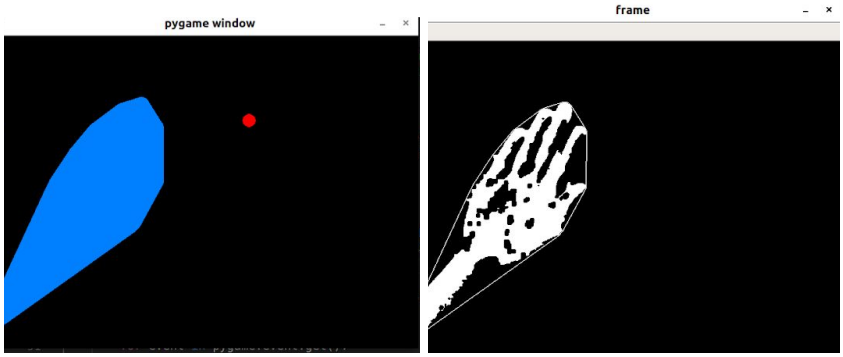
Mini Project 4

Project Overview:

The goal of this project was to use computer vision to mask out a user's hands, then to later detect when a polygon approximation collided with a point on screen. To accomplish computer vision, OpenCV was used to remove the background, the users face, and to mask everything out except skin tones, which ultimately left only the hands. Then, in pygame, a collision between hands and an arbitrary point was calculated.

Results:

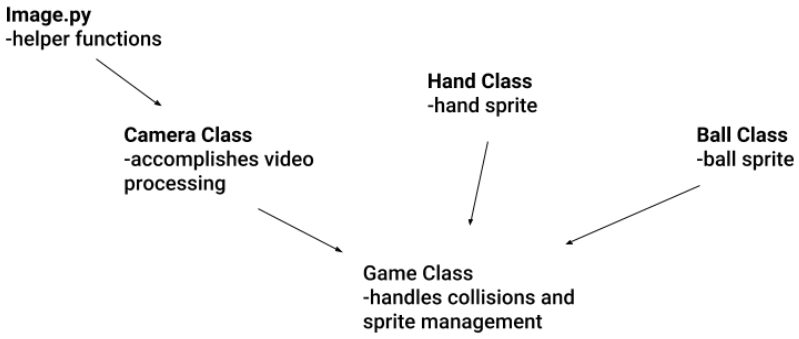


The right image shows the hand found with OpenCV while the left image shows the approximation in Pygame.

The project primarily accomplished reliable hand recognition, assuming your hands were not backlit, and also, pixel perfect collisions in pygame to see when your hand collided with the red dot on the screen. In terms of hand recognition, it proved to be more difficult than expected especially in terms of making it work in most lighting conditions. The image on the right above shows the final product after image processing. It is not perfect, and likely could be improved, but it is enough to give a good approximation. It also worked very rapidly, without lag.

The pygame half of the project worked more efficiently than expected. There was no delay at 60 frames a second even when using pixel perfect collision. In the image on the left above, when the blue hand approximation collides with the red dot, the red dot is placed in a random new position. The point of the game is to swat the red dots.

Implementation:



As seen in the UML diagram above, there are four main classes, and one python file dedicated to helper functions for the camera class. The camera class accomplishes all the video processing and outputs points that approximate the outline of any hands in the image. To do this, the face was removed, the background was removed, then a mask was applied to pull all parts of the image that fell in a YCrCb range for skin tones.

The hand approximations were then used in the game class to create Hand objects. The ball object is also created in the in the game class. With the objects defined, they were drawn on the screen then tested to see if a collision took place. If a collision occurs, the ball is placed in a new, random location on the screen.

An important design decision was to separate Image.py from the camera class to improve readability and useability. Initially, all the helper functions were inside the camera class, which made them a bit confusing to reference, so instead, they were placed in a separate file then imported. This made it clear when image processing was occurring, while the camera class could focus on just getting the hands.

Reflection:

Overall, I learned a lot from this project, both in terms of OpenCV and pygame. The part where I learned OpenCV was very rewarding, as time was spent understanding what the functions within OpenCV were doing, rather than just using them with little understanding. On the other hand though, I had very little time to learn pygame, so much of it was pieced together. This made my understanding of pygame unclear and scattered. There is a lot of room for improvement on the pygame side. I think that there are many places where the processes I used an unnecessary amount of memory and took longer than they could have.

Though the programming side of the project went well, there is a lot of room for improvement on the teaming side. The project became entirely one sided as a result of one sided communication. Messages were sent, but responses were either late or nonexistent. I ended up programming the entire project. Next time, I would establish better communication methods from the start, so that meeting times could be schedule and so the project would not be one sided.