Assignment 1

Wang Jinghan 23099359d

The code for the assignment is available on github: <https://github.com/sebastianvne/COMP4432-Assignment1>

1. Introduction

The assignment is meant for generating Lego style image from original image. The assignment simply mapped the original rgb vector to few discrete color vectors, determined by configurations by calculating the euclidean distance.

1. Design

module 1:

capturing the image using computer camera, which is embedded in opencv library.

module 2:

Task2 requires us to map the original image to a image with maximum 100 \* 100 bricks, each brick is of size 1\*1.

Hence, I designed the data flow as following: firstly determine the target size of downsampling, the function would calculate the long side and its size, then resize the long side to maximum of 100 bricks, while keeping the ratio of image long side size and short side size.

Secondly, calculating the euclidean distance of each original image pixels with the specific target mapping colors, then finding the color with the minimum distance and generate the Lego pixels.

Finally, resize the generated pixels to original size, and output the image.

module 3:

Task3 requires to use more colors and more size of bricks to generate the Lego style image. Based on task2, I implemented a greedy brick merge. The code is as follows, for each original 1\*1 bricks, the algorithm would iteratively use different size of bricks to check if using the specific brick would cover other bricks that originally not the same color as the initial 1\*1 brick.

Also, the original bricks is different from the implementation of task2, task3 would map the original image with more color vectors.

module 4:

For task4, I implemented a live mode that can capture the image from computer camera and transform it into Lego style in live mode.

1. Test

The test would only indicates the tests that conducted by executing python3 main.py in the repository, with suffix in test command part.

1. Task 1:
   1. Test if window of opencv can show the correct image captured from computer camera. Window showed as below. Run python3 main.py --mode capture



* 1. Test if Press q can be used to exit and press s can be used to save the image, and check if the image saved is correct. As shown in my github repository /outputs/test1-2.jpg, the image is correct.



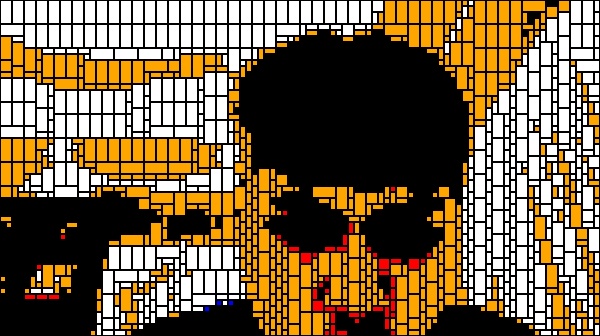
1. Task 2:

I used the image test1-2.jpg to test the output of task2. Running python3 main.py --mode task2 --input ./outputs/test1-2.jpg. The output is shown in the repository, within the outputs folder, test2-1.jpg shows the output of task2, the image uses black, white and gray to render the image.



1. Task 3:

Run command python3 main.py --mode task3 --input ./outputs/test1-2.jpg, the image will show in the output folder, brick\_summary\_3-1.txt shows the bricks summary, and test3-1.jpg is the output of task3



brick\_summary\_3-1.txt:

