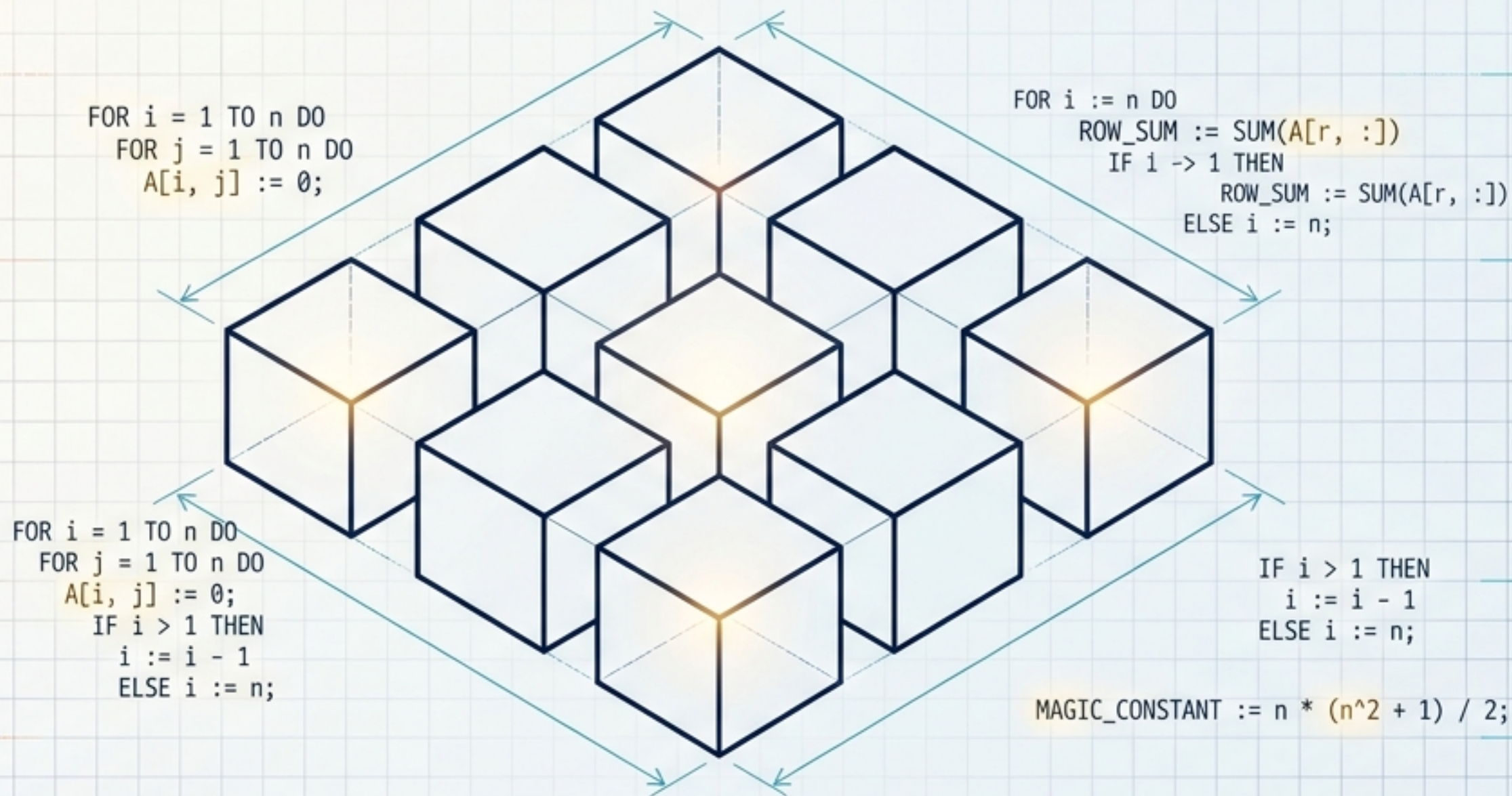


魔方陣のアルゴリズム解剖図

人間の「ルール」をコンピュータの「コード」に翻訳する思考のプロセス

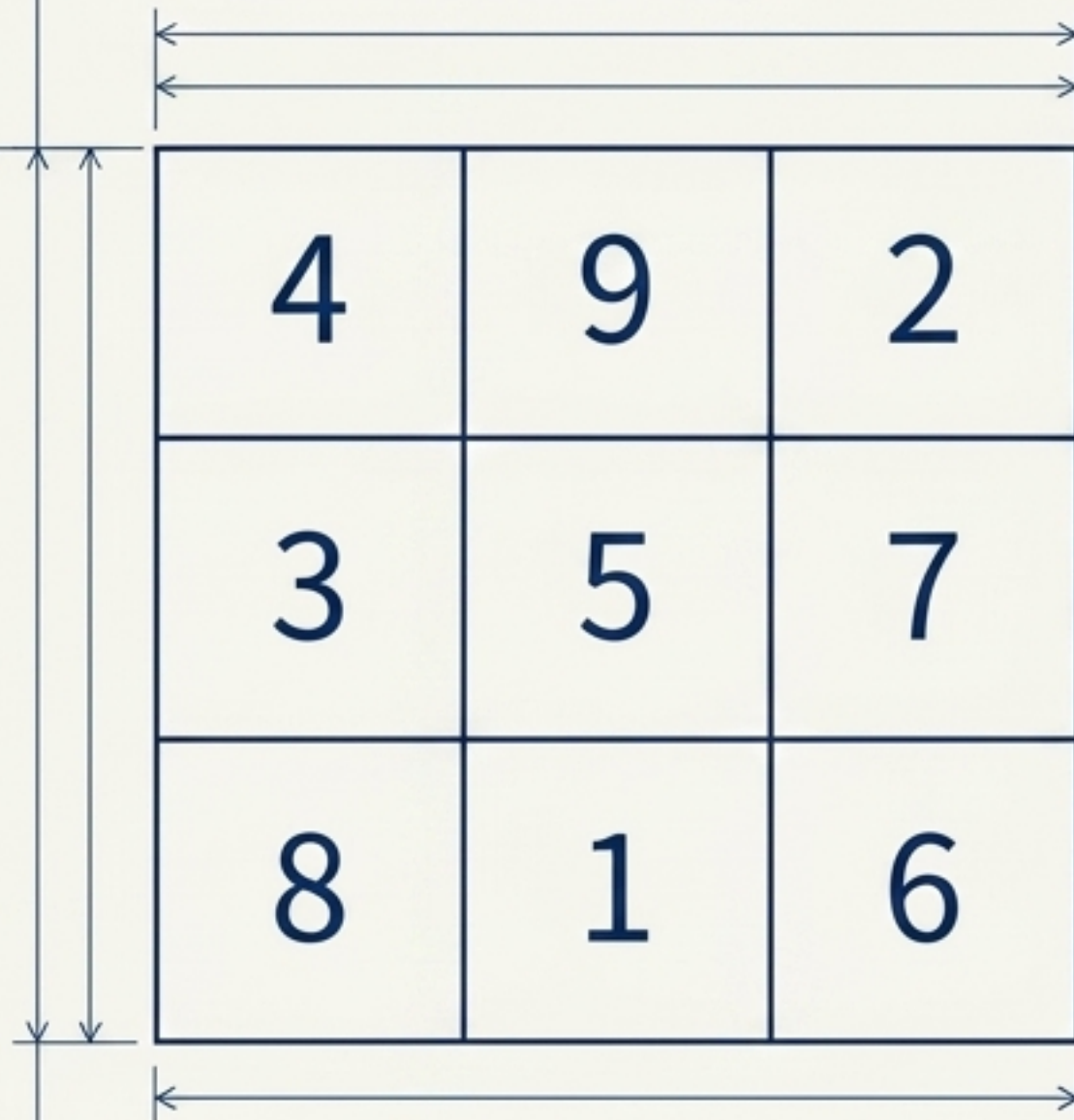


Note

【Script】

「今日は、応用情報技術者試験で出題された『魔方陣』のアルゴリズムを解読する旅に出かけましょう。このスライドの目的は、解答を丸暗記することではありません。問題文という『人間の言葉』で書かれたルールを、コンピュータが理解できる『コード』へと翻訳するプロセス、すなわちプログラマの思考のプロセス（青写真）を一緒に描き出すことです。では、設計図を広げてみましょう。」

