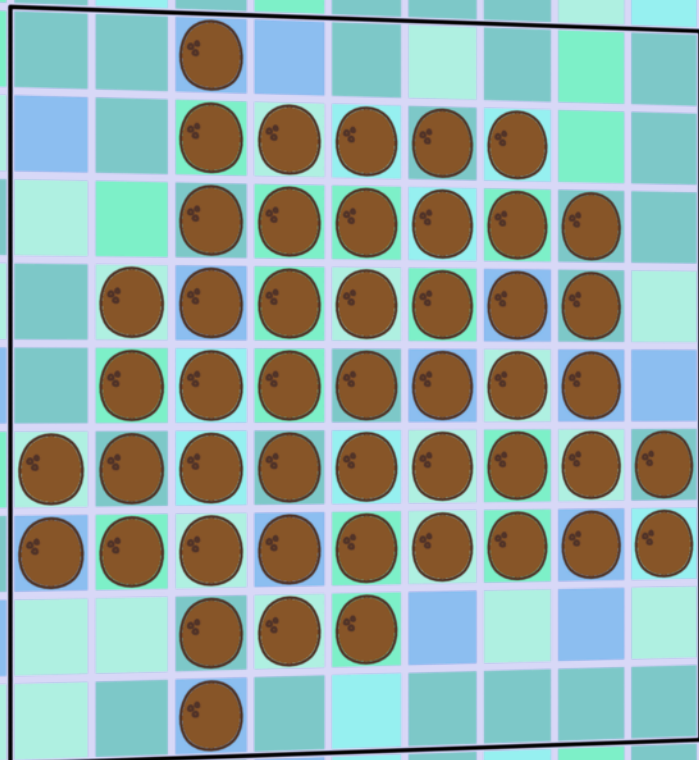
Proof idea. The number of rows and columns of given lengths are invariant.



2 2 4 9 7 7 6 6 5

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

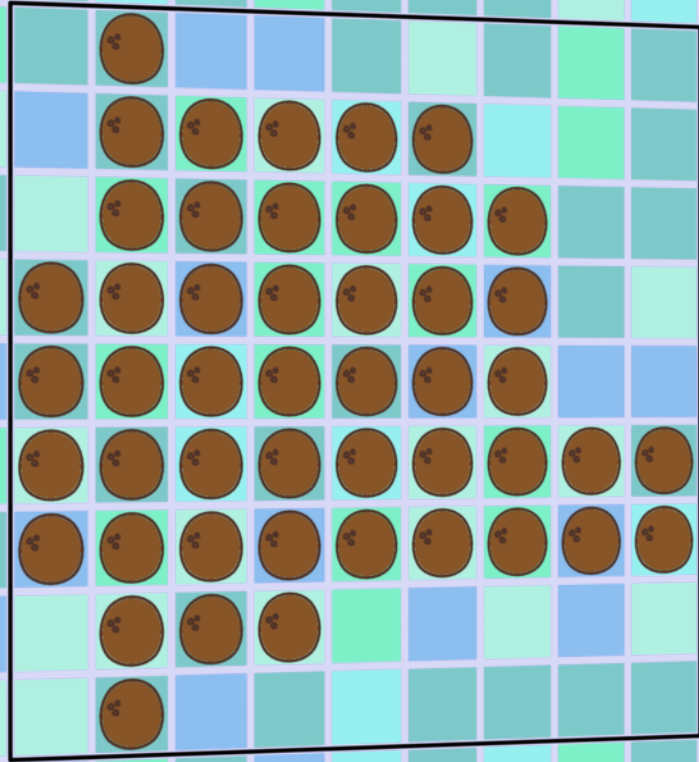
Proof idea. The number of rows and columns of given lengths are invariant.



2 4 9 7 7 6 6 5 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

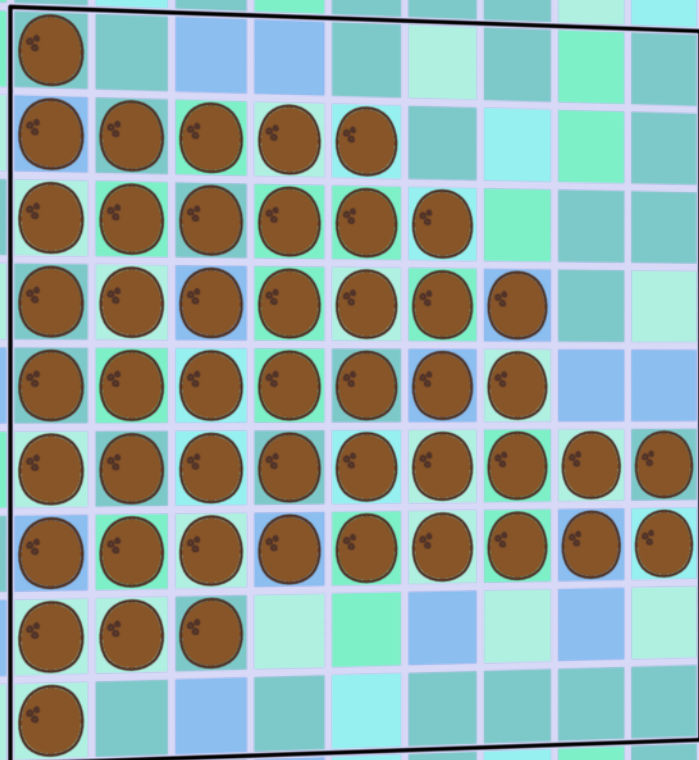
Proof idea. The number of rows and columns of given lengths are invariant.



4 9 7 7 6 6 5 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

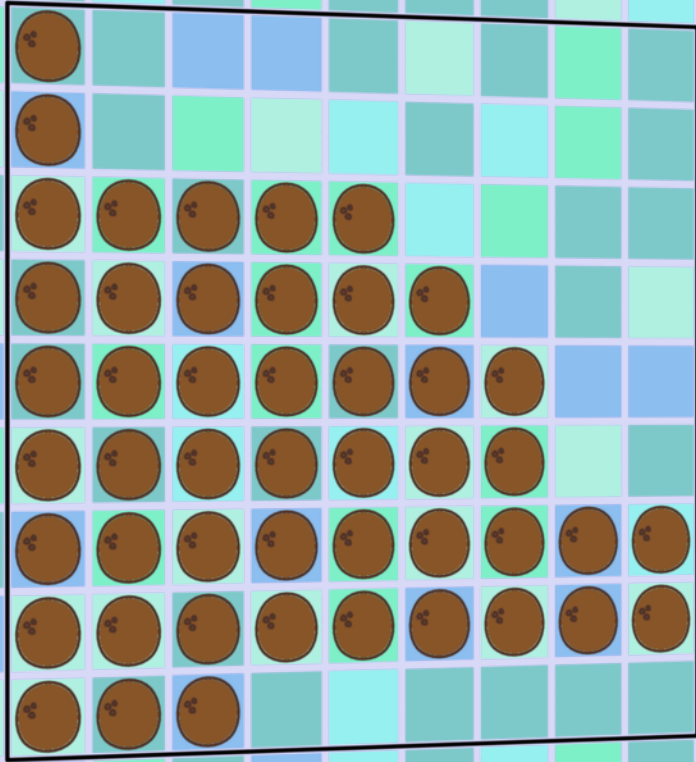
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

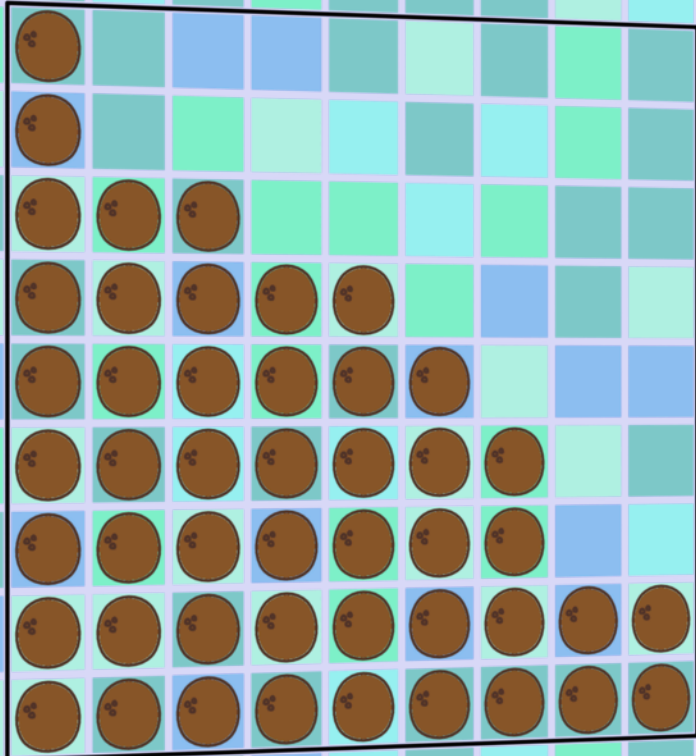
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

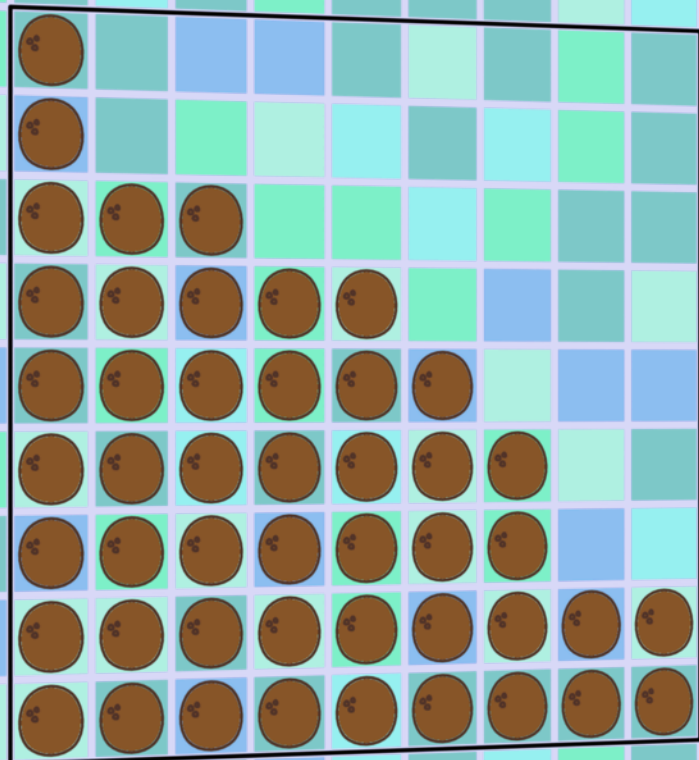
Proof idea. The number of rows and columns of given lengths are invariant.



9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.

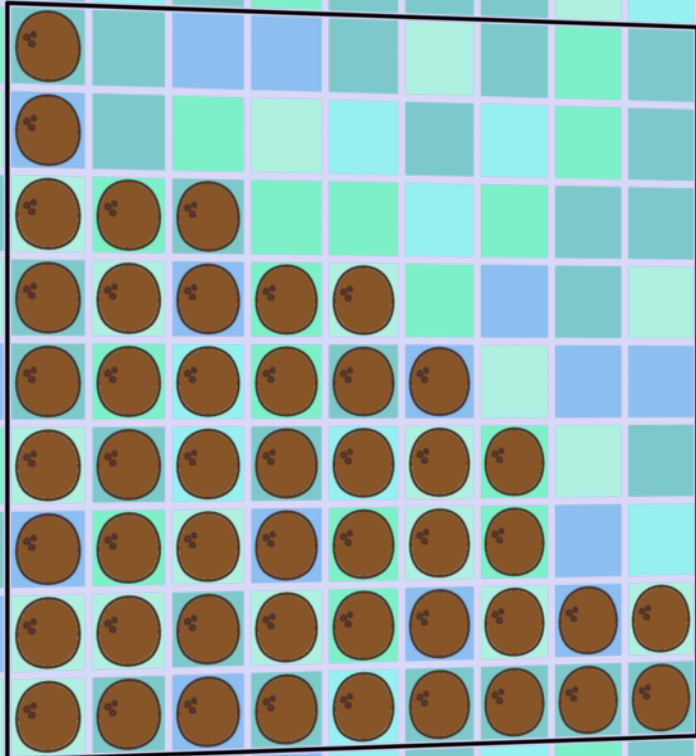


9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.

Lemma. The number of moves is always even.



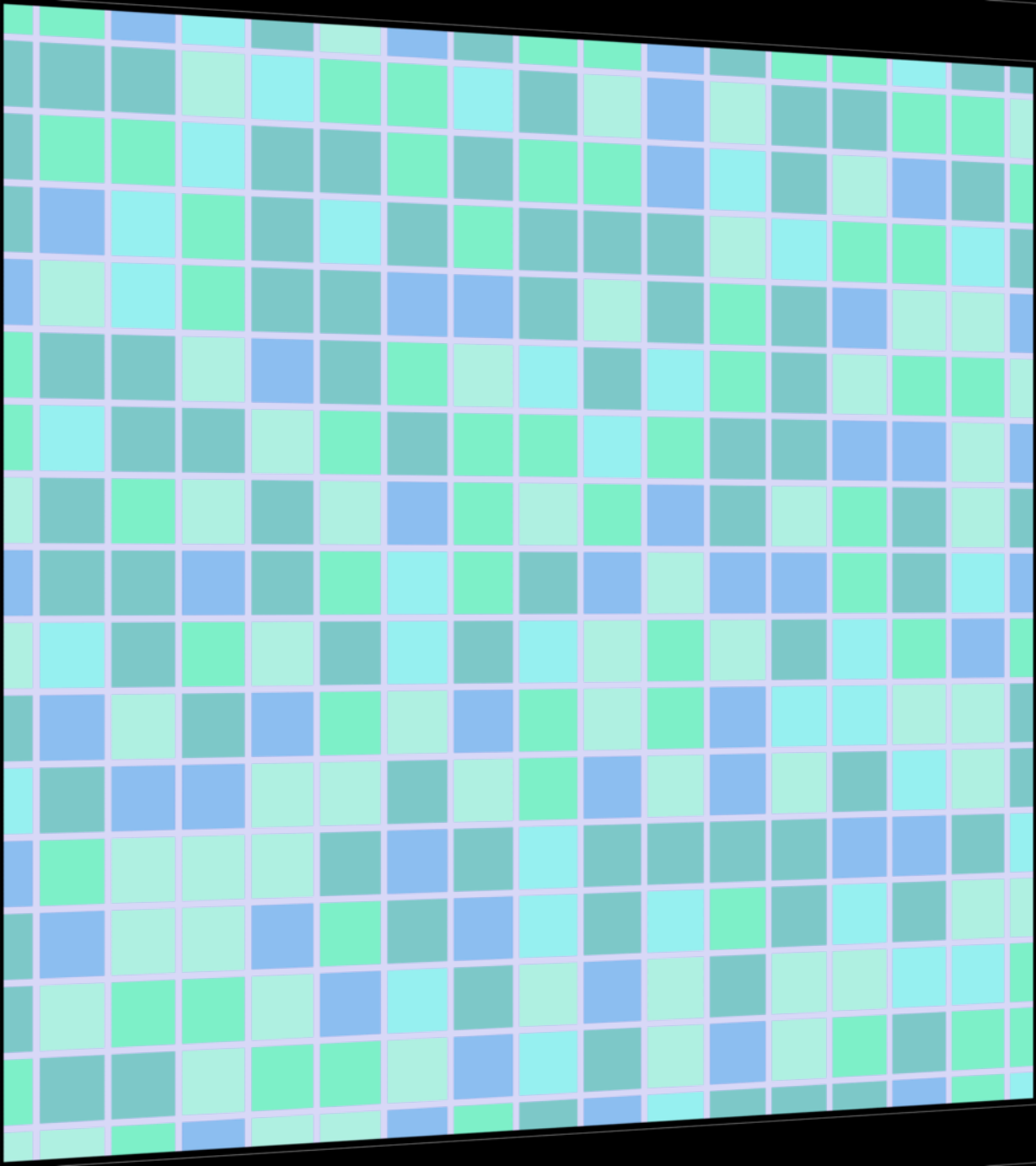
9 7 7 6 6 5 4 2 2

Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

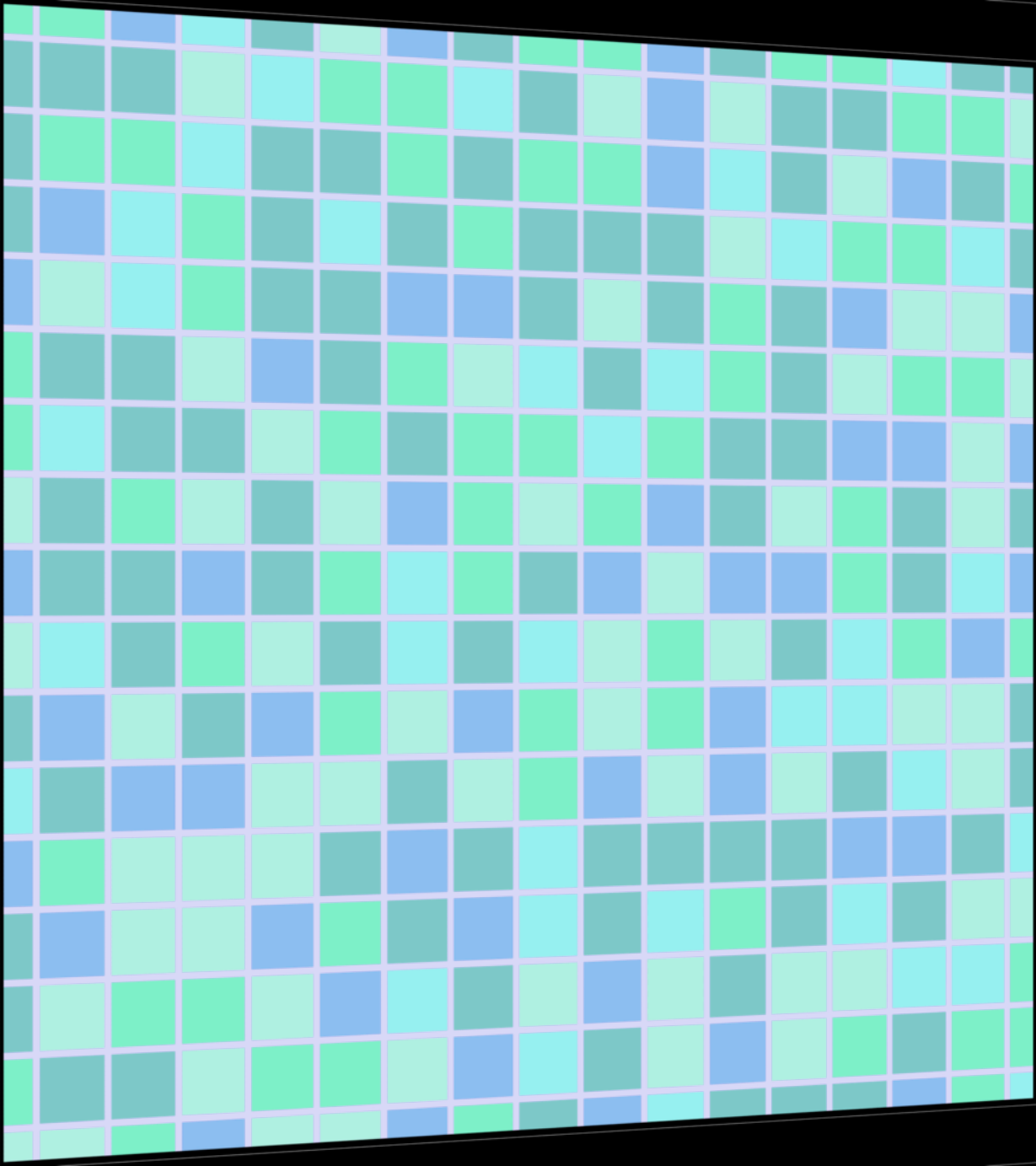
Proof idea. The number of rows and columns of given lengths are invariant.

Lemma. The number of moves is always even.

Proof. Watch the 9s.

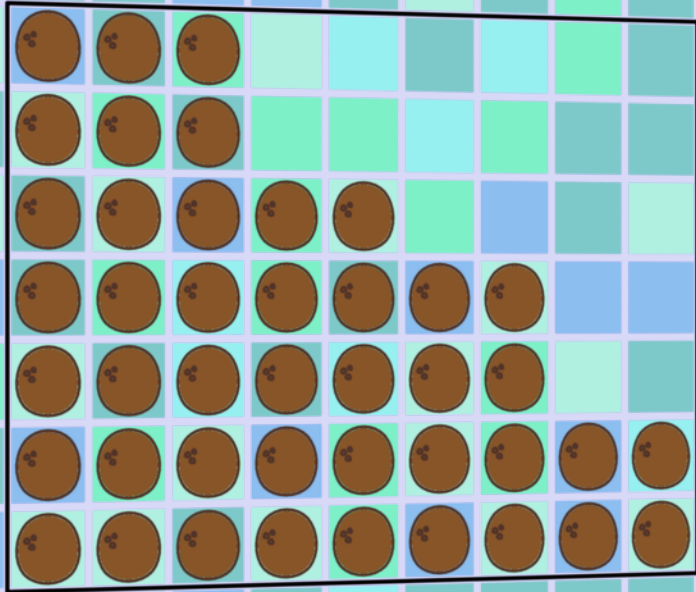


Theorem 2. Every
realizable permutation of
a canonical configuration
is even.



