**GeoPixel Counter**

**Julian Bertmaring**

**CS 470 –Project Write-up**

**April 29, 2010**

Table of Contents

[Abstract 3](#_Toc260321766)

[1. Introduction 4](#_Toc260321767)

[2. Project Overview 5](#_Toc260321768)

[2.1 Data Files 5](#_Toc260321769)

[3. Project Requirements 6](#_Toc260321770)

[3.1 Functional Specifications 6](#_Toc260321771)

[3.2 System Specifications 6](#_Toc260321772)

[4. System Design 7](#_Toc260321773)

[4.1 User Interface Design 7](#_Toc260321774)

[4.2 Data Structures 10](#_Toc260321775)

[4.3 System Architecture 10](#_Toc260321776)

[4.4 Algorithms 11](#_Toc260321777)

[5. Software Development Process 12](#_Toc260321778)

[5.1 User Interface challenge 12](#_Toc260321779)

[5.2 Prototyping and work breakdown 12](#_Toc260321780)

[6. Results 14](#_Toc260321781)

[6.1 Future plans/considerations 14](#_Toc260321782)

[7. Summary and Conclusions 15](#_Toc260321783)

[8. References 15](#_Toc260321784)

[**Appendix A: User Manual for GeoPixel Counter** 16](#_Toc260321785)

# Abstract

Rocks consist of different materials which can be separated by their color, texture and more. In Geology scientists are interested in specifying rock samples and finding out a specific breakdown of the different materials in a rock. For this they use a color index to specify rock samples and their breakdown. This project, GeoPixel Counter, was developed to open a picture of a rock and allow the user to calculate the breakdown of the composition of this rock based on the different colors of the grains. As the user chooses more colors the program will update the percentage breakdown.

# 1. Introduction

This project was developed for Dr. Kenrick Mock. Dr. Mock created a program in the past for Dr. Jeff Amato, who is a Professor of Geological Sciences at the New Mexico State University. This program was a GeoPixel counter, which helps the Geologists with determining specifics about rocks. In Geological Sciences the scientists often use a color index to determine the consistency and specifics of rock samples.

As I asked Dr. Jeff Amato about specific usage of the GeoPixel Counter he said:

“One particularly useful measurement is the amount (area in 2D, volume in 3D) of dark minerals (called "mafic" for the presence of MAgnesium and iron (ferric/ferros) elements in them), versus the lighter minerals.  The percentage of mafic minerals in these rocks is the "color index".  So a rock with 20% mafic minerals would have a color index (CI) of 20.  Higher CI corresponds to higher Fe and Mg concentrations.  This is also helpful in describing and identifying the rock.”

My project was to rewrite this existing application in Java and to modify it into a web applet so that Dr. Amato can use it on Apple computers and that he can easily distribute it to his students or co-workers.

Since Dr. Amato is in New Mexico and since there was an existing application which was developed by Dr. Mock, my interaction was mostly with Dr. Mock, but it was kept quite informal as my job was mostly clear.

# 2. Project Overview

The goal of this project is to develop a program that can determine the percentage of specific colors in an rock image based on user clicks. Colors in rocks are of particular interest in Geological Sciences due to a color index that rocks get specified by. This program will help Geological Scientist by providing an easier method to determine the percentage of different colors in a rock.

### 2.1 Data Files

The rock is presented as a picture of a slice of the rock which is loaded in from an existing image file. A sample picture of a rock slice can be seen in Figure 1.



Figure 1 A sample Rock slice

# 3. Project Requirements

The requirements for this project were set by the existing application. The finished product was supposed to have all the functionality of the original program and be able to run on Apple Computers as well as Microsoft based machines. I decided to use the prototyping method because of these requirements as well as the fact that Dr. Mock was available most of the time when I had a question about some part of the existing or new program. By using prototyping there were several risks involved:

* Working in a wrong direction and having to change a whole part of the program
* Changing requirements throughout the process
* Adding additional requirements while working on the project

These are just a few of the problems that might occur while using prototyping as the programming method. However the changing requirements could be excluded as the project requirements were given by an existing application.

### Functional Specifications

1. The system must have a load option to allow the user to specify which image to load.
2. Also an option to let the user chose a color for grain replacement is needed.
3. Furthermore it needs an option to let the user change the degree of color match.
4. The system must have an option to switch to “Select grain mode”.
5. There should be a field to enter the minerals name.
6. It must have a button to allow the user to add this mineral to a list.
7. And in the end there should be a table to display the minerals with their replacement color and percentage of the overall colors.

### System Specifications

The system must be constructed using Java. The resulting .jar file can run on any type of operating system which has Java support. Once the web-applet is implemented it can just run in any browser which has Java implemented. At a minimum and to be save, the system should have 512Mb of memory and run at a speed of 800 MHz or higher. The video adapter must be capable of a resolution of the picture the end-user wants to work on.

