Tetrix Project (Part 4).docx

Continue to work with your partner on this assignment. Don't forget to switch roles often.

Exercise 4.0: Death Row

We would like to clear any rows the user completes. First, go ahead and complete the private helper method `isCompletedRow` in the `Tetris` class.

```
//precondition:  0 <= row < number of rows
//postcondition: Returns true if every cell in the
//               given row is occupied;
//               returns false otherwise.
private boolean isCompletedRow(int row)
```

Next, given a row that is completed, we need to be able to clear that row, and then move down (by one row) all of the blocks that are above that row. Complete the following private helper method `clearRow` in the `Tetris` class. Don't forget to loop through every row above the given row in order to move down all of the blocks that are above. Remember that the row numbers will be decreasing as you move upwards. Also remember that both the `Block` and its `grid` are keeping track of the `Block`'s location, and that it's up to the `Block` class to keep these consistent. In other words, you just can't just remove a location from the grid, but rather a given `Block` must `removeSelfFromGrid` instead. *This can be a challenging method to write properly, so think about what you want to do before you write the code for this method.*

```
//precondition:  0 <= row < number of rows;
//               given row is full of blocks
//postcondition: Every block in the given row has been
//               removed, and every block above row
//               has been moved down one row.
private void clearRow(int row)
```

Now use the above helper methods to complete the following `Tetris` method. Remember that you want to start with the bottom row, and work your way up to the top. Don't forget that when a row is cleared and everything above moves down one row, you need to check that same row again to see if it should be cleared as well.

```
//postcondition: All completed rows have been cleared.
private void clearCompletedRows()
```

Whenever a tetrad stops falling, call `clearCompletedRows`. Now go play Tetris!

(Yes, your game will misbehave when you lose, but there's a simple work-around to this problem: Don't lose.)

Exercise 4.1: Last Request

Complete the private helper method `topRowsEmpty` in the `Tetris` class.

```
//returns true if top two rows of the grid are empty
//false otherwise
private boolean topRowsEmpty()
```

Use this method to break out of the `play` method when there is no more room to insert a new tetrad at the top of the grid.

Finish your game by gradually increasing the speed at which the tetrads fall, and by calculating and displaying a score in the window title.

You may add any enhancements of your own, if you wish. When you are finished with your game, please demonstrate your work for me so that I can know that you have completed the (entire) assignment. Your game should play correctly all the way through to the end without any error messages and automatically stop when the game is over. I will want to see that your game is capable of clearing two contiguous rows (two complete rows that are next to each other) at the same time.

A Note About Threads

Tetris, as we've implemented it, is a multi-threaded application. One thread begins in the main method, creates the grid and display, and then loops around inside the play method, continually shifting the active tetrad down one row, and occasionally clearing rows. The other thread is started automatically by Java to draw the user interface and handle key presses. This means that it may sometimes happen that the main thread is shifting the active tetrad down or clearing rows *at the same time as* the user is pressing an arrow key to translate/rotate the active tetrad. If this happens, all sorts of things may go wrong. Maybe only a couple blocks in the active tetrad will shift down, or maybe an exception will be thrown. Solving this problem is extremely tricky, *so it's best to pray that you never need to know about this stuff.*

Thankfully, for the most part, this problem seems to be extremely rare. But every now and then, for whatever reason, it'll happen that some student hits this problem constantly. The problem can be fixed by declaring some methods in the `Tetris` class as synchronized, meaning that only one of those methods may be called at a time. For example, it may be best to synchronize `upPressed`, `downPressed`, etc., along with the inside of the play method's loop (in other words, put the inside of the loop in a synchronized helper method).