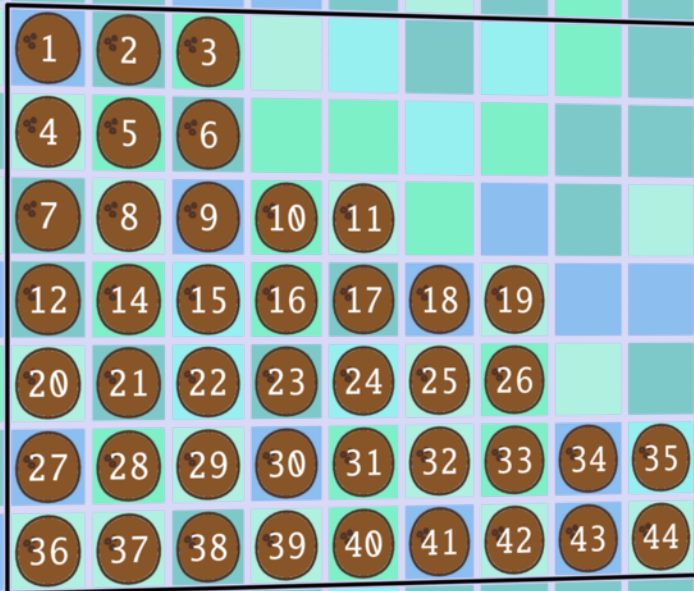
Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.



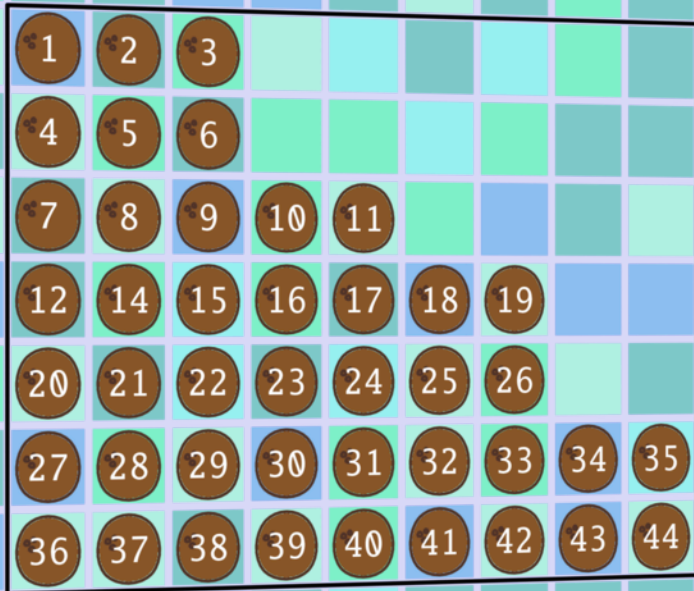
Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.



Theorem 2. Every realizable permutation of a canonical configuration is even.

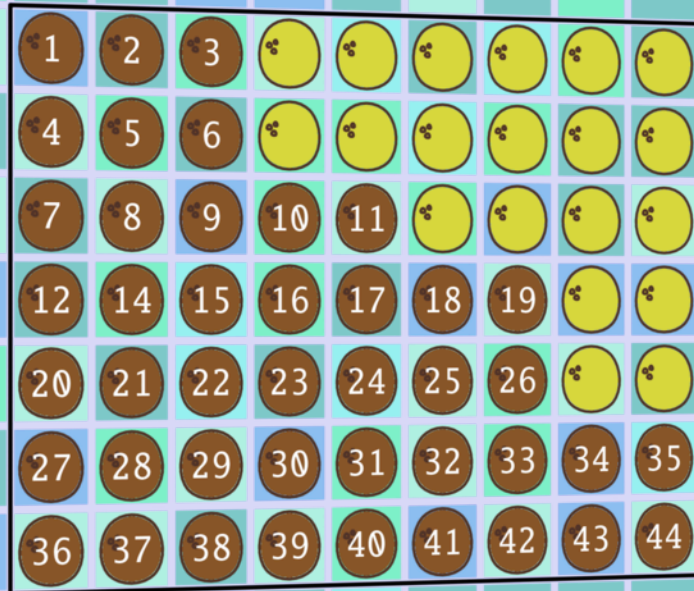
Proof idea. First reason about permutations on all positions, including *empty* ones.



Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.



Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.



Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antnut*.

*6	1	2	3	*1	*2	*3	*4	*5
*12	4	5	6	*7	*8	*9	*10	*11
*16	7	8	9	10	11	*13	*14	*15
*18	12	14	15	16	17	18	19	*17
*20	20	21	22	23	24	25	26	*19
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44

Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.



Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

*11	*12	1	2	3	*7	*8	*9	*10
*15	*16	4	5	6	10	11	*13	*14
*17	*18	7	8	9	16	17	18	19
*19	*20	12	14	15	23	24	25	26
27	28	21	22	31	32	33	34	35
36	28	29	30	40	41	42	43	44
*5	*6	37	38	39	*1	*2	*3	*4

Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.



Theorem 2. Every realizable permutation of a canonical configuration is even.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

Each move is a *cyclic*
permutation on some rows
or columns.



10	11	12	1	2	3	7	8	9
14	15	16	4	5	6	10	11	13
17	18	7	8	9	16	17	18	19
19	20	12	14	15	23	24	25	26
27	20	21	22	31	32	33	34	35
36	28	29	30	40	41	42	43	44
4	5	6	37	38	39	1	2	3

10	11	12	1	2	3	7	8	9
14	15	16	4	5	6	10	11	13
17	18	7	8	9	16	17	18	19
19	20	12	14	15	23	24	25	26
27	20	21	22	31	32	33	34	35
36	28	29	30	40	41	42	43	44
4	5	6	37	38	39	1	2	3

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

*10	*11	*12	*1	*2	*3	*7	*8	*9
*14	*15	*16	*4	*5	*6	10	11	*13
*17	*18	*7	*8	*9	16	17	18	19
*19	*20	12	14	15	23	24	25	26
27	20	21	22	31	32	33	34	35
36	28	29	30	40	41	42	43	44
*4	*5	*6	37	38	39	*1	*2	*3

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

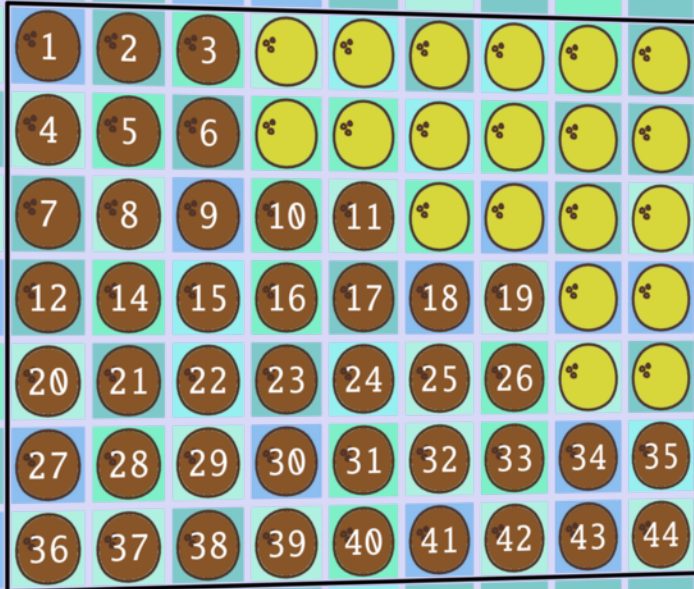
10	11	12	1	2	3	7	8	9
14	15	16	4	5	6	10	11	13
17	18	7	8	9	16	17	18	19
19	20	12	14	15	23	24	25	26
27	20	21	22	31	32	33	34	35
36	28	29	30	40	41	42	43	44
4	5	6	37	38	39	1	2	3

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

It could still be that the permutation of *only the coconuts* is odd!

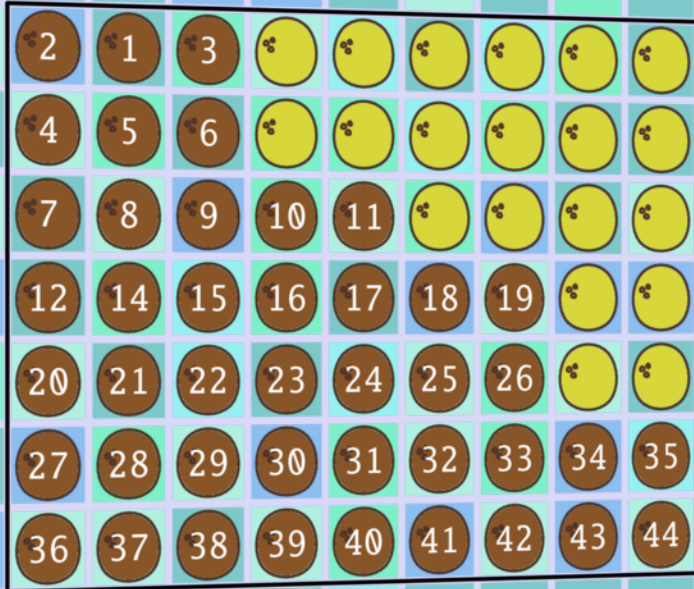


Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

It could still be that the permutation of *only the coconuts* is odd!

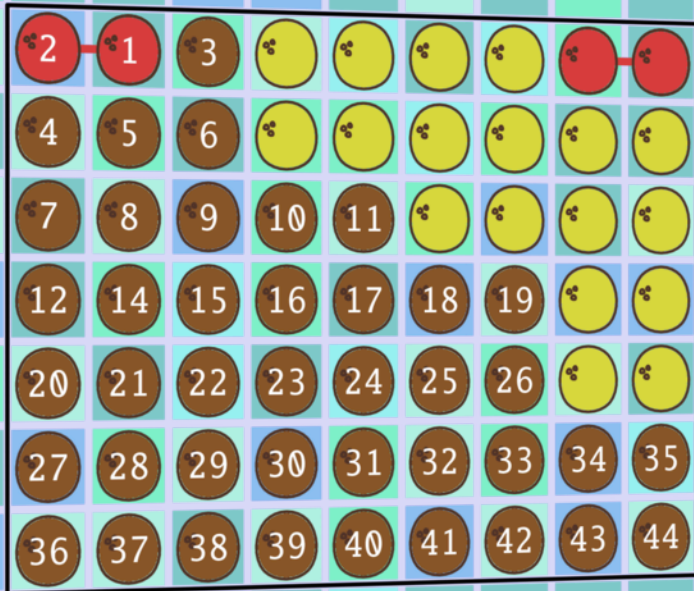


Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

It could still be that the permutation of *only the coconuts* is odd!

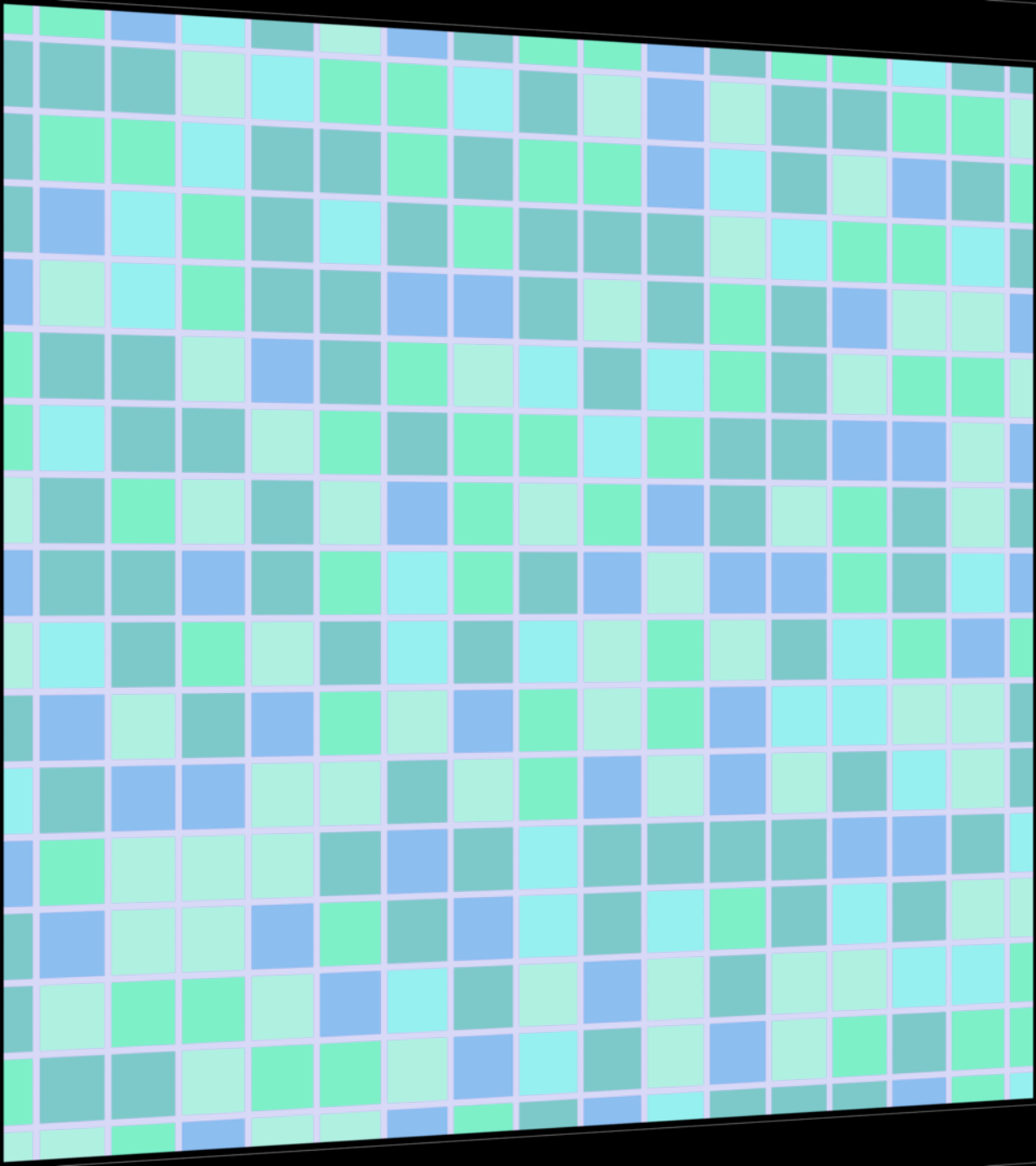


Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

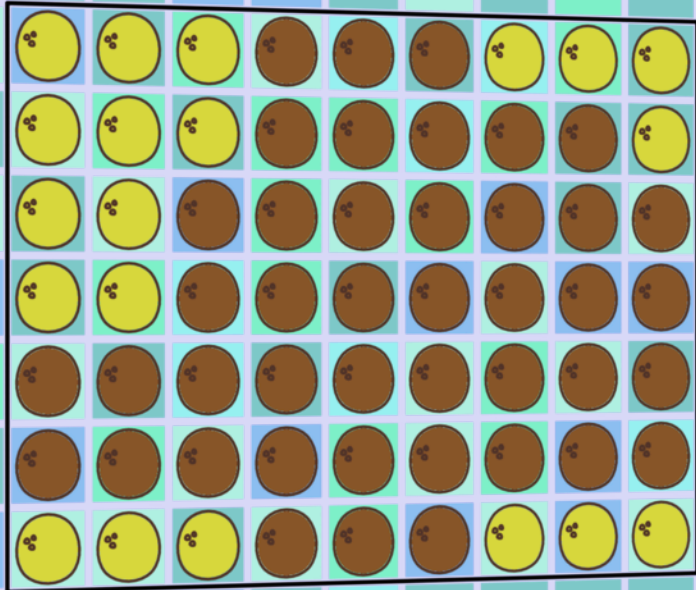
The final permutation on *all coconuts and antinuts* is even.

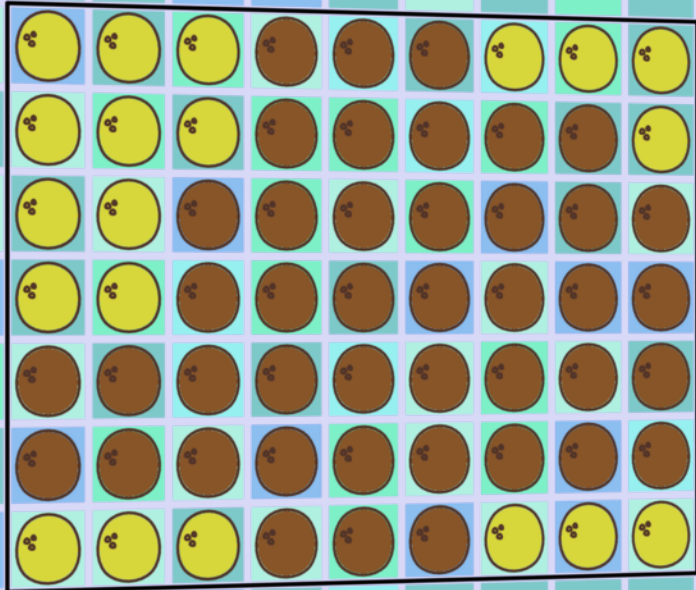
It could still be that the permutation of *only the coconuts* is odd!



Take a closer look at the antinuts.

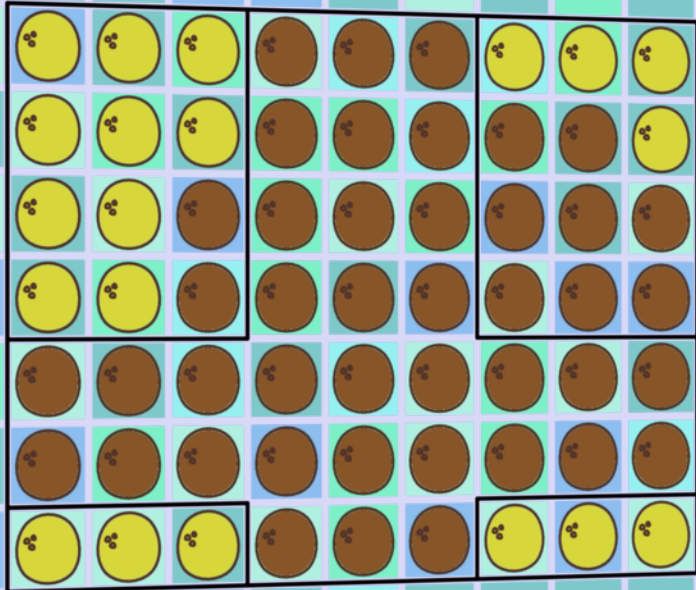
Take a closer look at the antinuts.





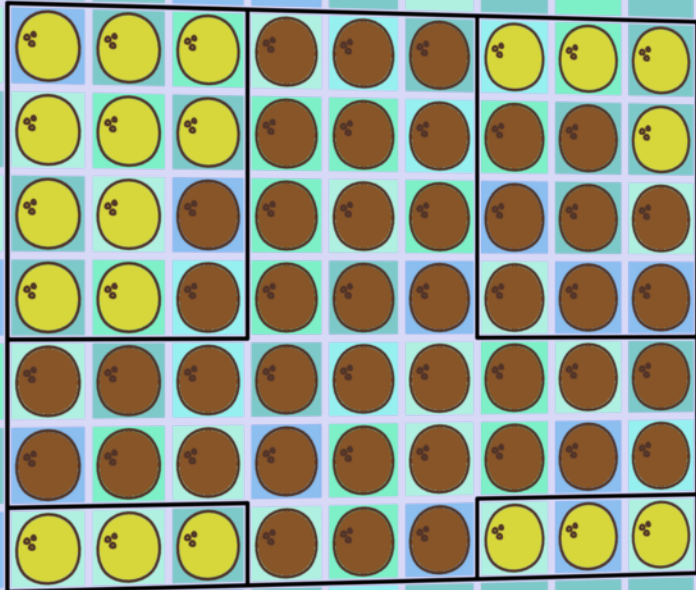
Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.



Take a closer look at the antinuts.

