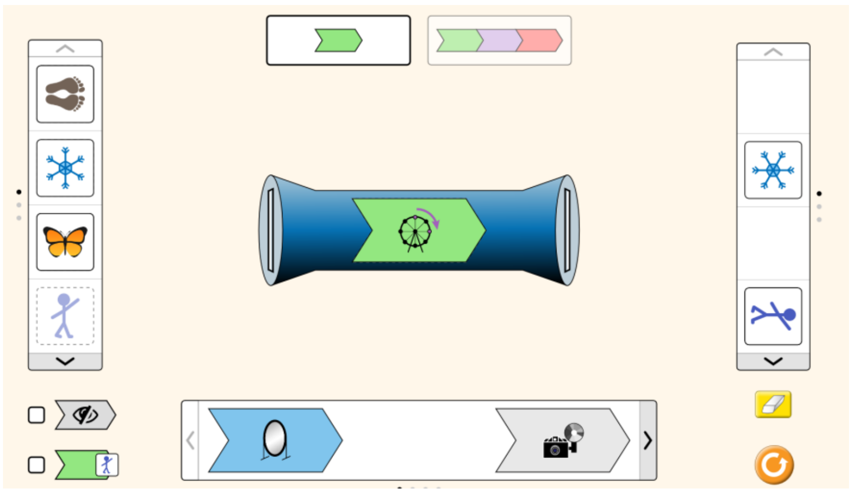
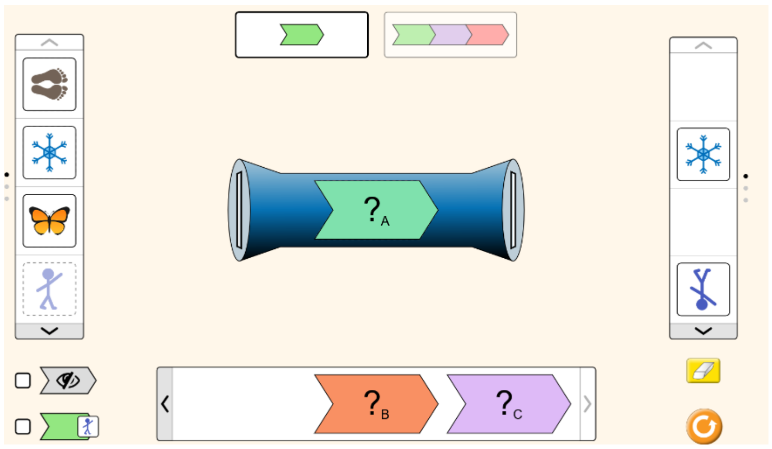
Image Project PART A
Theo Shin

1. From the project introduction, image manipulation at first, may appear as a "black box" where photos are magically transformed, their colors altered, flipped vertically/horizontally, etc.," however, computers can only deal with numbers and the manipulations that viewers see are akin to optical illusions. For example, the educational app at https://phet.colorado.edu/sims/html/function-builder-basics/latest/function-builder-basics_en.html was developed to show the function builder basics of image manipulation by inserting images on one end of a tube and dragging it through to the other end while implementing varying image manipulation tools.



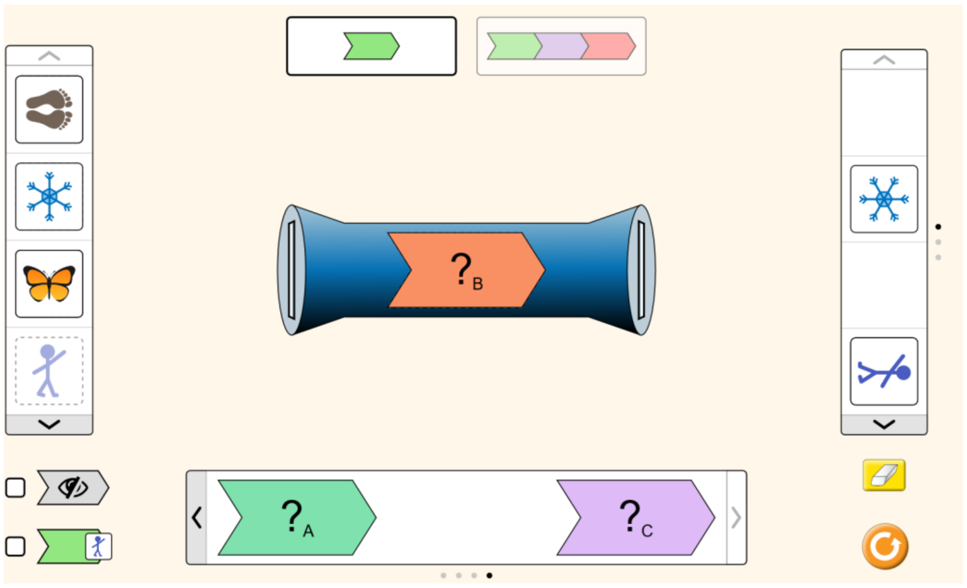
(Example from the website showing the Ferris wheel tool rotating images by 90 degrees)

Looking at the bottom toolbar, the very last tools are the "mystery" function cards shown as $?_A$, $?_B$, and $?_C$. $?_A$ is similar to the Ferris wheel tool except instead of rotating images by 90 degrees, it rotates the images by 180 degrees or more precisely stated, the images are rotated along a horizontal axis and are thus "upside-down."

