

Assignment # 4 – Part 1

Purpose

The purpose of this assignment was to complete `TetrisPiece.java` and `JPieceTest.java`. `TetrisPiece.java` contains the information regarding a tetris piece's block arrangement and next rotation and `JPieceTest.java` displays the seven pieces and their rotations, making it useful for testing the correctness of `TetrisPiece.java`.

Implementations

TetrisPiece.java

There were four primary concerns when `TetrisPiece` class: creating the skirt and implementing the `equals`, `getPieces`, and `nextRotation` methods. As per the assignment, the skirt contains the “lowest y extent of each x coordinate.” Thus, with a piece comprised of blocks with the following coordinates

```
[(0, 0), (1, 2), (1, 1)]
```

will have the following `int` array as a skirt:

```
[0, 1]
```

To create the skirt, the program declares the `int` array `skirt`, initializes it to an `int` array with a length equal to the tetris piece's width (so that each unique x-coordinate has a corresponding index in the array), and sets each index's value to `Integer.MAX_VALUE` (so that every y-extent is lower than the starting value). Finally, the program increments through each block in the piece and, if the y-coordinate of the block is less than the index of `skirt` corresponding to the block's x-value, sets that index to the piece's y-value. Thus, the skirt for the above tetris piece will progress as follows:

```
[Integer.MAX_VALUE, Integer.MAX_VALUE]  
[0, Integer.MAX_VALUE]  
[0, 2]  
[0, 1]
```

Implementing the `equals` method, which tests for equality between two tetris pieces, was pretty straightforward. The only somewhat unusual feature was that two identical tetris pieces were not required to have their blocks ordered identically. Thus, the tetris piece

```
[(0, 0), (1, 2), (1, 1)]
```

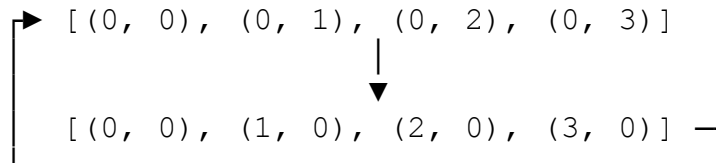
is equal to the tetris piece

```
[(1, 2), (1, 1), (0, 0)]
```

To account for this, the `equals` method increments through the `body` array of the first tetris piece and tries to find the block at the current index *somewhere* in the second tetris piece's `body` array. The first time that a block found in the first tetris piece cannot be found in the second, the method returns `false`. Otherwise, a block is found in one tetris piece if and only if it is found in the second piece and the method returns `true`.

This implementation assumes that a tetris piece containing multiple instances of the same block is equal to a tetris piece with only one instance of that block.

The `getPieces` method accesses the `Piece` class's `String` array `pieceStrings` and feeds each `String` from this array into the `Piece` class's `parsePoints` method, which returns the corresponding `Point` array. Each `Point` array is then used to make the corresponding `TetrisPiece`. The piece's protected member variable `next` is then assigned to the piece's `nextRotation`, which, in turn, has its protected member variable `next` assigned to its `nextRotation`, and so on until one of the subsequent piece rotations' `next` is referencing the initial `TetrisPiece`. For the "stick" tetris piece, this chain would look like



Finally, the `nextRotation` method, which provides the ninety-degree counter-clockwise rotation of the current piece, was relatively simple to create. Each block in the current tetris piece has a counterpart for obvious reasons (just look at the above "stick" example). The relationship of these two blocks is that the rotated block's y-coordinate is equal to the original block's x-coordinate and the rotated block's x-coordinate is equal to the height of the original tetris piece minus the original block's y-coordinate minus one.

JPieceTest.java

The bulk coding the `JPieceTest` class went toward the `drawPiece` method. When sent in a `Graphics` object, a `Piece` object, and a `Rectangle`, the `Piece` object and its rotations are drawn to the `Graphics` object in such a way as to not overstep the bounds specified by the `Rectangle` object. Additionally, each row of tetris pieces would be drawn in such a way as to make the pieces and rotations line up in aesthetically pleasing columns.

To create the columns of tetris pieces, the first tetris pieces in each row would have to start at the same x-coordinate, the second tetris pieces in each row would have to start at a different x-coordinate farther across the screen, and so on. This effect is achieved by drawing each tetris piece at an x-coordinate that is some constant C times the number of pieces that have already been drawn on that row. If C were to be 100, the first tetris piece would be drawn at 0, the second would be drawn at 100, the third would be drawn at 200, and so on.

Additionally, the blocks within each tetris piece should line up with the blocks in the other tetris pieces. Again, this is accomplished with some constant D multiplied by the x-value of the block. Thus, to find the position of a block on the board, multiply C by the number of tetris pieces already drawn on that row (which is the position of the tetris piece the block is a part of) and add to that D multiplied by the x-value of the block.

How are C and D found? C is equal to the screen width divided by the maximum number of rotations a block can have (four), which is equal to the amount of screen space allocated to each tetris piece. This means that every row has space allocated for four rotations, even if the tetris piece on the row only has two rotations. D is found by dividing C by the maximum width a piece can have, which again is four. Similar to C , the use of the constant D makes it so that when drawing any tetris piece enough space for four columns of blocks is allocated even when fewer than that are needed.

Testing

The extent of the testing performed on these two programs was running `JPieceTest` and seeing that the tetris pieces were displayed correctly.

Pair Programming

Initially, we did not meet as a pair because Colin was reliving childhood memories back at his home during the weekend. As a result, John spent about an hour and a half on Friday and two hours on Saturday coding and writing this report. However, less than an hour before this project was due, Colin attempted to debug John's code, but found it impeccable. Two golden stars for John.

Colin also added an 'and' and fixed a typo in this report. Only one golden star for that.