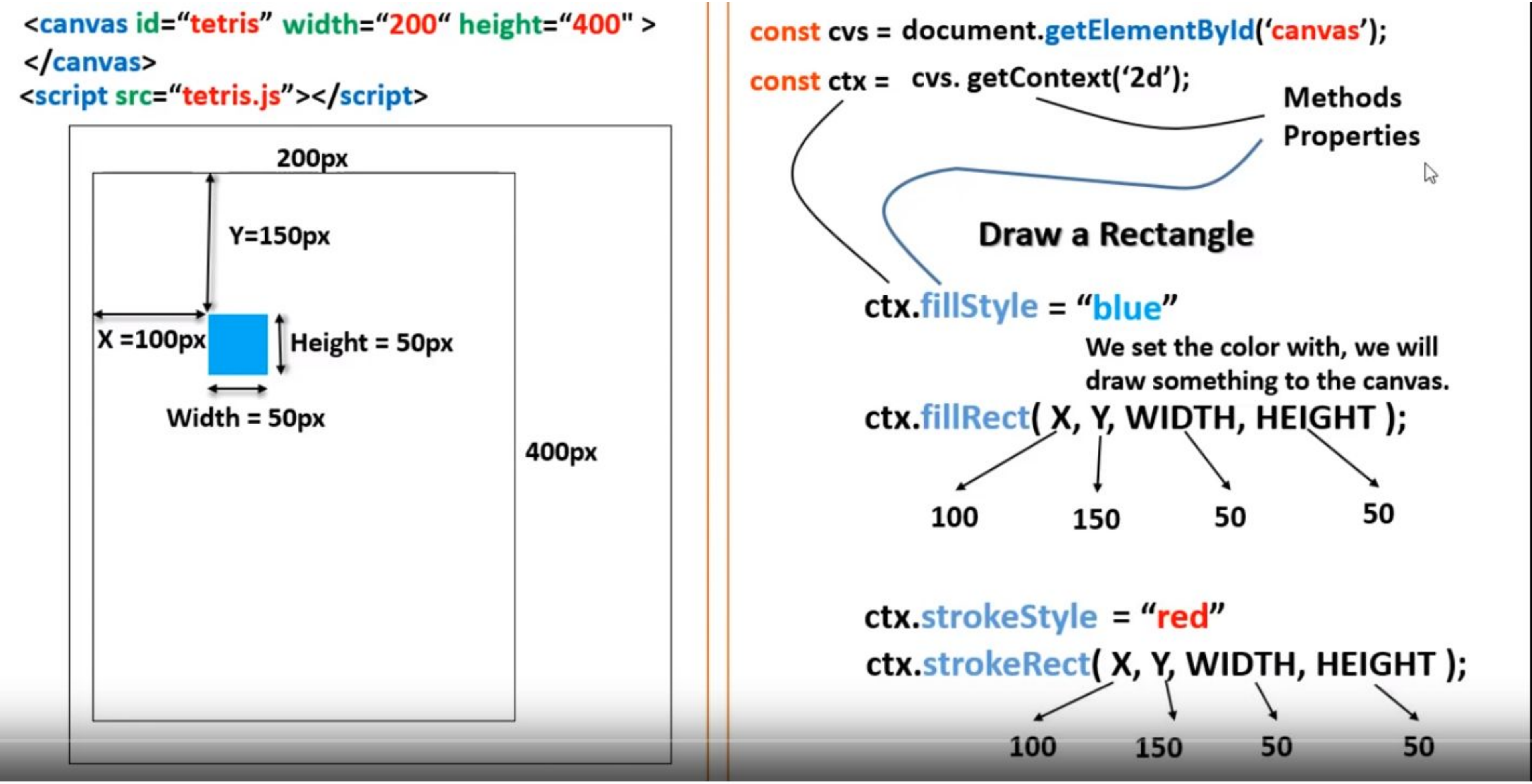
UD04 Práctica Tetris

01 de Octubre de 2019

Uso de canvas



```
<canvas id=“tetris” width=“200“ height=“400" >
</canvas>
<script src=“tetris.js”></script>
```

```
const cvs = document.getElementById(‘canvas’);
const ctx =  cvs. getContext(‘2d’);
```

Draw a Rectangle

```
ctx.fillStyle = “blue”
```

We set the color with, we will draw something to the canvas.

```
ctx.fillRect( X, Y, WIDTH, HEIGHT );
```

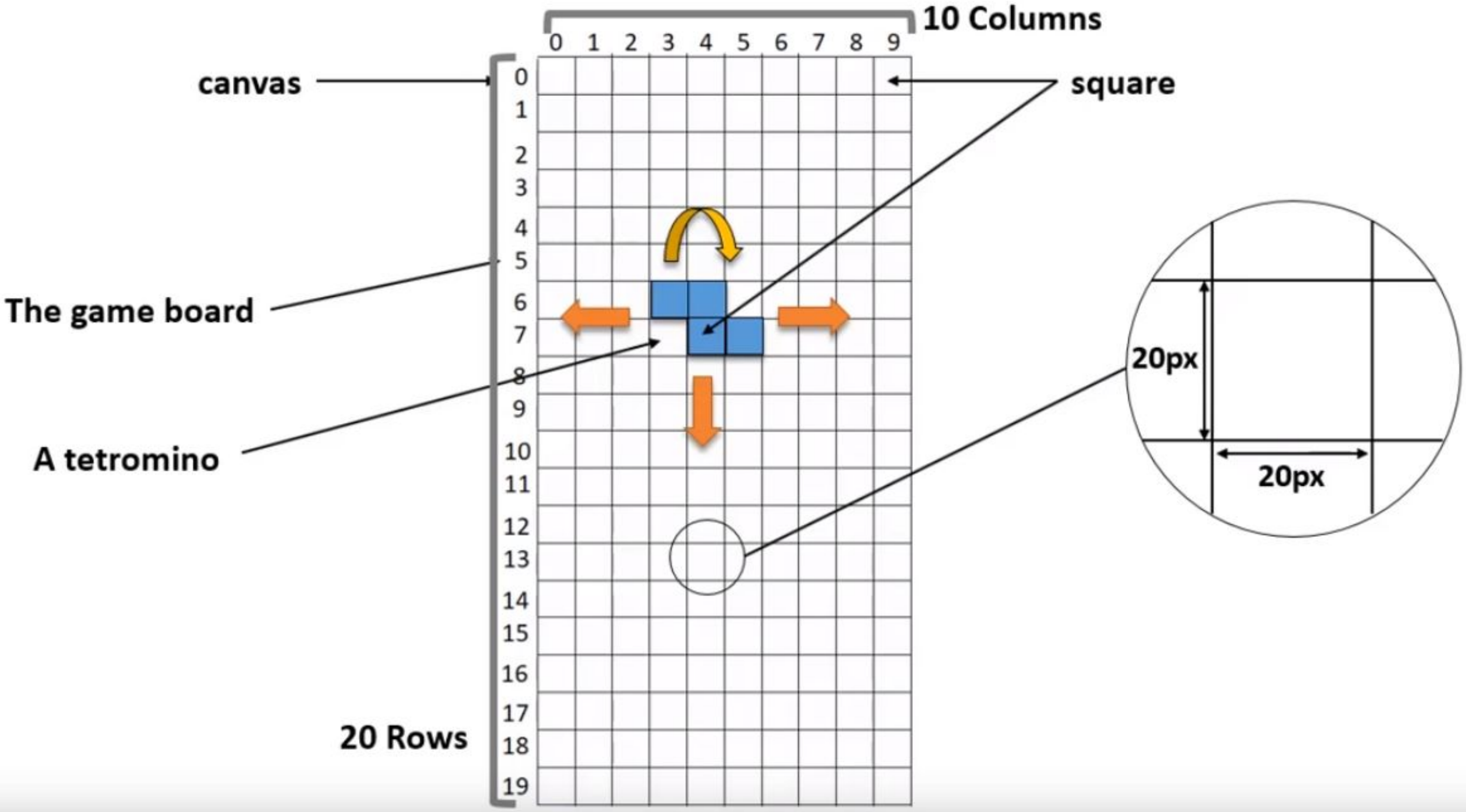
100 150 50 50

```
ctx.strokeStyle  = “red”
ctx.strokeRect( X, Y, WIDTH, HEIGHT );
```

100 150 50 50

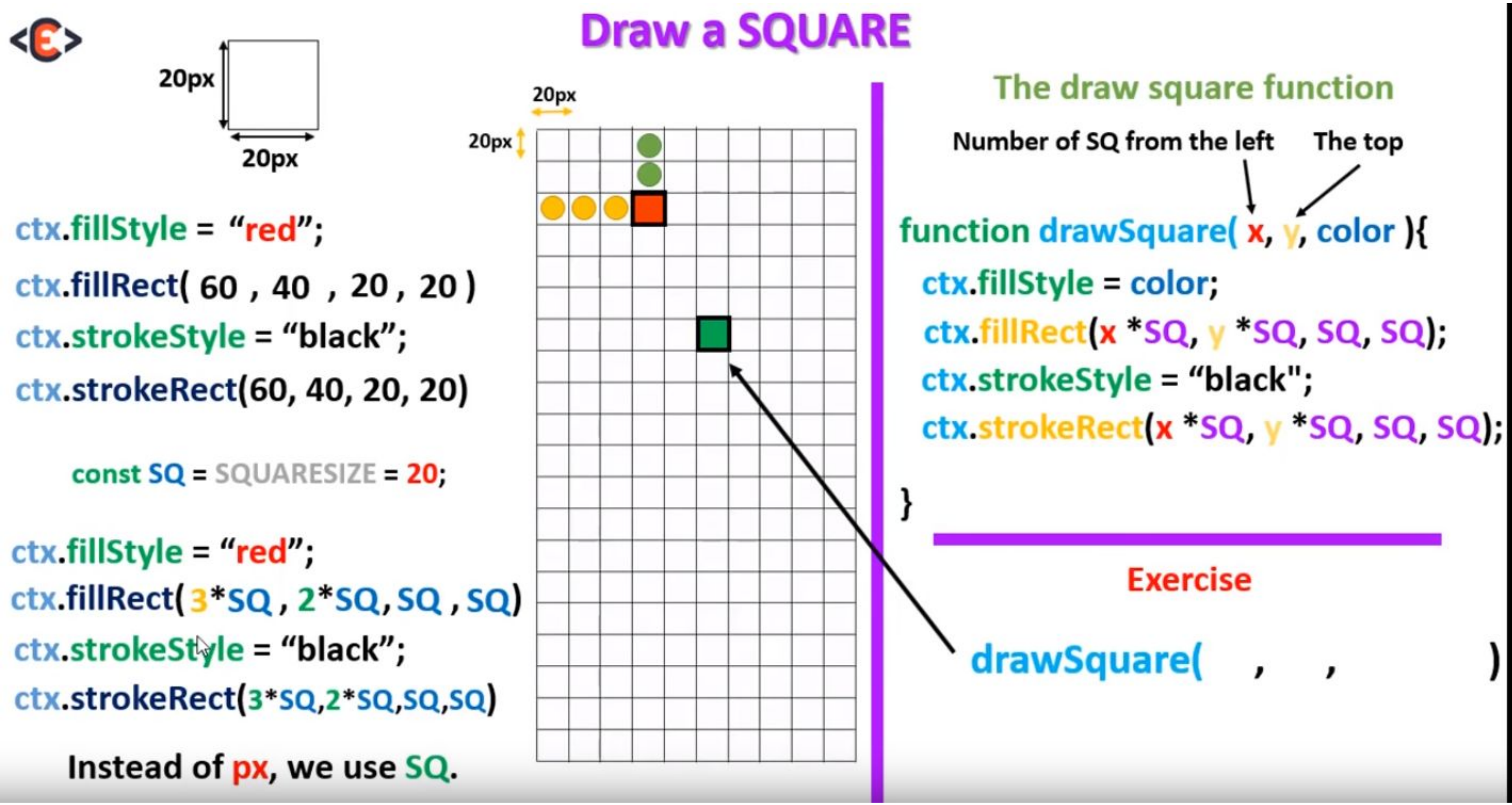
cc BY NC ND

Juego tetris



BY NC ND

Dibujar un cuadrado



Draw a SQUARE

```
ctx.fillStyle = “red”;
ctx.fillRect( 60 , 40  , 20 , 20 )
ctx.strokeStyle = “black”;
ctx.strokeRect(60, 40, 20, 20)
```

```
const SQ = SQUARESIZE = 20;
```

```
ctx.fillStyle = “red”;
ctx.fillRect( 3*SQ , 2*SQ, SQ , SQ)
ctx.strokeStyle = “black”;
ctx.strokeRect(3*SQ,2*SQ,SQ,SQ)
```

Instead of px, we use SQ.

