# BLG 453E
# Computer Vision
# Term Project Report

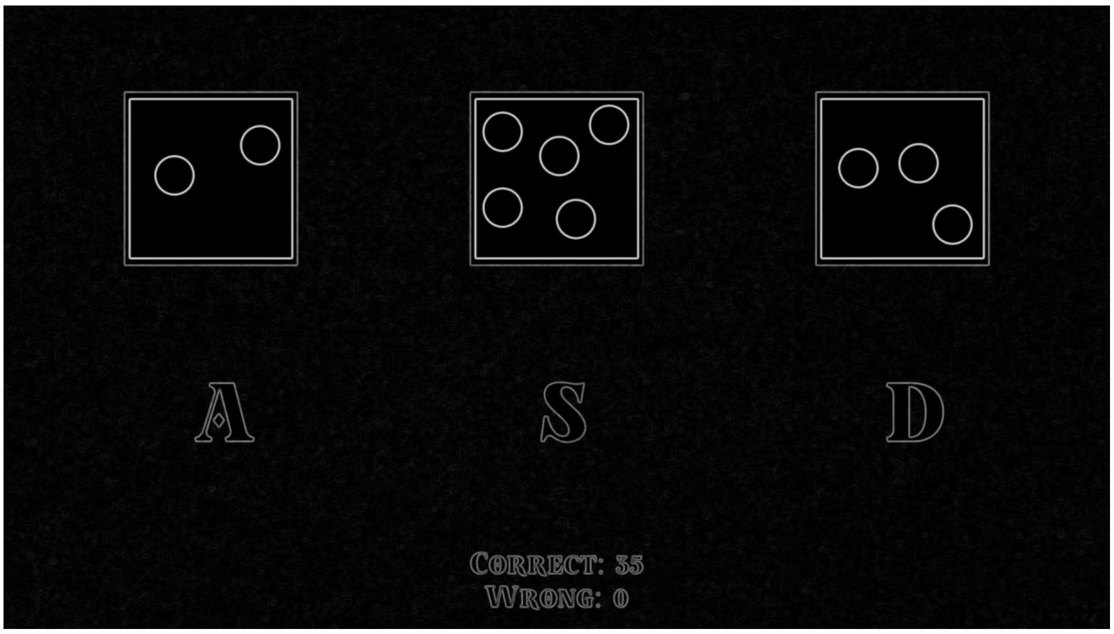
Ece Sancar
150170049

Bora Yöntem
150130806

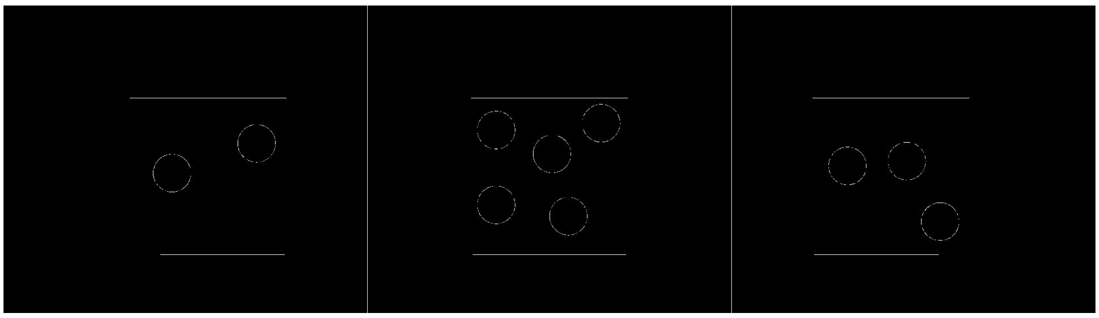
**Part 1: Dice Game**

game1.py and game2.py are implemented for the games.

