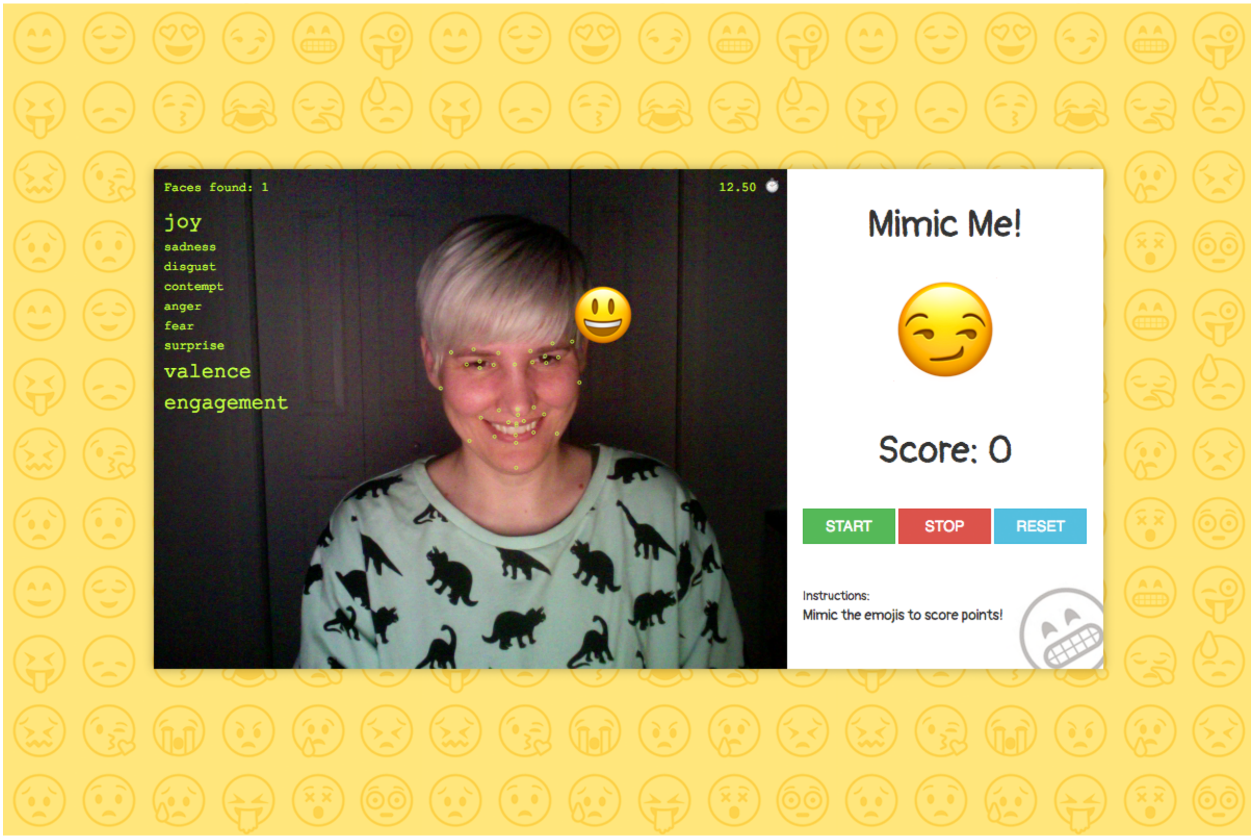
Natacha Fernández
AIND Project: Emotion Recognition with Affectiva
August 7, 2017

Mimic Me!

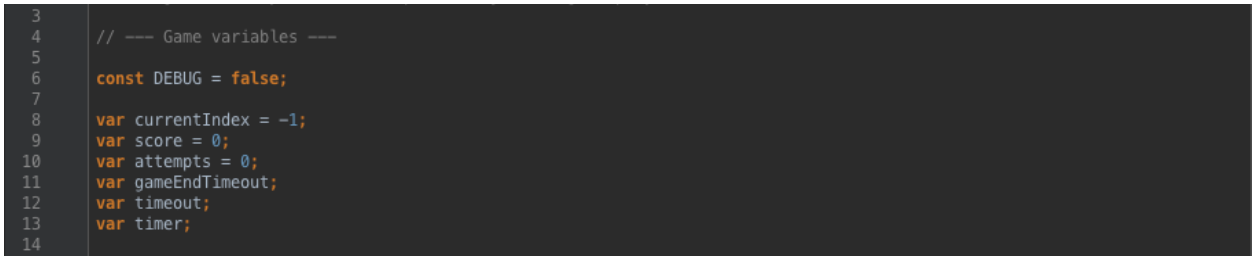
Game using Affectiva's Emotion-as-a-Service API



