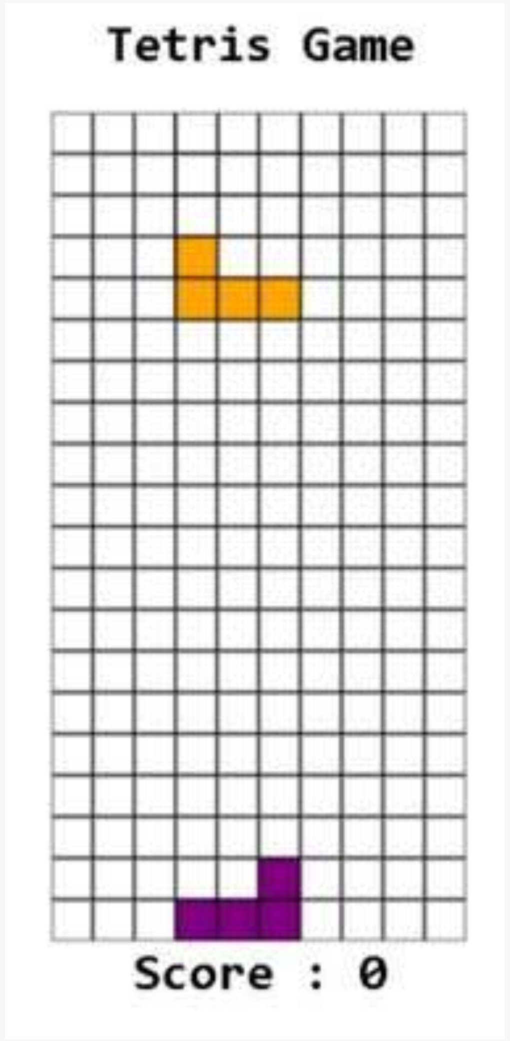
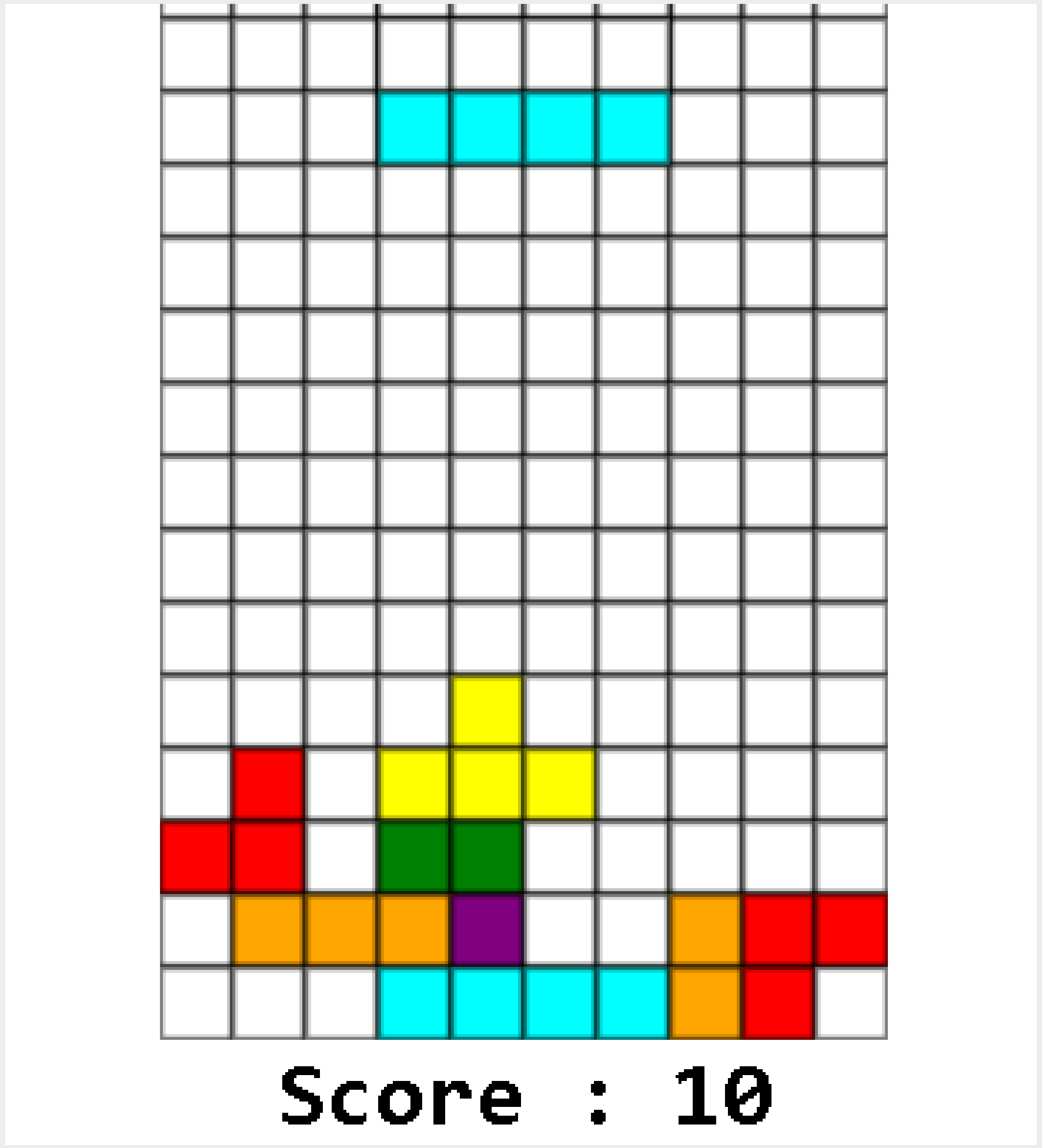
# Tetris Game JS



*Under Supervision: Doc Merihan*

*Made by :Raheem Amer*

## Contents:



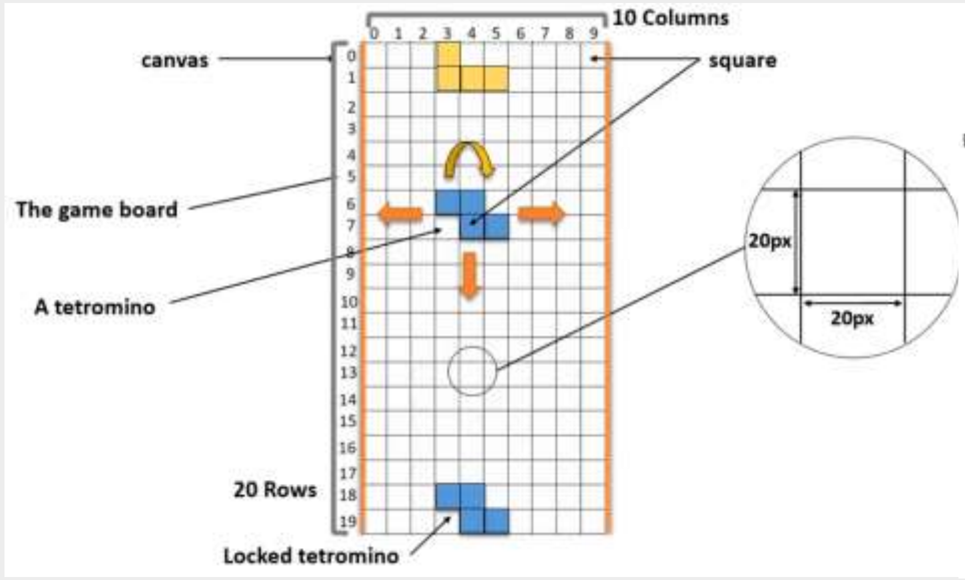
## Overview:

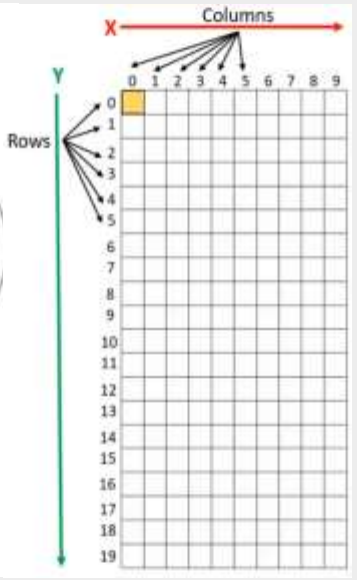
***How the game works?***

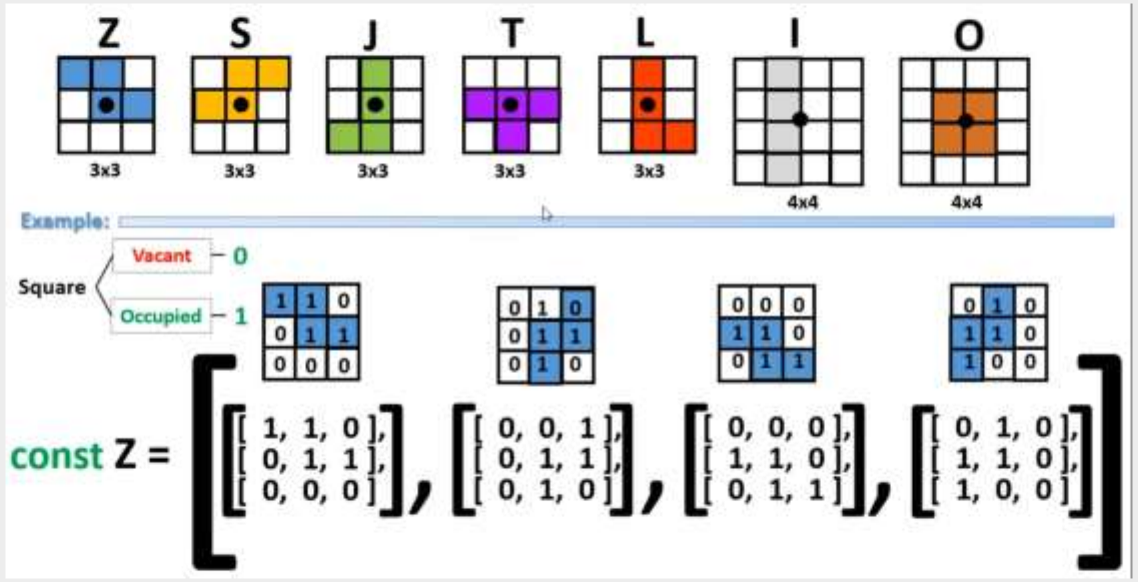
**- A tetromino piece drops and you have to align the tetromino horizontally to gain a score + 10**

**- If you didn't make a horizontal line the score will still the same but the pieces will be pile up until they reach the upper borders and the game will alert the user it's game over with his score**

