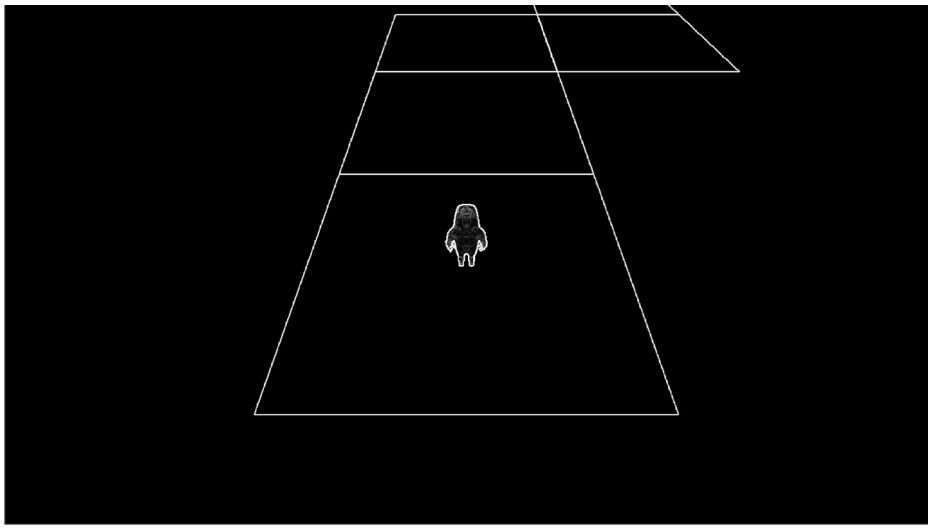
REPORT

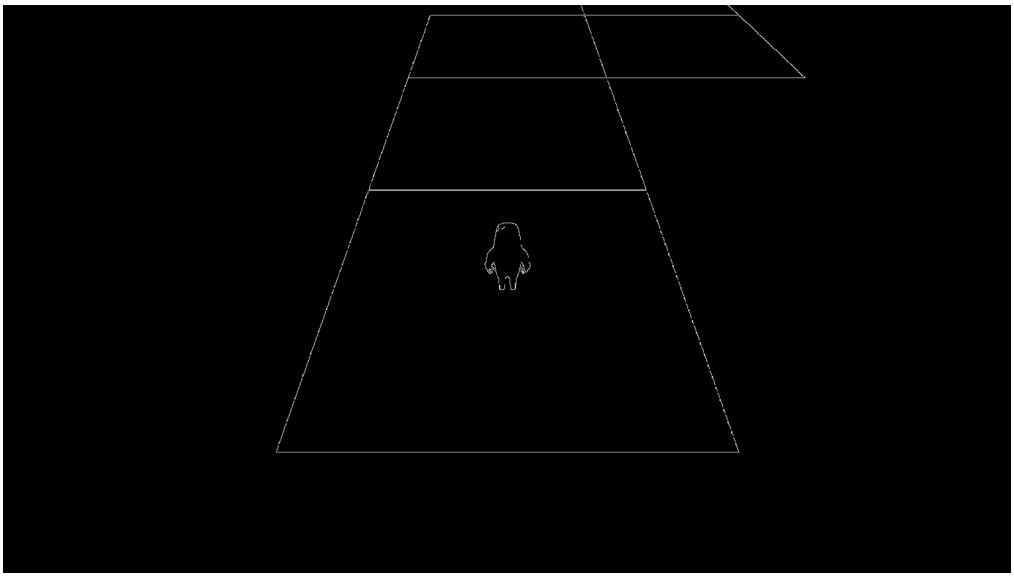
Part 1

In this part, I detected edges using the Sobel filter. Since we were not allowed to use library function, I implemented one which is definitely slower than library one. Instead of cropping the screenshot and detecting edges, I made the game full-screen so whole screenshot was only the game.



