COMP151 - Project 1 - Cat Craft!

For your first project you will drop my son and a two of our pet cats into a Minecraft world he's created in hopes of starting a YouTube channel. Below you'll find project requirements, due dates, and grading rubric. Read this document carefully and pay extra attention to the project versioning and it's relationship to grading.

Project Versioning

Larger pieces of software are developed in increments, or *versions*. Each version is a working piece of software that builds upon the previous step but that also stands on its own merits. Individual components of the software are often called features. These are the things you want your code to do. Initial versions of the software often only complete just a few features and sometimes only parts of complex features. New versions of the software are often completed by adding additional code but sometimes existing code must be refactored and partially-rewritten to incorporate some new features into the program.

Below you'll find a sequence of versions for you to follow. You are to complete each of them in order. Test and verify each version before moving on to the next version. When starting a new version, you should modify the code for the prior version. The projected is completed in version 2.0. If you need or want to turn in an incomplete project, then you should turn in a completed version of an earlier project and not something akin to a selection of each version with no single version complete. To reinforce the importance of versioning, your grade is largely dependent on your ability to successfully carry out this versioning plan.

Version 0

The first version of the program has two main goals: setup proper program structure and carryout at at least one of the desired image processing tasks. It's really just meant to get you started on the right foot without having to wade into too many of the program details. It gets you past staring at a blank page and gives you a program that produces images. Later versions will push those images closer to the desired goal. A finished version 0 should have a well defined main function that coordinates the following functionality: loading the images of my son and the cats, convert a rectangle containing my son and rectangles containing the cat's faces to a 8 bit color pallet using posterization, and display the three recolored images. The recoloring should be done with a single helper function designed to work on a specified region of an image. A complete version 0 program will, when run, display the three images (one kid and two cats) where the cat's faces and a small region including all of my son have been converted to an 8 bit color scheme but the rest of the image is unchanged.

Version 0.5

Version 0.5 of the program crops out the cat's faces. The main program should now display the full image of the boy and images of just the cat's faces. All three images should still have the 8 bit colorization from version 0. Your code design should allow you to do the re-coloring after the cropping, but with cropped image you end up re-coloring the whole image not just a region. Set things up so the cats are cropped then colorized.

Version 0.75

Version 0.75 of the program scales down the images to half size. You can add this step before or after the version 0.5 work, but doing it before the cropping and coloring will likely speed-up your program as they'll have fewer pixels to deal with. The trade-off is that you'll need figure out new starting and ending x and y coordinates for the smaller images. Go ahead and do the resizing first and the other effects after that. Your main program should now display the three smaller versions of the images and only these smaller versions.

Version 1.0

Only one thing is left before we have the images we need for constructing the Minecraft image. Scaling down the images likely made them a bit pixelly. While this certainly fits with the Minecraft asthetic, let's go ahead and apply blurring to the images to smooth them out a bit. At this point we need to be careful how we combine all the effects. Blurring after re-coloring will convert out 16 bit color pallet to something else. Be sure to setup you main so that the blurring takes place after the resizing but before the recoloring. The chain of effects for the cat should be this: resize, crop, blur, recolor. For the boy you drop the crop step but be

certain to only apply the blur and recolor to a smaller region containing the boy. You should not need or half separate functions for the boy and the cats.

Version 1.5

As of version 1.0 we have fully prepared images of my son and the cats. It's time to think about adding them to the Minecraft world. We'll start with the cats using a simple copy operation. Create a copy function and use it to copy the two cat heads onto the two cats in the Minecraft image that are closer to the camera. Try to get them roughly over the current cat heads. Modify your main so that it displays the image of my son and the Minecraft image with the cat faces copied into it.

Version 2.0

The last step is to add my son to the image. As you can see, he is standing on a green screen. I think you know that means: we're going to use chromakey to copy him into the image. The basic chromakey function from the book will not work though. What you want to do is place him at a specific point like copy and then, while copying, only copy pixels that are not the green. Notice that this is alomst the reverse of what we see in the book. There we are replacing green pixels with background pixels but in this code we want to copy non-green pixels into the background. Place my son so he's roughly between the front two cats. The main program should now complete and display the final image where in we see my son and two of my cats in the Cat Craft world. In addition to displaying the finished product, have your program write that image to your computer so you can submit it along with your code.

Grades and Dates

We'll dedicate two lab periods and some class time to this project. Use that time wisely. The final grade for the project is determined by how many versions you complete and the quality of the code you submit.

Important Dates

Date	Event
3/16	Lab time to begin project
3/23	Lab time for Project Work
3/26	**Project Due by 8am**

Projects should be submitted by the start of class on Friday 3/26. Submit your code to Gradescope and your final image to Classroom.

Grades

Your grade is determined based on the highest version number you complete (see the chart below) and the overall quality of the work submitted. Your program should not do bits and pieces of each version. For example, copying the heads to the minecraft characters without pixelating and posterizing them first does not award credit for version 1 and will be viewed as an incomplete version 0 or 0.5. So *follow the versioning*. Get version 0 working properly, then modify that program to complete version 0.5, and so on. Do not turn in a separate set of code for each version. Once you complete a version you can add and remove whatever you need to in order to complete the additional features of the current version.

Version	Grade
0	D
0.5	C-
0.75	C
1.0	C+

Version	Grade
1.5	B
2.0	A

Quality is measured by the cleanliness and clarity of the code and the appropriate use of sub-functions that exhibit reusable design. The versions determine the range in which you grade will fall and the quality of your code will determine where your grade lands within that range. For example, let's say you've finished version 1.5 and your code is neatly presented and well designed. Your grade starts at a B due to the version that you're presenting and could go up as high as a low A- based on the quality. Conversely, if your code is all crammed into a single function and your variables are poorly named, then your grade could go down as low as a high C+ due to poor quality.

Comments on Image Transformations

8-Bit color

The pixel colors we're used to are 24 bit, 8 bits per color channel. Older hardware used 8 total bits for color: 3 bits for red, 3 for green, and 2 for blue. We can attempt to convert 24 bit color to 8 bit color using posterization. For red and green we first imagine dividing the 256 color values into sequential chunks of size 32, i.e. 0 to 31, 32 to 63, and so on. This is because 3 bits of color allows for 8 possible colors and 32x8 is 256. We can then assign a single color to any value within that chunk. That value could naturally be the minimum or maximum in that range. We might also choose the average or handpick some representative color. The process for blue is the same but our chunks are size 64. You're welcome to play around with this conversion process, distribute the 8 bits between the color channels differently, and choose the target colors more purposefully so long as you produce an 8bit color pallet from the original 24bit pallet.