GDC
March 20-24, 2023
San Francisco, CA

Constrain Your Content: Generating Better Content With Constraint Programming

Marc Taylor

#GDC23

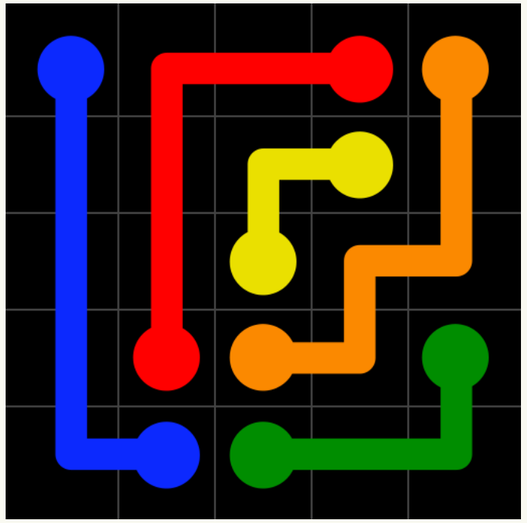
Marc Taylor

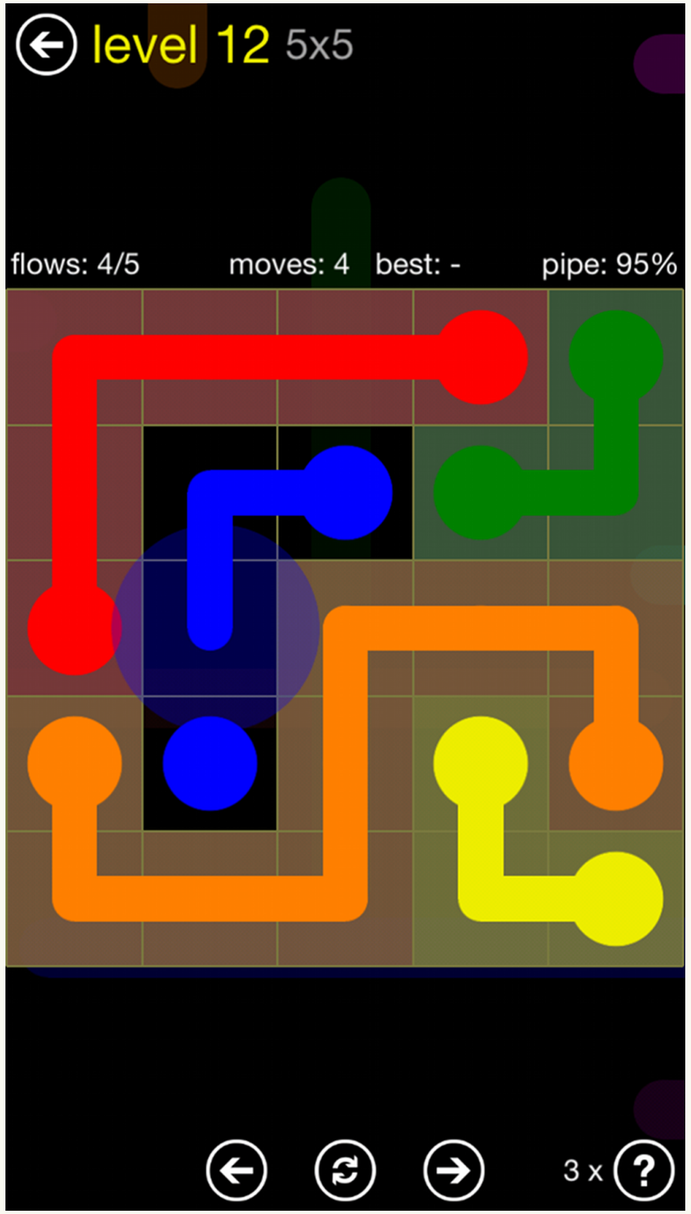
Puzzle Master
Tool Guy
Senior Developer

Big Duck Games

- Flow Free
  - Classic
  - Bridges
  - Hexes
  - Warps
- Flow Fit
  - Words
  - Sudoku

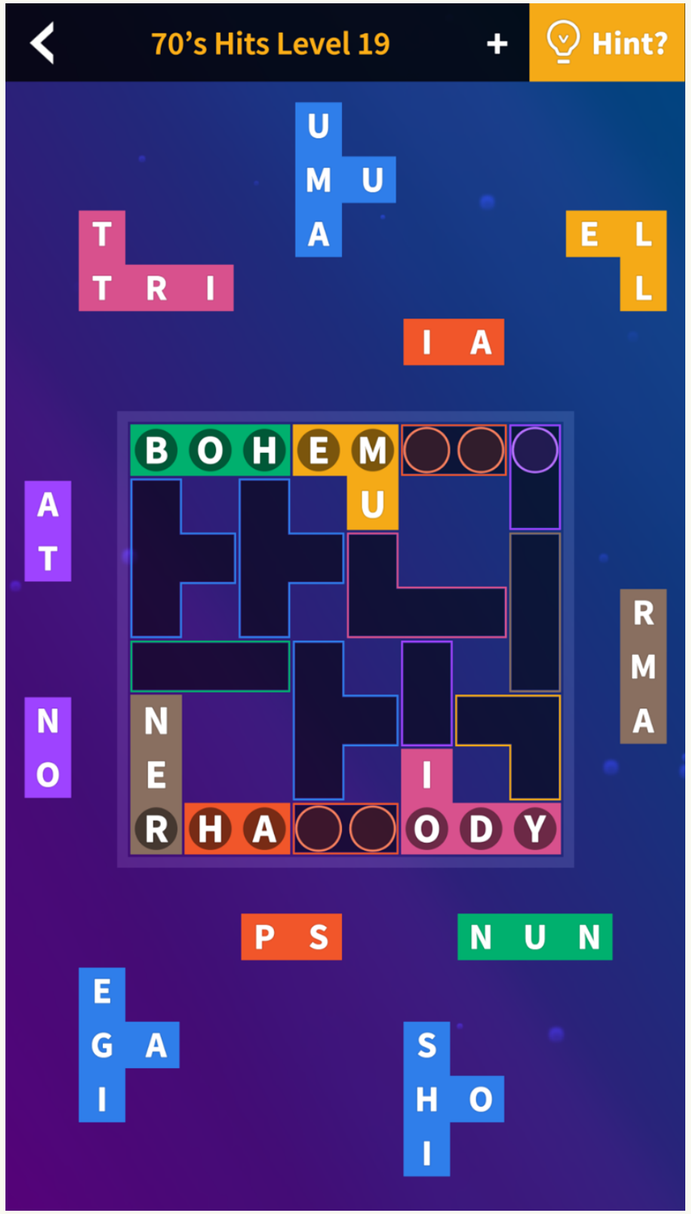
Flow Free





Flow Fit


F
I T


70's Hits Level 19
+
Hint?
U
M U
A
T
T R I
E L
L
I A
B O H E M
U
A
T
R
M
A
N
O
N
E
R H A
I
O D Y
P S
N U N
E
G A
I
S
H O
I

Flow Fit: Sudoku

What Is Constraint Programming (CP)?

Constraint Programming, CP, “is the name given to identifying feasible solutions out of a very large set of candidates, where the problem can be modeled in terms of arbitrary constraints.”
(https://developers.google.com/optimization/cp)

Shift Scheduling Problems?