## Layout:

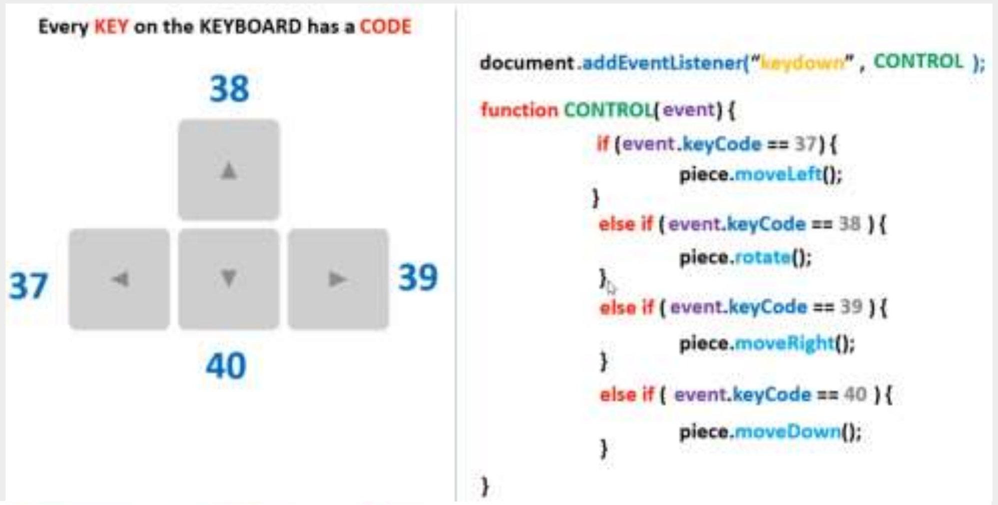




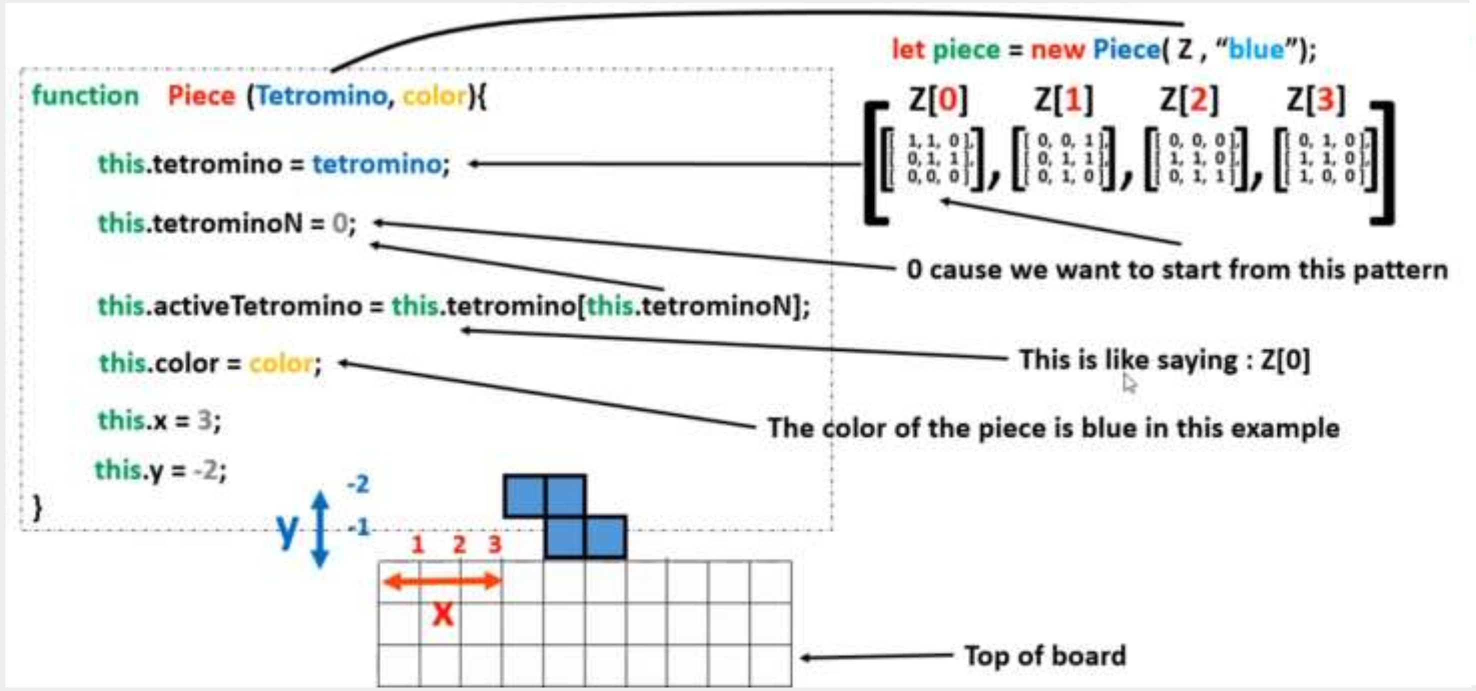


```
const Z = [
[[ 1, 1, 0 ],
 [ 0, 1, 1 ],
 [ 0, 0, 0 ]],
[[ 0, 0, 1 ],
 [ 0, 1, 1 ],
 [ 0, 1, 0 ]],
[[ 0, 0, 0 ],
 [ 1, 1, 0 ],
 [ 0, 1, 1 ]],
[[ 0, 1, 0 ],
 [ 1, 1, 0 ],
 [ 1, 0, 0 ]]
]
```

# Pieces:



```
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();
        }
}
```



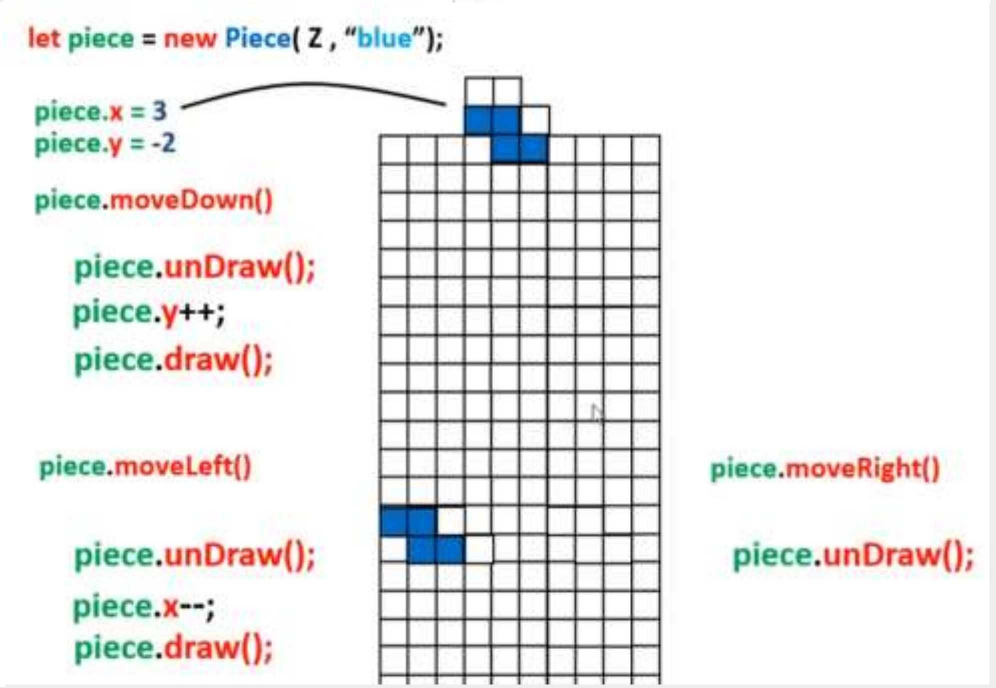
```
