Tetrix Project (Part 2).docx

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

Exercise 2.0: Four Blocks and Seven Shapes Ago, ...

Modify the `Tetrad` class's constructor so that it first picks a random shape number from 0 to 6. If it picks 0, it will create a red I-shaped tetrad as before. But, for each of the other numbers, it will pick one of the other 6 shapes. Now run your program a few times to test that a random tetrad appears at the top of your Tetris window. Make sure each one of the seven shapes appears at least once, and that they are in the proper positions (top center) on the grid.

Exercise 2.1: Unblocking

Complete the following two private helper methods in your `Tetrad` class.

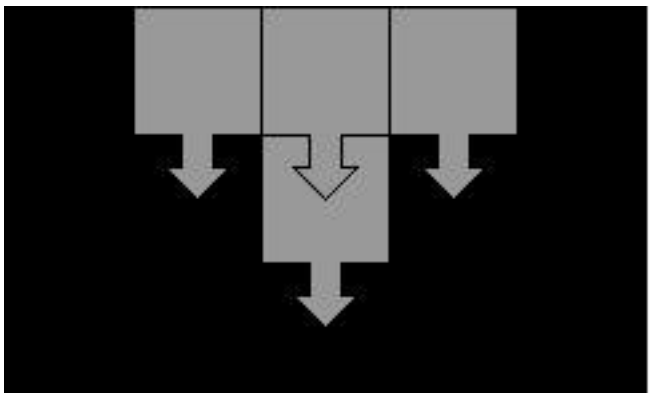
```
//precondition:  Blocks are in the grid.
//postcondition: Returns old locations of blocks;
//               blocks have been removed from grid.
private Location[] removeBlocks()


//postcondition: Returns true if each of locs is
//               valid (on the board) AND empty in
//               grid; false otherwise.
private boolean areEmpty(BoundedGrid grid, Location[] locs)
```

(We won't be able to test this code just yet.)

Exercise 2.2: Lost in Translation

We'll now complete `Tetrad`'s translate method, which will shift the `Tetrad` over and down by the given amount, first making sure that the `Tetrad`'s potential new position is valid and empty. For example, suppose we have a T-shaped `Tetrad` in the top middle of an empty grid, and we wish to move it down by one row, as shown below.



Clearly, we ought to allow this move. But notice that one of the locations we're attempting to move this `Tetrad` into is *already occupied by another block in the Tetrad itself.* So, it's not enough to make sure that the new locations are empty.

To make sure we handle this situation correctly, we'll always translate a `Tetrad` as follows:

1. Ask any block for its grid, and store it in a temporary variable.
2. Remove the blocks in the `Tetrad` (temporarily storing the old locations).
3. Create an array of new (possible) locations.
3. Check if the new locations are empty (and valid).
4. If the new locations are empty, add the `Tetrad` to the new locations, and return true. Otherwise, add the `Tetrad` back to its original locations, and return false.

Go ahead and complete the `Tetrad` class's `translate` method, making use of the helper methods `addToLocations`, `removeBlocks`, and `areEmpty`.

```
//postcondition: Attempts to move this tetrad deltaRow
//               rows down and deltaCol columns to the
//               right, if those positions are valid
//               and empty; returns true if successful
//               and false otherwise.
public boolean translate(int deltaRow, int deltaCol)
```

Now temporarily modify the `Tetris` constructor to translate the active tetrad, and make sure the tetrad appears in the correct location. Be sure to test a translation to an illegal position. (Don't forget to remove your temporary modification to the constructor when you are done!)

Exercise 2.3: Block Party

Now that blocks have the potential to move, let's teach them to dance! The `BlockDisplay` class keeps track of an `ArrowListener`. Whenever the `BlockDisplay` window has "focus" and an arrow key is pressed, a message is sent to the `ArrowListener`. The `BlockDisplay` class doesn't actually care what the `ArrowListener` chooses to do with this message, and hence `ArrowListener` has been defined as an interface, shown here.

```
public interface ArrowListener
{
     void upPressed();
     void downPressed();
     void leftPressed();
     void rightPressed();
     void spacePressed();
}
```

Modify the `Tetris` class so that it implements the `ArrowListener` interface. The first four `ArrowListener` methods should cause the `Tetris` game's active tetrad to move one row or column in the indicated direction (including up, for now). The `spacePressed` method should cause the active tetrad to fall all the way to the bottom of the grid. Be sure to call the display's `showBlocks` method to tell it to redraw itself whenever your tetrad moves.

In the `spacePressed` method, you can cause the active tetrad to fall to the bottom of the grid by using the `boolean` return value of the `translate` method. Just continue to call the

`translate` method (to move the active tetrad downwards) while the method returns `true` (meaning it is still capable of moving downwards).

In the `Tetris` class's constructor, when you create the `BlockDisplay`, call the `BlockDisplay` object's `setArrowListener` method, to tell the display that your `Tetris` object itself would like to be notified whenever an arrow key is pressed. The key word `this` will tell the display to call the methods you just implemented.

Now go ahead and test your code. A random tetrad should appear at the top of your `Tetris` window. You should be able to move it around with the arrow keys. Your program should prevent you from moving the tetrad outside of the window. When you are finished, demonstrate your work for me so that I will know you have completed this part of the assignment.