Thank you for your interest in improving the Cartoonify project! This is the programmer's guide to the application. This guide pertains to the version of the program available in the .zip folder on the website.

Setup

- Windows OS version 7.0 (minimum)
- Target framework: .NET 6.0
- No additional files needed.

Libraries

- System.Drawing namespace
- System.Drawing.Imaging namespace
- System.Drawing.Drawing2D namespace
- System.Runtime.InteropServices namespace
- ColorMine.ColorSpaces from the ColorMine package by Joe Zack (https://www.nuget.org/packages/ColorMine/)

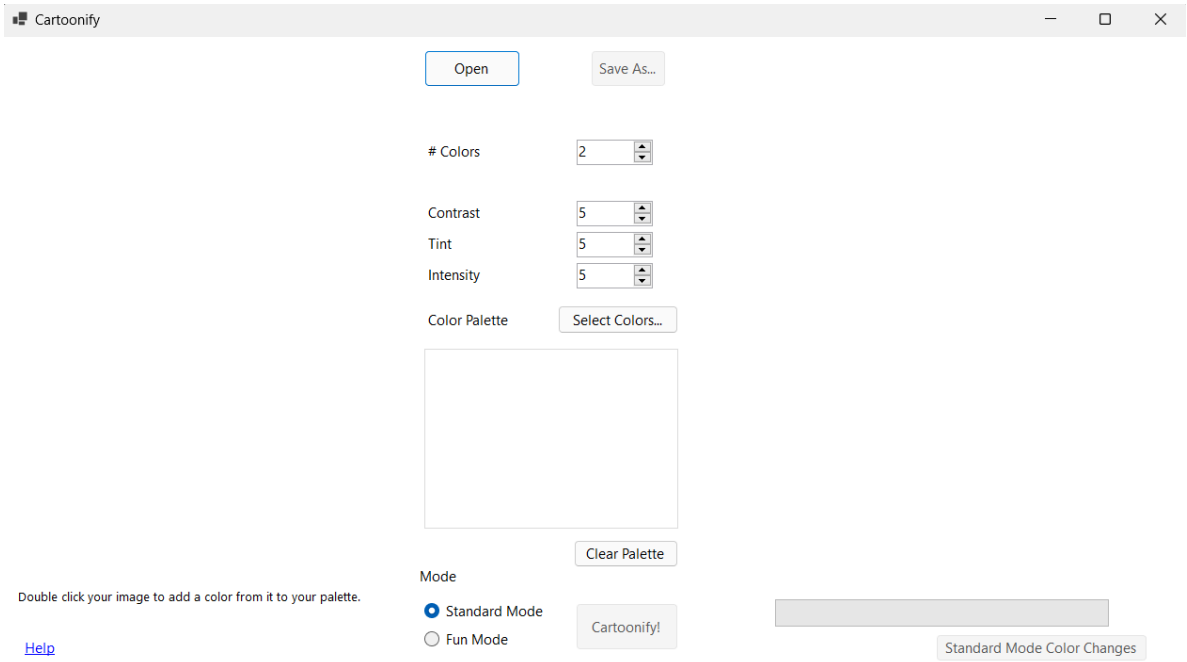
Files

- **CSharp_Test.sln** is the main solution.
    - It contains the source code for three Windows Forms.
- No specifically-named files or folders are required to run the application.
- In order to use the application for its intended purpose, an image with file type .JPG, .PNG, or .BMP is required.
- Images made by the application are saved as .BMP objects.

Form1

- The main Form of the application. Uses all of the required libraries. Contains many member variables. The ones explained here are used in the primary functions of the application.
    - **public int contrast , tint , intensity**
    - **public List<Color> colorOptions** : Stores all colors that are currently in the user's color palette. Starts empty, but has values added to it through two functions called by event handlers.
    - **public List<Color> checkedPalette** : Stores colors that are checked off in the user's palette. Starts empty, but has values added to it when the *KmeansRGB* function runs. Used to send information to Form2.