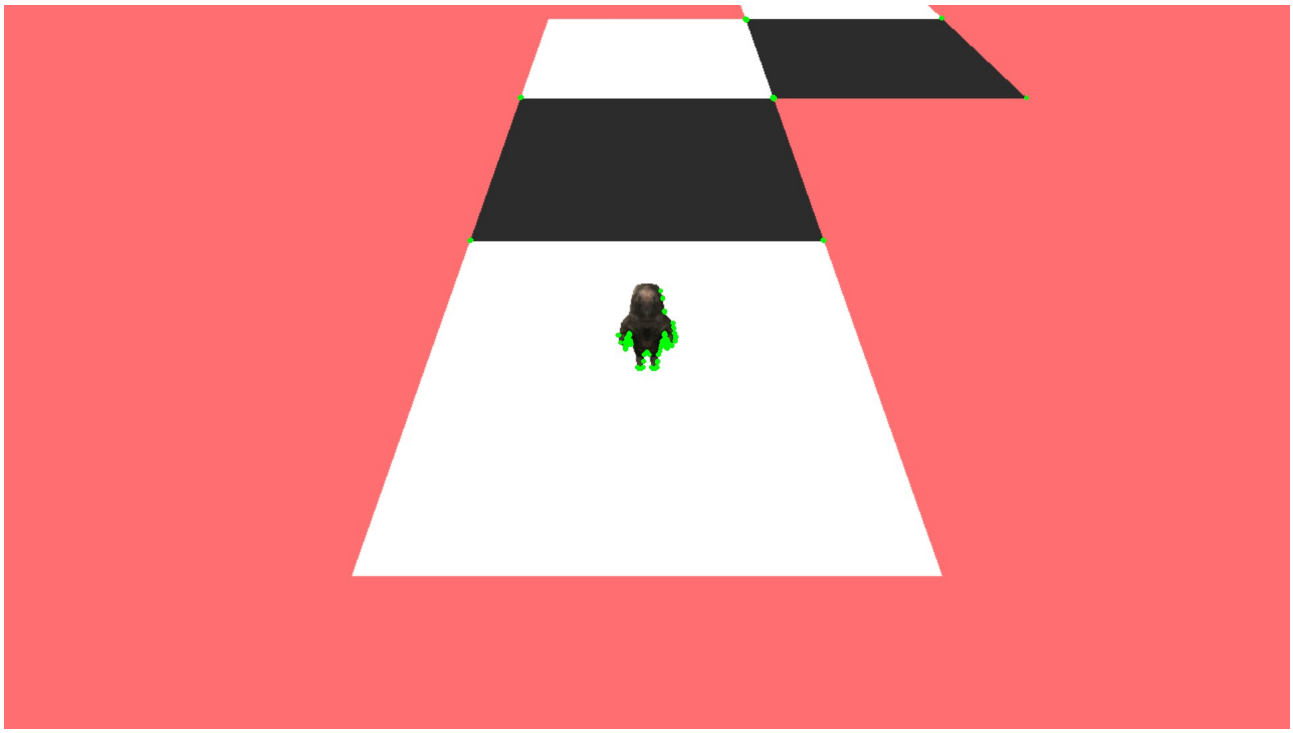
Part 2

By utilizing OpenCV's default Canny edge detector function (cv2.Canny), I detected edges here.



Part 3

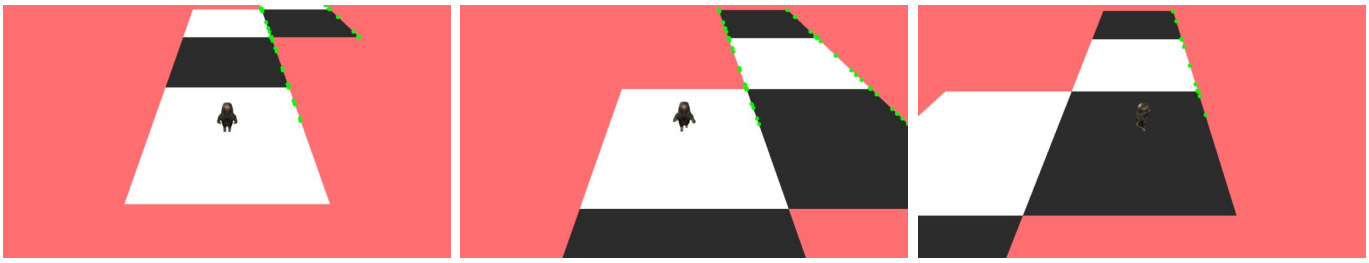
At this part, I made a Shi-Thomasi corner detection function. Instead of taking a screenshot and arranging it again, I read a saved screenshot as image and made all operations on it this time. Also detected the most appropriate quality level by trying and comparing the result with expected one on homework PDF.