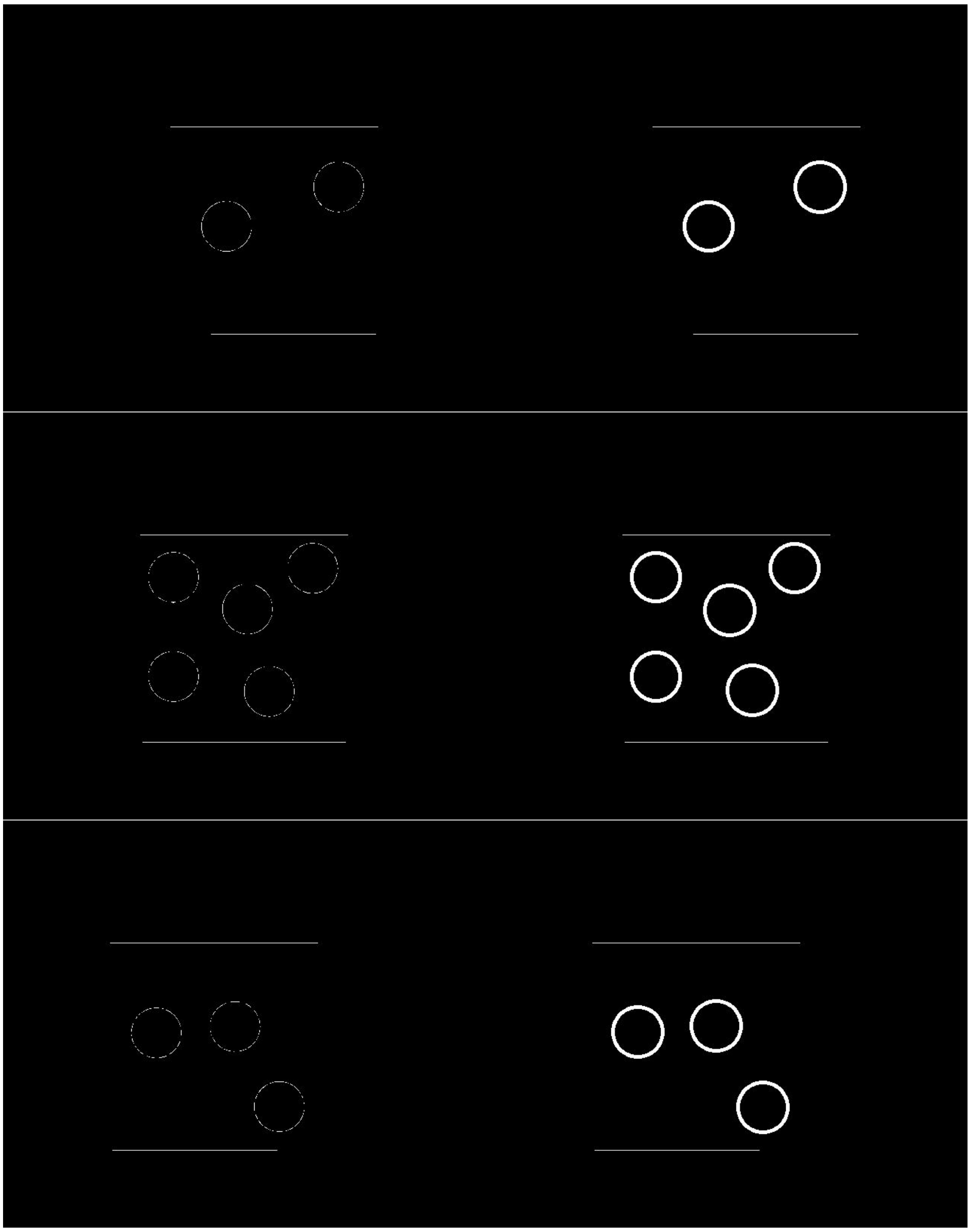
**Game 1:** Our program takes screenshots from the game and calculates the number of circles on each die. After converting to grayscale, sobel filtering is used for detecting the edges on the screenshot image. For processing each die separately, we used seperating_dice() function. It crops the lower part of the image and splits the upper part to three. We also applied a threshold value to get rid of the unwanted points in the background. We used a threshold value which we got the best results for circle detection. Three dice images are returned separately. In circle() function, cv2.HoughCircles() function is used to detect the circles. We tried different parameters for the function and used the ones that gave the best results. Number of the detected circles is returned. At the end, our program checks if number of detected circles for each die is between the numbers 0 and 7. It is done to be sure that the screenshot was taken after the game is started. If the numbers are between 0 and 7, corresponding key for the higher number is pressed. Results of our functions for an example game screenshot is shown below.