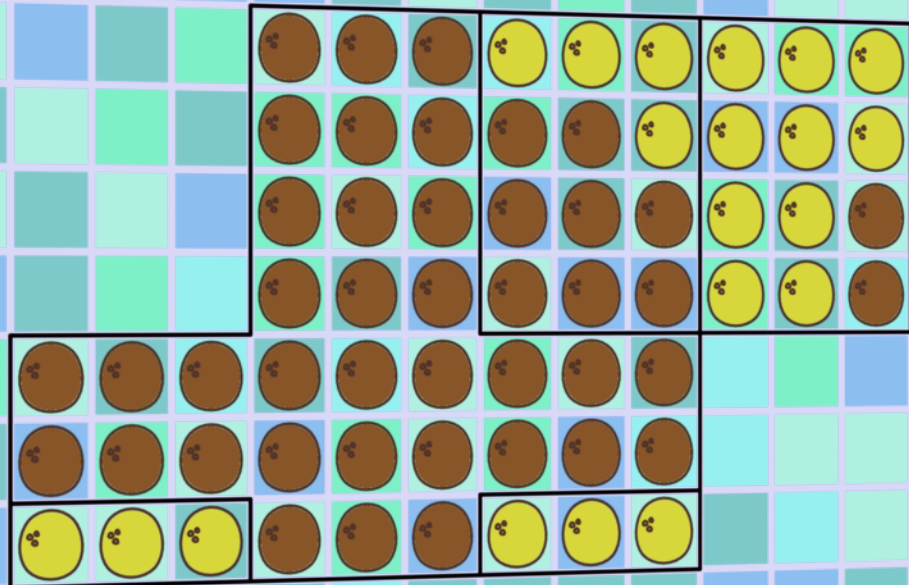
The full rows and columns separate them into four quadrants.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

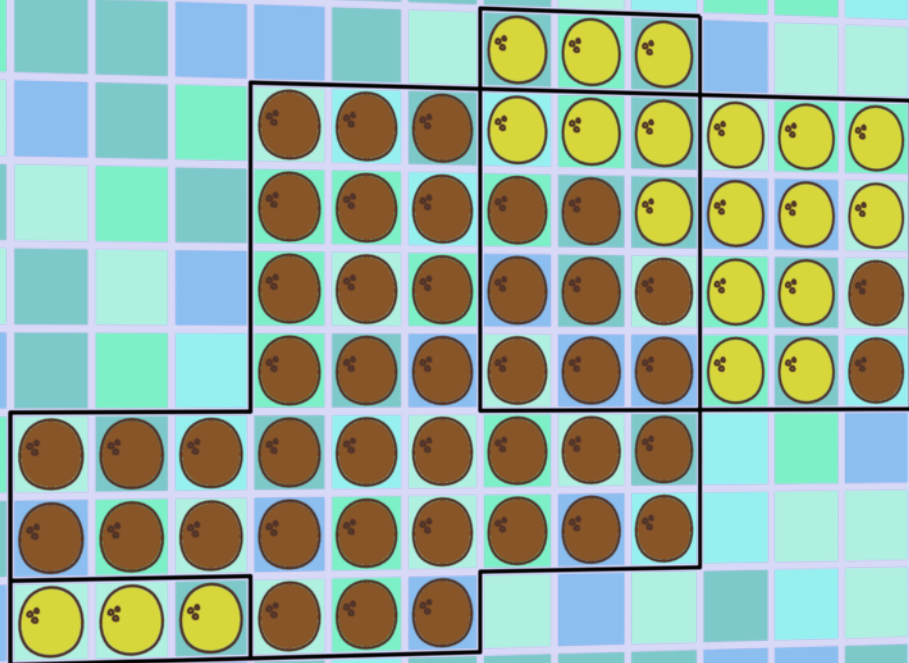
Put them together.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

Center the view and forget about the full rows and columns.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

Center the view and forget about the full rows and columns.

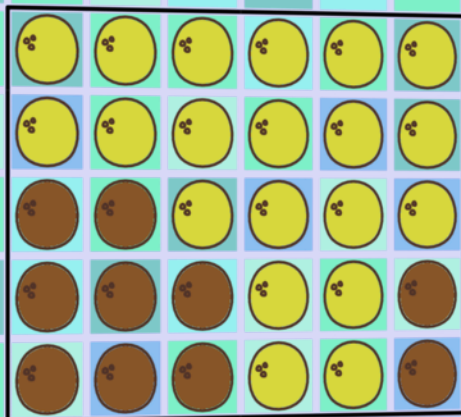


Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

Center the view and forget about the full rows and columns.



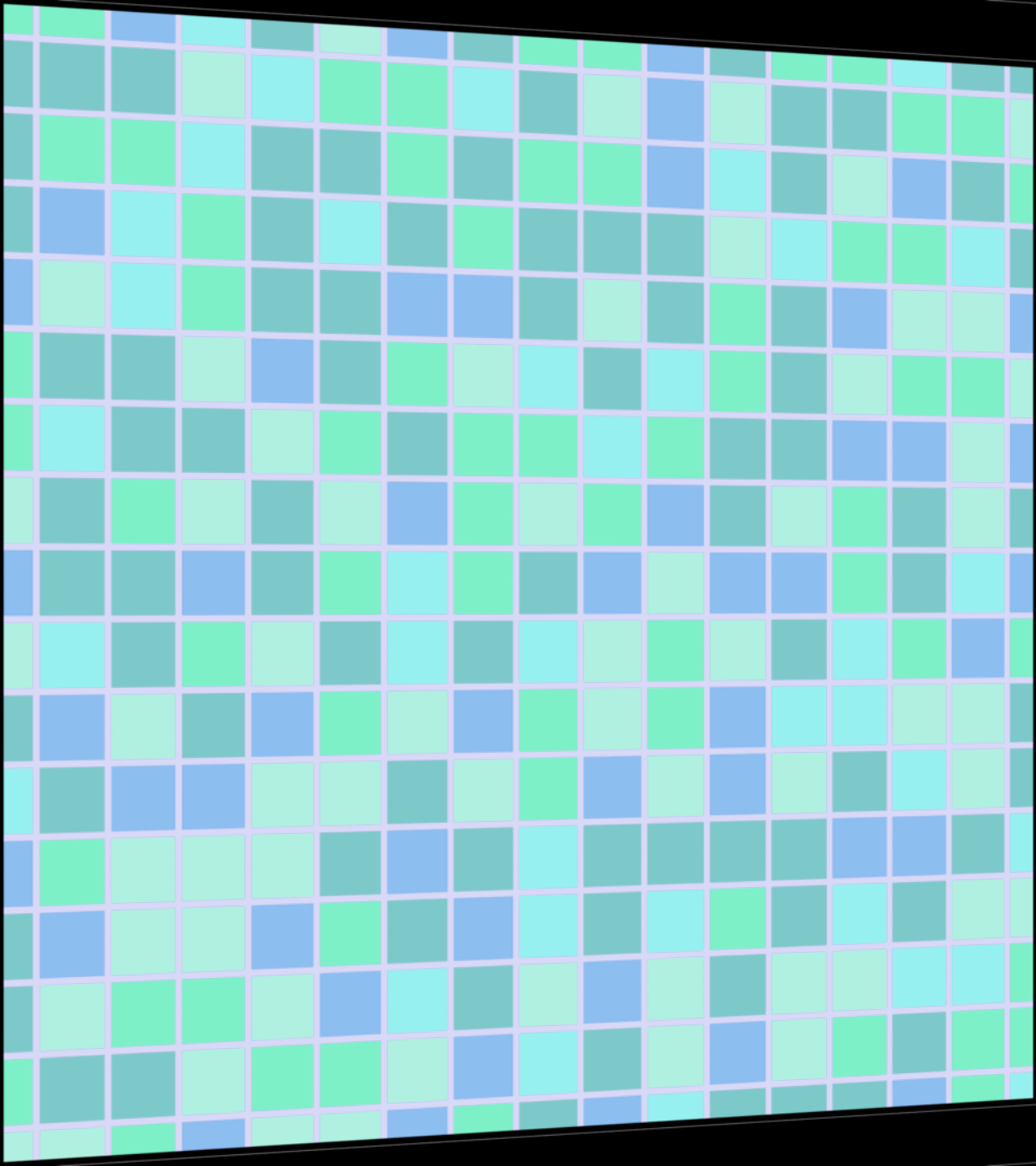
Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

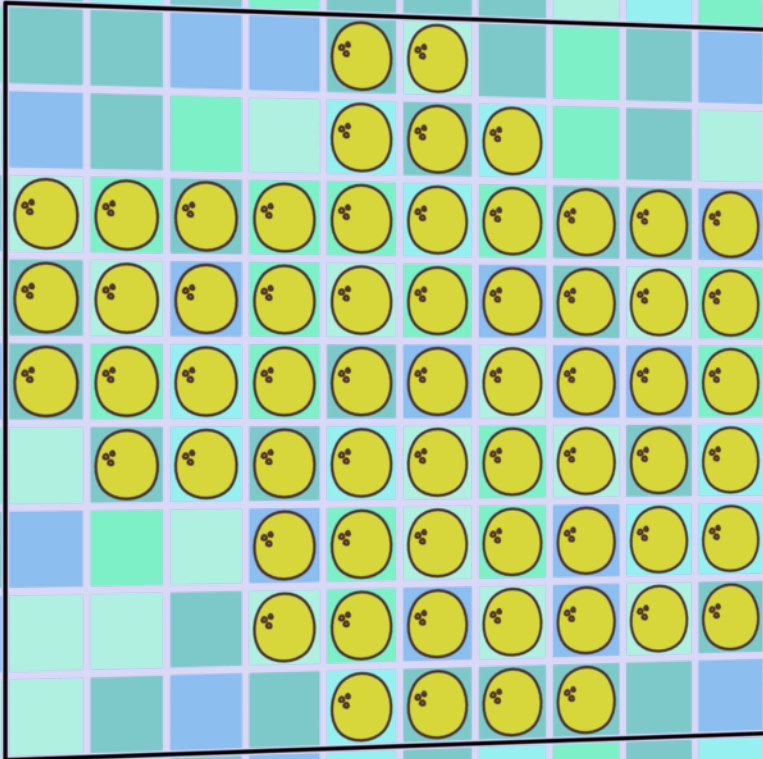
Center the view and forget about the full rows and columns.

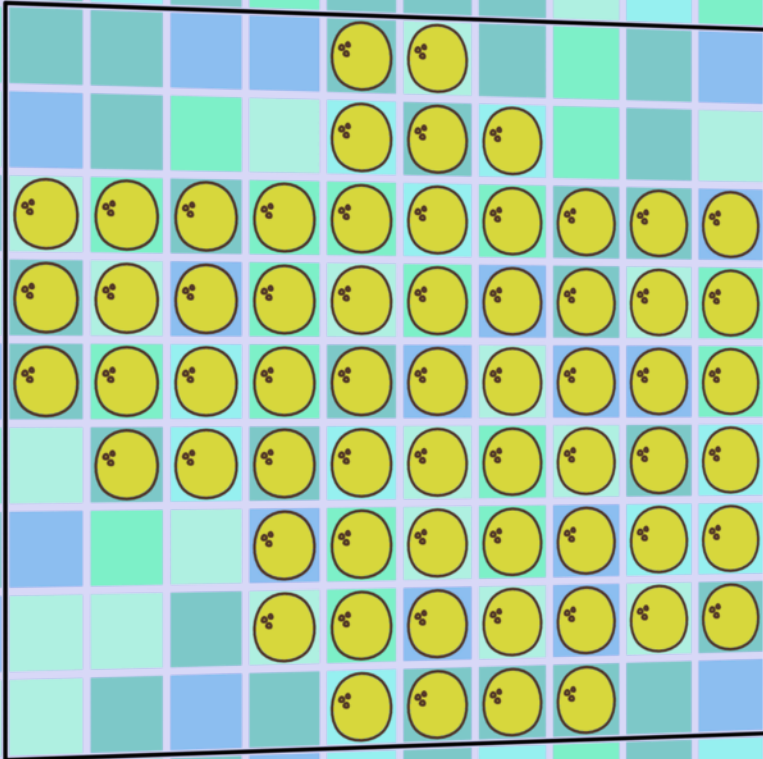
It's the *antinut-centric* view of a configuration.



We can play a dual game on
the antinuts!

We can play a dual game on
the antinuts!



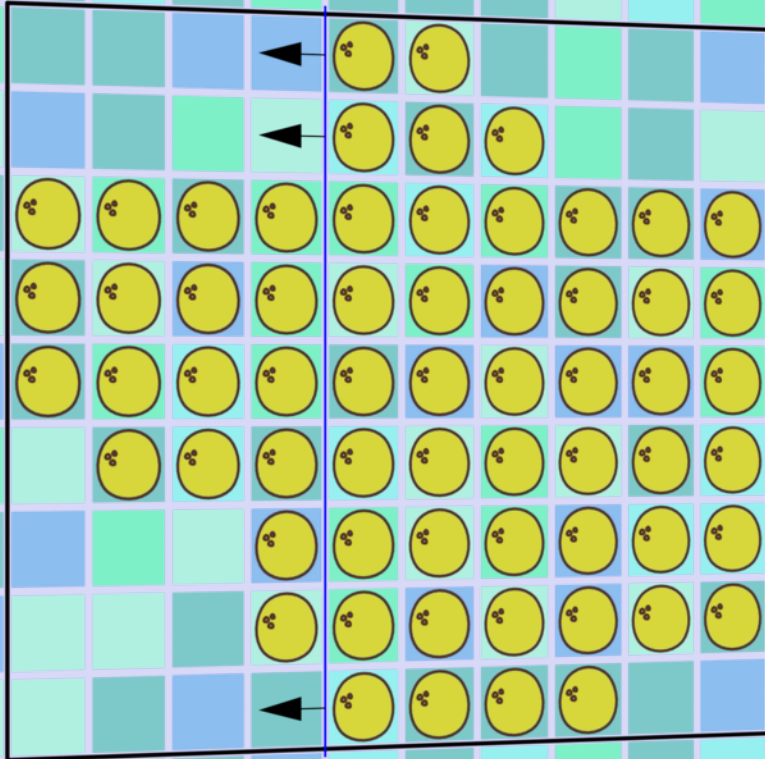


We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.

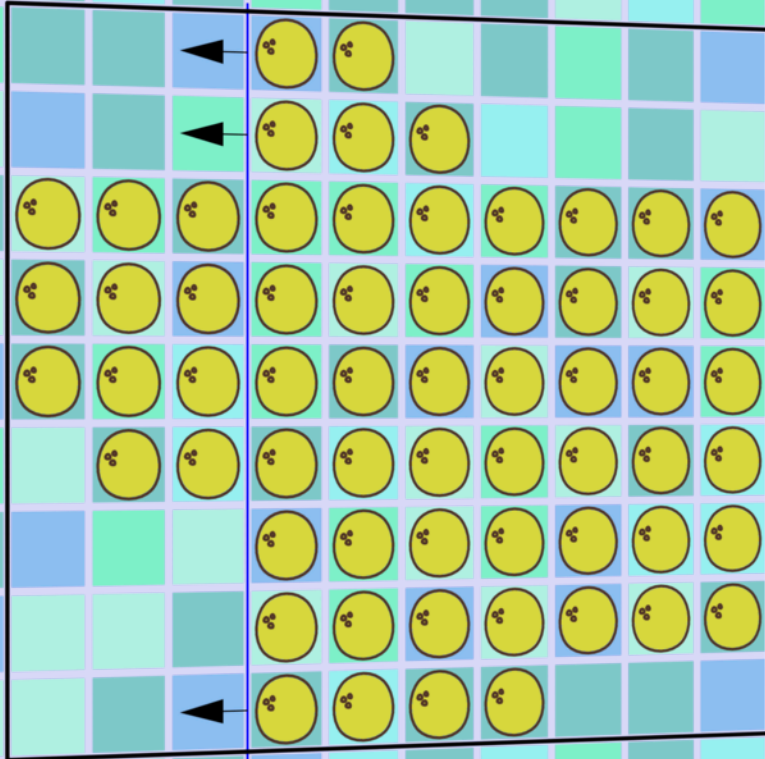
We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.



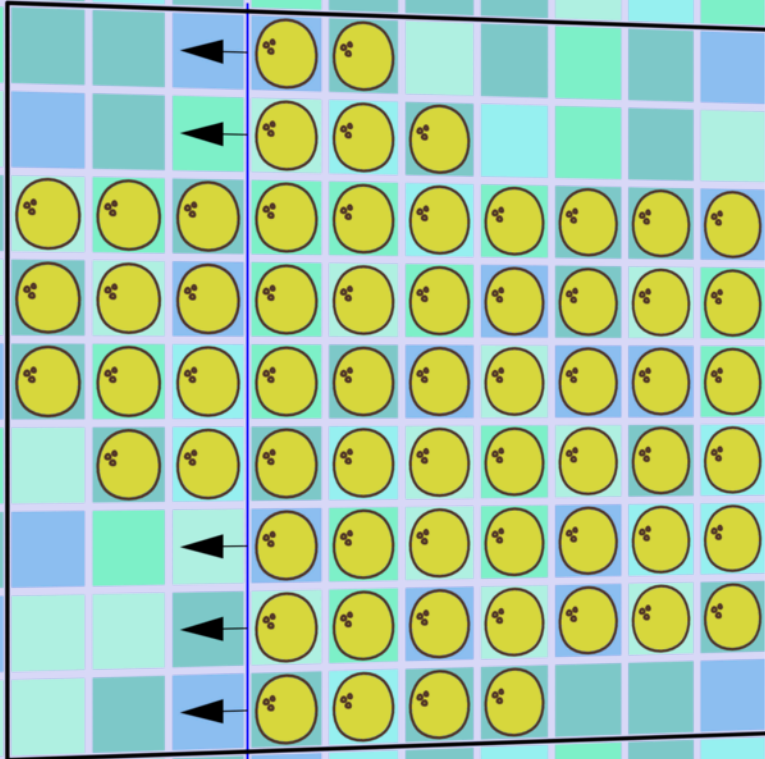
We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.



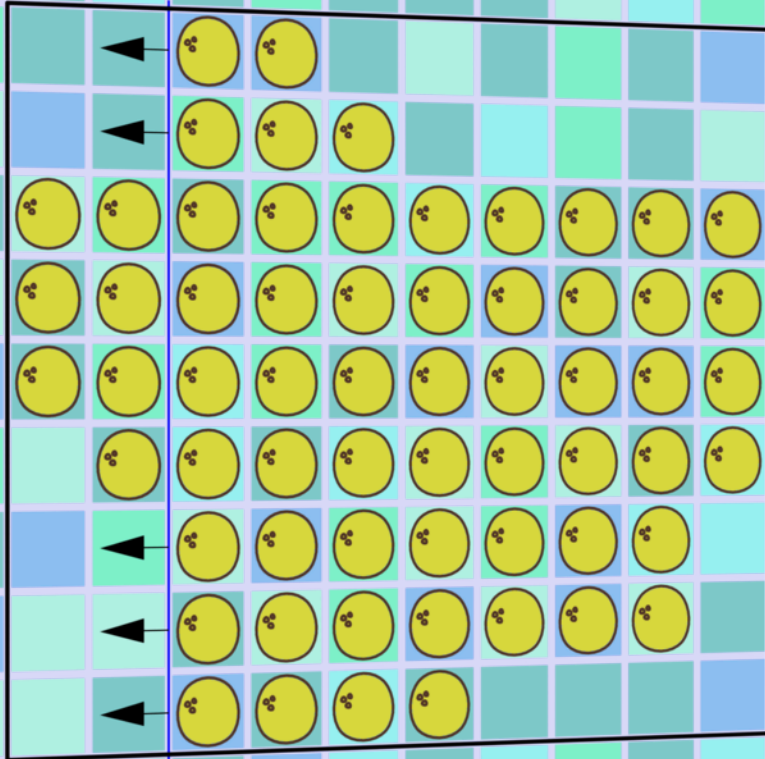
We can play a dual game on the antinuts!

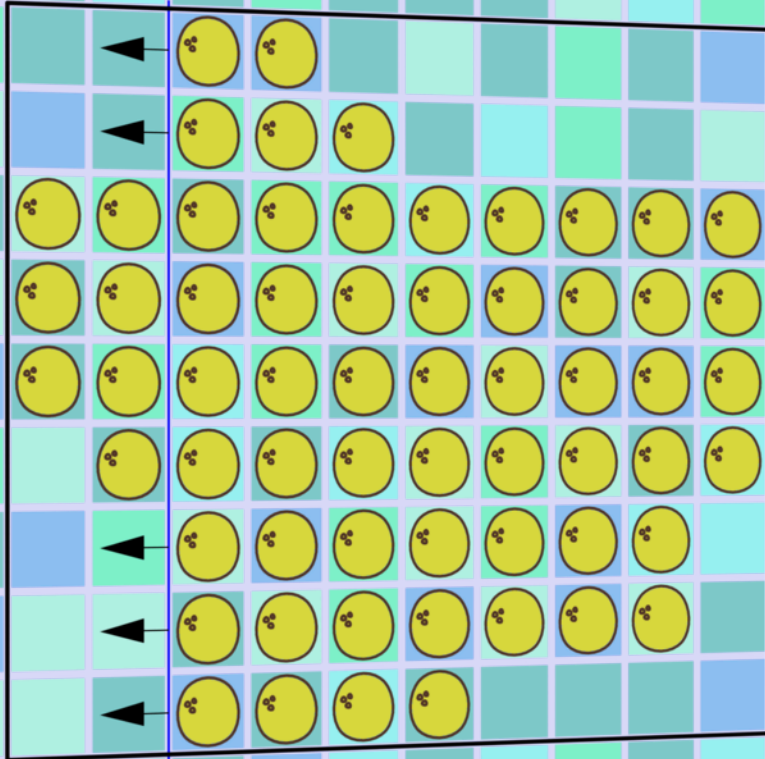
Pushes in the primal corresponds to *pulls* in the dual.