The draw square function

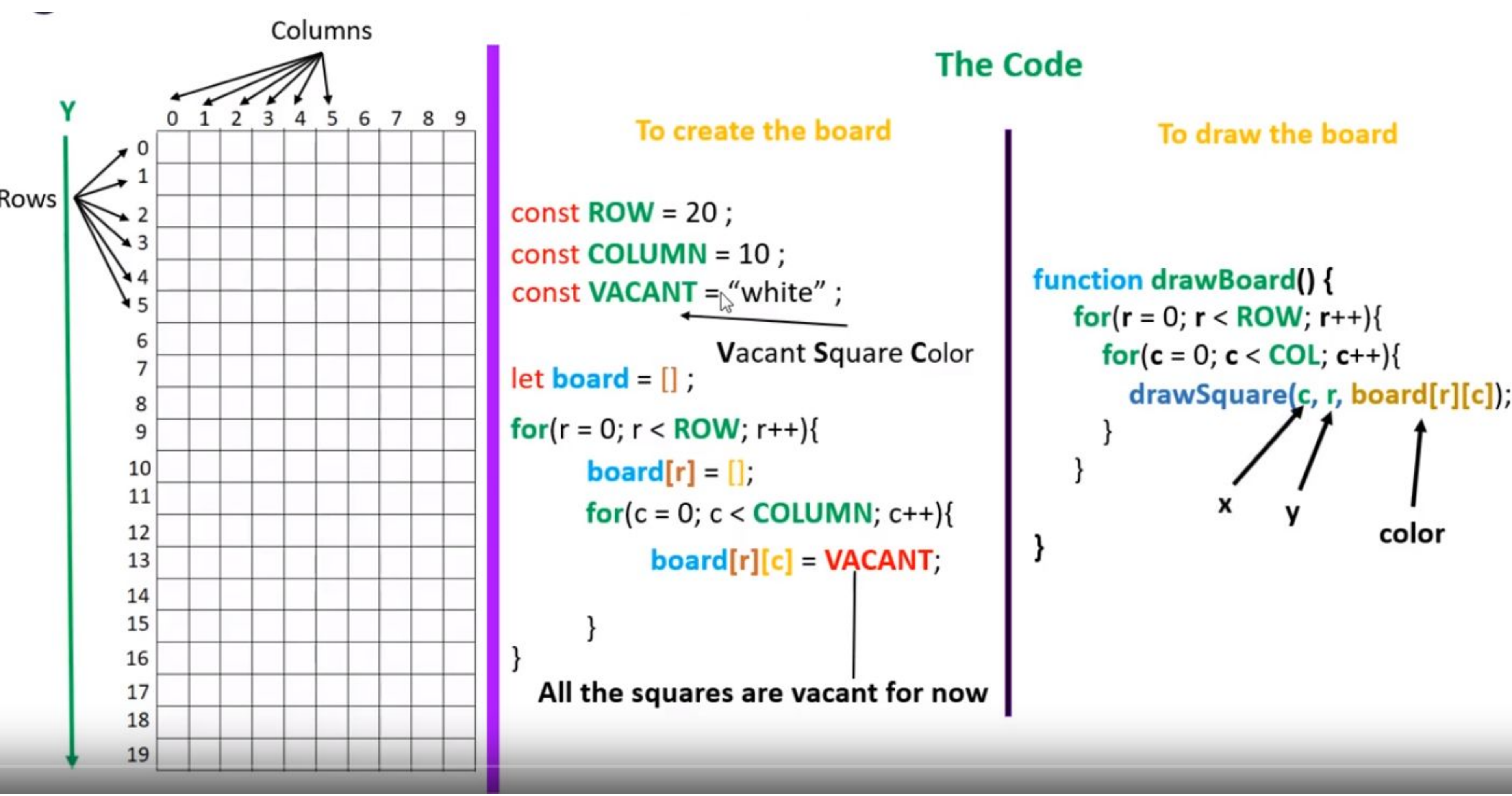
```
function drawSquare( x, y, color ){
  ctx.fillStyle = color;
  ctx.fillRect(x *SQ, y *SQ, SQ, SQ);
  ctx.strokeStyle = “black";
  ctx.strokeRect(x *SQ, y *SQ, SQ, SQ);

}
```

Exercise

```
drawSquare(    ,    ,          )
```

BY NC ND

Dibujar el tablero



The Code

To create the board

```
const ROW = 20 ;
const COLUMN = 10 ;
const VACANT = "white" ;
```

Vacant Square Color

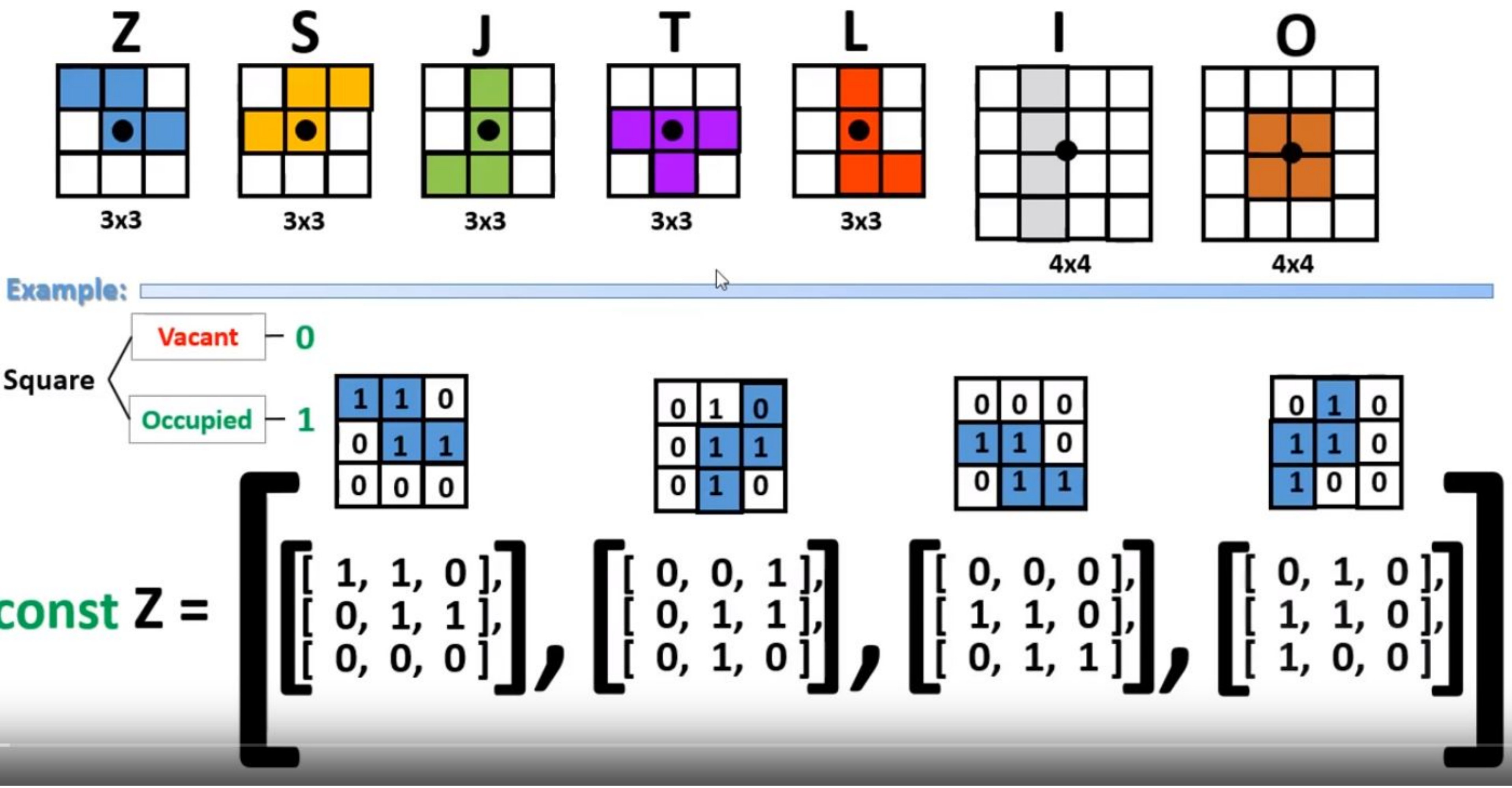
```
let board = [] ;

for(r = 0; r < ROW; r++){
        board[r] = [];
        for(c = 0; c < COLUMN; c++){
                board[r][c] = VACANT;

        }
}
```

All the squares are vacant for now

To draw the board

```
function drawBoard() {
    for(r = 0; r < ROW; r++){
        for(c = 0; c < COL; c++){
            drawSquare(c, r, board[r][c]);
        }
    }
}
```

