Cairo University
Faculty of Computers and Artificial Intelligence
CS213– Object Oriented programming

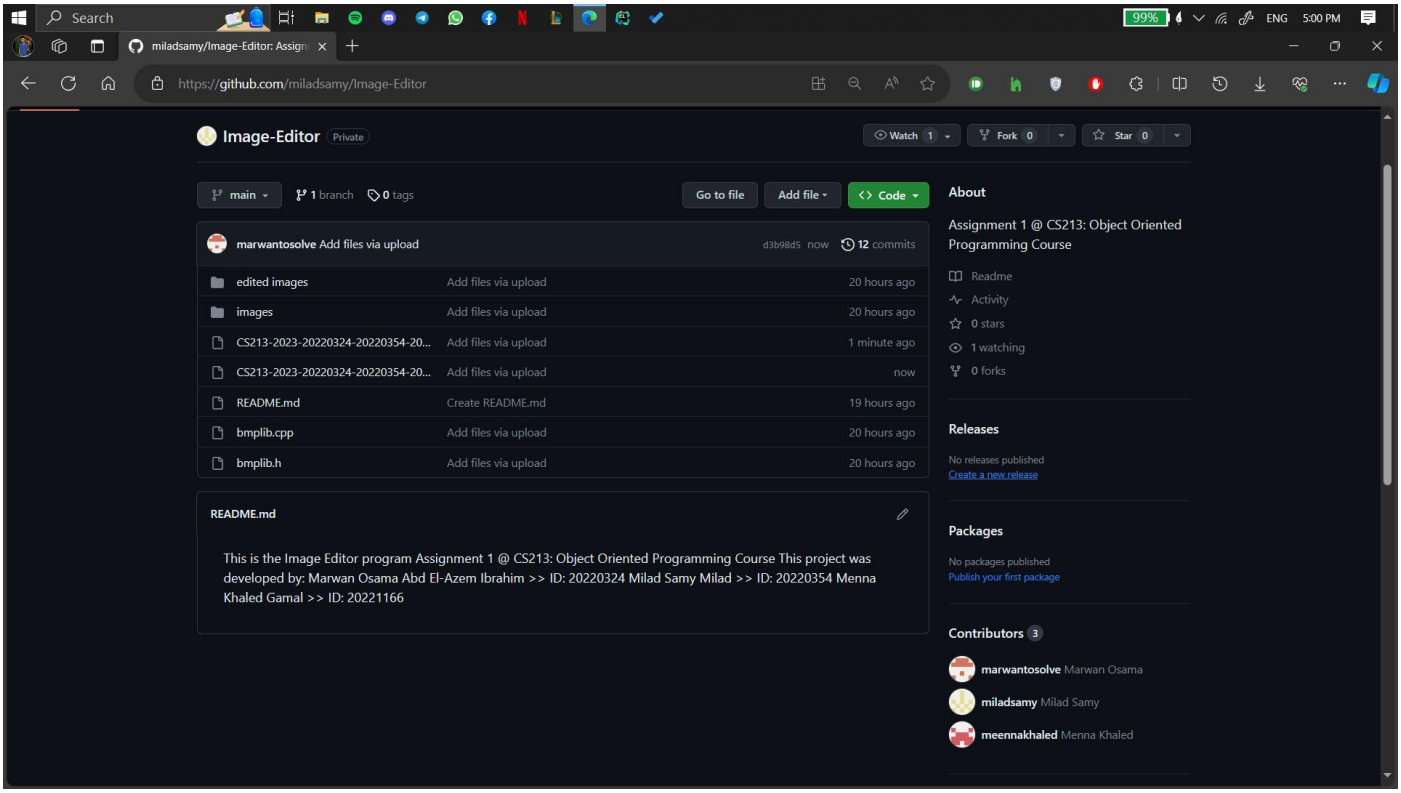
Assignment 1

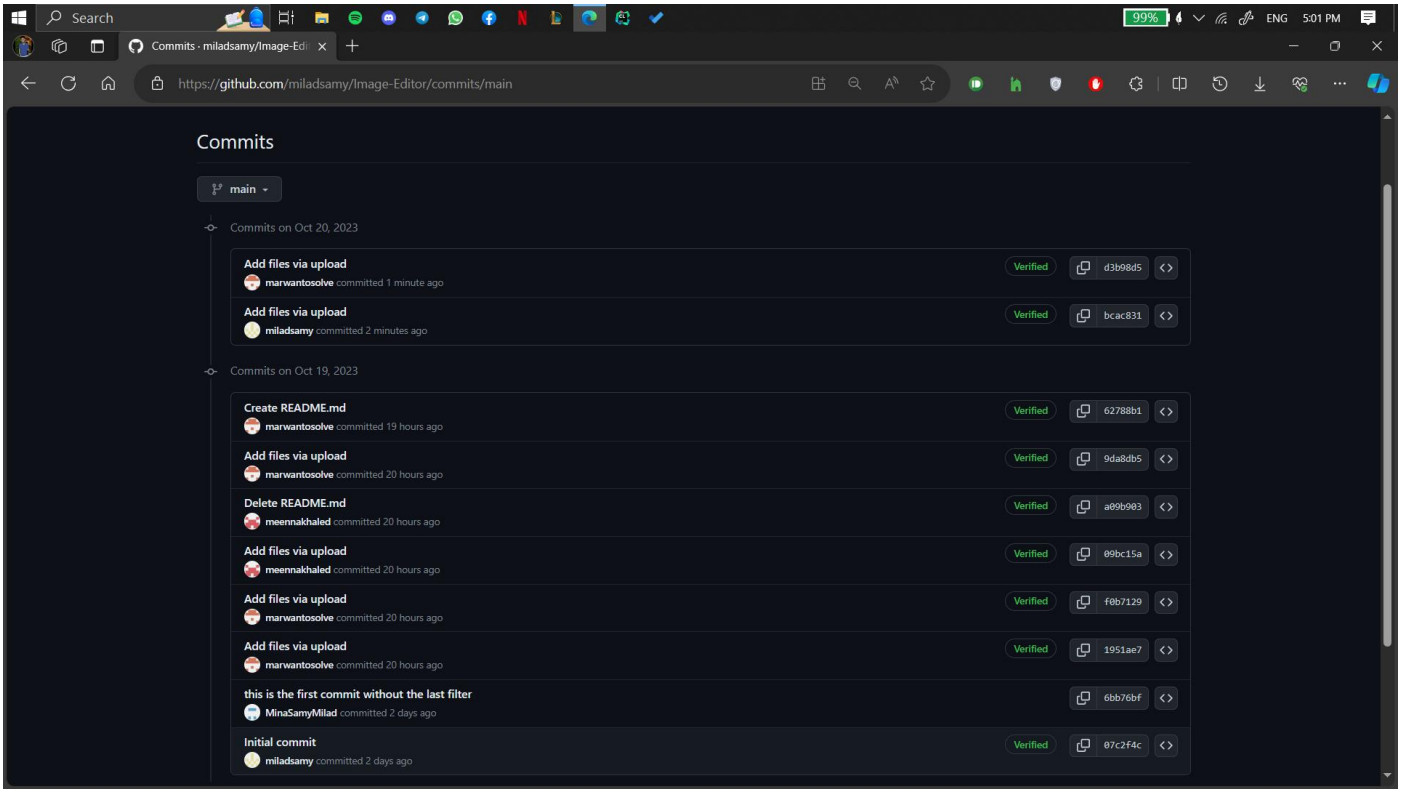
Image Editor

Name	ID
Marwan Osama Abd El-Azem	20220324
Milad Samy Milad	20220354
Menna Khaled Gamal	20221166

First:

• A screenshot showing using Git workflow and the collaboration on the project:





• As you can see the contributions, commits and we uploaded the whole project after we did it together as a great teamwork.

• Link: https://github.com/miladsamy/Image-Editor

Second:

• The algorithms of the filter's functions with steps:

• Black and White filter:

The function named `BW` performs thresholding to convert an image to black and white. The algorithm used in this code is very straightforward:

1. Loop through each pixel in the image, where `SIZE` likely represents the dimensions of the image (assuming a 2D array structure for `image`).