We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.

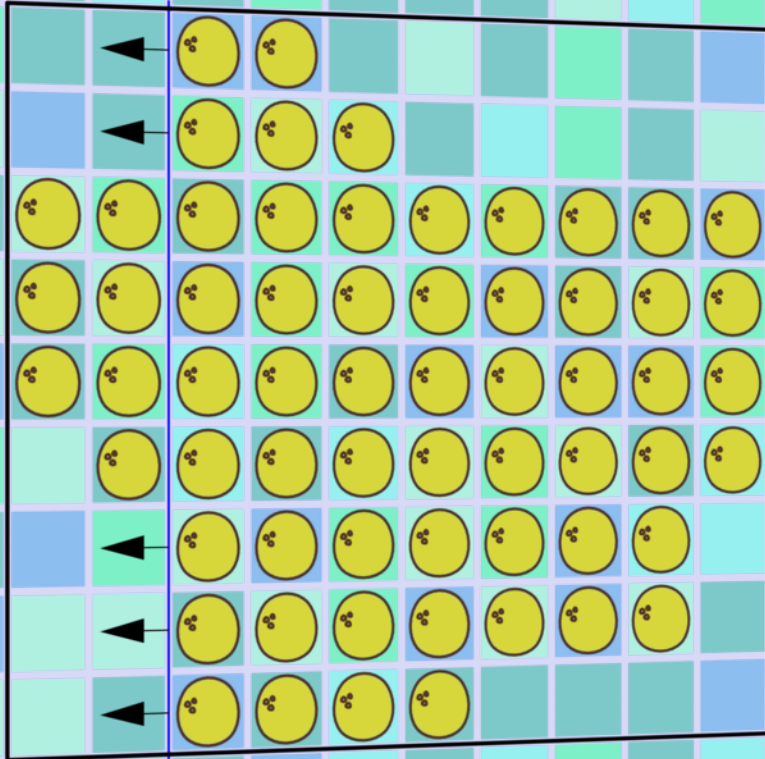




We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.

Most statements about the primal game have dual analogues.

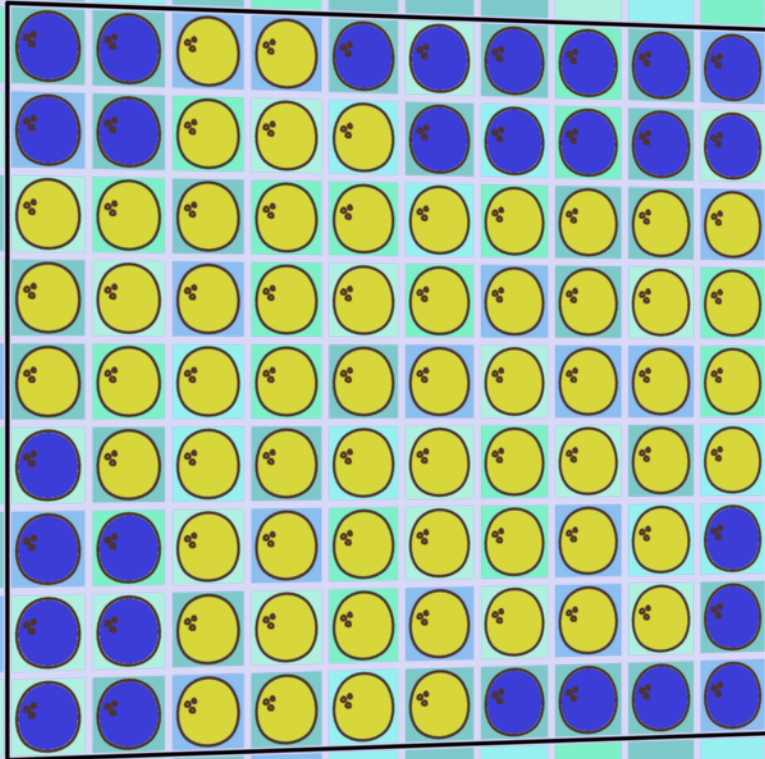


We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.

Most statements about the primal game have dual analogues.

In particular, every achievable permutation on antinuts and *anti-antinuts* is even.



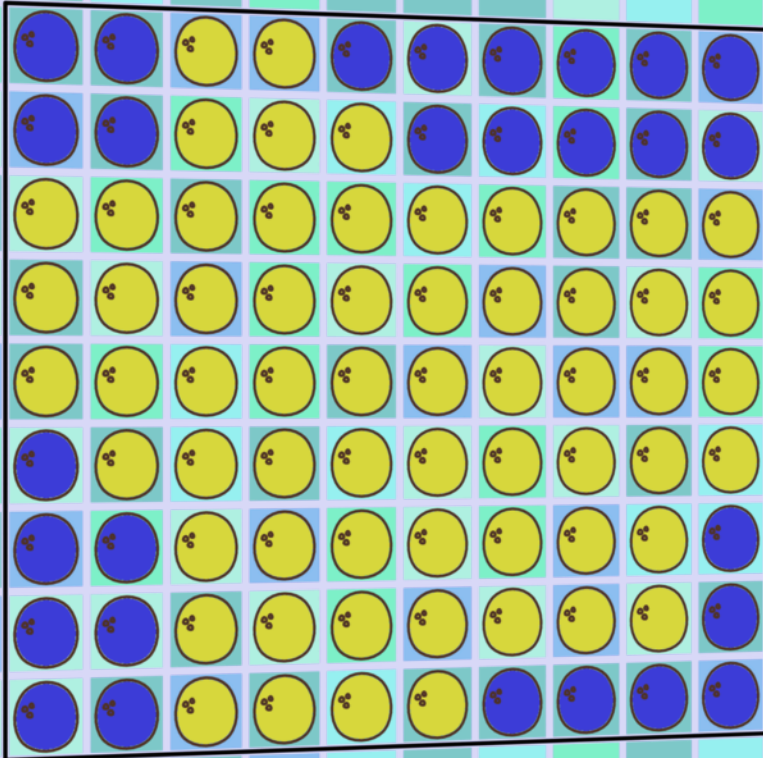
We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.

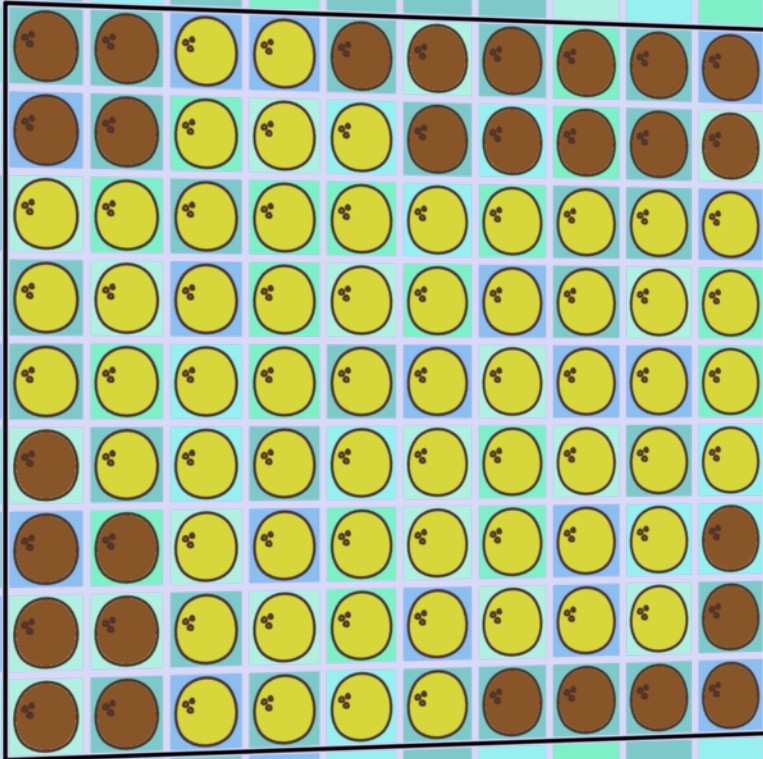
Most statements about the primal game have dual analogues.

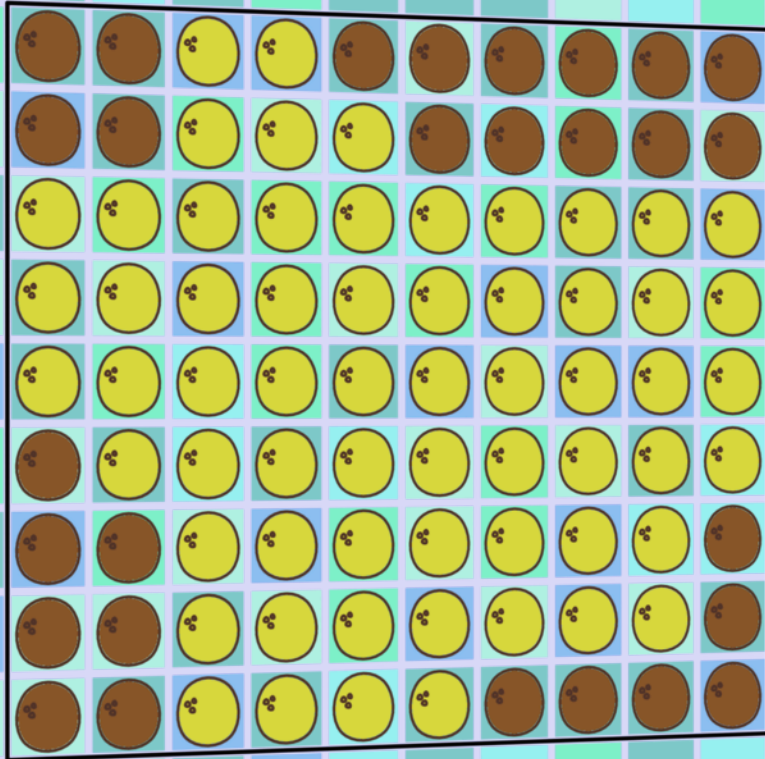
In particular, every achievable permutation on antinuts and *anti-antinuts* is even.

Of course, anti-antinuts
are just our original
coconuts...



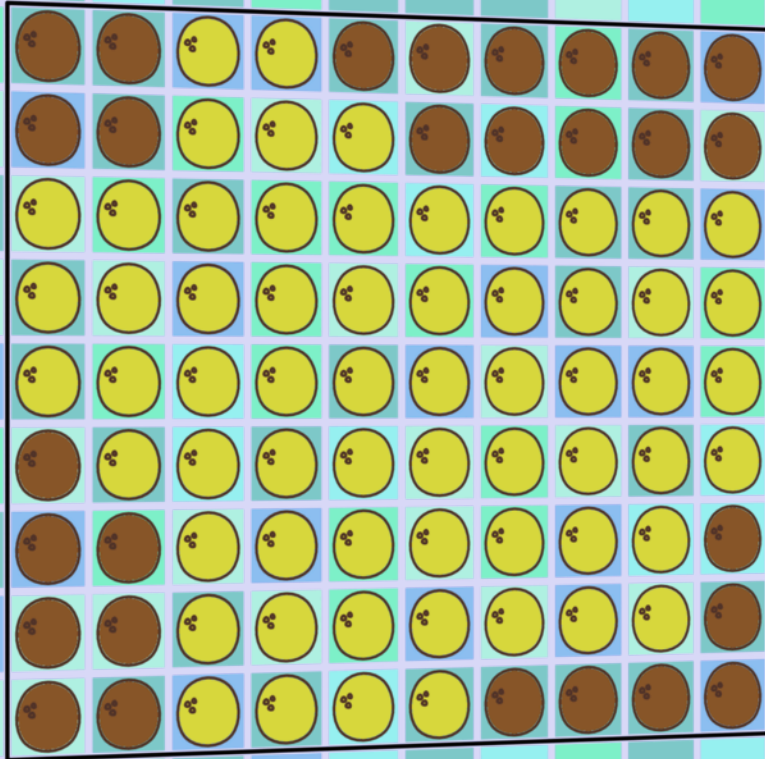
Of course, anti-antinuts
are just our original
coconuts...





Of course, anti-antinuts
are just our original
coconuts...

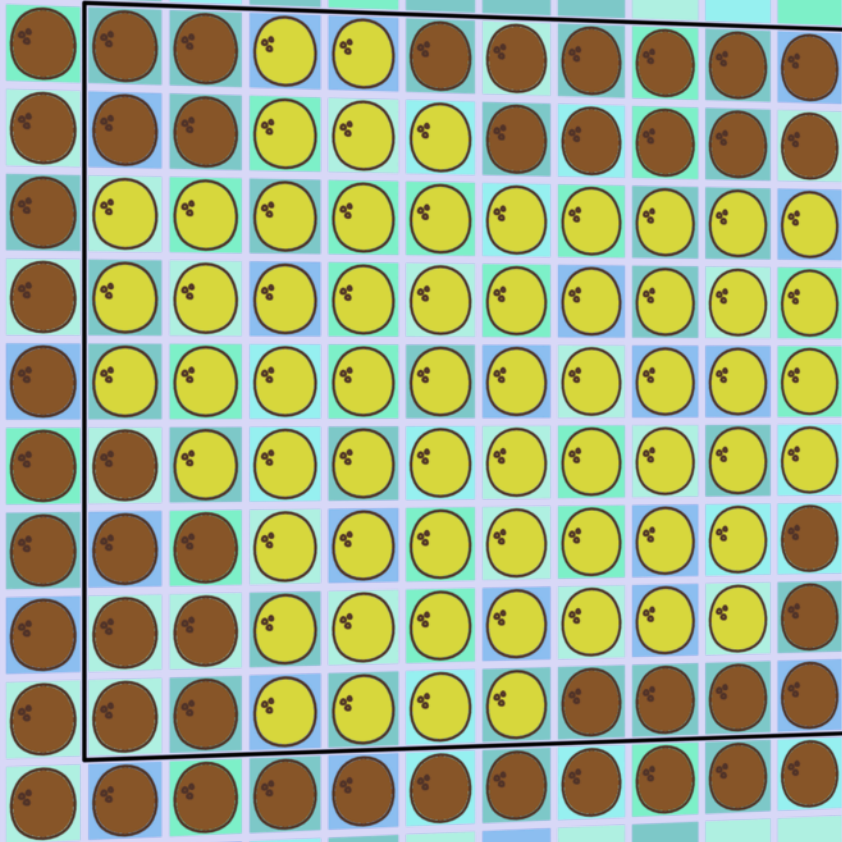
...but not *all* of our
original coconuts!



Of course, anti-antinuts
are just our original
coconuts...

...but not *all* of our
original coconuts!

There must have been
at least one full row
and column outside the
bounding box of the dual
game.

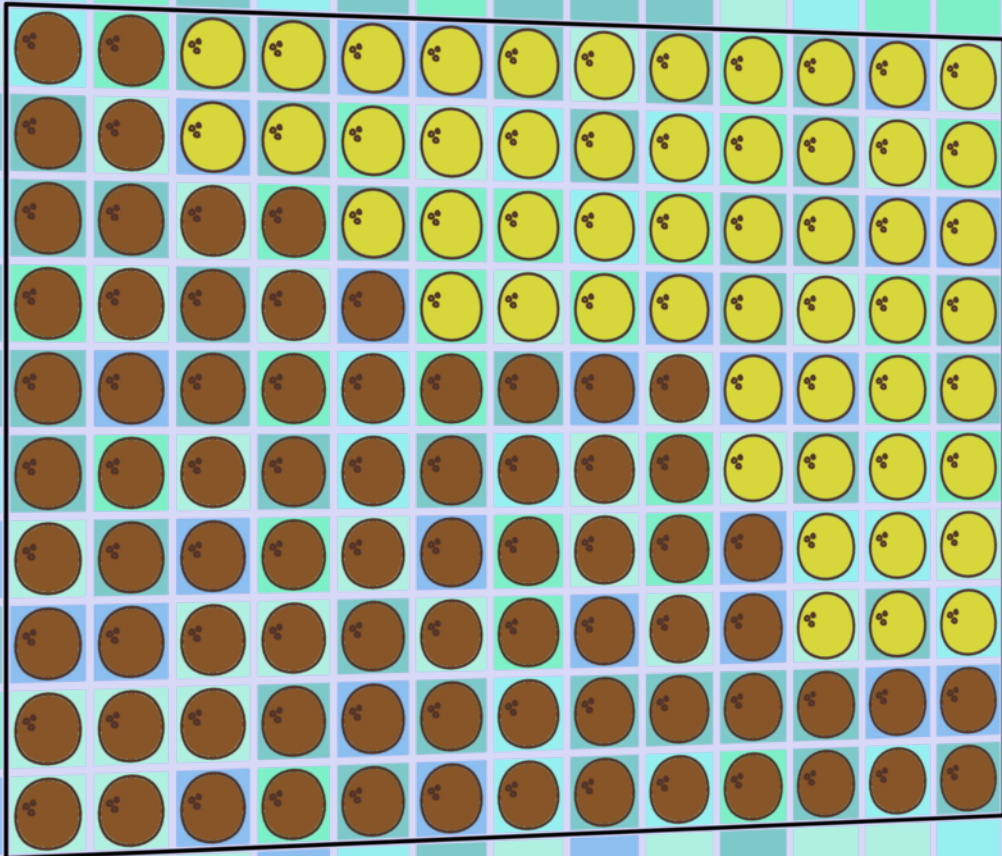


Of course, anti-antinuts
are just our original
coconuts...

...but not *all* of our
original coconuts!

There must have been
at least one full row
and column outside the
bounding box of the dual
game.

Proof by induction.



Proof by induction.



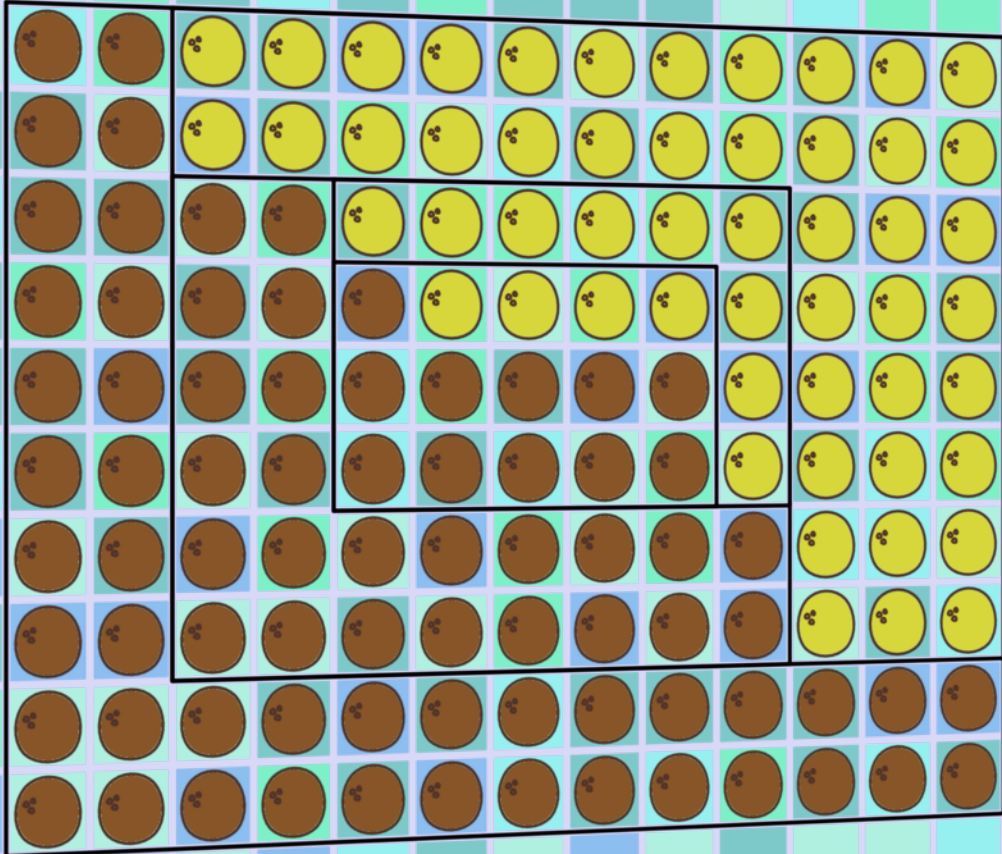
Proof by induction.



