CSD2341 Computer Graphics Programming

Workshop 3:

Assessment: 4 Marks. Due in the next workshop.
For on-campus students: demonstrate in the next workshop session.
For off-campus students only: zip your project directory and submit to blackboard.

Related Objectives from the Unit outline:

- To understand the programming techniques and algorithms used for developing 2D graphics software.

Learning Outcomes:

In this workshop, you will implement several area filling algorithms.

For this workshop, you will be using the `rasterDemo` package again, this time to implement several polygon fill algorithms. A Java application is provided which implements a *4-connected* flood fill. Your task is to add an *8-connected* flood fill and a boundary fill.

1 Get Started

- Create a folder for this workshop somewhere, say on the Desktop.
- Go to BlackBoard, locate and download the zip file AreaFill.zip to the folder.
- Extract the files e.g. by right-clicking on the zip file and choosing "Extract here".

This folder contains the source files for this workshop.

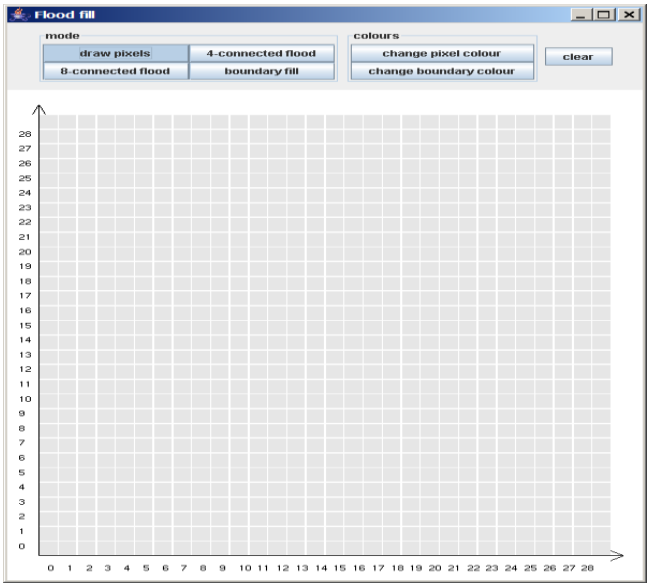
- Start NetBeans.
- Using the "File" menu, create a new project, but this time, select "Java Project with Existing Sources"
- When the "Java Project with Existing Sources" dialog comes up, choose a name for the project, e.g. "Area Fills", and use the "Browse" button to choose your workshop folder as the project folder.
- Next, using "Add Folder", select the extracted folder "AreaFills" as the source package folder.
- Next, you must add in the two jar files jogl.jar and SimpleGL.jar as libraries --- see last week's workshop for instructions on how to do this.
- Now you should be able to build the project – Right click on the project in the "Project" view, and select "Build Project".

The package `Edge` contains classes that handle information about edges of polygons. The package `rasterDemo` is the same as the one used for the line drawing workshop last week. The class `AreaFill` uses the `rasterDemo` package to illustrate a scan conversion algorithm for polygons. This algorithm was

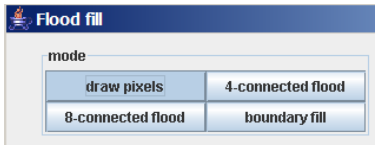
discussed in lectures. The class `FloodFill` uses the `rasterDemo` package to illustrate flood fill and boundary fill algorithms. It is partly completed – **your job is** to finish the implementation.

- Run `FloodFill.java`.

You will see a display like this:

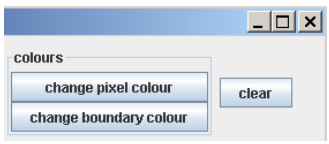


There are 4 drawing modes available:



- **draw pixels** – in this mode, hold down the mouse and drag it around to "paint" pixels in the current colour;
- **4-connected flood fill** – in this mode, click anywhere to flood fill an area starting at the pixel that you clicked on, using the current colour (i.e. all pixels that are 4-connected to the starting pixel by pixels of the same colour will be repainted);
- **8-connected flood fill** – same thing using 8-connection. This *is not implemented* yet;
- **boundary fill** – click anywhere to do a boundary fill, starting at the clicked pixel, and continuing to spread out until a pixel of the boundary colour is reached. This is also *not yet implemented*.