Total bricks: 1520

2x4: 426

2x2: 95

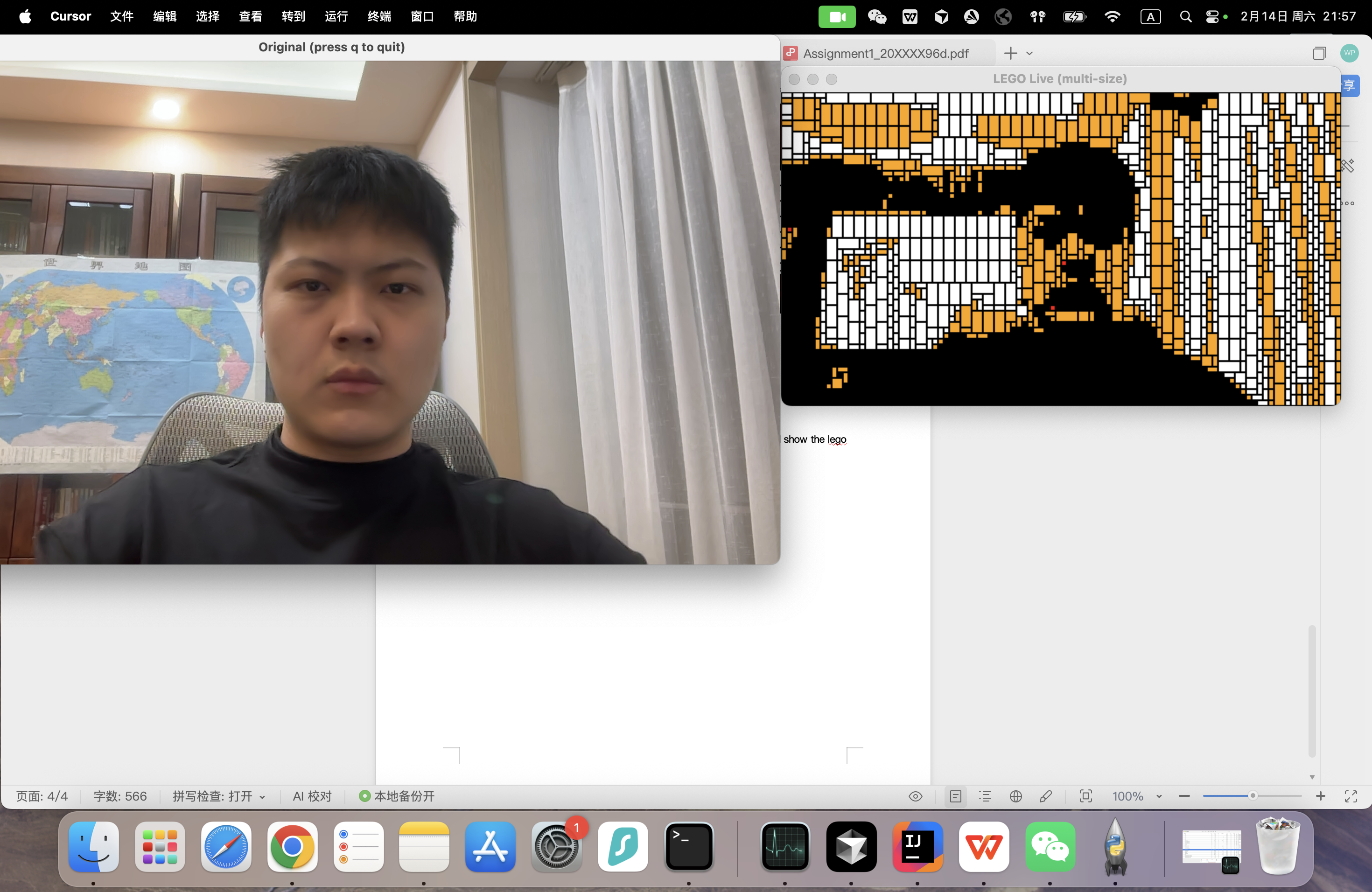
1x2: 813