2. For each pixel, compare its intensity value with a threshold value of 127 (the line `if (image[i][j] > 127)`).

3. If the pixel's intensity is greater than 127, set the pixel's value to 255 (white), effectively making it a white pixel.

4. If the pixel's intensity is less than or equal to 127, set the pixel's value to 0 (black), making it a black pixel.

This algorithm essentially converts an image to black and white by thresholding pixel intensities. Pixels brighter than the threshold become white, and pixels darker become black. This is a simple form of image processing and thresholding used in computer vision and image analysis applications.

• Invert filter:

The function named `INVERT`, which performs image inversion. The algorithm used in this code is quite simple:

1. It loops through each pixel in the image, with `SIZE` likely representing the image's dimensions (assuming a 2D array structure for `image`).

2. For each pixel, it calculates the negative of the pixel's value by subtracting the current pixel value from the highest possible value, which is 255 (the line `image[i][j] = 255 - image[i][j]`).

This operation effectively inverts the pixel values. Pixels that were originally dark become light, and pixels that were originally light become dark. It's a common image processing technique used for various purposes, including enhancing image contrast or creating special visual effects.

• Merge Filter:

The `MERGE` function merges two grayscale images. Here's a brief explanation of the algorithm:

1. It prompts the user to enter the name of the image file to merge with. The user is expected to enter the name of the image file (excluding the extension) that should be located in the "images" folder.

2. It appends the ".bmp" extension to the entered filename, ensuring that it's a BMP image file.

3. It reads the second image from the specified BMP file using the `readGSBMP` function and stores the pixel values in the `image2` array.

4. It merges the two images by taking the average of corresponding pixel values. For each pixel at position `(i, j)` in both `image` and `image2`, it computes the average of the pixel values and assigns this average value to the corresponding pixel in the `image` array.

The merging process involves taking the average of pixel values from both images, effectively blending them together. This can create interesting visual effects or combine information from two images.

• Filb filter:

The `FLIB` function allows the user to flip an image either horizontally (h) or vertically (v). Here's a brief explanation of the algorithm:

1. It prompts the user to enter whether they want to flip the image horizontally or vertically. The user can enter 'h' for horizontal flip or 'v' for vertical flip.

2. It reads the user's choice into the variable `flib`.

3. If the user chose 'v' (vertical flip), the function performs the following steps:

   - It loops through the top half (up to `SIZE/2`) of the image, and for each row (`i`) and column (`j`), it swaps the pixel values at `(i, j)` with the pixel values at the corresponding position in the bottom half of the image (`SIZE - i - 1`).

   - This effectively flips the image vertically by reversing each column in the 2D array.

4. If the user chose 'h' (horizontal flip), the function performs the following steps:

   - It loops through all rows in the image, and for each row (`i`) and the left half (up to `SIZE/2`) of the columns (`j`), it swaps the pixel values at `(i, j)` with the pixel values at the corresponding position in the right half of the image (`SIZE - j - 1`).

   - This effectively flips the image horizontally by reversing each row in the 2D array.

5. If the user enters an invalid choice, an error message is displayed.

The function provides a simple way to flip an image horizontally or vertically, which can be useful for various image manipulation tasks.

• Darken and Lighten filter:

The `DL` function allows the user to either darken or lighten an image. Here's a brief explanation of the algorithm:

1. It prompts the user to enter whether they want to darken (d) or lighten (l) the image.

2. It reads the user's choice into the variable `dl`.

3. If the user chose 'd' (darken), the function performs the following steps:

- It loops through each pixel in the image.
- For each pixel, it reduces the pixel value by 50% (multiplies by 0.5) to darken it.
- It also ensures that the pixel value stays within the valid range of 0 to 255. If the pixel value becomes less than 0, it's set to 0.

4. If the user chose 'l' (lighten), the function performs the following steps:

- It loops through each pixel in the image.
- For each pixel, it increases the pixel value by 50 (adding 50) to lighten it.
- It also ensures that the pixel value does not exceed the maximum value of 255.