    - **public List<Color> colorMorph** : Made to store the contents of an array made in the *KmeansRGB* function. Starts empty. Used to send information to Form2.
    - **public int maxColorTransfer** : Stores the number of buckets (groups) from the most recent cartoonification. Starts empty, but has a value added to it when the *KmeansRGB* function runs. Sent to Form2 to be used as a stopping point for a *for* loop.
    - **public int pixelCount** : Stores the number of pixels in the uploaded image. Starts empty, but has a value added to it when the *KmeansRGB* function runs. Sent to Form2 to calculate averages.
    - **public List<int> bucketCounts** : Stores the number of pixels that each bucket appears in inside the cartoon. Starts empty, but has a value added to it when the *KmeansRGB* function runs. Sent to Form2 to calculate averages.
    - **public Pen boxPen**
    - **public SolidBrush boxBrush**
    - **Random random** : Initializes RNG seed for use in the Fun Mode of the application. Used in the *KmeansRGB* function under certain conditions.
- The appearance of the Form when first opened:



- The following are elements of Form1's window. The ones that are used by the primary functions will be explained.
    - **selectedPicture** : a PictureBox object that displays the user's uploaded image. Starts empty. Display mode is set to StretchImage; this is done to prevent the image from appearing under the UI.

- **cartoon** : a PictureBox object that displays the resulting cartoon. Starts empty. Display mode is set to StretchImage; this is done to prevent the image from appearing under the UI.
- **openFileButton** : a Button object that opens an OpenFileDialog. The OpenFileDialog allows the user to upload a .JPG, .PNG, or .BMP file from anywhere on their device. This button is normally enabled, but it temporarily disables when the program is processing an image.
- **numColorsLabel** : a Label object that serves to label the box where the user can adjust the number of colors that will appear in the cartoon.
- **contrastLabel**
- **tintLabel**
- **intensityLabel**
- **saveButton** : a Button object that opens a SaveFileDialog that allows the user to save the Image object stored in *cartoon*. This button starts disabled but becomes enabled when a cartoon is created. It will also be disabled while the program is processing an image.
- **specColorsLabel**
- **selectColors**
- **cartoonify** : a Button object that runs an image processing function when clicked.
- **contrastHelp**
- **contrastBox**
- **tintHelp**
- **tintBox**
- **intensityHelp**
- **intensityBox**
- **photoNameLabel**
- **numColorsBox** : a NumericUpDown object where users can set how many colors will appear in the cartoon. Minimum and default value is 2, while the maximum value is 16. The value inside this box can also be changed by certain functions within the Form. The value is also used by the main image processing algorithm.
- **colorProgBar** : a ProgressBar object that gives users a visual indicator of how far along the application is inside image processing.
- **eyedropperInfoLabel**
- **modeLabel**
- **modeHelp**
- **modePanel** : a Panel object that contains *modeRadioButton* and *modeRadioButton2*.

