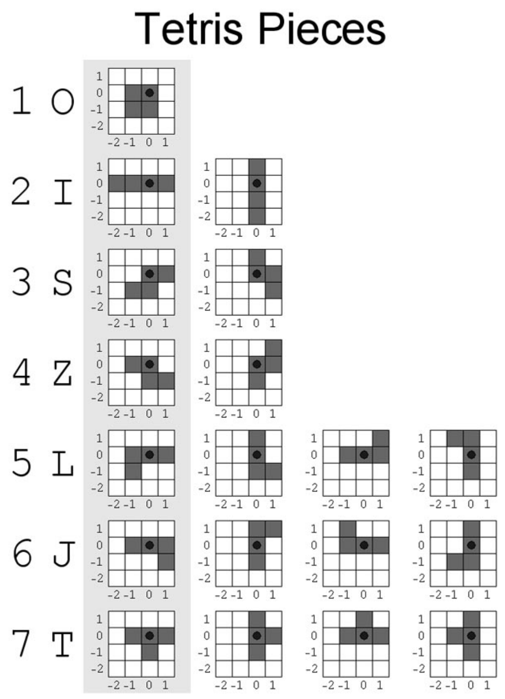
Tetris For Unity

Logic

In the Board script, when starting (Awake), we find the tilemap and the piece, in addition to initializing Tetrominoes. Tetrominoes have the following properties:

1. Tetromino itself
2. Tile (cell image)
3. Cell coordinates
4. Wall Kicks
    a. A wall kick occurs when the player rotates a piece when no space exists in the squares the tetromino would normally occupy after rotation.

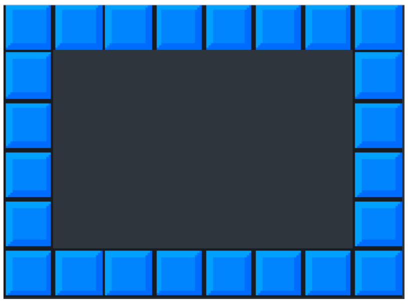
In general, we will have 7 tetrominoes:



To make the code easier to read, the Tetromino class is initialized by taking its coordinates and wall kicks ready from the static class, which cannot be implemented, Data. Cell coordinates and wall kicks are defined with { get; private set; }, that is, we are saying that the property can be read from outside the class, but can only be defined inside the class, this offers greater control over how information is accessed and modified within the class.

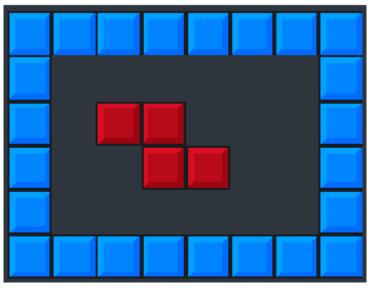
After initialization, the general control code (Board) is started (Start), where two functions are used:

1. SetNextPiece()
    a. Visual cleaning carried out if a piece is already defined

i. 

b. Generated a random tetromino and initialized it in the preview position (Initialize())

c. Visual creation of the new tetromino (Set()) made

i. 

2. SpawnPiece()

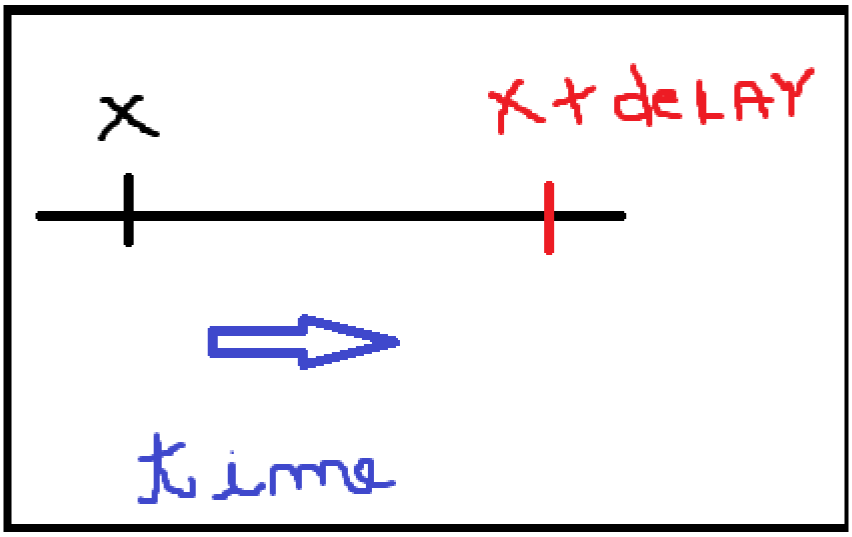
   a. Initialized a tetromino at spawn position

