CSCI 135 Programming Exam #1
Fundamentals of Computer Science I
Fall 2013

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #1 dropbox. Please ***double check you have submitted all the required files***.

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. You will lose a substantial number of points if your code does not compile or crashes on typical inputs.

Overview. Wildfires are burning all across Montana and the forest service needs your help! You need to develop a program that can schedule missions for its fleet of tanker planes. To help plan tanker missions, the forest service has obtained infrared satellite photos of areas of interest. Here is an example image:

The white blobs represent currently burning wildfires. The whiter the color, the more intense the heat at that location. Any pixel that is completely black with a red-green-blue (RGB) value of (0, 0, 0) is considered to be not on fire. All other non-black pixels are considered to be on fire. You will be using images such as the one above to decide where the tankers should drop water. We adopt a simplified model of firefighting based on the pixels in the image:

- Tankers always fly from north-to-south in a straight vertical line.
- Tankers have unlimited water and can completely extinguish all areas on fire below them.
- Tankers extinguish fires within a given width of their north-to-south flight path.

You will be first implement `FireUtil.java` which contains a library of static methods useful for firefighting-related things given a satellite image. Once you have your library of handy methods, you will implement `FireMissions.java` which actually determines where to send each tanker mission.

To get started, create an empty Eclipse project and extract the contents of this zip file into your project directory: [fire.zip]

Part 1: FireUtil. This class provides a library of static methods, here is the API you are to implement:

```
public class FireUtil
```

```
    long getTotalPixels(Picture pic)                // Get the total pixels in Picture pic
 boolean isOnFire(Picture pic, int x, int y)        // See if the pixel at (x,y) is on fire
     int getNumOnFireAtX(Picture pic, int x)        // Count fire pixels at vertical slice = x
  double getPercentOnFire(Picture pic)              // Calc. % of total pixels that are fire
    void dropWaterAtX(Picture pic, int x)           // Turn pixels black at vertical slice = x
   void dropWaterAroundX(Picture pic, int x, int w) // Turn pixels black in [x-w, x+w]
```

We have provided a stub version of `FireUtil.java` that includes more extensive comments describing what each method should do. In particular, your methods need to be *defensive*; they should avoid crashing if the client asks for silly things (e.g. specifying out-of-range x- or y-coordinate). The comment at the top of each method declaration describes exactly what to return if a client asks for something nonsensical.

Note: You may want to develop your methods in the order listed above. This will help you avoid repeated code; the earlier methods are often quite useful for implementing the latter methods.

Your program will be making use of the provided `Picture.java` class. Here is the relevant API:

```
public class Picture
```

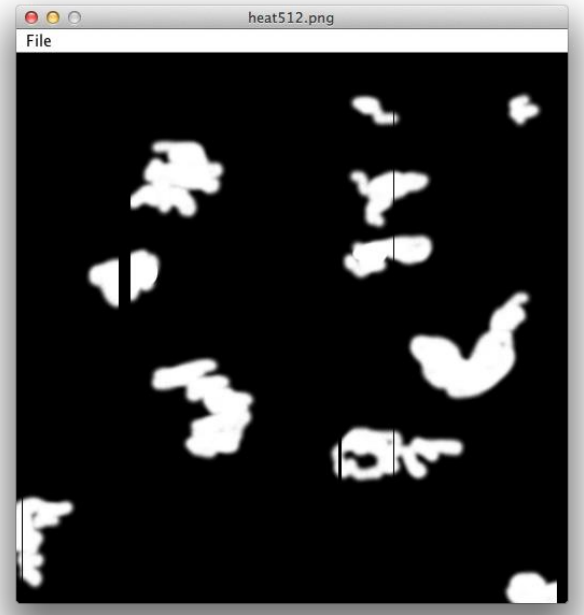
```
      Picture(String filename)   // Load a picture from the specified image filename
Color get(int i, int j)          // Get color of pixel at (i, j), note (0,0) is upper-left
 void set(int i, int j, Color c) // Set color of pixel (i, j) to c, note (0,0) is upper-left
 void save(String name)          // Save the picture to a file in a standard image format
  int width()                    // Return the width of the picture in pixels
  int height()                   // Return the height of the picture in pixels
 void show()                     // Display the picture in a window (useful for debugging)
```

Note: `Picture`'s `get` and `set` methods use `(i, j)` to specify the location of pixels. In terms of our `FireUtil` API, `i` is a pixel's x-coordinate and `j` is a pixel's y-coordinate. You will also need to make use of Java's built-in `Color` class to determine if a pixel is on fire or not. Here is the relevant API:

```
public class java.awt.Color
```

```
int getRed()    // Returns the red component in the range 0-255
int getGreen()  // Returns the green component in the range 0-255
int getBlue()   // Returns the blue component in the range 0-255
```