    - **modeRadioButton2** : a RadioButton object that, when checked, enables the “Fun Mode” of cartoonification for the user. Part of *modePanel*.
    - **modeRadioButton** : a RadioButton object that, when checked, enables the “Standard Mode” of cartoonification for the user. Part of *modePanel*. This radio button starts checked by default.
    - **gbColorPalette** : a GroupBox object that contains the user’s color palette. It starts with no controls inside, but certain functions within the Form add CheckBox controls to this GroupBox.
    - **kMeansButton** : a Button object that sends data to Form2 and opens Form2 when clicked. It starts disabled and is only enabled under certain conditions.
    - **helpLink** : a LinkLabel object that opens Form3 (the Help menu) when clicked.
    - **progBarLabel** : a Label object located underneath colorProgBar that is intended to tell the user which stage of image processing the application is on.
    - **button1** : This unlabeled button is not used by the primary functions of the algorithm, but it is used to clear out the user’s color palette.
- The following are the functions inside Form1. The primary functions will be explained.
    - public **Default constructor** (no parameters)
    - private void **Save_Button_Click** (object sender, EventArgs e)
    - private void **selectedPicture_DblClick** (object sender, EventArgs e)
    - private void **openFileButton_Click** (object sender, EventArgs e)
    - private void **cartoonify_Click** (object sender, EventArgs e)
    - private void **selectColors_Click** (object sender, EventArgs e)
    - private void **kMeansButton_Click** (object sender, EventArgs e)
    - private void **paletteClearBtn_Click** (object sender, EventArgs e)
    - private float **EuclideanRGB** (Color c1, Color c2) :
        - Called by *KmeansRGB*.
        - Color *c1* represents the palette color that the process is currently on; Color *c2* represents the current pixel color.
        - Defines a float variable called *avg*, and three double variables called *RCalc*, *GCalc*, and *BCalc*.
        -

```
RCalc = Math.Abs((c1.R - c2.R)^2);
GCalc = Math.Abs((c1.G - c2.G)^2);
BCalc = Math.Abs((c1.B - c2.B) ^ 2);
```

- The resulting *RCalc*, *GCalc*, and *BCalc* are then added together (done first), and then divided by 3 to find *avg*. The result is typecast as a float, so it can be assigned to *avg*.
- Returns *avg*.

- public void **KmeansRGB** (bool modeTag) :
  - *modeTag* is *true* when *cartoonify_Click* finds that *modeRadioButton* is Checked; *false* when *cartoonify_Click* finds that *modeRadioButton2* is Checked. (It is *true* if the user is running Standard Mode and *false* if running Fun Mode.) This affects how the cartoon is created near the end of this function.
  - Disables the *openFileButton*, *saveButton*, and *cartoonify* buttons.
  - Creates several objects and variables:
    - Bitmap **pic** = new Bitmap(*selectedPicture*.Image)
    - int **maxX** = *pic*.Width
    - int **maxY** = *pic*.Height
    - int[,] **cBuckets** = new int[*maxX*, *maxY*]
      - Parallel to the image's height and width. Each entry represents a pixel, and it holds an int representing the bucket that pixel was placed into.
      - Used to color cartoon at the end.
    - List<Color> **selColors** = new List<Color>()
      - Grabs colors from Checked items in *gbColorPalette*.
      - The int *paletteIndex* is used in the *foreach* loop that adds colors to this List. The loop goes through each Color inside *gbColorPalette*'s controls, and if the color is checked, then the color in *colorOptions* at index *paletteIndex* is added to *selColors*.
    - Color[] **selColorsArray** = new Color[*selColors*.Count]
      - Contents of *selColors* added to this array.
    - int **numClusters** = (int)*numColorsBox*.Value
  - Checks if *selColors* is less than 2, if *selColors.*Count is greater than *numClusters*, or if *selColors*.Count is less than *numClusters*. If any of these three are true, a MessageBox with the appropriate error message appears, re-enables the three buttons that were disabled, resets *progBarLabel*, and the function returns.
  - Creates parallel arrays and Lists that rely on *numClusters*:
    - List<Color>[] **buckets** = new List<Color>[*numClusters*]
      - Stores the pixel colors inside each bucket.
    - int[] **bucketCount** = new int[*numClusters*]
      - Stores how many pixels are in each bucket.