let piece = new Piece( Z , "blue");

function   Piece (Tetromino, color){

    this.tetromino = tetromino;

    this.tetrominoN = 0;

    this.activeTetromino = this.tetromino[this.tetrominoN];

    this.color = color;

    this.x = 3;

    this.y = -2;
}
```

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

0 cause we want to start from this pattern

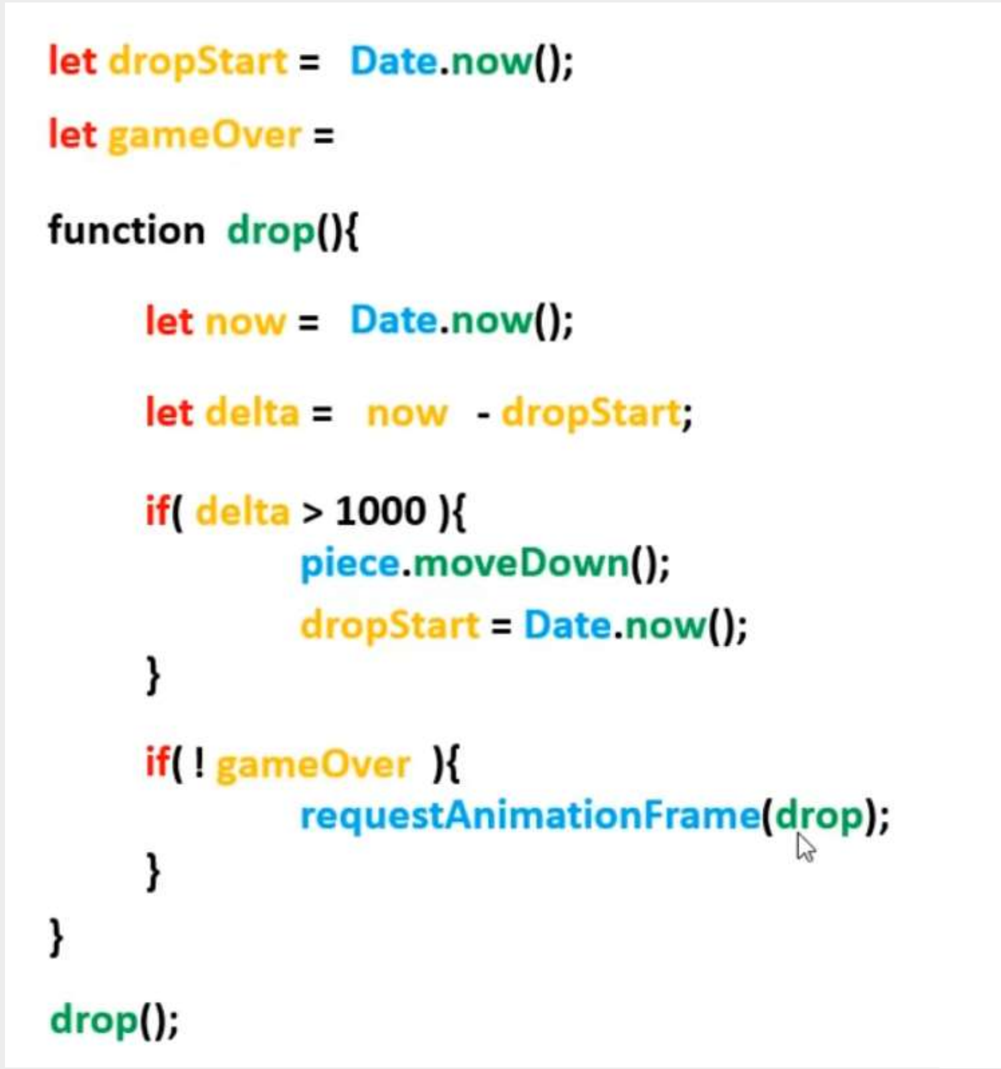
This is like saying : Z[0]

