CS 301: Assignment – Shape-Matching

This will be your first Android app assignment, but you will not need to know much about the Android Java API because the entire GUI is supplied to you as part of the starter project. The focus of the assignment is that of pattern-matching of two-dimensional arrays.

Note: if you do the basic assignment, and it works properly and is reasonably well commented, you can expect to receive a grade of 'B'. If you want a better grade, you should implement at least some of the functionality listed under "Additional functionality".

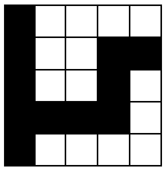
Overview

You are given two square (2D) boolean arrays:

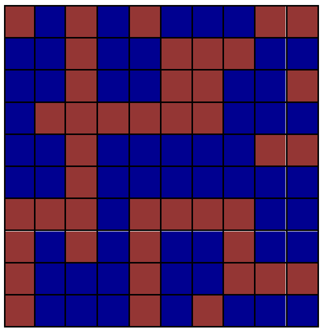
- A "shape" array, in true values in the array are part of the shape, and *false* values are not.
- A "world" array, in which *true* values represent filled spaces, and *false* values represent empty spaces.

You goal is to implement a *check()* method (in the `MyShapeSolver` class) that finds a location in the array where the "shape" fits into the "world".

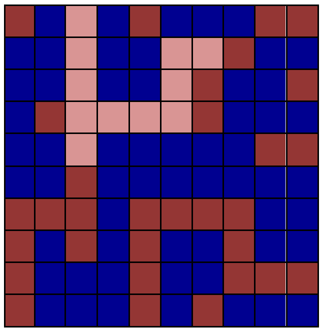
For example, consider the following shape-array (where black is *true* and white is *false*):



And the following world-array, where dark (blue) indicates *true*, and medium (red) indicates *false*:



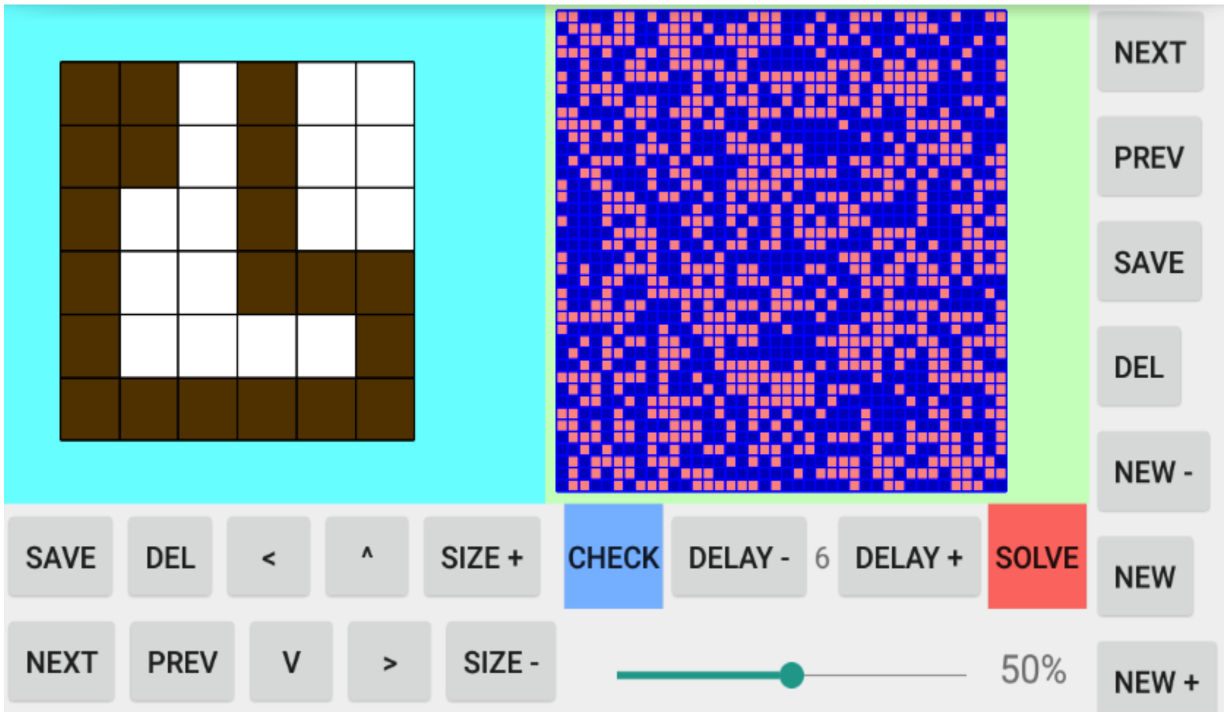
In this case, your method could report a solution at row 0, column 2, as illustrated by the light (red) squares here:



If there is more than one solution, your method can report *any* correct solution. If there is no solution, it should report that as well.

GUI

The graphical user interface looks something like this:



The large pane on the left displays the *shape*; the pane immediately to its right displays the *world*.

It allows the user to:

- Run your *solve()* method ('SOLVE' button) using the current shape and the current world.
- Run your *check()* method ('CHECK' button—more on that later) using the current shape.
- Create/modify shapes and worlds by touching/dragging on squares to toggle them.
  - You can also pan and magnify these views using two-finger drags.

    - You can also create random worlds using the 'NEW', 'NEW-' and 'NEW+' buttons.
        - When a random world is created, the probability of any given square being *true* is controlled by the slider-bar.
        - When 'NEW' is pressed, the random world created is the same size as the current one.
        - When 'NEW-' (or 'NEW+') is pressed, the random world created is one smaller (or larger) in each dimension than the current one.
    - You can change the size of the square in the *shape* view using the 'SIZE-' and 'SIZE+' buttons.
    - You can move the shape within the *shape* view using the '<' (left), '>' (right), 'V' (down) and '^' (up) buttons.
- Save the current shape or world, so that it is available during subsequent runs of the program ('SAVE' button).
- Cycle through the set of existing saved shapes or worlds ('NEXT' or 'PREV' buttons).
- Delete the current saved shape or world ('DEL' button).
- The programmer has the option of displaying each location that is attempted during the running of the *solve()* method. If this is done, the 'DELAY-' and 'DELAY+' buttons allow the user to control the amount of artificial delay that is injected when an attempt is displayed. This has the effect of speeding up or slowing down the animation of the attempts as they are tried. The amount of delay (in milliseconds) is displayed between these two buttons.

The *solve()* Method

You need to modify the *solve()* method in the `MyShapeSolver` class so that it searches for a place where the shape fits into the world. You should test the shape not only in its given orientation, but also consider all 90-degree rotations, including mirror images:

- The shape-array will be given in the inherited instance variable named *shape*.
- The world-array will be given in the inherited instance variable named *world*.
- When you find a solution, you should call the *display()* method as follows:

  ```
  display(topRowPosition, leftColumnPosition, orientation);
  ```

  where

  - `topRowPosition` is the row position in the world array that superimposes on (0, 0) in the shape-array.
  - `leftColPosition` is the column position in the world array that superimposes on (0, 0) in the shape-array.
  - `orientation` is a value that specifies the orientation of the shape (see below)

If no solution is found, the method:

```
undisplay();
```

should be called.

Once a solution has been found, the *solve()* method should return immediately after calling the *display()* method.

If you want to view each "try" as it occurs (even the unsuccessful ones), you can make a call to *display()* after each position/orientation. Your solution should, ensure that the last call to *display()* is with a correct solution—or that the final call is to *undisplay()* if there is no correct solution.

Orientation

As was mentioned above, you also need to consider 90-degree rotations and reflections. An `Orientation` class has been provided that defines eight objects, one for each possible orientation:

- `Orientation.ROTATE_NONE`: no rotation
- `Orientation.ROTATE_CLOCKWISE`: 90-degree clockwise rotation
- `Orientation.ROTATE_180`:180-degree rotation
- `Orientation.ROTATE_COUNTERCLOCKWISE`: 90 degree counterclockwise rotation
- `Orientation.ROTATE_NONE_REV`: left-right reflection
- `Orientation.ROTATE_CLOCKWISE_REV`: 90-degree clockwise rotation; then left-right reflection
- `Orientation.ROTATE_180_REV`: 180-degree rotation; then left-right reflection
- `Orientation.ROTATE_COUNTERCLOCKWISE_REV`: 90 degree counterclockwise rotation; then left-right reflection

If you find, for example a "fit" corresponding to a 90-degree counter-clockwise rotation at row 4, column 8, you could make the call:

```
display(4, 8, Orientation.ROTATE_COUNTERCLOCKWISE);
```

The starter project gives an example call to display(). It does not even exam the arrays, but simply reports a "fit" at a constant location and orientation.