Summary

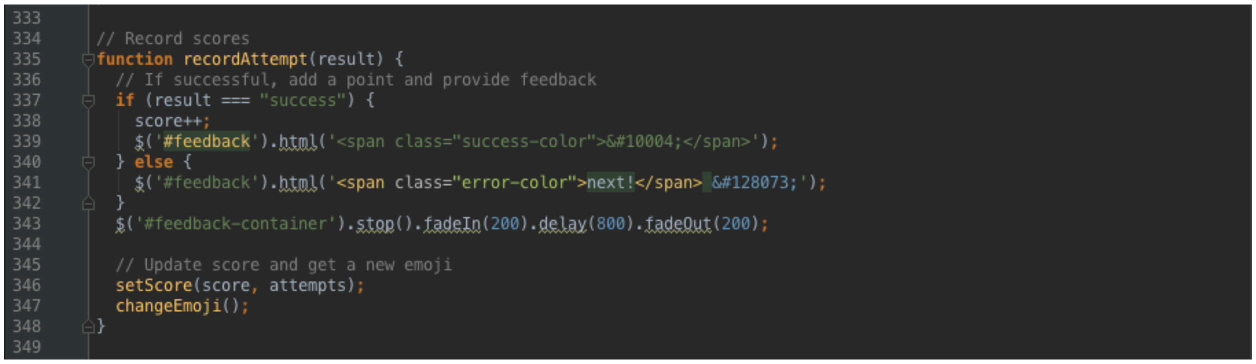
In the ***Mimic Me!*** game the player must mimic a random emoji displayed by the computer to earn points. Starting with the provided code, several additions have been implemented to turn it into a fun game. In this particular version, the player has 8 seconds to correctly mimic a given emoji and a total of 90 seconds to try mimicking as many as possible.

1. Game Variables



```
// --- Game variables ---

const DEBUG = false;

var currentIndex = -1;
var score = 0;
var attempts = 0;
var gameEndTimeout;
var timeout;
var timer;
```

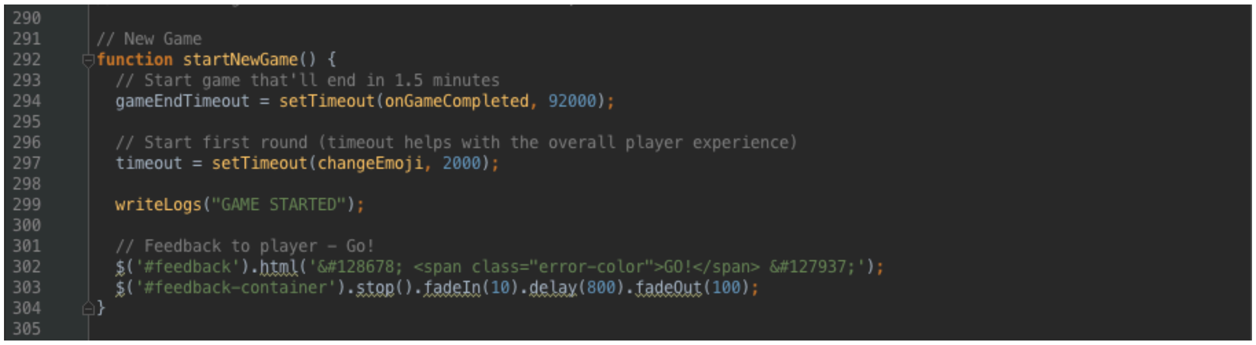
Some game variables are necessary to keep track of progress such as ***score, attempts*** and ***currentIndex*** (current emoji.) Other important variables are ***gameEndTimeout,*** a timeout that brings the end of the game after 90 seconds, and ***timer,*** which calls the ***recordAttempt*** function every 8 seconds to record a failed attempt if player runs out of time trying to mimic the current emoji.



```
// Record scores
function recordAttempt(result) {
  // If successful, add a point and provide feedback
  if (result === "success") {
    score++;
    $('#feedback').html('<span class="success-color">&#10004;</span>');
  } else {
    $('#feedback').html('<span class="error-color">next!</span> &#128073;');
  }
  $('#feedback-container').stop().fadeIn(200).delay(800).fadeOut(200);

  // Update score and get a new emoji
  setScore(score, attempts);
  changeEmoji();
}
```

2. Starting the Game

When the player presses the ***START*** button (an additional larger button was added for an enhanced user experience), the ***onStart*** function is called and the Affectiva ***CameraDetector*** object is started. Once the detector is successfully initialized, the ***startNewGame*** function is invoked and the game begins.