5. If the user enters an invalid choice, an error message is displayed.

The function provides a simple way to adjust the brightness of an image by either darkening or lightening it. It's important to note that the choice to lighten the image has an upper limit of 255 to prevent pixel values from exceeding the maximum value allowed in an 8-bit grayscale image.

• Rotate filter:

The `ROTATE` function allows the user to rotate an image by 90, 180, 270, or 360 degrees. Here's a brief explanation of the algorithm for each rotation:

1. If the user chooses to rotate the image by 90 degrees:

- The function transposes the 2D array, which effectively swaps rows and columns.

- It reverses each column in the transposed array to complete the 90-degree rotation.

2. If the user chooses to rotate the image by 180 degrees:

- The function reverses each row and column in the 2D array to achieve the 180-degree rotation.

3. If the user chooses to rotate the image by 270 degrees (which is essentially 90 degrees in the opposite direction):

- The function transposes the 2D array, swapping rows and columns.

- It reverses each row in the transposed array to complete the 270-degree rotation.

4. If the user chooses to rotate the image by 360 degrees, no changes are made to the image, as a 360-degree rotation has no effect.

5. If the user enters an invalid choice, an error message is displayed.

The `ROTATE` function provides a way to perform simple image rotations, which can be useful for various image processing tasks. It uses basic array manipulations to achieve the desired rotation angles.

• Detect edges filter:

The `EDGES` function is designed to detect edges in an image using the Sobel operator and then invert the colors of the resulting edge-detected image. Here's a brief explanation of the algorithm:

1. Create a copy of the original image called `image2` to preserve the original data. This ensures that the edge detection process does not modify the original image.

2. Copy the pixel values from the original image to the copy `image2` to have an identical starting point.

3. Apply the Sobel operator to detect edges in the image. The Sobel operator calculates gradients in both the x and y directions to highlight edges. The process is performed in the following steps:

- Iterate through the inner region of the image (excluding the border pixels) using nested loops.
- Calculate gradient values in both the x and y directions for each pixel using neighboring pixels.
- Compute the magnitude of the gradient as the absolute sum of the x and y gradients.
- Update the pixel in the original image with the calculated magnitude value. This process enhances the edges in the image.

4. After detecting the edges, the `INVERT` function is called to invert the colors of the edge-detected image. This step can be useful for better visualization of the edges, as dark edges are typically displayed on a light background.

The result is an image that highlights the edges with enhanced contrast, and the colors are inverted for better visibility. This is a common technique used in image processing and computer vision for edge detection and feature extraction.

• Enlarge filter:

The `ENLARGE` function allows the user to select and enlarge one of the four quarters of an image. Here's a brief explanation of the algorithm:

1. It prompts the user to choose which quarter of the image to enlarge (1, 2, 3, or 4).

2. It reads the user's choice into the variable `quarter`.

3. Create a copy of the original image called `image2` to preserve the original data.

4. Depending on the user's choice of quarter, the function enlarges the selected quarter while keeping the rest of the image unchanged. The process is performed as follows:

- If the user selects quarter 1, it loops through each pixel in the image and assigns the corresponding pixel value from `image2` to the quarter. This effectively enlarges the first quarter.

- If the user selects quarter 2, it enlarges the second quarter, moving the pixel values from `image2` accordingly.

- If the user selects quarter 3, it enlarges the third quarter.

- If the user selects quarter 4, it enlarges the fourth quarter.

5. If the user enters an invalid choice, an error message is displayed.

The `ENLARGE` function allows the user to focus on and enlarge a specific area of the image, which can be useful for detailed analysis or visualization.

• Shrink filter:

The `SHRINK` function allows the user to shrink an image by a specified ratio (1/2, 1/3, or 1/4). Here's a brief explanation of the algorithm:

1. It initializes a character array `image2FileName` and attempts to concatenate it with ".bmp." However, the array is not initialized with any value in the provided code, which might lead to unpredictable behavior.