「魔方陣」完成の条件



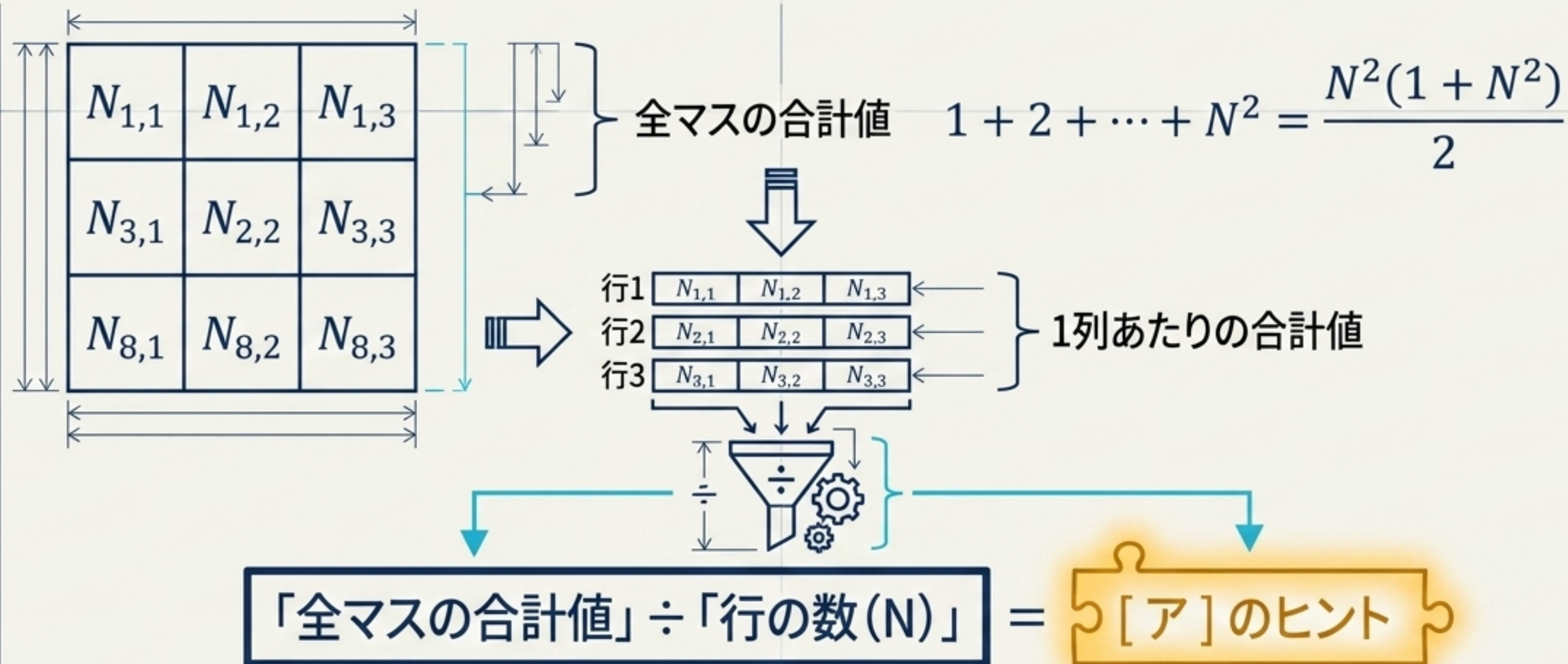
4	9	2
3	5	7
8	1	6

- ✓ 1. $N \times N$ のマス目 (N は奇数)
- ✓ 2. 1から N^2 までの数字を1つずつ配置
- ✓ 3. 縦・横・斜めの合計が「すべて同じ」

【Script】

「まずは完成図の確認です。魔方陣とは、正方形のマス目に数字を並べ、縦、横、斜め、どの列を足しても合計が同じになるパズルのことです。ここでは例として、 N が3、つまり 3×3 の魔方陣を見えています。マスは全部で9つ。1から9までの数字を一度ずつ使い、すべての列の合計を一致させる。これが私たちのプログラムが目指す最終ゴールです。」