# 4. System Design

Due to the fact that this was mostly a User Interface design project in the first place this was an Event-Driven design with some features of other designs.

### 4.1 User Interface Design

The whole program lies within a JFrame which includes all the button controls and the image if one is loaded. A screenshot of this can be seen in Figure 2:

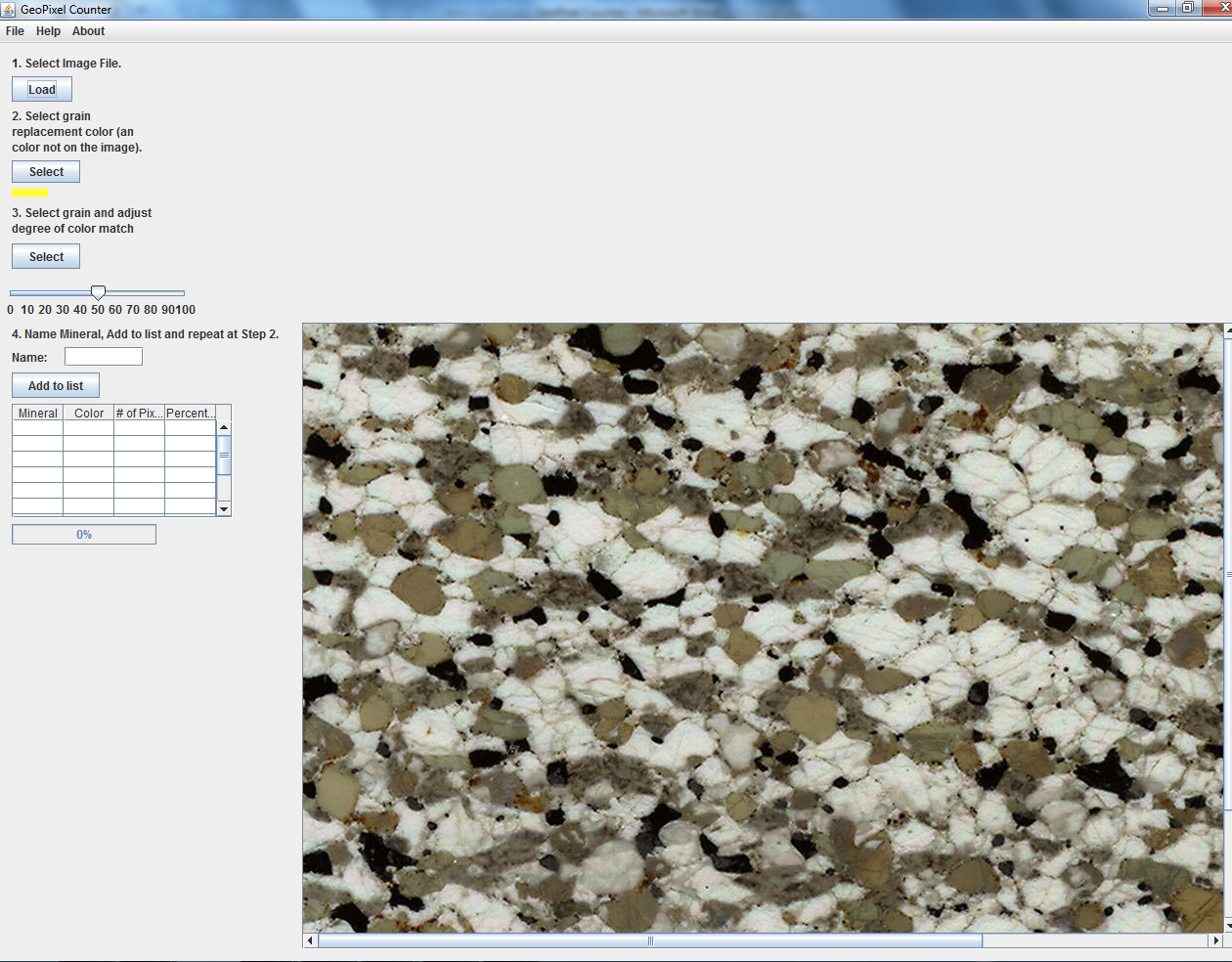


Figure 2 GeoPixel Counter UI

Under the menu options are:

* File, Exit : to exit the application with a pop-up dialog box which ensures that the user really wants to exit (can be used for saving later)
* Help, About: an option to be implemented in the future
* About, About: shows the credentials

The controls on the left of the screen are for:

* 1. Load button: allows the user to load a picture and checks for a valid format. The Load screen can be seen in Figure 3 and the message when a wrong file type got chosen can be seen in Figure 4.

