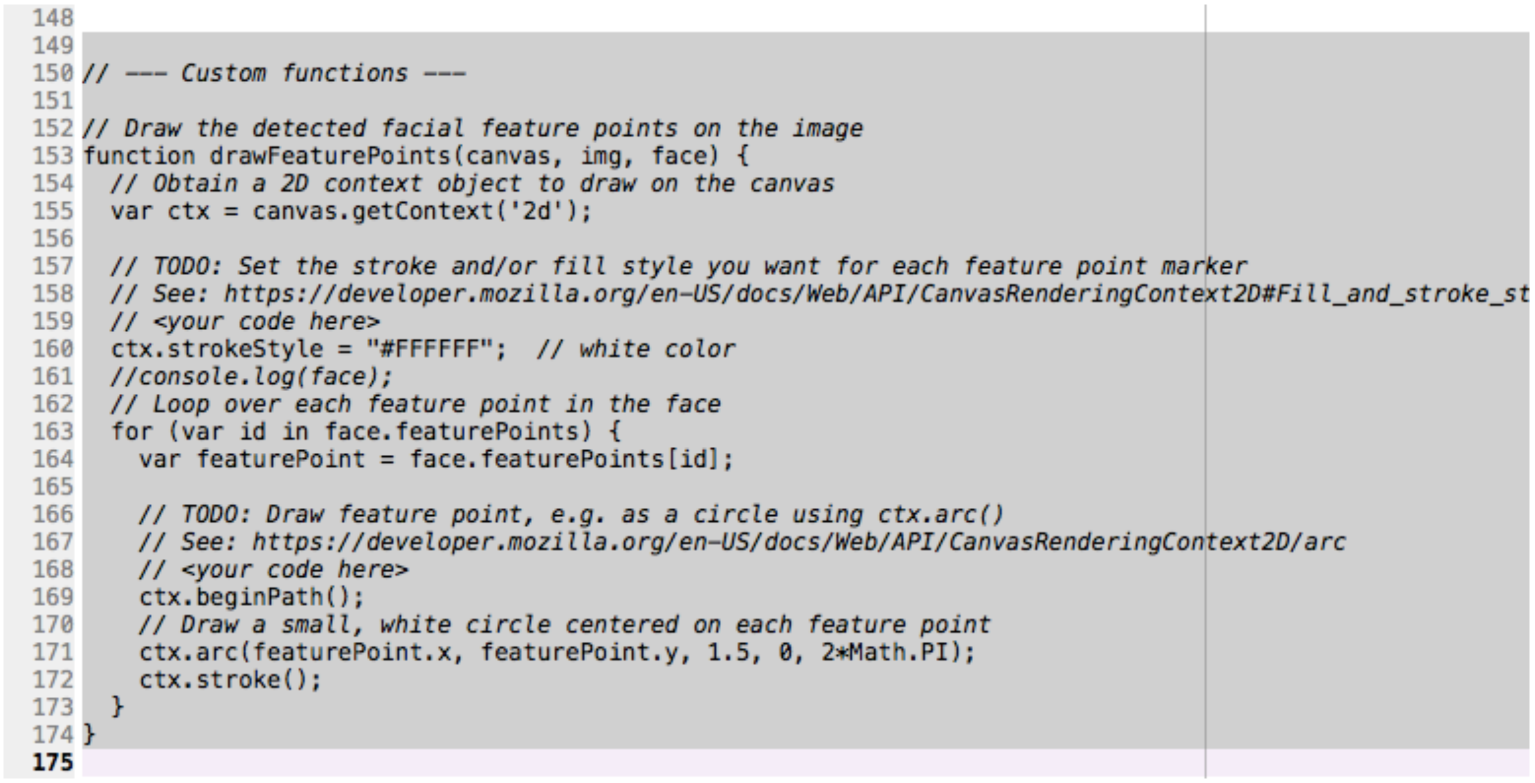
“Mimic Me” Project Report

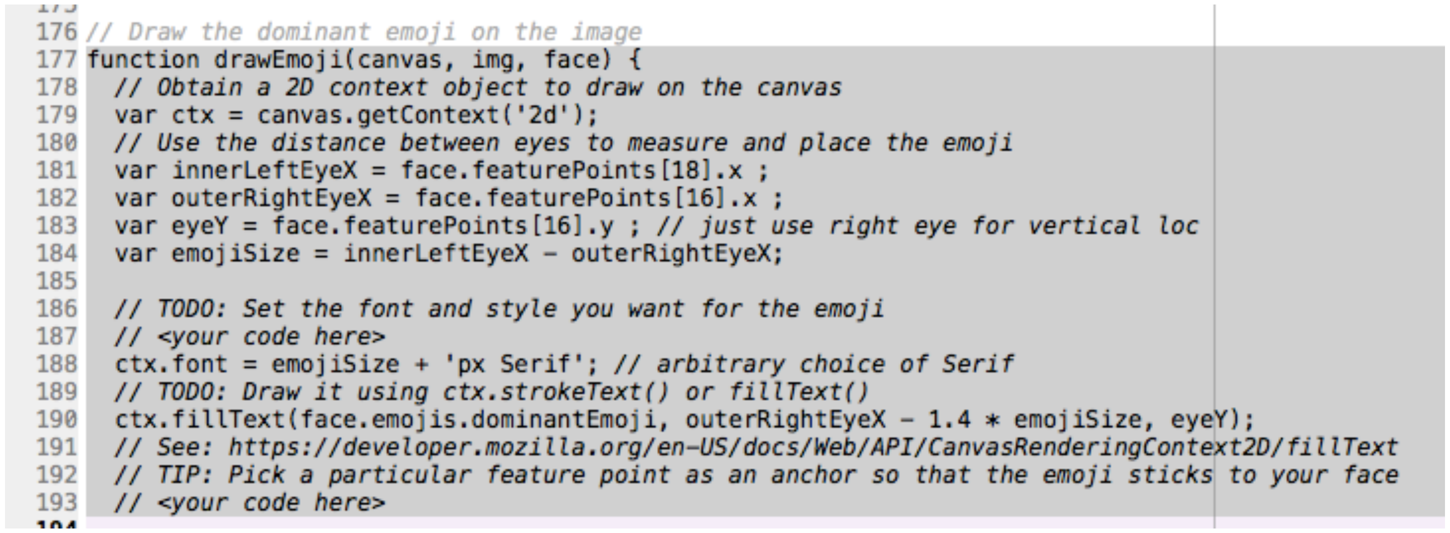
Task 1— Display Feature Points:



```

// --- Custom functions ---

// Draw the detected facial feature points on the image
function drawFeaturePoints(canvas, img, face) {
  // Obtain a 2D context object to draw on the canvas
  var ctx = canvas.getContext('2d');

  // TODO: Set the stroke and/or fill style you want for each feature point marker
  // See: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D#Fill_and_stroke_st
  // <your code here>
  ctx.strokeStyle = "#FFFFFF";  // white color
  //console.log(face);
  // Loop over each feature point in the face
  for (var id in face.featurePoints) {
    var featurePoint = face.featurePoints[id];

    // TODO: Draw feature point, e.g. as a circle using ctx.arc()
    // See: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/arc
    // <your code here>
    ctx.beginPath();
    // Draw a small, white circle centered on each feature point
    ctx.arc(featurePoint.x, featurePoint.y, 1.5, 0, 2*Math.PI);
    ctx.stroke();
  }
}

```

Using an Affectiva CameraDetector object, as already implemented in the starter code, and an EventListener hooked up to it and the computer’s webcam, the callback from the Listener provides a ‘faces’ object (of which the first face is passed into the above function, as ‘face’). This face object contains several key feature points and their coordinates in the image that accompanies the callback (passed in as ‘img’ in the above function). For Task 1, it’s just a matter of using the canvas provided in the starter code and passed into this function, choosing a color, size and shape for each point, and drawing the shape onto the canvas at each feature point. Then this function is called every time the Listener successfully processes a new image.