WEEKLY EMPLOYEE SCHEDULE														
WEEK OF 3/20/23														
ASSIGNMENT	MON (AM)	MON (PM)	TUE (AM)	TUE (PM)	WED (AM)	WED (PM	THU (AM)	THU (PM)	FRI (AM)	FRI (PM)	SAT (AM)	SAT (PM)	SUN (AM)	SUN (PM)
John Doe	1	0	0	0	1	0	1	0	1	0	0	0	0	1
Jane Doe	0	1	0	1	0	0	0	0	0	1	0	1	1	0
Bob Dole	1	0	1	0	0	1	0	0	0	0	1	0	0	1
Bobbi Dole	1	0	1	0	0	1	0	0	0	0	1	0	0	1
Jebediah	0	1	0	1	0	1	0	1	0	1	0	0	0	0

John Doe can't work on Tuesdays
Jane Doe cannot work with John Doe
Bob Dole can only work with Bobbi Dole
Bobbi Dole cannot work on Thursdays
Jebediah cannot work on Sunday
All employees must work exactly 5 shifts per week
Each shift needs at least one employee
No employee can work both morning and afternoon shifts on the same day

Constraint Programming

- Designer creates the model
- Solver runs the model *quickly* (So fast it got italicized)
- Focus can be placed on the design of the model instead of optimizing the brute force

What Can CP Do For You?

Make good content

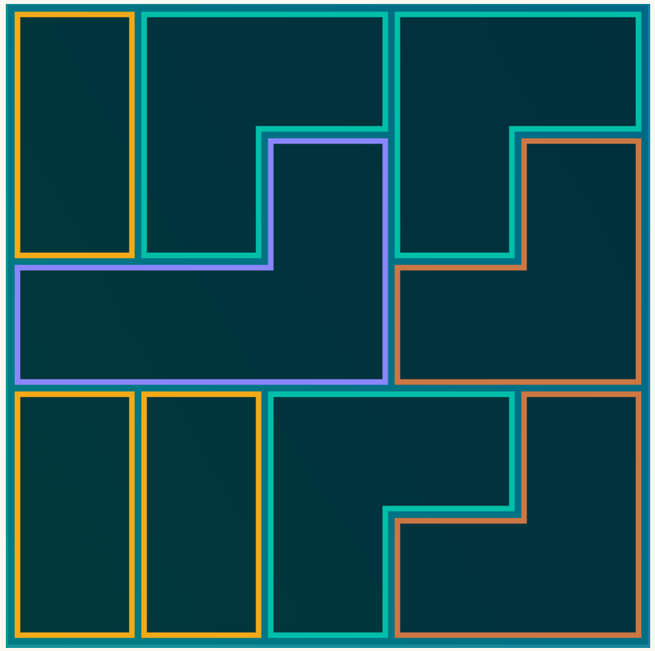
Latin Square + Slicing = Level

Content
(Latin Square)

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

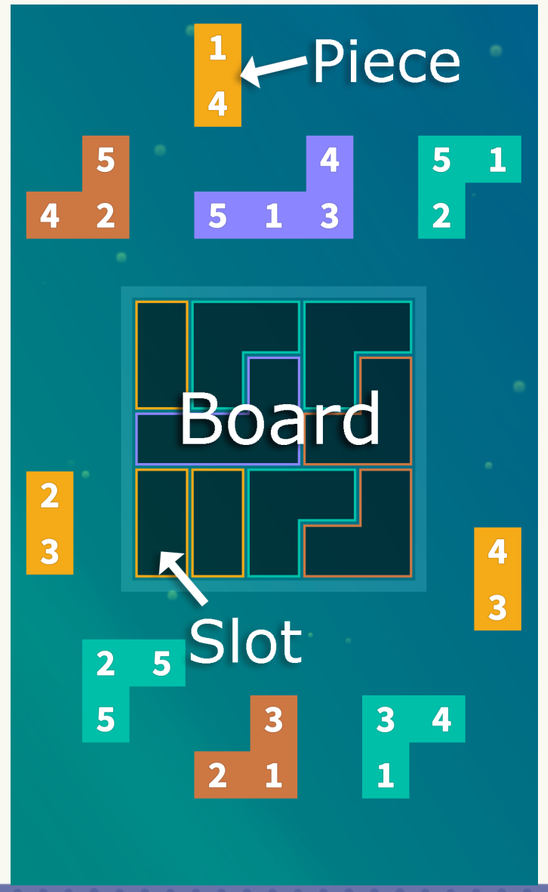
+

Slicing



=

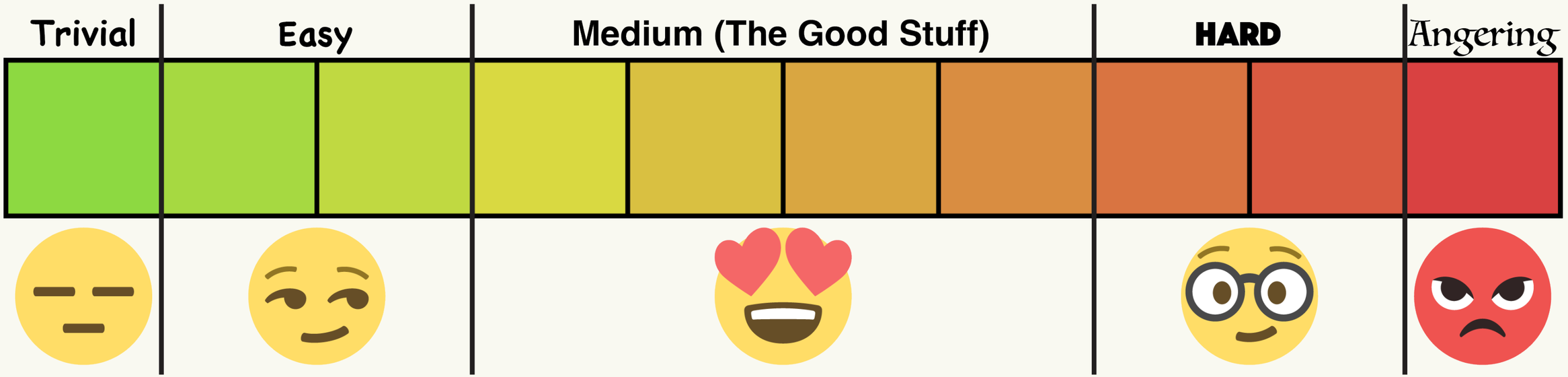
Level



What Makes A Good Level?