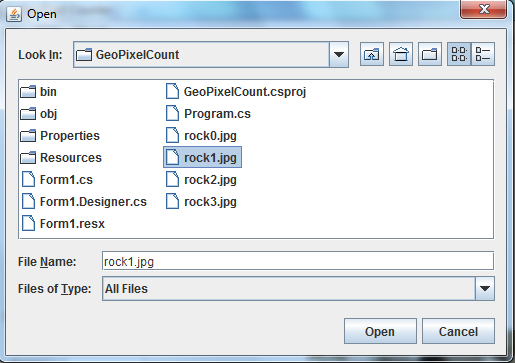


Figure 3 Load Screen

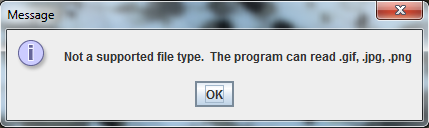


Figure 4 Error message: wrong file type

* 2. Select grain replacement color button: Allows the user to select a color for grain replacement. The Color Chooser can be seen in Figure 5

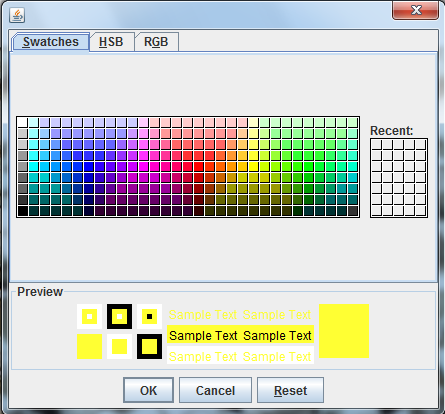


Figure 5 Color Chooser

* Color sample picture: A small stripe of the color that is currently active
* 3. Select grain and adjust degree of color match button: This button toggles the mode and lets the user chose a point on the picture which makes the program replace the chosen color on the picture with the color select under point 2. A sample of it can be seen on the updated UI in Figure 6