Task 2 — Show Dominant Emoji:

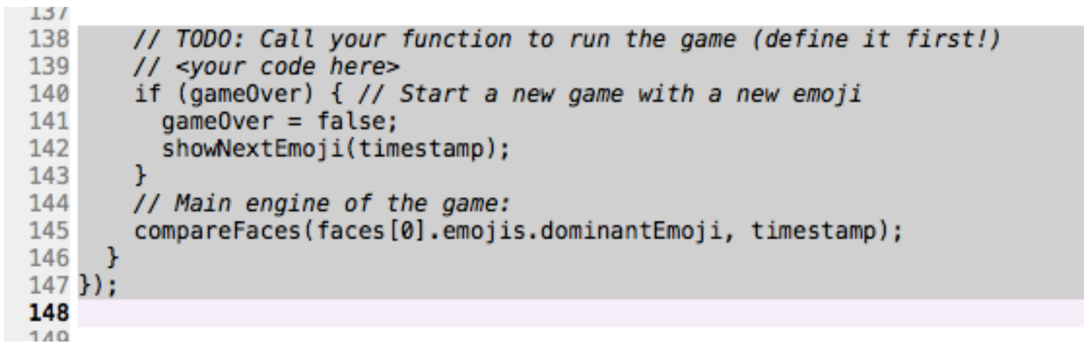


```
// Draw the dominant emoji on the image
function drawEmoji(canvas, img, face) {
  // Obtain a 2D context object to draw on the canvas
  var ctx = canvas.getContext('2d');
  // Use the distance between eyes to measure and place the emoji
  var innerLeftEyeX = face.featurePoints[18].x ;
  var outerRightEyeX = face.featurePoints[16].x ;
  var eyeY = face.featurePoints[16].y ; // just use right eye for vertical loc
  var emojiSize = innerLeftEyeX - outerRightEyeX;

  // TODO: Set the font and style you want for the emoji
  // <your code here>
  ctx.font = emojiSize + 'px Serif'; // arbitrary choice of Serif
  // TODO: Draw it using ctx.strokeText() or fillText()
  ctx.fillText(face.emojis.dominantEmoji, outerRightEyeX - 1.4 * emojiSize, eyeY);
  // See: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/fillText
  // TIP: Pick a particular feature point as an anchor so that the emoji sticks to your face
  // <your code here>
```

As in Task 1, the ‘face’ and ‘img’ objects from the Affectiva Detector’s Listener callback are used in Task 2’s function to calculate a good emoji size and location. I used the difference in x-coordinates of the inside of the left eye and the outside of the right eye as my emoji size, and pinned the emoji 1.4x that distance outside of the right eye, at the same vertical location. As far as which emoji to display, this was again provided by the same face object. As specified in the starter code, I used the canvas’s context’s fillText() method to draw the face’s dominant emoji. I arbitrarily chose Serif as the font family, since I didn’t notice any difference in the other choices I tried.

“Mimic Me” Project Report

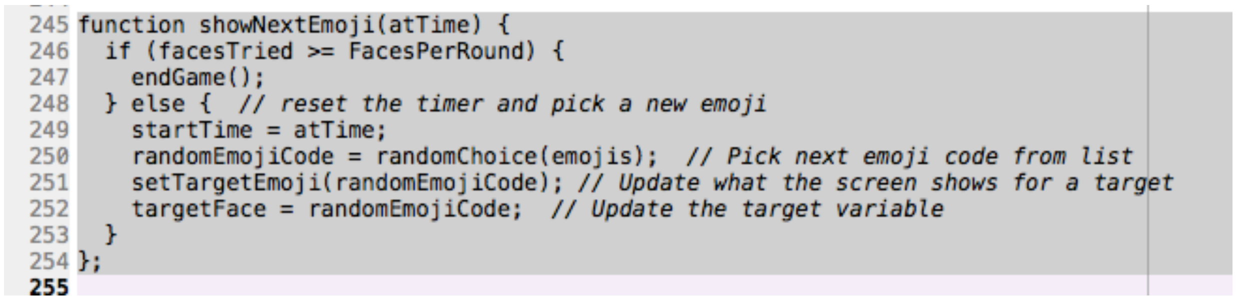
Task 3 — Implement Mimic Me!:



```
    // TODO: Call your function to run the game (define it first!)
    // <your code here>
    if (gameOver) { // Start a new game with a new emoji
      gameOver = false;
      showNextEmoji(timestamp);
    }
    // Main engine of the game:
    compareFaces(faces[0].emojis.dominantEmoji, timestamp);
  }
});
```