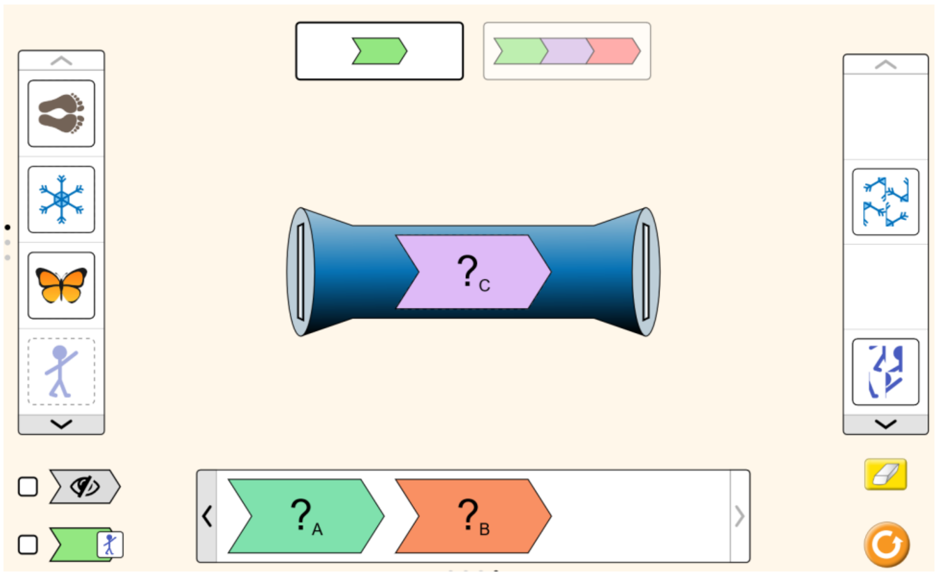


(Example of "mystery" function A)

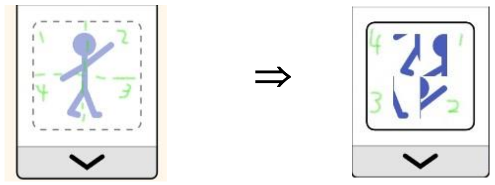
Here, we can see the stick figure has been turned upside down and although the snow flake looks untouched it has been flipped (Since the snowflake is symmetrical it still looks the same).



"Mystery" function B again deals with rotating images, but this time, flips the image along the vertical axis (Looking at the stick figure, the raised right arm now rotates to a raised left arm and its feet are pointing in the opposite direction) and then the image is rotated 90 degrees not unlike the Ferris wheel function.

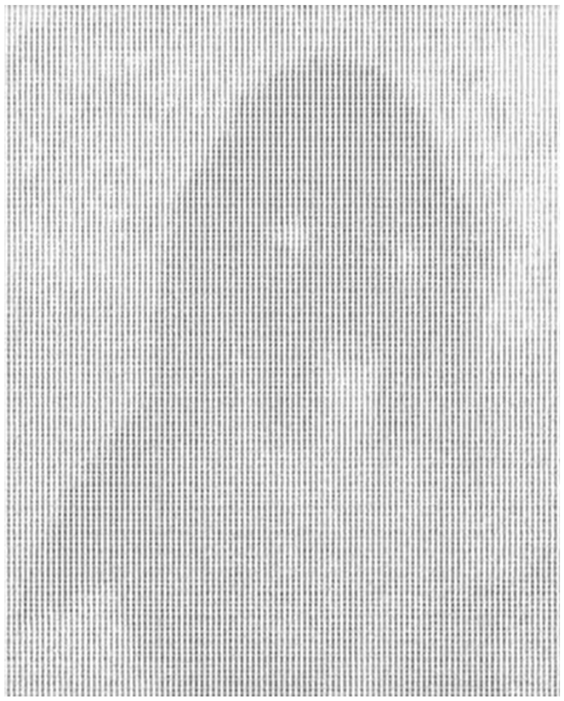


"Mystery" function C breaks apart the image into 4 different quadrants and then rearranges the quadrants. Splitting the stick figure into 4 quadrants (split vertically/horizontally) then labeling each quadrant 1 through 4 in clockwise fashion, "mystery" function C rotates each quadrant

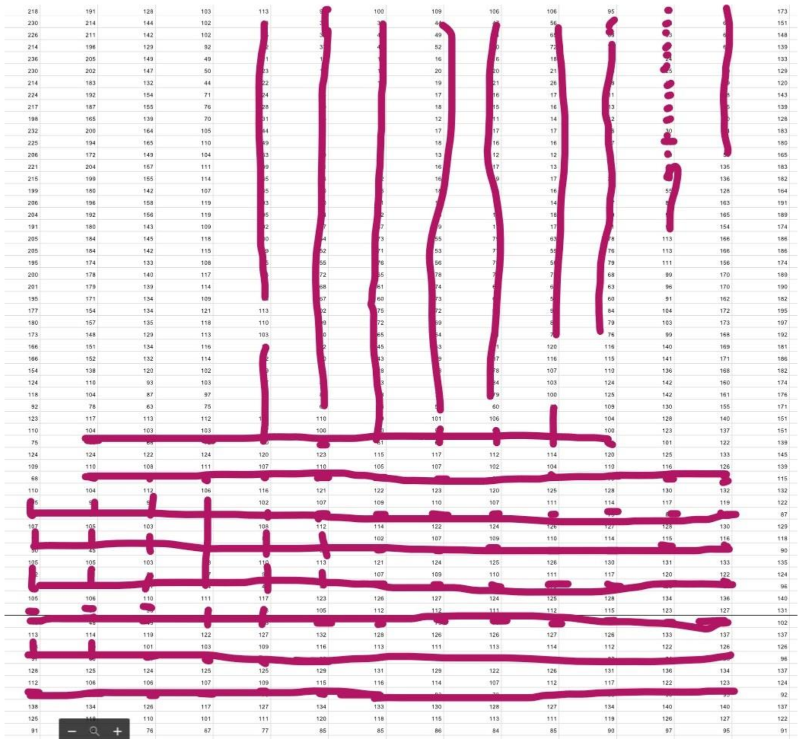


clock-wise, so that quadrant 1 replaces the original location of quadrant 2, quadrant 2 replaces the original location of quadrant 3, and so on. The developers of this app must have had a good understanding of image spreadsheets and how to manipulate the corresponding RGB and positional values to alter the images. In order for the tools to change the image colors, the developers would have to think about how they would want the colors to change and which values to add/subtract from RGB values to create the "illusion" the image is now shown in negative, or with a purplish hue, or various intended color effects. Since developers understand the "optical illusion" of seeing spreadsheet values from a zoomed out perspective, images are transformed by inserting them through one end and having the modified image outputted on the other end through conditional formatting (not "magically"). The developers approach to this image modifying application is very similar to approaching coding problems in general. That is, establishing an end goal and separating coding tasks into manageable sections for better cohesion and success in reaching a desired end state. In order for an image to be rotated "x" degrees, the developers have to consider which value needs to be adjusted for every pixel so that the resulting image appears rotated. In actuality, the positional values of each pixel in the spreadsheet are adjusted to achieve this desired effect.