Piezas de tetris



1	1	0
0	1	1
0	0	0

0	1	0
0	1	1
0	1	0

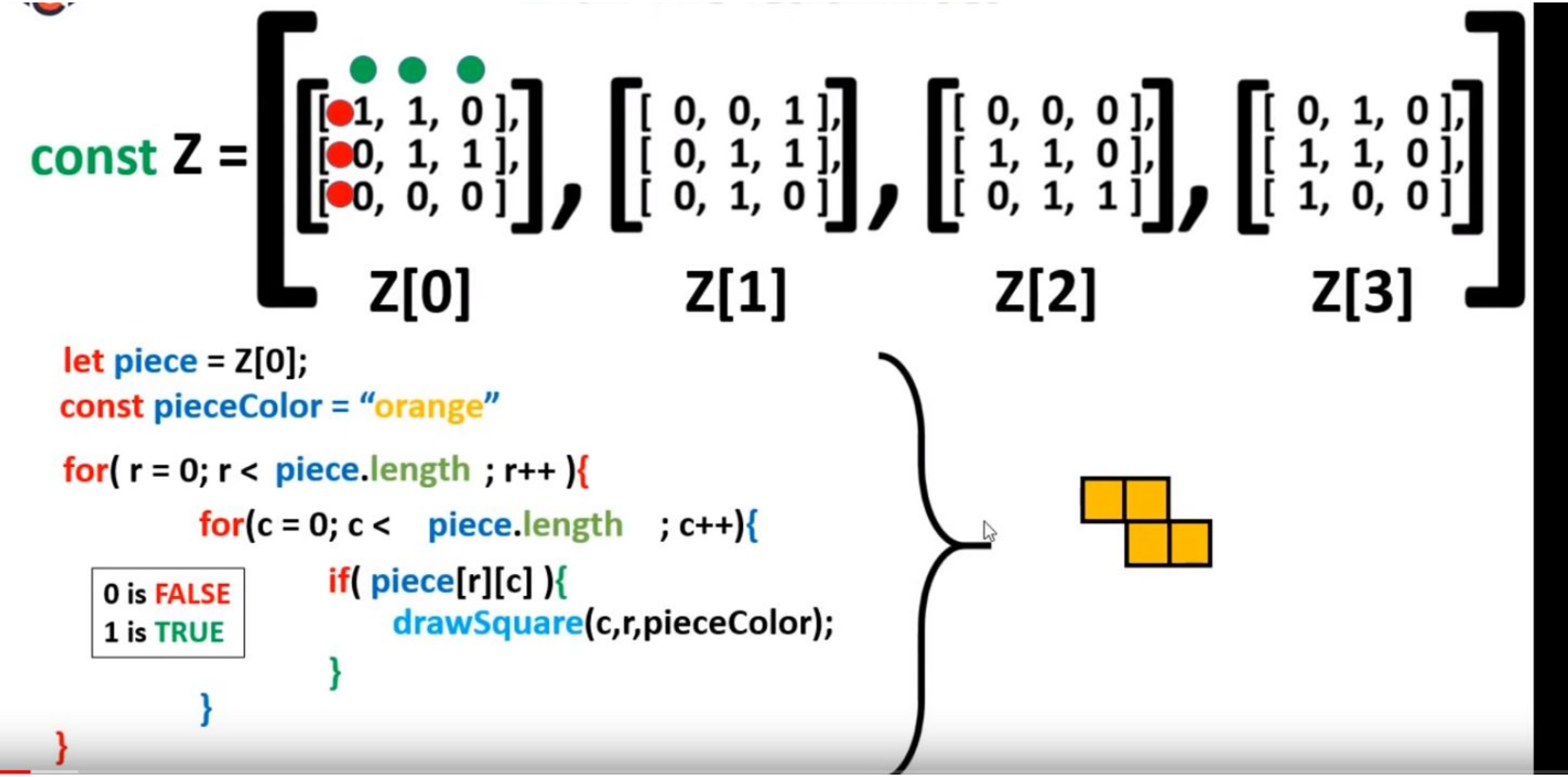
0	0	0
1	1	0
0	1	1

0	1	0
1	1	0
1	0	0

```
const Z = [
  [[ 1, 1, 0 ],
   [ 0, 1, 1 ],
   [ 0, 0, 0 ]],
  [[ 0, 0, 1 ],
   [ 0, 1, 1 ],
   [ 0, 1, 0 ]],
  [[ 0, 0, 0 ],
   [ 1, 1, 0 ],
   [ 0, 1, 1 ]],
  [[ 0, 1, 0 ],
   [ 1, 1, 0 ],
   [ 1, 0, 0 ]]
]
```

cc BY NC ND

Dibujar las piezas



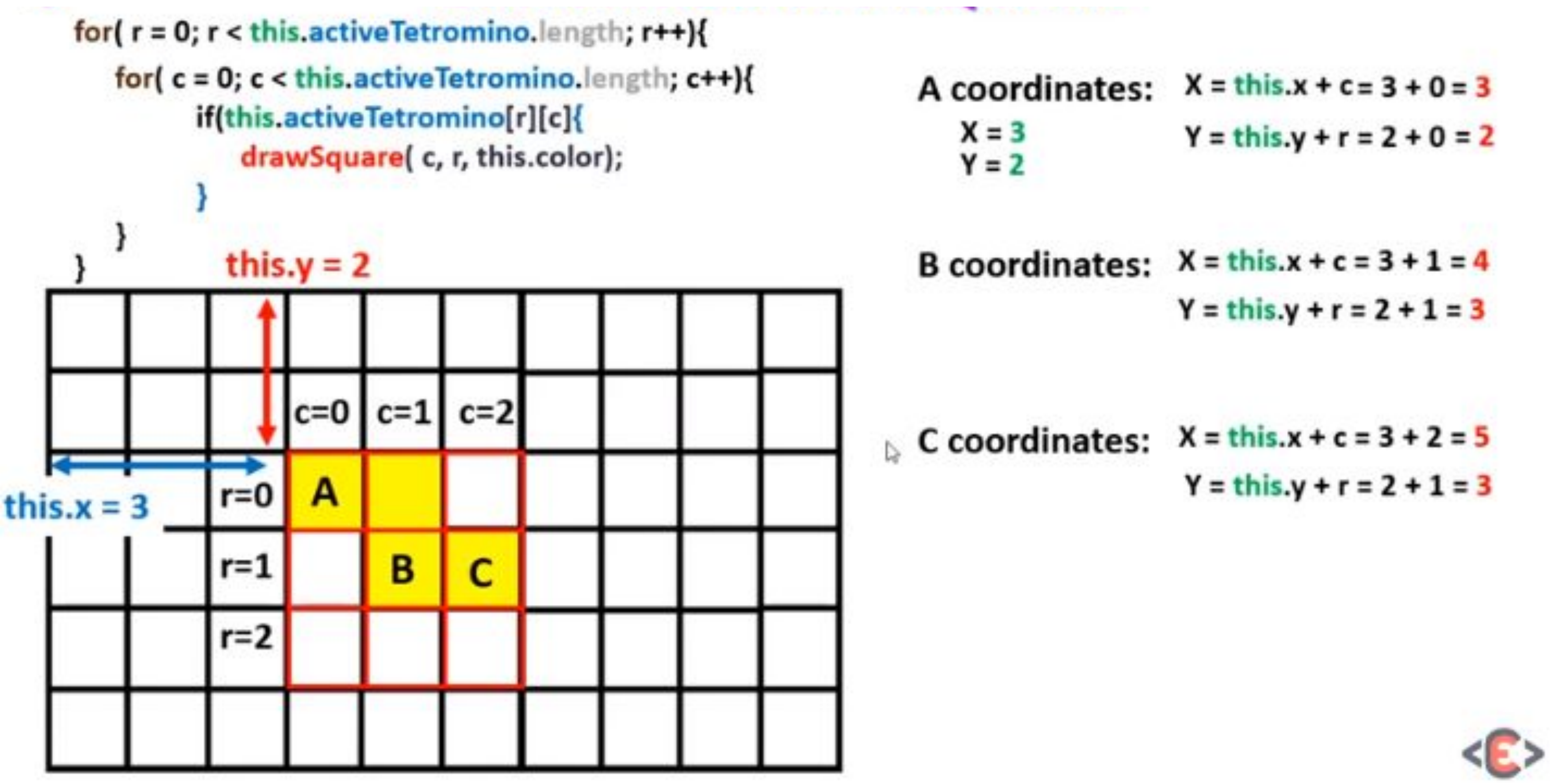
```
const Z = [ [[1, 1, 0 ],
             [0, 1, 1 ],
             [0, 0, 0 ]],  [[ 0, 0, 1 ],
                            [ 0, 1, 1 ],
                            [ 0, 1, 0 ]],  [[ 0, 0, 0 ],
                                            [ 1, 1, 0 ],
                                            [ 0, 1, 1 ]],  [[ 0, 1, 0 ],
                                                            [ 1, 1, 0 ],
                                                            [ 1, 0, 0 ]] ]
              Z[0]            Z[1]            Z[2]            Z[3]
```

```
let piece = Z[0];
const pieceColor = "orange"

for( r = 0; r <  piece.length  ; r++ ){
        for(c = 0; c <   piece.length    ; c++){
                if( piece[r][c] ){
                    drawSquare(c,r,pieceColor);
                }
        }
}
```

0 is FALSE
1 is TRUE

cc BY NC ND

Coordenadas de un cuadrado



```
for( r = 0; r < this.activeTetromino.length; r++){
    for( c = 0; c < this.activeTetromino.length; c++){
            if(this.activeTetromino[r][c]{
                drawSquare( c, r, this.color);
            }
    }
}
```

A coordinates:
X = 3
Y = 2

X = this.x + c = 3 + 0 = 3
Y = this.y + r = 2 + 0 = 2

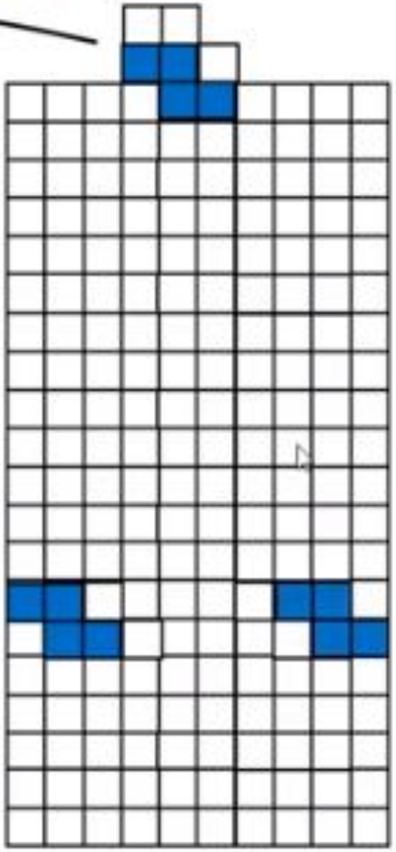
B coordinates:
X = this.x + c = 3 + 1 = 4
Y = this.y + r = 2 + 1 = 3

C coordinates:
X = this.x + c = 3 + 2 = 5
Y = this.y + r = 2 + 1 = 3

BY NC ND

Mover la pieza activa

```
let piece = new Piece( Z , “blue”);

piece.x = 3
piece.y = -2
```