Result of Sobel Filter Edge Detection

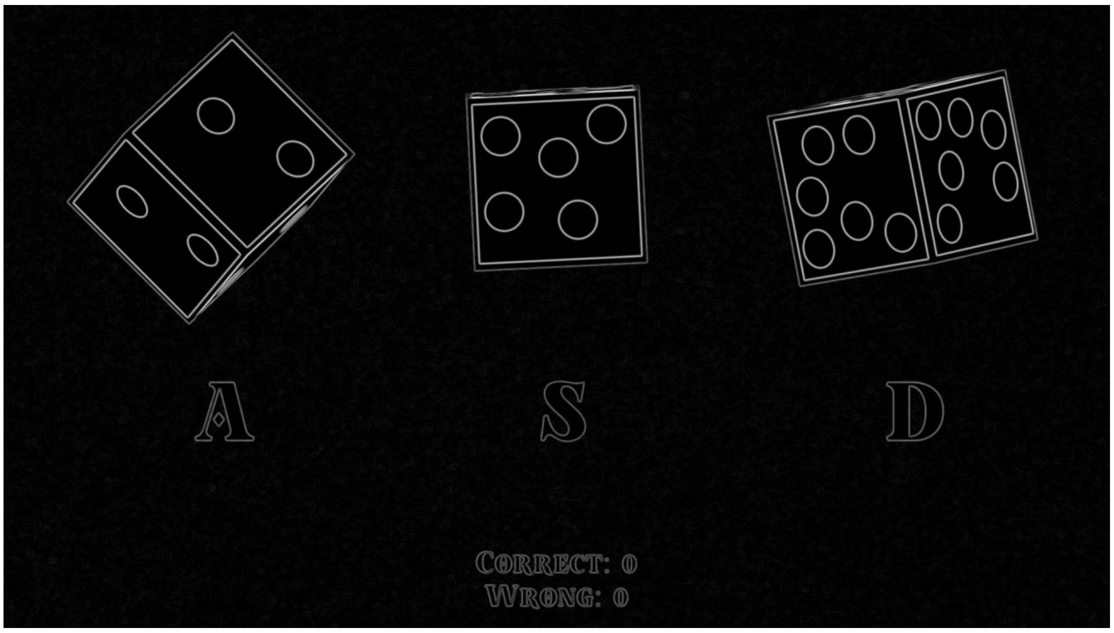


Images of Three Dice After seperating_dice() Function

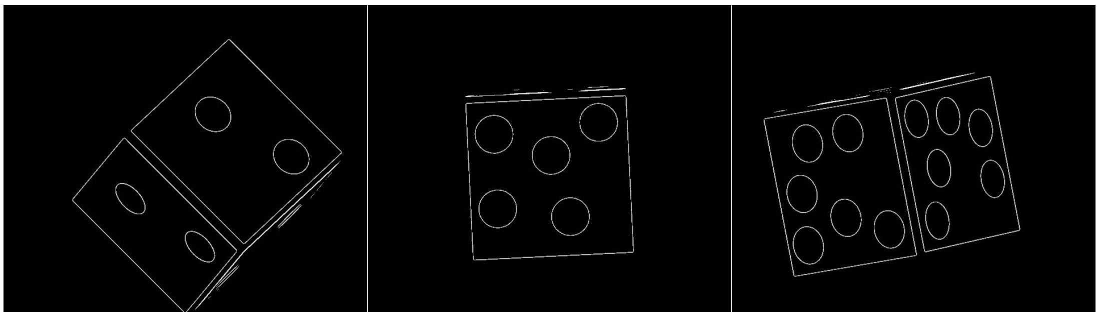


Detected Circles From Dice Images

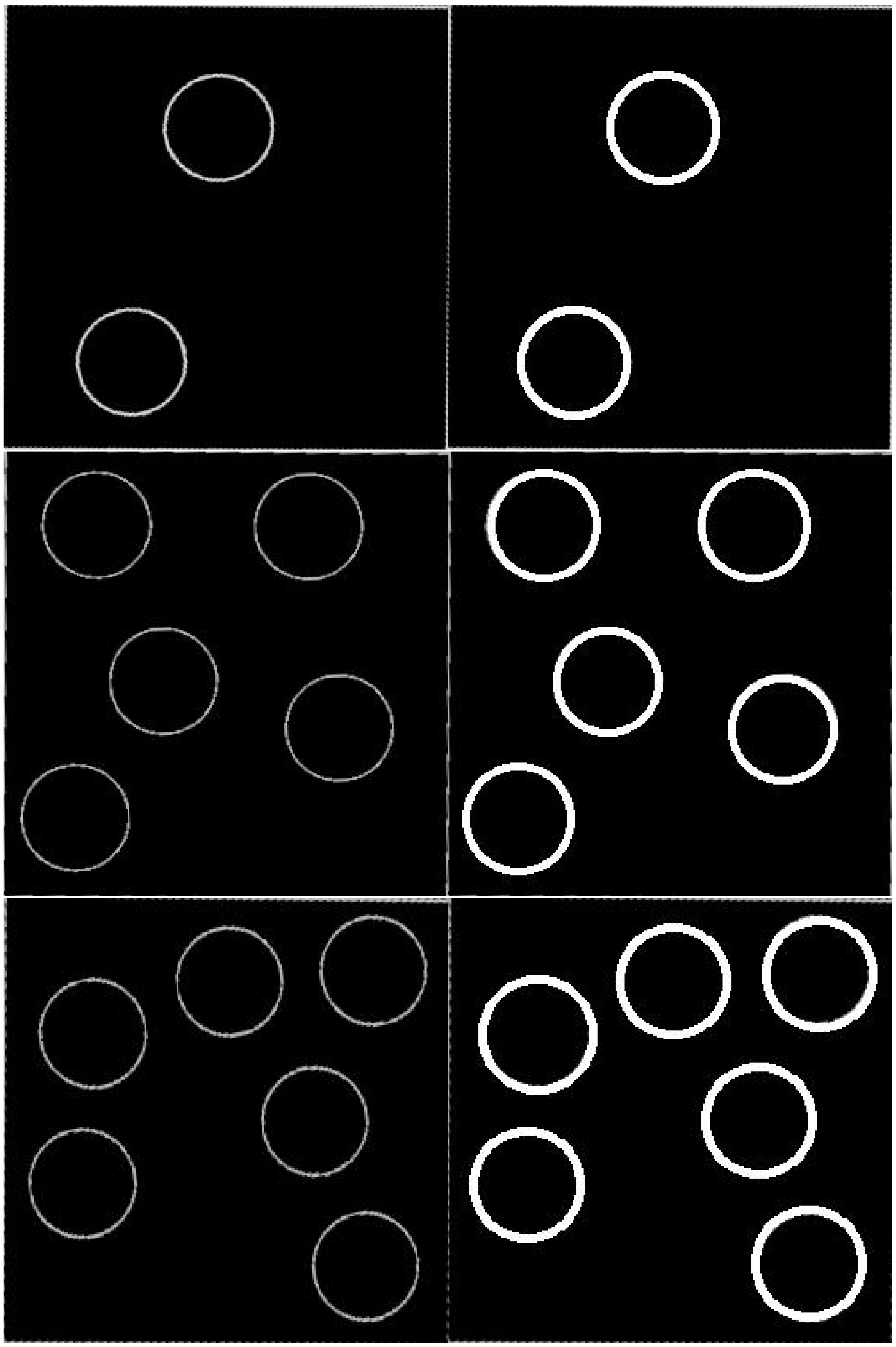
**Game 2:** The only difference of our program for this game from the previous one is, we used find_square() function for detecting one surface from the rotating dice images and performing perspective transform. Circle detection function is used on these transformed dice images. In the find_square() function, we used cv2.findContours() function for detecting the borders of surfaces that can be seen in the dice images. After detecting all the surfaces, we determined the surface which we will apply perspective transform by finding the surface which has the largest area. After finding the borders of the largest square, we found and ordered the corners with order_points() function. We used cv2.getPerspectiveTransform() function to obtain the transformation matrix by using these ordered points. Finally, cv2.warpPerspective() function is used to perform the transformation. Results of our functions for an example game screenshot is shown below.