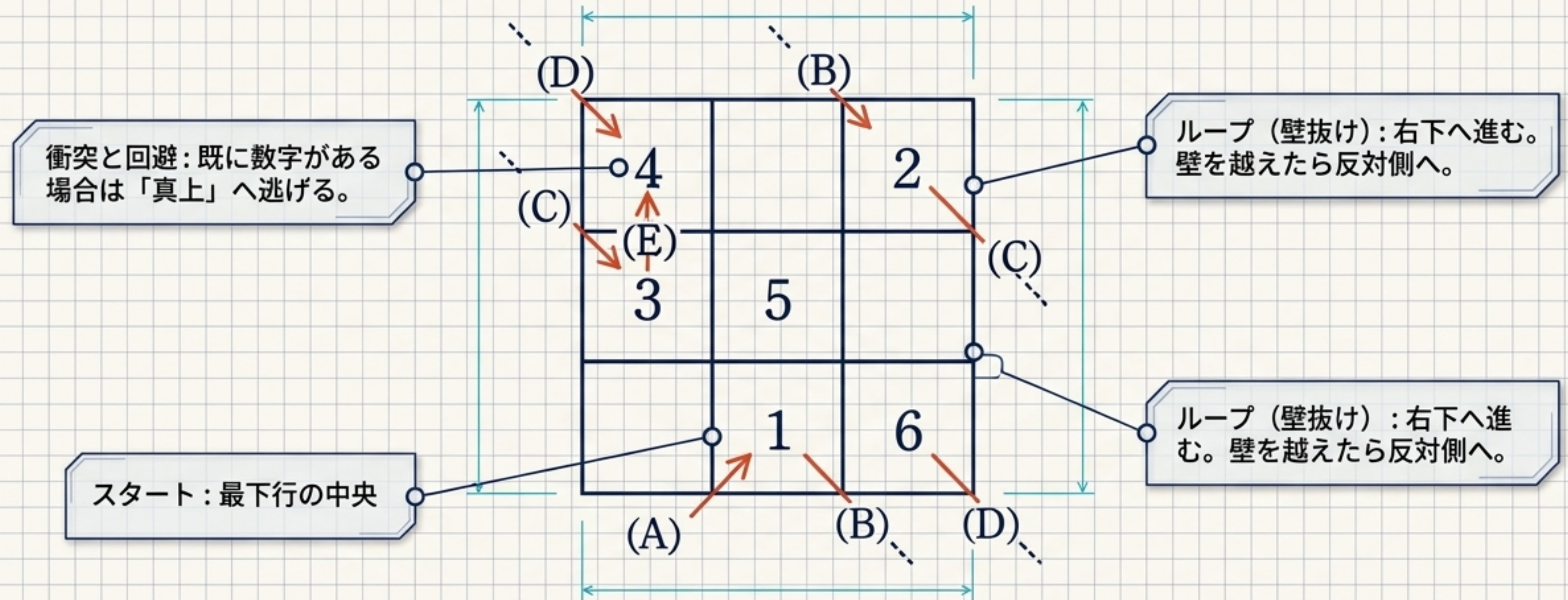
設問[ア]へのアプローチ：全体から部分を逆算する



【Script】

「設問のアでは、対角線の合計値を求める式が問われています。ここで『魔法の公式』を暗記する必要はありません。論理的に考えましょう。まず、盤面全体に入っている数字の総和を求めます。次に、魔方陣にはN個の行があり、それぞれの合計は等しいという性質を利用します。つまり、全体の総和を、行の数であるNで『割り算』すれば、1列あたりの合計が出ます。問題文の式に合うように整理してみましょう。」

アルゴリズムのルール（人間の思考）



【Script】

「次に、この魔方陣を『どう作るか』というルールを視覚化します。問題文には細かな手順が書かれていますが、動きのパターンはシンプルです。スタートが一番下の中央。基本は『右下』へ進み続けます。盤面の端を突抜けた場合は、パックマンのように反対側から出てきます。そして、移動しようとした場所にすでに数字が置かれていた場合、衝突を避けるために『一つ上』に移動します。この人間の頭の中にあるルールを、どうプログラムにするかが腕の見せ所です。」

ルールをデータ構造にマッピングする（番兵の配置）

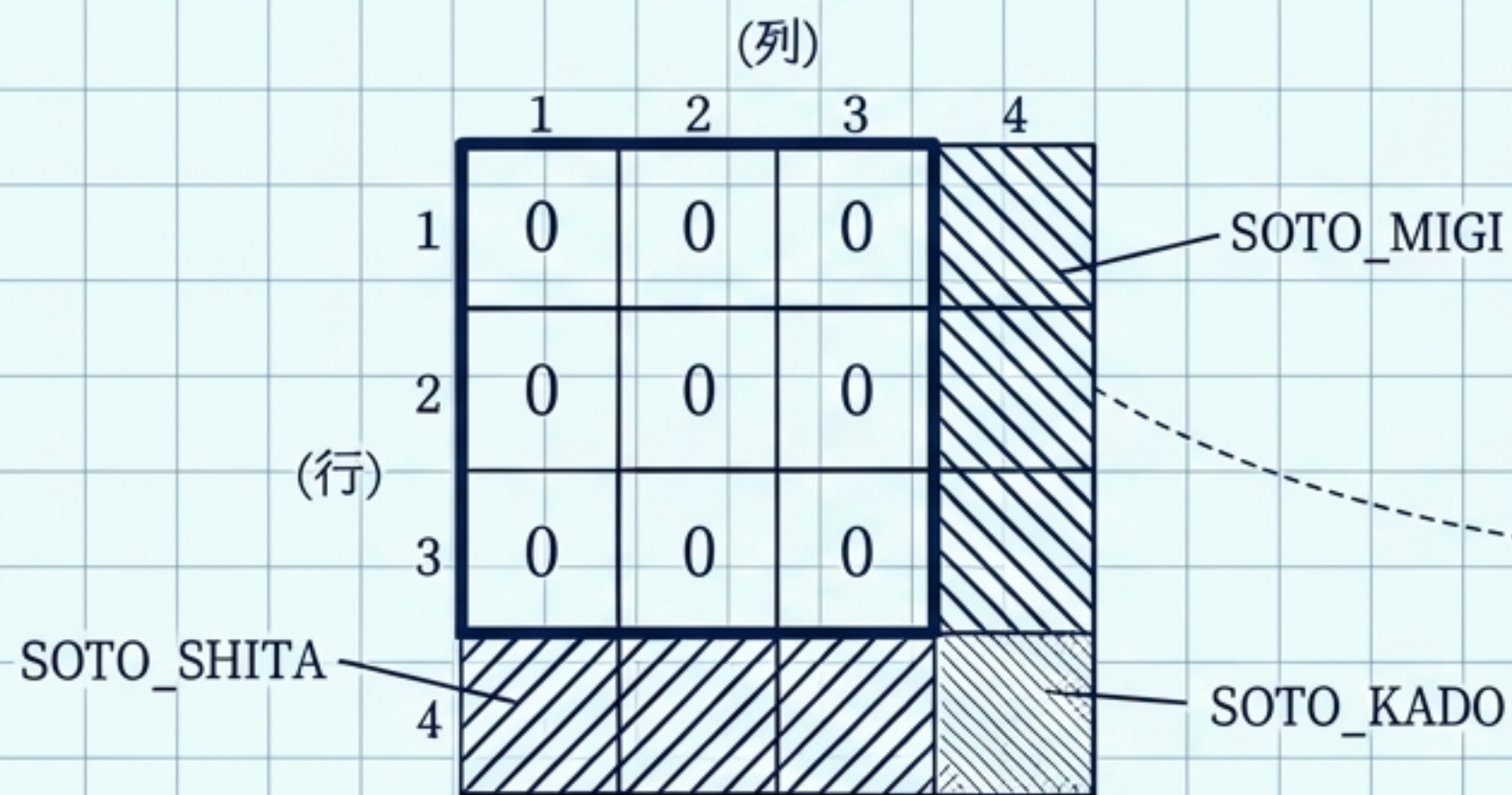


図3 配列 `houjin` の初期値 (N が 3 の場合)

人間の認識	プログラムの実装
盤面の外壁（壁抜け判定用）	($N+1$)サイズの配列と「番兵 (<code>SOTO_***</code>)」定数

【Script】

「プログラムの世界に翻訳するための第一歩が、データ構造の設計です。盤面の外側にはみ出したことをコンピュータにどう気付かせるか？ここでは『番兵』というテクニックが使われています。本来の $N \times N$ の盤面よりも1回り大きい $(N+1) \times (N+1)$ の配列を用意し、外側のマスに『ここは外の右側だよ』『ここは外の下側だよ』という目印をあらかじめ置いておくのです。」

設問[イ][ウ]のヒント：番兵の「座標」を特定する

```
function shokika(N)
  for( y を 1 から N まで 1 ずつ増やす )
    for( x を 1 から N まで 1 ずつ増やす )
      houjin[y][x] ← 0
    endfor
    [イ] ← SOTO_MIGI
  endfor
  for( x を 1 から N まで 1 ずつ増やす )
    [ウ] ← SOTO_SHITA
  endfor
  ...
endfunction
```

y行目の「一番右の列 (Nの次)」は
x = [?]