Part 4

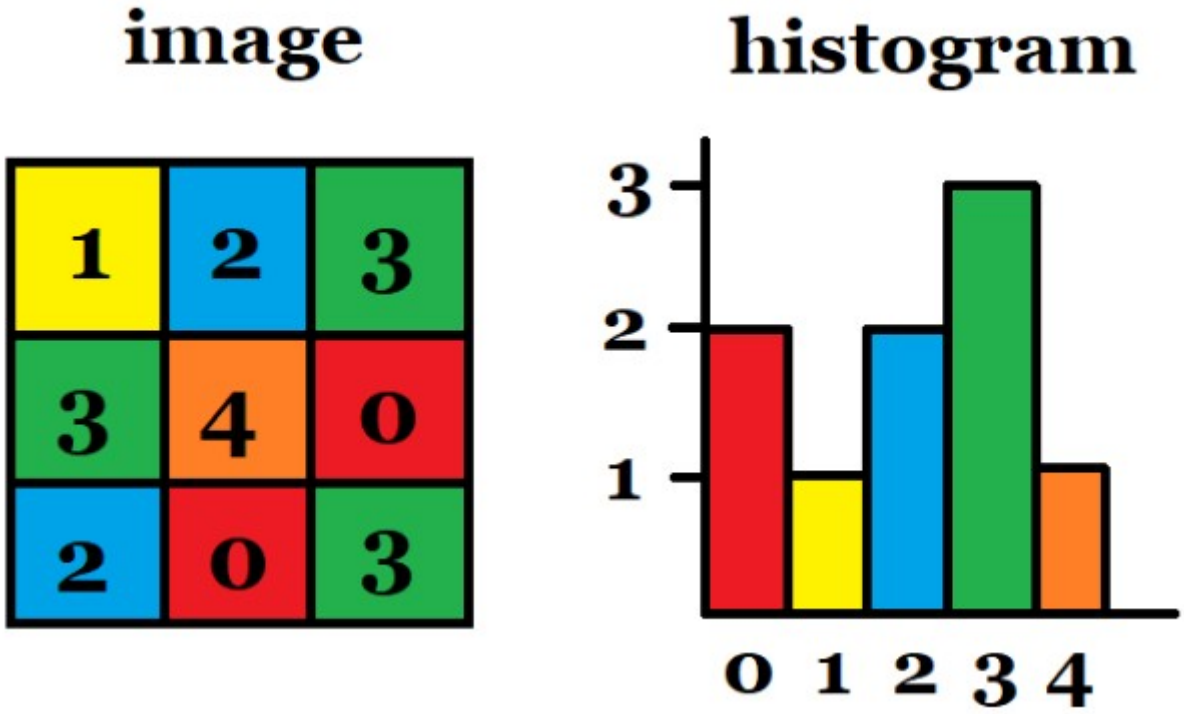
In this part, first I detected edges with Canny edge detector. Then I found coordinates of corners by utilizing the Shi-Thomasi corner detector that I implemented on part3. Since the creature is always at the center of screen, I could discard corners found on creatures easily. I also discarded corners on left side and bottom, since the path of map goes to the right and forward. Then I compared the average of corners with center's x and y, decided which way the creature goes. If the difference between average of corner's x and center's x is higher than the difference of average of corner's y and center's y, creature goes right for 2 seconds. Else, moves forward for 4 seconds.

After running the file, please switch to game in full-screen mode and click screen once. Make sure to hide mouse cursor since it's edges might effect the result.



Part 5

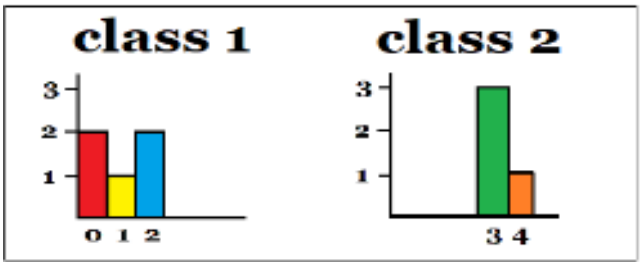
Consider an image with 9 pixels. Assume pixels' intensity range is between 0 and 4 (For the sake of easiness).



The variance formula is $\sigma^2 = \frac{\sum_{i=0}^{N} (x_i - \mu)^2}{N}$. We subtract the mean intensity from each pixel's intensity, take square of it to eliminate negative results, add all and divide to number of pixels. Higher variance means more dispersed image.

The variance of this image is $\frac{16}{9}$.

The purpose of Otsu thresholding is to select optimal threshold value to divide image (let's say, foreground and background of image). The variance within classes should be minimal and the variance between classes should be maximal. Which means each classes' pixels' intensity values should be closer to its mean. Let's choose 2 to divide this image to two classes.



To find the variance within class, we find the variance of each class in itself, multiply it with the number of pixels in that class divided with total number of pixels of whole image, (For example, class 1 has 5 pixels so 5/9) then add the result of all classes.

$V_w = \frac{5}{9} x \frac{4}{5} + \frac{4}{9} x \frac{3}{16} = \frac{4}{9} x \frac{19}{16}$

When we think naturally, the total variance must be equal to the sum of variances of each class (in other words, variance within class) and variances between classes. If there is only 2 classes, we can find the between class variance with this formula:

W_i = number of pixels in class i / total pixels

$$W_1 W_2 (\mu_1 - \mu_2)^2$$

So the between-class variance for this image is:

$$\frac{5}{9} x \frac{4}{9} x \left(1 - \frac{13}{4}\right)^2 = \frac{5}{4}$$

As expected, when we subtract within class variance from total variance of image:

$$4 x \frac{4}{9} - \frac{19}{16} x \frac{4}{9} = \frac{45}{36} = \frac{5}{4}$$