CS 273 Laboratory 7: Arrays II

This laboratory will give you some practice with two-dimensional arrays. We will be implementing the logic for capturing pieces in the game of "Go".

Preliminaries

If you are not familiar with the game of "Go" read the following description:

- Go is played on a square board with a square grid.
- One player plays black stones; the other plays white. Players take turns placing a stone of their color on one of the grid intersections.
- A player may "capture" the pieces of the other player. A group of adjacent pieces of one color is captured and removed from the board when they are completely surrounded by pieces of the other color. Another way of thinking of this, for programming purposes, is that none of the pieces in the group is adjacent to a "hole" (i.e., an empty space). If any piece in the group has even one "hole" next to it, then the group is not captured.
- The word "adjacent" for the purposes of this game means to be "immediately next to" in one of the **four** directions, up, down, left or right. The diagonal directions are not considered.

Review

Compile and run `GoFrame.java` in the `lab7` project that you created last time. At the present time, the project causes a frame to be displayed that contains four buttons and a Go board that by default is 19x19. The only buttons that are implemented is the "Spiral" button, and the "Clear" button that you implemented in the last lab. You should also be able to click near an intersection and see its contents toggle between white, black and blank, which you also implemented in the last lab.

Review the code you wrote.

The variable `board` is a two-dimensional, rectangular array that represents our board in the game of Go. An element of the array typically contains one of the values `WHITE`, `BLACK` or `EMPTY`, depending on whether or not the square is occupied. These names are defined as symbolic constants. The actual values are small integer constants such as 0, 1 and 2, but it is good programming practice to refer to them by their symbolic names: `WHITE`, `BLACK` and `EMPTY`. Whenever one of these squares changes in the array, it is reflected on the display the next time that the `board` is repainted. Your task is to modify and extend `GoFrame.java` so that it performs each of the tasks described below.

Laboratory

Part 1: Fill the board with randomly-placed stones

Modify the `randomizeBoard` code so that it sets each space "randomly" as follows: approximately 20% empty, 40% black, 40% white. (This code will be run whenever the `Random` button is pressed.)

checkpoint 1 (15 points): Demonstrate to your instructor or lab assistant that pressing the Random button causes stones to be placed on the board as specified above.

Overview of Parts 2-7: Capturing stones

The rest of this lab involves the `removeCapturedStones` operation, whose purpose is to remove any stones that should be captured according to the rules of Go. This code is run whenever the `Capture` button is pressed.

The logic for capturing stones is more complex than for the previous tasks; a stone should be captured and removed from the board EXCEPT when:

- it is adjacent to an empty space, or
- it is adjacent to a stone of the same color that is adjacent to an empty space, or
- it is adjacent to a stone of the same color that is adjacent to a stone of the same color that is … that is adjacent to an empty space.

We will therefore use two temporary "colors", BLACKINPERIL and WHITEINPERIL, which will help us figure out which stones to remove. These will display respectively as red and pink on the board. The idea of BLACKINPERIL and WHITEINPERIL is that these are stones that should be captured. However, we won't actually remove them until the “Capture” operation is completely implemented and we know that they have no path to an empty space

Capturing will be performed as follows:

- Initially set all BLACK stones to BLACKINPERIL and all WHITE stones to WHITEINPERIL.
- Color each INPERIL stone that is in a "safe group" back to its original color. (A stone is in a "safe group" if it is adjacent to an empty space or to a stone of its color that is in a "safe group".)
- Remove all remaining INPERIL stones from the board.

Thus after the entire capture has taken place, all spaces will be BLACK, WHITE or EMPTY. When the capture operation is completely implemented, the user will never see the INPERIL stones.

Part 2: Capturing stones, step 1

Implement the first part of the Capture operation for white stones. First, change all the WHITE stones on the board to be WHITEINPERIL. Again, WHITEINPERIL means that it's a white stone, but we don't yet know if it is in a "safe" group or a "captured" group.

Do this much; then test your code. The board will display the WHITEINPERIL stones as pink, so that you can see that you've done this much correctly.

checkpoint 2 (10 points): Demonstrate to your instructor or lab assistant that pressing the Capture button causes all white stones to turn pink.

