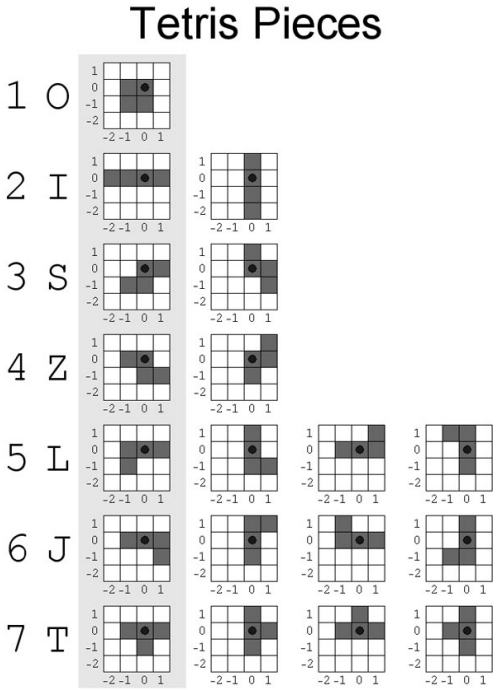
Tetris para Unity

Lógica

No script Board, ao iniciar (Awake), achamos o tilemap e a peça, além de inicializar o Tetrominoes. Os Tetrominoes têm as seguintes propriedades:

1. Tetromino em si
2. Tile (imagem da célula)
3. Coordenadas das células
4. Wall Kicks
    a. Um wall kick ocorre quando o jogador rotaciona uma peça quando nenhum espaço existe nos quadrados onde o tetromino normalmente ocuparia após a rotação

Em geral, teremos 7 tetrominoes:



Para facilitar a leitura do código, a classe do Tetromino é inicializada pegando suas coordenadas e wall kicks já prontos da classe estática, a qual não pode ser implementada, Data. As coordenadas das células e os wall kicks são definidos com { get; private set; }, ou seja, estamos dizendo que a propriedade pode ser lida de fora da classe, mas só pode ser definida dentro da classe, isso oferece maior controle sobre como as informações são acessadas e modificadas dentro da classe.

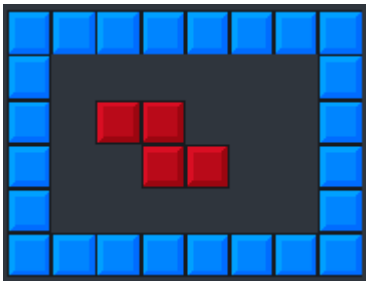
Após a inicialização, o código geral de controle (Board) é startado (Start), onde duas funções são usadas:

1. SetNextPiece()
    a. Feita a limpeza visual caso tenha uma peça já definida

i. 

b. Gerado um tetromino aleatório e inicializado na posição de preview (Initialize())

c. Feita a criação visual do novo tetromino (Set())

i. 

2. SpawnPiece()

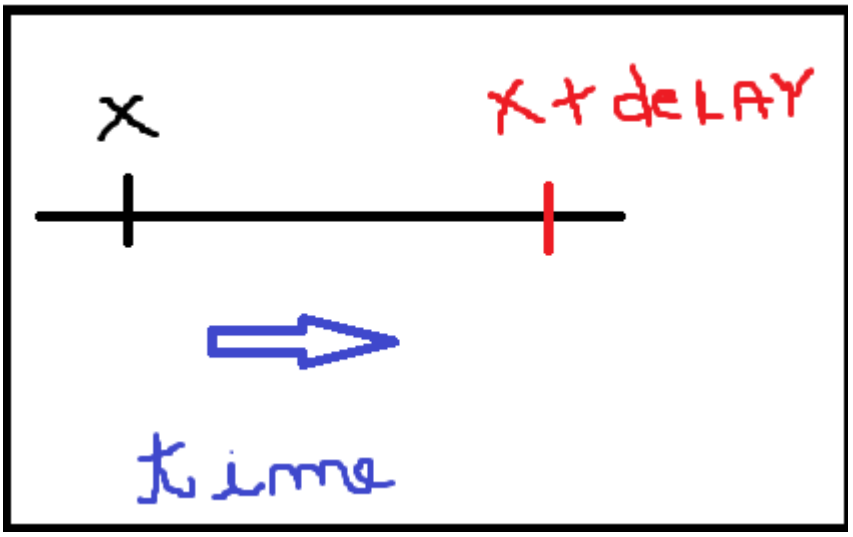
    a. Inicializado um tetromino na posição de spawn

    b. IsValidPosition()

