

Mathew deWet and Christopher Wiley
October 9, 2006

TETRISPIECE

Overview:

The general goal of this milestone was to implement TetrisPiece, which is an implementation of the Piece interface. Sticking to the interface will allow us to use it in the actual Tetris game, provided we managed not to entangle ourselves in the implementations. Implementation was completely up to the designers, however, it was hinted at in class that this might be a good time to try some witty optimizations for speed, as JAVA GUI is not supposed to be terrific.

Algorithm Analysis:

In general, we tried to optimize for speed rather than space. Toward this end, once we calculate the skirt and the height, we store each in an instance variable. Width is simply the length of the array representing the skirt. In this way, we've managed to implement a TetrisPiece in such a way that all the getters run in $O(1)$ time. All calculation is finished after the first call to `getPieces()`. Not bad eh?

Implementation:

The TetrisPiece class is actually a node in a linked list which contains references to other pieces which are also nodes which eventually refer back to the beginning. This is in fact a circularly linked list. The list is created in the `getPieces()` method. The method returns an array of Pieces which are the initial rotations of each linked list. The meat of the method is a while loop which loops until it sees that it has linked back to the initial node.

Our rotation method begins by flipping the points of each piece over the line $y=x$. This is equivalent to switching the y and x values. Then the resulting point is flipped over the y axis, which is equivalent to negating the x value. The final step is to translate the points so that the origin is once again in the bottom left corner instead of the bottom right. This process rotates pieces counter clockwise.

Itches:

At several points, our program returns references to arrays which maintain state information about the individual pieces. This bugged us but seemed to be what the documentation was describing. We made `calcStuff()` a method, but really it should be in the constructor. Its just easier to look at it that way.

Testing:

The `JPieceTest` exposed the fact that we had a problem with our `equals` method. It turns out that we returned false whether or not we meant to return true or not. It was a matter of changing which keyword we typed out. Other than that, as the assignment said, `JPiece Test` is sort of a guarantee that the implementation is essentially correct.

JPIECE TEST

Overview:

This was a tester for the TetrisPieces class. The gist of it is that the `JTestPiece` is actually a component which is drawn into a

JFrame. In reality this was a pain and a nuisance, because graphics are no fun. This program essentially drew out all the rotations of each of the pieces, and highlighted the skirts. It also drew out some basic information about the width and height.

Algorithm Analysis:

There really were no algorithms for this to speak of, just an implementation of the specification. This was all about reading documentation or making up rules as we went. The algorithm was just to delegate the actual job of drawing the stuff into a very general method called drawPiece which did the actual work. The paintComponent method calculates how much area each rotation is allowed to get by dividing its available area into four rectangles. It then passes each rectangle and a Piece rotation into drawPiece which does the actual graphics work.

Implementation:

Once again, a loop that goes until it sees it got back to where it started is used to get the rotations in paintComponent. Once a new rotation is developed, drawPiece is called which a reference to the rectangle in which the drawing is to occur. The method drawPiece just subdivided that area further into a four by four square and filled the squares which represented the piece. The square doesn't always fit perfectly into the rectangle given. After the squares were drawn, we drew a string with the height and width over top in red.

Itches:

This is the only part we really felt we hacked through. The String in red at the bottom for instance, we kind of did by looking at it and saying, that's not especially good, let's move it over by a pixel or two. I'm a little miffed why this wasn't written for us. All in all graphics is a messy creature because what's required is messy, but that doesn't make it better.

We used all the code which was provided to us in our creation of this monstrosity.