You might be interested to know that there is a convenient way to iterate through all eight of the orientations:

```
for (Orientation or : Orientation.values()) {
   ...
   display(rowPos, colPos, or);
}
```

(You will learn this officially when we study enum-classes later in the term.)

Additional Functionality

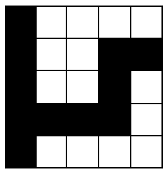
As was already mentioned, if you do the basic assignment and it works and is reasonably well commented, you can expect a 'B' on the assignment. If you wish to improve your grade, consider doing one or more of the following:

- Optimize your *solve()* method in the following sense [+0.75 grade]:

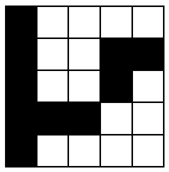
  > If a rotation or a reflection of a shape is identical to the shape itself, do not test the redundant version of the shape. For example, if the shape is a square, no orientations other than the unrotated one should be tested, since a square is identical to itself no matter how it is rotated (in the 90-degree sense) or reflected; a vertical line would require that only two orientations be tested; a "T" shape would require only four orientations to be tested.

  If you do this enhancement, make sure to call *display()* on each position/orientation that you test, so that the grader can see which once you are avoiding.

- Implement the *check()* method. The check method should test whether the shape is completely connected. In the sense that it should be possible to reach every location on the shape by moving up/down/left/right without leaving the shape. For example, the following shape is connected [+1.0 grade]:



while the following one is *not* connected:



An empty shape should be considered to be *not* connected.

The shape-array is available as the inherited instance variable, *shape*. The *check()* method should return *true* if the shape is connected, and *false* if not. You can test your *check()* method as follows:

- Press the ‘CHECK’ button.
- If it returns *true*, the button turns (or stays) *blue*; if it returns *false*, the button turns (or stays) *red*.
- (The check() method in the starter file randomly returns *true* or *false*, each with 50% probability.)

If you do both of the above enhancements successfully, it is possible to receive a grade that is above 100%.

Getting Started

https://classroom.github.com/a/OS2cvn4S

Your starter code is in the form of an Android Studio project. Visiting the link above and follow the instructions will cause a private *GitHub* repository to be created for you. You can then create a local Android Studio project that is connected to the repository. I would recommend you use GitHub Desktop to clone the repository, then open your local files using Android Studio.

Notes

If you have any questions about this assignment, please ask your instructor.

Comment your program. Make sure that the comment at the beginning of `MyShapeSolver.java` includes your name, etc. You should also keep the comments in your code consistent with your modifications to the program's behavior. The difference in grade between commenting your program well and commenting it poorly can easily be a full letter grade.

Remember to keep a log of problems that you encounter, including the resources you used to solve the problem. If you do not keep a log, you can expect your grade to be lowered by about two grades. If you did not use any outside help at all, your log file should contain a note that states that you did not use any outside resources. Keep your log in the (already created) file `log.txt`; you can find it in `res/raw` in the project.

Make sure to back up your work on a regular basis by doing *commit* and *push* operations.

When you call the *display()* method, you should report the row/column position in the world that corresponds to the top-left element the shape-array. If this element happens to be false, the position you report will be a position that is not actually part of the shape.

The only project file that you should need to modify is `MyShapeSolver.java`. You should not even need to look at the other Java files. Feel free, however, to examine them if you are curious.

Handing it in

You do not need to do anything to hand this project in, other than to *commit* and *push* your final version before the deadline. *If you are not finished with the project when it is due, email your instructor before the deadline. Otherwise, you run the risk of having your project graded before you have finished it.* Again, make sure to commit and push.