2. Through conditional formatting of the spreadsheet values associated with the puppy image, the lower the value (range from 0-255) the darker the intensity and the higher the value (for technical purposes, value 256 would correspond to "white") the lighter the intensity. Even without adding the color rules, when zoomed out and just looking at a spreadsheet of numbers, a faint image of the puppy remains visible:



In contrast to the puppy image following color formatting, it's apparent the lighter intensities (puppy has a white coat of fur and blackish pigment for nose and eyes) represent the shadowed portions of the spreadsheet numbers in the above image. The puppy's whitish coat is displayed as the darker portions and provides the outlines of the dog, and in addition the puppy's eyes and nose are displayed as whitish outlines. Due to the lighter intensities of the puppy's white coat represented as higher values (ranging from roughly 190-250) and the darker portions of the image represented as lower values (ranging from roughly 45-100), the differences represent the varying shades of lightness/darkness.

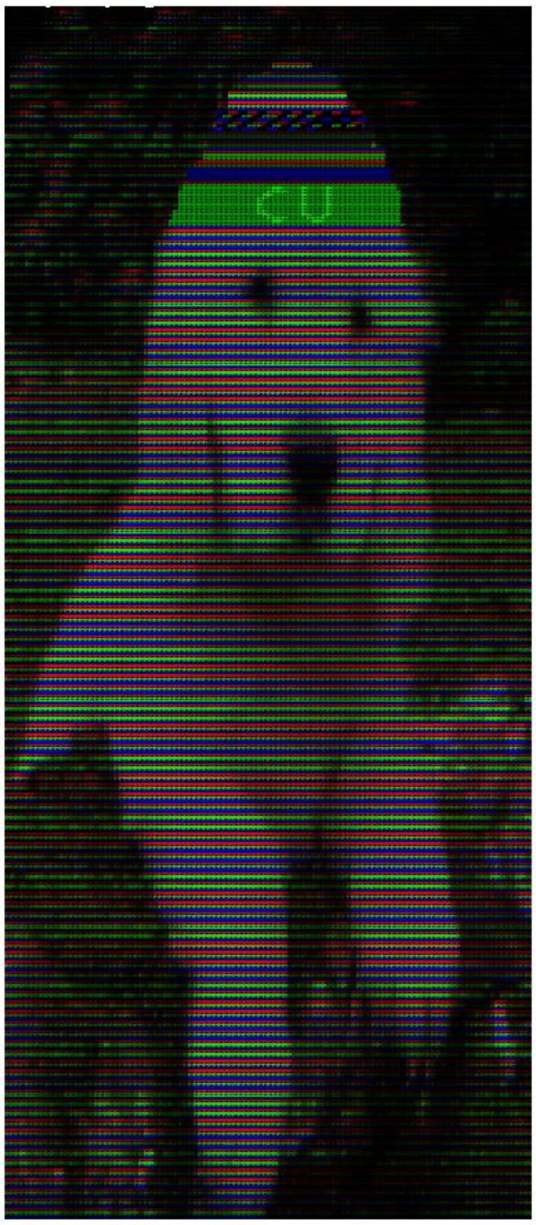


218	191	128	103	113	[illegible]	100	109	106	106	95	[illegible]	[illegible]	173
230	214	144	102	[illegible]	[illegible]	[illegible]	44	[illegible]	56	[illegible]	[illegible]	[illegible]	151
226	211	142	102	[illegible]	[illegible]	[illegible]	49	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	148
214	196	129	92	[illegible]	[illegible]	[illegible]	52	[illegible]	72	[illegible]	[illegible]	[illegible]	139
236	205	149	49	[illegible]	[illegible]	[illegible]	16	[illegible]	18	[illegible]	[illegible]	[illegible]	133
230	202	147	50	23	[illegible]	[illegible]	20	20	21	[illegible]	[illegible]	[illegible]	129
214	183	132	44	22	[illegible]	[illegible]	19	21	26	[illegible]	[illegible]	[illegible]	120
224	192	154	71	24	[illegible]	[illegible]	17	16	17	11	[illegible]	[illegible]	143
217	187	155	76	28	[illegible]	[illegible]	18	15	16	13	[illegible]	[illegible]	139
198	165	139	70	31	[illegible]	[illegible]	12	11	14	[illegible]	[illegible]	[illegible]	128
232	200	164	105	44	[illegible]	[illegible]	17	17	17	[illegible]	30	[illegible]	183
225	194	165	110	49	[illegible]	[illegible]	18	16	16	[illegible]	[illegible]	[illegible]	180
206	172	149	104	53	[illegible]	[illegible]	13	12	12	[illegible]	[illegible]	[illegible]	165
221	204	157	111	[illegible]	[illegible]	[illegible]	16	17	13	[illegible]	[illegible]	135	183
215	199	155	114	[illegible]	[illegible]	[illegible]	16	[illegible]	17	[illegible]	[illegible]	136	182
199	180	142	107	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	16	[illegible]	55	128	164
206	196	158	119	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	14	[illegible]	[illegible]	163	191
204	192	156	119	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	18	[illegible]	[illegible]	165	189
191	180	143	109	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	17	[illegible]	[illegible]	154	174
205	184	145	118	[illegible]	[illegible]	73	55	[illegible]	63	78	113	166	186
205	184	142	115	[illegible]	52	71	53	[illegible]	[illegible]	76	113	166	186
195	174	133	108	[illegible]	55	76	56	[illegible]	[illegible]	79	111	156	174
200	178	140	117	[illegible]	72	65	78	[illegible]	[illegible]	68	99	170	189
201	179	139	114	[illegible]	68	61	74	[illegible]	[illegible]	63	96	170	190
195	171	134	109	[illegible]	67	60	73	[illegible]	[illegible]	60	91	162	182