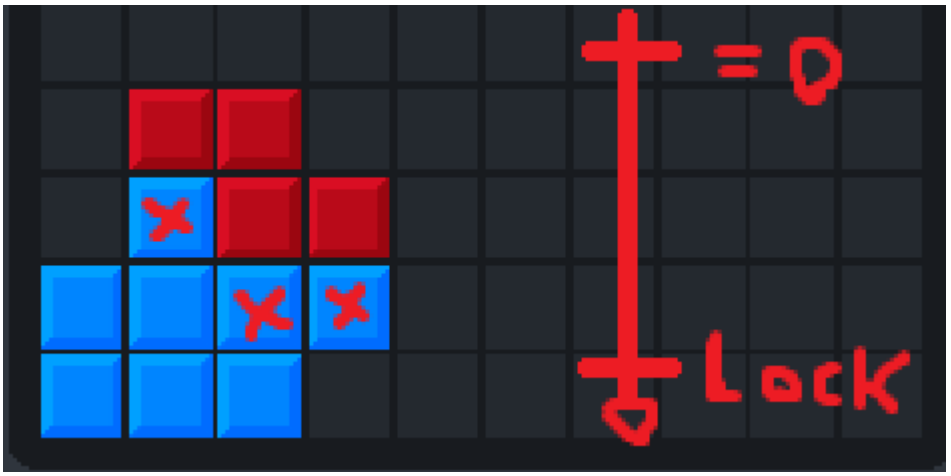
A limpeza e criação visual são feitas respectivamente pelas funções Clear() e Set(), as quais passam célula por célula da peça, limpando suas coordenadas no tile

Toda lógica acima foi feita no código de Board, vamos agora para o código da próprio peça que foi spawnada

Para uma peça ser inicializada, ela precisa saber dos dados do Tetromino e também da posição onde será colocada. Após ela ser inicializada, a cada X segundos ela irá descer automaticamente por meio da função Step(). Como é feita a comparação de tempo? Inicialmente é definido um valor para o delay, que são os X segundos de espera para descer. Depois é criado uma outra variável que vai somar o delay ao tempo atual daquele momento. Quando o tempo atual, o qual é atualizado constantemente, passar a variável anterior, significa que o delay de X segundos foi alcançado, portanto a peça precisa mover para baixo



Ademais, há também o lockDelay, que define quanto tempo demora para uma peça ser “depositada” em um soft drop. Toda vez que movemos uma peça, o lockTime é resetado para 0, e constantemente, ou seja, a cada frame, o lockTime é somado. Quando, ao fazer o step(), o movimento para baixo for inválido, ou seja, não é possível descer a peça pois ela já está no limite, e o lockTime já ultrapassou o lockDelay, a peça será então depositada por meio da função da Lock()



A função Lock() é muito simples, ela:

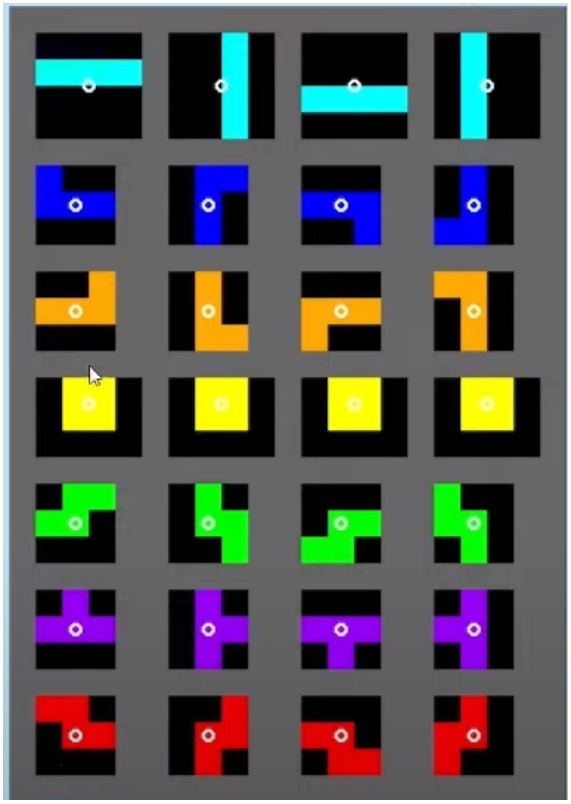
1. Seta os tiles finais da peça depositada
2. **ClearTiles()**
3. Cria uma peça nova (Initialize())
4. Seta a próxima peça (Set())