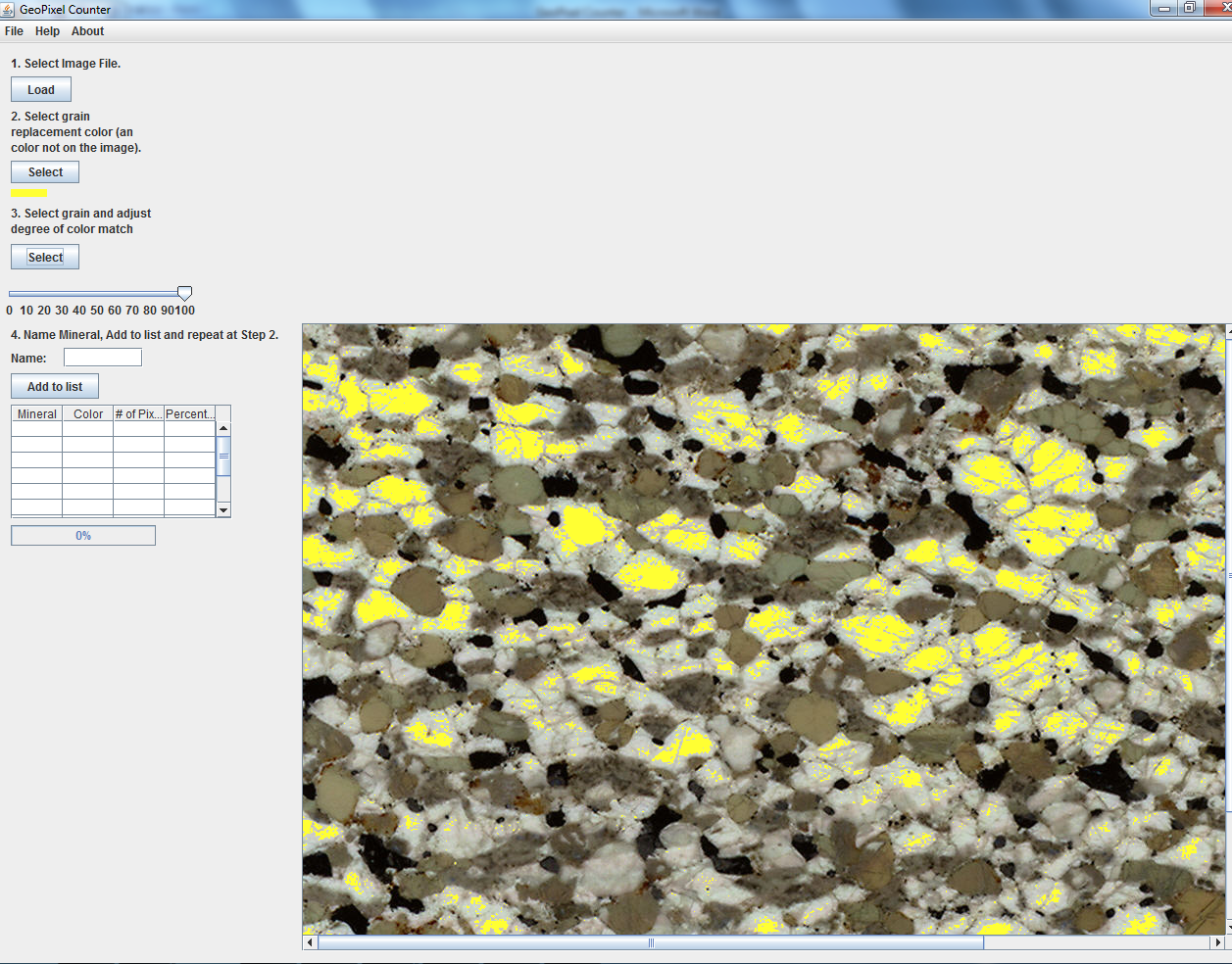


Figure 6 Updated UI with replaced grains

* Degree adjustment bar: A slider bar that lets the user adjust the degree of color match
* 4. Name mineral Textbox: Lets the user enter a mineral name. Figure 7 contains a screenshot of this textbox with text entered



Figure 7 Text entered in Name mineral Textbox

* Add to list button: This button adds the text entered in the “Name Mineral” Textbox to the list and calculates the percentage
* Table with statistics: This table is the part that keeps track of which Minerals are added to the list and shows the number of pixels and percentage of the Mineral. A screenshot can be seen in Figure 8 where 2 minerals were added

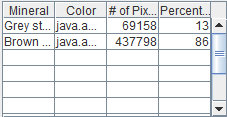


Figure 8 Table

* Progress bar: This progress bar is used when the processing takes too long and shows the progress

The image on the right is displayed in a JLabel field, which allows quick readjustments like positioning.

### 4.2 Data Structures

The data structures used in GeoPixel Counter are relatively simple as there is not too much data which requires storing. The processing on the image is done in memory for quick processing. The storage of which pixels were already processed is done in a two dimensional array of the type integer. Furthermore the colors which are suggested for usage to replace the grains are stored in an array of type Color and to keep track of the number of pixels and percentages another integer array got created.

### 4.3 System Architecture

The main system architecture consists of just one class which is Frame.java. However the way the methods are involved is similar to the way the user input works, which is shown in Figure 9. First the user loads an image which activates the Main Frame and enables the Select Grain function. At any given time the user can choose a different color which will switch the replacement color with the color the user selected.

When the user hits the “Select Grain” button the system switches modes and when he selects a grain the Main Frame gets updated with the new image which includes the updated color. The Layout and original image are given in by the Main Frame and the replacement color is given in by the color the user chose or the default color. After this the user can add a mineral to the list which will update the table. Only after the table got updated the color resets to the next default color and the user is able to select another grain on the image.

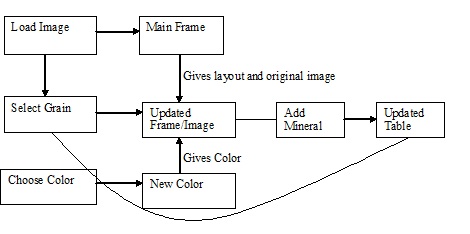


Figure 9 User input diagram

### 

### 4.4 Algorithms

In the GeoPixel Counter application the algorithms are fairly simple. Most algorithms are involved in the bookkeeping of the application; however there are some involved in the image tracking which aren’t. I try to keep the original image which gets displayed after it is loaded stored and do all the computations and modifications to a copy of this image. This requires not only bookkeeping algorithms but also some other modifications.

When the image is drawn and the user chooses a pixel from the image I have to get this pixel, then calculate all the pixels if they are in range of this pixels RGB value and then not just color them in the new color but also send them to an array which keeps track of which pixels already got colored. This has to be this way, because otherwise they might get drawn over next time the user chooses a new pixel.

To do this I created a BufferedImage of this original image.

# 5. Software Development Process

I decided to use the prototyping method, because there was an existing application in C# available, which I had to rewrite in Java. I created the UI for this application, then showed it to Dr. Mock, made some adjustments and then showed it to him again. After this I started on the mechanics of this program and handled it the same way.

This was the best choice for me as Dr. Mock was available most of the time and I could program loosely in the time I had available.

### 

### 5.1 User Interface challenge

One of the biggest challenges for me was creating the User Interface. Since this project was very User Interface heavy I took the first step by creating the User Interface before filling it with functionality. However this part was also hard, because I had to think ahead when creating certain items of the User Interface and see if it will be feasible to do it in one way or the other because of the functionality it might have later on.

The hardest part was to come up with the field that the picture will be presented in. I researched as to what parts might be feasible in the end and came up with the type of JLabel. However until I got to the point as to where I created the algorithm to modify the picture I was not sure that this type might actually work for this functionality.

Another challenge was to get to know NetBeans IDE to create this User Interface and to get everything to interact in a way I wanted it to. After learning some of it, it made my development easier, but every time it got easier something else was harder. The automatically generated code provides a challenge as some modifications can only be done by overwriting existing methods or through the NetBeans Interface.