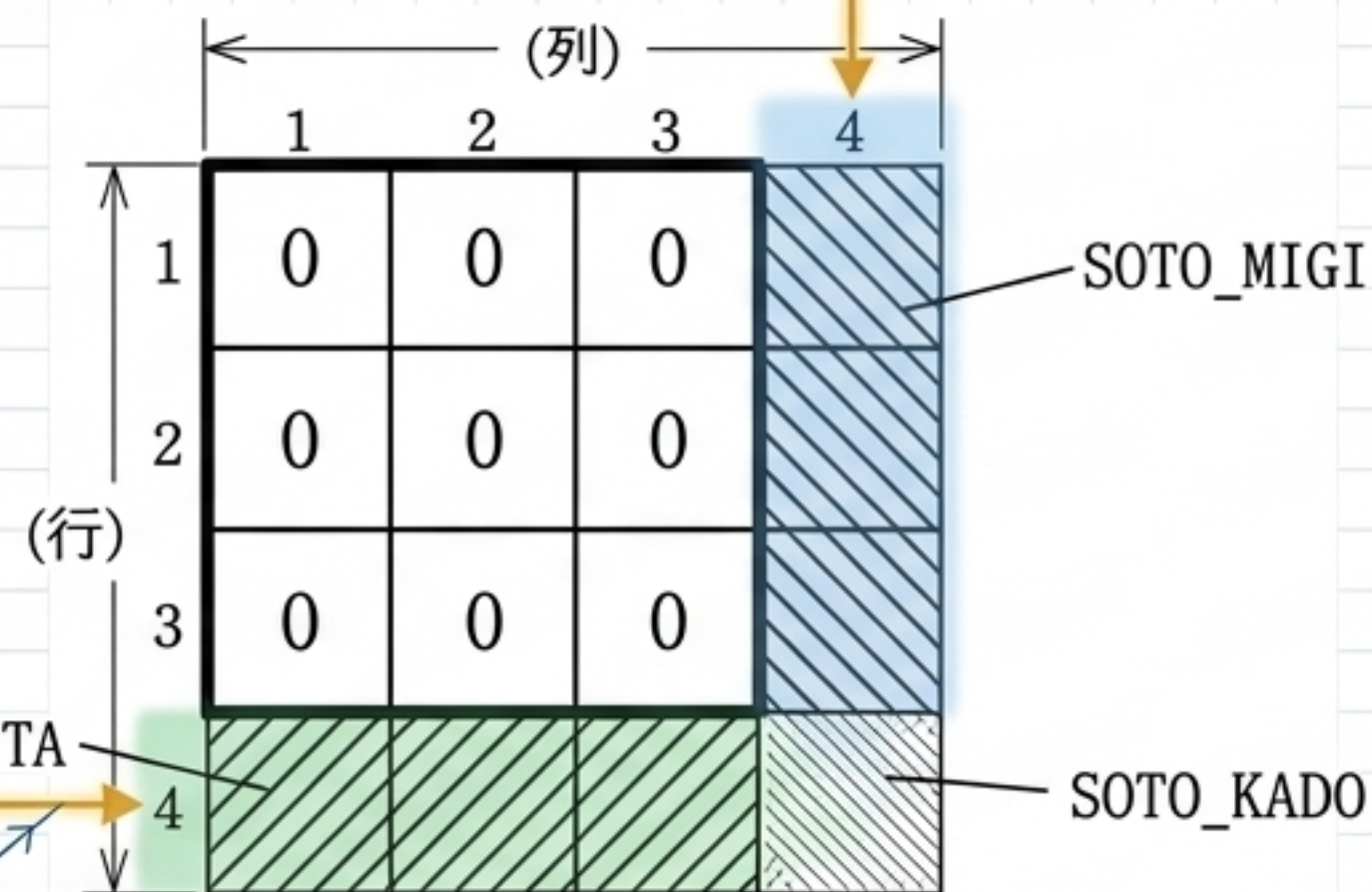
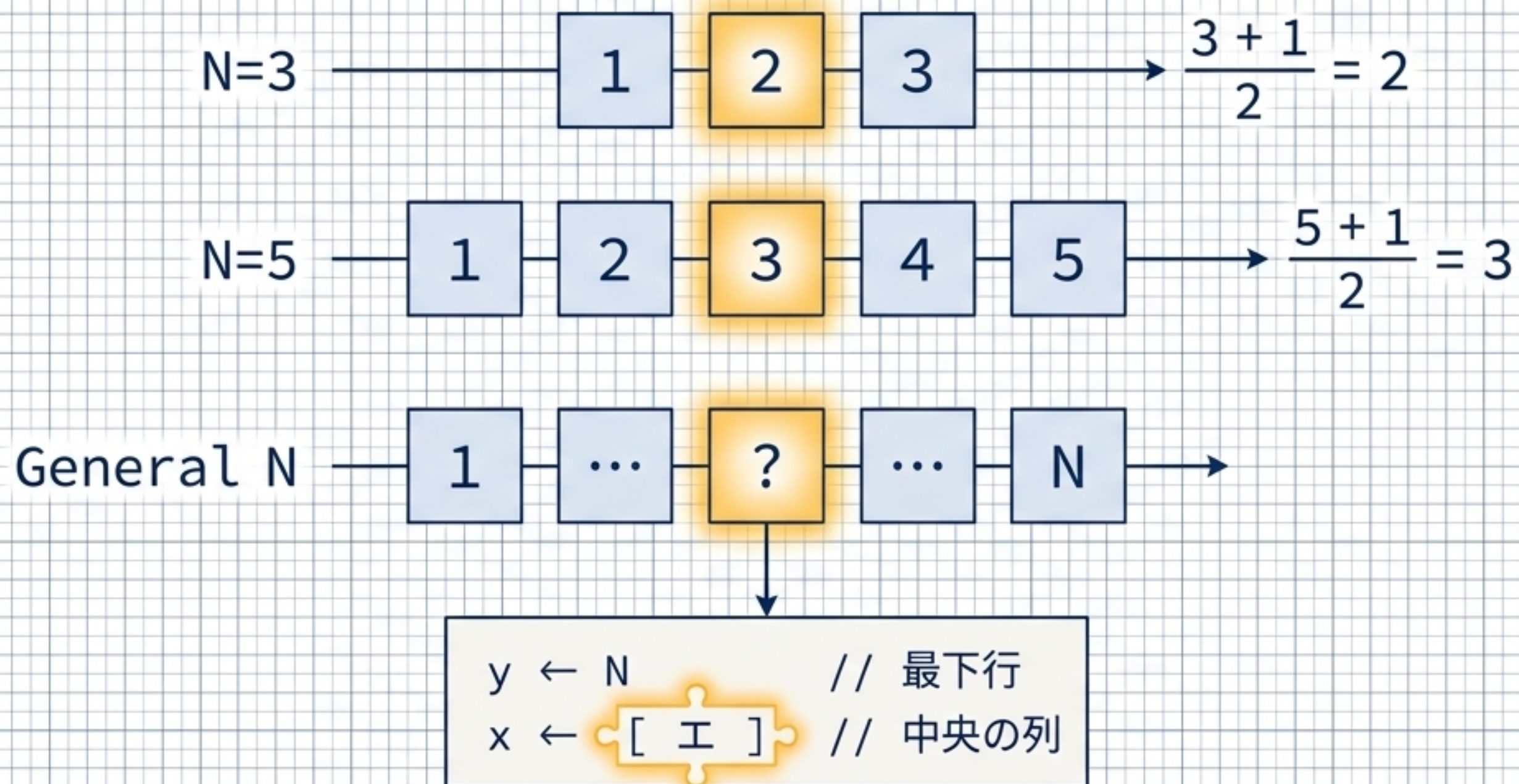