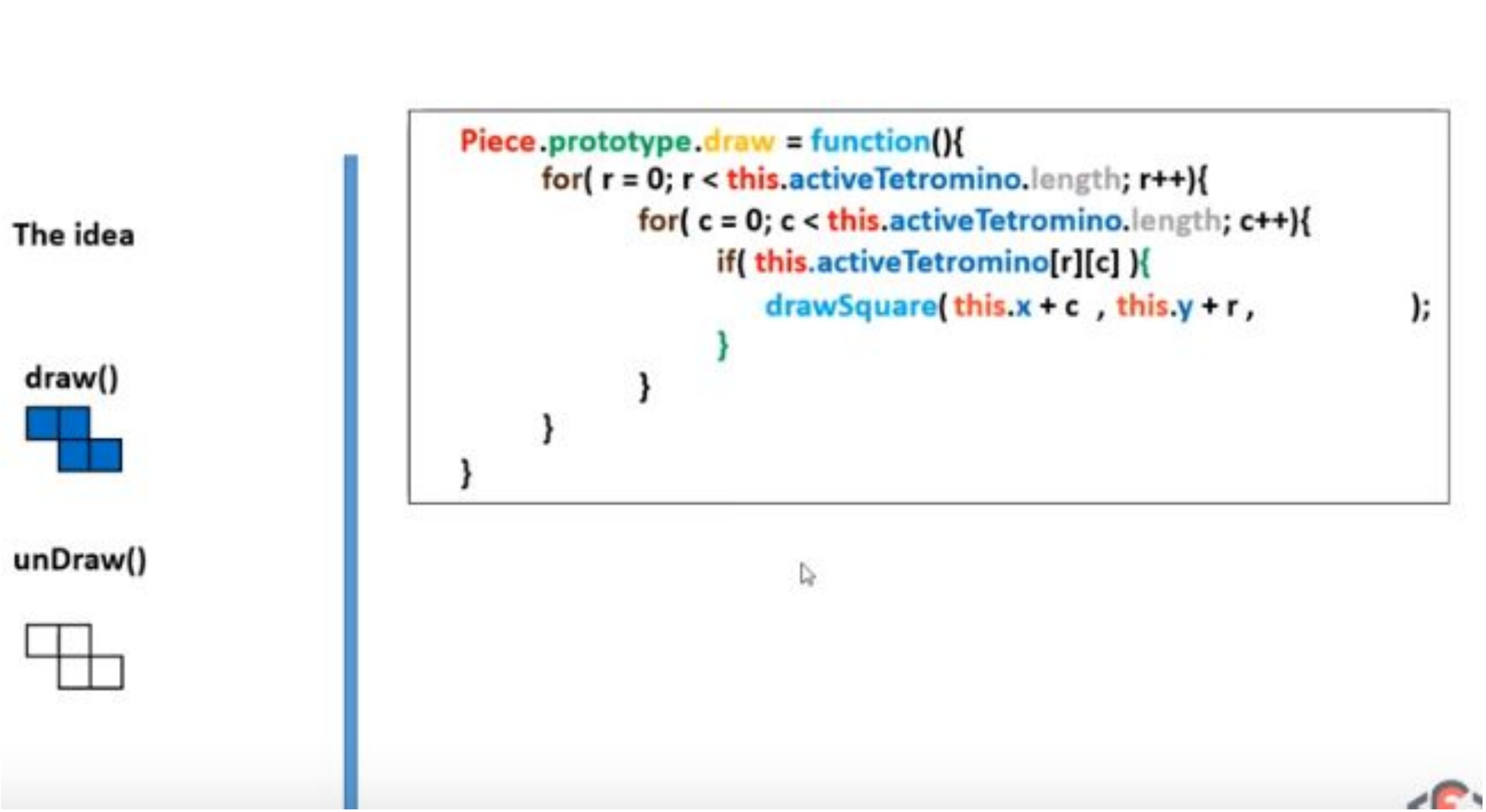
```
piece.moveDown()

    piece.unDraw();
    piece.y++;
    piece.draw();
```

```
piece.moveLeft()

    piece.unDraw();
    piece.x--;
    piece.draw();
```

```
piece.moveRight()

    piece.unDraw();
    piece.x++;
    piece.draw();
```

BY NC ND

Mover la pieza activa



The idea

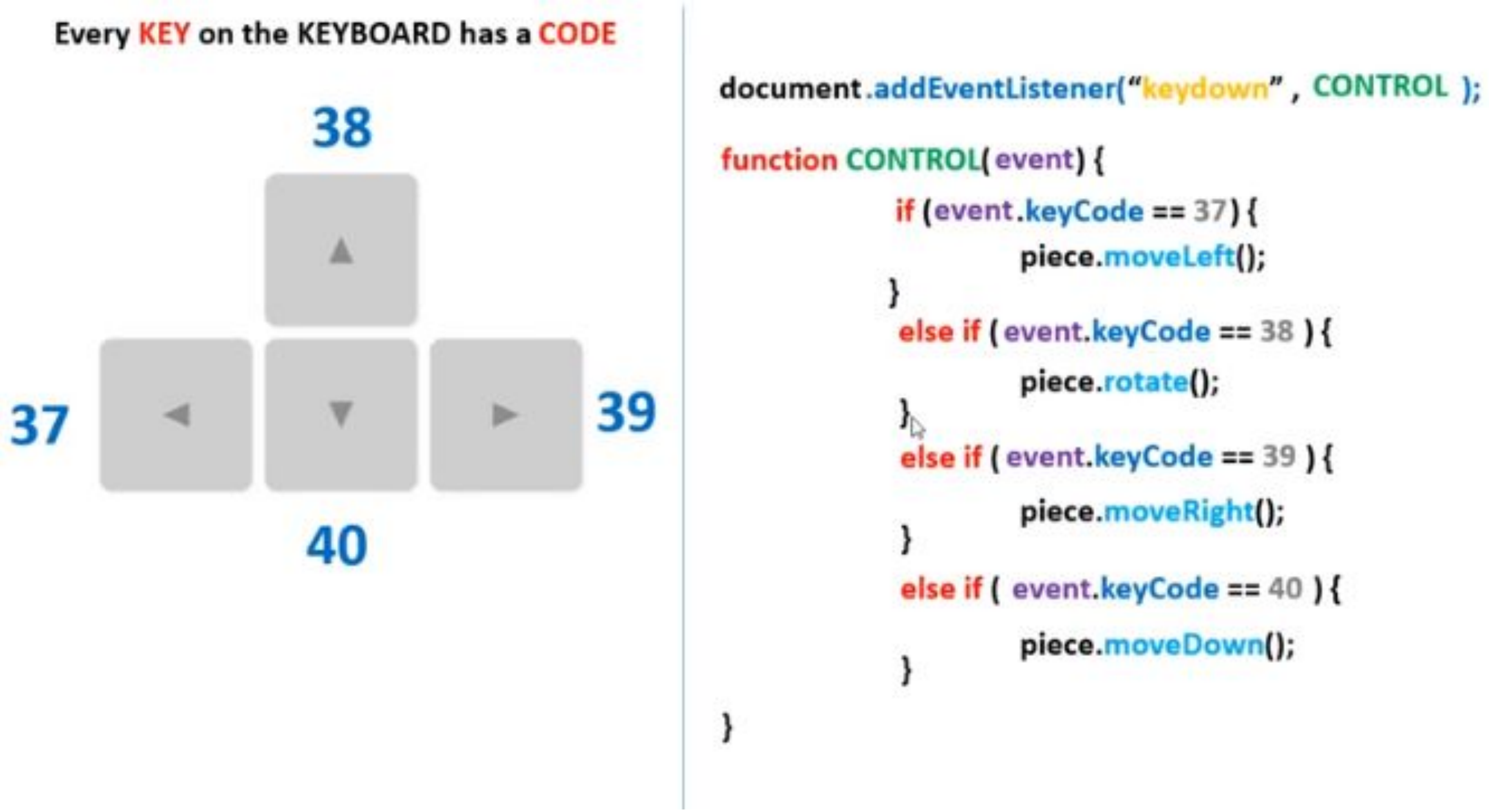
draw()

unDraw()

```
Piece.prototype.draw = function(){
        for( r = 0; r < this.activeTetromino.length; r++){
                for( c = 0; c < this.activeTetromino.length; c++){
                        if( this.activeTetromino[r][c] ){
                                drawSquare( this.x + c  , this.y + r ,              );
                        }
                }
        }
}
```

BY NC ND

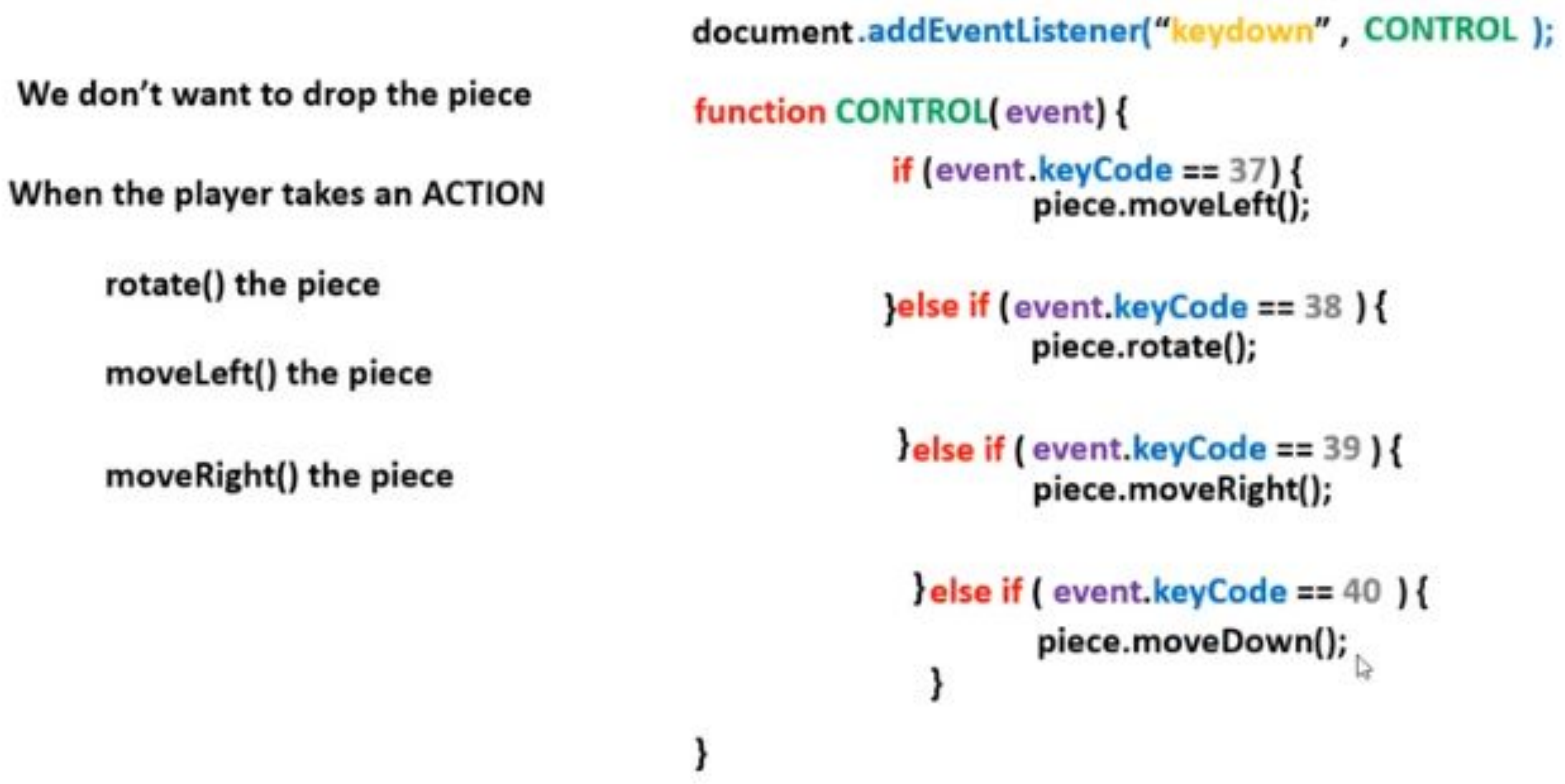
Mover la pieza activa



Every KEY on the KEYBOARD has a CODE

```
document.addEventListener("keydown" , CONTROL );

function CONTROL( event) {
            if (event.keyCode == 37) {
                        piece.moveLeft();
            }
            else if ( event.keyCode == 38 ) {
                        piece.rotate();
            }
            else if ( event.keyCode == 39 ) {
                        piece.moveRight();
            }
            else if ( event.keyCode == 40 ) {
                        piece.moveDown();
            }
}
```