Result of Sobel Filter Edge Detection



Images of Three Dice After seperating_dice() Function

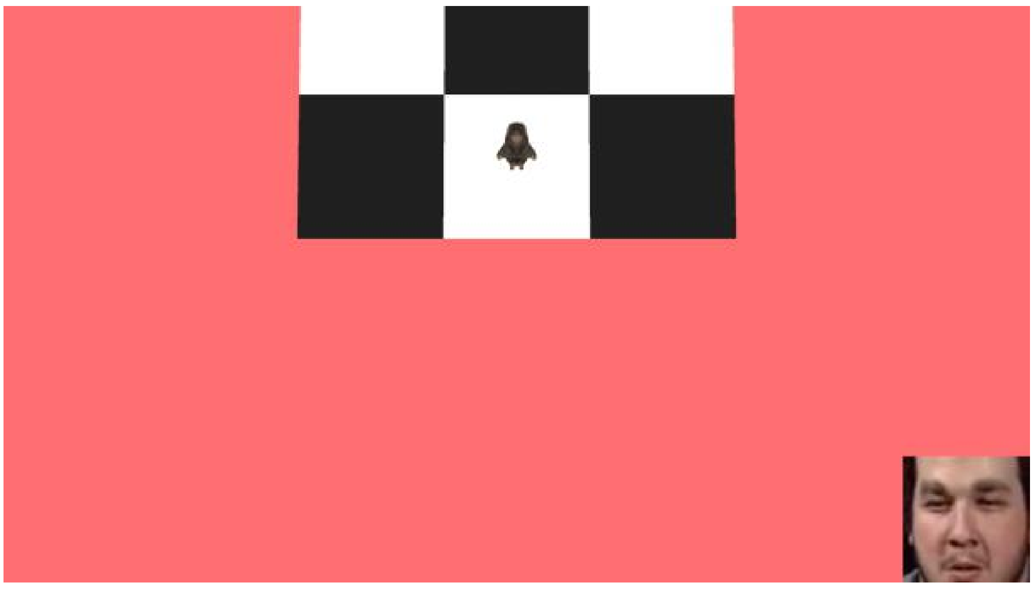


Detected Circles From Transformed Dice Images

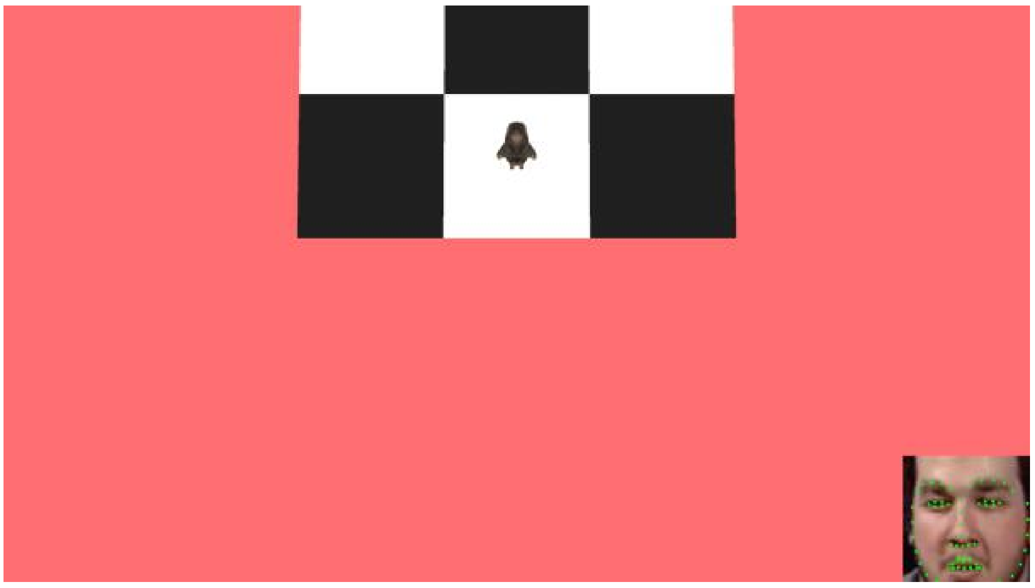
**Part 2: Mine Game**

term_2.ipynb is implemented for the games.