   b. IsValidPosition()

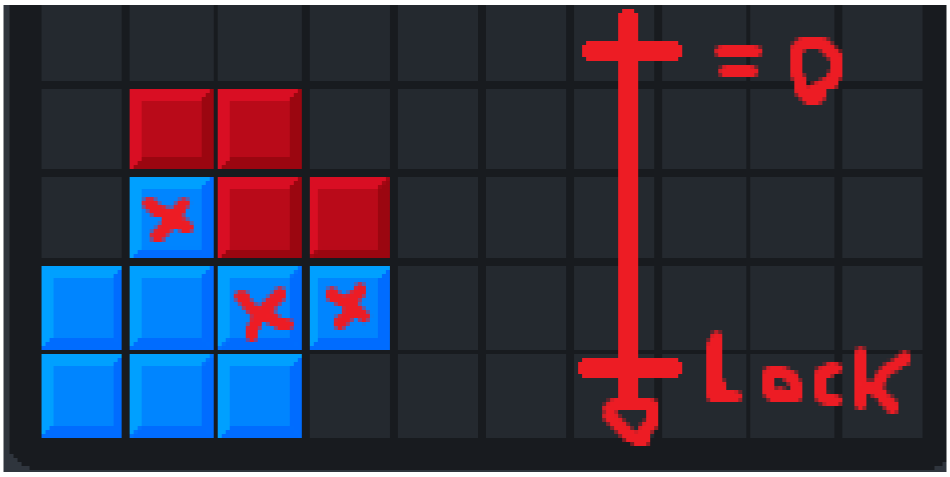
Cleaning and visual creation are done respectively by the Clear() and Set() functions, which pass cell by cell of the part, clearing its coordinates on the tile

All the logic above was done in the Board code, now let's go to the code of the piece that was spawned

For a part to be initialized, it needs to know the Tetromino data and also the position where it will be placed. After it is initialized, every X seconds it will automatically descend through the Step() function. How is the time comparison made? Initially, a value is defined for the delay, which is the X seconds of waiting to descend. Then another variable is created that will add the delay to the current time at that moment. When the current time, which is constantly updated, passes the previous variable, it means that the delay of X seconds has been reached, therefore the piece needs to move downwards