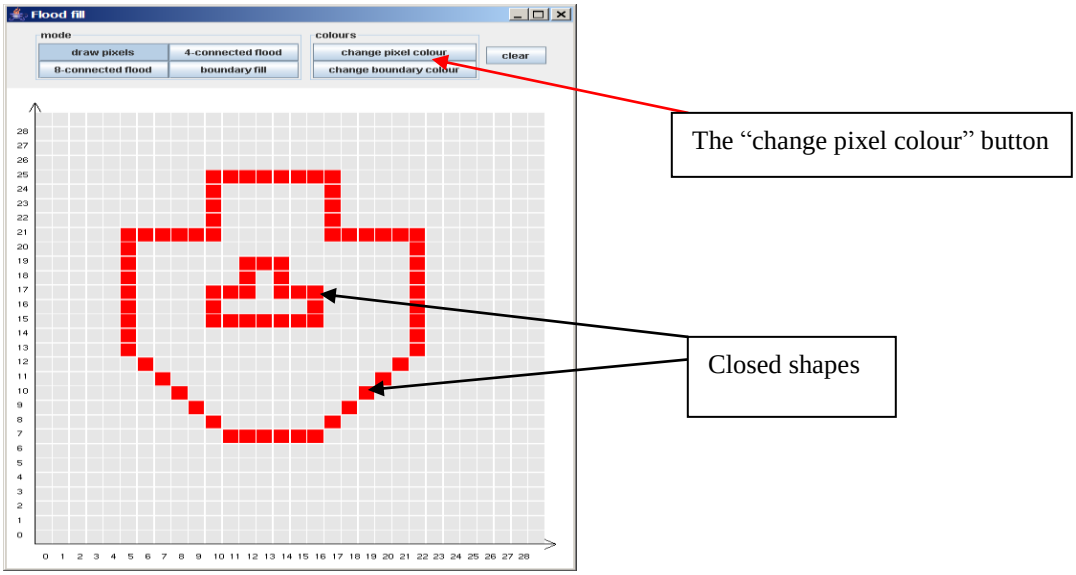
There are two colours that can be set by clicking on "**change pixel colour**" and "**change boundary colour**". Lastly, the raster can be cleared by clicking on the "**clear**" button.



Look and Feel

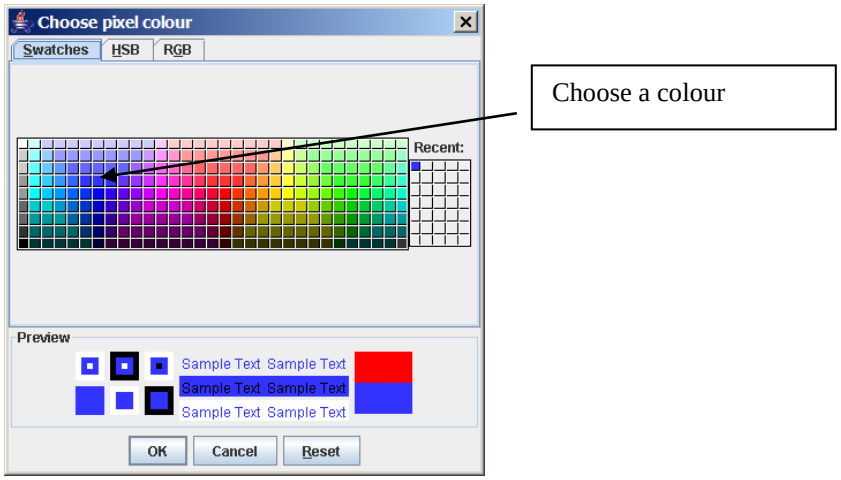
1. Try out the following:

    a. Use the mouse to click-and-drag on the "raster" and draw some **closed** shapes similar to these shown below.