図3 配列 houjin の初期値 (N が 3 の場合)

【Script】

「初期化のプログラムを見てみましょう。空欄イとウには、番兵を配置する配列の要素が入ります。右の図と見比べてください。SOTO_MIGIは、各行の『一番右側』の列、つまりN列目の『次』の列に配置されていますね。同様にSOTO_SHITAは、各列の『一番下』の行、N行目の『次』の行に配置されます。ループ変数yとxに対して、固定されるべき添字はいくつになるのでしょうか？」

設問[エ]のヒント：スタート地点の「中央」を数式で表す

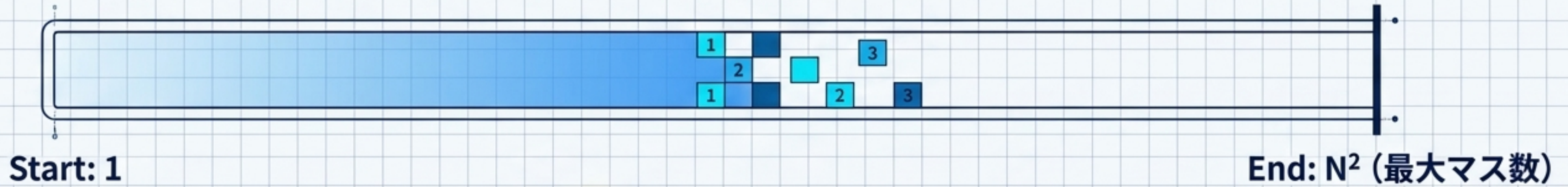


Note Panel

【Script】

「いよいよ魔方陣を作り始めます。最初の数字『1』を置くスタート位置は、『最下行の中央』でした。行の変数yは最初からNに設定されています。では、列の変数xはどうでしょう？Nが3の時は2列目、Nが5の時は3列目が中央になります。この『Nから中央のインデックスを導き出す計算式』が空欄エに入ります。」

設問[オ]のヒント：いつ作業を終了するのか？



while(suuji が [オ])

...

suuji ← suuji + 1

suujiが配置すべき最大の数字（全マス数）に到達する「直前」まで繰り返す。

【Script】「配置が始まりました。このwhileループはいつまで続くのでしょうか？ループの最後で数字（suuji）に1が足され、次の数字へ準備が進むことに注目してください。盤面のマス目は全部で $N \times N$ 個、つまり最大値は N の2乗です。『次に書く数字』が N の2乗以下の間は、まだ空白が残っているということ。この継続条件を論理式で表してみてください。」

コードの心臓部：移動とワープの処理

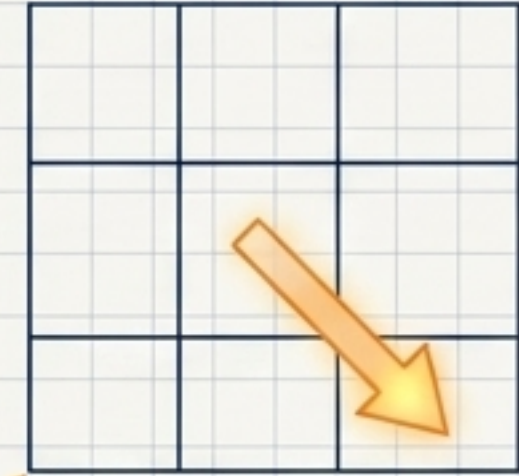
```
yb ← y, xb ← x
```

```
y ← y+1, x ← x+1
```

```
if(houjin... == SOTO_SHITA)  
  y ← 1
```