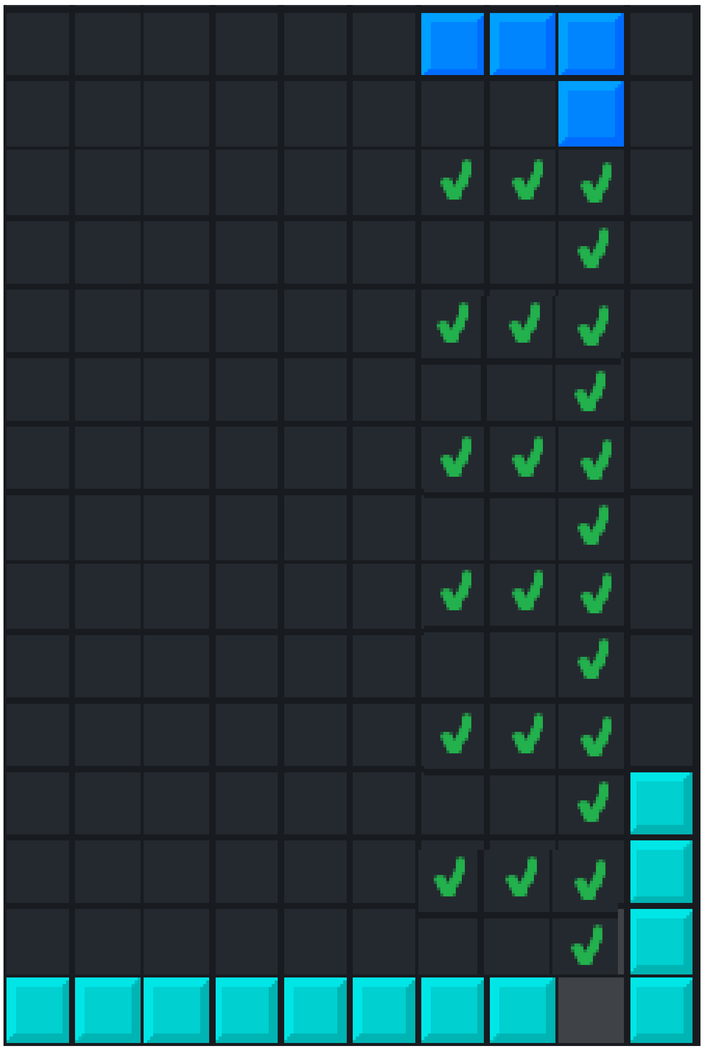
Furthermore, there is also lockDelay, which defines how long it takes for a part to be "deposited" in a soft drop. Every time we move a piece, the lockTime is reset to 0, and constantly, that is, every frame, the lockTime is added. When, when doing step(), the downward movement is invalid, that is, it is not possible to lower the piece as it is already at the limit, and the lockTime has already exceeded the lockDelay, the piece will then be deposited using the step() function. Lock()



The Lock() function is very simple, it:

1. Sets the final tiles of the deposited piece
2. **ClearTiles()**
3. Create a new part (Initialize())
4. Set the next piece (Set())

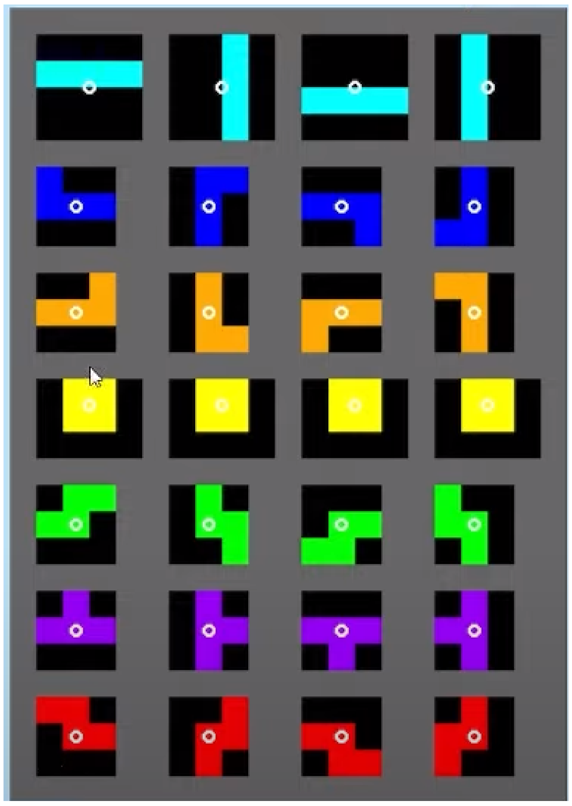
This Step() check is done for the Soft Drop, but what about the Hard Drop? Hard Drop is a downward movement loop, which is stopped when the check returns false. At that moment, the part is deposited



Furthermore, checking whether the next position is valid is done by the IsValidPosition() function in the board code, checking cell by cell whether the next position after applying the vector is valid or not. If any of the cells return false, the movement is not valid.