1. Difficulty
2. Piece uniqueness
3. Piece variety
4. No “bad” pieces
5. Single solution

1. Difficulty

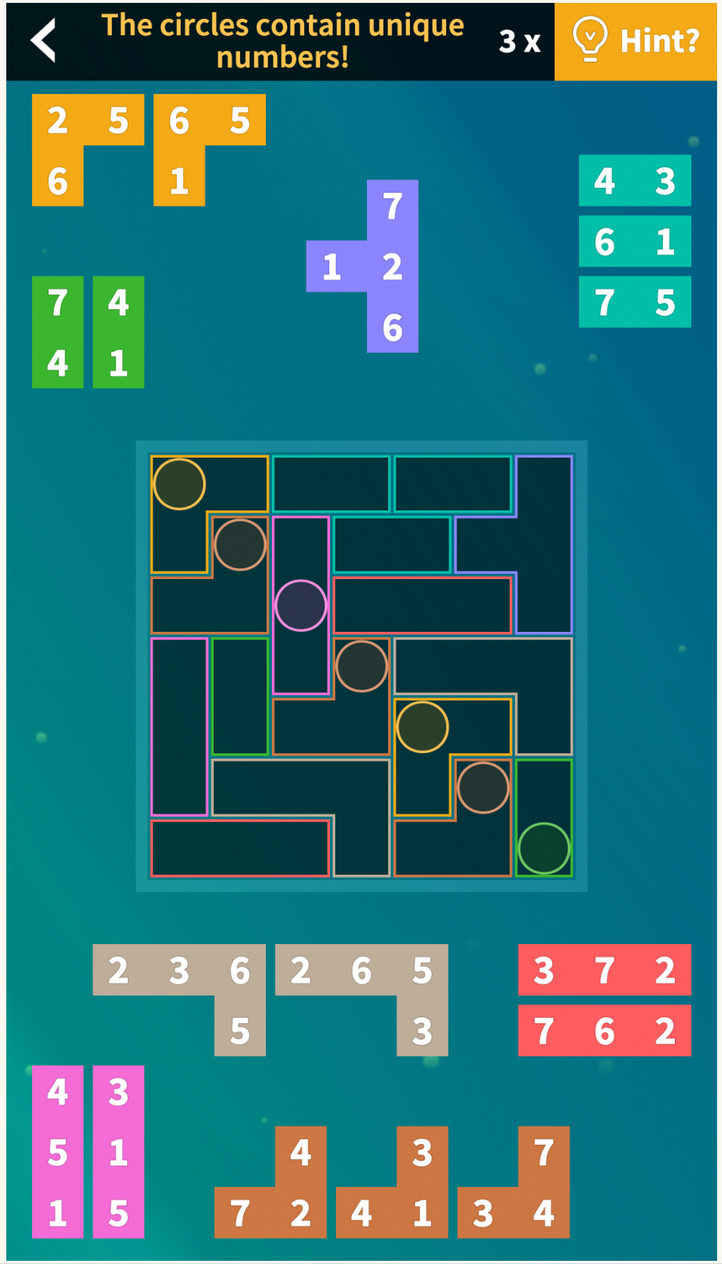


Determining Difficulty

- Combinatorial Complexity
  - How many ways the pieces can be fit
- Initially Guaranteed Pieces
  - How many pieces only have one slot
  - Gives initial footholds into puzzle solution

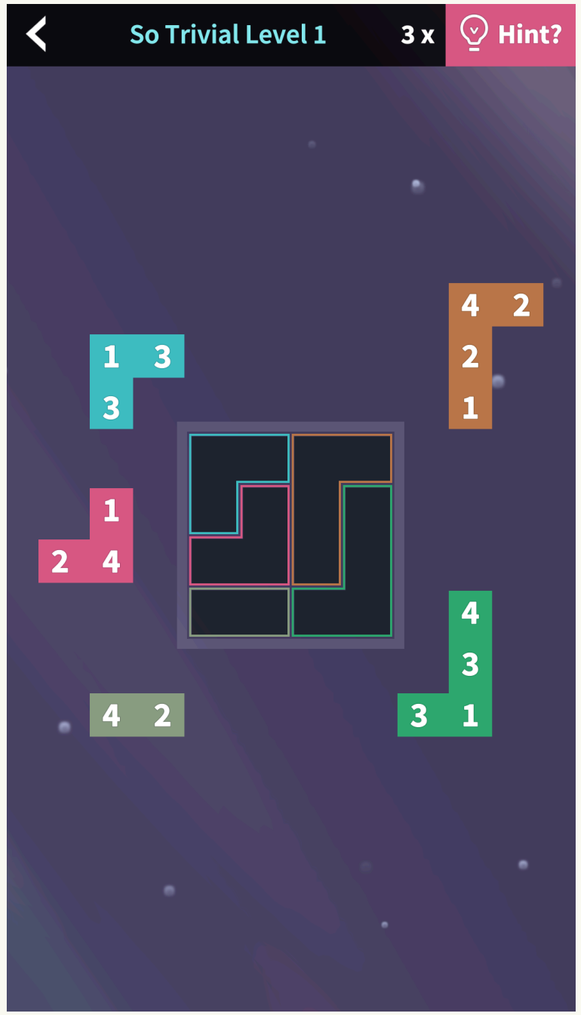
2. Piece Uniqueness

- No two pieces should be identical
- When no two pieces are identical, many logical solving techniques become more apparent

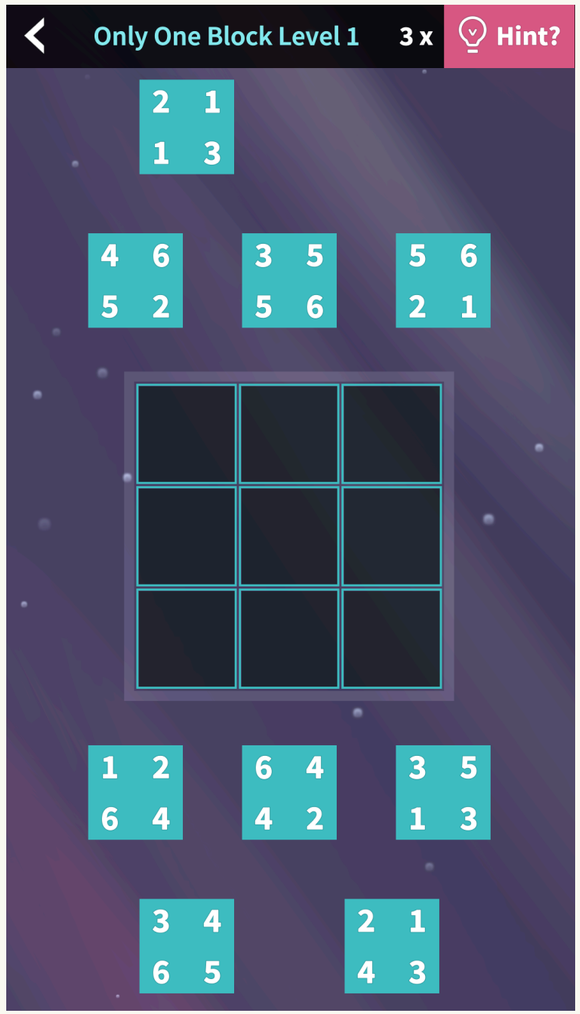


3. Piece Variety

Unique Pieces, Trivial!



Same Pieces, Hard!