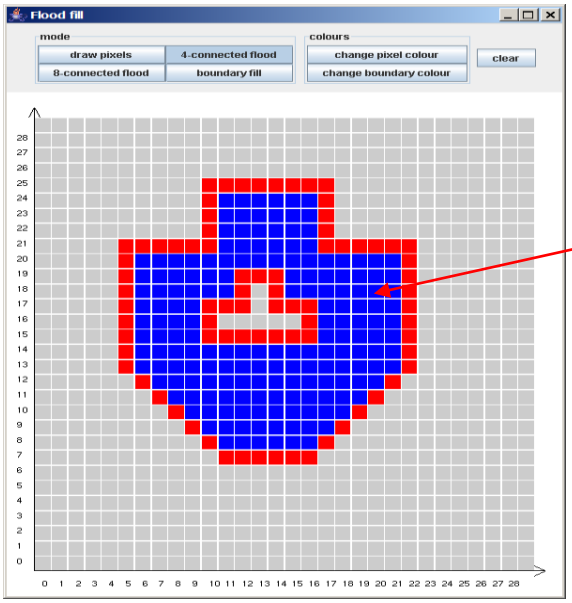


    b. After that, click on the "**4-connected flood fill**" button.

    c. And then click on the "**change pixel colour**" button and select a different colour, for example a blue colour. Click **OK.**



    d. Now *click inside* your closed shape. The interior of the shape should be filled. The algorithm has been slowed down so that you can see the process of the pixels being changed.



FloodFill Algorithm

2. Now examine the code for **FloodFill.java**. With time and effort, you should be able to follow it, but for the moment, focus on the method `flood4()`.

   Compare it with the code on page 205 of Hearn and Baker. There are some differences, but the logic is the same. A few things that are different:

   a. Instead of interiorColor, we use startColour
   b. Instead of fillColor, we use an attribute called color, and this is not a parameter of the method
   c. Colours are Java Color objects, instead of int's
   d. getPixel doesn't need the color parameter, instead it returns the Color of the pixel
   e. Each recursive call is protected by a test like "if(col > 0)". This is to make the flood stop at the boundary of the raster.

   Here is the code for this method:

```
/**
 * Here is the flood fill routine. Performs a flood fill using the
 * currently selected colour.
 *
 * This method should be called in a separate thread
 *
 * @param col - the column number for the start pixel
 * @param row - the row number for the start pixel
 * @param startColour - the seed colour
 */
private void flood4(int col, int row, Color startColour)
{
   // here's the algorithm from Hearn and Baker, page 205

   if(startColour.equals(getPixel(col, row))) // stop if the pixel is the wrong colour
   {
      setPixel(col, row);
```

```
            // this bit makes the animation work by
            // drawing intermediate results, and slowing the updates down

            draw();
            try
            {
               Thread.sleep(10); // creates a small delay
            }
            catch(InterruptedException e){}

            // now recursively call ourselves for each neighbour

            if(col > 0)
            {
               flood4(col-1, row, startColour); // go left
            }
            if(col < COLS-1)
            {
               flood4(col+1, row, startColour); // go right
            }
            if(row > 0)
            {
               flood4(col, row-1, startColour); // go up
            }
            if(row < ROWS-1)
            {
               flood4(col, row+1, startColour); // go down
            }
         }
      }
```

Note the calls to the method `getPixel(int col, int row)`, which returns a **java.awt.Color** (i.e. returns the colour of the pixel in this column and row of the raster), and `setPixel(int col, int row)`, which sets the colour of this pixel to be the current colour.

Exercises

Now we come to your bit. You will see that there are two methods, `flood8()` and `boundary()` with no bodies. That's because you **have to provide them!**

Task 1 (2 Marks)

Implement `flood8()`. This should be pretty similar to `flood4()` – you just need to include the cases for the other 4 neighbouring pixels. Remember to include the checks to make sure you stay within the raster.

Task 2 (2 Marks)

Implement `boundary()`. This can be based on the code in Hearn and Baker.

Task 3 (0 Marks)

Answer the **Tutorial Questions** over the page.

Solutions *will be posted* on **BlackBoard** for both the programming task and the tutorial questions.

Tutorial questions on Area Fills

1. For this question, use the pseudo-code given in lectures, rather than the algorithm from the textbook (Suggestion: use the AreaFill program from the workshop to check your answer).

   Show the **sorted edge table** for the polygon composed of the lines connecting the points (1, 2), (2, 6), (7, 7), (8, 1), (6, 5) and (4, 3).

2. Given a polygon defined by the vertices (4, 3), (5, 8), (7, 10), (8, 3), and two points A = (2, 4) and B = (6, 4). Use the nonzero winding method and only dot product or cross product calculations to find the winding number for A and B and classify each as either an exterior or an interior point. Show your working.