        - Color[] **targetAvgs** = new Color[*numClusters*]
            - Intended to be used to check best fit in subsequent iterations.
        - Color[] **prevAvgs** = new Color[*numClusters*]
            - Keeps track of the previous averages for the next iteration of the main loop.
            - When initialized, the contents of *selColorsArray* are copied into it. (It starts with the palette colors stored.)
- Creates variables used for the main loop:
    - float **avg**
    - float **min** = float.MaxValue
    - int **bucketIndex** = -1
    - int **numPasses** = 0
    - float **avgR** = 0; float **avgG** = 0; float **avgB** = 0;
    - int **sumR** = 0; int **sumG** = 0; int **sumB** = 0;
    - bool **bucketChange** = true
    - int **maxPasses** = 7
        - Can be set to 3 if *pic*'s Height or Width are above a certain value. This is done to reduce processing time.
    - *colorProgBar*'s Maximum is set to *pic*.Width * (*maxPasses* + 1).
- The main *while* loop runs; its process is described in the **K-Means Algorithm Explanation** section.
- After the *while* loop finishes:
    - Checks if *modeTag* is *false*.
        - If *false* (user is in Fun Mode), swaps colors associated with each bucket randomly. The *random*.Next call uses 0 as the min and *numClusters* as the max.
    - Creates the cartoon using *cBuckets* and a nested *for* loop.
        - If *modeTag* is *true* (the user is running Standard Mode), the pixel at (*x*, *y*) is colored using the Color stored in *MinAvgs* at the index indicated by the value of *cBuckets*[*x*,*y*]. (Standard Mode uses the updated averages found during the K-Means algorithm to color the cartoon.)
        - If *modeTag* is *false* (the user is running Fun Mode), the pixel at (*x*, *y*) is colored using the Color stored in *selColorsArray* at the index indicated by the value of

cBuckets[*x*,*y*]. (Fun Mode preserves the original palette colors in the cartoon.)

- Checks if *modeTag* is *true*.
  - If it is *true*, sets up the Form1 member variables *pixelCount*, *bucketCounts*, *colorMorph*, *checkedPalette,* and *maxColorTransfer* to send to Form2.
    - *pixelCount* is set using *pic*.Width * *pic*.Height.
    - *bucketCounts*, *colorMorph*, and *checkedPalette* are cleared out before being set up in a *for* loop that runs with *i* < *numClusters* as its condition.
      - *checkedPalette* is set up with *selColorsArray*'s contents.
      - *colorMorph* is set up with *prevAvg*'s contents.
      - *bucketCounts* is set up with the contents of *bucketCount*.
    - *maxColorTransfer* is set equal to *numClusters*.
  - Enables *kMeansButton* ("Standard Mode Color Morph" button on the application that opens Form2).
- Resets the progress bar.
- Sets *cartoon*'s Image as *pic*. (At this point, *pic* has been altered.)
- Runs *applyContrast()*, *applyTint()*, and *applyIntensity()* in that order.
- Sets *progBarLabel*'s text to "Done!" to tell the user that the cartoon is complete.

- private void **helpLink_LinkClicked** (object sender, LinkLabelLinkClickedEventArgs e)
- public void **applyContrast** ()
- public void **applyTint** ()
- public void **applyIntensity** ()

K-Means Algorithm Explanation

This function, alongside EuclideanRGB, will be available in a separate document.

Refer to KmeansRGB's description in the Form1 functions section for an explanation of the variables and data structures needed.

K-Means is a grouping algorithm that groups similar numbers together and updates groups (buckets, clusters) based on the averages (means) of the elements of each group. Color objects consist of multiple ints representing red, green, and blue values, so they can be used with this algorithm.