4. No “Bad” Pieces

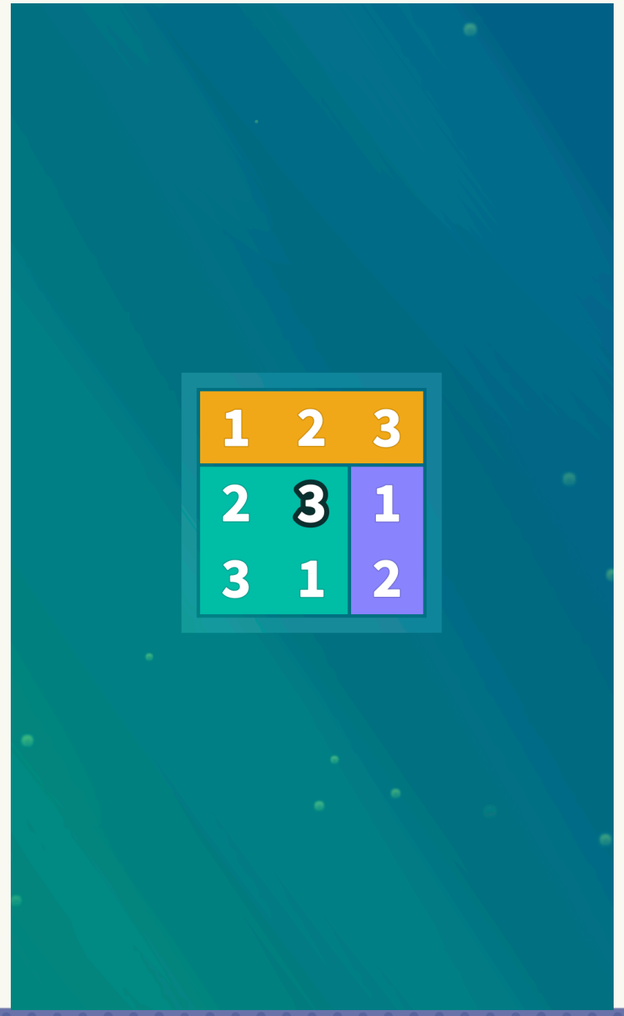
Any block that is not fun is bad

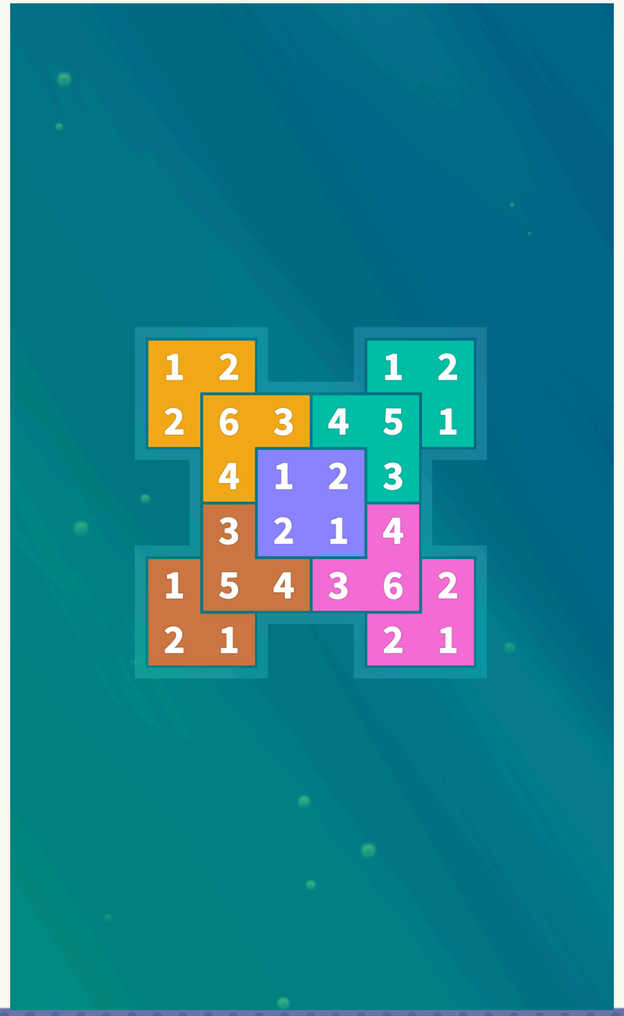
Luckily, there is only one key block that is generally considered bad.

The 1x1 piece

We hates it!

5. Single Solution





How To Actually Use CP

1. Choose a Solver

2.Make A Model

3. Run the model in the solver
4. Extract the solution

CP Solvers

- Optimize running of models
  - Domain Reduction
  - Constraint Aggregation
  - Propagation of Arithmetic Constraints
  - And More!

CP Solvers

- Library added to the codebase
- Also used to make the models
- We used Google OR-Tools Original CP Solver

Parts Of A CP Model

- Variables
- Constraints
- Objectives

Variables

There is only one variable type in Google’s Original CP Solver:

IntVar

- Name
- Lower Bound (Inclusive)
- Upper Bound (Inclusive)

Constraints

These are the arbitrary constraints which the solver must fulfill when creating a solution.

Some examples of constraints:

2x + 7y + 3z	<=	50
3x - 5y + 7z	<=	45
5x + 2y - 6z	<=	37
x	==	2y
All Different		[x, y, z]

Objectives

- Optional
- Scoring function to maximize or minimize

Some examples of objectives:

Maximize 2x + 3y

Minimize x - 2z

The Decision Builder

The decision builder is the main input to the original CP solver. It contains the following:

- vars — An array containing the variables for the problem.
- A rule for choosing the next variable to assign a value to.
- A rule for choosing the next value to assign to that variable.

(https://developers.google.com/optimization/cp/original_cp_solver#solve)

Let’s Make A Level!

Rethinking The Approach

1. Identify and add the key variables
   A. What results are needed?
2. Identify and add constraints
   A. What is the constraint?
   B. How to apply the constraint?
   C. What new variables need to be added?

Key Variables

- Each cell is a variable with a value between 0 and 9.
- Value 0 represents an obstacle

```
IntVar[] cellVars = solver.MakeIntVarArray(cellCount, 0, 9, "CellVar");
```

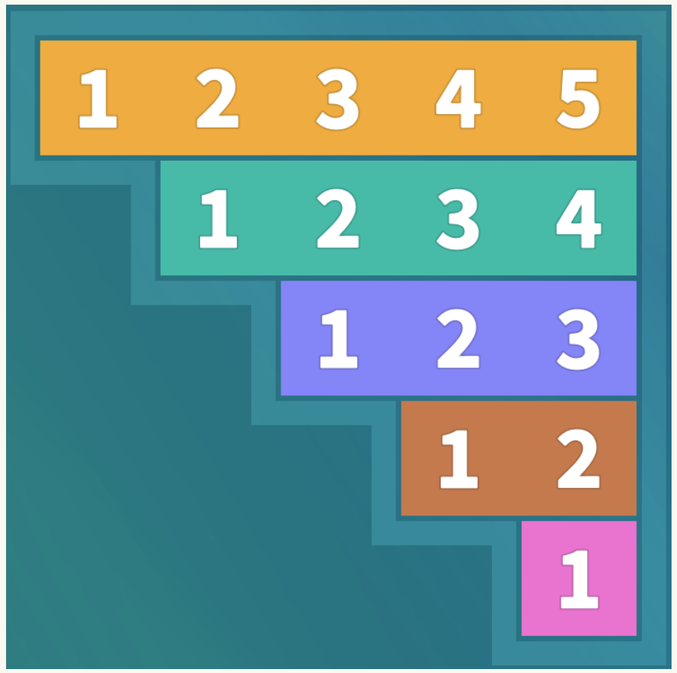
Constraint: All Cells Filled

- All non-obstacle cells must be greater than 0

```
for (int cell = 0; cell < cellCount; cell++)
{
    char cellValue = board.GetValue(cell);

    if (board.IsCellObstacle(cell))
    {
        solver.Add(_cellVars[cell] == 0);
    }
    else
    {
        solver.Add(_cellVars[cell] > 0);
    }
}
```

Defining Sudoku Regions

Each row and column are sudoku regions




```
List<IntVar[]> sudokuRegions = new List<IntVar[]>();
int[][] sudokuRegionSegments = board.GetSudokuSegments();
for (int i = 0; i < sudokuRegionSegments.GetLength(0); i++)
{
    int[] sudokuRegionSegment = sudokuRegionSegments[i];

    List<IntVar> sudokuRegion = new List<IntVar>();
    foreach (int cell in sudokuRegionSegment)
    {
        sudokuRegion.Add(_cellVars[cell]);
    }

    sudokuRegions.Add(sudokuRegion.ToArray());
}
```

Constraint: Sudoku Region

- Each cell value is unique in the region
- Each cell value less than or equal to the count of cells in the region

```
foreach (IntVar[] sudokuRegion in sudokuRegions)
{
    // Constraint: No number can be used multiple times.
    solver.Add(solver.MakeAllDifferent(sudokuRegion));

    // Constraint: No number greater than the region length.
    foreach (IntVar cell in sudokuRegion)
    {
        solver.Add(cell <= sudokuRegion.Length);
    }
}
```