2. It creates a copy of the original image called `image2` to preserve the original data.

3. It sets all pixel values in the original image to 255 (white).

4. It prompts the user to specify the shrink ratio (1/2, 1/3, or 1/4) and reads the choice into the variable `ratio`.

5. Depending on the user's choice of shrink ratio, the function shrinks the image by selecting every second, third, or fourth pixel while copying the pixel values from `image2` to `image`. The process is performed as follows:

- If the user selects a ratio of "1/2," the function selects every second pixel and copies their values to shrink the image to half of its original size.

- If the user selects a ratio of "1/3," the function selects every third pixel to shrink the image to one-third of its original size.

- If the user selects a ratio of "1/4," the function selects every fourth pixel to shrink the image to one-fourth of its original size.

6. If the user enters an invalid choice, an error message is displayed.

The `SHRINK` function allows the user to reduce the size of the image while preserving the relative arrangement of pixels based on the specified shrink ratio.

• Mirror filter:

The `MIRROR` function allows the user to mirror an image in four different directions: left (l), right (r), upper (u), and lower (d). Here's a brief explanation of the algorithm for each direction:

1. It prompts the user to choose which side to mirror the image (left, right, upper, or lower) and reads the choice into the variable `mirror`.

2. Depending on the user's choice, the function mirrors the image in the specified direction. The process is performed as follows:

- If the user selects 'l' (left), the function mirrors the image to the left by reversing the pixel order in each row. It effectively flips the image horizontally.

- If the user selects 'r' (right), the function mirrors the image to the right by reversing the pixel order in each row, essentially flipping the image horizontally but in the opposite direction.

- If the user selects 'u' (upper), the function mirrors the image to the upper side by reversing the pixel order in each column. This operation flips the image vertically.

- If the user selects 'd' (down), the function mirrors the image to the lower side by reversing the pixel order in each column, effectively flipping the image vertically but in the opposite direction.

3. If the user enters an invalid choice, an error message is displayed.

The `MIRROR` function provides a way to mirror the image in different directions, effectively changing its orientation or creating a mirrored effect.

• Shuffle filter:

The `SHUFFLE` function shuffles the quarters of an image based on user input, and here's a brief explanation of the algorithm:

1. The function initializes a new 2D array called `image2` to store the shuffled image. This array will be used to preserve the original data.

2. It prompts the user to enter the new order of quarters, which should be specified as numbers 'a', 'b', 'c', and 'd', separated by spaces. The values 'a', 'b', 'c', and 'd' represent the order in which the quarters of the image will be arranged.

3. The function reads the user's input for 'a', 'b', 'c', and 'd'.

4. Depending on the user's inputs 'a', 'b', 'c', and 'd', the function proceeds to shuffle the quarters of the image based on the specified order. The process

involves copying portions of the original image to the corresponding positions in `image2`. The specific steps vary depending on the selected order of quarters.

5. Once the shuffling is complete, the shuffled image stored in `image2` is copied back to the original `image` array, effectively replacing the original image with the shuffled version.

The `SHUFFLE` function provides a way to rearrange the quarters of an image according to user-defined orders, allowing for various creative effects or data analysis tasks.

• Blur filter:

The `BLUR` function applies a blurring effect to an image by averaging neighboring pixel values. Here's a brief explanation of the algorithm:

1. The function initializes a variable `blurring_level` with the value 4, indicating the number of times the blurring process will be repeated. This means the blurring effect will be applied four times, which results in a stronger blur.

2. It enters a loop that repeats the blurring process for the specified number of `blurring_level` times.

3. Within each blurring iteration, it iterates through the inner region of the image, excluding the border pixels. For each pixel in the inner region:

- It calculates the average of neighboring pixel values by summing the values of the pixel itself and its 8 neighboring pixels (those within a 3x3 square centered around the pixel).

- It then updates the pixel value with the calculated average. The division by 9 is used for normalization, as there are 9 pixels involved in the average (including the pixel itself).

4. The loop repeats the blurring process for the specified number of times, which is controlled by the `blurring_level`.

As a result, the `BLUR` function applies a blurring effect to the image by smoothing out pixel values, making it appear less sharp. The more iterations it performs, the stronger the blurring effect. This kind of blurring is often used in image processing to reduce noise and create a smoother appearance.