So far the creation of the pieces has been explained, their automatic movement downwards, the Soft Drop, the Hard Drop, the Lock, the Lock Time, the Step Time and the movement to the sides (simply applying the move, which already validates movement). The next step is to understand the rotation of the parts

Each of the 7 different pieces has 4 rotation possibilities:



All these possibilities are also stored in the Data class. We can rotate it in a counterclockwise direction (which would be to the right in the image above) or in a clockwise direction. When we do this, it is similar to manipulating a list, that is, when it passes coordinate 3 (element 4), it must return to zero. To rotate the part, manipulation is carried out piece by piece via the mathematical function below:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta \\ \text{sen}\,\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Example of a cell at coordinate (-1,1)

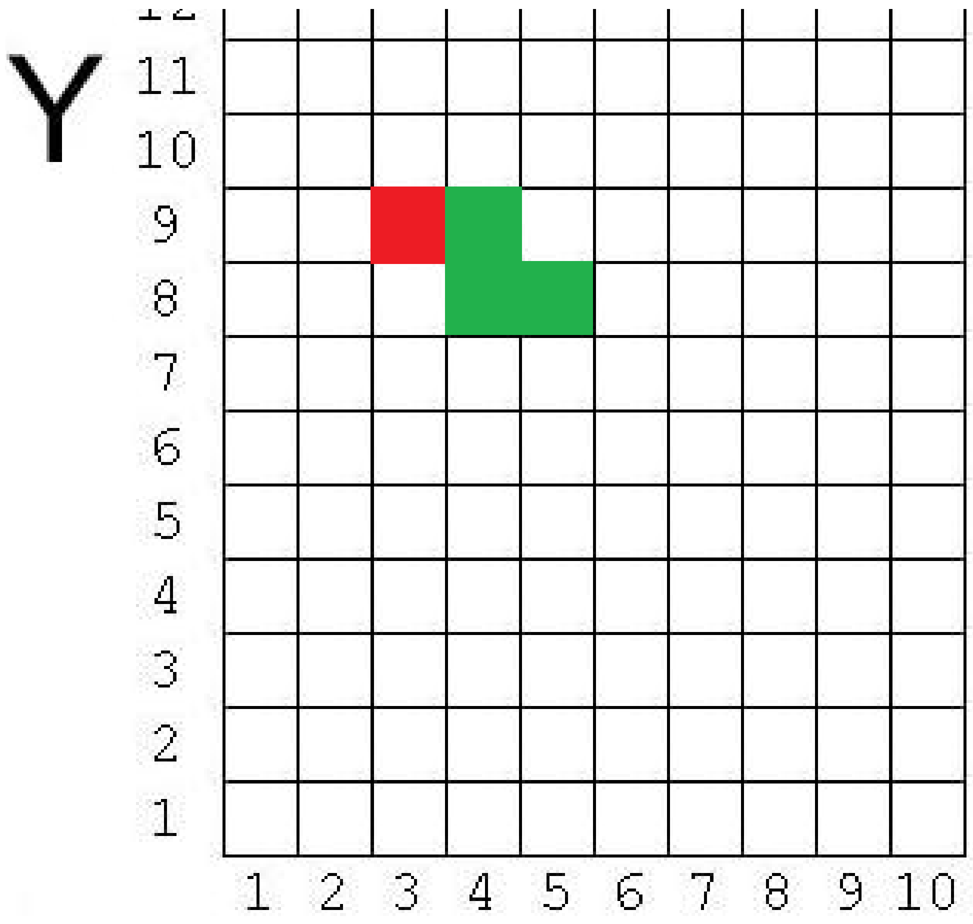
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta \\ \operatorname{sen}\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos 90 & -\operatorname{sen} 90 \\ \operatorname{sen} 90 & \cos 90 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & +1 \\ -1 & +0 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

The red cell below is similar to the code shown above



If we do the calculation for all 4 coordinates and assuming it is a rotation to the right (90°):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

