BLG453E
Term Project

Res. Asst. Yusuf H. Sahin
sahinyu@itu.edu.tr

- You should write all your code in Python language.
- Cheating is highly discouraged. If you are planning to use different libraries or functions, please ask me about it.
- You will be graded according to your report, code and demo.
- Please state your screen resolutions in your report.
- You are allowed to use image reading/writing operations, warpPerspective and Harris corner detector from OpenCV at this point. However, if you want to use another function, please ask for it using the Message Board.
- You can download the material from https://web.itu.edu.tr/sahinyu/TermProjectFiles.zip

1 - Part 1: Dice Game (25 pts.)

Input: *A simple computer game prepared in Unity3D,* ***Expected Output:*** *Good scores*

In this part of the project, you will write a Python code to play a dice game. As in the 3^{rd} homework, your script will use *pyautogui* to take screenshots from the game and simulate pressing the keyboard buttons. In the game, three dice are shown to the player and the player should select the die with the greatest number. There are two game types depending on whether the dice are randomly rotating. The Game 1 and Game 2 screens are as given in Figure 1 and Figure 2. You will select one of them, switch to your IDE/terminal and start your program, switch back to game to watch your program playing the game. You should simulate A, S and D buttons to select a die.

Here, the following strategies should be followed:

- **Game 1:** Every die should be detected using edge detection. Then, you should implement Hough Circle Detection algorithm to count the dots on these dice. In addition to Hough Circle detection, you can also make a connected components analysis to be sure about your results.

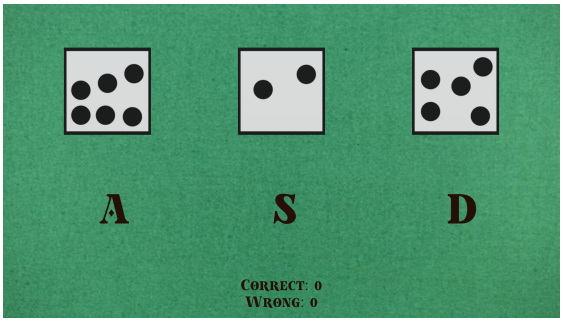

Figure 1: Dice Game: Game 1

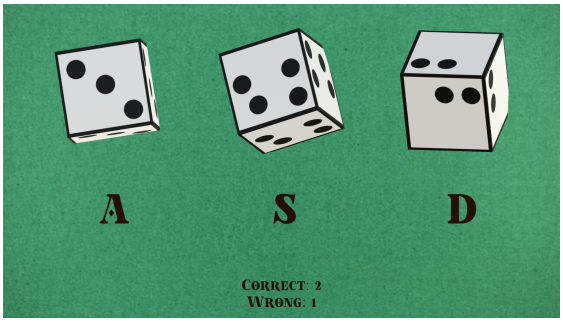

Figure 2: Dice Game: Game 2

- **Game 2:** Since the dice are rotating, you should calculate a perspective transform for the square on top surface to flatten the surface. Then you should use your Hough Circle Detection algorithm on this flattened image. As in the previous part, you can also use connected component analysis. (**Hint:** It is a good idea to use more than one screenshots. You can also define an error rate for Hough Circle Detection.)

2 - Part 2: Mine Game (25 pts.)

Input: *A simple computer game prepared in Unity3D,* ***Expected Output:*** *The character should reach the last grids.*

I prepared another game in which the main character is trying to reach the end of the map given in Figure 3. By default the character can be controlled using arrow keys and shift key makes him run faster. You should also use *pyautogui* to take screenshots of the board, decide on the next action and simulate these actions.

The problem with these grids is, some of them are mined. However, if our character is close to a mined grid (according to grid borders), a face image having a "shocked"

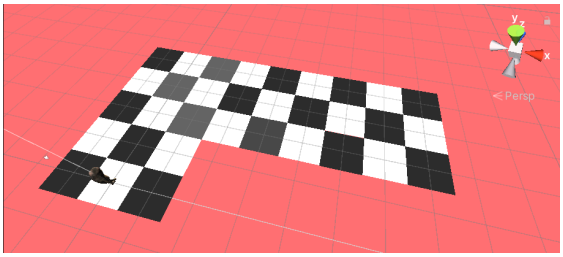

Figure 3: Mine Game: The Map

expression appears on the bottom right as given in Figure 4.

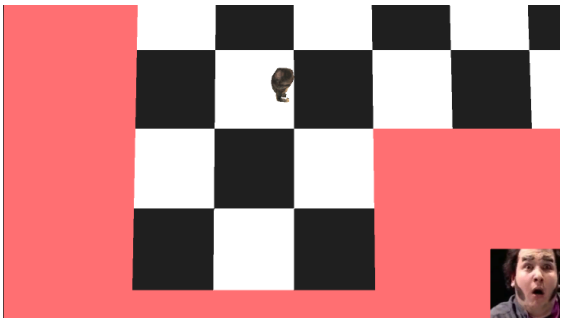

Figure 4: Mine Game: The character is very close to a mined grid.