### 5.2 Prototyping and work breakdown

Prototyping was my choice of programming method, because it provided me the most freedom and Dr. Mock was available most of the time when I had questions about this project. However he is not the end user and since the end user was not around I created the project based on my knowledge and Dr. Mocks input as a user.

My first step was to create a User Interface and have Dr. Mock inspect this and give me his ideas. This was done straight forward, because of the existing application and I tried to just copy the layout. My idea behind this was to create as little change for the end user as possible.

After creating part of this User Interface, I started working on building some functionality into the program and whenever I had something new working I consulted Dr. Mock as to whether it is a valid solution or needs to be changed. However here was not much surprise either as the functionality was kept as close to the original as possible.

The biggest challenge with this prototyping approach was that there was no formal deadline for anything and it was mostly up to me to get things done in a timely manner.

Figure 10 shows the approximate breakdown of the work that was done.

Figure 10 Project work approximation chart

# 6. Results

The GeoPixel Counter has a working version finished, which includes all of the features which were available in the original application. However some features don’t have the functionality that were available in the original application yet. I haven’t gotten any feedback yet from the end user, but I plan on spending spare time on this application to bring it up to the standards of the original application.

The user manual for the application can be found in Appendix A and shows all the functionality of this application as well as a list of known bugs.

### 6.1 Future plans/considerations

As I stated, I plan on at least bringing this program up to the standards of the original version and even transform it into a web applet.

Further considerations for me or any other programmer who wants to continue on this project are trying to make the loaded image resizable or automatically scale to a specific resolution. This will help the end user, because if small pictures are loaded it is hard to see which grain gets selected. A zoom function would solve this problem as well.

Another function that was talked about and is noted in the bug list is that the slider bar doesn’t work after a grain got selected and the sensitivity of this slider is not as big as it should be.

Furthermore a save option would be really nice to have and changing the exit pop-up dialog to asking if you want to save or not.

# 7. Summary and Conclusions

GeoPixel Counter was developed in Java 1.5 with the NetBeans IDE 6.8 for Dr. Mock with the goal of rewriting an existing C# application Java. The project had a working application completed in time. The final product might meet the requirements for the end user, however it doesn’t have all the functionality of the original C# application and should be developed further to implement the last functions.

I found this project a challenge to schedule my time around a project and to plan things ahead of time. I started a bit late in the semester and could have finished the product to meet my own standards if I would have started earlier. It gave me a good insight as to how programming in the free industry might work and that things don’t always work out the way we expect them to work out.

One thing that I would like to change in the future if working on such projects is to have formal deadlines right away, as they will help me concentrate on specific parts of the project up to these deadlines. Just working on it and seeing how much can be done is not a good option and having several deadlines for subparts of a project will help staying on track and finishing things in order.

Overall I’m satisfied that I finished a project of this extend, but I’m not satisfied with the way the application looks at this point. I know I could have done better with more time invested and therefore I will work on this project in my spare time to not disappoint the end user.

# 8. References

[1] Java API (n.d.) from <http://java.sun.com/j2se/1.4.2/docs/api/>

[2] Java BufferedImage (n.d.) from http://www.exampledepot.com/egs/java.awt.image/Image2Buf.html

[3] Java jLabel background color (n.d.) from http://www.rgagnon.com/javadetails/java-0304.html

**Appendix A: User Manual for GeoPixel Counter**

**Minimum System Requirements**

Any Operating System

512 Mb of memory

1 Ghz CPU

Video resolution of 1024 x 768

**Installation**

No installation needed.

**Starting the Program**

Run the program by double-clicking “GeoPixelCounter.jar”.

The program will launch and bring up a screen without a picture loaded. Click on the “Load” button and a pop-up window will appear which will lets you chose a file as seen in Figure 11:

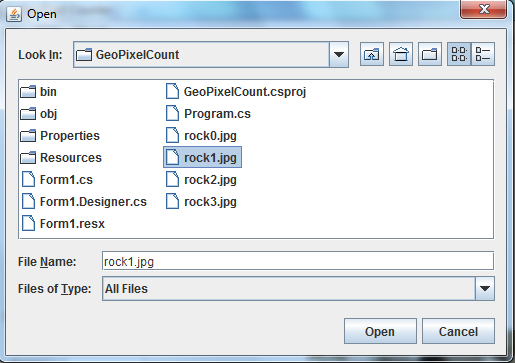


Figure 11 Open File Dialog Box

You want to choose pictures that end with .jpg, .png or .gif.

If you choose a wrong file type you will get the error message shown in Figure 12:

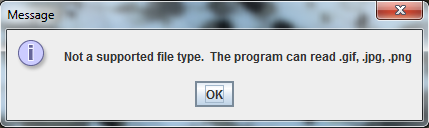


Figure 12 Not supported File Type Error Message

If you get this message, just click on the “Load button again and choose a supported file.