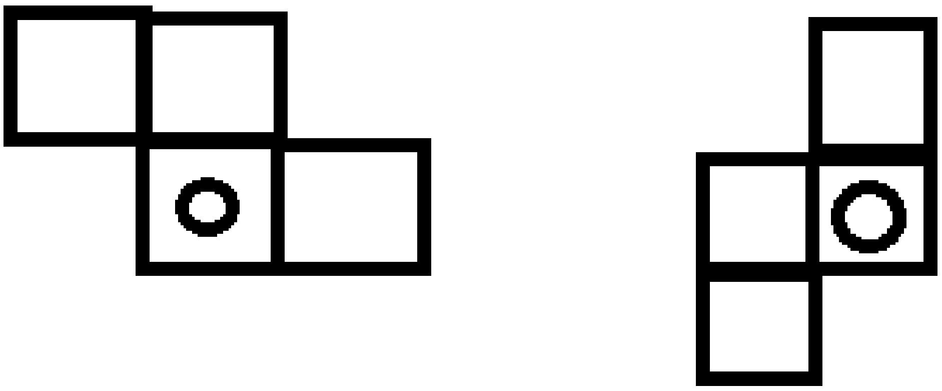
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

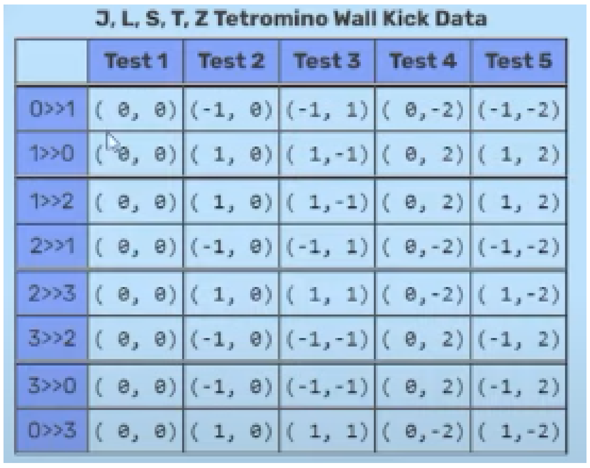
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Finally, we will have the past (left) and the future (right):



But there is still one problem left, the wall kick, which must be resolved by several movement tests.