Set Up The DecisionBuilder

- Pass in all of the variables we created
- Choose the solving strategy
  - Assigning to a random variable
  - Assigning a random value

```
DecisionBuilder decisionBuilder = solver.MakePhase(_cellVars.ToArray(), Solver.CHOOSE_RANDOM, Solver.ASSIGN_RANDOM_VALUE);
```

Run The Solver

```
solver.NewSearch(decisionBuilder);
solver.NextSolution();
```

Extract The Solution

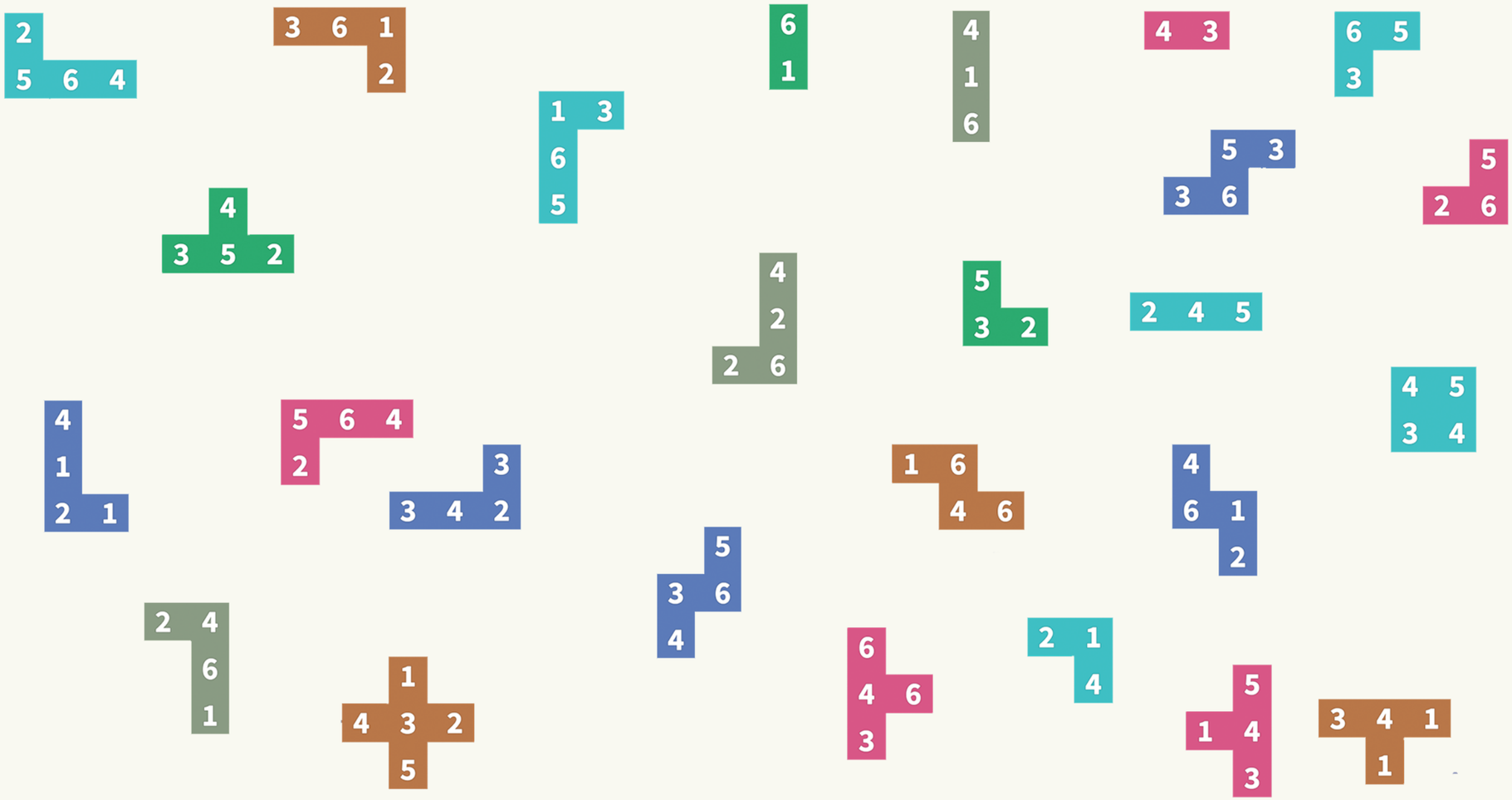
```
char[,] solutionGrid = new char[board.Height, board.Width];
for (int y = 0; y < board.Height; y++)
{
    for (int x = 0; x < board.Width; x++)
    {
        int cellIndex = (y * board.Width) + x;
        int cellIntValue = (int)cellVars[cellIndex].Value();

        solutionGrid[y, x] = (char)('1' + cellIntValue - 1);
    }
}
```