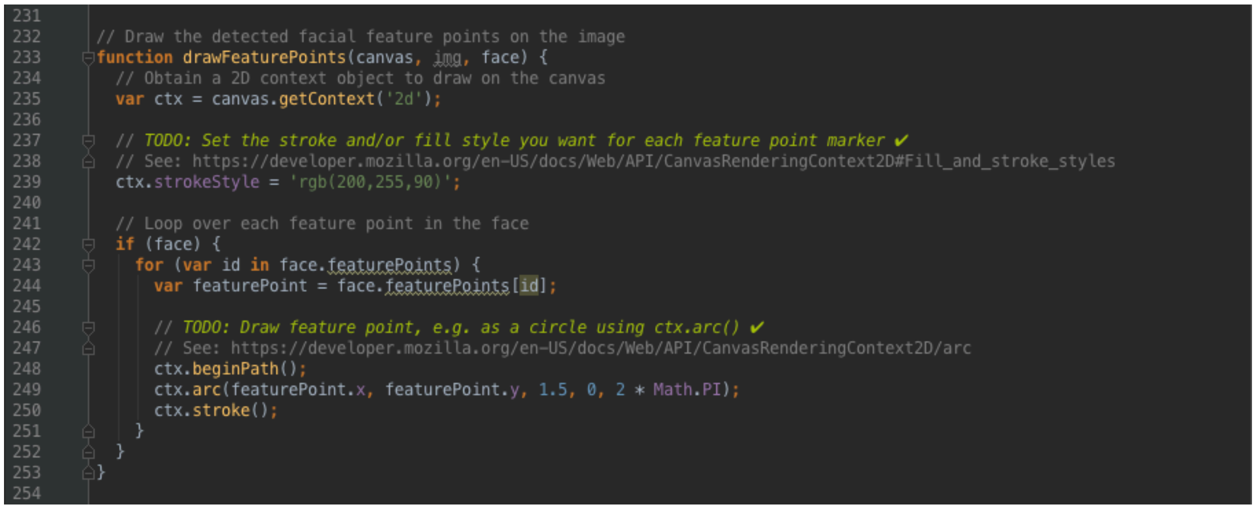
```
// New Game
function startNewGame() {
  // Start game that'll end in 1.5 minutes
  gameEndTimeout = setTimeout(onGameCompleted, 92000);

  // Start first round (timeout helps with the overall player experience)
  timeout = setTimeout(changeEmoji, 2000);

  writeLogs("GAME STARTED");

  // Feedback to player - Go!
  $('#feedback').html('&#128678; <span class="error-color">GO!</span> &#127937;');
  $('#feedback-container').stop().fadeIn(10).delay(800).fadeOut(100);
}
```

3. Updating the Game

In the callback when receiving results from processing the video, these functions are invoked:

- ***drawFeaturePoints*** to draw tracking points on top of the detected face.
- ***drawEmoji*** to draw a reference emoji next to the player's face.
- ***updateGame*** to check if the player has successfully matched the provided emoji.

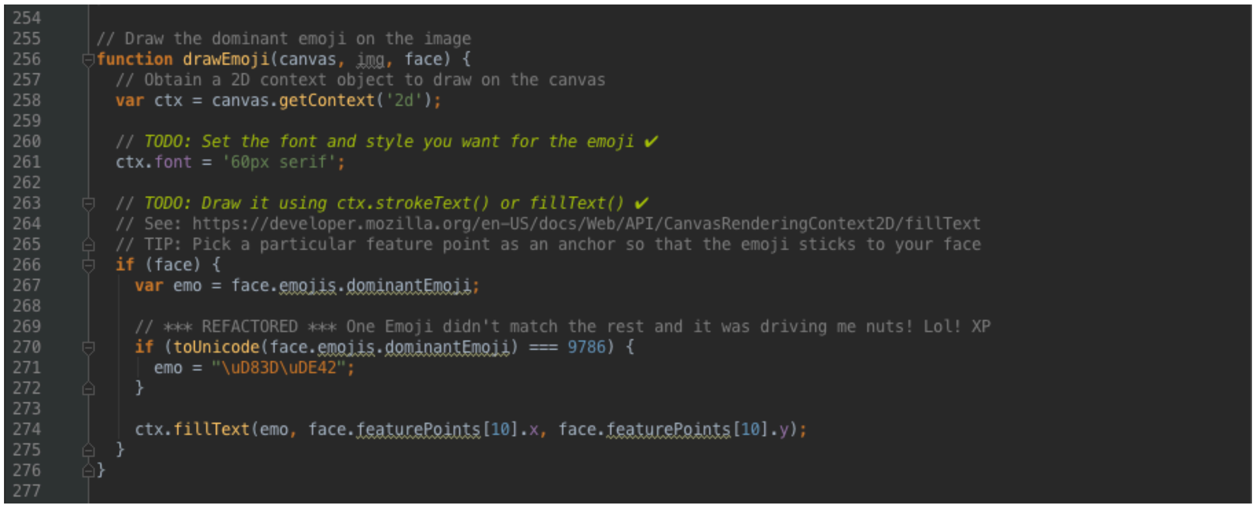
3.1. Display Feature Points (TASK 1)



```
// 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_styles
  ctx.strokeStyle = 'rgb(200,255,90)';

  // Loop over each feature point in the face
  if (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
      ctx.beginPath();
      ctx.arc(featurePoint.x, featurePoint.y, 1.5, 0, 2 * Math.PI);
      ctx.stroke();
    }
  }
}
```