Esta verificação do Step() é feita para o Soft Drop, mas como fica o Hard Drop? O Hard Drop é um loop de movimentações para baixo, o qual é parado quando a verificação retorna falso. Naquele momento, a peça é depositada

Ademais, a verificação de se a próxima posição é válida é feita pela função IsValidPosition() no código de board, verificando célula a célula se a próxima posição depois da aplicação do vetor é válida ou não. Caso qualquer uma da célula retorne falso, o movimento não é válido

Até agora foi explicado a criação das peças, a sua movimentação para baixo automaticamente, o Soft Drop, o Hard Drop, o Lock, o Lock Time, o Step Time e a movimentação para os laterais (simplesmente aplicação do move, o qual já valida a movimentação). O próximo passo é entender a rotação das peças

Cada uma das 7 diferentes peças tem 4 possibilidades de rotação:



Todas essas possibilidades ficam armazenadas também na classe de Data. Podemos rotacionar na direção anti-horária (que seria para a direita na imagem acima) ou na direção horária. Quando fizermos isso, é semelhante a manipulação de uma lista, ou seja, quando passar da coordenada 3 (elemento 4), deve voltar ao zero. Para rotacionar a peça, é feita a manipulação peça a peça via a função matemática abaixo:

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

Exemplo de uma célula na coordenada (-1,1)

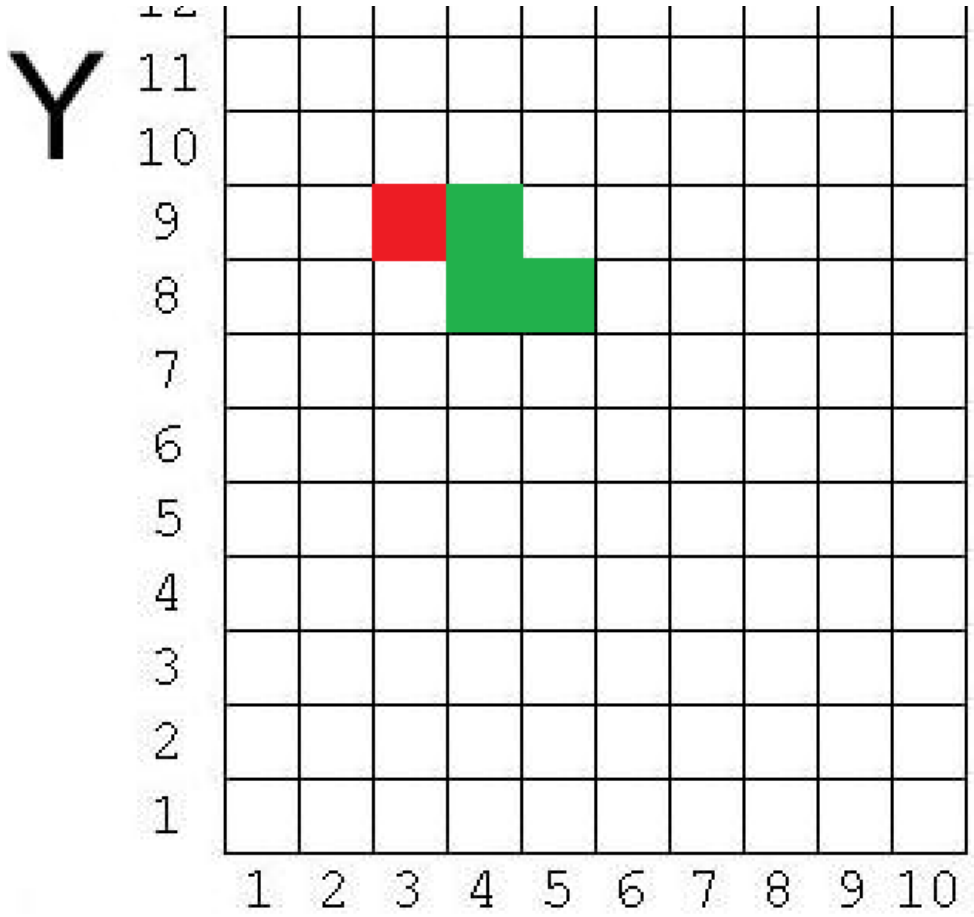
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta \\ \text{sen}\,\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

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

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

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

A célula em vermelho abaixo é semelhante ao código apresentado acima



Se fizermos o cálculo para todas as 4 coordenadas e supondo que seja uma rotação para direita (90°):