The color of the piece is blue in this example

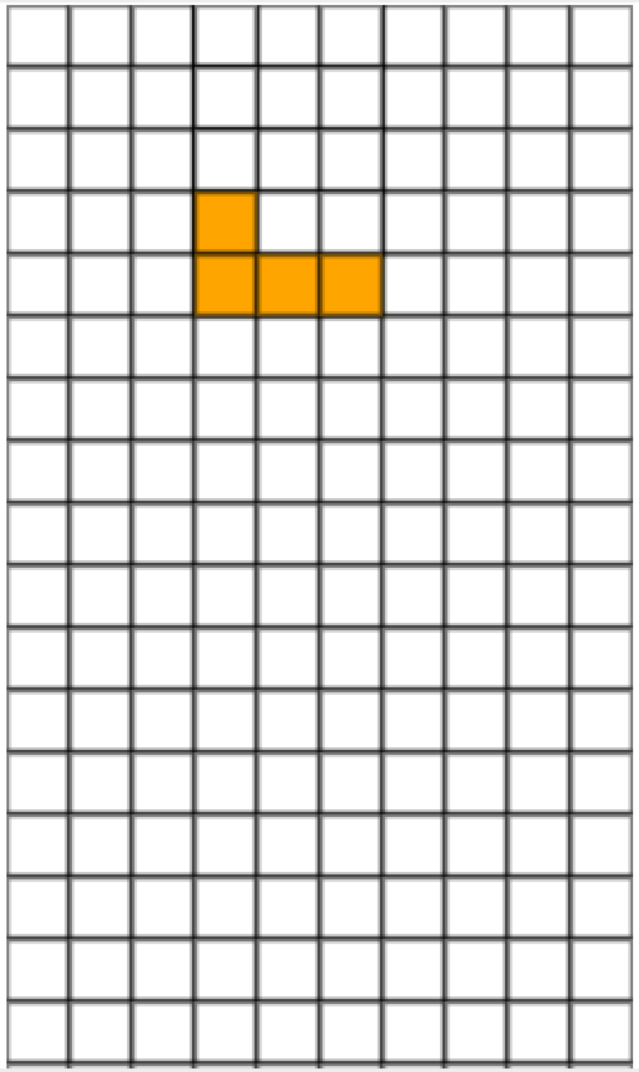


```
let piece = new Piece( Z , "blue");

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();
```

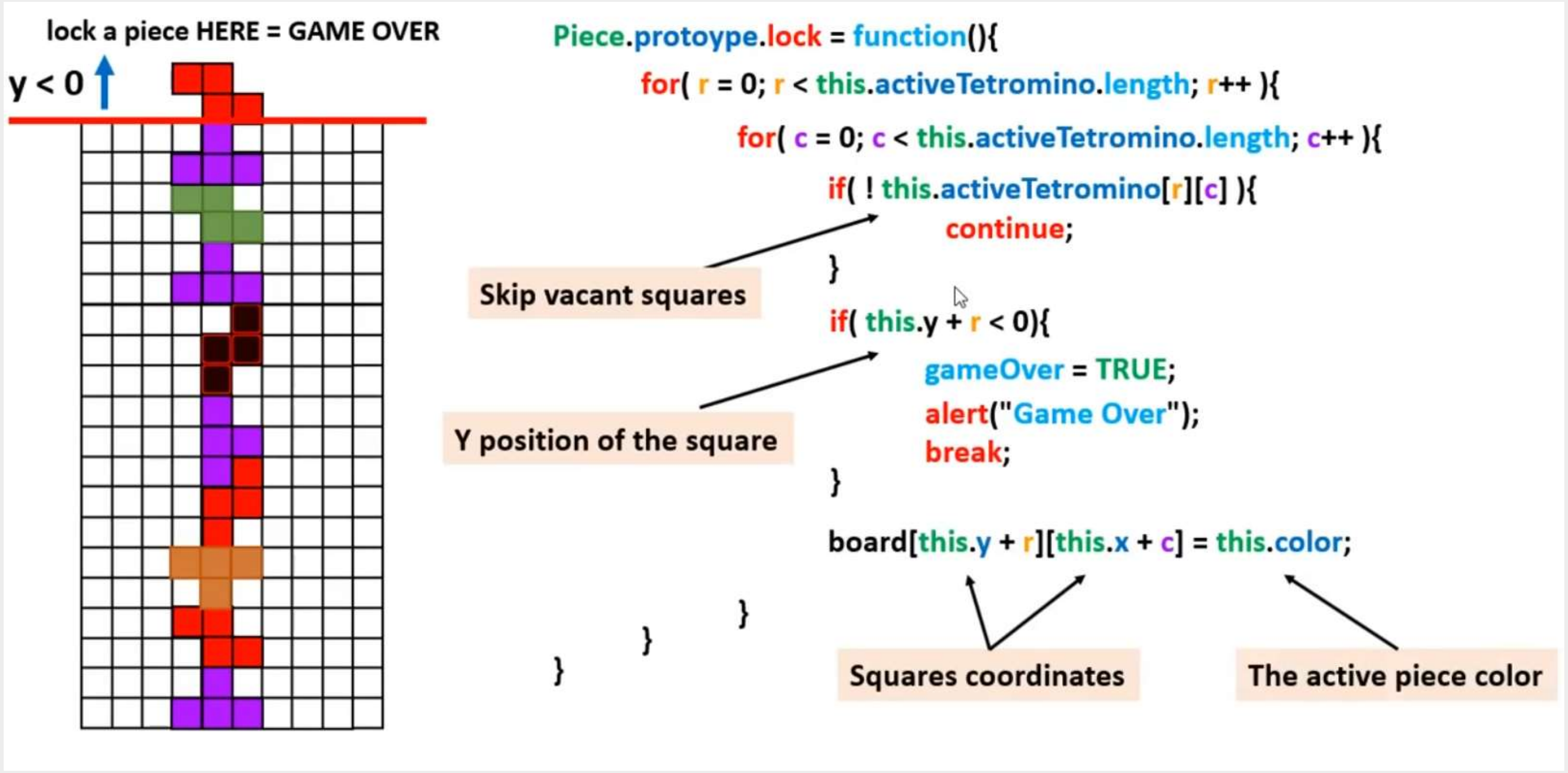
## Drop a piece:



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

let gameOver =

function  drop(){

        let now =  Date.now();

        let delta =  now  - dropStart;

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

        if( ! gameOver  ){
                requestAnimationFrame(drop);
        }
}

drop();
```

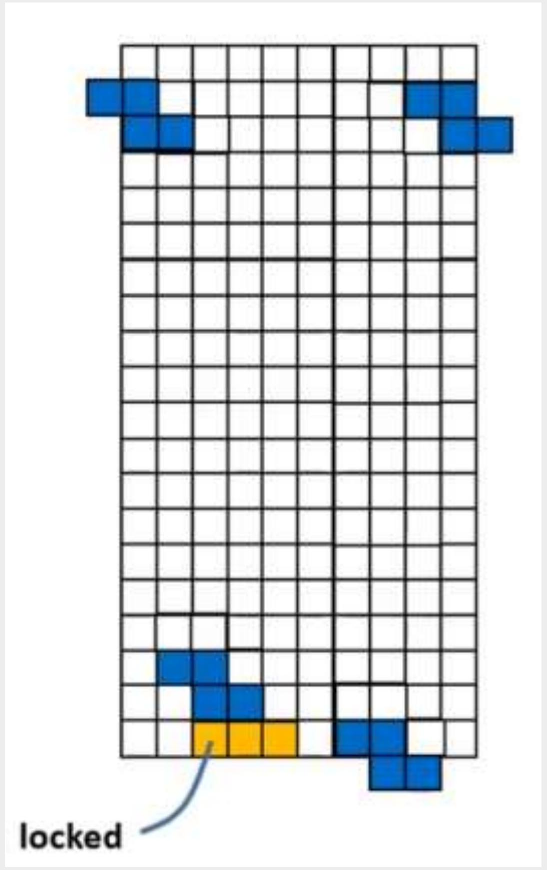


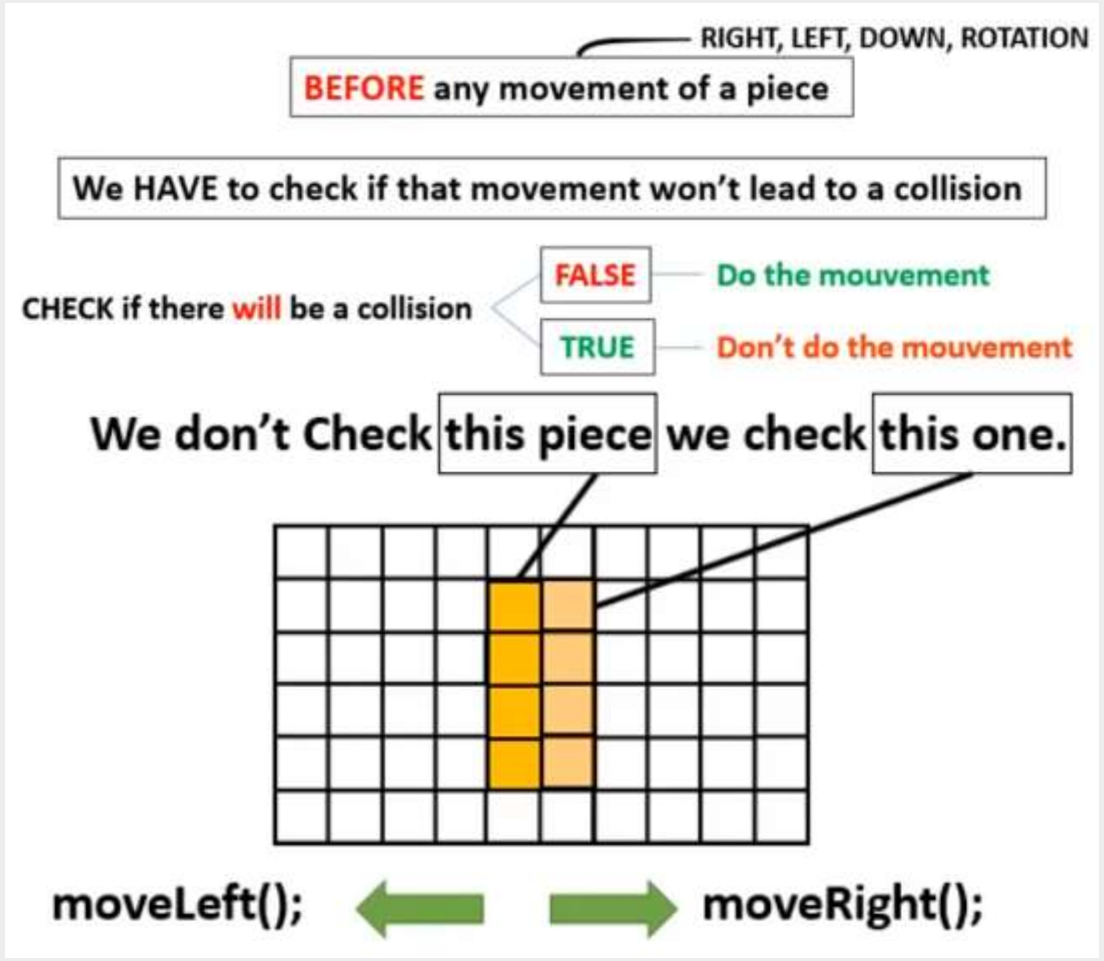
## lock a piece:



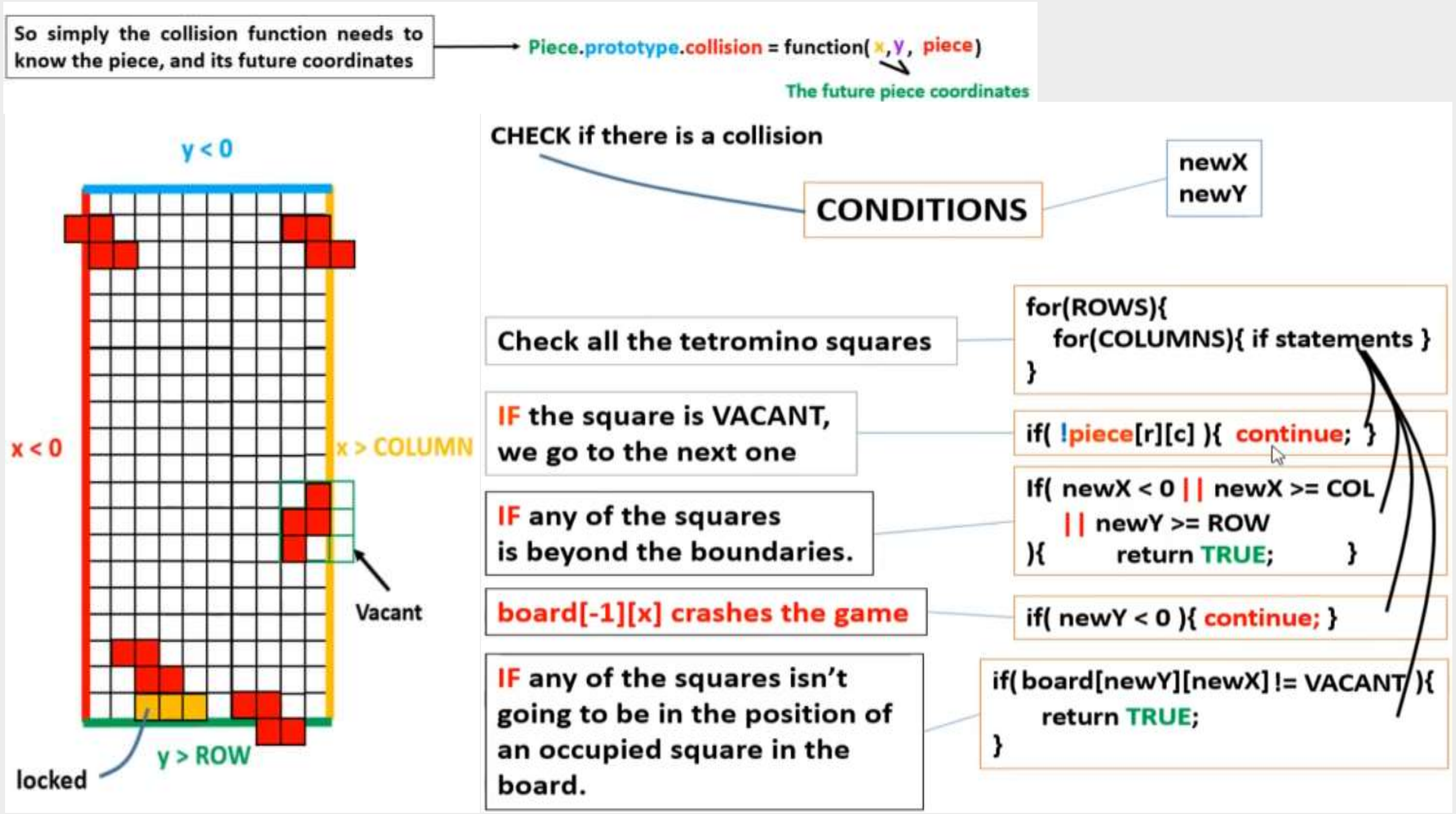
```
Piece.protoype.lock = function(){
        for( r = 0; r < this.activeTetromino.length; r++ ){
                for( c = 0; c < this.activeTetromino.length; c++ ){
                        if( ! this.activeTetromino[r][c] ){
                                continue;
                        }
      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