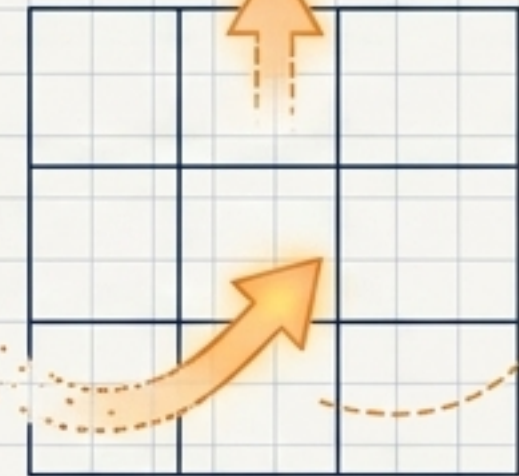
退避（セーブ）



右下へ移動



右下へ移動



ワープ（壁抜け）

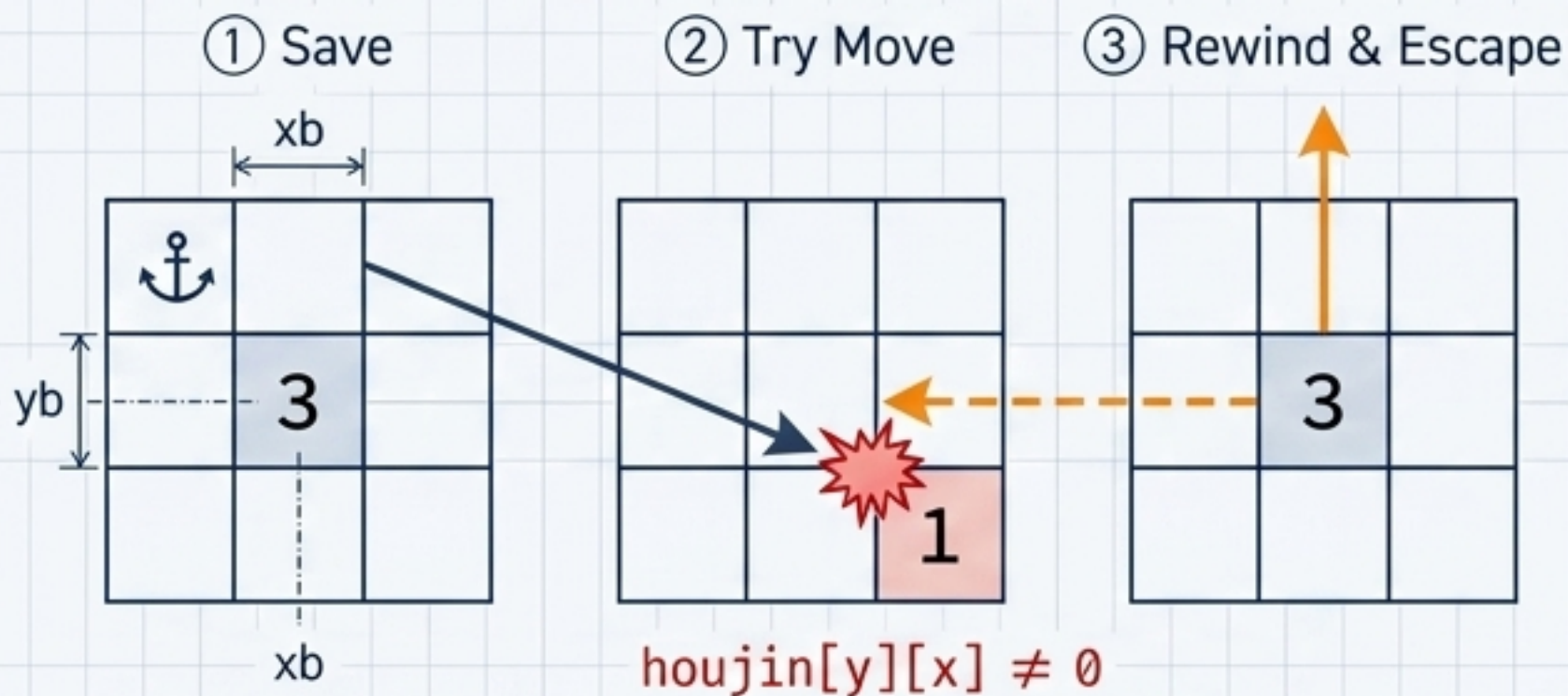
【Script】

「ループの中身、移動の処理を見てみましょう。まず現在の位置をybとxbに『退避（セーブ）』します。そしてy+1、x+1として右下へ移動します。もし移動した先が先ほど配置した『番兵』だったら？if文がそれを検知し、yを1に書き換えることで『反対側からのワープ』を見事に実現しています。これがコンピュータの壁抜けのロジックです。」

Note Panel

設問[力][キ]のヒント：衝突時の回避ルート（真上へ逃げる）

Human Thought



Computer Logic

if (houjin[y][x] が 0 と等しくない)

y ← [力]

x ← [キ]

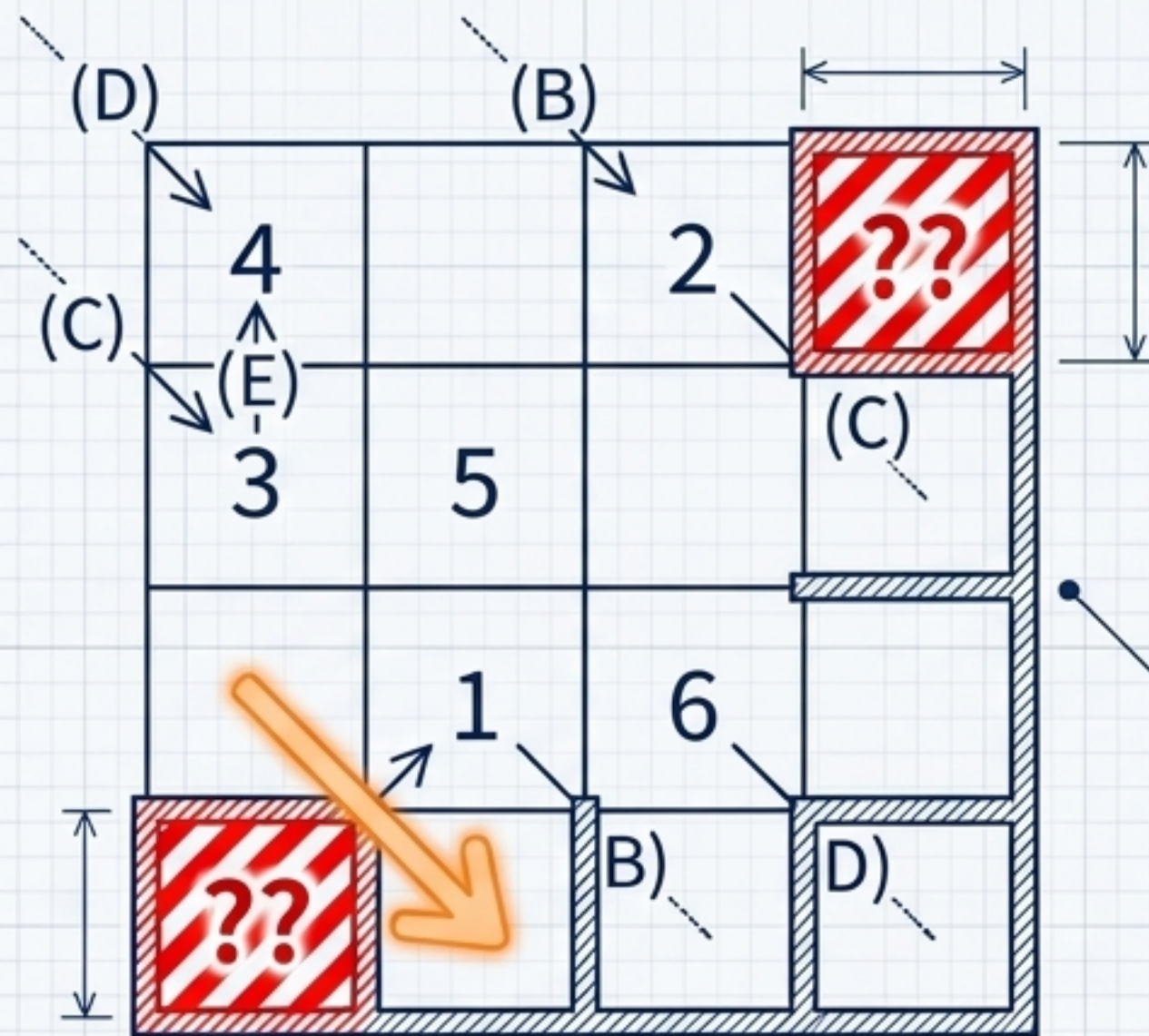
endif

Note Panel

【Script】

「しかし、移動先にすでに別の数字が置かれていたらどうなるでしょう？コード上の『0と等しくない』という条件がこれに当たります。この時、ルールでは『元の現在位置の真上』に移動しなければなりません。ここで活きるのが、移動前にセーブしておいた元の座標、ybとxbです。退避させておいた座標を基準にして、『真上（行を1つ減らす、列はそのまま）』を表現してみてください。」