Mover la pieza activa



We don’t want to drop the piece

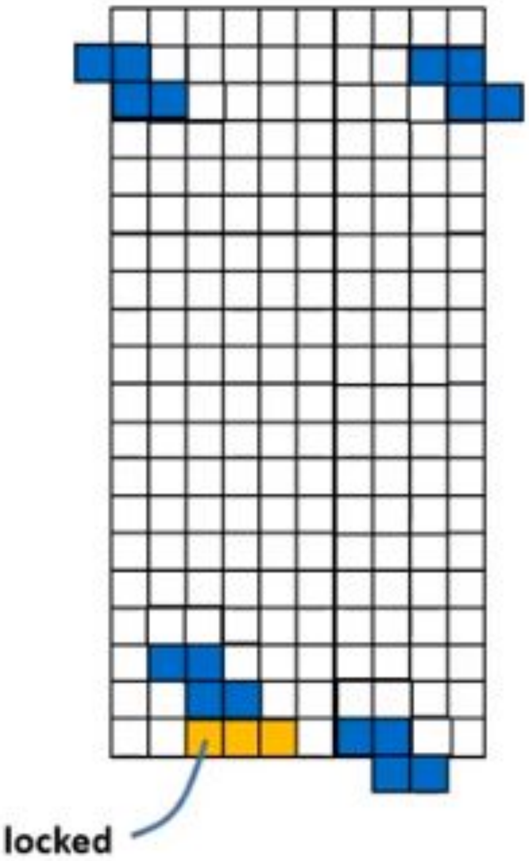
When the player takes an ACTION

- rotate() the piece
- moveLeft() the piece
- moveRight() the piece

```
document.addEventListener(“keydown” , CONTROL );

function CONTROL( event) {
            if (event.keyCode == 37) {
                    piece.moveLeft();

            }else if (event.keyCode == 38 ) {
                    piece.rotate();

             }else if ( event.keyCode == 39 ) {
                    piece.moveRight();

              }else if ( event.keyCode == 40  ) {
                    piece.moveDown();
               }

}
```

BY NC ND

colisión



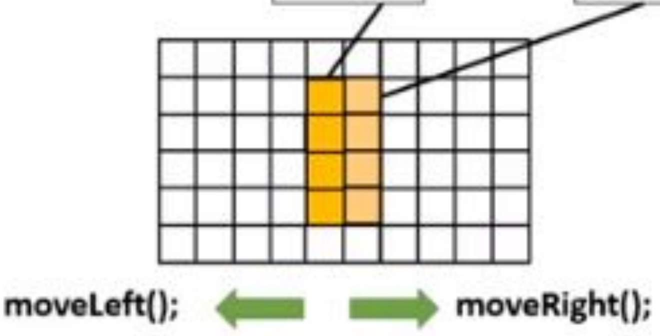
BEFORE any movement of a piece — RIGHT, LEFT, DOWN, ROTATION

We HAVE to check if that movement won't lead to a collision

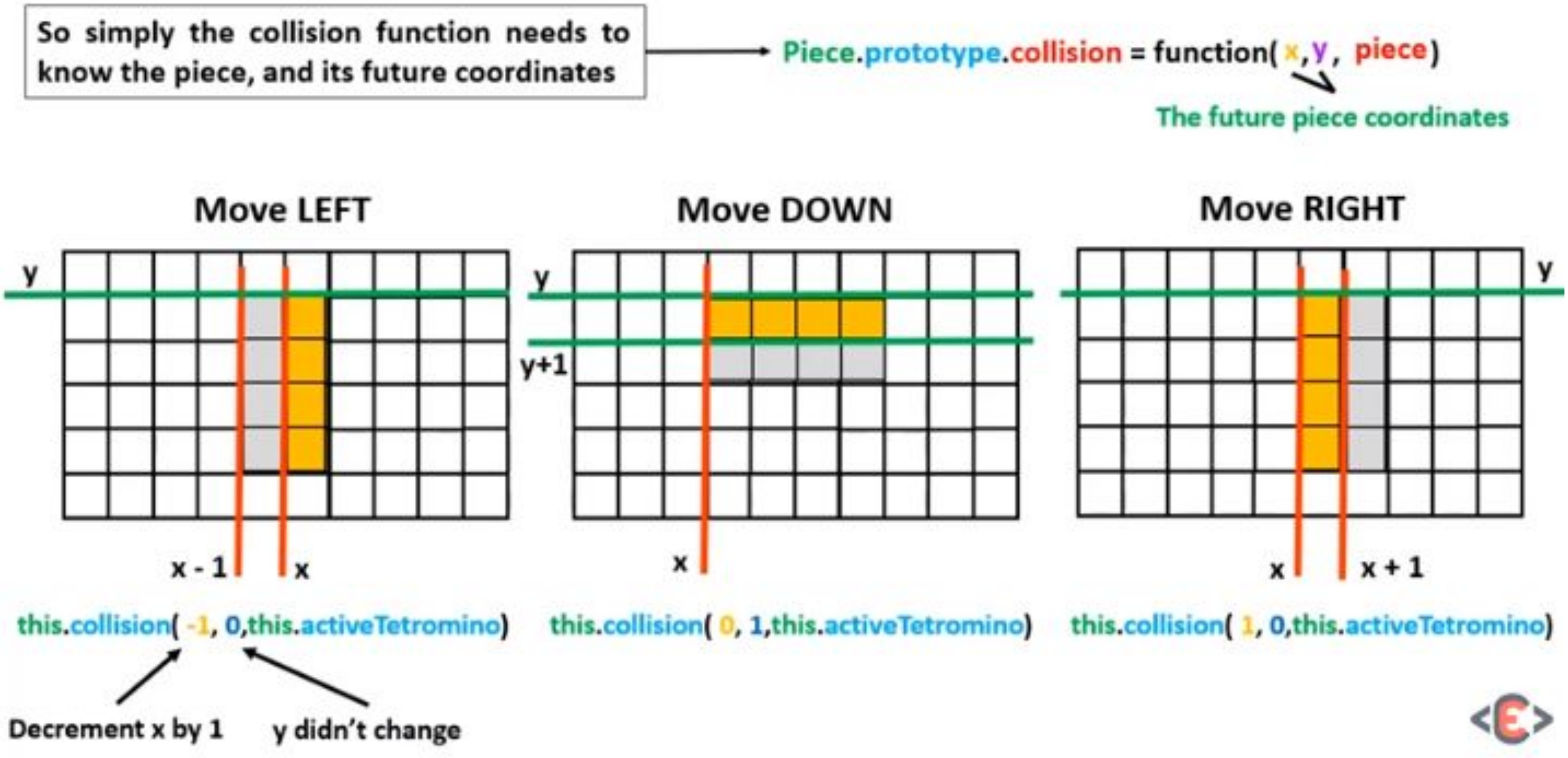
CHECK if there will be a collision
- FALSE — Do the mouvement
- TRUE — Don't do the mouvement

Example:

We don't Check this piece we check this one.



colisión



So simply the collision function needs to know the piece, and its future coordinates

```
Piece.prototype.collision = function( x,y, piece)
```

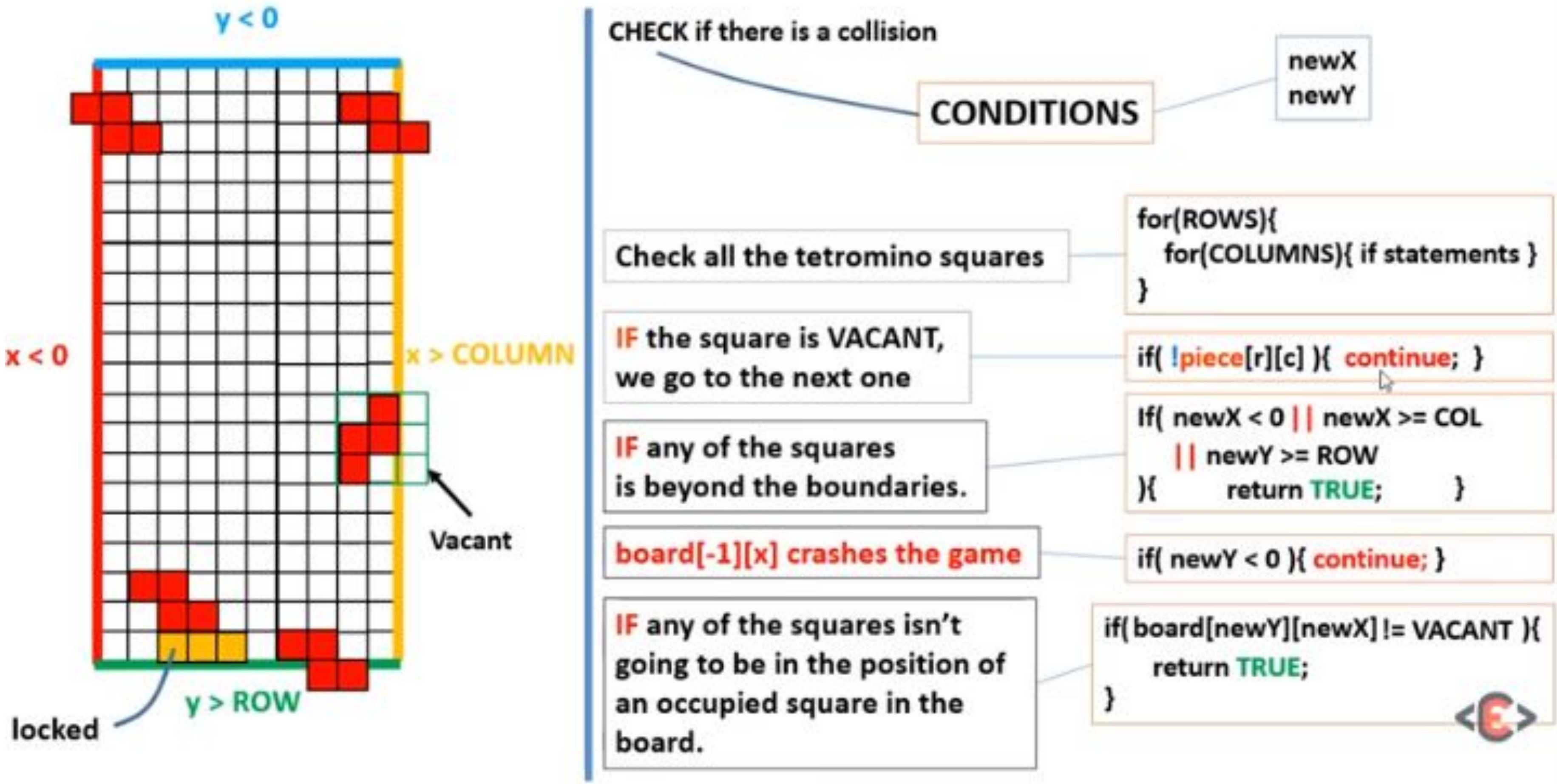
The future piece coordinates

```
this.collision( -1, 0,this.activeTetromino)
this.collision( 0, 1,this.activeTetromino)
this.collision( 1, 0,this.activeTetromino)
```