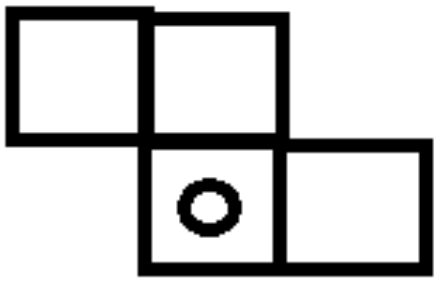
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

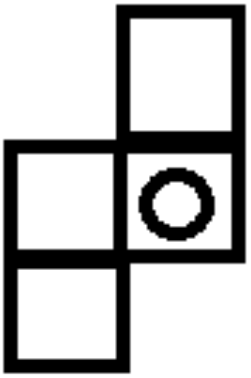
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

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

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

Por fim, teremos então o passado (esquerda) e o futuro (direita):





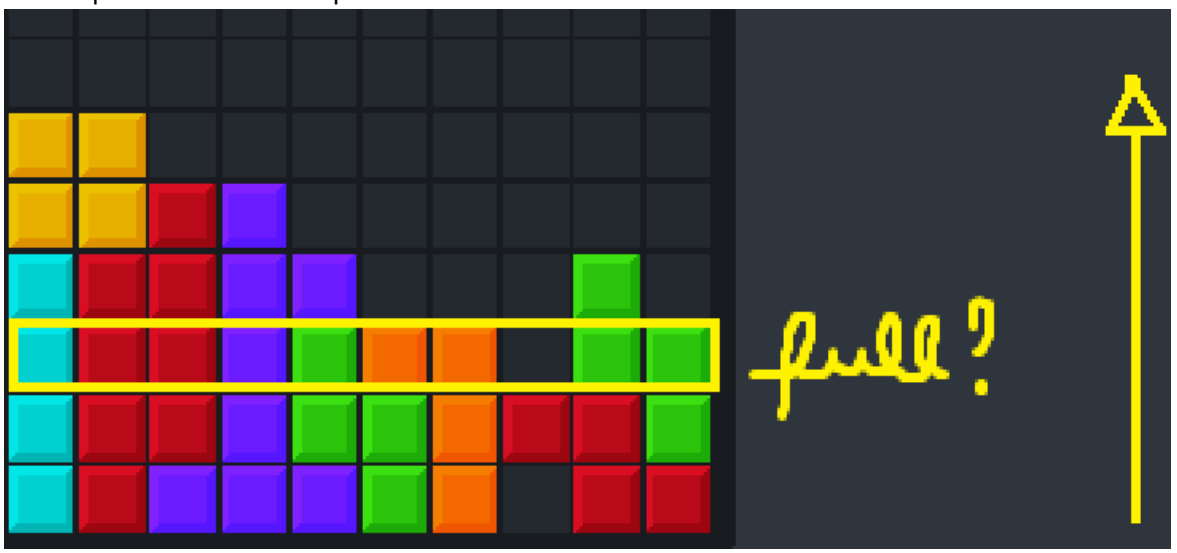
Mas ainda sobra um problema, o wall kick, o qual deve ser resolvido por diversos testes de movimentação.

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

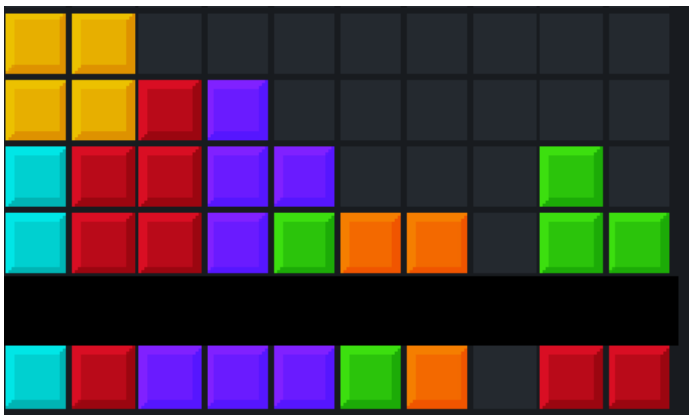
	Test 1	Test 2	Test 3	Test 4	Test 5
0>>1	(0, 0)	(-1, 0)	(-1, 1)	(0,-2)	(-1,-2)
1>>0	(0, 0)	(1, 0)	(1,-1)	(0, 2)	(1, 2)
1>>2	(0, 0)	(1, 0)	(1,-1)	(0, 2)	(1, 2)
2>>1	(0, 0)	(-1, 0)	(-1, 1)	(0,-2)	(-1,-2)
2>>3	(0, 0)	(1, 0)	(1, 1)	(0,-2)	(1,-2)
3>>2	(0, 0)	(-1, 0)	(-1,-1)	(0, 2)	(-1, 2)
3>>0	(0, 0)	(-1,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

Somente se todos os testes forem válidos, a rotação é feita. Se ele não for válido, é feita a rotação no sentido inverso.