- All inside a *while* loop that runs until the maximum number of passes is reached, or if none of the averages change.
    - Before the nested *for* loops, set *bucketChange* to *false*.
    - Nested *for* loops; one is (int x = 0; x < pic.Width; x++) while the inner loop is (int y = 0; y < pic.Height; y++)
        - Creates *pixelColor* using the color of the pixel at (x, y).
        - Creates Color *c* with a temporary value; this is used in the following loop that compares pixel color to colors in *prevAvgs* (which stores the user's palette colors on the first pass, and the average of each pixel in each bucket on subsequent passes), and finds the closest bucket color to the current pixel
        - for (int i = 0; i < *numClusters*; i++)
            - *avg* is set equal to the result of calling *EuclideanRGB* with *prevAvgs*[i] and *pixelColor* as parameters.
            - *avg* is then compared to *min*
                - If *avg* < *min*
                    - *min* is set equal to *avg*
                    - *c* is set to the Color at index i
                    - *bucketIndex* is set to i (this bucket represents the one that the pixel is currently closest to)
                    - *bucketChange* is set to *true* (an average changed)
        - The pixel color is added to the bucket at *bucketIndex*. The amount in the *bucketCount* array at *bucketIndex* is also incremented by 1. *bucketIndex* also becomes the value in *cBuckets*[x, y].
        - Before moving on to the next pixel, *bucketIndex* and *min* are reset.
    - After going through pixel by pixel in the current pass:
        - for (int i = 0; i < *numClusters*; i++) loop that finds the average of the pixel colors in each bucket.
            - In a *foreach* loop that goes through the Colors in *buckets[i]* , the *sumR, sumG,* and *sumB* are found by adding Color *c*'s R, G, and B values to their respective sums.
            - If the *bucketCount[i]* is not 0, then each of *sumR, sumG,* and *sumB* are divided by the value of *bucketCount[i]*. These values are set into the variables *avgR, avgG*, and *avgB*. If

any of the three values end up being above 255, the respective value will be set to 255, and if any of them end up being below 0, then the respective value is set to 0. This is to prevent exceptions in the next step.
        - A new Color object is created using *avgR, avgG*, and *avgB*. This Color object is set into *targetAvgs[i]*.
        - Before the next iteration of this inner loop, *sumR, sumG,* and *sumB* are reset to 0.
    - The contents of *targetAvgs* are copied into *prevAvgs* using a loop.
    - *numPasses* increments.
    - If conditions are met to run the *while* loop again for another pass of the algorithm, then the contents of *buckets* are cleared and the *bucketCounts* array is zeroed out.
        - The contents of these two arrays will remain if this is the last pass of K-Means, as they are used in other areas of the application.