177	154	134	121	113	[illegible]	75	72	[illegible]	[illegible]	84	104	172	195
180	157	135	118	110	[illegible]	72	69	[illegible]	[illegible]	79	103	173	197
173	148	129	113	103	[illegible]	65	[illegible]	[illegible]	[illegible]	76	99	168	192
166	151	134	116	[illegible]	[illegible]	45	[illegible]	[illegible]	120	116	140	169	181
166	152	132	114	[illegible]	[illegible]	43	[illegible]	[illegible]	116	115	141	171	186
154	138	120	102	[illegible]	[illegible]	[illegible]	[illegible]	78	107	110	136	168	182
124	110	93	103	[illegible]	[illegible]	[illegible]	[illegible]	84	103	124	142	160	174
118	104	87	97	[illegible]	[illegible]	[illegible]	[illegible]	79	100	125	142	161	176
92	78	63	75	[illegible]	[illegible]	[illegible]	[illegible]	60	[illegible]	109	130	155	171
123	117	113	112	[illegible]	110	[illegible]	101	106	[illegible]	104	128	140	151
110	104	103	103	[illegible]	100	[illegible]	[illegible]	[illegible]	[illegible]	100	123	137	151
75	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	61	[illegible]	[illegible]	[illegible]	[illegible]	101	122	139
124	124	122	124	120	123	115	117	112	114	120	125	133	145
109	110	108	111	107	110	105	107	102	104	110	116	126	139
68	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	115
110	104	112	106	116	121	122	123	120	125	128	130	132	132
[illegible]	[illegible]	[illegible]	[illegible]	102	107	109	110	107	111	114	117	119	122
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	87
107	105	103	[illegible]	108	112	114	122	124	126	127	128	130	129
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	102	107	109	110	114	115	116	118
[illegible]	45	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	90
105	105	103	[illegible]	110	113	121	124	125	126	130	131	133	135
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	107	109	110	111	117	120	122	124
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	96
105	106	110	111	117	123	126	127	124	125	128	134	136	140
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	105	112	112	111	112	115	123	127	131
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	102
113	114	119	122	127	132	128	126	126	127	126	133	137	137
[illegible]	[illegible]	101	103	109	116	113	111	113	114	112	122	126	126
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	96
128	125	124	125	125	129	131	129	122	126	131	136	134	137
112	106	106	107	109	115	116	114	107	112	117	122	123	124
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	92
138	134	126	117	127	134	133	130	128	127	134	140	140	137
125	[illegible]	110	101	111	120	118	115	113	111	119	126	127	122
91	[illegible]	76	67	77	85	85	86	84	85	90	97	95	91

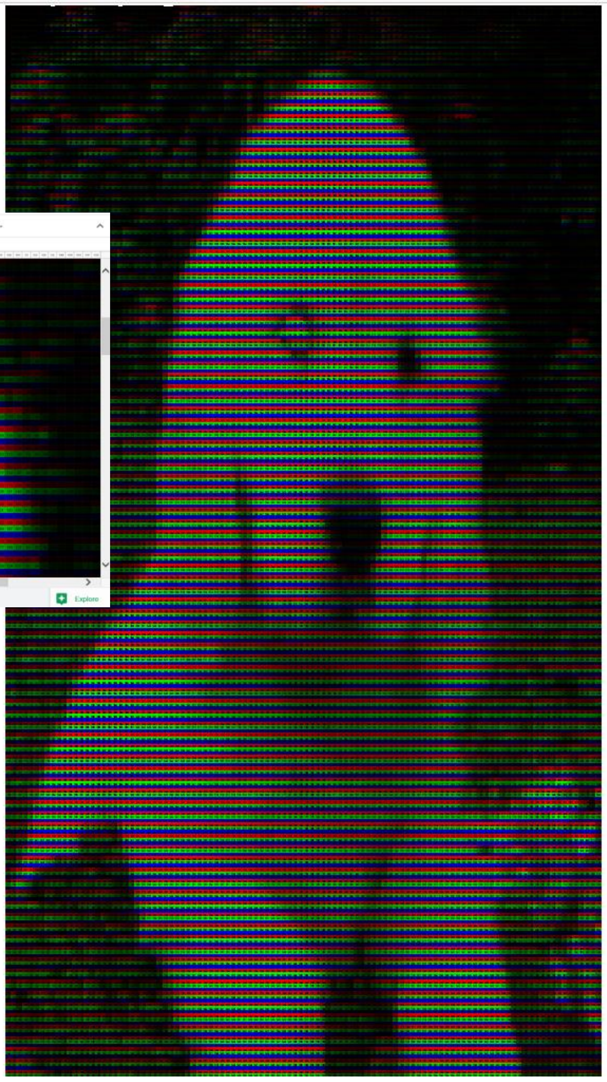
Observing a zoomed-in image for the spreadsheet values (above), the purplish marks make up the values under 100 (darker intensities) and the traces of these marks form the shape of the puppy's nose. All photos will not show visible traces without color information (at least not as starkly as the puppy image). Those images without a strong light/dark contrast would become difficult in viewing. If all the spreadsheet values are less than 100 or if the variances in values are less than 50, no visible outline could form in order to obtain even faint images (i.e. photo of all clouds, or possibly waves in an ocean, images with intense uniformity). The spreadsheet values of a strongly uniformed image (without stark contrasts) would need further color information to more adequately display the image.