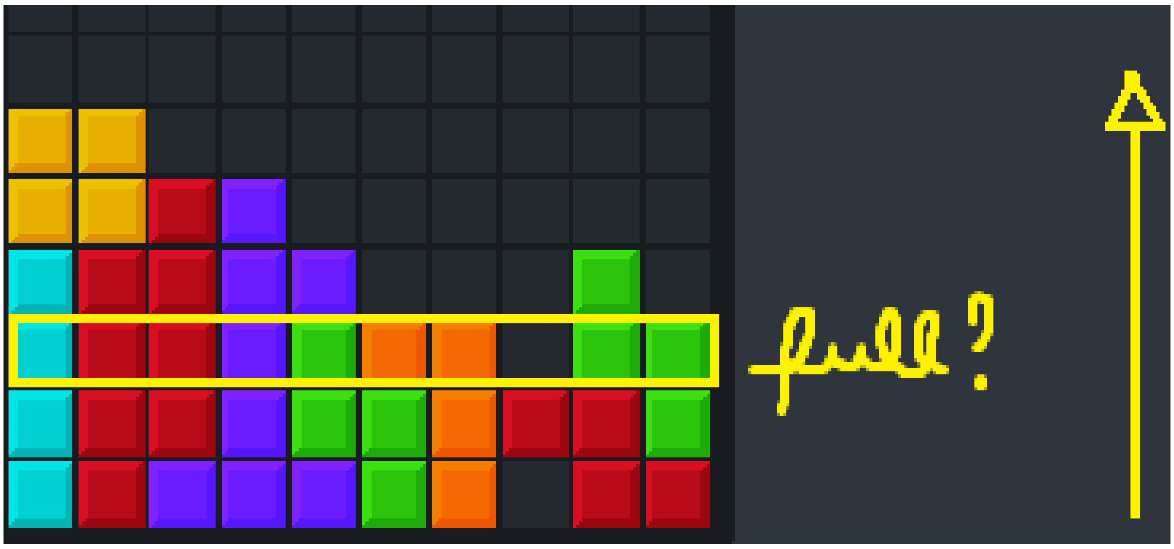


J, L, S, T, Z Tetromino Wall Kick Data

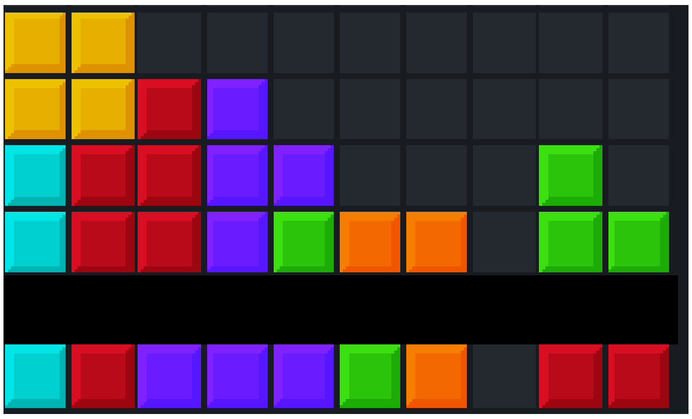
	Test 1	Test 2	Test 3	Test 4	Test 5
0>>1	(0, 0)	(-1, 0)	(-1, 1)	(0,-2)	(-1,-2)
1>>0	(0, 0)	(1, 0)	(1,-1)	(0, 2)	(1, 2)
1>>2	(0, 0)	(1, 0)	(1,-1)	(0, 2)	(1, 2)
2>>1	(0, 0)	(-1, 0)	(-1, 1)	(0,-2)	(-1,-2)
2>>3	(0, 0)	(1, 0)	(1, 1)	(0,-2)	(1,-2)
3>>2	(0, 0)	(-1, 0)	(-1,-1)	(0, 2)	(-1, 2)
3>>0	(0, 0)	(-1, 0)	(-1,-1)	(0, 2)	(-1, 2)
0>>3	(0, 0)	(1, 0)	(1, 1)	(0,-2)	(1,-2)

Only if all tests are valid is the rotation done. If it is not valid, rotation is performed in the opposite direction.

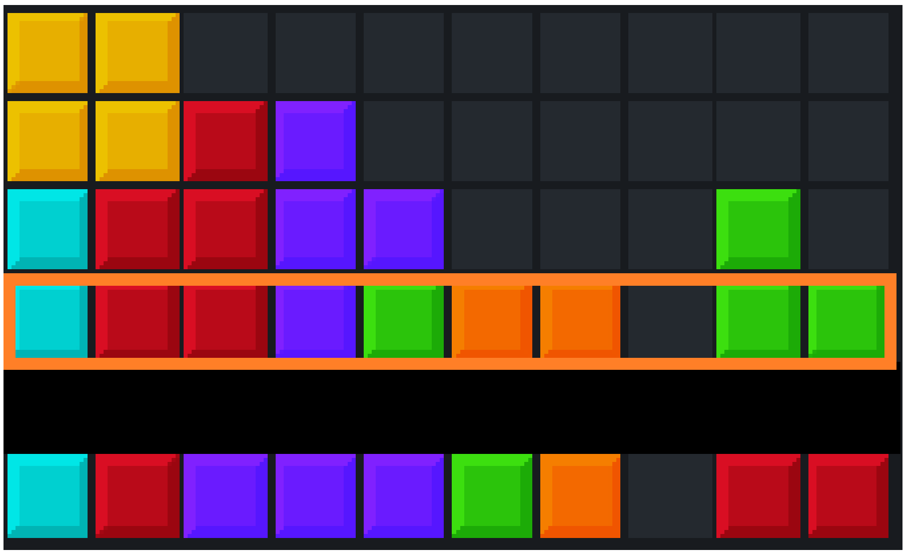
Now we deal with all the functions of the piece, from its spawn to wall kicks. But after the part is deposited, how is it cleaned? Cleaning is done in the Board code using the ClearLines() function, which is called every time a part is deposited, whether by Soft Drop or Hard Drop.

1. First, a rectangle is created with the same properties as the rectangle on the screen, to arrive at the edges (limits)
2. Reading is done by line and from bottom to top
    a. 