• Crop filter:

The `CROP` function allows the user to crop a specified region of an image. Here's a brief explanation of the algorithm:

1. It initializes a new 2D array called `image2` to store a temporary copy of the original image. This copy is used to preserve the original data within the cropped region.

2. It iterates through the entire image, copying pixel values from the original `image` to the temporary `image2`.

3. It initializes the entire image to a white background by setting all pixel values to 255, which represents white.

4. It prompts the user to enter the coordinates (x, y) of the top-left corner and the dimensions (length and width, denoted as l and w) of the region to be cropped.

5. It reads the user's input for the coordinates and dimensions (x, y, l, w).

6. It copies the selected portion from the temporary image `image2` to the current image `image`. This copying process is done within a nested loop that iterates over the specified region using the coordinates and dimensions provided by the user.

As a result, the `CROP` function crops the image, preserving only the selected region specified by the user while filling the rest of the image with a white background. This operation is commonly used to focus on a specific area of interest within an image.

• Skew filter:

The `SKEW` function allows the user to skew the image either horizontally or vertically based on user input. Here's a brief explanation of the algorithm:

1. The function first checks if the user wants to skew the image horizontally ('e') or vertically ('f'). It prompts the user accordingly and reads the user's choice into the variable `op`.

2. If the user chooses to skew the image horizontally ('e'), the following steps are taken:

- The user is prompted to enter the degree of skew to the right.
- The degree in radians is calculated from the user's input.
- The angle in radians is used to calculate the variable `angle`, which represents the tangent of the angle.
- A variable `l` is calculated to determine the size of the skew operation.
- A temporary 2D array `image2` is initialized with a white background.
- The image is skewed to the right by repositioning pixels in the `image2` array based on the calculated skew angle.
- The skewed content is repositioned back to the original `image` array to complete the horizontal skew effect.

3. If the user chooses to skew the image vertically ('f'), similar steps are taken:

- The user is prompted to enter the degree of skew upwards.
- The degree in radians is calculated from the user's input.

- The angle in radians is used to calculate the variable `angle`, which represents the tangent of the angle.

- A variable `l` is calculated to determine the size of the skew operation.

- A temporary 2D array `image2` is initialized with a white background.

- The image is skewed upwards by repositioning pixels in the `image2` array based on the calculated skew angle.

- The skewed content is repositioned back to the original `image` array to complete the vertical skew effect.

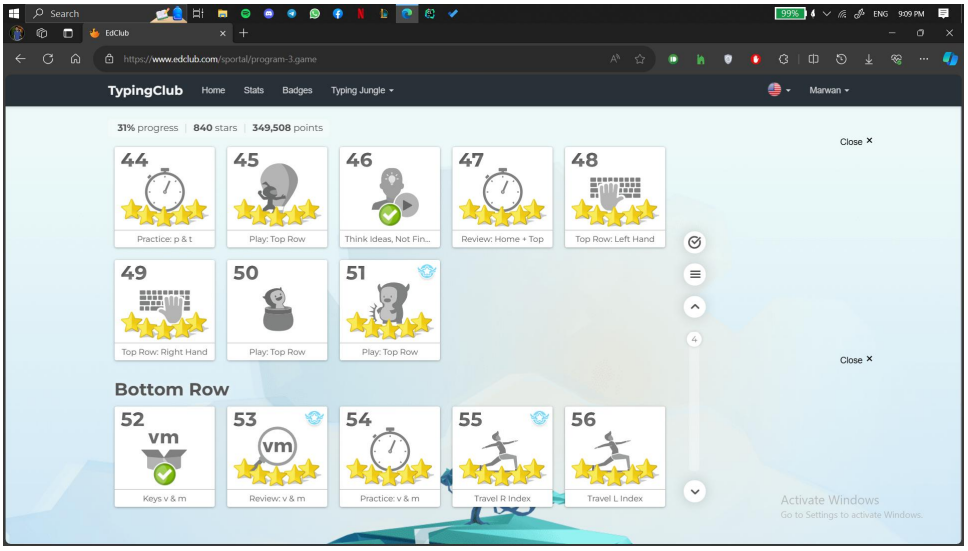
4. If the user provides an invalid input for skewing (neither 'e' nor 'f'), an error message is displayed.

