01/02/2021

İSTANBUL TECHNICAL UNIVERSITY

Faculty of Computer Science and Informatics

TERM PROJECT

TERM PROJECT REPORT

Serhat DEMİRKIRAN, Sercan AYDIN

150170719, 150170707

Instructor: Gözde Ünal

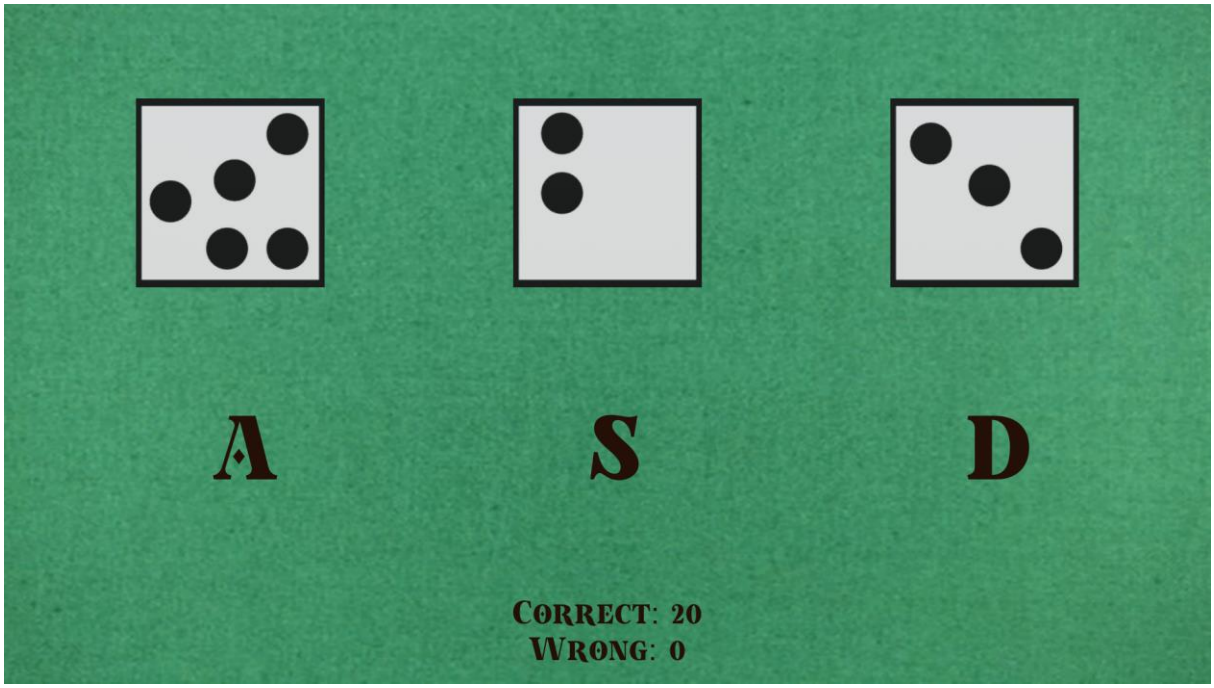
Teaching Assistant: Yusuf Hüseyin Şahin

Course Title: Computer Vision

Course Code: BLG 453E