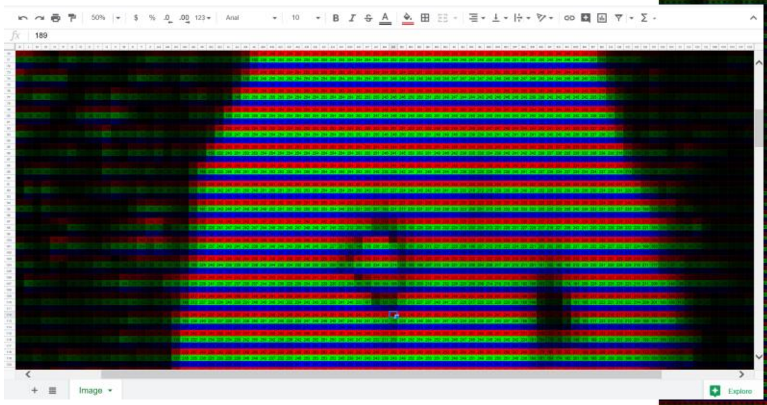
3. I decided to knit the puppy a multicolored University of Colorado beanie using spreadsheet values.

Resulting Image



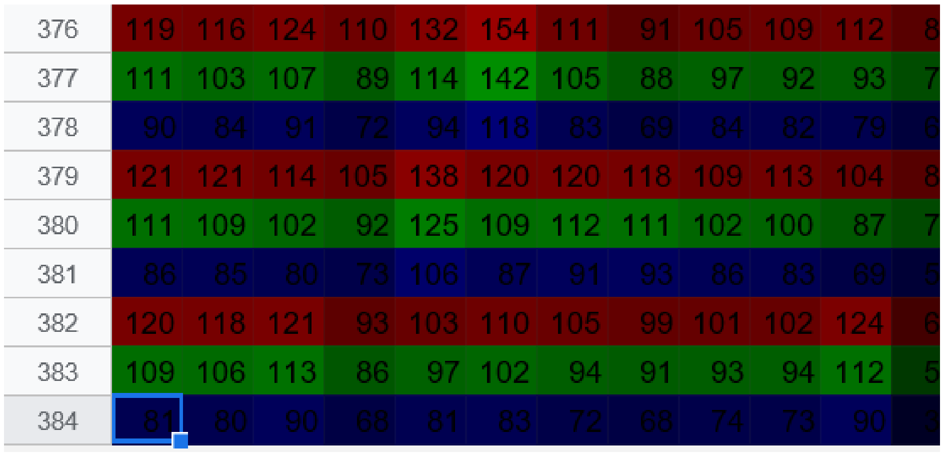
4. In order to produce a negative image, the RGB value is subtracted from 255 (255-value). I decided to implement negative values for the puppy's right eye to obtain this result ("poor puppy…"):





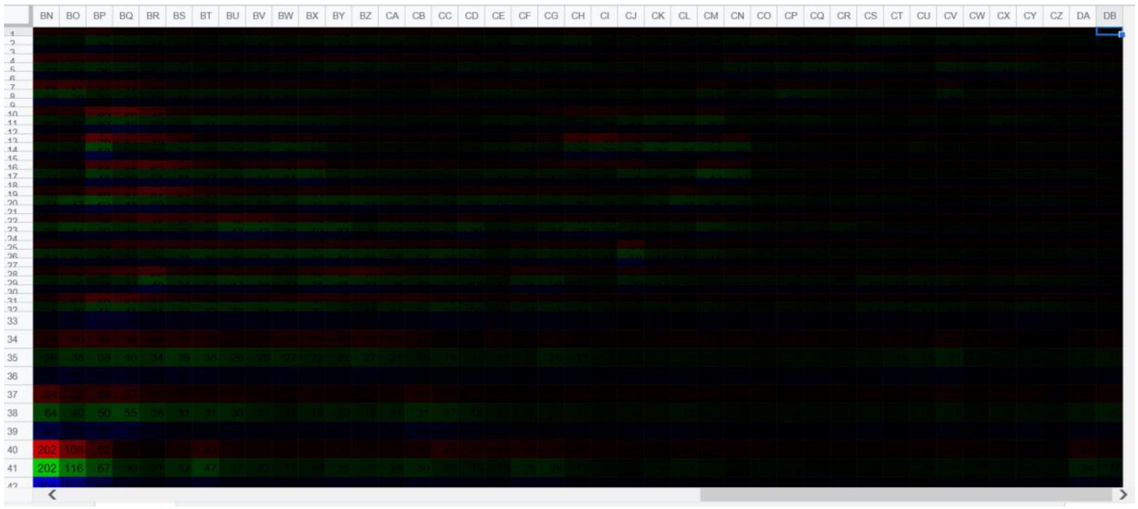
In order to create a negative of the entire image, the function 255-value would be need to be adjusted for every pixel-value cell contained in the image. For demonstration purposes, I chose to only implement the negative values to a small portion of the puppy image.

5. In order to flip the image vertically (along the x-axis), the A1 cell and its value would need to be mirrored along the x-axis and moved to the bottom-left cell represented as A384 (Left most column, bottom most row) and continued along corresponding cells A2 ⇒ A383, B2 ⇒ B384, B2 ⇒ B 383, and so on:



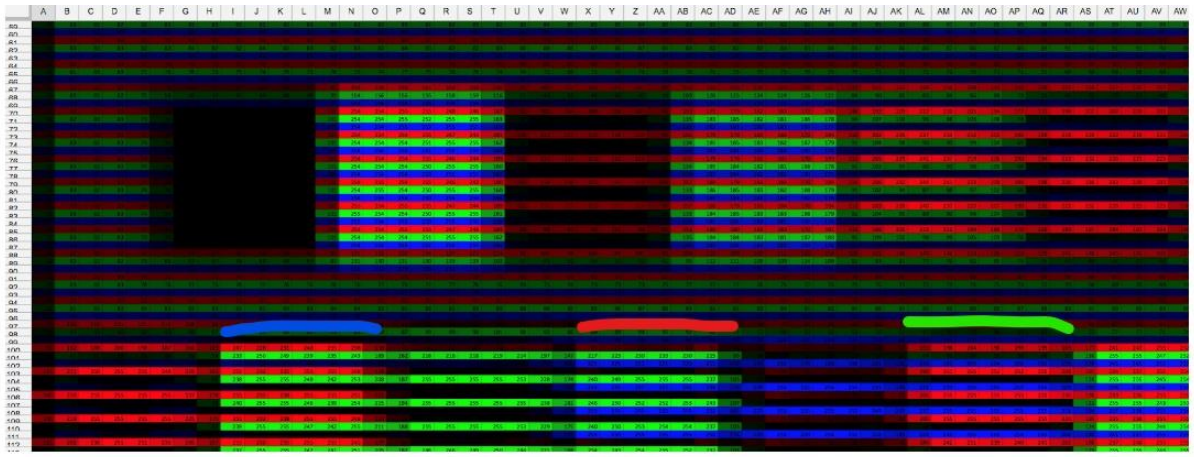
376	119	116	124	110	132	154	111	91	105	109	112	8
377	111	103	107	89	114	142	105	88	97	92	93	7
378	90	84	91	72	94	118	83	69	84	82	79	6
379	121	121	114	105	138	120	120	118	109	113	104	8
380	111	109	102	92	125	109	112	111	102	100	87	7
381	86	85	80	73	106	87	91	93	86	83	69	5
382	120	118	121	93	103	110	105	99	101	102	124	6
383	109	106	113	86	97	102	94	91	93	94	112	5
384	81	80	90	68	81	83	72	68	74	73	90	[illegible]

Similarly, to invert the image horizontally (along y-axis), cell A1 and its value would need to be switched with cell DB1 (DB2 ⇒ A2; DA1 ⇒ B2; and so on):

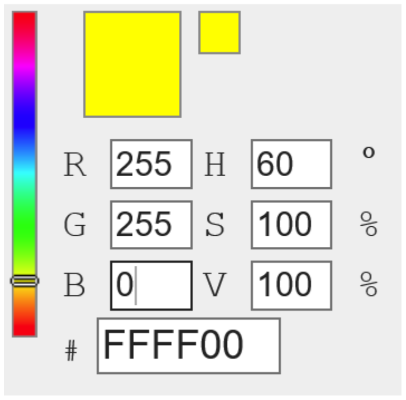

BN BO BP BQ BR BS BT BU BV BW BX BY BZ CA CB CC CD CE CF CG CH CI CJ CK CL CM CN CO CP CQ CR CS CT CU CV CW CX CY CZ DA DB

The desired “flip” effect could computationally be achieved by implementing arrays/matrices within an algorithm and using height/width parameters to modify (x,y) pixel coordinates (possibly using sin and cosine calculations), in order to rotate the images accordingly. In the case of flipping the images, the algorithm output would need to associate the value in cell A1 to cell A384, along with each respective cells across the horizontal plane (part_B intuitively uses the swap function).

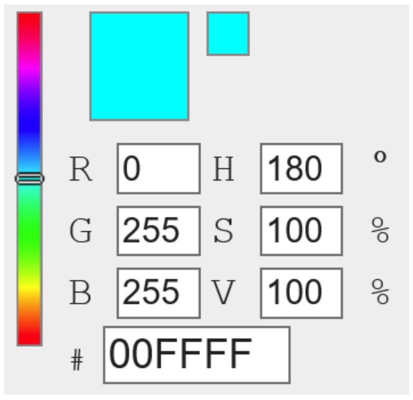
6.



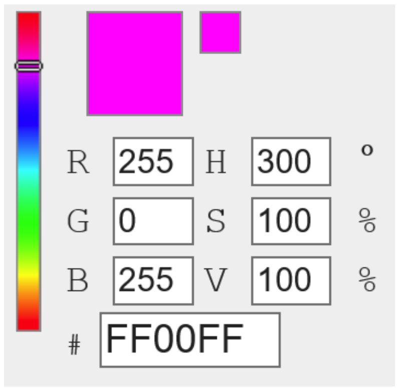
Using the color swatch from (https://think-maths.co.uk/spreadsheet), I've drawn a colored line above the blue, red, and green filters (blue = 0, red = 0, green = 0) respectively. Looking at the second row at the boxes with the identifying blue, red, and green filters, yellow comes from the absence of blue, turquoise comes from the absence of red, and magenta coems from the absence of green. Corresponding colors resulting from filters shown below (https://www.rapidtables.com/web/color/RGB_Color.html):