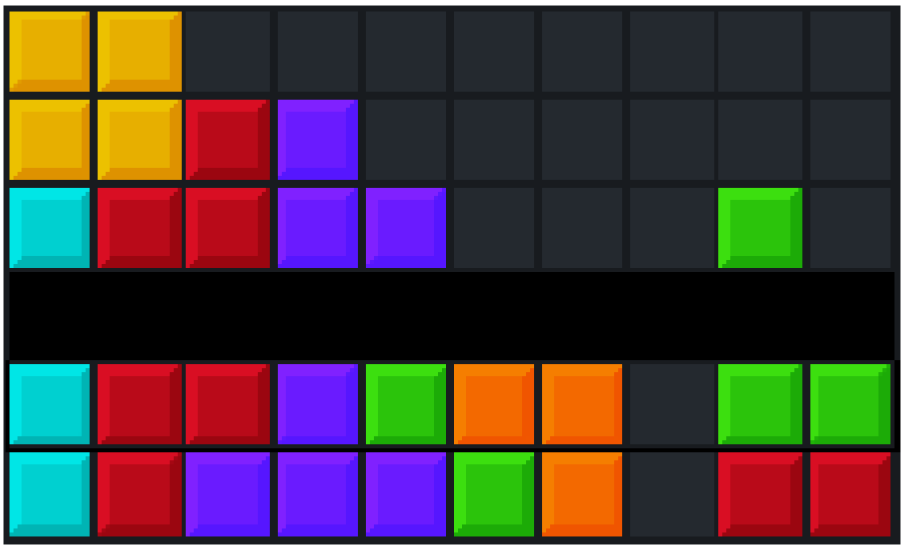
3. For each row, each column is read (similar to reading a matrix)
4. If all columns in the line are full, a LineClear() will be done, in which the current line is first cleaned



a.
b. Then save the data from the line above



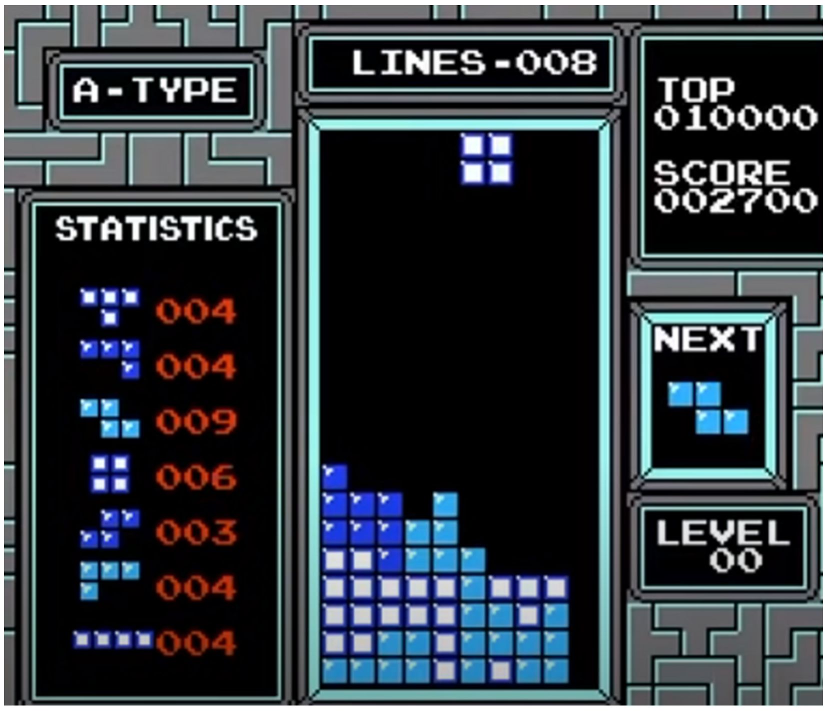
c.
d. And throw that line down



e.
f. This will be done on all lines in a loop, which only ends when it reaches the upper limit

g.

With all the topics presented, it is possible to create a Tetris game just like the NES game


A-TYPE
LINES-008
TOP
010000
SCORE
002700
STATISTICS
004
004
009
006
003
004
004
NEXT
LEVEL
00