Cleaning Up

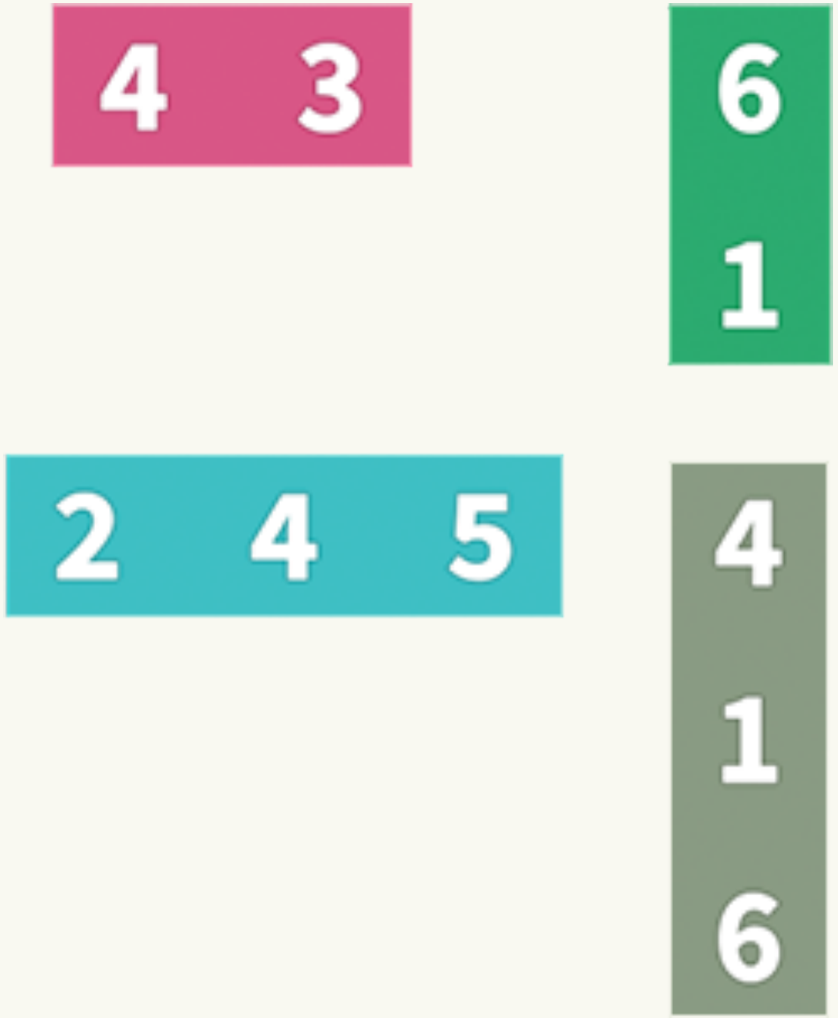
```
solver.EndSearch();
```

Slicing The Board

The Piece Library



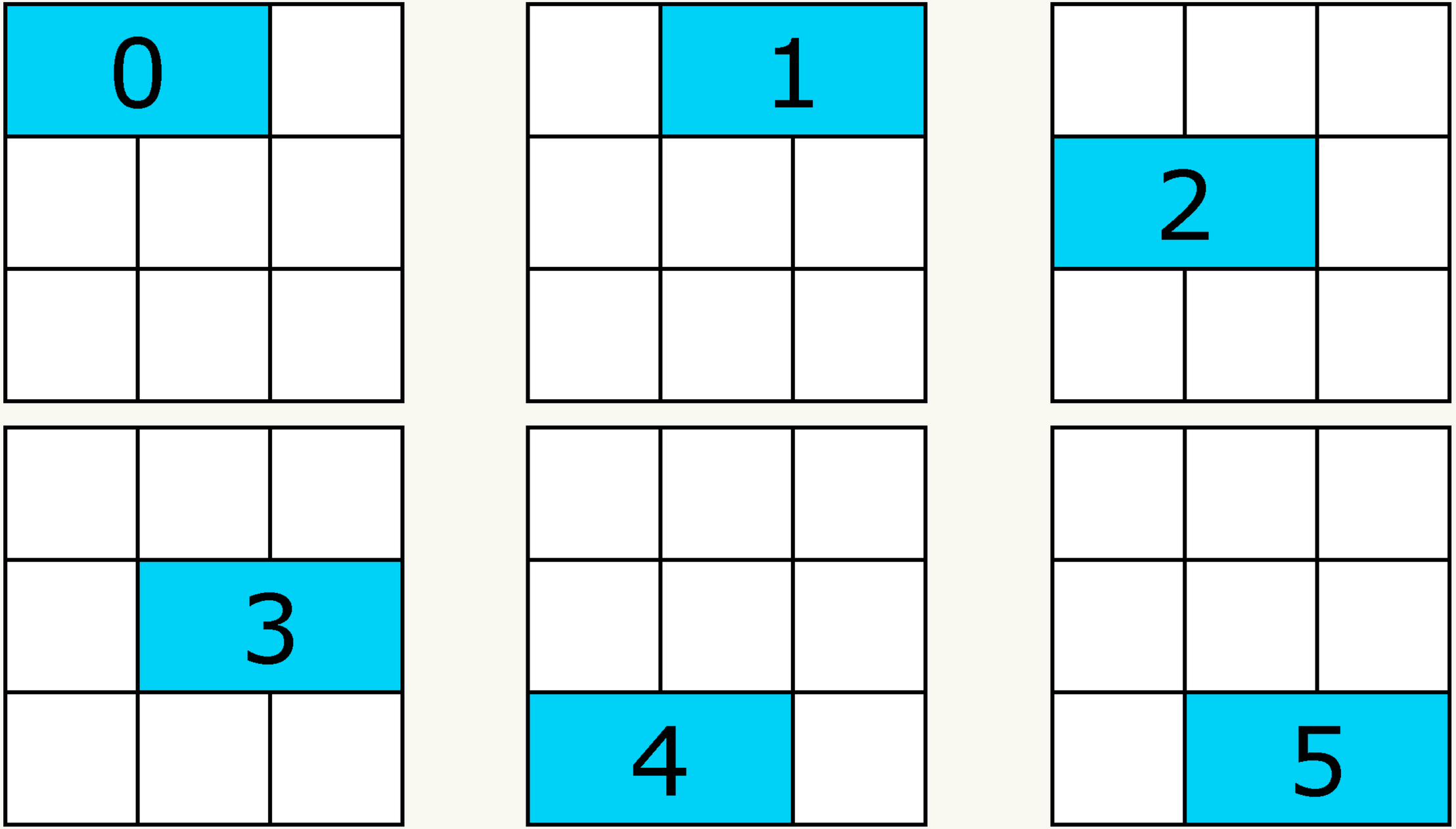
Pieces Library

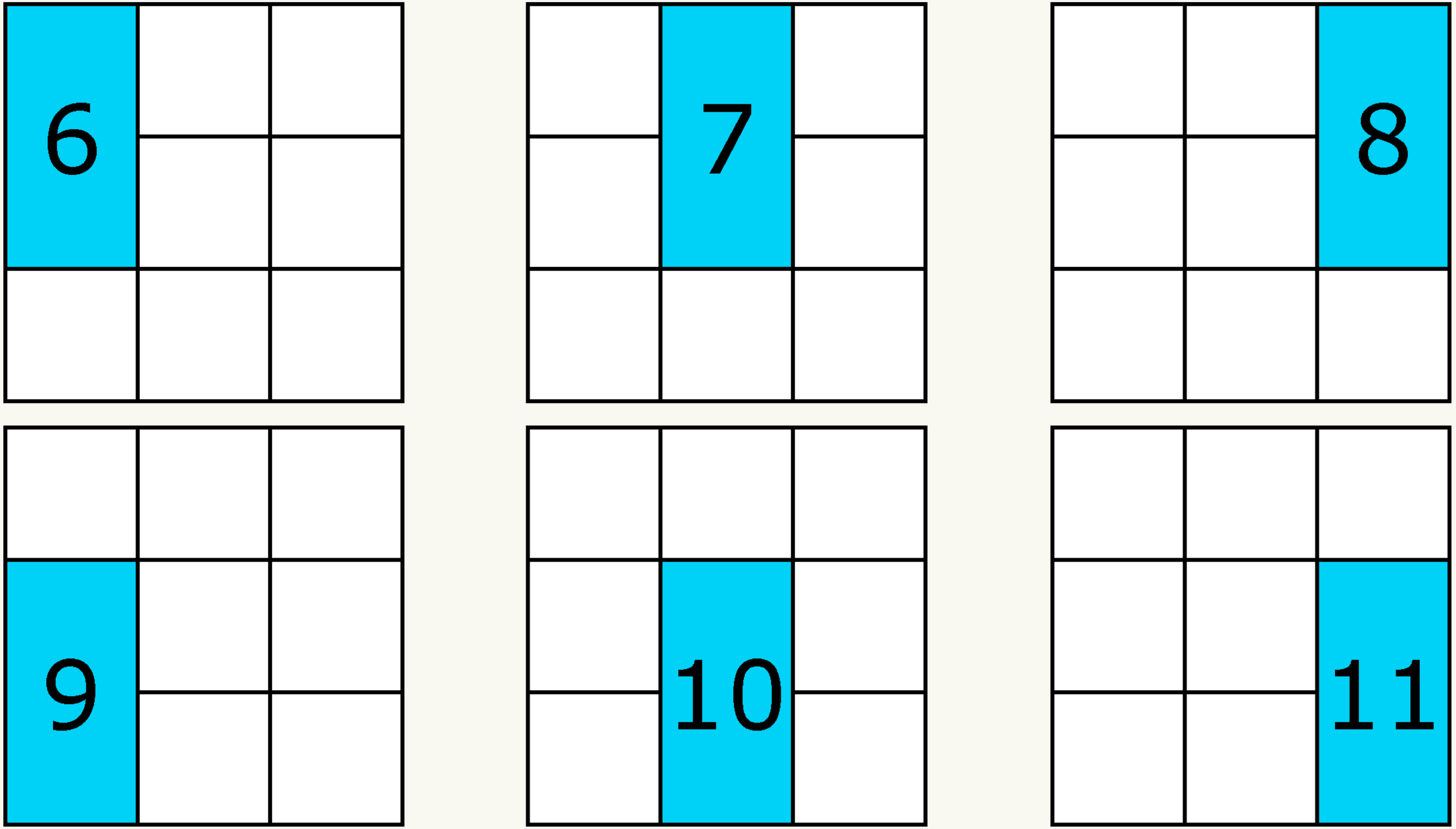