Decrement x by 1

y didn't change

colisión



CHECK if there is a collision

CONDITIONS

newX
newY

Check all the tetromino squares

```
for(ROWS){
   for(COLUMNS){ if statements }
}
```

IF the square is VACANT, we go to the next one

```
if( !piece[r][c] ){  continue;  }
```

IF any of the squares is beyond the boundaries.

```
If( newX < 0 || newX >= COL
     || newY >= ROW
){        return TRUE;        }
```

board[-1][x] crashes the game

```
if( newY < 0 ){ continue; }
```

IF any of the squares isn't going to be in the position of an occupied square in the board.

```
if( board[newY][newX] != VACANT ){
     return TRUE;
}
```

BY NC ND

colisión

colision (x,y,piece)

```
// para cada casilla de la tetronimio activo

for(var f = 0; f < piece.length; f++){
            for(var c = 0; c < piece.length; c++){
                // si la casilla está vacía la obviamos
                if(!piece[f][c]){
                    continue;
                }
                // nuevas coordenadas de la casilla
                // despúes del movimiento

                let nuevaX = this.x + c + x;
                let nuevaY = this.y + f + y;
```

BY NC ND

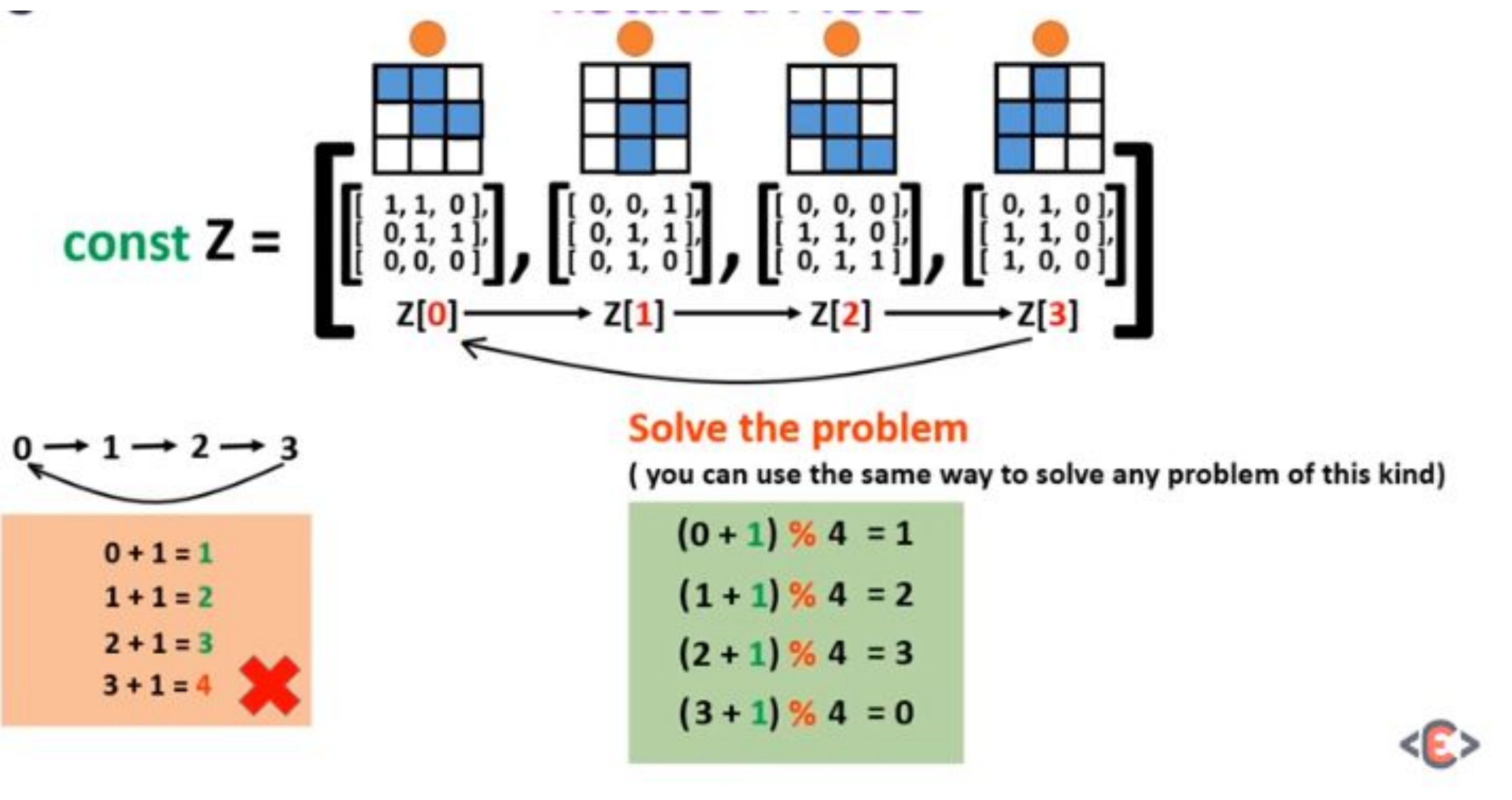
colisión

colision (x,y,piece)

```
 // condiciones
    if(nuevaX < 0 ||
       nuevaX >= this.tablero._columnas ||
       nuevaY >= this.tablero._filas){
                     return true; // sale del tablero
     }
     if(nuevaY < 0){ // para evitar acceder a tablero[-1]
         continue;
     }
     if( !this.tablero.esVacio(nuevaY,nuevaX)){
           return true;
     }
    }
   }
return false;
```

BY NC ND

giro de piezas



```
const Z = [
  [[ 1, 1, 0 ],
   [ 0, 1, 1 ],
   [ 0, 0, 0 ]],
  [[ 0, 0, 1 ],
   [ 0, 1, 1 ],
   [ 0, 1, 0 ]],
  [[ 0, 0, 0 ],
   [ 1, 1, 0 ],
   [ 0, 1, 1 ]],
  [[ 0, 1, 0 ],
   [ 1, 1, 0 ],
   [ 1, 0, 0 ]]
]
```

$$\begin{tikzcd}
Z[0] \arrow[r] & Z[1] \arrow[r] & Z[2] \arrow[r] & Z[3] \arrow[lll, bend left]
\end{tikzcd}$$

$$\begin{tikzcd}
0 \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[lll, bend left]
\end{tikzcd}$$

$0 + 1 = 1$

$1 + 1 = 2$

$2 + 1 = 3$

$3 + 1 = 4$ ✖

Solve the problem

(you can use the same way to solve any problem of this kind)