Here, the following strategies should be followed:

- Using facial landmark detection, detect the difference between "shocked" face and "normal" face. Try to not enter mined grids.
- You can keep a list of visited grids so that, the character should not tend to visit some grids over and over. Thus, you should benefit from corner and edge detection to detect whether the character is entered to another grid.
- Make the character reach the last grids safely.

3 - Part 3: Bouncing Balls (25 pts.)

Input: *Video,* ***Expected Output:*** *A comparison of the ball velocities.*

With the project files, a video in which four different balls continuously moving in a rectangular field as in Figure 5 is given. Each ball is trying to bounce on a different wall. However, since they hit each other, they also deflect. Compare the average speed of each ball. Use optical flow algorithm.

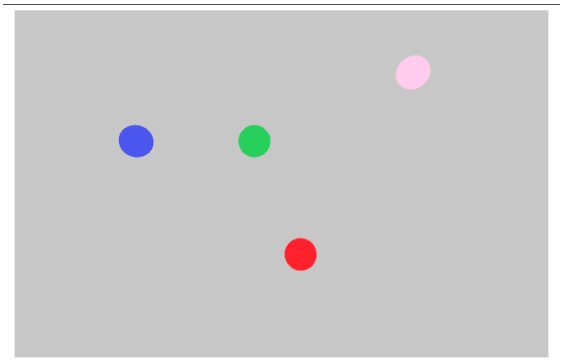

Figure 5: Bouncing Balls: A frame from the video

4 - Part 4: Vascular Segmentation (25 pts.)

Input: *NIfTI files of vascular structure and its ground truth,* ***Expected Output:*** *NIfTI files for each segmentation task and their Dice scores.*

With the project document, I've also uploaded a 3D image and its segmentation. An example look of the image is given in Figure 6. As stated by Jerman et. al [1] , in medical imaging, scans are generally corrupted by a Poisson noise. Thus, to obtain a reliable synthetic image (e.g. to test a vessel segmentation algorithm), a certain Poisson noise may be applied. Thus, I applied a Poisson noise with ($\lambda = 1$) onto the image.

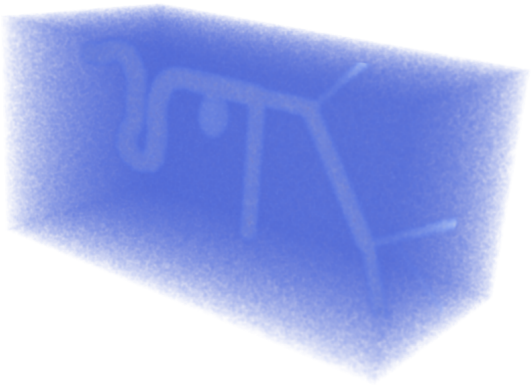

Figure 6: Vascular Segmentation: 3D Image

In this part, you will implement region growing algorithm using different parameters

[1] Jerman, T., Pernus, F., Likar, B., Spiclin, Z. (2015). Blob enhancement and visualization for improved intracranial aneurysm detection. IEEE Transactions on Visualization and Computer Graphics, 22(6), 1705-1717.

and approaches. You are free to select growing criteria and threshold value. For each task given below, you should calculate the Dice coefficient which indicates the overlap between segmentation result and ground truth. Dice coefficient can be calculated as;

$$Dice(X_{seg}, X_{gt}) = \frac{2 * |X_{seg} \cap X_{gt}|}{|X_{seg}| + |X_{gt}|} \tag{0.1}$$

where X_{seg} represents the algorithm's segmentation result and X_{gt} represents the ground truth. **Try to obtain high Dice scores since good scores will also affect your grade from this part.**

The provided files are in NIfTI file format, which is widely used in medical imaging. You can use nibabel library[2] to read and write NIfTI files. Also to visualize the the image and your results, you can benefit from Aliza [3] (2D slices and 3D look of the image) and ITK-Snap [4] (2D slices and some interactive operations).

- **Task 1:** Select at most 15 seed points. Implement region growing algorithm to obtain a segmentation output similar to the given ground truth. Obtain independent results for each axial (z-dimension) slice. Use 8-neighborhood. Calculate the Dice score.
- **Task 2:** Reimplement the approach in Task 1, but use 4-neighborhood.
- **Task 3:** Select at most 5 seed points. Implement the region growing algorithm in 3D. Use 26-neighborhood. Calculate the dice coefficient.
- **Task 4:** Reimplement the approach in Task 3, but use 6-neighborhood.

[2] https://nipy.org/nibabel/
[3] https://www.aliza-dicom-viewer.com/
[4] http://www.itksnap.org/pmwiki/pmwiki.php