Part 3: Capturing stones, step 2

The next step is to identify the white stones that are part of safe groups, and to change their status back to WHITE. The intuition is that a stone is safe if it's next to an EMPTY space. Furthermore, if any stone that is part of a group is safe, then all the stones in that group are safe. This means that a WHITEINPERIL that is next to a stone that is WHITE (and therefore already found to be safe) is also safe, so we should change it from WHITEINPERIL to WHITE.

So, after the code you just wrote, do this as follows, looping through all locations on the board:

- If the stone is WHITEINPERIL, then see whether any of the adjacent locations are EMPTY or contain a WHITE (safe) stone. If so, change the WHITEINPERIL stone to WHITE.
- **Be careful** not perform any out-of-bounds array accesses when checking stones on the edges of the board. Otherwise you'll see that big black 'X'.
- Otherwise, leave it alone.

Again, stop and test your code to make sure that one pass through the board works correctly. This should cause all WHITEINPERIL stones that are next to an empty space to change to WHITE, and may change other WHITEINPERIL stones that are nearby. It will likely not change a WHITEINPERIL stone that is "far away" from an empty space. That will be implemented in a later checkpoint.

checkpoint 3 (25 points): Demonstrate to your instructor or lab assistant that pressing the Capture button causes only white stones that are "far away" from an empty space to turn pink.

Part 4: Capturing stones, step 3

If any stones were changed from WHITEINPERIL to WHITE, then we need to check the entire board again to see if stones next to the newly-WHITE stone need to be changed. So "wrap" your WHITEINPERIL-to-WHITE nested loop in an additional loop that repeats the process three times.

Stop and test your code.

When you've looped through the entire board three times, then most safe stones that had been WHITEINPERIL should have been changed back to WHITE. The only ones that are not WHITE are those whose only path from a blank square contains more than three up/left length moves.

checkpoint 4 (10 points): Demonstrate to your instructor or lab assistant that pressing the Capture button causes all of the capture-able white stones (and some of the non-capture-able ones) to turn pink.

Part 5: Capturing stones, step 4

Replace the for-loop in the above step with a while-loop that keeps iterating until an entire pass has been made through the board without any stone being changed. To do this:

- Declare a boolean variable before your outermost loop that you will use to indicate whether a change has been seen in the previous pass through the board.
- The while-loop condition should cause the loop to continue if a change has been seen on the previous pass (i.e., if the boolean variable's value is true).
- Initialize the boolean variable to true at the time it is declared so that the loop body is executed the first time through.
- As soon as you enter the while-loop, set the value of the boolean variable to false. This will indicate that on the current pass over the board, no change has been seen. (This is trivially true because no square on the board has yet been examined--the pass has not yet started.)
- In the body of the loop, any time a stone is set from WHITEINPERIL to WHITE, set the boolean variable's value to be true.

Stop and test your code.

When you've looped through the entire board until no changes have been detected, then all safe stones that had been PINK should have been changed back to WHITE.

checkpoint 5 (15 points): Demonstrate to your instructor or lab assistant that pressing the Capture button causes only the capture-able white stones to turn pink.

If you have *fully* completed all the above checkpoints, you now have a grade of C (75) for this lab.

Part 6: Capturing stones, step 5

After the loop for the previous checkpoint, loop through the entire board again, removing any remaining WHITEINPERIL stones (i.e., change them to EMPTY).

checkpoint 6 (10 points): Demonstrate to your instructor or lab assistant that pressing the Capture button causes (only) the capture-able white stones to be removed from the board.

If you have *fully* completed all the above checkpoints, you now have a grade of B (85) for this lab.

Part 7: Capturing stones, step 6

Implement steps 1 through 5 (above) for black stones. In other words, change the places in the code that deal with WHITE and WHITEINPERIL stones to do the same for BLACK and BLACKINPERIL stones.

checkpoint 7 (15 points): Demonstrate to your instructor or lab assistant that pressing the Capture button causes (only) the capture-able black and white stones to be removed from the board.

Part 8: Finish up

Log off. You're done.