1) Part I: Dice Game

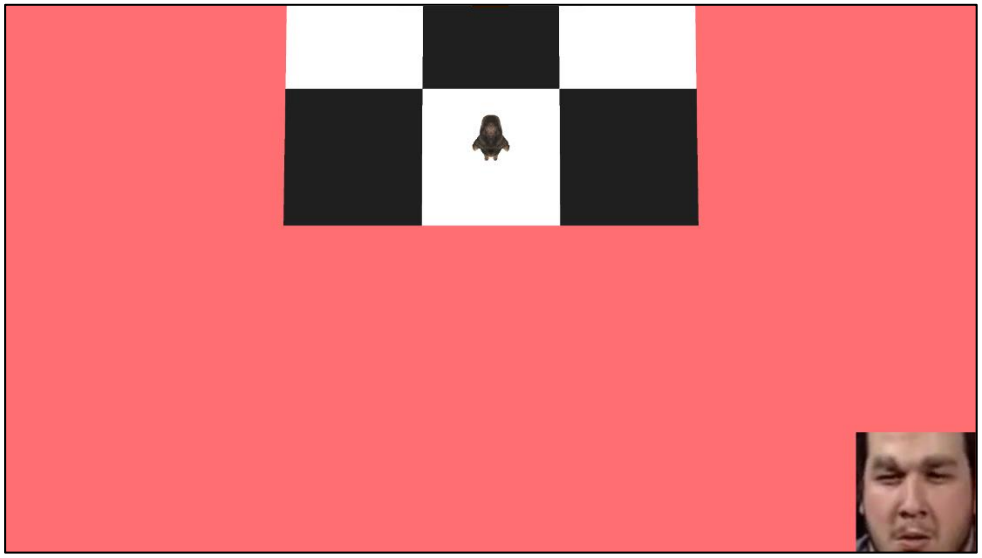
In part1.1, we implemented Hough circle detection algorithm to find the dots in given dices screenshot.



2) Part II: Mine Game

Goal: The character in the mine game should reach the last grid:

My screen resolution: 1920 x 1080

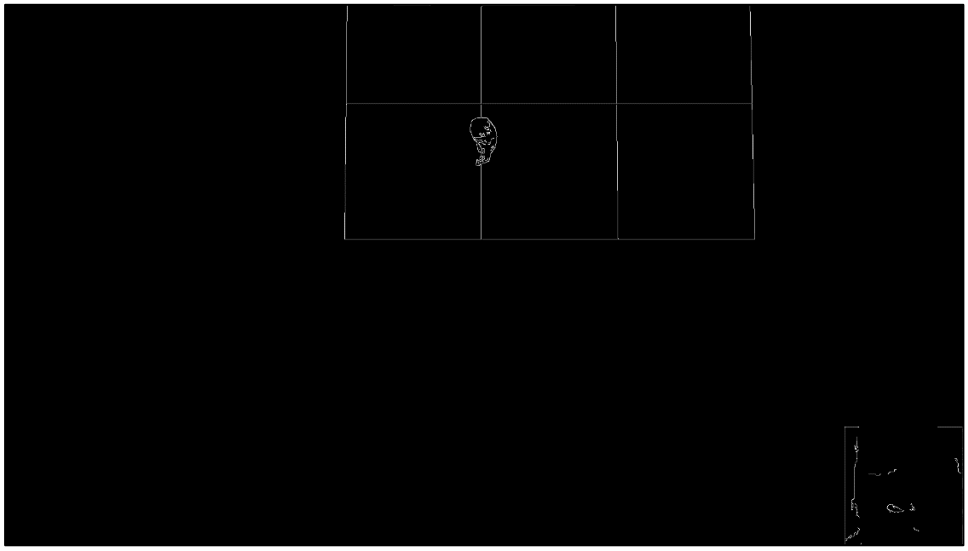


Helper Functions:

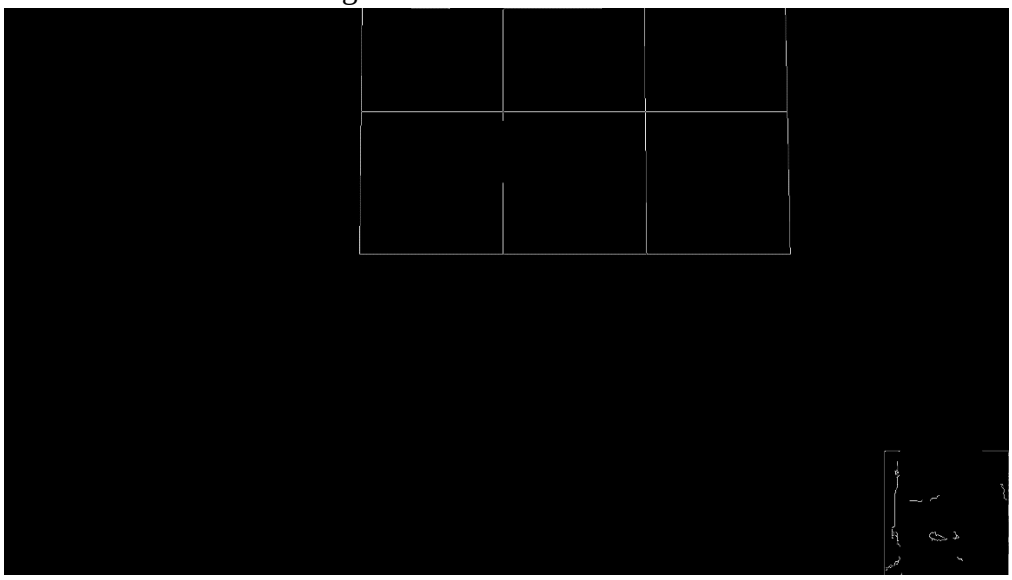
- Bool shockedFace (img_gray)
- Bool onEdge (img_gray, direction)

shockedFace(img_gray): It takes the cropped gray scale image and returns True if it is shocked face. After cropped the face at right bottom, I applied face detection as in hw2. I compared the x positions of point 36. X positions of points of shocked face is much bigger than normal face. For normal face, x position of point 36 is near 50. On the other hand, x position of point 36 for shocked face is near 120. So its clear that I just can compare these two numbers to detect shocked face.

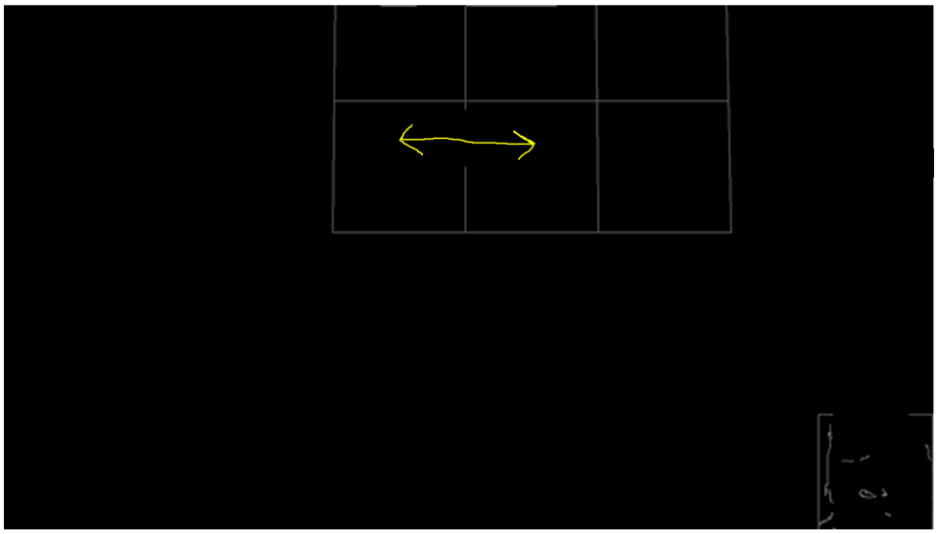
onEdge(img_gray, direction): For given screenshot and move direction of character, it tells whether the our character is on the edge or not. First, I apply edge detection on screenshot. Below is a image after edge detection.



After edge detection, as I know the x.y position of the character(it doesn't change), I remove the character from the image.



After removing the character, some part of the edge also disappears. I am able to detect this by checking the nearby pixels . If I cant find any white pixel, it means I removed the some part of the edge which shows the character is on the edge.



Game Strategy: I created three 7 x 11 matrix. One of them marks game area, **mine_map** keeps the positions of mine and **visit_map** for the visited squares.