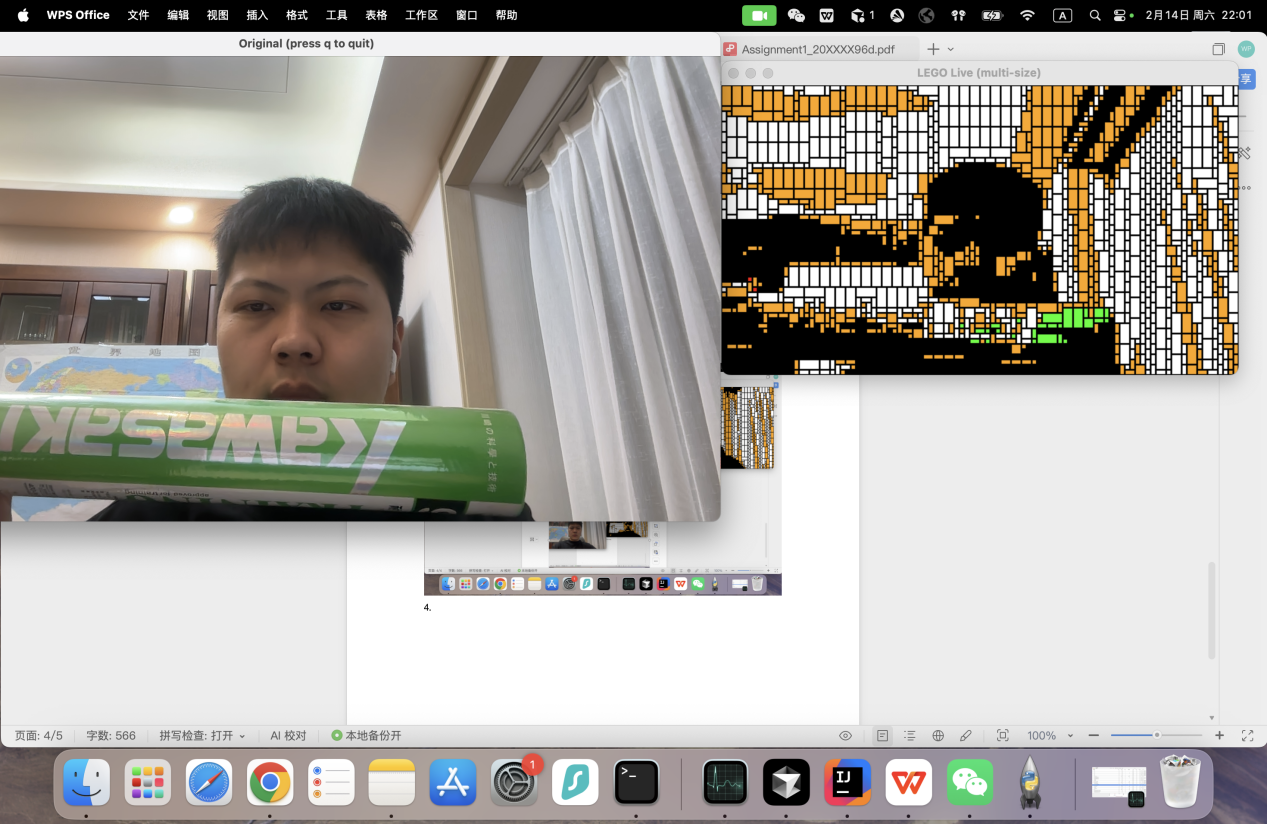
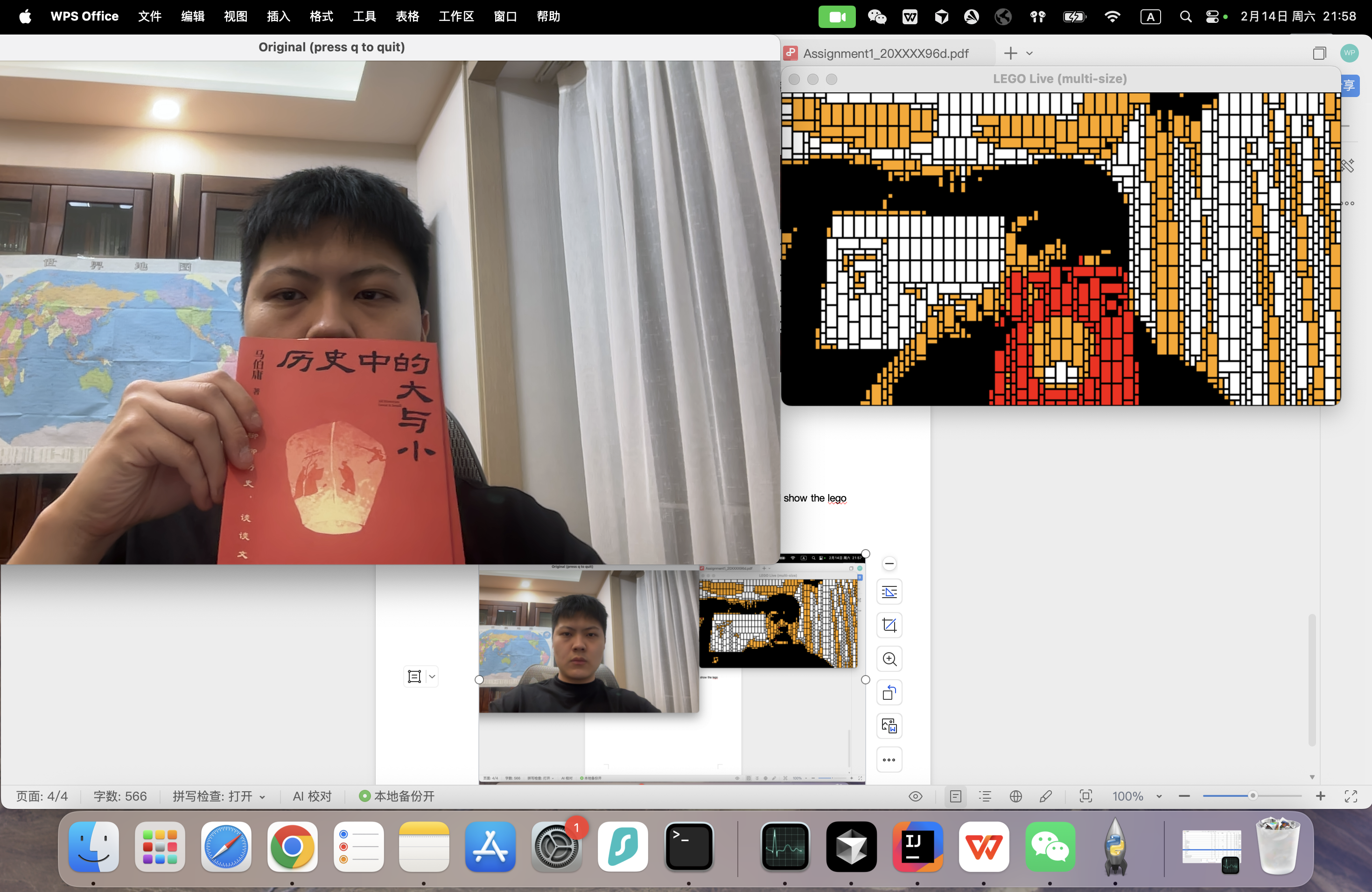
1x1: 186