We have provided a `main` method that extensively tests the methods in `FireUtil`. The `main` method reads in the filename specified as the first command-line argument. It also opens a window showing the image after some firefighting has occurred. Here is an example run including the resulting image (the black streaks represent the tanker drops that were performed by my test `main`):




```
% java FireUtil heat512.png
getTotalPixels(pic)               = 262144

isPixelOnFire(pic,    10,    10) = false
isPixelOnFire(pic,   129,   145) = true
isPixelOnFire(pic,   135,   128) = true
isPixelOnFire(pic,  9999,   128) = false
isPixelOnFire(pic,    -1,   128) = false
isPixelOnFire(pic,    10,  9999) = false
isPixelOnFire(pic,    10,    -1) = false

getNumPixelsOnFireAtX(pic,    -1) = 0
getNumPixelsOnFireAtX(pic,     5) = 82
getNumPixelsOnFireAtX(pic,    58) = 0
getNumPixelsOnFireAtX(pic,  9999) = 0

getPercentOnFire(pic)             = 14.6320

dropWaterAtX(pic,    -1), getPercentOnFire(pic) = 14.6320
dropWaterAtX(pic,     5), getPercentOnFire(pic) = 14.6008
dropWaterAtX(pic,    58), getPercentOnFire(pic) = 14.6008
dropWaterAtX(pic,  9999), getPercentOnFire(pic) = 14.6008

dropWaterAroundX(pic, -9999,     5), getPercentOnFire(pic) = 14.6008
dropWaterAroundX(pic,   100,     5), getPercentOnFire(pic) = 14.3429
dropWaterAroundX(pic,   512,    10), getPercentOnFire(pic) = 14.2406
dropWaterAroundX(pic,   300,     1), getPercentOnFire(pic) = 14.1842
dropWaterAroundX(pic,   350,     0), getPercentOnFire(pic) = 14.1331
dropWaterAroundX(pic,   400,    -5), getPercentOnFire(pic) = 14.1331
dropWaterAroundX(pic,  9999,    50), getPercentOnFire(pic) = 14.1331
```

Part 2: FireMissions. This program decides how to execute a series of 0 or more tanker missions. `FireMissions.java` will make use of the static methods you created in `FireUtil.java`. The program takes 4 command-line arguments:

- ***image*** - Filename of the satellite image.
- ***max missions*** - How many tanker missions to simulate. If this is set to -1, the program continues to simulate missions until all fires in the given image have been extinguished.
- ***width extinguished*** - A non-negative integer specifying how wide a swath the tanker extinguishes. A width of 0 indicates to put out fire only directly on the flight path (pixels at `x`). A width of 1 would put out fire on the flight path as well as pixels one to the left and right (pixels at `x-1`, `x`, and `x+1`), and so on.
- ***out image*** - Output filename for the image resulting from all the executed missions.

If fewer than 4 command line arguments are passed to it, your program should print the message shown in this example run:

```
% java FireMissions
FireMissions <image> <max missions, -1=no limit> <width extinguished> <out image>
```

If 4 command-line arguments are given, your program should load the specified image and first print out the percent of fire in the image (rounded to two decimal places). The program then simulates 0 or more tanker missions. Your scheduling algorithm uses a simple *greedy* strategy:

- The next flight should be centered at the x-coordinate with the highest number of fire pixels.
- *Note:* For purposes of finding the best x-coordinate, we ignore the width of the tanker's spray. We simply find the vertical pixel slice through the image that currently has the most fire pixels. This is suboptimal but simpler.
- In the event of a tie (multiple x-values with the same number of fire pixels), use the smallest x-value.
- Missions continue until either all fire has been extinguished or the specified number of missions have been reached.

Each mission should print out the mission number, the number of fire pixels at the peak location, and the x-coordinate of the peak. Your program should then drop water at this x-coordinate using the width specified on the command-line. After each mission, it should print out the percentage of fire remaining (rounded to two decimal places). After all missions are complete, save the final image to the filename specified as the last command-line argument.

Here are a series of sample runs and the corresponding output image:

```
% java FireMissions heat512.png 0 10 out.png
At start, percent on fire: 14.63
```

% java FireMissions heat512.png 2 10 out.png
At start, percent on fire: 14.63

Mission #1
Peak of 180 fire pixels at x = 335
After drop, percent on fire: 13.26

Mission #2
Peak of 171 fire pixels at x = 324
After drop, percent on fire: 12.61

% java FireMissions heat512.png -1 100 out.png
At start, percent on fire: 14.63

Mission #1
Peak of 180 fire pixels at x = 335
After drop, percent on fire: 8.62

Mission #2
Peak of 164 fire pixels at x = 164
After drop, percent on fire: 3.23

Mission #3
Peak of 140 fire pixels at x = 463
After drop, percent on fire: 1.18

Mission #4
Peak of 92 fire pixels at x = 13
After drop, percent on fire: 0.00

% java FireMissions heat512.png 20 100 out.png
At start, percent on fire: 14.63

Mission #1
Peak of 180 fire pixels at x = 335
After drop, percent on fire: 8.62

Mission #2
Peak of 164 fire pixels at x = 164
After drop, percent on fire: 3.23

Mission #3
Peak of 140 fire pixels at x = 463
After drop, percent on fire: 1.18

Mission #4
Peak of 92 fire pixels at x = 13
After drop, percent on fire: 0.00

% java FireMissions black.png 10 10 out.png
At start, percent on fire: 0.00