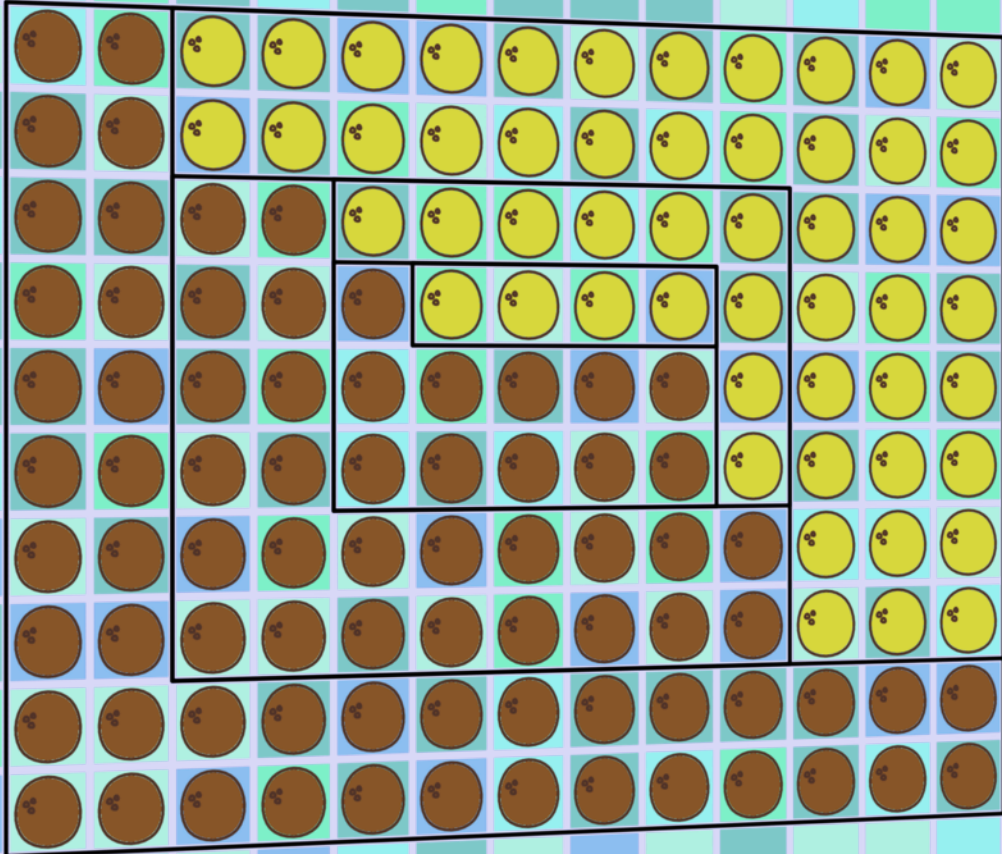
Proof by induction.

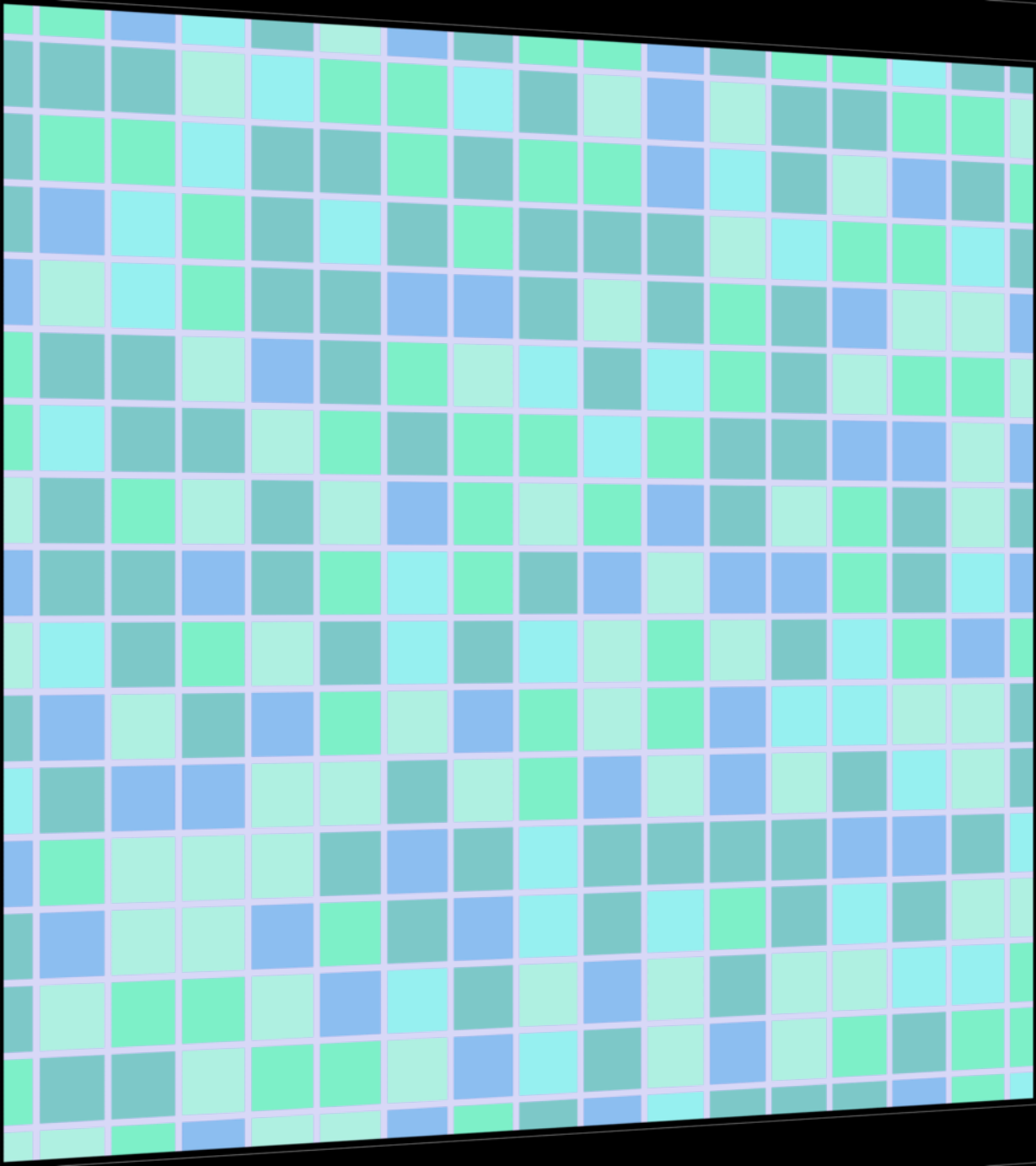


Proof by induction.




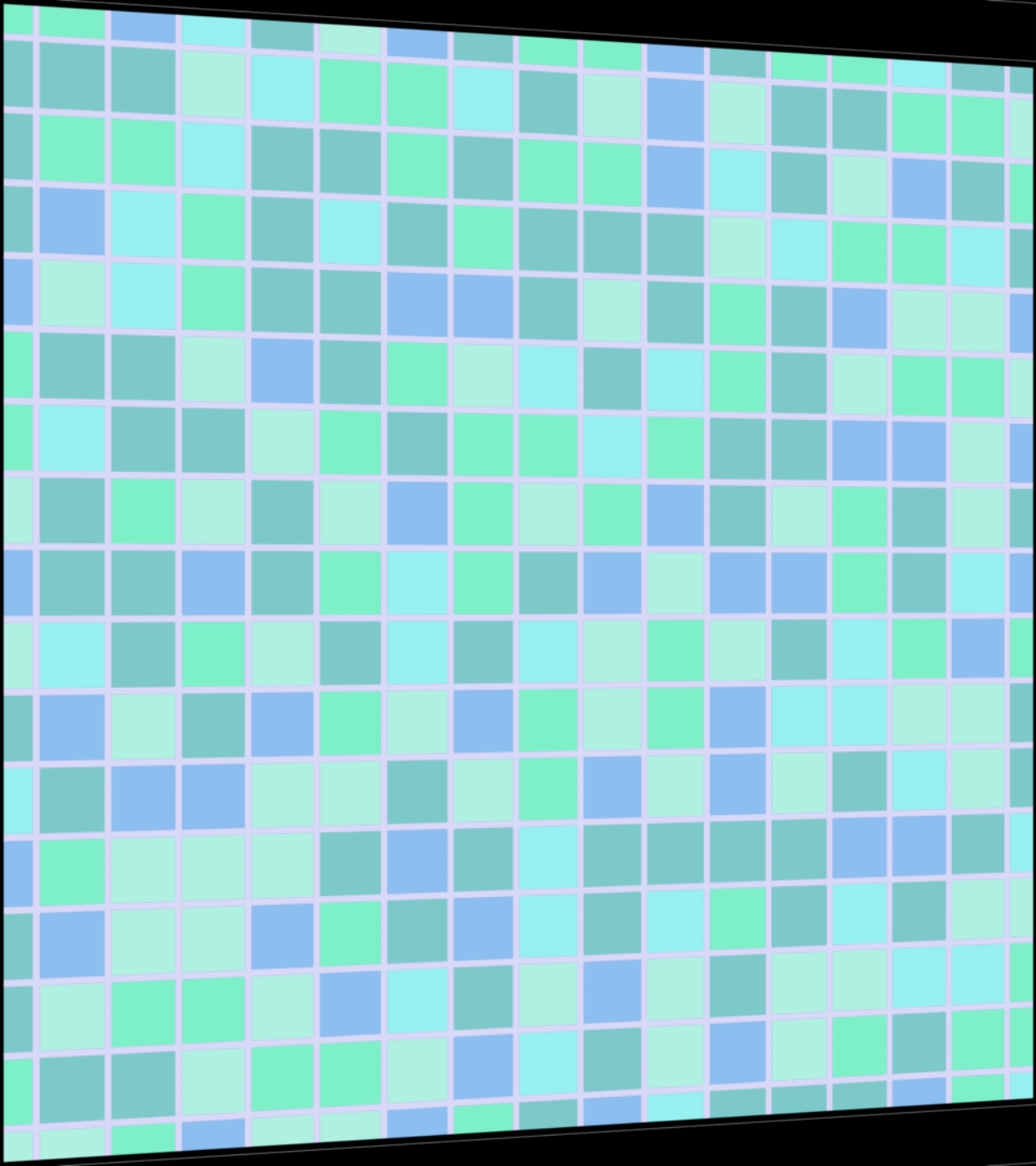
Proof by induction.





Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

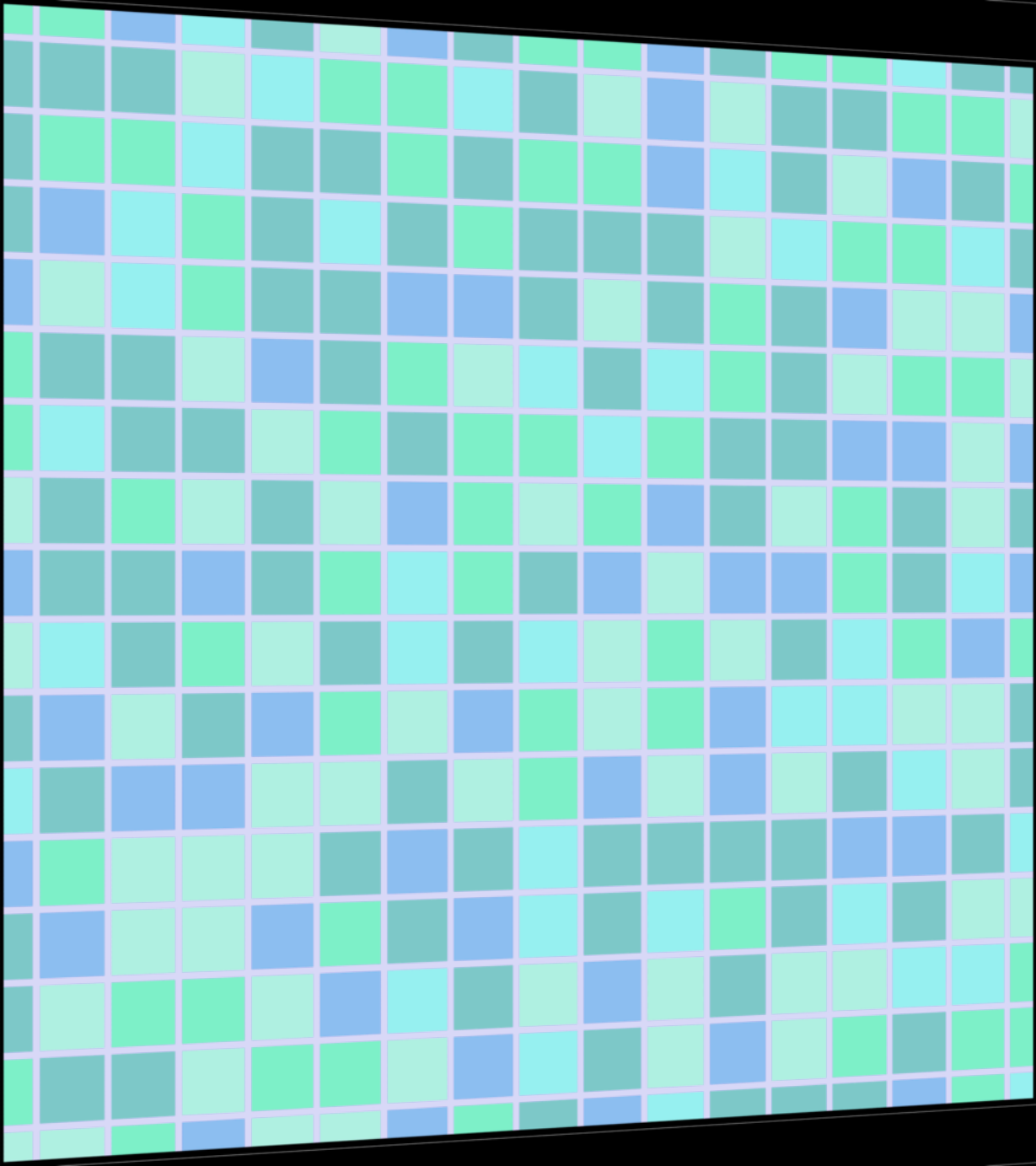


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape:



Proof idea. Cover by cycles that reach all coconuts.

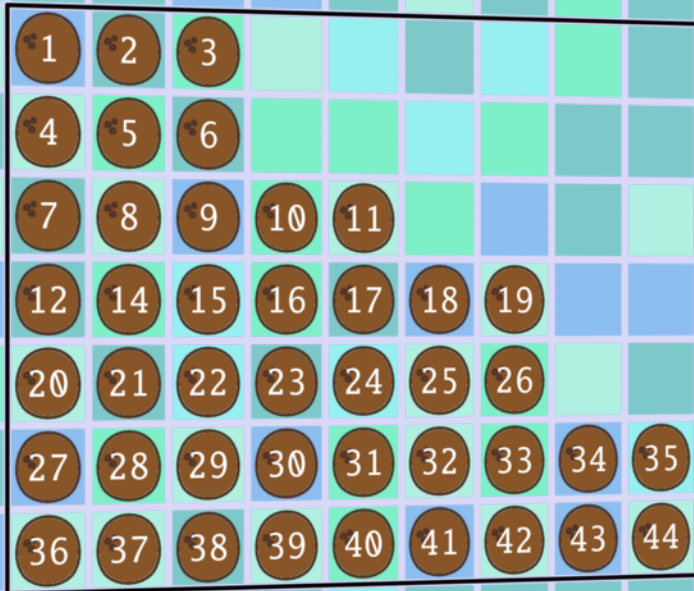


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape:



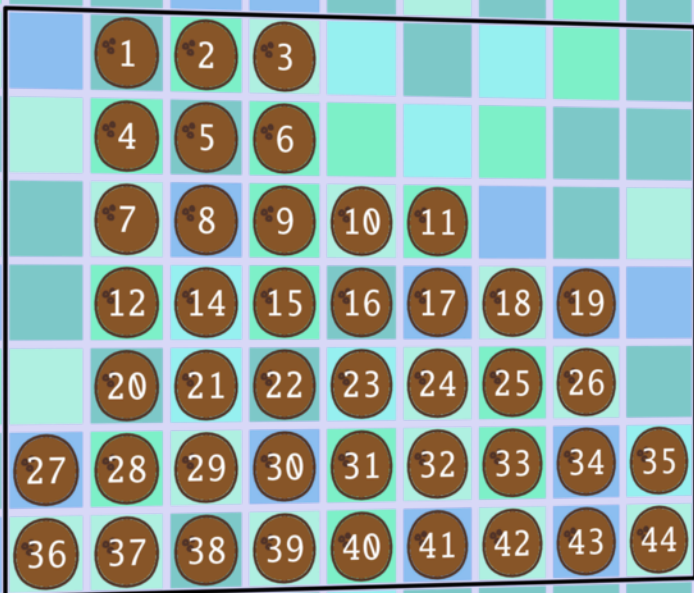
Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

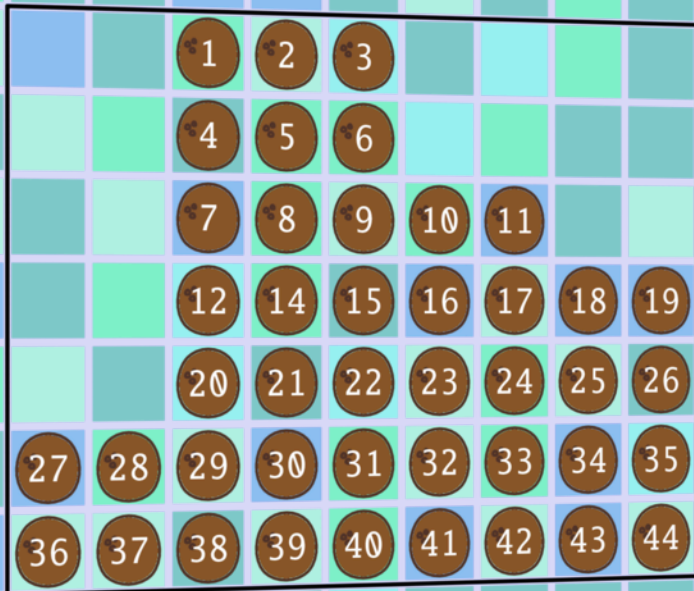


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape:



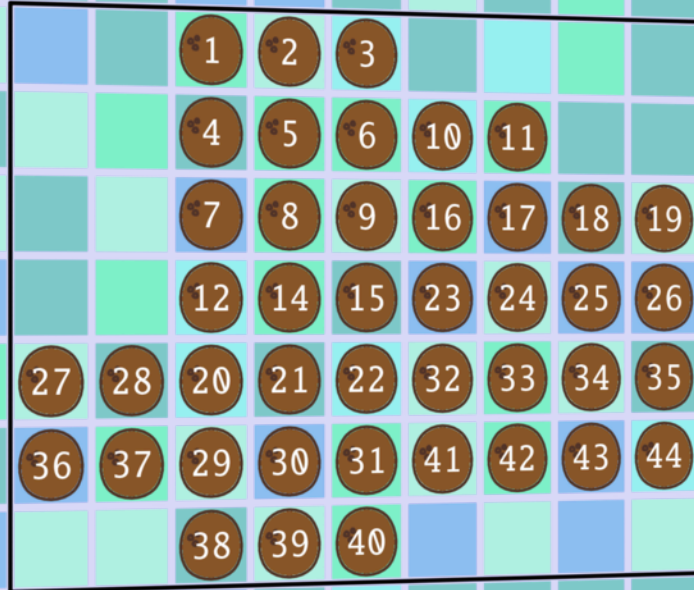
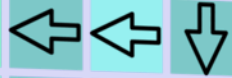
Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

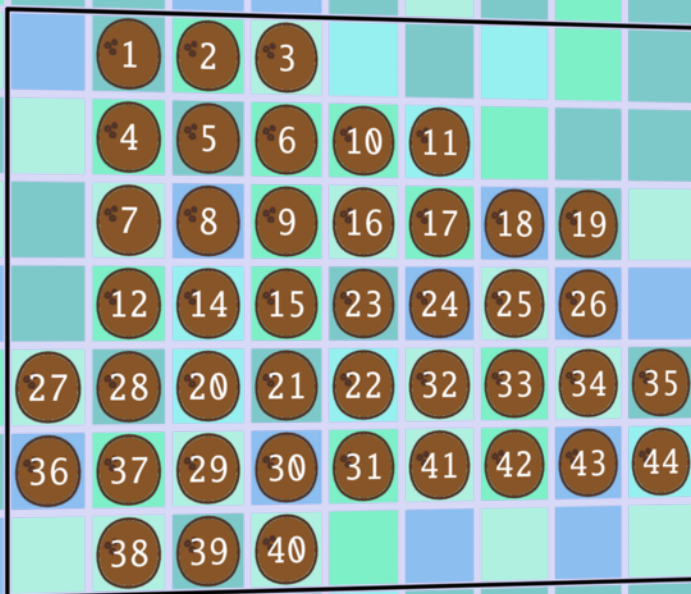
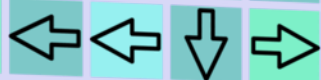
Proof idea. Cover by cycles that reach all coconuts.




Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape:

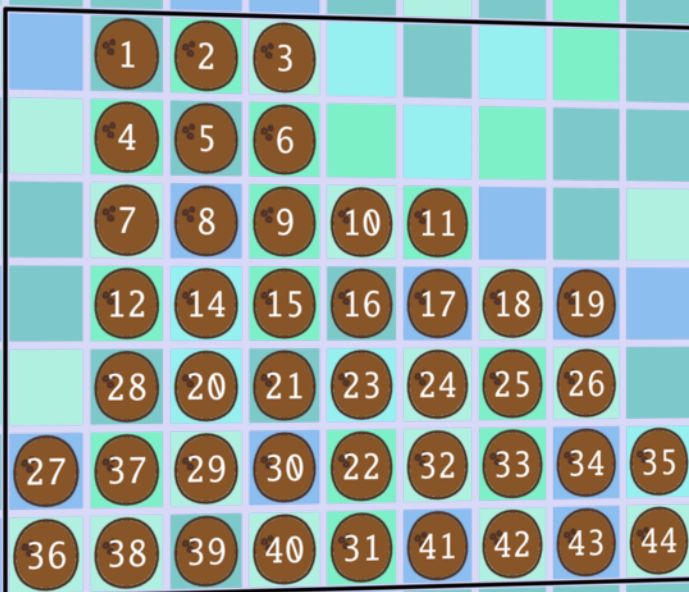
Proof idea. Cover by cycles that reach all coconuts.




Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

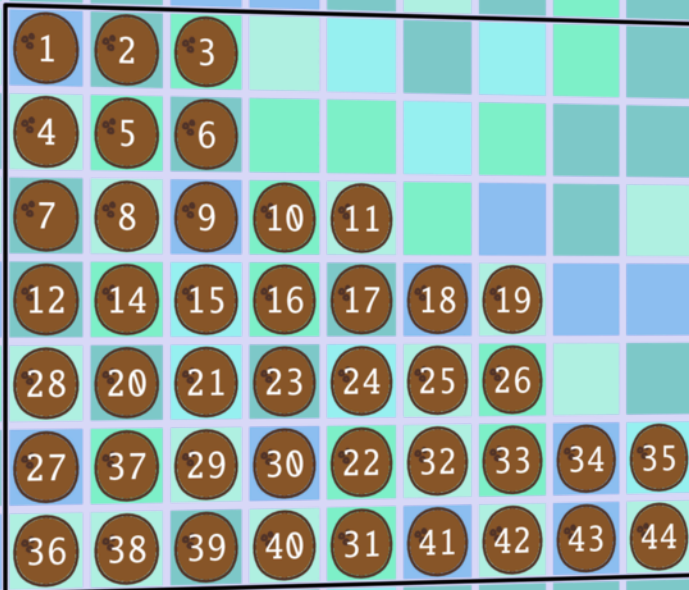
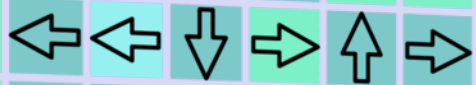
Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

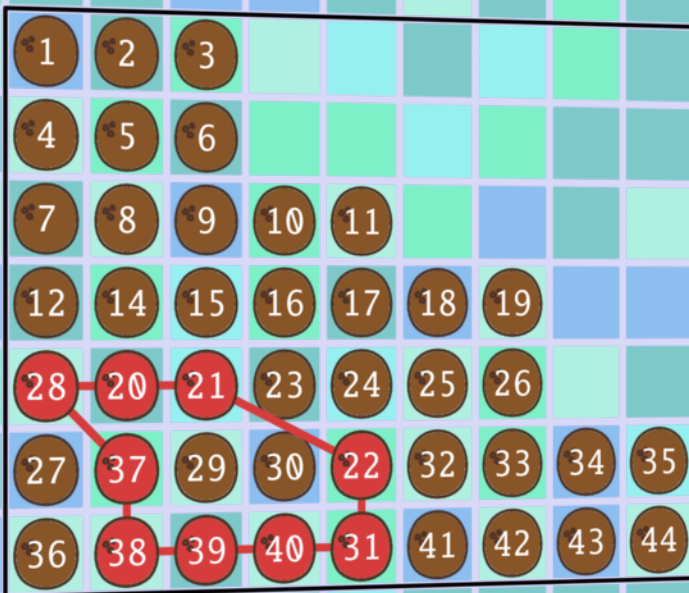
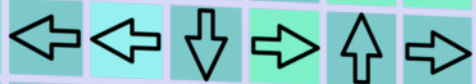
Proof idea. Cover by cycles that reach all coconuts.




Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape:

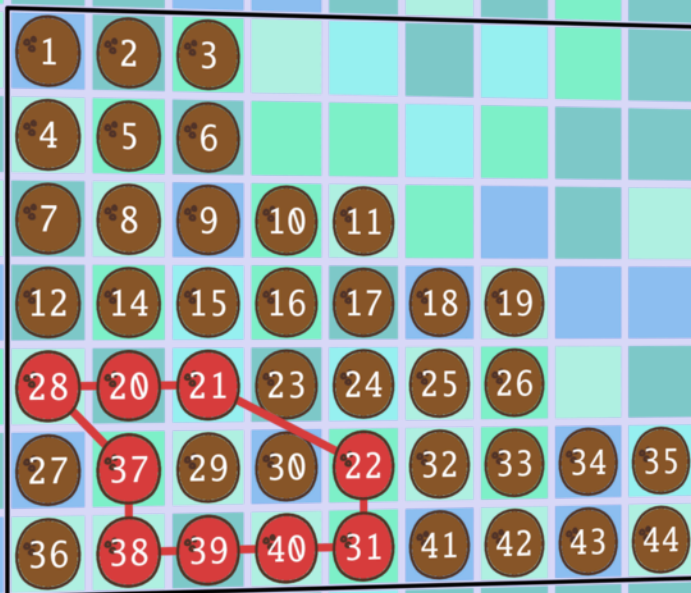
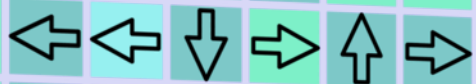
Proof idea. Cover by cycles that reach all coconuts.




Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.




Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

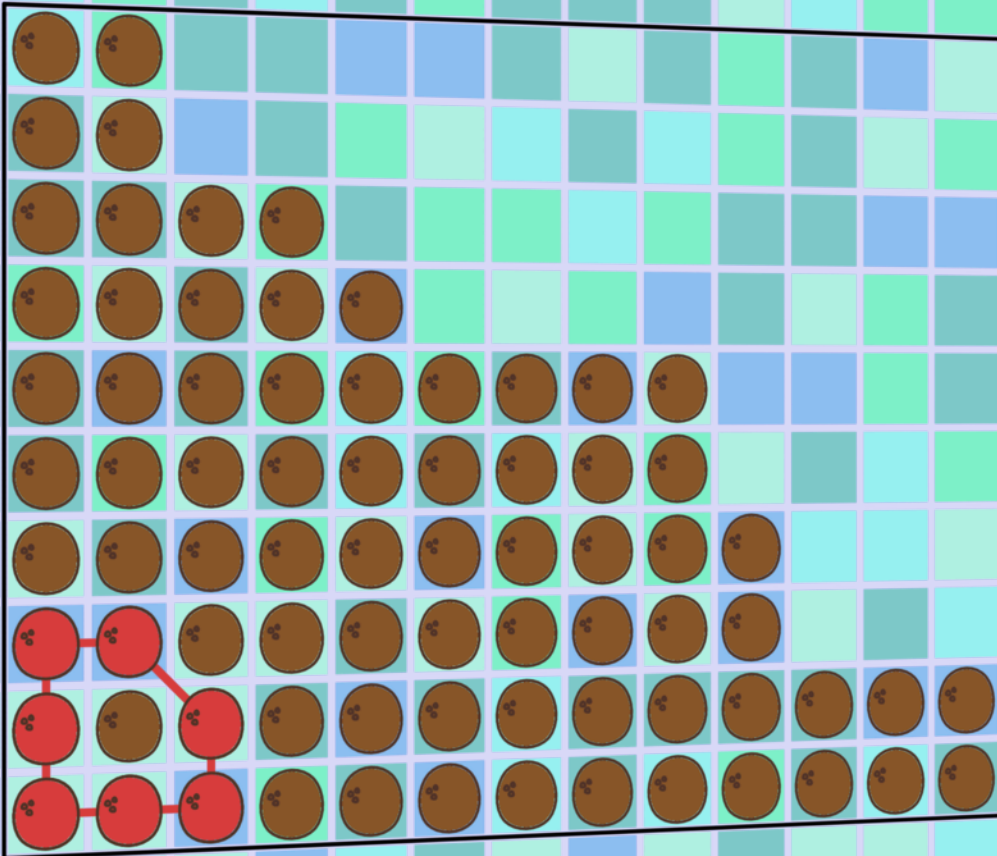
Theorem [Jones]. A good set of cycles implies we get all even permutations.

Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

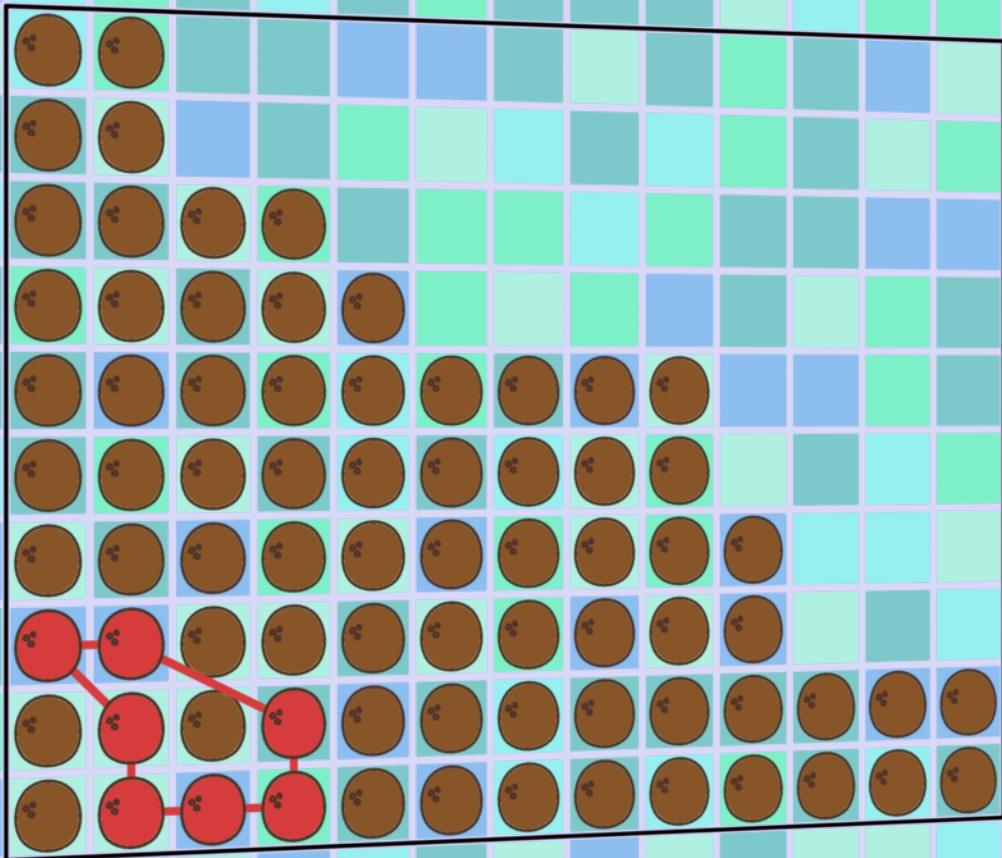


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

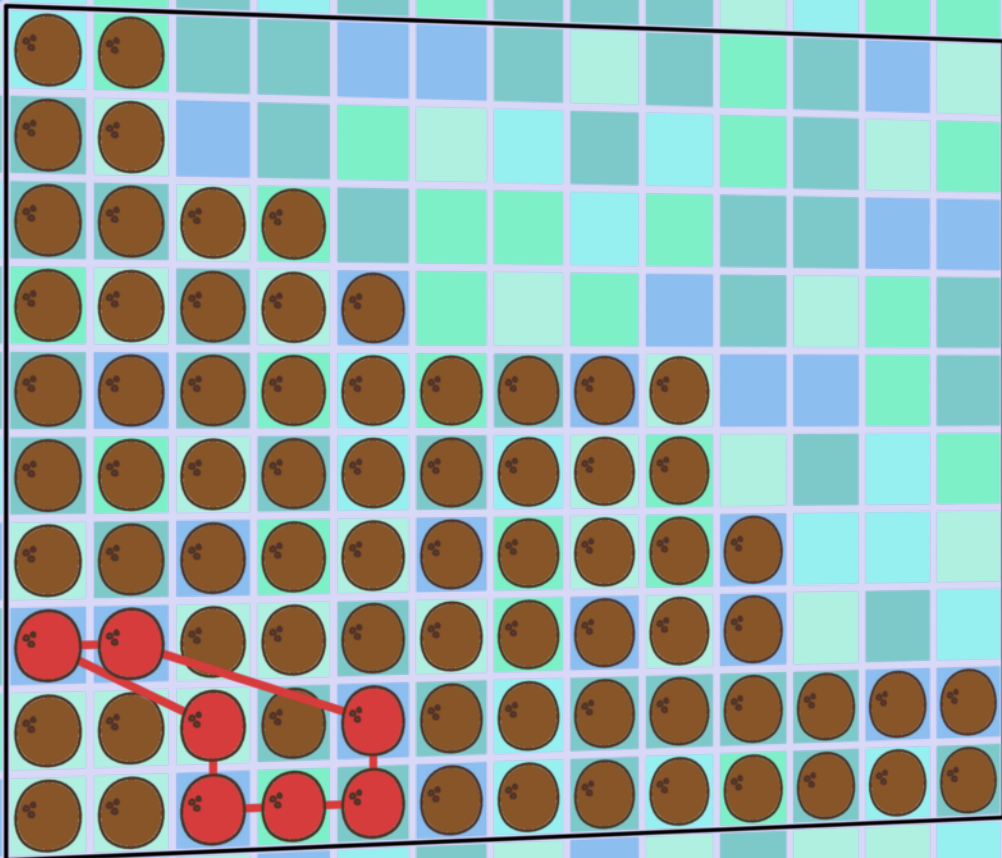


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

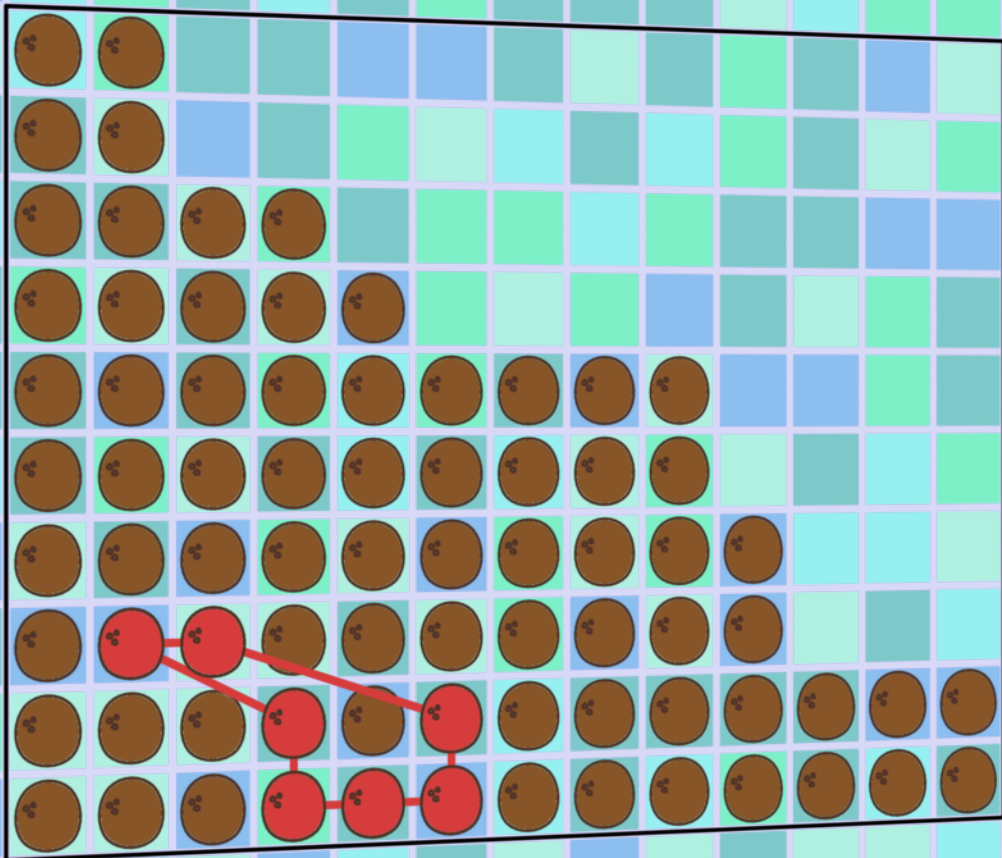


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.



Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

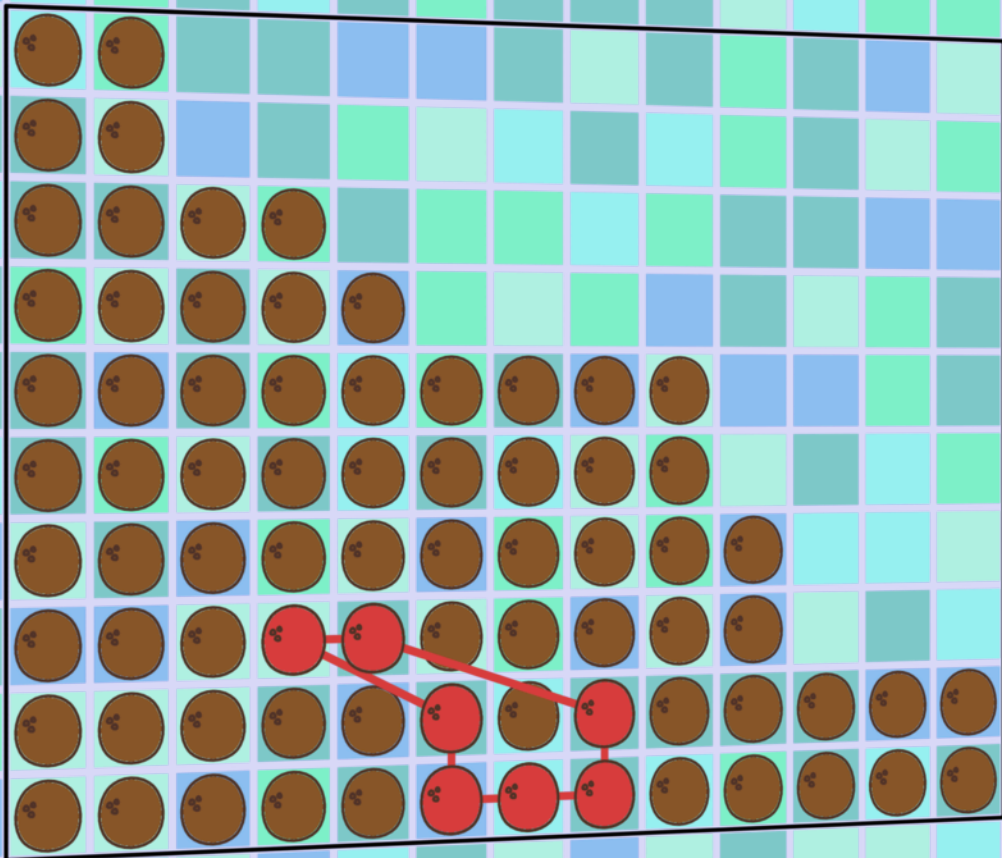


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

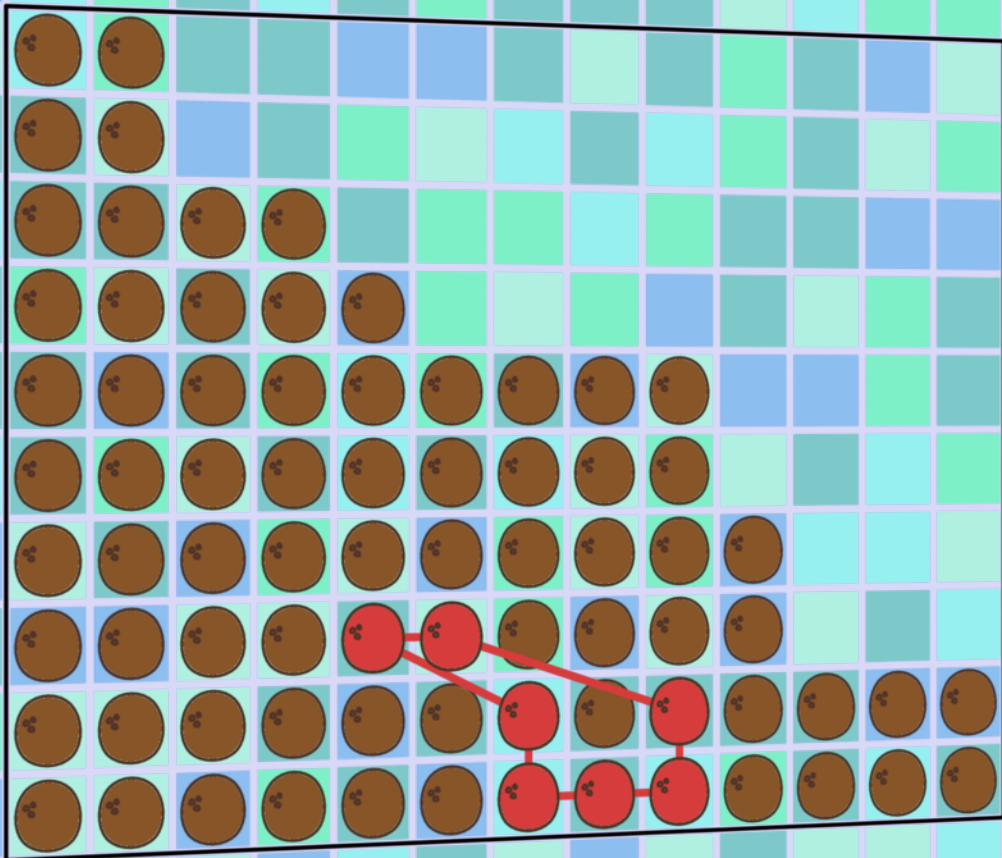


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

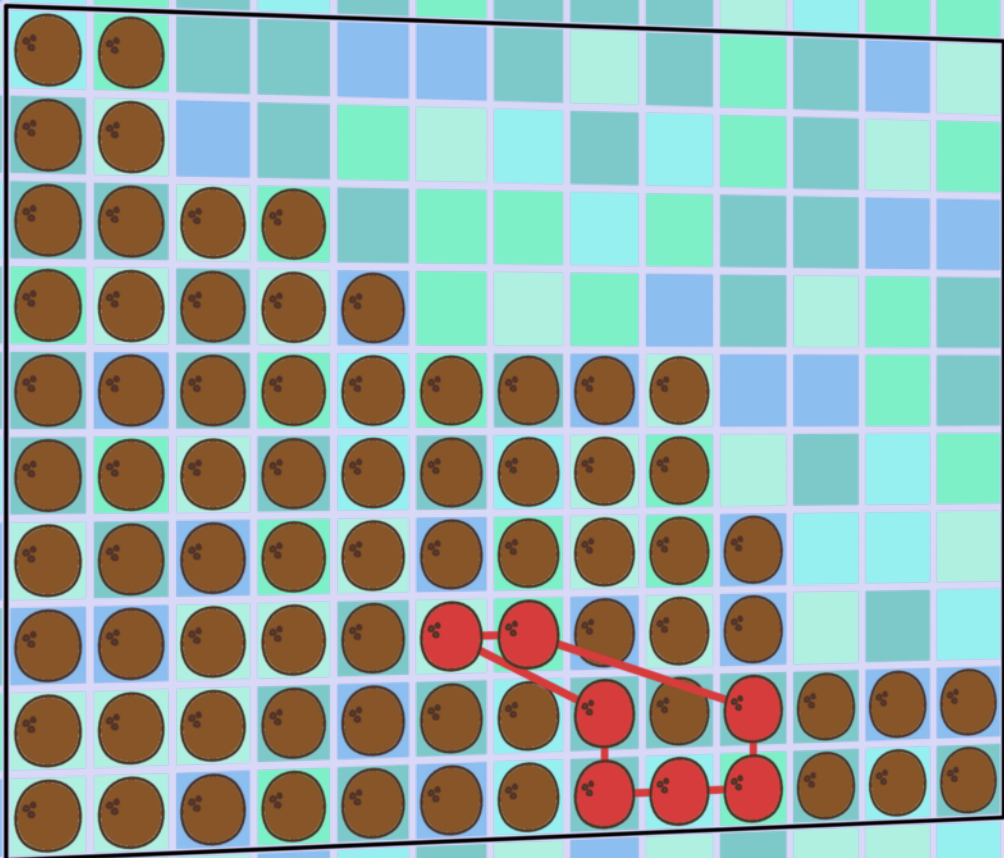


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

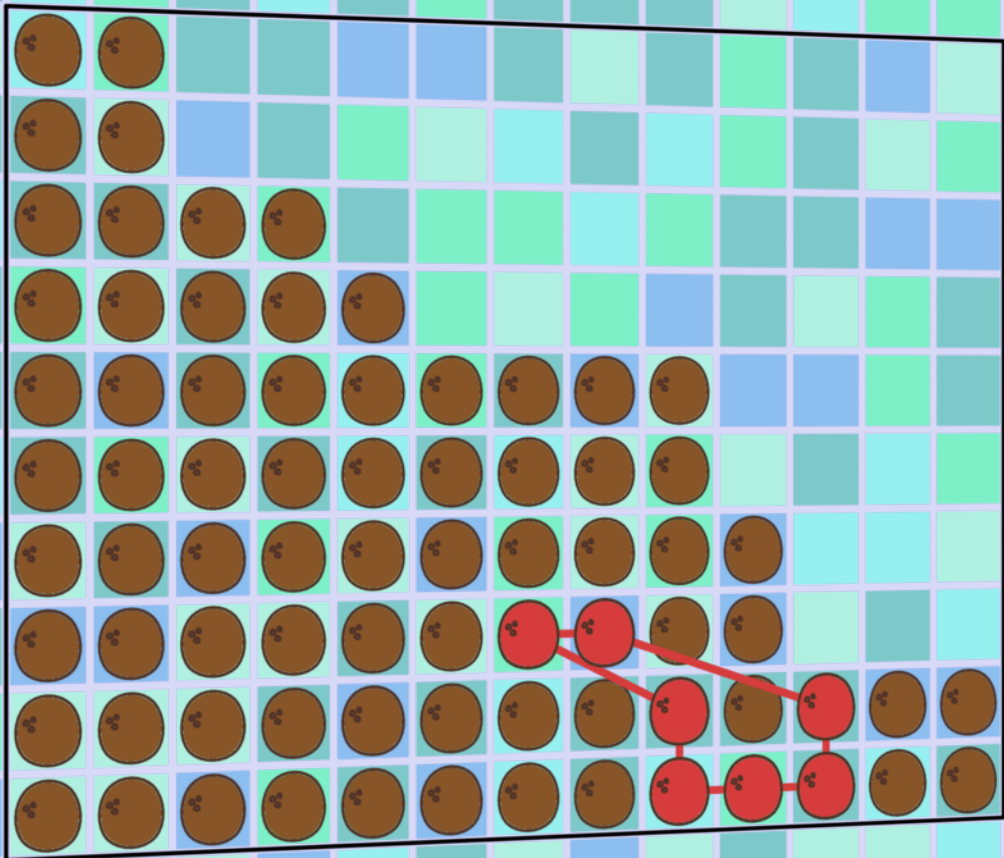


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

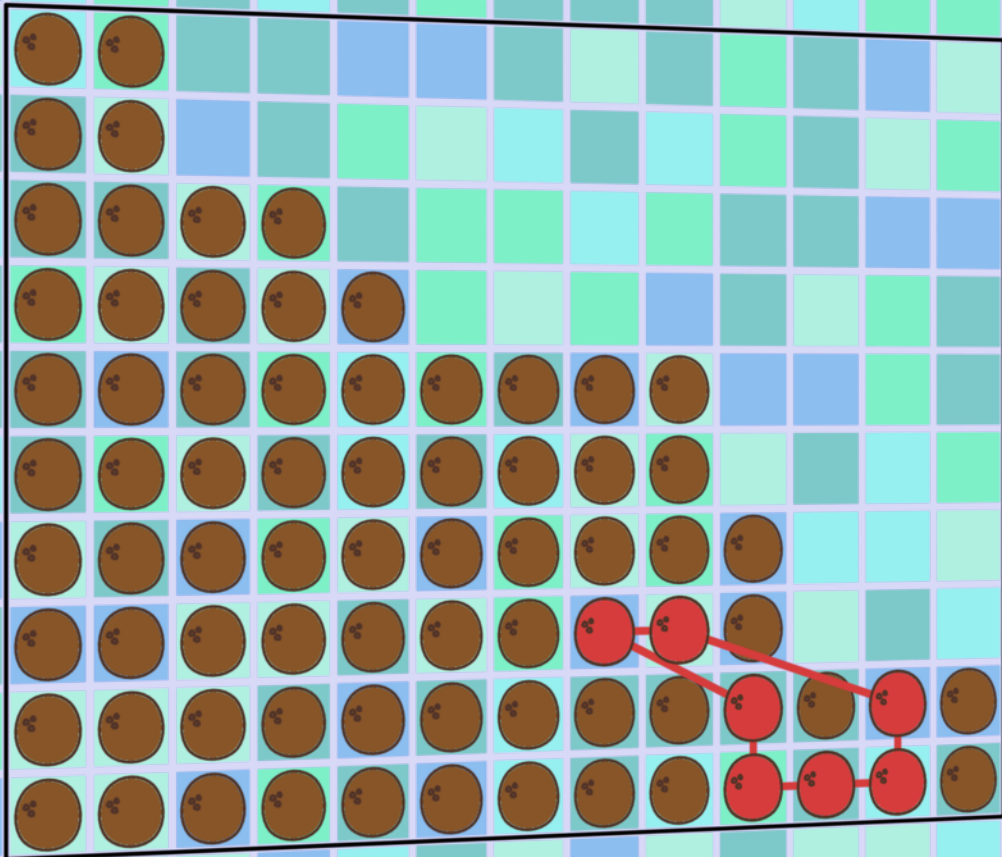


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

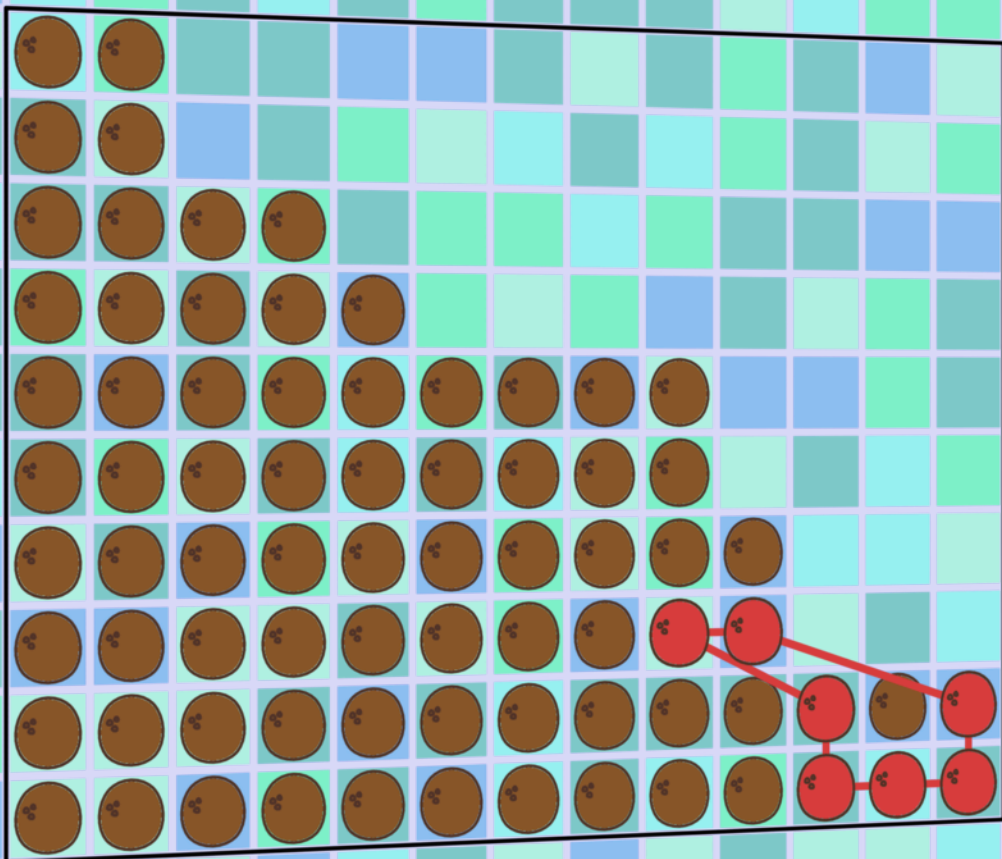


Theorem 3. Every even permutation can be realized unless:


- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

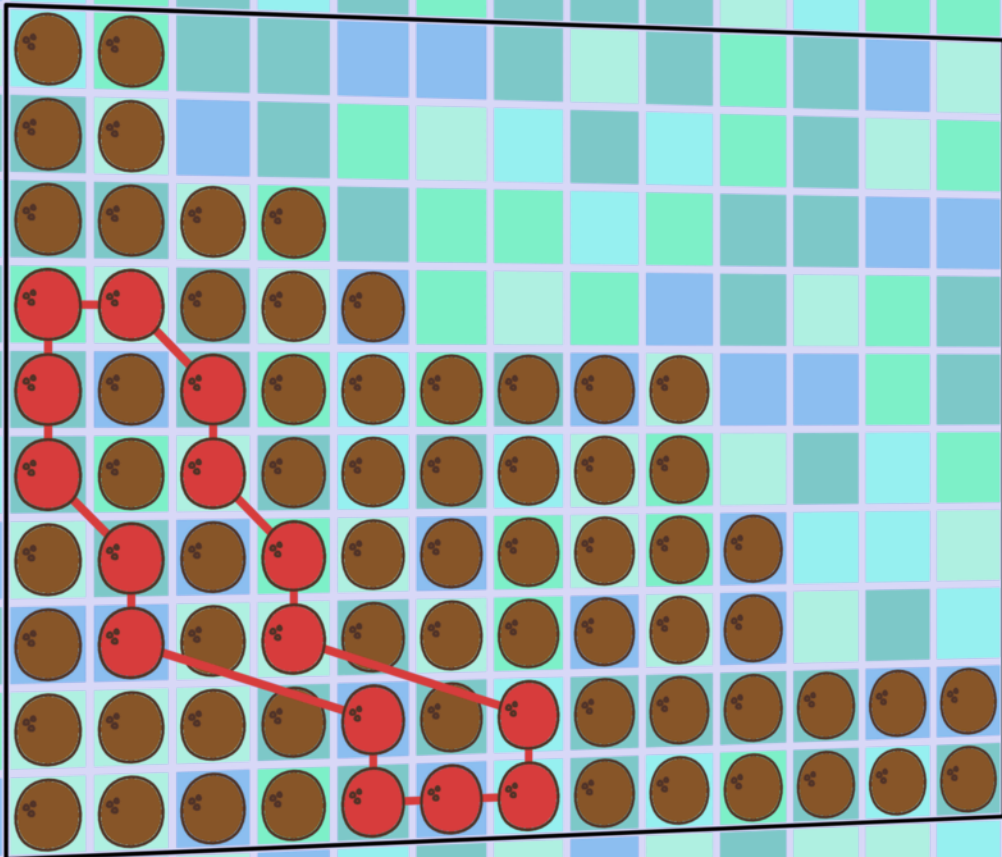


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.



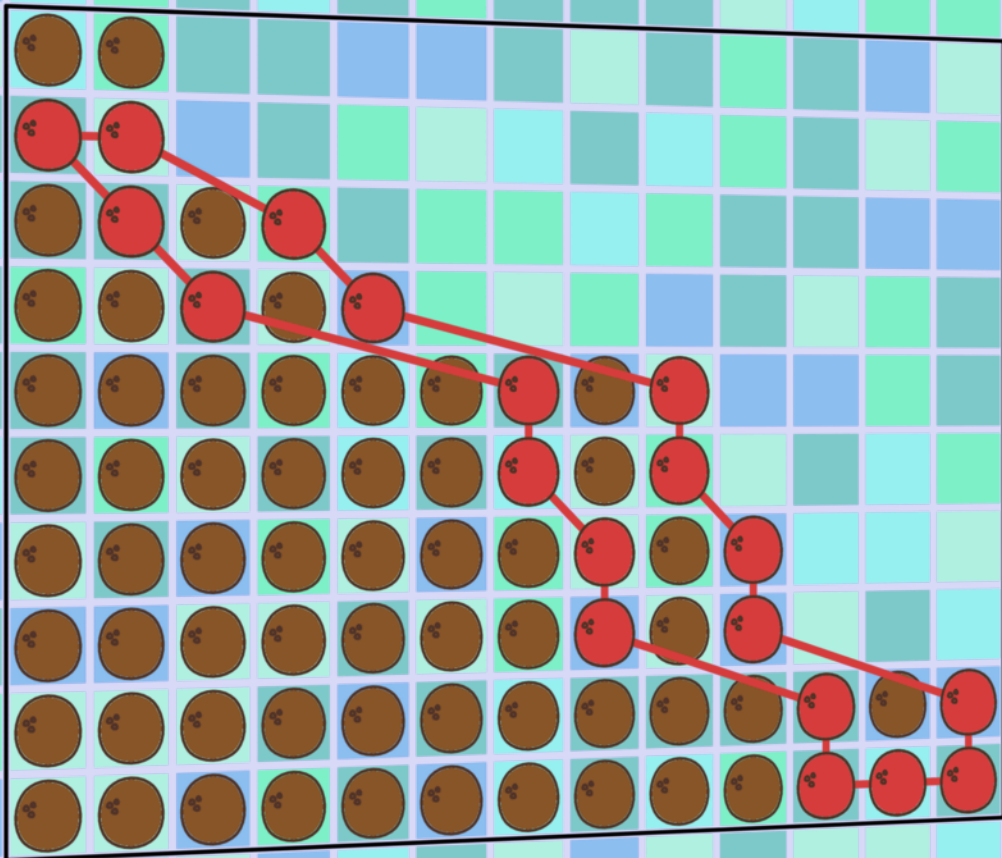
Theorem 3. Every even permutation can be realized unless:

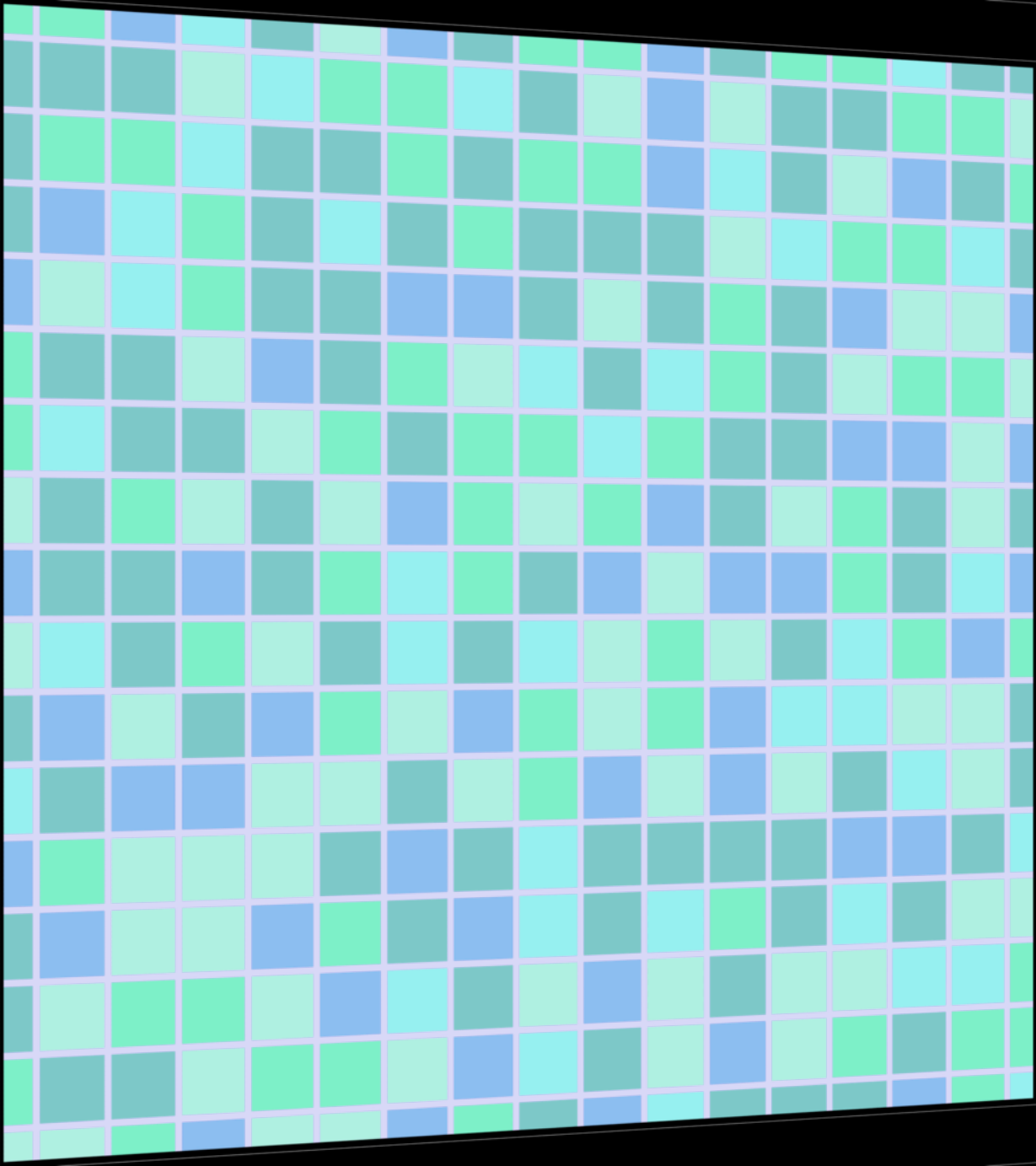
- there is a *core*; or
- it has this shape:




Proof idea. Cover by cycles that reach all coconuts.