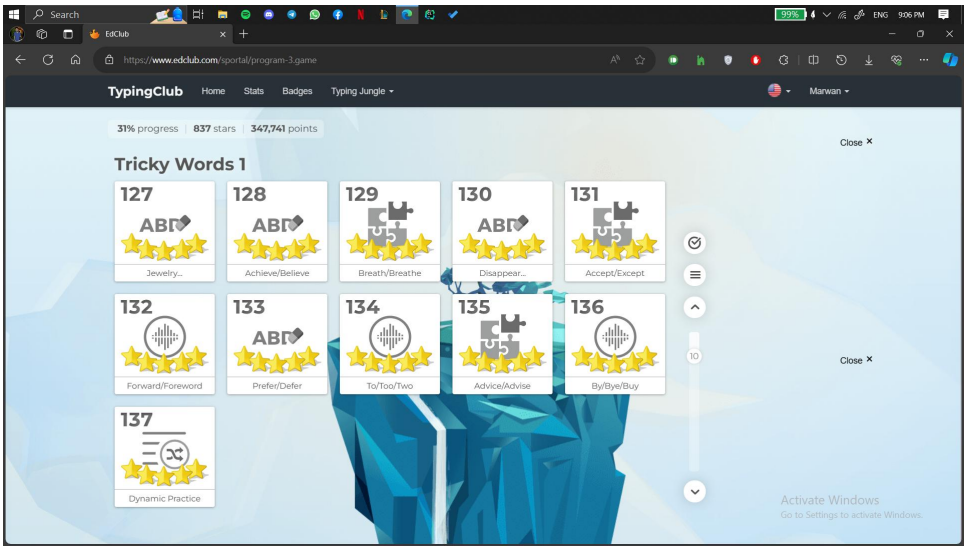
The `SKEW` function provides a way to apply skew transformations to the image, altering its appearance based on user-defined angles and directions (horizontal or vertical). Skewing is a common image transformation used in various image processing applications.

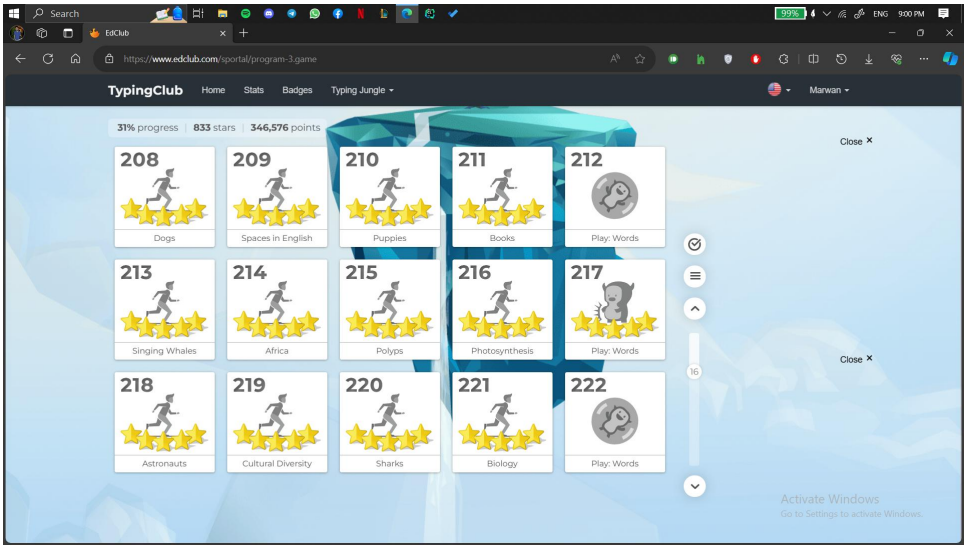
Third:

• typlingclub.com screenshots of the last level every one of us has achieved:

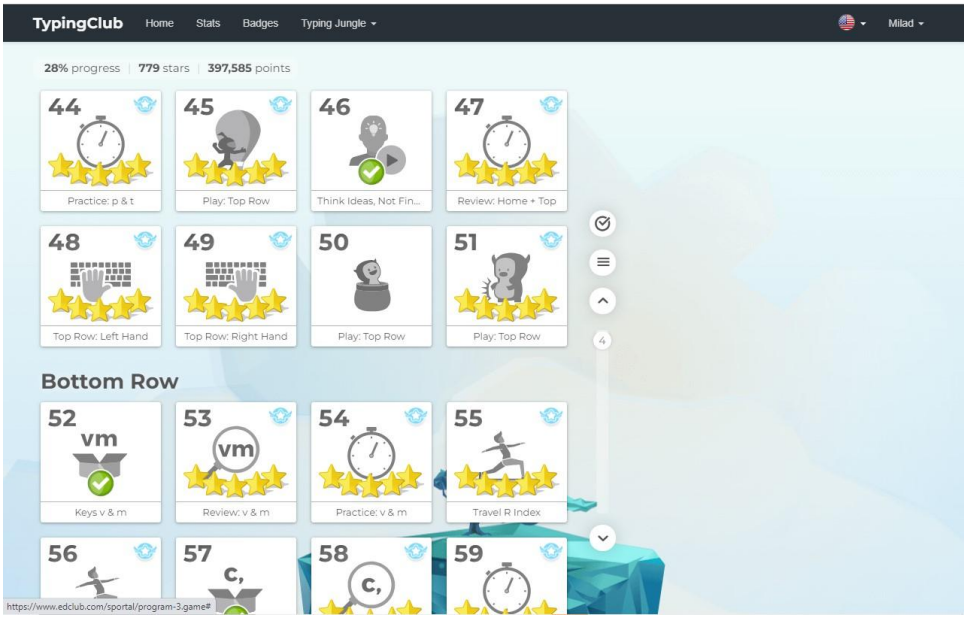
Marwan Osama Abd El-Azem Ibrahim ID: 20220324

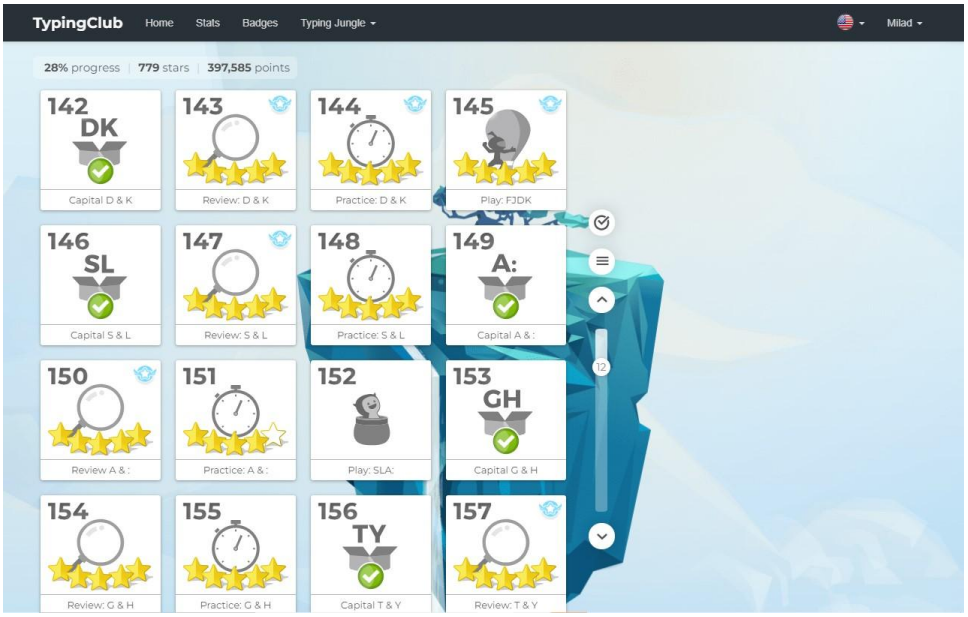






Milad Samy Milad Abdo ID: 20220354





Menna Khaled Gamal Mohaseb ID: 20221166