Phase I: Image Search and Detection

Group 3:

Cody Ambrose, Rodney French, John Kaiser, Orry Newell

University of Maryland Global Campus

Revision: 1.0

Date: 11/22/2020

Class #: CMSC 495

**Revision Log**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Date** | **Editor** | **Description** |
| 1 | 11/22/2020 | Cody Ambrose | Original Document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# User’s Guide

## Phase1\_Shapes.py

1. Download the Group3\_Phase1.zip folder.

2. Unzip/Extract the contents of Group3\_Phase1.zip folder.

3. From Spyder IDE 4.1.4 using Python 3.8.3, open Phase1\_Shapes.py.

4. Click the green “Run file” arrow in the tool bar.

5. A GUI should appear (you may need to click over to it in your task bar).

6. Select which shape you want to search for and click “Find”.

7. The GUI will display the shape you chose to search for, the larger “puzzle” image of shapes, and the shape you chose highlighted in green within the “puzzle” image.

8. You may choose other shapes and click “Find” to find them as well or close the application.

## Phase1\_ShapeColors.py

1. Download the Group3\_Phase1.zip folder.

2. Unzip/Extract the contents of Group3\_Phase1.zip folder.

3. From Spyder IDE 4.1.4 using Python 3.8.3, open Phase1\_ShapeColors.py.

4. Click the green “Run file” arrow in the tool bar.

5. A GUI should appear (you may need to click over to it in your task bar).

6. Select which color shape you want to search for and click “Find”.

7. The GUI will display the shape color you chose to search for, the larger “puzzle” image of shapes with different colors, and the shape color you chose highlighted in green within the “puzzle” image.

8. You may choose other shapes and click “Find” to find them as well or close the application.

\*Note: The target images and “puzzle” images are included in the Group3\_Phase1.zip folder at the appropriate file paths for Phase1\_Shapes.py and Phase1\_ShapeColors.py to find them. You must run Phase1\_Shapes.py and Phase1\_ShapeColors.py from their extracted location within the Group3\_Phase1 folder to ensure images are properly pulled and referenced.

# Original Phase I Milestone

The original Phase I Milestone was to create a basic GUI with pre-determined images. The first program was to use OpenCV to find a user-selected shape within a larger “puzzle” image of shapes. The second program was to use OpenCV to find a user-selected shape color within a larger “puzzle” image of shapes with different colors. The goal of Phase I testing was to correctly find the user-selected images in both programs while building a foundation of how template matching using OpenCV works within a Python programming environment.

# Time Management/Schedule

Based on the results of Phase I testing, our group is on schedule according to our original milestone. We have successfully developed two Python programs, Phase1\_Shapes.py and Phase1\_ShapeColors.py, during Phase I that accomplished the goals we set out to accomplish. Phase1\_Shapes.py successfully finds the user-selected shape within the larger “puzzle” image of shapes 100% of the time. In addition, Phase1\_ShapeColors.py successfully finds the user-selected shape color within the larger “puzzle” image of shapes with different colors 100% of the time. As a reference, our original Test Matrix for Phase I is provided below:

|  |  |  |
| --- | --- | --- |
| Target Image | Expected Output | Reasoning |
| Square | Program places a box around the square | The program finds the square based on template matching. |
| Star | Program places a box around the star | The program finds the star based on template matching. |
| Triangle | Program places a box around the triangle | The program finds the triangle based on template matching. |
| Red | Program places a box around red object | The program finds the color matching the RGB value red. |
| Green | Program places a box around green object | The program finds the color matching the RGB value green. |
| Blue | Program places a box around blue object | The program finds the color matching the RGB value blue. |