After the picture has been loaded you will see it in the screen which will look similar to the one in Figure 13 :

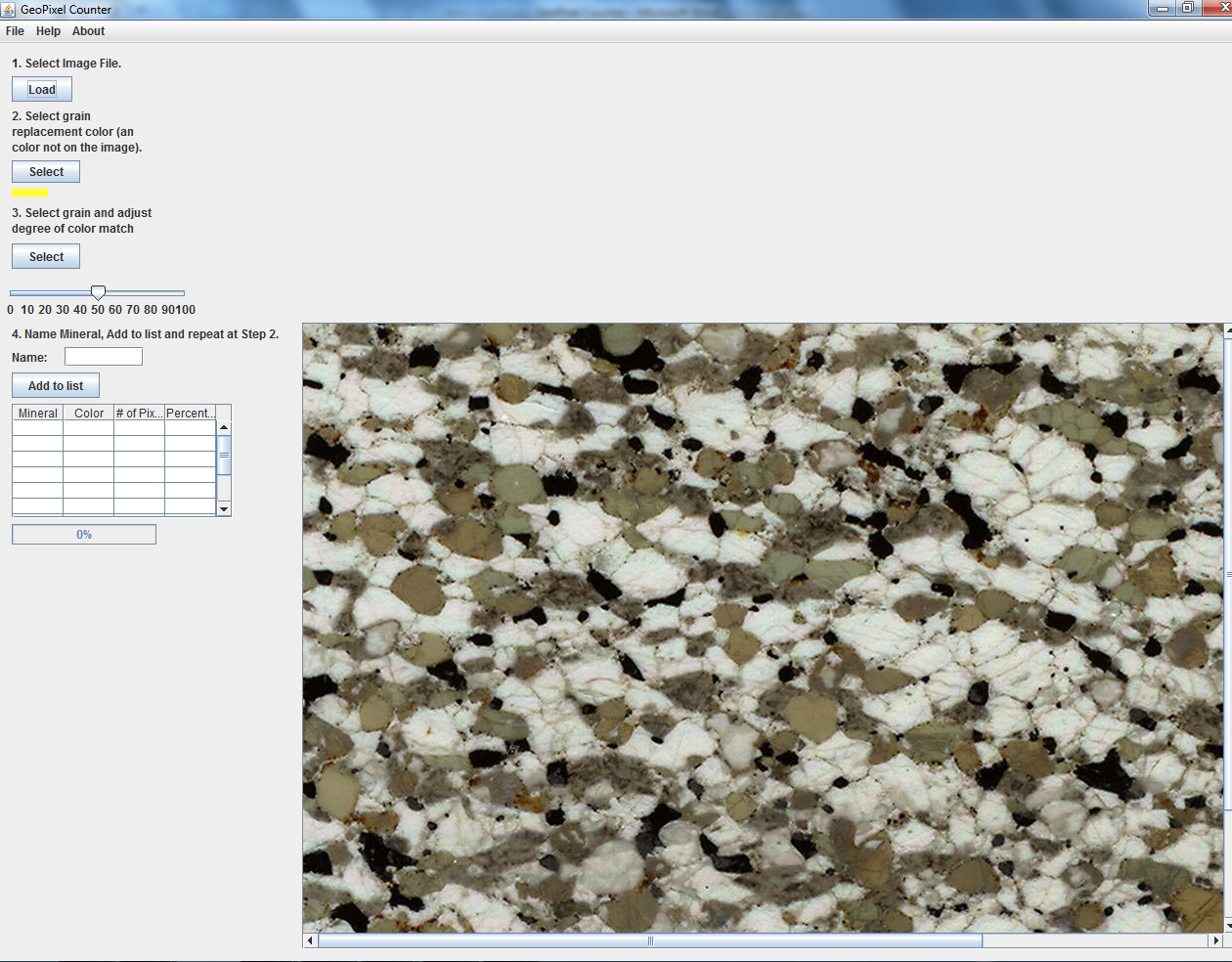


Figure 13 User Interface with image

**Selecting a different color to replace the grain**

You can choose a different color to replace the grain, however the best choice is to just go with the automatically created colors. If you still want to choose a different color, here is how it is done.

Click the “Select” button under the point 2 as you can see in Figure 14

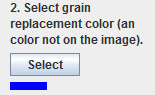


Figure 14 Select grain replacement Button + Color

and you will get a pop-up menu which looks like the one you can see in Figure 15:

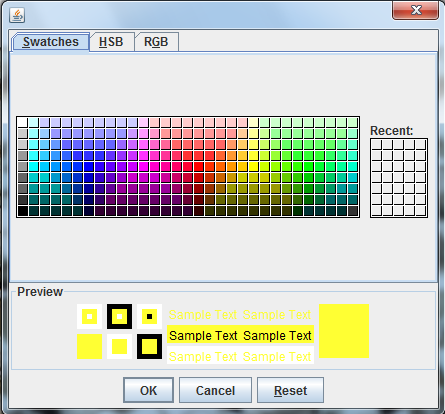


Figure 15 Color Chooser

In this menu you can either just click on a new color of your choice or you can select a new color based on HSB or RGB. For this you need to click on the top Tabs “HSB” or “RGB”. HSB lets you select a color based on where you click on a picture of colors and RGB lets you select a color based on red, green and blue values. After you have the color you want click “OK”.

