COMP151 - Project 1 - Make it Minecraft

For your first project you will attempt to put my kids and one of my cats into a video game. More specifically, you need to pixelate, crop, and resize some head shots of my children and one of my cats, and then copy their faces onto the bodies of some Minecraft characters and place them into a Minecraft scene along side some pandas. They love pandas. You'll be provided with the images of my kids, my cat, the minecraft characters, and the panda scene. If you'd rather work with different images, then that's allowed as well. You just need to carry out the same transformations: crop, scale, pixelate and copy three heads onto three bodies and then scale and move that into some other scene. Just run your proposed images by me first.

Below you'll find descriptions of each version of the project, details about key image transformations, and finally the details about grades and due dates.

Project Versioning

Larger pieces of software are developed in increments, or versions. Each version produces a working piece of software that builds upon the previous step but that also stands on its own merits. Initial versions of the software often only complete some of the desired tasks and in some cases does so in a simplified way. 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. For your project you must following the versioning laid out below. To reinforce the importance of versioning, your grade is largely dependent on your ability to successfully carry out the given 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. The finished product should:

- Have a well defined main function (as seen in the prep lab) that coordinates the following functionality:
  - Loading the images of the kids and the cat
  - Converting the color palate of the region around their faces to 8 bit color using posterization.
  - Display the results of the color conversion.

A complete version 0 program will, when run, display three images (two kids and a cat) where the faces in those images have been converted to an 8 bit color scheme but the rest of the image is unchanged. Remember that any and all image manipulation tasks should be handled by sub-functions that following general reusable design patterns as seen in the prep lab from last week.

Version 0.5

The second version of the program adds pixelization to the faces (see below) before doing the 8-bit color conversion. You can play around with the size of the pixels but you need to do at least a 6x6 region.

Version 0.75

The third version of the program crops and scales down the faces. You should do this after the color conversion and pixelation. So at this point, the process is as follows: pixelate the faces, change the color to 8 bit color, crop and shrink the faces into new images. Your program should now display just the 8-bit, pixel faces rather than the whole images.

Version 1.0

Now that you have 8-bit, pixelated versions of the faces you can copy them onto the Minecraft character image. In order to do this you'll need to shrink each head down to half size. I recommend you first check the size of the heads in the minecraft image and then revisit the size of the face region your image cropping produces so that the cropped face is a close to twice the size of the target minecraft face as you can get. It doesn't need to be perfect but do put a little thought into it. The end result of this program is the minecraft character image with my kids' and cat's heads placed over the characters. This image should be the only image displayed by your main function. You should also modify the main function so that the image is also written to a file.

Version 1.5

Now that you can create minecraft versions of my kids and one of my cats you can start placing them into the panda scene. Try top copy a tight box around the characters rather than the entire image. Don't worry about the background from the character image for now. If you need to scale the characters down or up, then you can do so. This program should show and write to a file the image with the minecraft version of my family with their beloved pandas.

Version 2.0

The last thing to do is to add a chromakey effect as you copy the characters into their new background. Modify your program so that background pixels from the character image are replaced with the appropriate pixel from the final background. Lucky for you, the background of the character image is a pretty clear, specific color that doesn't seem to appear in the characters themselves and is not a part of the 8-bit scheme used in the heads. The main program should, of course, display and write to the the hard drive the final product.

Version 2.25 (Extra)

Go beyond version 2.0! A great place to start is trying to more carefully crop out the kids and cat's head so that the backgrounds of those images do not appear in the final product. After that, you can get as creative as you'd like with the project so long as it's not a major departure from the goal of placing my kids in a game-scene and retains the essentials from version 2.

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.

Artificial Pixelization

In the book we learn about the pixelization effect that occurs when you scale an image up and how blurring can be used to smooth out the result. For this project we want to artificially create blocky, pixel-like spots of color. To accomplish this we first imagine grouping pixels into square clusters that will become the new "pixels". You can play with the size of the cluster for your project but need to use at least 6 pixels by 6 pixels for a good end result. I found I liked how 8x8 clusters looked. Now our goal is to make every pixel in a cluster the same color. To do this you go cluster by cluster, compute the average pixel color in the cluster, and then assign that color to every pixel in the cluster. This is very, very similar to blurring. In blurring the clusters are 2 by 2 and we let them overlap. To artificially pixelate the image we we use larger clusters that do not overlap.

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
10/12	Open Lab time to begin project
10/19	Lab time for Project Work

Date	Event
10/20	Lab time for Project Work
10/26	**Project Due**

Projects should be submitted by 11:59 midnight on Friday, a.k.a midnight Saturday. Submit your code and your final image to Moodle.

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+
1.5	B
2.0	A
2.25	Bonus

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.