$(0 + 1)\ \%\ 4 = 1$

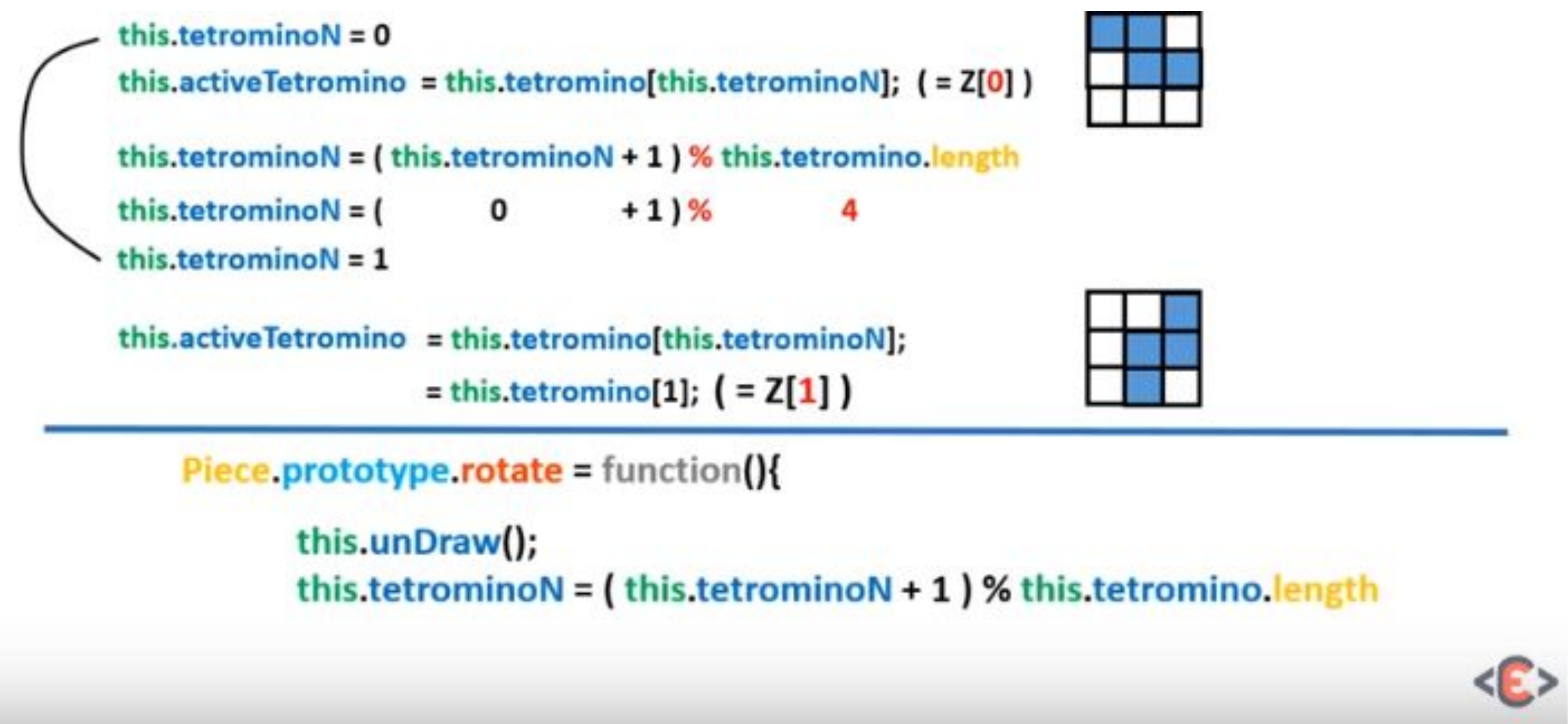
$(1 + 1)\ \%\ 4 = 2$

$(2 + 1)\ \%\ 4 = 3$

$(3 + 1)\ \%\ 4 = 0$

cc BY NC ND

giro de piezas



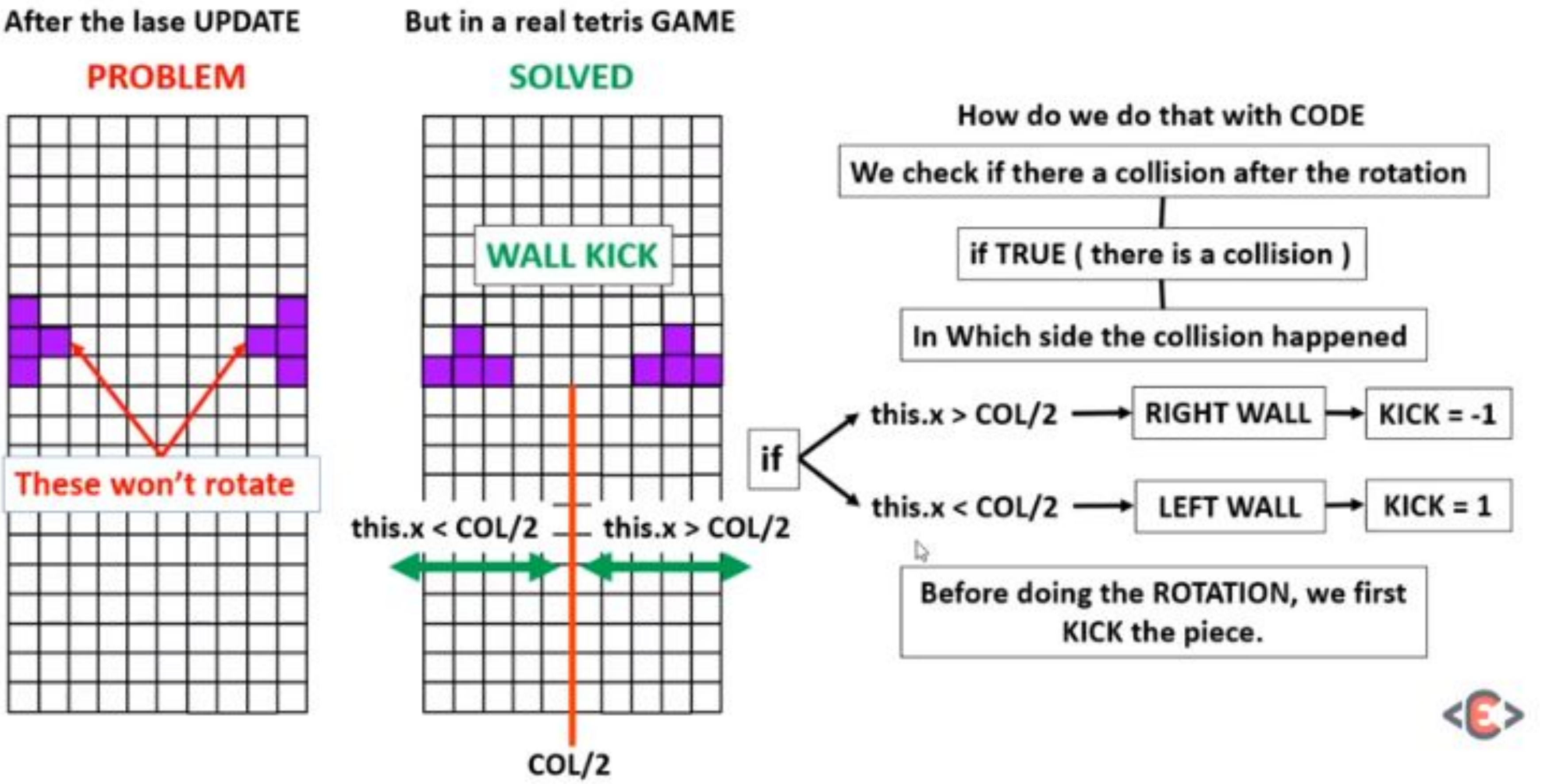
```
this.tetrominoN = 0
this.activeTetromino  = this.tetromino[this.tetrominoN];  ( = Z[0] )

this.tetrominoN = ( this.tetrominoN + 1 ) % this.tetromino.length
this.tetrominoN = (          0          + 1 ) %          4
this.tetrominoN = 1

this.activeTetromino  = this.tetromino[this.tetrominoN];
                      = this.tetromino[1];  ( = Z[1] )
```

■	■	□
□	■	■
□	□	□

□	□	■
□	■	■
□	■	□

```
Piece.prototype.rotate = function(){

        this.unDraw();
        this.tetrominoN = ( this.tetrominoN + 1 ) % this.tetromino.length
```

giro de piezas



After the lase UPDATE

PROBLEM

These won't rotate

But in a real tetris GAME

SOLVED

WALL KICK

this.x < COL/2

this.x > COL/2

COL/2

How do we do that with CODE

We check if there a collision after the rotation

if TRUE (there is a collision)

In Which side the collision happened

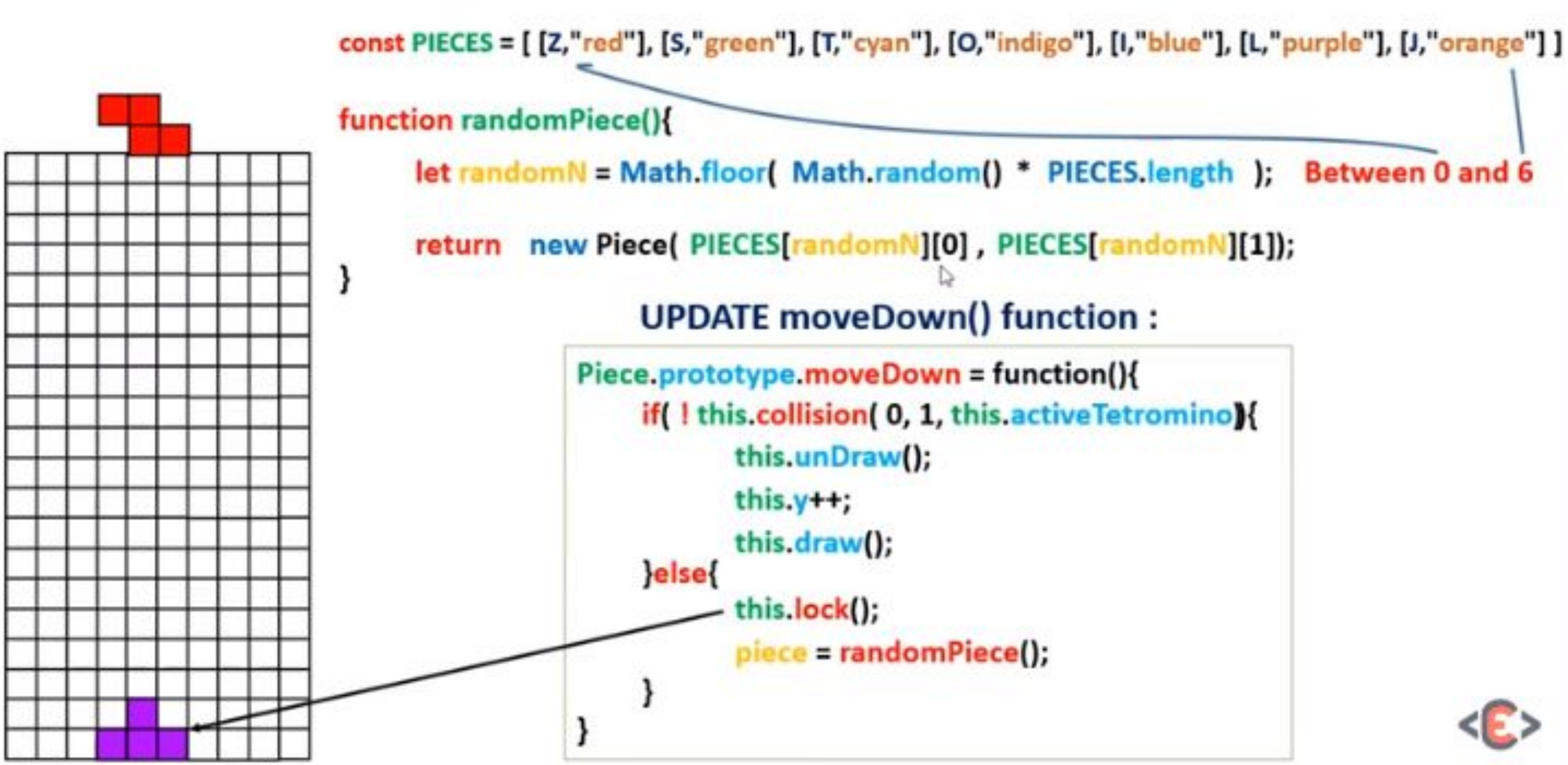
if

this.x > COL/2 → RIGHT WALL → KICK = -1

this.x < COL/2 → LEFT WALL → KICK = 1

Before doing the ROTATION, we first KICK the piece.

pieza aleatoria



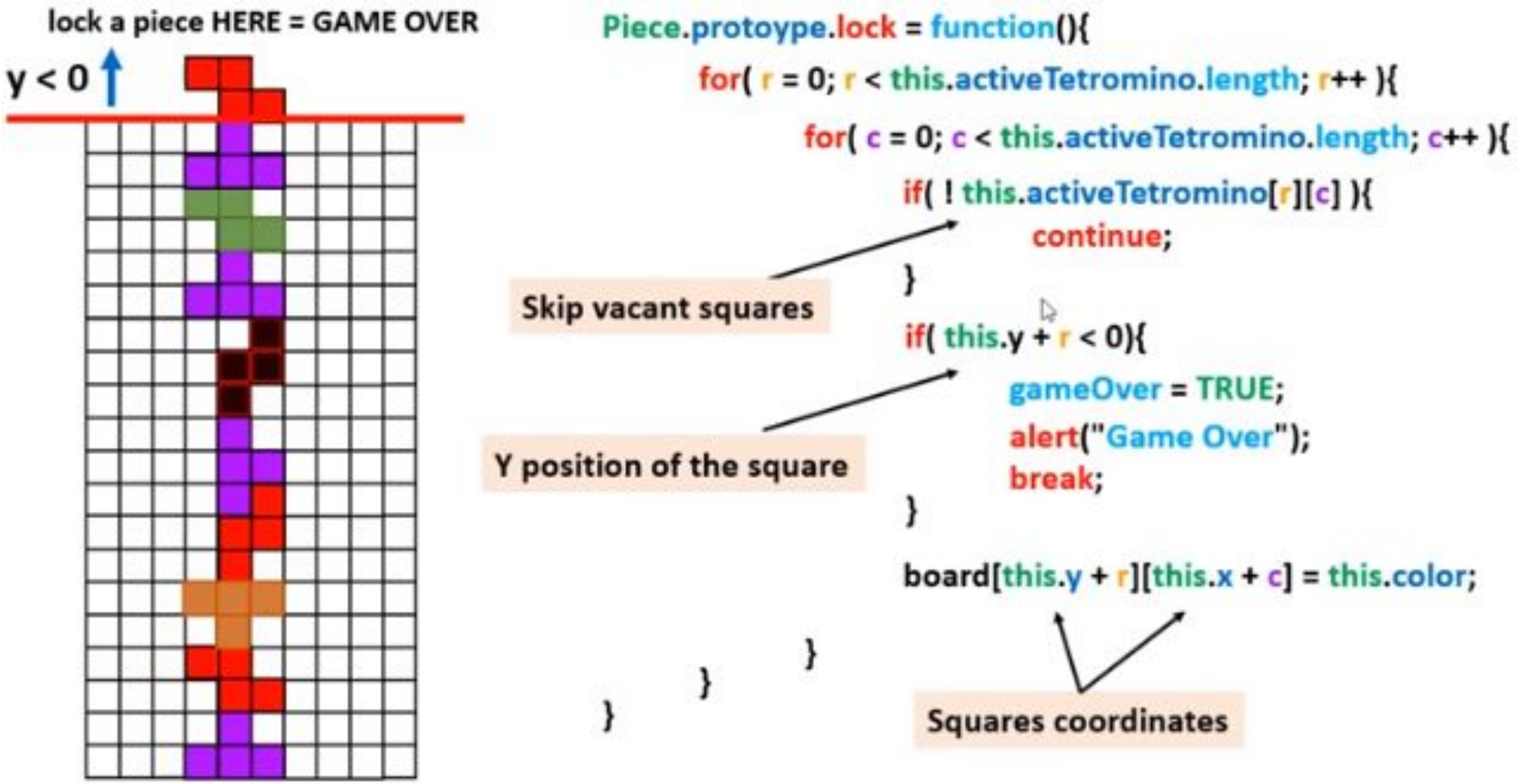
```
const PIECES = [ [Z,"red"], [S,"green"], [T,"cyan"], [O,"indigo"], [I,"blue"], [L,"purple"], [J,"orange"] ]

function randomPiece(){
        let randomN = Math.floor( Math.random() * PIECES.length );

        return   new Piece( PIECES[randomN][0] , PIECES[randomN][1]);
}
```