設問2(2)のヒント：決して踏まれない「番兵」はどこか？



右下（斜め）への移動と、上（垂直）への回避しか存在しない。絶対にたどり着けない外壁マスが存在する。

Note Panel

【Script】「設問2の(2)は、配列の中で『一度も参照されないマス』を探す面白い問題です。現在位置から移動する方向は常に『右下』か、衝突回避の『真上』だけです。図を使って軌跡を追ってみてください。右下に向かって進む限り、絶対にたどり着かない番兵（SOTO_MIGIやSOTO_SHITAの一部）が2箇所、死角として存在することに気づくはずです。」

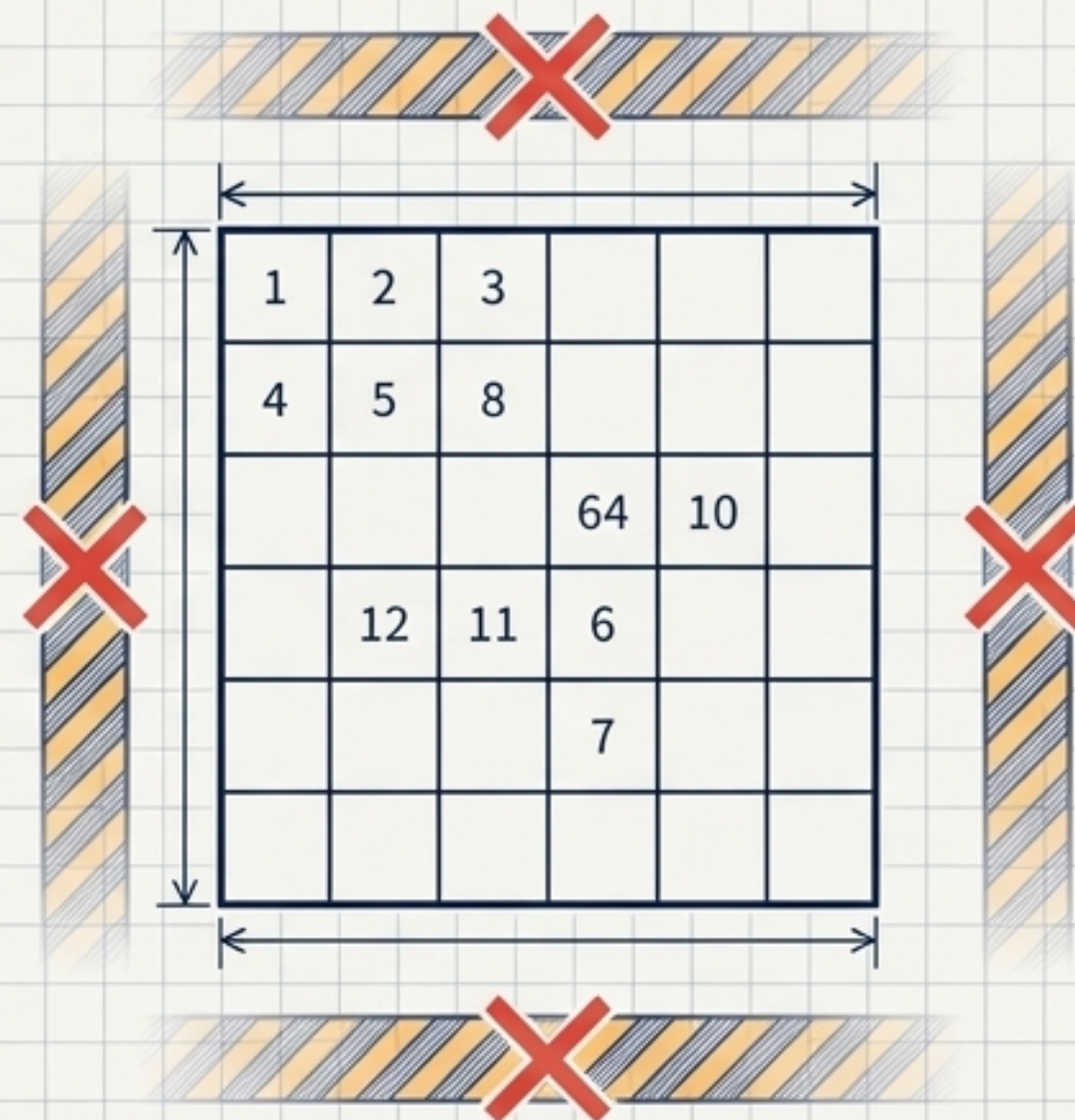
プログラムの改修：メモリの節約



番兵を削除し、
配列を最適化

Computer Logic

After: 番兵なし (メモリ節約)



Note Panel

【Script】

「さて、アルゴリズムは完成しましたが、優秀なプログラマはさらにコードを磨きます。設問3ではメモリ使用量を削減するため、配列をN×Nに縮小し、『番兵』を廃止する改修を行います。番兵という目印がなくなった今、プログラムはどうやって『壁を越えた』ことを検知すればよいのでしょうか？」

設問[ク][ケ]のヒント：座標そのもので境界を判定する

Human Thought (mental models, sketches)

$y \leftarrow y + 1$

$x \leftarrow x + 1$

if(y が **ク** よりも大きい)