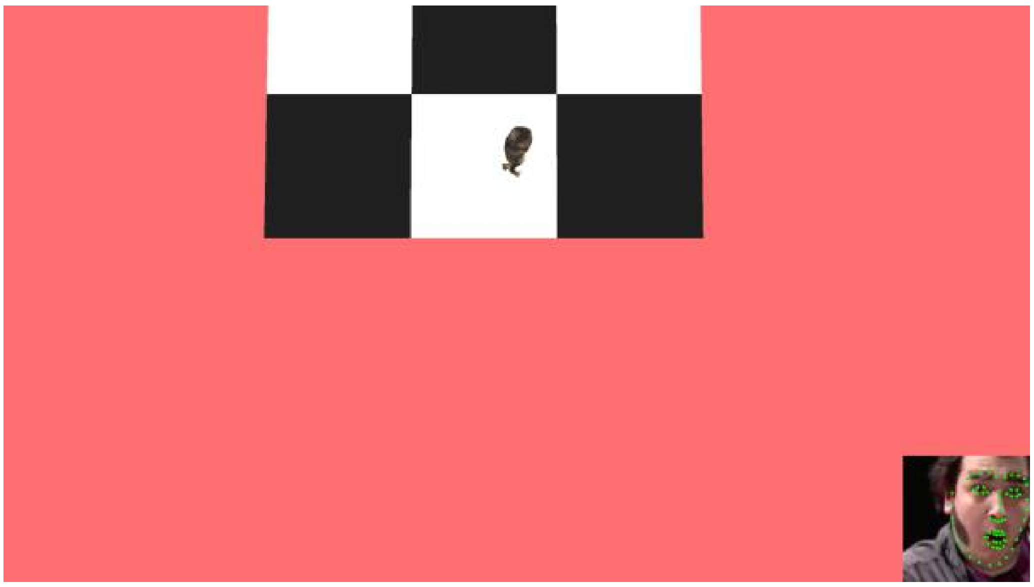
**Game :** When we run the application code, first we saved a screenshot. From this screenshot we took Şahan's first facial expression and saved it as "normal face". Then we turned to the right square and saved Şahan's changing facial expression as "shock face".