## Collision detection:




RIGHT, LEFT, DOWN, ROTATION
BEFORE any movement of a piece
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
We don't Check this piece we check this one.
moveLeft();
moveRight();

## Collision detection:



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

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;         }
```

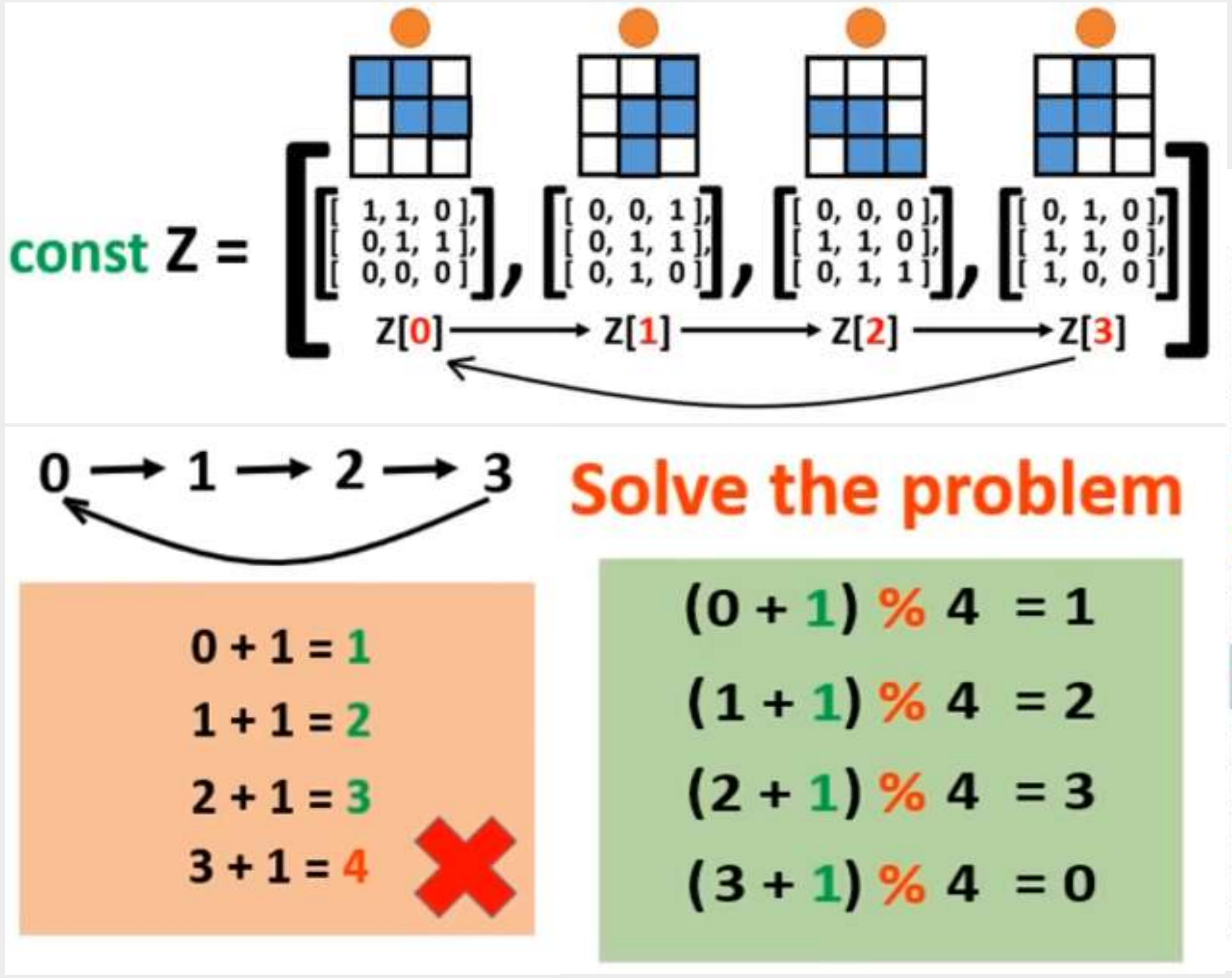
board[-1][x] crashes the game

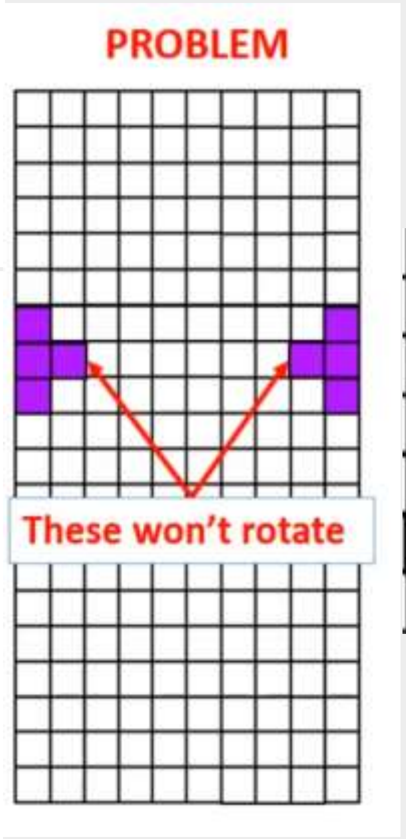
```
if( newY < 0 ){ continue; }
```

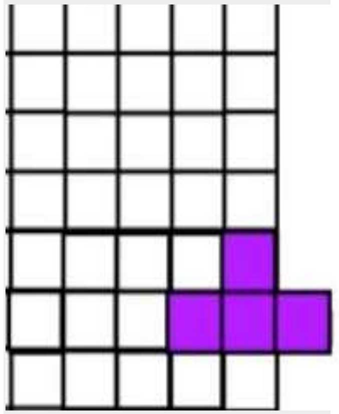
IF any of the squares isn't going to be in the position of an occupied square in the board.

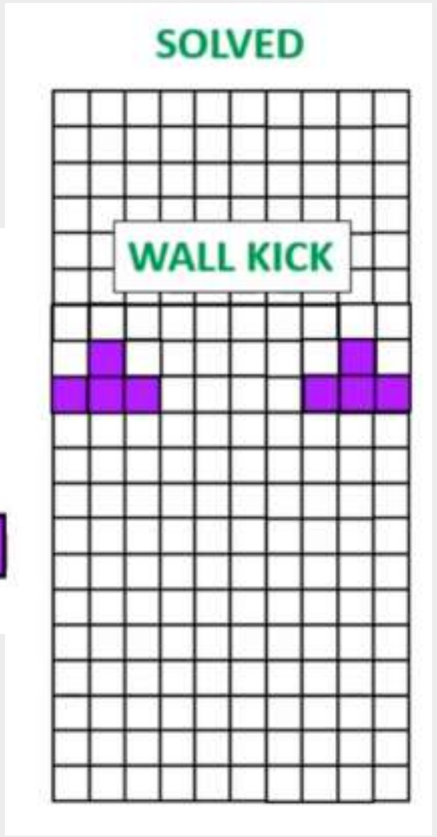
```
if( board[newY][newX] != VACANT ){
    return TRUE;
}
```

## Rotate a piece:

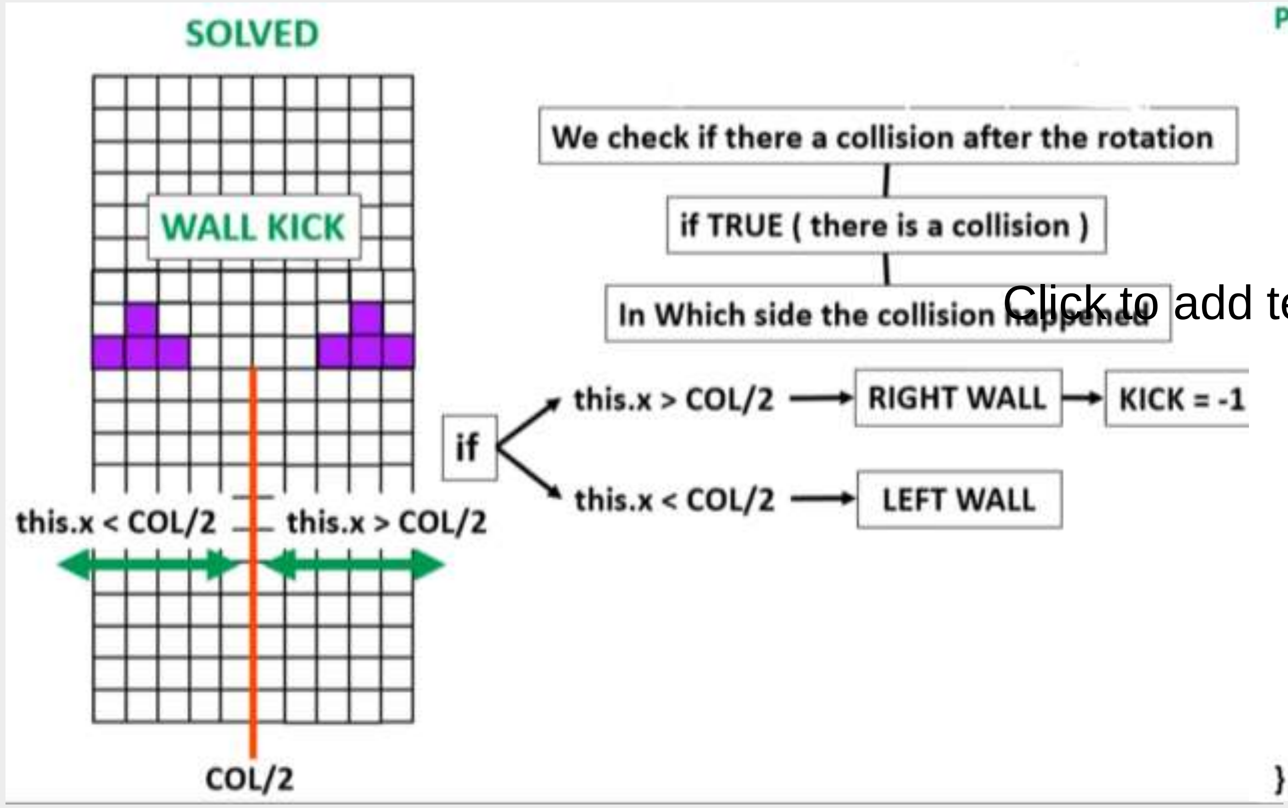




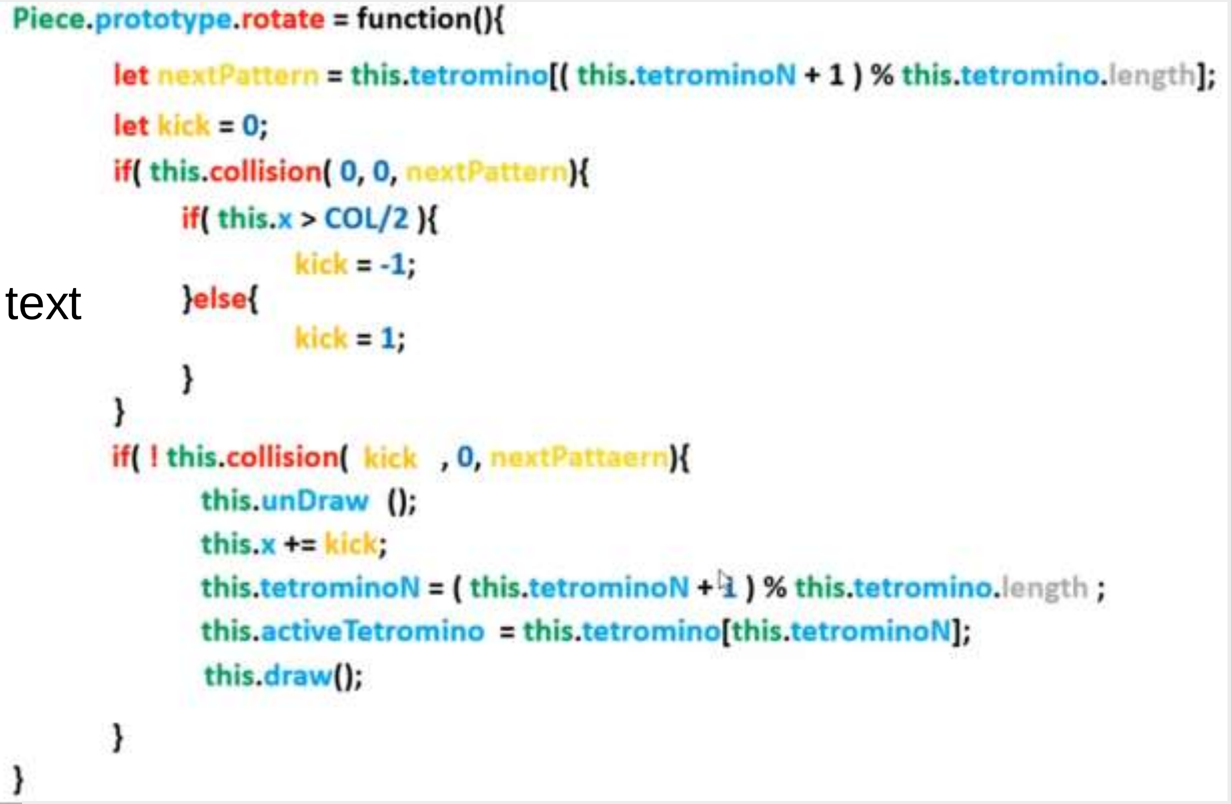




## Rotate a piece:

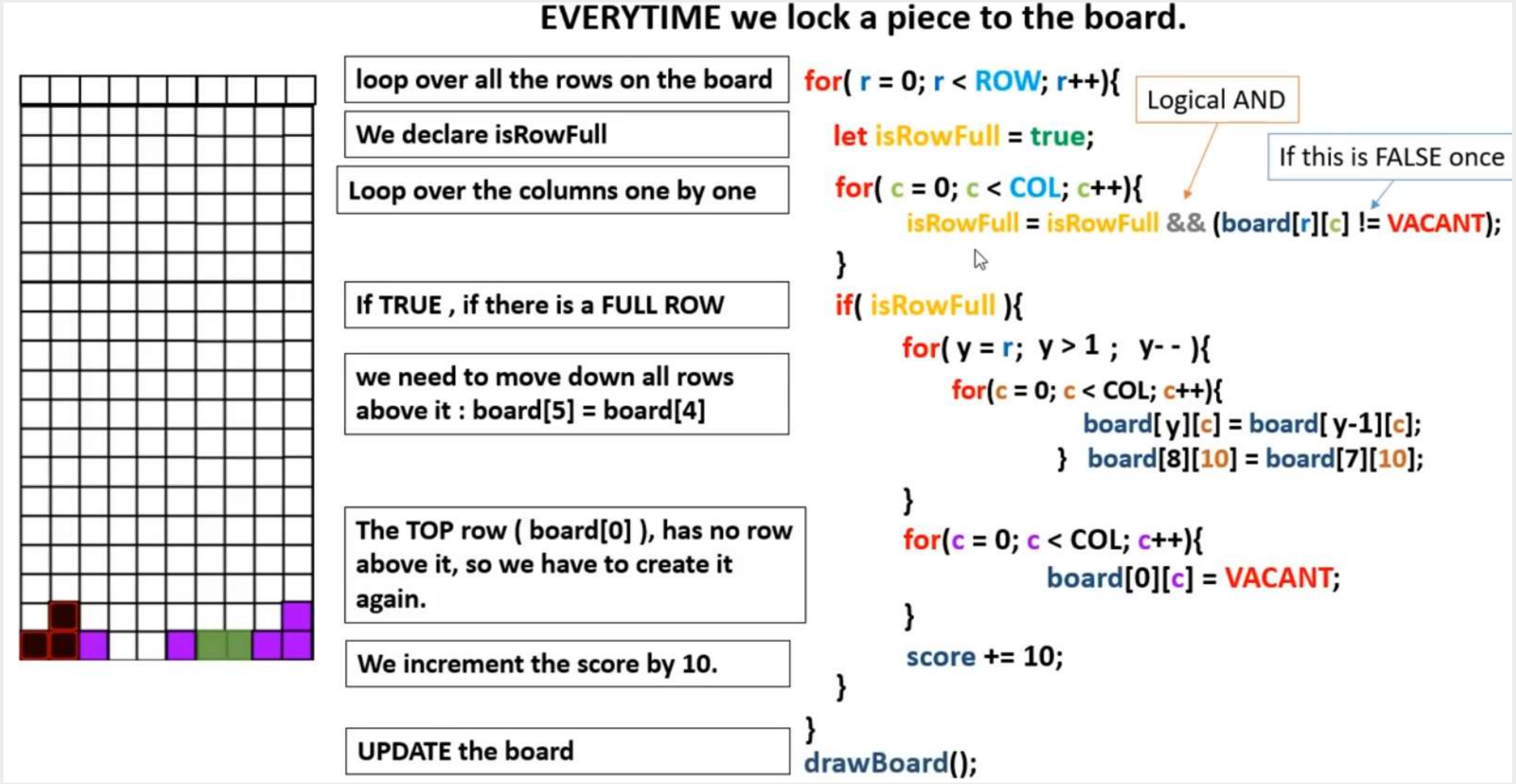

SOLVED
WALL KICK
We check if there a collision after the rotation
if TRUE ( there is a collision )
In Which side the collision happened
if
this.x > COL/2
RIGHT WALL
KICK = -1
this.x < COL/2
LEFT WALL
this.x < COL/2
this.x > COL/2
COL/2

Click to add text



```
Piece.prototype.rotate = function(){
    let nextPattern = this.tetromino[( this.tetrominoN + 1 ) % this.tetromino.length];
    let kick = 0;
    if( this.collision( 0, 0, nextPattern){
        if( this.x > COL/2 ){
                kick = -1;
        }else{
                kick = 1;
        }
    }
    if( ! this.collision(  kick  , 0, nextPattaern){
        this.unDraw  ();
        this.x += kick;
        this.tetrominoN = ( this.tetrominoN + 1 ) % this.tetromino.length ;
        this.activeTetromino  = this.tetromino[this.tetrominoN];
        this.draw();

    }
}
```

## Score and update block:



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();
```