$y \leftarrow$ **ケ**

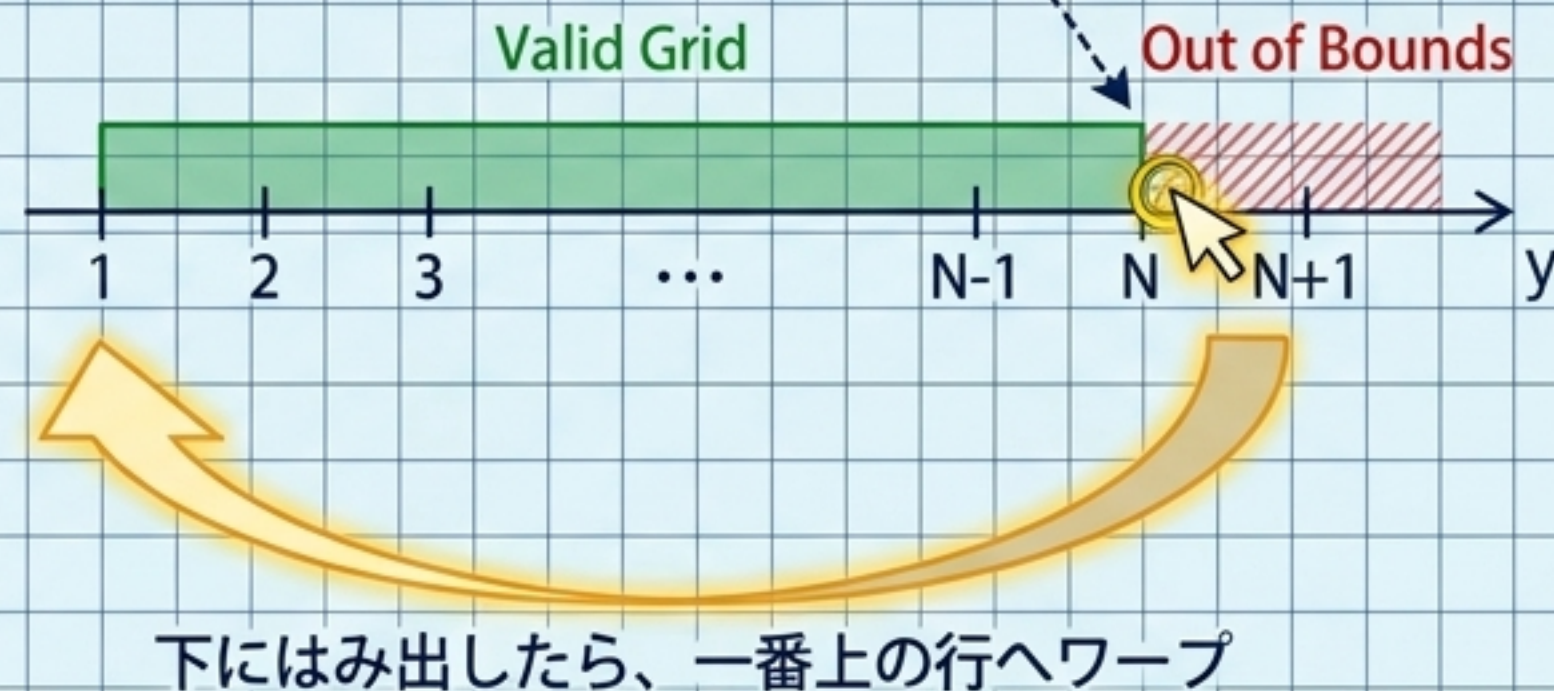
endif

if(x が **ク** よりも大きい)

$x \leftarrow$ **ケ**

endif

Computer Logic (arrays, variables, code snippets)

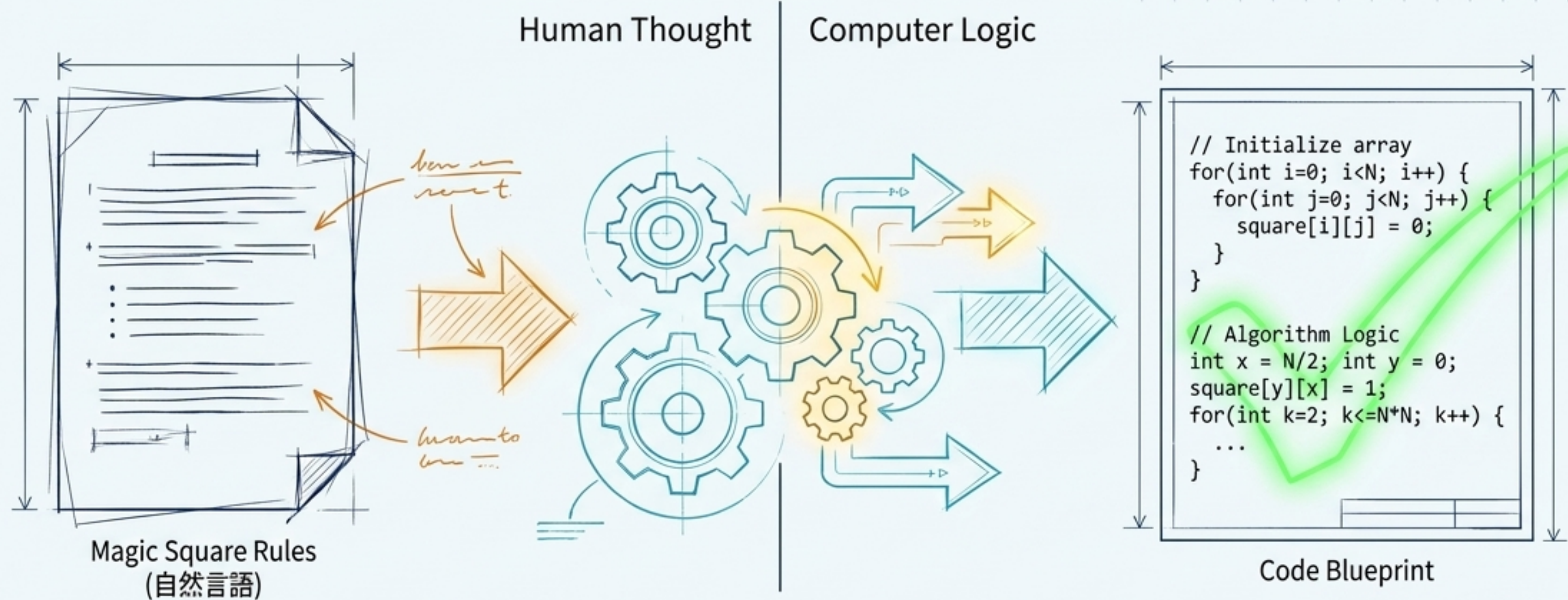


Note Panel

【Script】

「新しいプログラムを見てください。番兵がない代わりに、座標の値そのもので判定を行っています。
 y や x が『どこ』より大きくなったら、盤面をはみ出したと見なせるでしょうか？盤面の最大インデックスは N ですね。
そして、下にはみ出した場合、ワープする先は『一番上の行』です。これを変数と定数で表現すれば、改修は完了です。」

まとめ：ルールの意図を読み解く力



自然言語の制約を「数式」へ

物理的な動作を「配列と変数」へ

不要な要素を削り「最適化」へ

【Script】

「お疲れ様でした。ルールをコードに落とし込み、さらに最適化するまでのプロセスをたどりました。応用情報技術者試験のアルゴリズム問題で真に問われているのは、文法的な知識だけではありません。人間の言葉で書かれた要件を、いかにコンピュータのデータ構造とロジックにマッピングするかという『設計力』です。ぜひ、この視座を持ってご自身で解答を導き出してみてください。」