*First Screen*

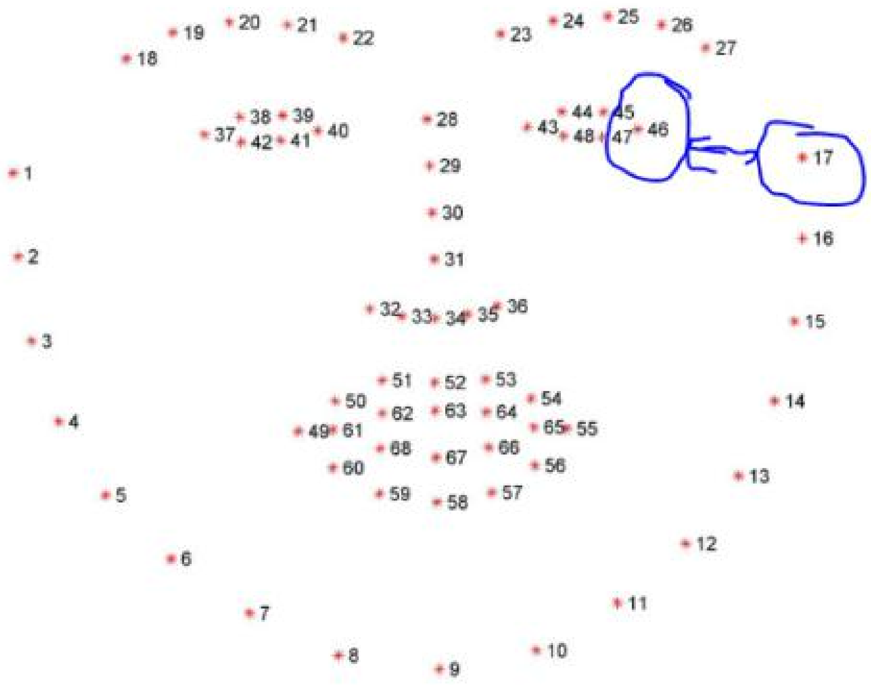


*Şahan Normal Face*



*Şekil 3 Şahan’s Shocked Face*

We used dlib library to take advantage of the differences of Şahan’s facial expressions. We used Facial Landmark Detector and compared facial expression at both 2 images of Şahan’s. We determined that the most obvious difference was the 17th point of the jaw and 46th point of the left eye.



*Distance between letf eye to jaw*

***math.sqrt((left_eye[5][0]-jaw[16][0])**2+(left_eye[5][1]-jaw[16][1])**2)***

There was a huge difference between normal face and shock face rates which eased our comparison. (Normal Face > Shocked Face)

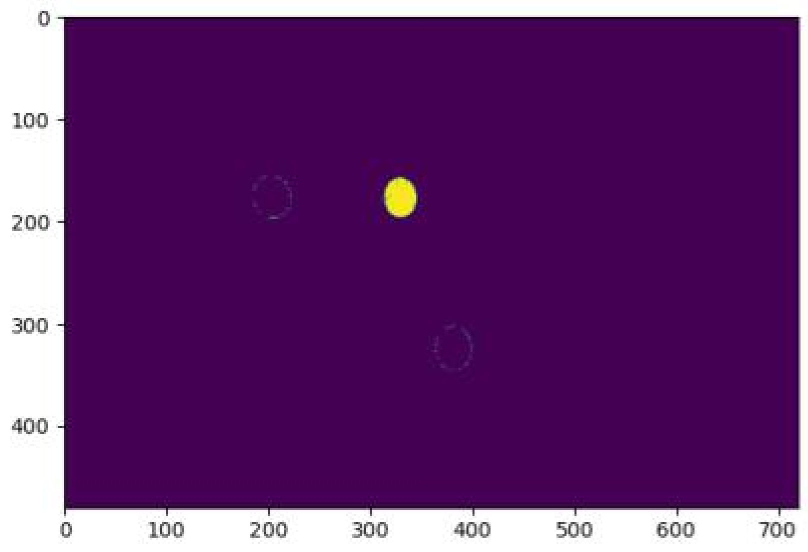
When we first built the game we planned that after detecting the player depending on Şahan's facial expression player would move to the center point of the favorable square. We detected player, squares and center points of the squares however we couldn't move the player therefore we changed the design and moved based on time.

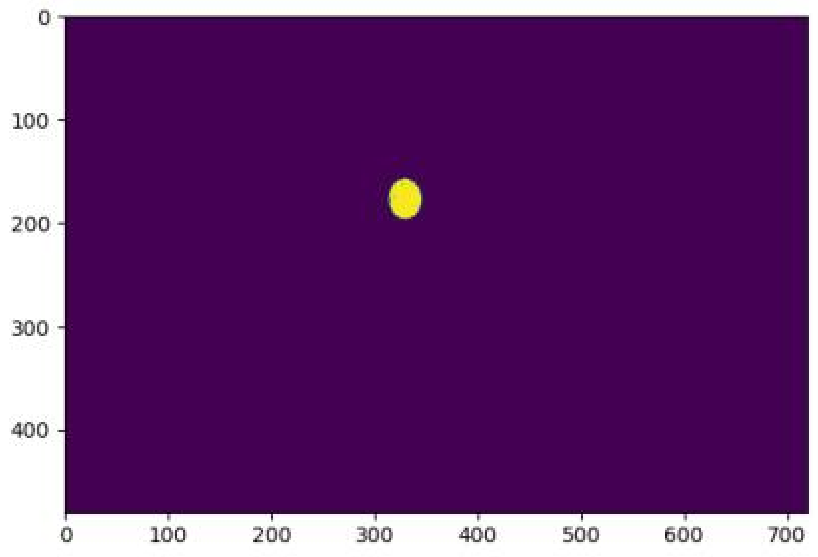
Progression of the game; after saving the differences between the facial expressions player will make a move to left, up, right and down respectively and at his each try after getting close to the next square it will take a screenshot of Şahan's face, determines whether it's a normal of shock face and returns to its former position. If the facial expression is a shock face player will make a move to another square. If the facial expression is a normal face then player will move and tries to stay as close to the center of the square. Thus it will scan surrounding squares till the end of the game.