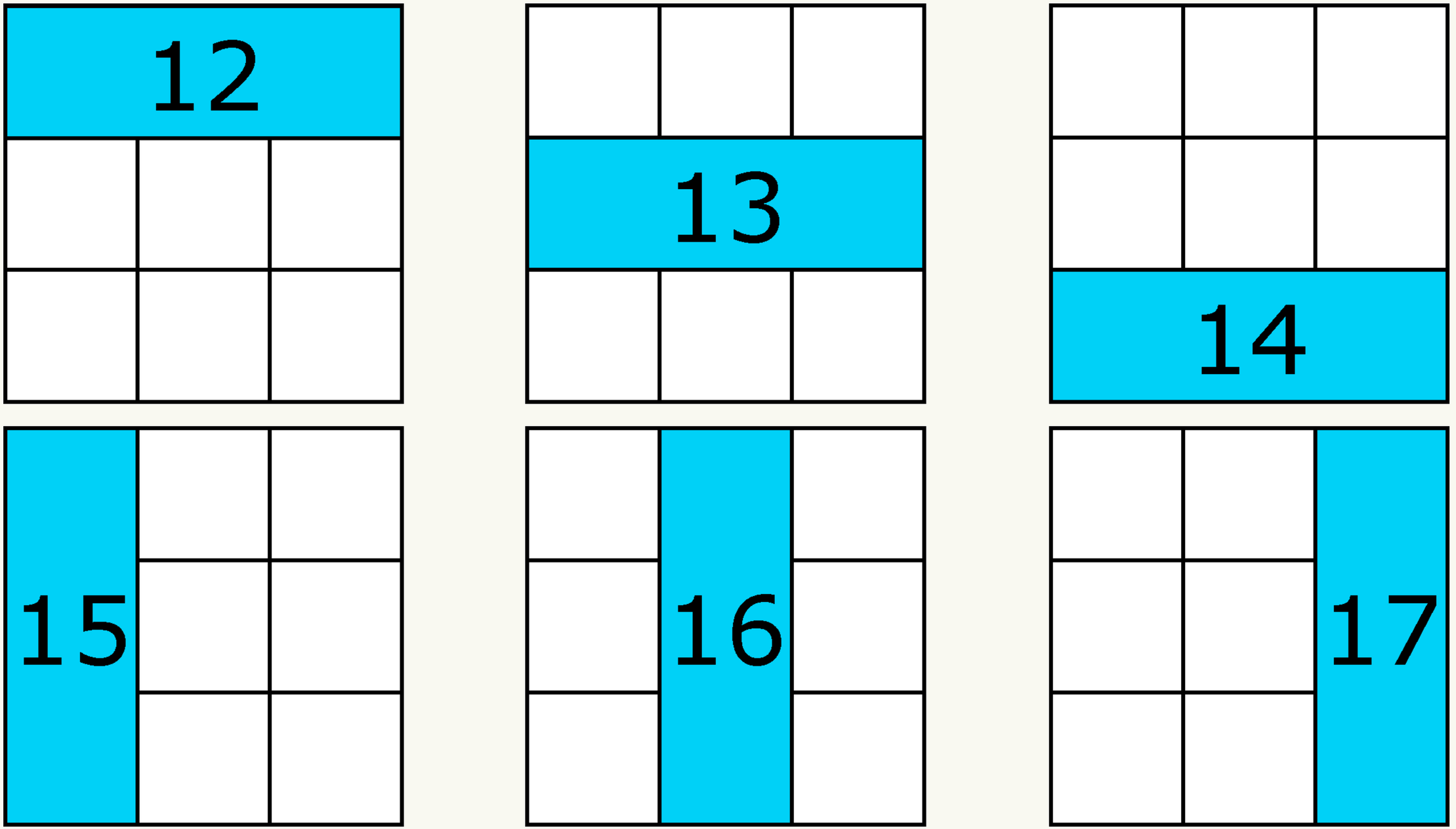
Piece Placements

Each piece placement will contain:

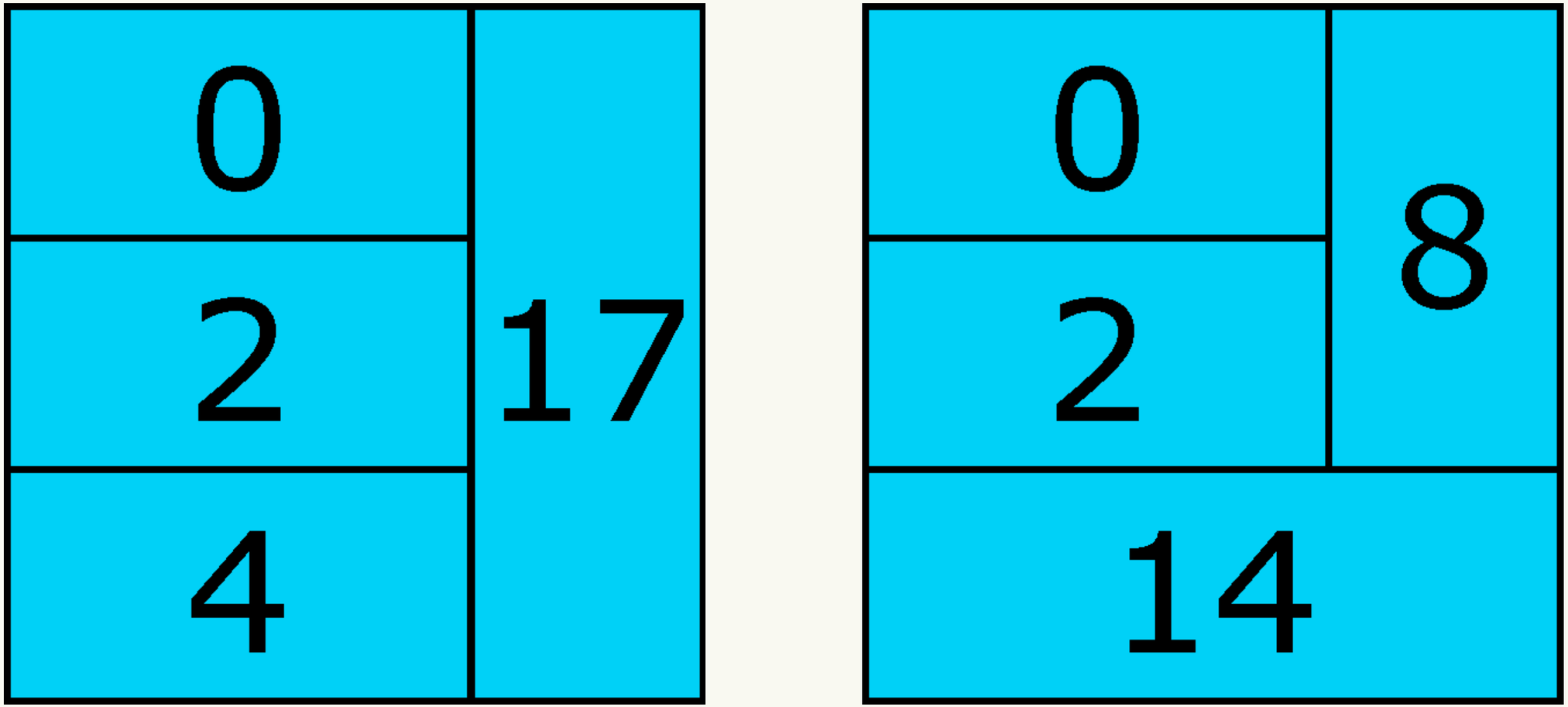
- Placement ID
- Location
- Cells the piece is on (pre-calculated)
- Piece Type