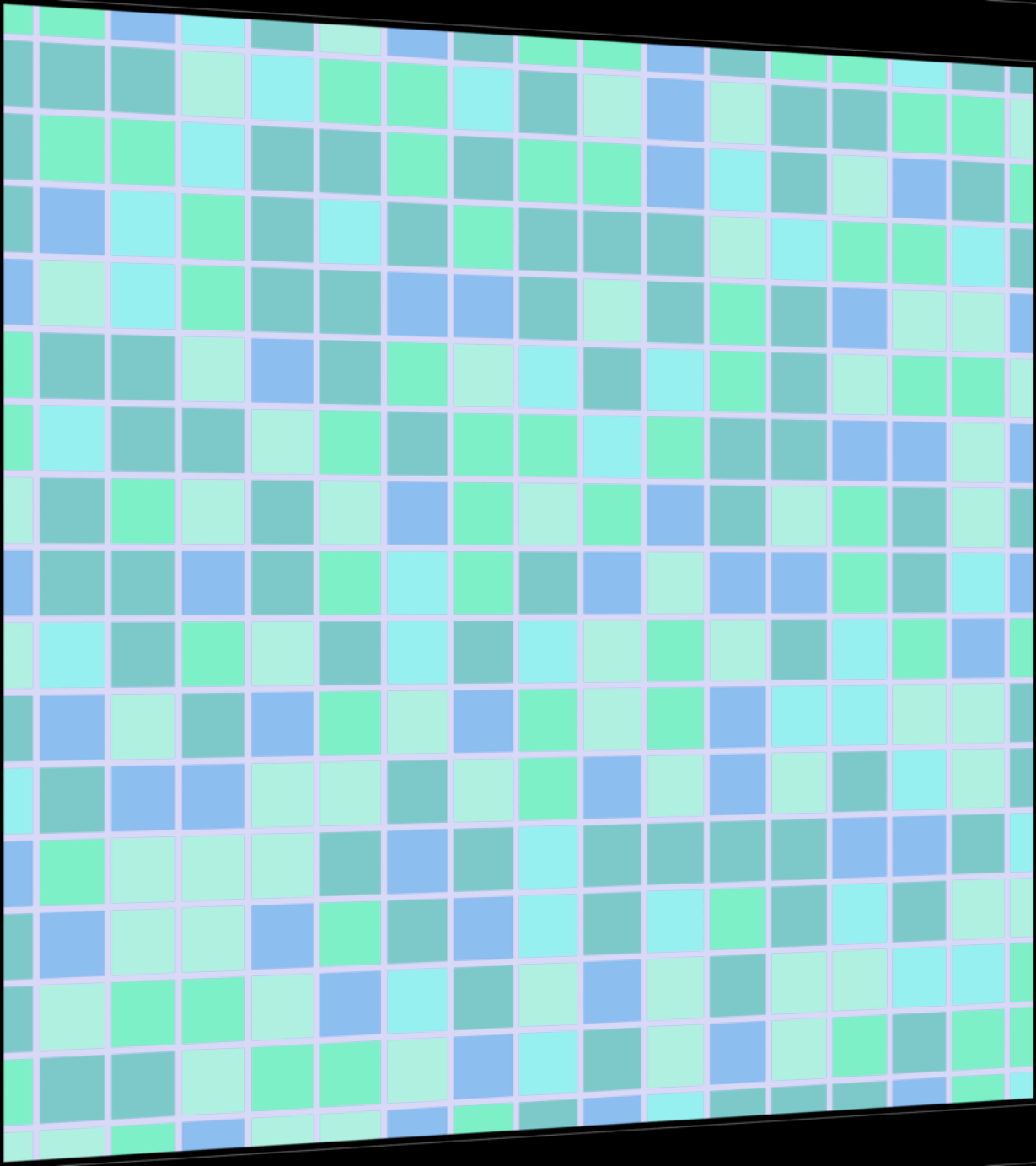
Theorem [Jones]. A good set of cycles implies we get all even permutations.






Theorem 3. Every even permutation can be realized unless:

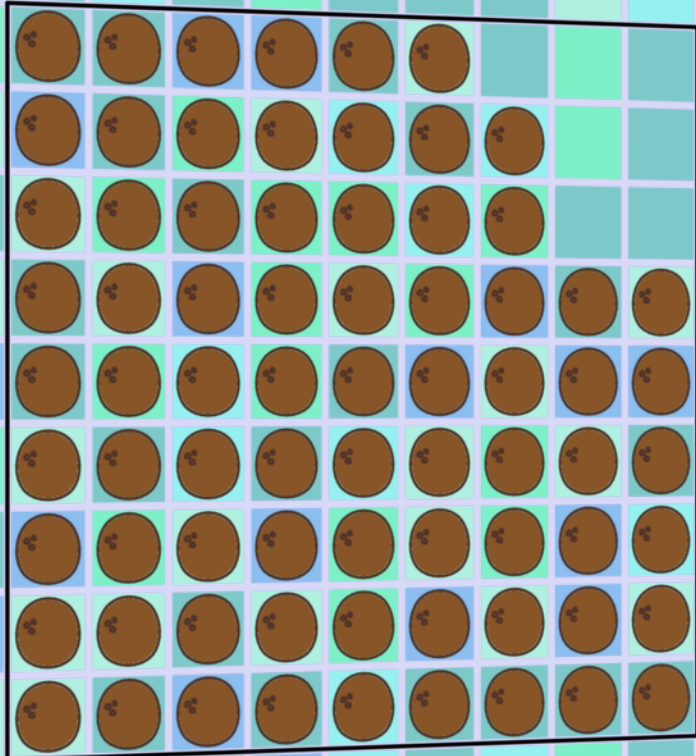
- there is a *core*; or
- it has this shape: 




Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

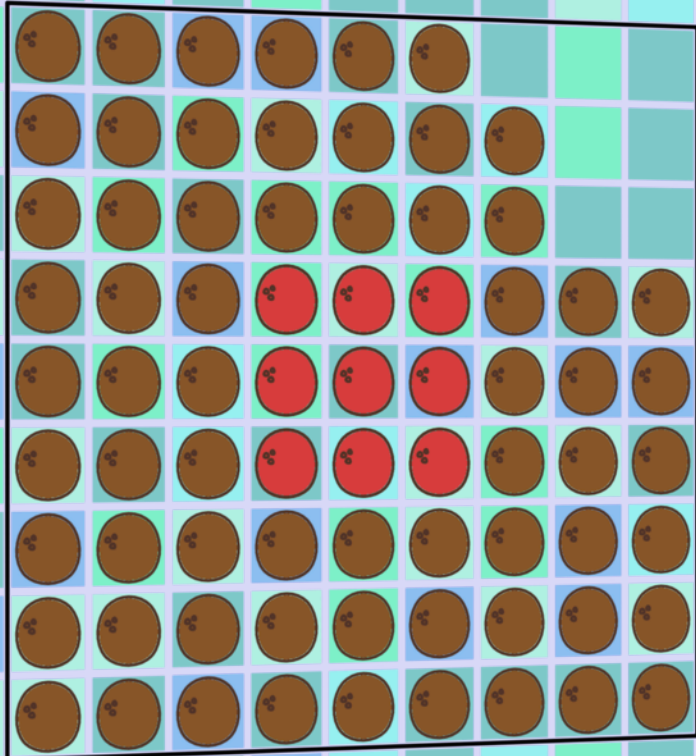
A configuration has an unmovable *core* if at least half of the columns are full and at least half of the rows are full.




Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

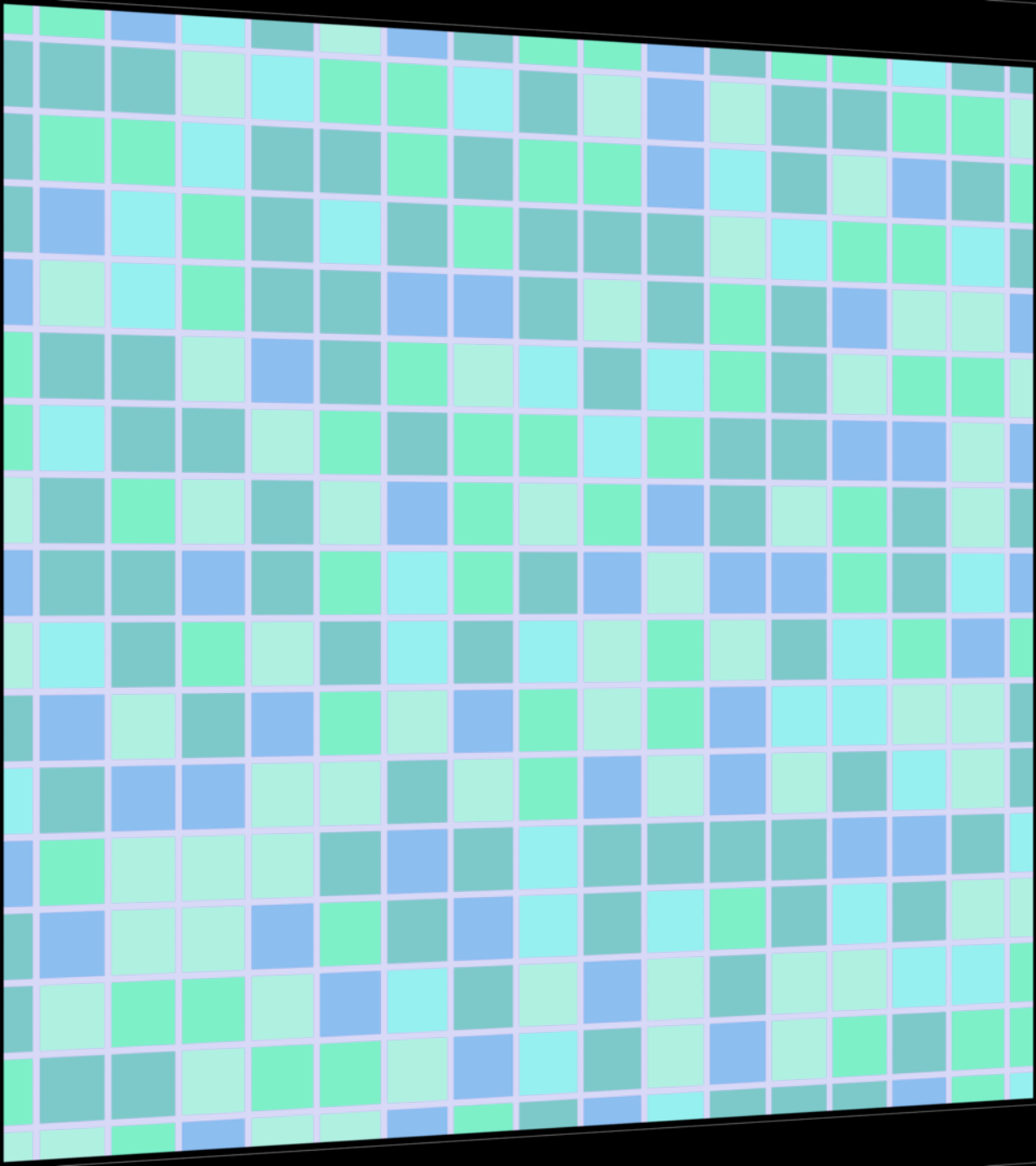
A configuration has an unmovable *core* if at least half of the columns are full and at least half of the rows are full.



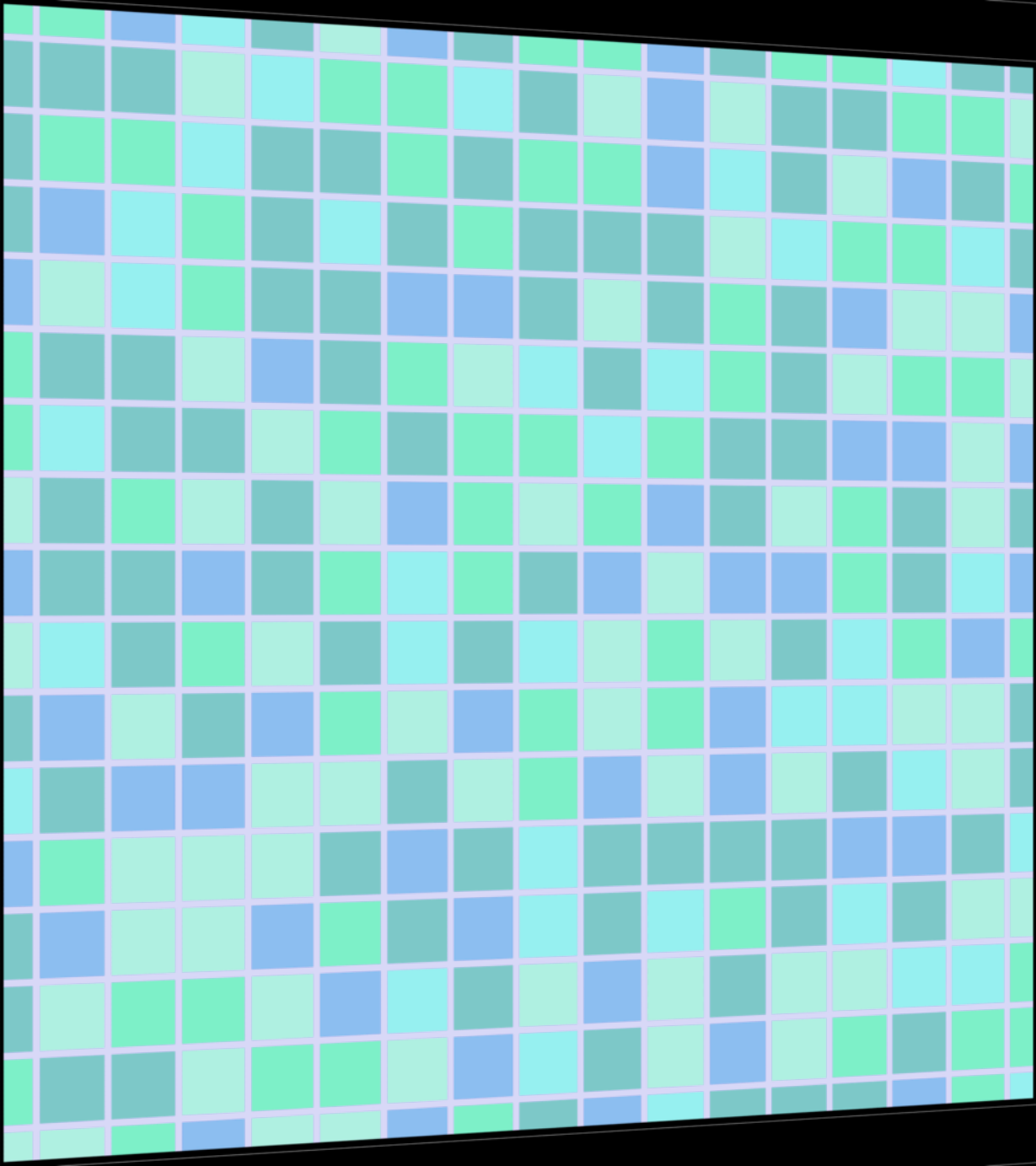
Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

A configuration has an unmovable *core* if at least half of the columns are full and at least half of the rows are full.

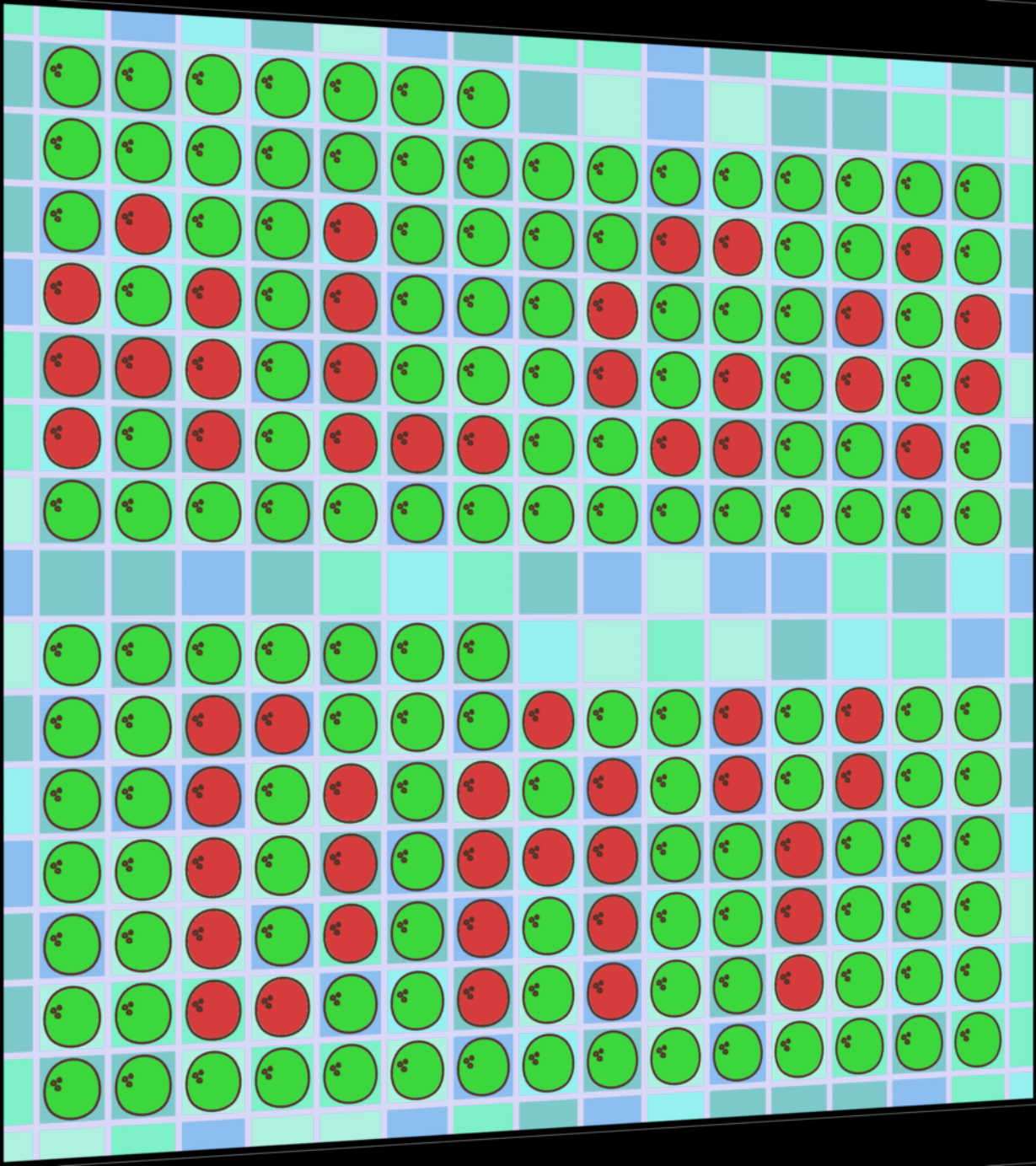


Open problem. Algorithms?



Open problem. Algorithms?

The theorem by Jones is
not constructive.



Labeled Reconfiguration by Compaction



Maarten Löffler

Based on joint work with

Hugo Akitaya
Greg Aloupis
Anika Rounds
Giovanni Viglietta