# Problems Encountered

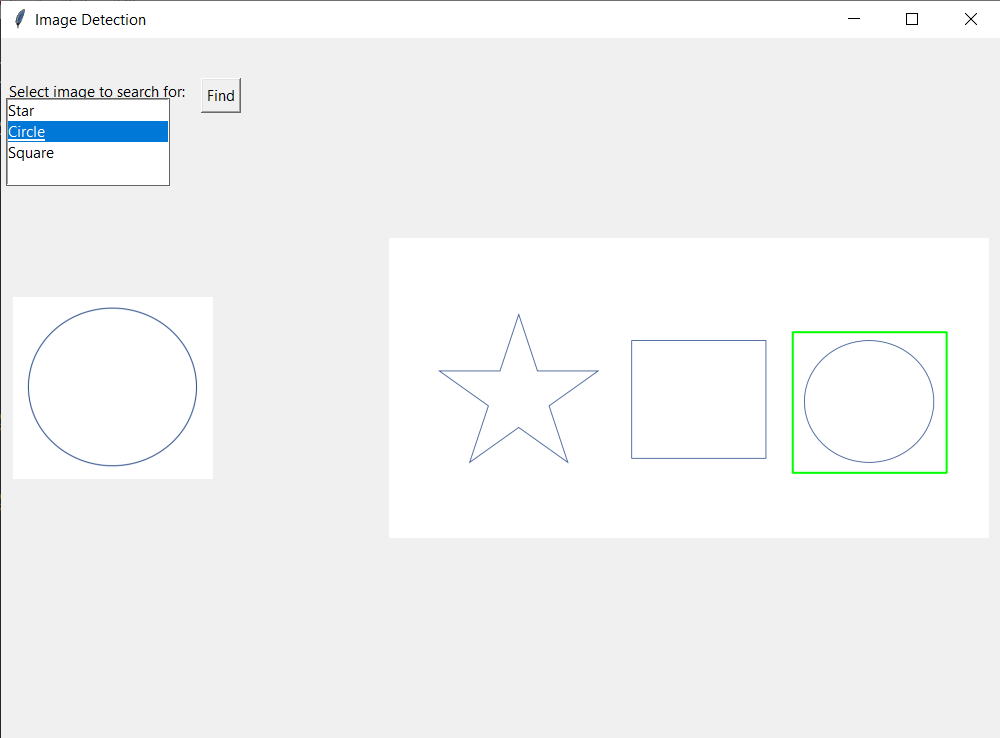
To find shapes of different colors using OpenCV template matching in Phase1\_ShapeColors.py, we could not use the same template matching method that was used in Phase1\_Shapes.py. To simply find different shapes, the TM\_CCOEFF template matching method worked. However, to find shapes of different colors, we had to use the TM\_SQDIFF\_NORMED template matching method which uses minLoc instead of maxLoc for the program to work. OpenCV offers six different template matching methods (TM\_CCOEFF, TM\_CCOEFF\_NORMED, TM\_CCORR, TM\_CCORR\_NORMED, TM\_SQDIFF, and TM\_SQDIFF\_NORMED) to detect images within images. Experimenting with these template matching methods was a large part of the problem-solving process during Phase I.

# Reevaluation

Reflecting on the results of Phase I, we believe that our choice of Python and OpenCV is still the most appropriate solution for our larger “Where’s Waldo” project we are building towards. However, we may have to explore more complex methods outside of simple template matching to find a character like Waldo within a “Where’s Waldo” puzzle. Additional methods to research for Phase II include edge detection and adaptive thresholding, as well as feature matching. We also believe that our choice of tkinter for the GUI interface, pathlib library for file paths (to standardize our data file structures across operating systems), input formats (.jpg, .jpeg, .png), output formats (.jpeg), test cases, and performance estimate benchmarks are all still appropriate to carry on to Phase II.

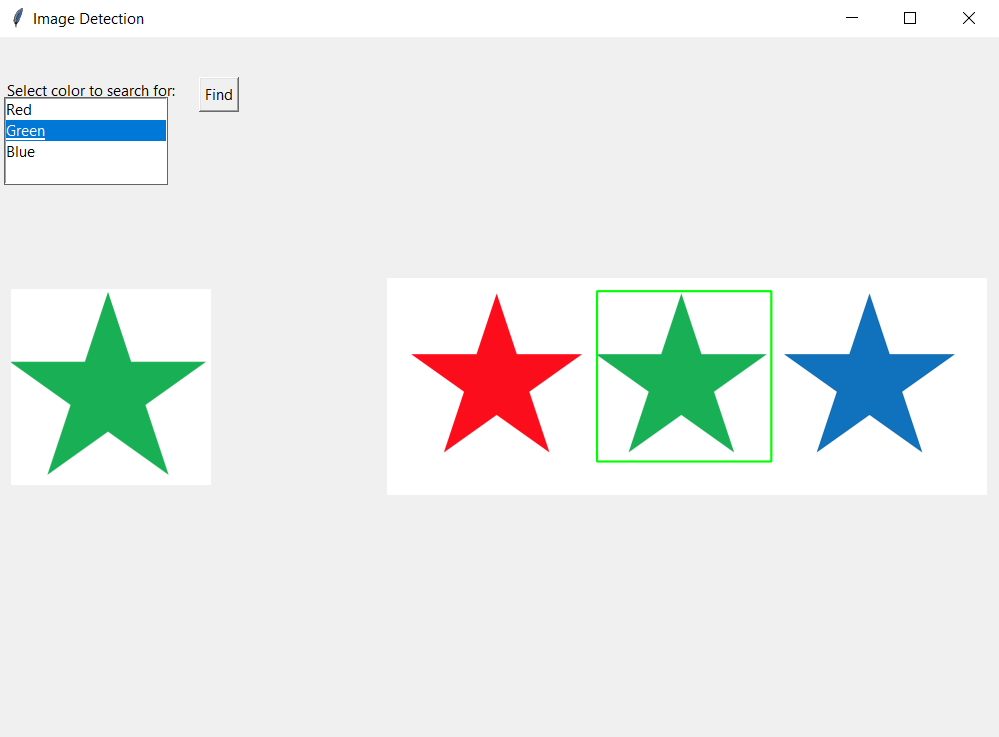
# Results

## Phase1\_Shapes.py



The program successfully finds the user-selected shape within the larger “puzzle” image of different shapes 100% of the time within our 10 second performance estimate threshold.

## Phase1\_ShapeColors.py



The program successfully finds the user-selected shape color within the larger “puzzle” image of different color shapes 100% of the time within our 10 second performance estimate threshold.