R 255 H 60 °
G 255 S 100 %
B 0 V 100 %
FFFF00

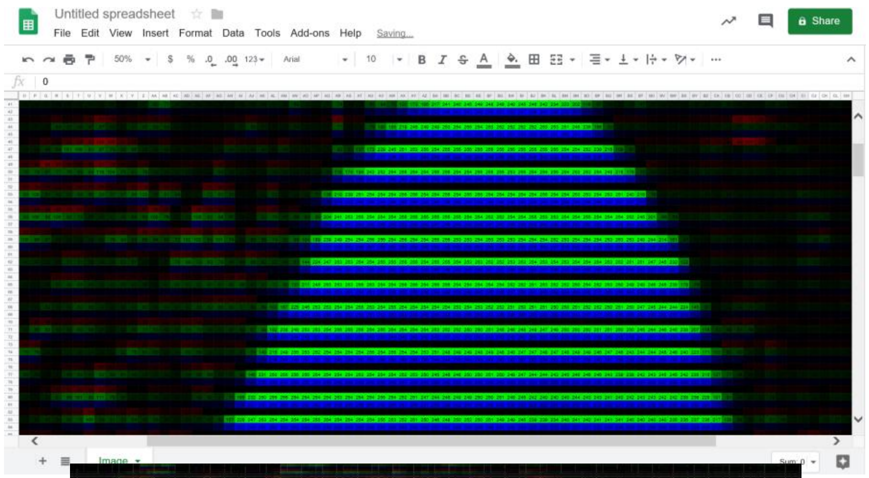


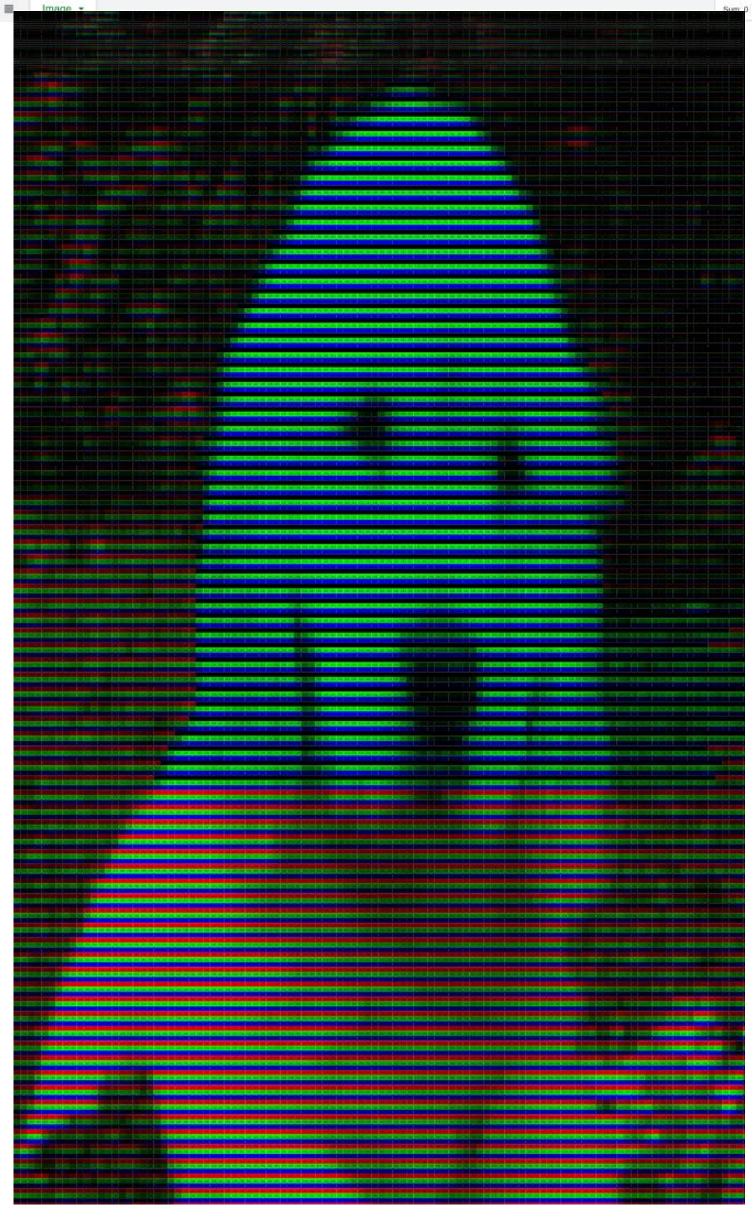
R 0 H 180 °
G 255 S 100 %
B 255 V 100 %
00FFFF



R 255 H 300 °
G 0 S 100 %
B 255 V 100 %
FF00FF

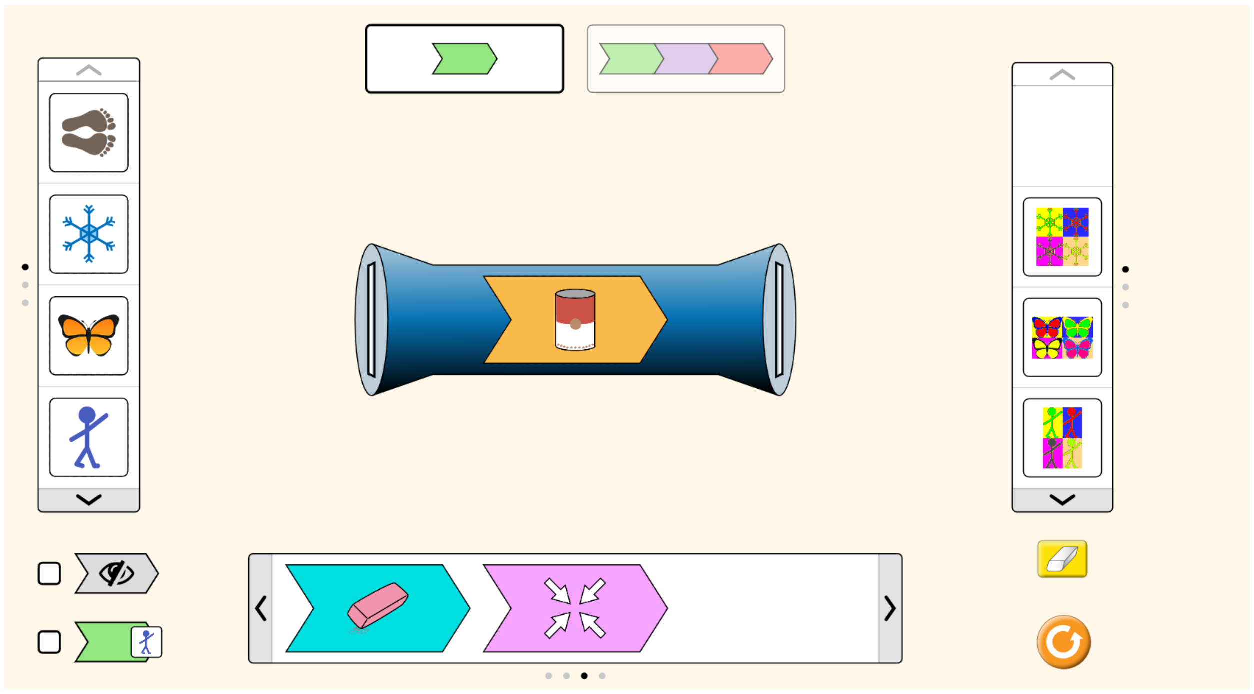
Adding a red filter to the puppy image, we can predict that setting red values equal to zero will yield a turquoise-washed look to the original image (Both zoomed-in and zoomed-out images displayed on following page):


