ID2090 Assignment-6 Report

Arivoli Ramamoorthy EE23B008

1 Q1: Treasure Hunt II

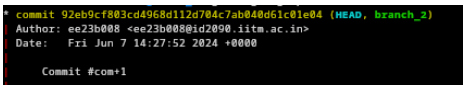
1.1 Introduction

- This time, you are tasked in searching for the Arceus - hill by hill, mountain by mountain, and when you spot it, don't hesitate to throw a Master Ball, and there, you conquer the whole world!!
- You are given a shell binary (script.sh.x). Run it in your home directory. The result will be a cpp file (question 2.cpp), which is committed and branched multiple times, similar to how Arceus has multiple hideouts and tunnels for escape.
- You can search for Arceus by traversing these branches and nodes. Once you found the Legendary Arceus, immediately merge the commit to the HEAD of the MASTER branch, just as you would catch Arceus using the Master Ball. The result ? The Pokemon world and hence the treasure hunt - is yours

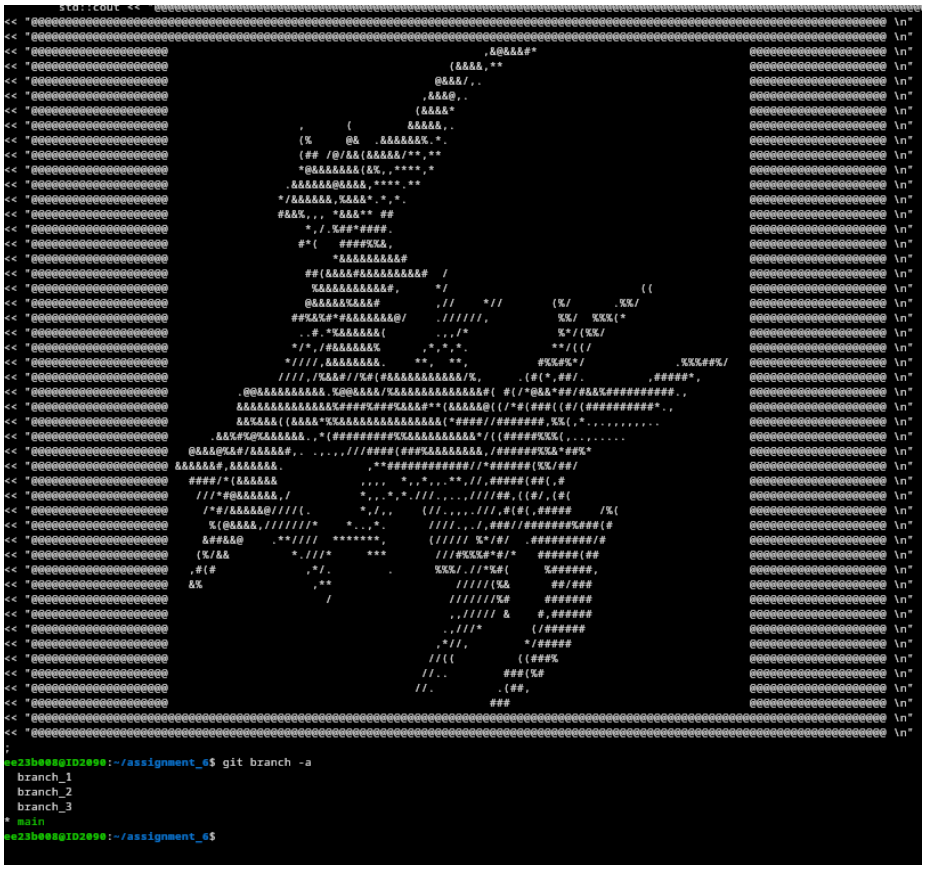
Figure 1: Ascii.txt Arceus

1.2 Solution

First run the given file script.sh.x .
Now, use the command **git branch -a** to list all the branches.
Go branch by branch using **git checkout branch_x** to checkout the three branches listed.
We will notice that the right Arceus is in branch_2 and we already know that master branch isn't created. So now we do **git checkout -b master** which creates and switches to the master branch.
Use git log to find the checksum. For us it is in branch_2 so its 92eb9cf803cd4968d112d704c7ab040d61c01e04



Final soln screen:



1.3 Commands used:

- git init - to initialize the git environment
- git checkout branch - to change branches
- git log question 4.cpp - to show the logs of all commits made in a branch
- git merge

2 Q2: Image Processing

2.1 Introduction

In this project, we were given two images of Lena—one clear and one noisy—and tasked with performing image processing on them using Python. The goal was to apply different image processing kernels and noise reduction techniques.