0
1
2
3
4
5


6
7
8
9
10
11


12
13
14
15
16
17

Example Output



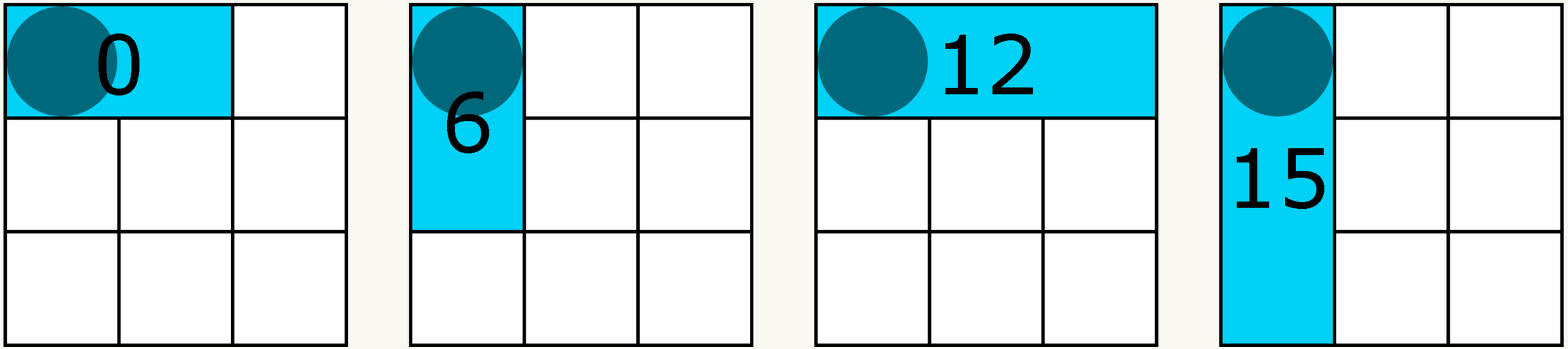
Constraint: Fill The Board

Intermediate Variables:

PlacementsForCell[0] = SUM(placement[0], placement[6], placement[12], placement[15])

Constraints:

PlacementsForCell[0] == 1



Constraint: Piece Type Count

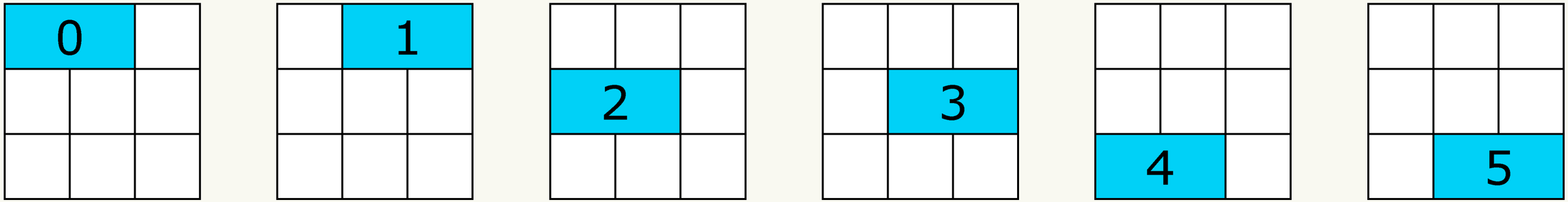
Intermediate Variables:

PieceTypeCount[0] = SUM(Placements[0-5])

Constraints:

PieceTypeCount[0] >= 0

PieceTypeCount[0] <= 3



Constraint: Piece Type Variety

Intermediate Variables:

PieceTypeUsed[0] = MAX(Placement[0-5])

TotalPieceTypesUsed = SUM(PieceTypeUsed[0-n])

Constraints:

TotalPieceTypesUsed >= 2

TotalPieceTypesUsed <= 4

```
pieceUsedVars[b] = piecePlacementVars.Max().Var();
```

Constraint: Initial Guaranteed Pieces

Intermediate Variables:

PieceTypeUnique[x] = (PieceTypeCount[x] == 1)

TotalPieceTypesUnique = SUM(PieceTypeUnique[0-n])

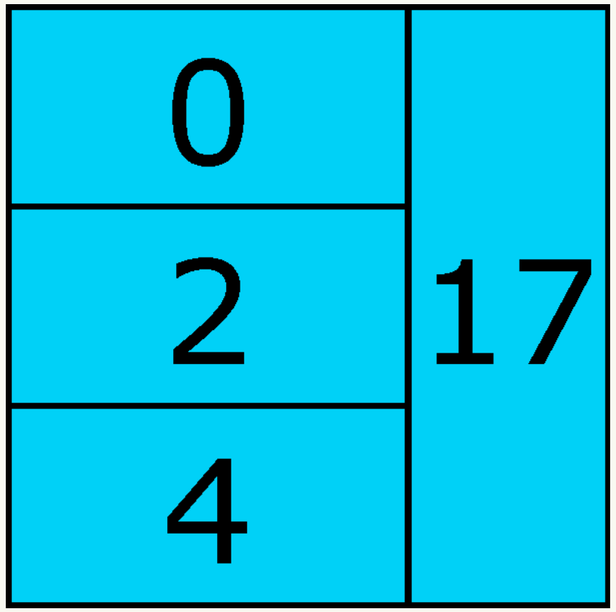
Constraints:

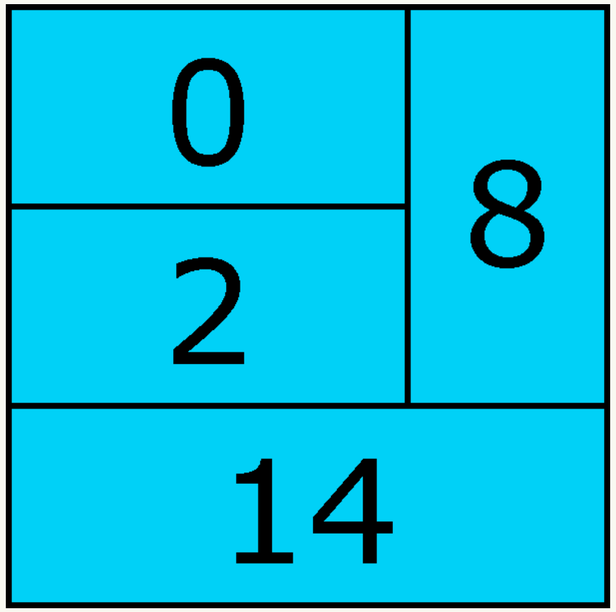
TotalPieceTypesUnique >= 1

TotalPieceTypesUnique <= 2

Extracting The Slices

Which placement variables are 1?





Things Not In The CP Models

- Exact combinatorial difficulty
  - Factorials don’t play nicely with CP
- Piece Uniqueness
  - Requires the outputs of both models
- Single Solutions
  - This is a lie, this is in a CP model.

The End

Marc Taylor
Marc@BigDuckGames.com
@DarqueFlux