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=uint8)
```

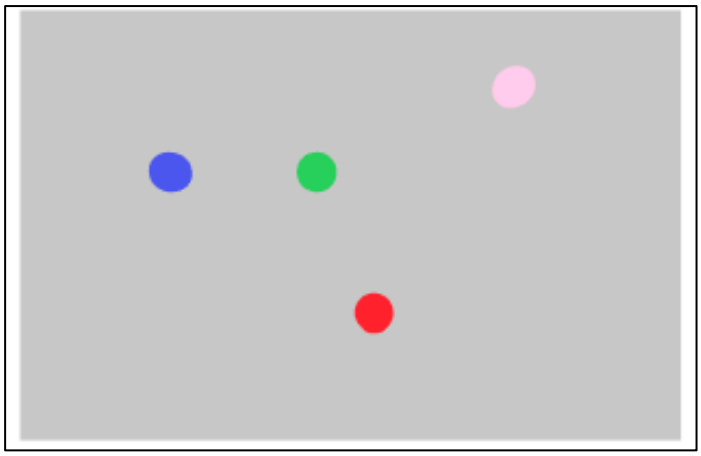
Map matrix

Algorithm:

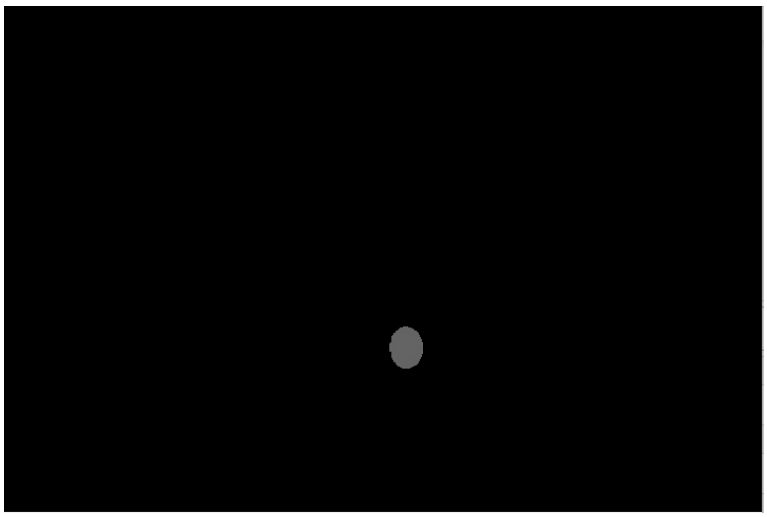
1) Check neighbors. Select the one that we didn't visit before and have no mine. Then move to that square.
2) While moving, take screenshots and check for shocked face.
    a. If a shocked face is detected at right bottom, stop moving. Mark the point as mined. Turn around and move enough to escape from that shocked face.
    b. If no shocked face is detected, keep moving
3) Check screenshot as the character could be on edge. If the character is on the edge, add the point to the visited_map. Than hold the appropriate arrow key for the character to cross the edge securely and reach next square. Jump to the first step.
4) Final condition: If the character reaches the last grids, its done.

3) Part III: Bouncing Balls

Goal: Find and compare the average speed of each ball in the video.



For every ball, I did background subtraction before applying Lucas Kanade for more accurate results. For example, for red ball:



Method:

I used **cv2.calcOpticalFlowPyrLK** built-in function from OpenCV. We provide two consecutive frame and a point to track for the function. It returns next point if it finds any. Parameters for the function are :

```
lk_params = dict( winSize  = (51,51),
                  maxLevel = 3,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

Metric and Results:

$$\sqrt{(x2-x1)^2+(y2-y1)^2}.$$

For consecutive frames, we have (x1,y1) and (x2,y2) that we track. After calculating the amounts and summing all these values, I took average of these values. Below are results:

Red Ball: 13.422 Pixel/Frame

Blue Ball: 13.411 Pixel/Frame

Green Ball: 14.692 Pixel/Frame

Pink Ball:14.085 Pixel/Frame

4) Part IV: Vascular Segmentation

References