You can always see which color is currently selected by looking at the color underneath the “Select” button in point 2.

**Selecting a grain on the picture and replace it with the chosen color**

If you are happy with your color selection and want to select a grain on your picture with this color here is how it is done. First you want to adjust the slider bar (seen in Figure 16) to the sensitivity you want your grain replacement to be.

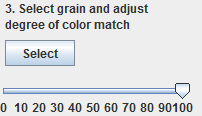


Figure 16 Grain adjustment slider + Select grain button

100 on this scale means it is really sensitive and takes more grains into consideration which aren’t that close to the selected grains color.

0 is not sensitive and replaces only the closest related colors of the chosen grain.

After you have selected the correct sensitivity you click on “Select” button and you will see that the screen changed and you see “Please Select a grain” on the top middle of your screen.

This means that you can click onto a grain on the picture to replace this grain and anything in related colors.

It should look like Figure 17:

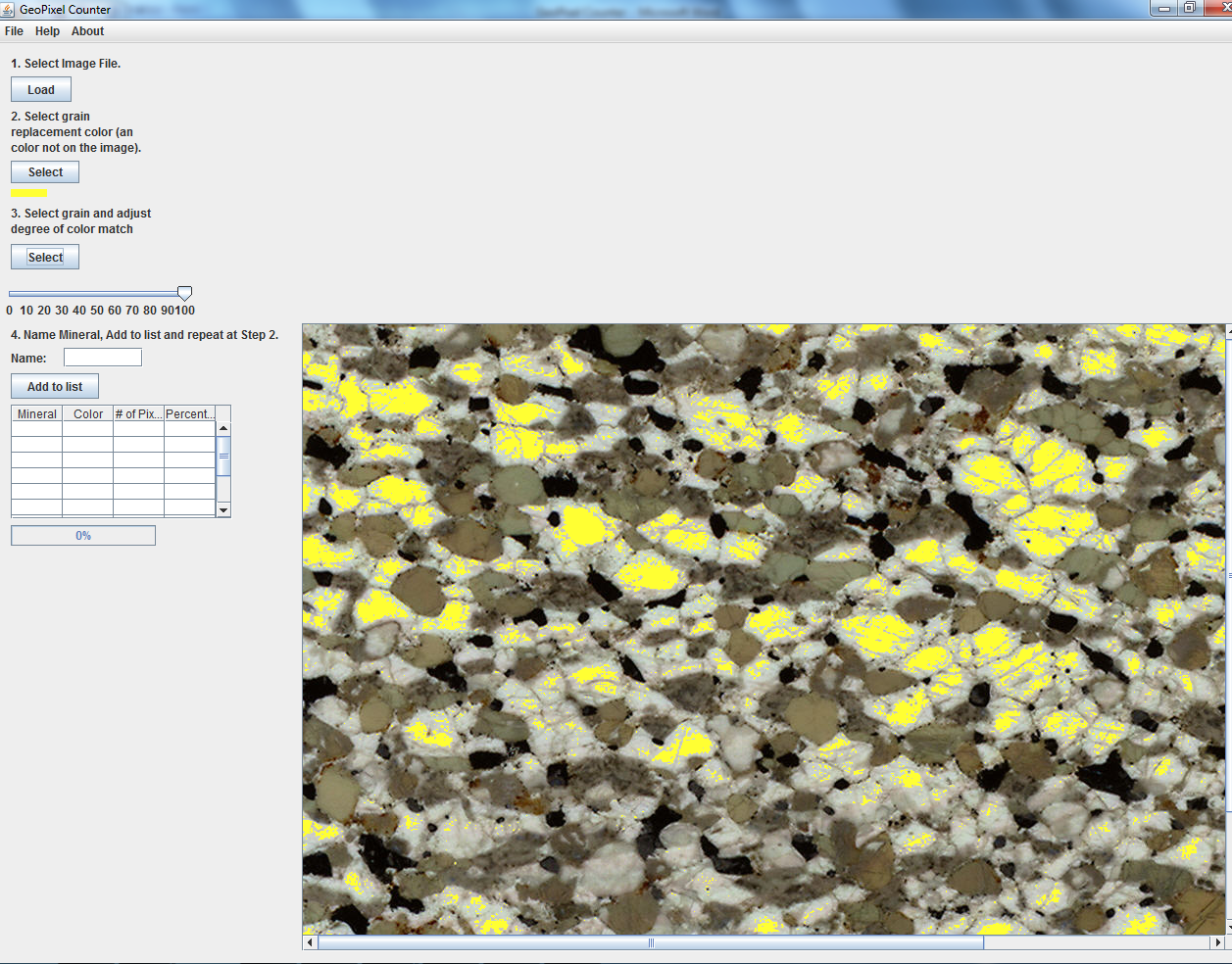


Figure 17 User Interface with replaced grains

**Adding a mineral name and adding it to the list**

After you have chosen and replaced a grain you want to add this to the list. To do so you need to enter a name under point 4.