Untitled spreadsheet
File Edit View Insert Format Data Tools Add-ons Help Saving...
Share
Image

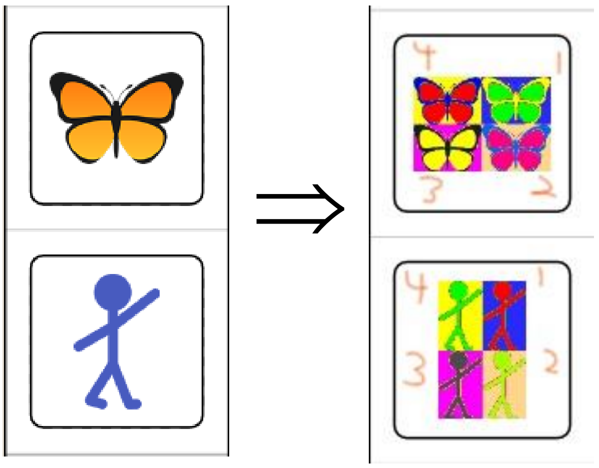


(Adding red filter to portion of “Puppy” image to display contrast)

7. The "Andy Warhol" function resizes the image into approximately 25% of original size, creates 4 duplicates, and arranges the images into 4 distinct quadrants similar to the layout for "mystery" function C. Once the layout is set up, each quadrant is manipulated using various coloring schemes/filters to create the colorful pop art effect. (*Note – not taking into account the blurred image effect popularly used by Andy Warhol where the outline of images are a bit more etched. Hypothetically, pixel manipulation would need to adjust the images against some type of threshold for this intended effect).



Staying with the same numbering and labeling scheme as "mystery" function C, four quadrants are formed with varying color schemes. Do these color schemes look vaguely familiar?



Following discussions on color/light filtering, the corresponding color values for each quadrant are manipulated to enhance the color effect, as well as the altering of wavelengths emitted from computer screens (https://jakubmarian.com/the-illusion-of-rgb-screens/). For example, observing the back ground colors for quadrants 1-4 (In order: blue, gold, magenta, and yellow), we know how to obtain three out of the four colors already. For quadrant 1, filtering out red and green (red = 0 and green = 0) and maxing blue (blue = 255) we obtain a blue background. For quadrant 2, the gold color closely resembles yellow, so by adding a blue filter (blue = 0) the red and green values will need to be adjusted. Since gold is a bit darker in intensity, we can estimate the value for red and green to be around 200 (Gold: R = 204, G = 204, B = 0). For quadrant 3, it's already been determined (from question 6) that to obtain magenta, a green filter needs to applied (Magenta: R = 255, G = 0, B = 255). And for quadrant 4, the yellow color is obtained by adding a blue filter and leaving the red and green values at 255.

The "Andy Warhol" pop art effect uses saturated colors of green, blue, yellow, and red. By adjusting the color gradients and manipulating the images (along with fuzzy outline bordering), the puppy image would need to be separated into the four quadrants, with each background set with predetermined filters, and then saturating the images with color filtering (Also, for added effect, enhanced edges with a bit of border color shading). Looking at the stick figure and butterfly originals, the "Andy Warhol" effect turned the orange wings of the butterfly to bright green, red-pink, and yellow with varying shades of blue and yellowish bordering, while the stick figure is changed from blue to magenta, yellow, green, and greenish-brown color with green, neon-yellow, and red edges.

The spreadsheet for creating this transformation would invariably entail scaling the image to roughly 0.25 * (bytes/pixels; 1024 x 2048) with some inevitable pixel overlap (C++ code would need to take into account varying pixel sizes and create a uniformed cropped image). Each pixel in an image is distinct by color along with its location, therefore functions could point to spreadsheet values for manipulating effects (we could use the C++ structure format and manipulate the values for height, width, and RGB color information for the following instructions). After the image is scaled down, it needs to separated into four quadrants by assigning image pixels to distributed arrays [rows, columns] and set to be duplicated in respective quadrants(i.e. upper-left corner, upper-right corner, bottom-right corner, and bottom-left corner). After the image is segmented into their respective quadrants, color filtering rules (for background/image) would then be applied to the specific pixels for each quadrant. Lastly, a rule/code would need to be applied to the border of the image along with creating a slightly hazy look to create the highly contrasted and "edgy" pop-art effect.

8. As mentioned in the video walkthrough for part B, reading data from a spreadsheet to a cs1300 bmp structure was conceptually the most challenging to implement. However, part_2.cc was also the most gratifying to complete and informative in its coding nature as elements of a 2D array in a spreadsheet were translated into corresponding arrays for the BMP structure. With pictures not always readily available in the .bmp format, this method of reading data from a spreadsheet is highly convenient for its intended purposes. Following the accessing of elements within a structure, along with initializing and iterating through the ROWS and COLUMNS, the part_2 code started to piece together and with the "readData(csvdata* _csv_data, string& _file_name)" function, part_2 enabled the reading of spreadsheet data, creating a BMP

structure and making it possible to manipulate the images, and then writing the images to disk. The entire implementation reminded me of our previous Python assignment using .csv data (Problem/Trouble Courses). Progressing through each of the four parts of the project and seeing how pointers were being referred through the individual “main” functions, I came to intuitively understand the TODO sections.

A separate function I found extremely helpful was the swap function used in v_flip (swap(_inImage->color[channel][row][col], _inImage->color[channel][_inImage->height - row][col]);). Having the opportunity to play around with how the swap function changes the structure of the images, along with other aspects of the code, was informative for future reference when images may need implementing and manipulating. Using pointers, structures, and images was an extremely helpful assignment, especially since the pre-written code was broken down and commented for beginning C++ users to easily digest, while demonstrating the robust and powerful nature of its many functions/tools.