1. Task 4:

Run python3 main.py --mode live, a window would pop up and show the lego style image and the original image from computer camera.

Some images are shown below.





1. Robustness

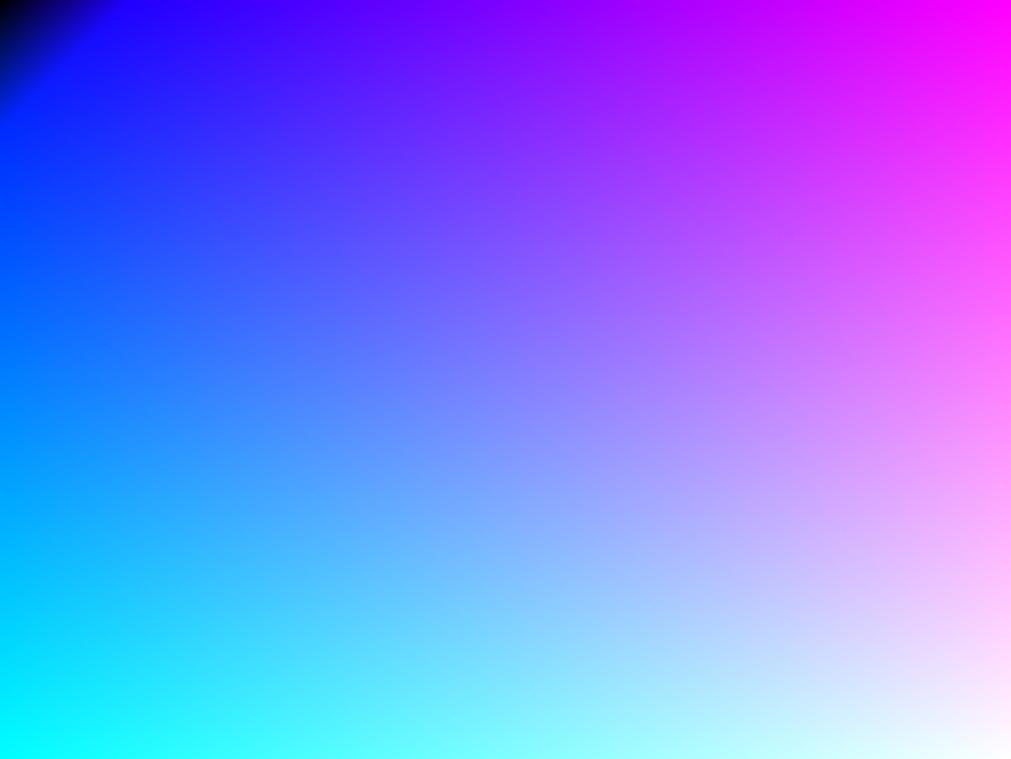
In real scenario, there could be too light, too dark, or the image could be too large or too small.