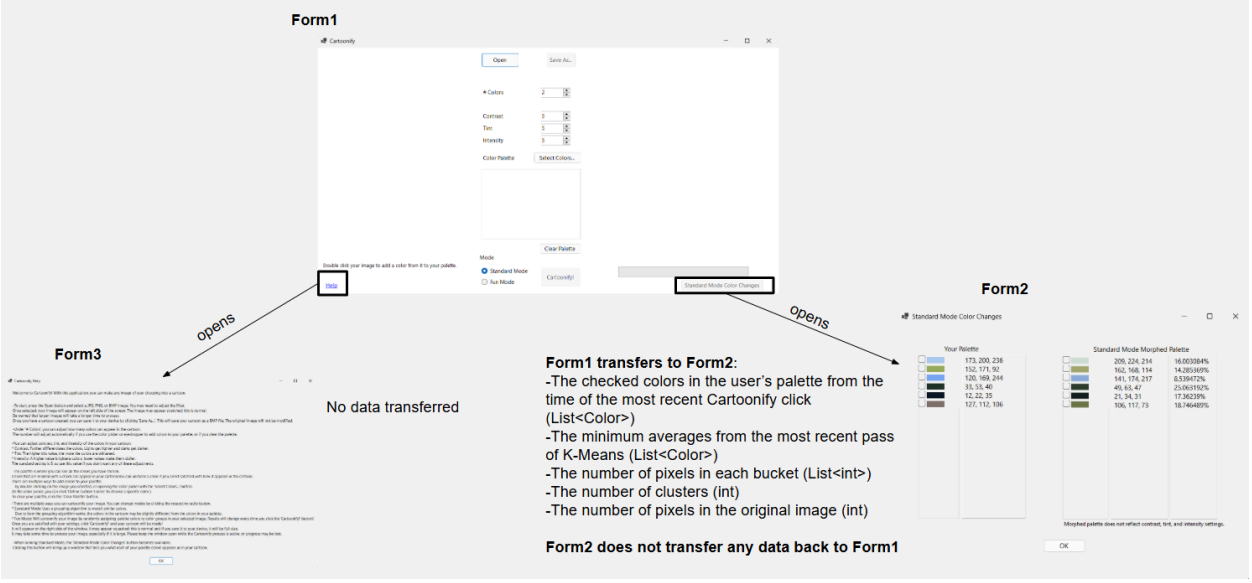
Form2

- This Form can be opened by clicking on the “Standard Mode Color Changes” button on the main Form.
- It has 7 member variables, 5 of which are used to store received data from Form1. More information is in the **Data Flow** section. The other two member variables are Pen and SolidBrush objects used to draw rectangles to the screen.
- The primary function of this Form is to display the user’s checked palette colors and their RGB values to compare them to the final averages from the K-Means algorithm. The rectangles of those colors and their RGB values are displayed on the right side of the Form.
    - This Form also displays the percentages of how often each color bucket appears in the user’s cartoon. These are found by dividing the number of pixels in each bucket by the total number of pixels, then multiplying by 100. If there are 0 pixels in a bucket, then the percentage displays 0.
    - The code used to draw colored rectangles to the screen is originally used in two functions in Form1. The CheckBox objects are a holdover from this and don’t serve any function in this Form.
- While this Form is open, Form1 can still be interacted with. This is intended as a supplementary window for the user, so they can see how colors shifted during the K-Means algorithm in Standard Mode.

Form3

- This Form is the Help window. It can be opened by clicking the LinkLabel object labeled “Help” on the bottom left of Form1.
- Form3 contains 7 Label objects whose contents are initialized upon opening, as well as one Button object that closes the Form.
- Form3 does not send or receive information from any of the other two Forms.

Data Flow



Possible Extensions

- As another form of palette control, allowing the user to remove colors from the palette individually.
- Similar to how the ‘# Colors’ counter increments whenever a color is added via eyedropper or color picker, this counter should increment or decrement whenever the user checks or unchecks a color in the palette.
    - The CheckBox objects that make up the color palette’s controls are created when the color is added, so they are not defined as part of the form. Otherwise, it could be handled with event handlers.
- Allowing the user to run additional passes of K-Means while in Standard Mode, and also letting them set how many additional passes.

    - I believe that this could be achieved with several more variables. *KmeansRGB* would take an additional boolean parameter that would be *true* if the image has already been cartoonified with the same settings in Standard Mode. This boolean parameter would be a global variable, and it would be set to *false* if the user adjusts settings, uploads a new image, or switches to Fun Mode and then cartoonifies again.
    - The colors shifted by the K-Means algorithm are saved in a global variable in order to be sent to Form2, so this variable could be used for the starting *MinAvgs* array if the above-mentioned boolean parameter is *true*.
- Adding threading so that the application does not seize up while processing larger images. This would also allow the progress bar and status messages to update as intended.
    - This may require an update to the minimum .NET framework.
- Changing the file type that resulting cartoons are saved as.
    - .BMP files may take up more file size than expected for larger images. A change in the file type that resulting cartoons are saved as may reduce the file size but keep the quality.

My Research Sources

- **For coding:**

- Windows apps in C# : https://www.geeksforgeeks.org/introduction-to-c-sharp-windows-forms-applications/
- Image viewer in C# : https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-windows-forms-picture-viewer-controls?view=vs-2022
- Picture box & color dialog basics: https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-windows-forms-picture-viewer-code?view=vs-2022&tabs=csharp
- Color dialog: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.colordialog?view=windowsdesktop-9.0
- Drawing rectangles: https://learn.microsoft.com/en-us/dotnet/api/system.drawing.graphics.drawrectangle?view=windowsdesktop-9.0
- TextBox class: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.textbox?view=windowsdesktop-9.0