**Part 3: Bouncing Balls**

For this part, we used frames from the bouncing balls video to compute optical flow vectors. In the for loop, each frame is captured and find_balls() function is used for masking each ball. It returns four grayscale images of the frame which have only one of the balls. After obtaining separate images of the balls, find_V() function is used for calculating the optical flow vector values. This function takes three parameters as the grayscale image of one ball, grayscale image of the next frame, and window size. Before computing, std() function is used on the ball image because after masking, some points that are not inside the ball may be included. This function calculates the standard deviation of ball coordinates, and makes the points outside the ball equal to zero by checking their distance to the middle point of the ball. After this, middle point is determined again for calculating optical flow vector. If the window size is equal to zero, it computes the vector by using all points of the corresponding ball. Optical flow vector is computed by solving $V = -(A^TA)^{-1}A^Tb$ equation. After each computation of optical flow vector, we added the displacement amount for each ball $((x^2 + y^2)^{1/2})$ to x_red, x_green, x_blue, x_pink variables. At the end, this x_red, x_green, x_blue, and x_red values are compared and printed by using compare() function.

We tried different sizes of windows while implementing find_V() function. They gave slightly different numbers, but the order of the velocities was not changed. Different window sizes can be used by giving the third parameter as the requested window size.





Example Images of a Ball Before and After Using std() Function

**Part 4: Vascular Segmentation**

Before starting this part, we examined the images by using Aliza. After understanding the concept, we started to implement region growing algorithm. At the beginning, our program reads NIfTI files and obtains images. After that, it sets the seed values for 2D and 3D operation. For selecting these seed points, we plotted the 3D objects’ different depth images, and tried to choose appropriate points which have the value 1.

region_growing_2D() and region_growing_3D() functions are implemented to perform region growing algorithm in 2D and 3D. These functions take image, seed points, requested neighbor amount and the threshold value as parameters. Inside the region_growing_2D() function, for each given seed point a while loop calls the corresponding neighbor finding function. Then these neighbor points and the seed points are analyzed with the given threshold value. If they are in the same region, that point is converted to one in the “done” array. If they are not from the same region, corresponding value is set to 20 because if it stayed as 0, that point will be considered as not checked yet. This “done” array is used for finding the new points that are set to one in the last while loop operation and stacking them to the “points” . And in the next while loops, these points in the “points” array are used as new seed points. Region_growing_3D() function does the same operations. The only difference is that the third dimension is added to the calculations.

find_8_neighbors(), find_4_neighbors(), find_26_neighbors(), and find_6_neighbors() functions are used for finding new neighbor points to analyze. The first parameter they take is “points” : current seed points. And they take the height, width, and depth of the object for deciding if the calculated neighbor points are inside the object. Only the ones inside the image are returned.

We tried different threshold values and used the one which gave the best result in our program. Dice scores of our segmentation operations are calculated as :

| | |
|---|---|
| score of 8-neighbors: | 0.8971897119629967 |
| score of 4-neighbors: | 0.8960251095027287 |
| score of 26-neighbors: | 0.9084382242013904 |
| score of 6-neighbors: | 0.9090867746043296 |