Considering that the capability of computer camera can be limited and cannot get detailed pixels in dark scenario and too light scenario, I consider to keep the over light image rendered as original, the dark scenario as well.

For too large image and small image, I generated several gradient image to test the large and small image, and check if for each image, program can output correct image (without exceeding 100\*100 limit). The generation script is ‘generate\_test\_image.py’ 。

I picked one small image and one large image to test task2 and task3 to show the robustness of my programme. The large image is of 2000\*1500 pixels, and the small image is of 10\*10 pixels. The results are shown below:

Large original:



Large task2 result:



Large task3 result:

brick summary:

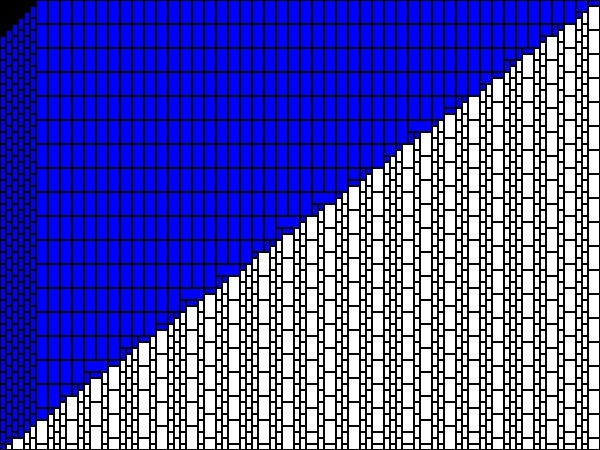
Total bricks: 1856

2x4: 632

2x2: 25

1x2: 1145

1x1: 54



Small original(scaled larger):

test_small_10x10

Small task2 result:

lego_task2_small

Small task3 result:

brick summary:

Total bricks: 28

2x4: 8

1x2: 16

1x1: 4

lego_small

1. Problems and Solutions
2. Originally, when I follow AI’s idea to map the image into 3 colors, AI recommend to transform it into HSV range to attribute the color of each bricks instead of using RGB to calculate the euclidean distance, however, the range may not cover all the colors, hence may cause several pixels has no color after handled. I manually revised the code into using euclidean distance, and using the lowest distance to ensure that each pixel can have a color to attribute.
3. Originally I added outlines on bricks and then resize the whole image, which makes the outlines too large and weaken the final generated image. Also, I ignored the boundary then the boundary of each brick may be too long that exceed the expected position. I revised that by adding outlines after resize the whole image, and add boundary check in the code.
4. GenAI for Implementation

I simply prompt engineering a prompt indicating that AI should be focusing on implementing cv algorithms that can implement the task requirements.

I accept some of the implementation and go through the code for review and revise the code when AI did not correctly understand the tasks.

1. Understanding from GenAIs

GenAI treated the task as an image-to-LEGO pipeline: (1) resize so the longest side ≤ 100 to cap brick count; (2) map each pixel to the nearest palette color using squared Euclidean distance; (3) greedily merge same-color regions into larger bricks (2×4, 2×2, 1×2, 1×1); (4) draw brick outlines on the upscaled image. Task 2 uses 3 colors and 1×1 only; Task 3 uses 7 colors and multiple brick sizes. Live mode applies this pipeline to each camera frame and shows both the original and LEGO views.

1. Limitations

AI lack of implementing a extensible codes, AI generated a simple python script with everything within one file. Functions are chaotic and hard to understanding for a programmer new to this assignment.

AI implemented a greedy strategy to implement merge of 1\*1 bricks into larger size bricks, which maybe is not optimal and better and faster implementation can be conducted.