Figure 2: Original image

Figure 3: Noisy image

2.2 Implementation of Kernels

Two image processing methods were implemented: "Brightness Change" and "Contrast Change".

2.2.1 Brightness Change

Brightness change involves adding a constant value to all pixel intensities. This can be achieved by:

$$I_{\text{new}}(x, y) = I_{\text{old}}(x, y) + \text{value}$$

Listing 1: Brightness Change Implementation

```
# Brightness change
def change_brightness(image, value):
    bright_image = cv2.convertScaleAbs(image, alpha=1, beta=value)
    return bright_image

# Apply brightness change
brightness_value = 50
bright_image = change_brightness(original_image, brightness_value)
cv2.imwrite('bright_image.png', bright_image)
```

2.2.2 Contrast Change

Contrast change involves scaling pixel values around the mean intensity. This can be achieved by:

$$I_{\text{new}}(x, y) = \alpha \cdot (I_{\text{old}}(x, y) - \text{mean}) + \text{mean}$$

Listing 2: Contrast Change Implementation

```
# Contrast change
def change_contrast(image, alpha):
    mean = np.mean(image)
    contrast_image = cv2.convertScaleAbs(image, alpha=alpha, beta=(1 - alpha) *
    return contrast_image

# Apply contrast change
contrast_value = 1.5
contrast_image = change_contrast(original_image, contrast_value)
cv2.imwrite('contrast_image.png', contrast_image)
```

2.2.3 Results

Figure 4: Brightness Changed Image

Figure 5: Contrast Changed Image

2.3 Noise Reduction

To reduce the noise in the noisy image, a Gaussian blur kernel was applied. The Gaussian kernel is defined as:

$$k_{\text{Gaussian}} = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Listing 3: Gaussian Blur for Noise Reduction

```
# Gaussian blur kernel for noise reduction
```

```
gaussian_kernel = np.array([[1, 4, 6, 4, 1],
                            [4, 16, 24, 16, 4],
                            [6, 24, 36, 24, 6],
                            [4, 16, 24, 16, 4],
                            [1, 4, 6, 4, 1]], np.float32) / 256

# Apply Gaussian blur to noisy image
gaussian_blurred_image = cv2.filter2D(noisy_image, -1, gaussian_kernel)
cv2.imwrite('gaussian_blurred_image.png', gaussian_blurred_image)
```

2.3.1 Results

Figure 6: Gaussian Blurred Image

2.4 Frequency Filtering

The Fourier Transform of the image maps the 2D intensity array into frequencies. A low-pass filter was applied to remove high-frequency noise. The low-pass filter is defined by:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

The filtered image in the frequency domain was obtained and then transformed back to the spatial domain.

Listing 4: Frequency Filtering Implementation

```
# Fourier transform for frequency filtering
dft = np.fft.fft2(noisy_image)
dft_shifted = np.fft.fftshift(dft)

# Create a low-pass filter
```

```
M, N = noisy_image.shape
X, Y = np.meshgrid(np.arange(N), np.arange(M))
centerX, centerY = N // 2, M // 2
radius = 60
low_pass_filter = np.sqrt((X - centerX)**2 + (Y - centerY)**2) <= radius

# Apply the low-pass filter
dft_filtered = dft_shifted * low_pass_filter
dft_inverse_shifted = np.fft.ifftshift(dft_filtered)
filtered_image = np.fft.ifft2(dft_inverse_shifted)
filtered_image = np.abs(filtered_image)
cv2.imwrite('frequency_filtered_image.png', filtered_image.astype(np.uint8))
```

2.4.1 Results

Figure 7: Frequency Filtered Image

2.5 Conclusion

In this project, we applied image processing kernels and frequency filtering to perform noise reduction on images. The Gaussian blur kernel effectively reduced the noise in the spatial domain, while the low-pass filter successfully eliminated high-frequency noise in the frequency domain. Each method has its advantages depending on the type and nature of the noise present.