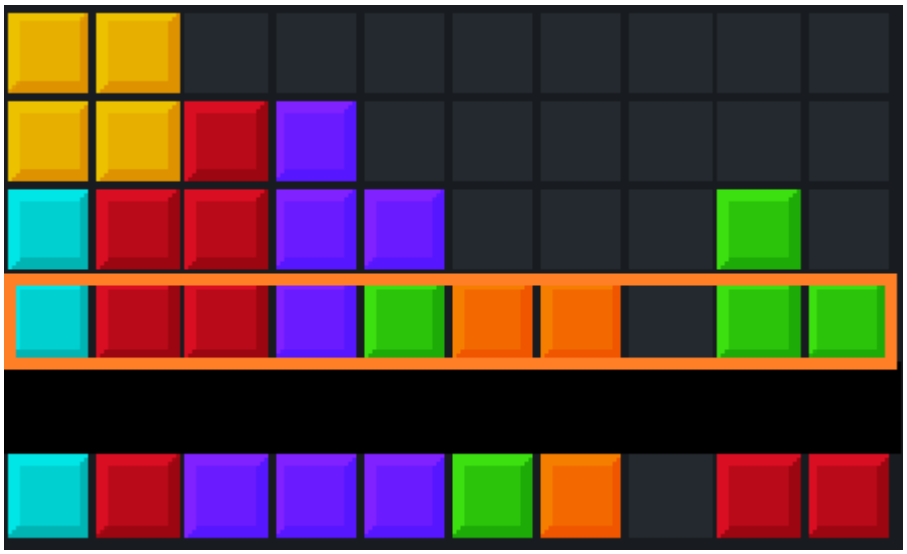
Agora tratamos de todas as funções da peça, desde de seu spawn até os wall kicks. Mas depois que é feito o depósito da peça, como é feita a limpeza? A limpeza é feito no código Board por meio da função ClearLines(), a qual é chamada toda vez que uma peça é depositada, seja por Soft Drop ou Hard Drop.

1. Primeiro é feita a criação de um retângulo com as mesmas propriedades do retângulo na tela, para chegarmos assim nas bordas (limites)
2. A leitura é feita por linha e de baixo para cima
   a. 

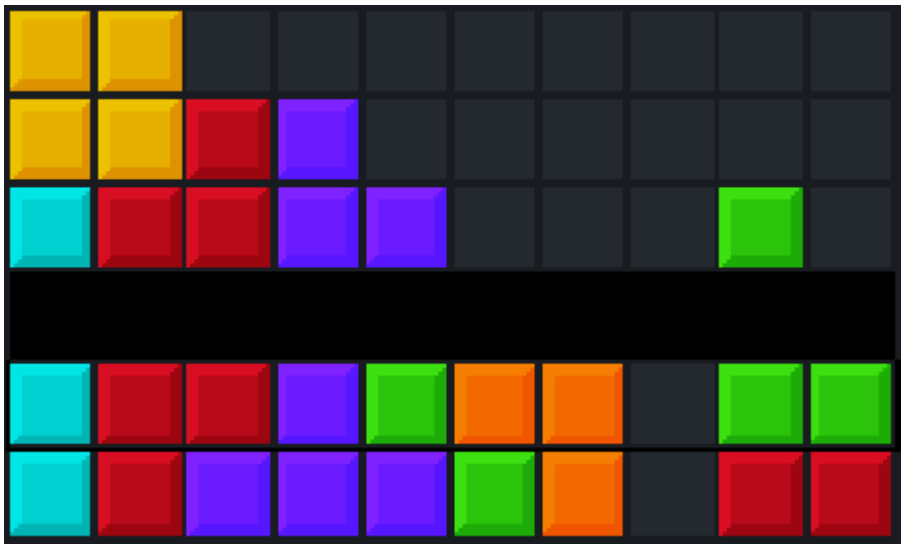
3. Para cada linha, é feita a leitura de cada coluna (semelhante a leitura de uma matriz)
4. Se todas as colunas da linha estiverem cheias, será feita um LineClear(), no qual primeiro é feita a limpeza da linha atual

a. 

b. Depois salva os dados da linha acima

c. 

d. E jogar essa linha para baixo

e. 

f. Isso será feito em todas as linhas em um loop, o qual só finaliza quando chega no limite superior

g. 

Com todos os tópicos apresentados, é possível criar o jogo Tetris igual ao de NES