Fig. 1

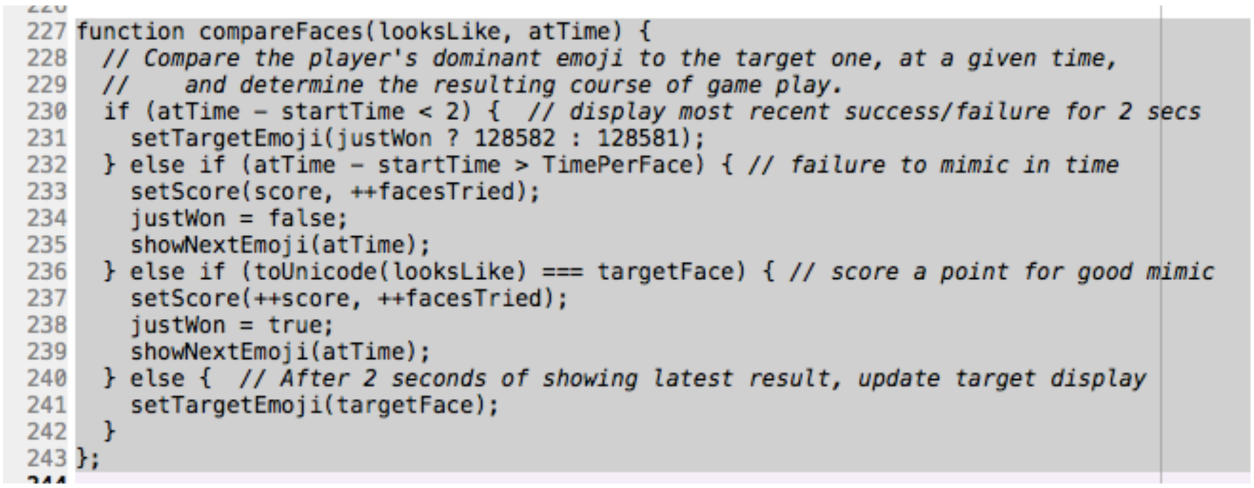
The third parameter provided to the Detector’s Listener callback used in Tasks 1 and 2, which is a timestamp for when the event occurred, turns out to be the key to finishing the gameplay implementation required in Task 3. That’s because the game is a time-based one. So above, in the same callback that called the functions in the first two Tasks, I check if the game needs to be restarted, and if it does, then I show the next emoji for the player to mimic.



```
function showNextEmoji(atTime) {
  if (facesTried >= FacesPerRound) {
    endGame();
  } else {  // reset the timer and pick a new emoji
    startTime = atTime;
    randomEmojiCode = randomChoice(emojis);  // Pick next emoji code from list
    setTargetEmoji(randomEmojiCode); // Update what the screen shows for a target
    targetFace = randomEmojiCode;  // Update the target variable
  }
};
```

Fig.2

As long as the number of faces mimicked hasn’t hit its limit, I reset the timer to the timestamp passed in from Fig.1 and set the new target emoji as a displayed variable. Back in Fig.1, which is the main driver of the game, after the game state has been toggled to not new, the Listener repeatedly calls the compareFaces() function, passing it the dominant emoji calculated from the player’s face and the timestamp. Here’s the main function:



```
function compareFaces(looksLike, atTime) {
  // Compare the player's dominant emoji to the target one, at a given time,
  //    and determine the resulting course of game play.
  if (atTime - startTime < 2) {  // display most recent success/failure for 2 secs
    setTargetEmoji(justWon ? 128582 : 128581);
  } else if (atTime - startTime > TimePerFace) { // failure to mimic in time
    setScore(score, ++facesTried);
    justWon = false;
    showNextEmoji(atTime);
  } else if (toUnicode(looksLike) === targetFace) { // score a point for good mimic
    setScore(++score, ++facesTried);
    justWon = true;
    showNextEmoji(atTime);
  } else {  // After 2 seconds of showing latest result, update target display
    setTargetEmoji(targetFace);
  }
};
```

Fig.3

For the first 2 seconds I show an emoji representing success or failure of the previous attempt, and then replace it with the new target emoji. If the player’s face’s dominant emoji matches the target within 5 seconds, a point is scored and the score displayed, and the flow returns to Fig.2. If time expires, the score is updated and flow returns to Fig.2 as well. There, the timer is reset and a new emoji chosen, or else the game ends. But it is Figure 3 that is repeatedly called by the Listener during the bulk of the game.