- NumericUpDown class: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.numericupdown?view=windowsdesktop-9.0
- Image handling: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.imagelist?view=windowsdesktop-9.0
- System.Drawing namespace: https://learn.microsoft.com/en-us/dotnet/api/system.drawing?view=windowsdesktop-9.0
- Save file dialog: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/how-to-save-files-using-the-savefiledialog-component?view=netframeworkdesktop-4.8
- System.Drawing.Drawing2D namespace: https://learn.microsoft.com/en-us/dotnet/api/system.drawing.drawing2d?view=windowsdesktop-9.0
- Converting from HSV back to RGB color + ColorMine package: https://stackoverflow.com/questions/1335426/is-there-a-built-in-c-net-system-api-for-hsv-to-rgb
- Dynamic Lists in C#: https://stackoverflow.com/questions/20451747/getting-a-value-from-dynamic-list
- Progress Bars: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.progressbar?view=windowsdesktop-9.0
- Mouse event handler: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.mouseeventhandler?view=windowsdesktop-9.0
- Grouping radio buttons: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/how-to-group-windows-forms-radiobutton-controls-to-function-as-a-set?view=netframeworkdesktop-4.8
- GroupBox objects: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.groupbox?view=windowsdesktop-9.0
- 2D arrays: https://www.w3schools.com/cs/cs_arrays_multi.php
- Making an array of Lists: https://stackoverflow.com/questions/7464724/an-array-of-list-in-c-sharp
- Opening a second Form: https://stackoverflow.com/questions/5718183/how-to-open-the-second-form
- Passing data to another Form: https://www.c-sharpcorner.com/UploadFile/834980/how-to-pass-data-from-one-form-to-other-form-in-windows-form/

https://www.c-sharpcorner.com/UploadFile/009464/pass-data-from-one-form-to-other-using-properties-in-C-Sharp/

- Marshal.Copy method: https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.marshal.copy?view=net-9.0&redirectedfrom=MSDN#overloads

- **Adjusting contrast:**
  - https://efundies.com/adjust-the-contrast-of-an-image-in-c/
  - https://softwarebydefault.com/2013/04/20/image-contrast/

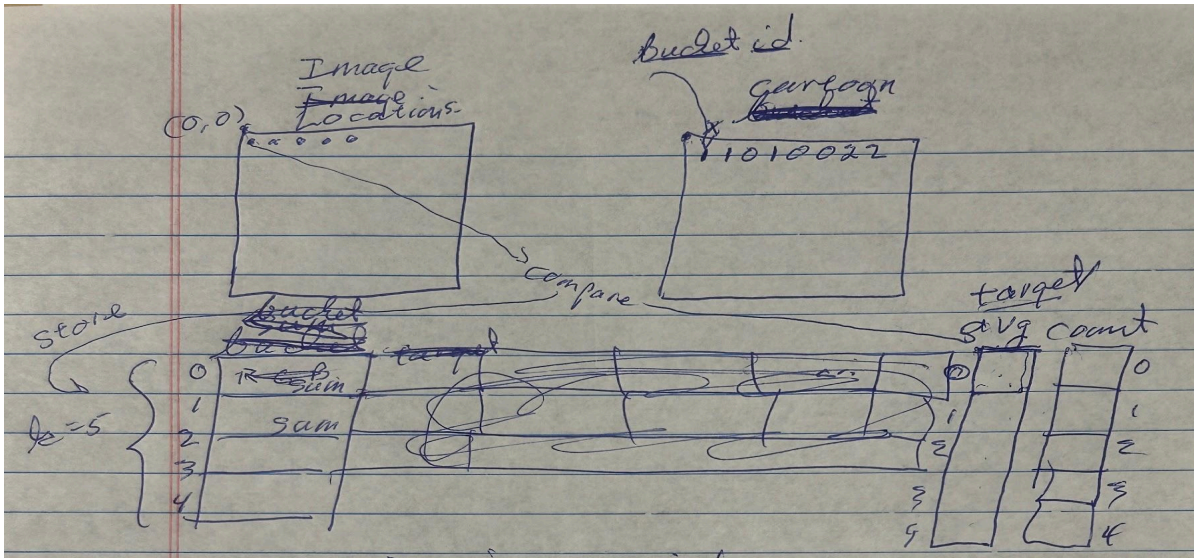
- **Sources used for elements that did not appear in final project:**
  - Progress bar with threads: https://www.c-sharpcorner.com/article/code-for-progressbar-in-windows-application-using-c-sharp-net/
  - Passing by reference: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/ref
  - System.Math namespace: https://learn.microsoft.com/en-us/dotnet/api/system.math?view=net-9.0
  - Dynamic arrays: https://www.csharp.com/article/what-are-c-sharp-dynamic-arrays/ https://www.geeksforgeeks.org/object-and-dynamic-array-in-c-sharp/
  - Converting to HSV: https://www.youtube.com/watch?v=Kmlqxmn980g https://www.rapidtables.com/convert/color/hsv-to-rgb.html
  - Clipping with a polygonal region: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/advanced/how-to-use-clipping-with-a-region?view=netframeworkdesktop-4.8
  - Other image recoloring methods using matrices: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/advanced/recoloring-images?view=netframeworkdesktop-4.8
  - Mouse hover events: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.control.mousemove?view=windowsdesktop-9.0#system-windows-forms-control-mousemove
  - CheckedListBox object: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.checkedlistbox?view=windowsdesktop-9.0