Between 0 and 6

UPDATE moveDown() function :

```
Piece.prototype.moveDown = function(){
        if( ! this.collision( 0, 1, this.activeTetromino)){
                this.unDraw();
                this.y++;
                this.draw();
        }else{
                this.lock();
                piece = randomPiece();
        }
}
```

Game Over



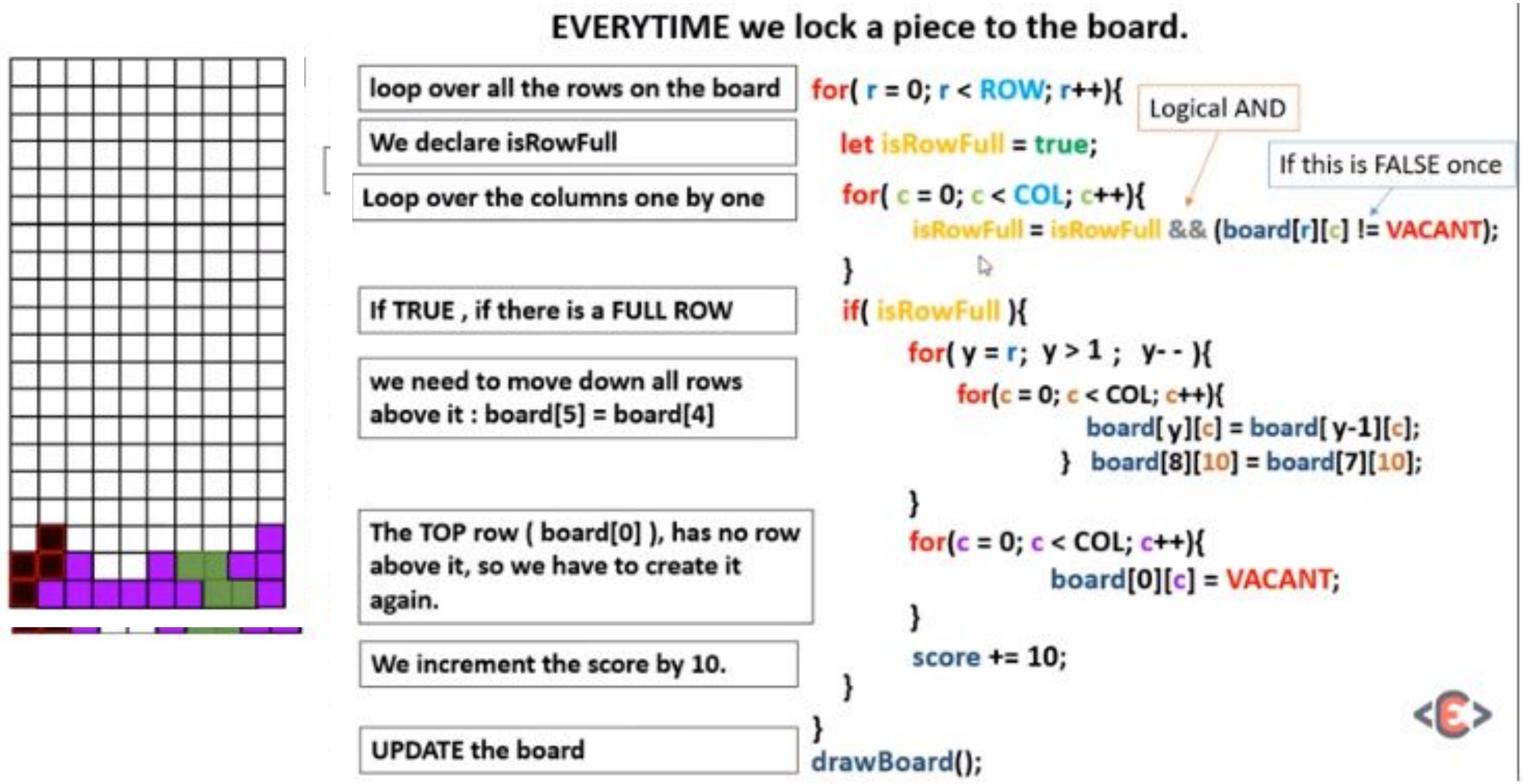
```
Piece.protoype.lock = function(){
    for( r = 0; r < this.activeTetromino.length; r++ ){
        for( c = 0; c < this.activeTetromino.length; c++ ){
            if( ! this.activeTetromino[r][c] ){
                continue;
            }
            if( this.y + r < 0){
                gameOver = TRUE;
                alert("Game Over");
                break;
            }
            board[this.y + r][this.x + c] = this.color;
        }
    }
}
```

Skip vacant squares

Y position of the square

Squares coordinates

Eliminar fila



EVERYTIME we lock a piece to the board.

loop over all the rows on the board

We declare isRowFull

Loop over the columns one by one

If TRUE , if there is a FULL ROW

we need to move down all rows above it : board[5] = board[4]

The TOP row (board[0]), has no row above it, so we have to create it again.

We increment the score by 10.

UPDATE the board

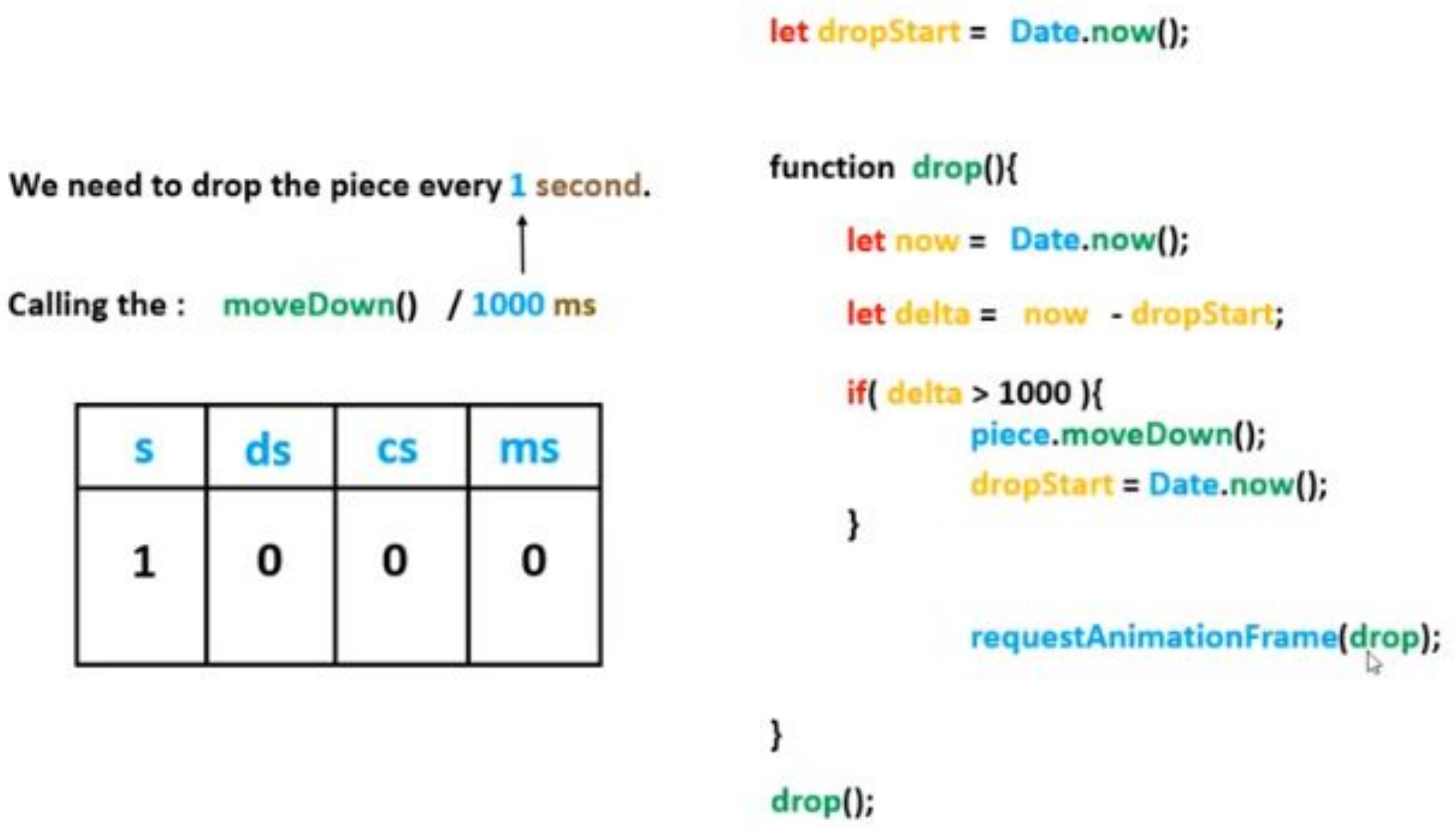
Logical AND

If this is FALSE once

```
for( r = 0; r < ROW; r++){
    let isRowFull = true;
    for( c = 0; c < COL; c++){
        isRowFull = isRowFull && (board[r][c] != VACANT);
    }
    if( isRowFull ){
        for( y = r;  y > 1 ;  y- - ){
            for(c = 0; c < COL; c++){
                board[ y][c] = board[ y-1][c];
            }  board[8][10] = board[7][10];
        }
        for(c = 0; c < COL; c++){
            board[0][c] = VACANT;
        }
        score += 10;
    }
}
drawBoard();
```

cc BY NC ND

Función de caída (update delta)



We need to drop the piece every 1 second.

Calling the : moveDown() / 1000 ms

s	ds	cs	ms
1	0	0	0

```
let dropStart =  Date.now();


function  drop(){

      let now =  Date.now();

      let delta =   now  - dropStart;

      if( delta > 1000 ){
               piece.moveDown();
               dropStart = Date.now();
      }


               requestAnimationFrame(drop);

}
drop();
```

cc BY NC ND

BY NC ND