Figure 18 Name Mineral Textbox

After you have entered the name you want to click on the “Add to list” button and you should see the list and screen updated like Figure 19:

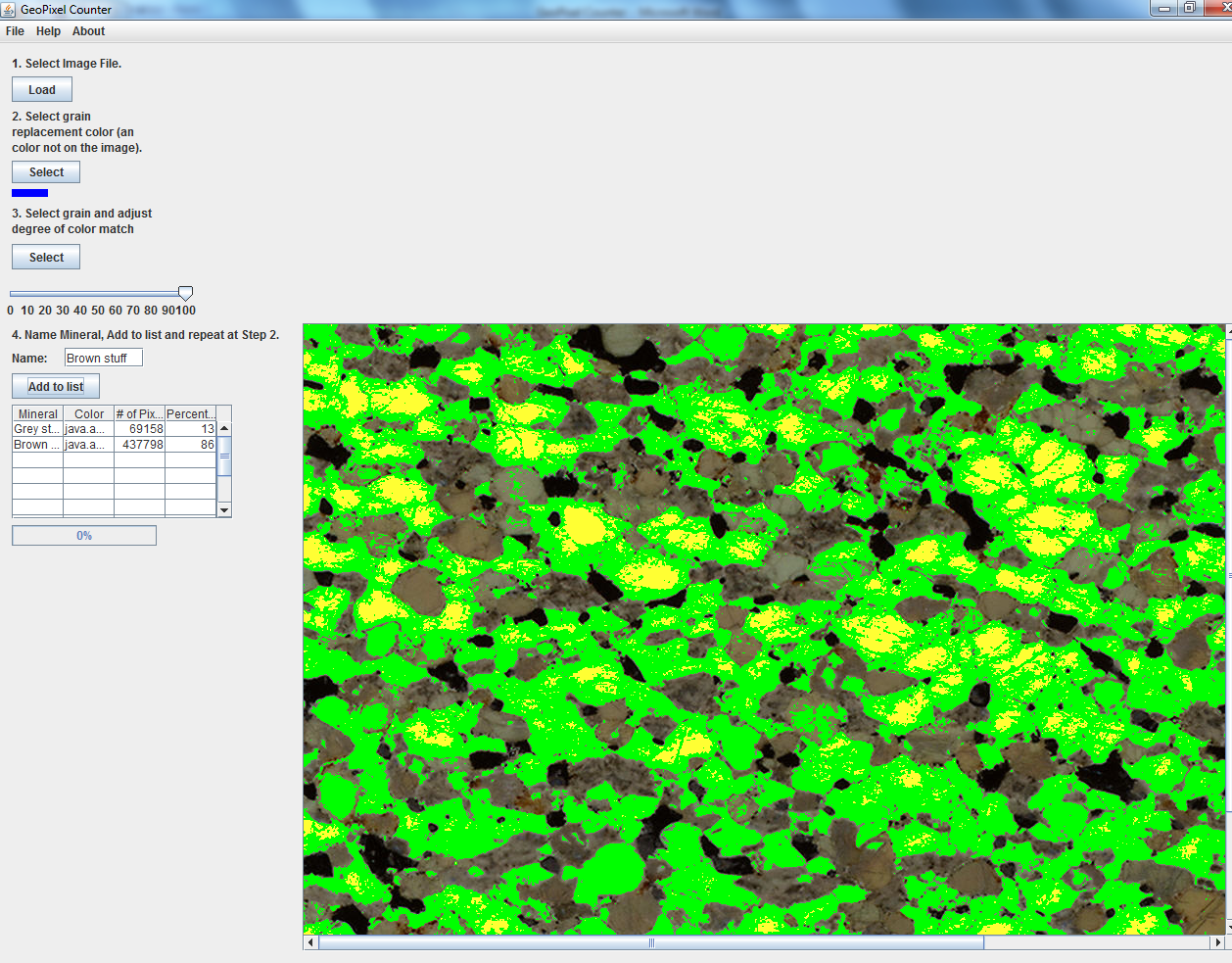


Figure 19 Updated User Interface with table contents

As you can see in this screenshot there are items in the list and the color has updated to the next in the pre-generated color list.

**Exiting**

When you would like to exit the application, choose File, Exit or click the X in the upper right corner.

**Known Bugs**

A couple things are known bugs in this program at the moment and will be resolved in the future:

* The slider bar has to be adjusted before a selection
* The slider bar option is not very sensitive and a selection of 100 doesn’t give many related pixels
* No option to save a selection
* No option to undo a change
* Percents don’t add to 100% but round up

If you discover any new bugs please send an email to Jbertmaring@uaa.alaska.edu