    - Drawing rectangles inside the CheckedListBox object: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.drawmode?view=windowsdesktop-9.0
    - Applying color to text: https://stackoverflow.com/questions/3966503/c-sharp-apply-color-to-font

- **K-Means algorithm:**

    - https://www.geeksforgeeks.org/k-means-clustering-introduction/
    - https://visualstudiomagazine.com/articles/2013/12/01/k-means-data-clustering-using-c.aspx
    - https://visualstudiomagazine.com/Articles/2023/12/01/k-means-data-clustering.aspx?Page=2
    - https://www.codeproject.com/Articles/985824/Implementing-The-K-Means-Clustering-Algorithm-in-C
    - RGB distance between two colors: https://dev.to/bytebodger/determining-the-rgb-distance-between-two-colors-4n91

- The following two pages contain notes on the K-Means algorithm from Dr. Pankratz (my primary resource while coding):



Display

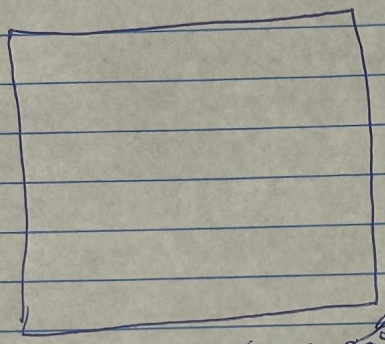
```
for(
  for(
    color[ cartoon[x,y] ]
    color[ avg[ cartoon[x,y] ] ]
  }
}
```

Fix m colors as targets. m <= k

Steps: set new targets original.

```
newtarget[i] = target[i]
                i < m
else
newtarget[i] = avg[i].
```

k means



Step -1: create a ~~set~~ k buckets to save closest ~~pixels~~ colors

Step 0: pick k target colors (user or random)

~~Step 1: Find the "sum" of diff for each~~

Step 1: Find distance from ~~each~~ first pixel (0,0) color (RGB) to each of k target colors.

Step 2: ~~Save~~ compute the min ~~in t~~ and assign it to the correct bucket. (keep count of # of colors in each bucket.

Step 3: Find the avg of the colors in each bucket.

Step 4: test ~~to~~ for goodness of fit.
(avgs from this iteration didn't chg much from previous iteration?
(N times is good enough? (chg N ~~it~~ that

Step 5
Set avgs in buckets to new targets

Step 6: Go to Step 1

Display: ~~for each pixel~~