After obtaining the canvas context so we can draw on it, we set a color (expressed in RGB values) for the feature points to be drawn. Then we make sure the ***face*** parameter is not null and proceed to loop through all its ***featuredPoints*** and draw them on the canvas.

3.2. Show Dominant Emoji (TASK 2)

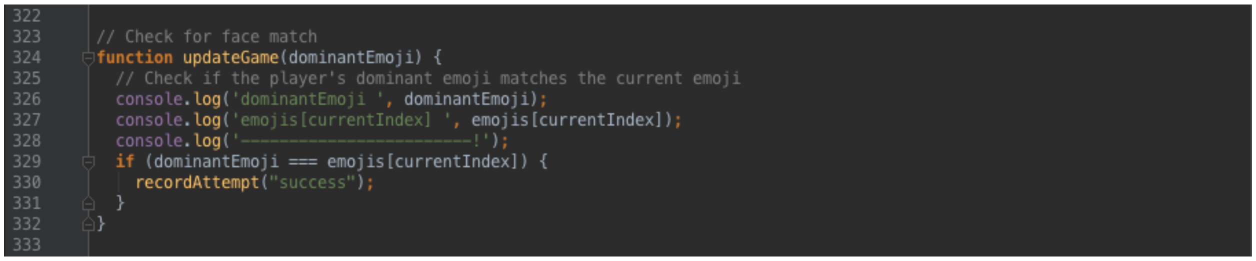


```
// Draw the dominant emoji on the image
function drawEmoji(canvas, img, face) {
  // Obtain a 2D context object to draw on the canvas
  var ctx = canvas.getContext('2d');

  // TODO: Set the font and style you want for the emoji ✔
  ctx.font = '60px serif';

  // TODO: Draw it using ctx.strokeText() or fillText() ✔
  // 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
  if (face) {
    var emo = face.emojis.dominantEmoji;

    // *** REFACTORED *** One Emoji didn't match the rest and it was driving me nuts! Lol! XP
    if (toUnicode(face.emojis.dominantEmoji) === 9786) {
      emo = "\uD83D\uDE42";
    }

    ctx.fillText(emo, face.featurePoints[10].x, face.featurePoints[10].y);
  }
}
```

In this function, we also draw the player's dominant emotion on the image*, to provide feedback to the player so she can adjust her expressions as needed to match the given emoji. We use the coordinates of one of the featured points to attach the dominant emoji so it always appears next to the player's face.

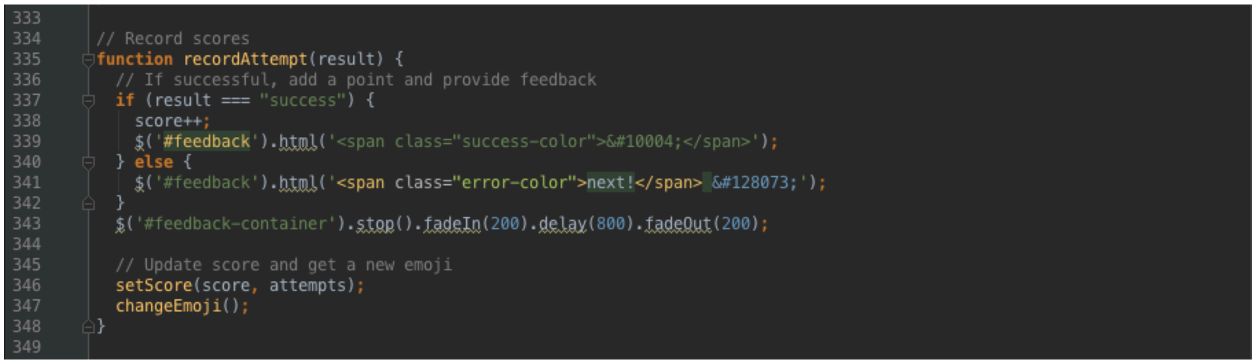
** (Lines 267 - 272) Note that there is an emoji (9786) that does not match the visual style of all other emojis, so it was replaced with one that does match the style of the yellow emojis. This is just an aesthetic enhancement to make the game feel more polished.*

3.3. Checking for a Match



```javascript

// Check for face match
function updateGame(dominantEmoji) {
  // Check if the player's dominant emoji matches the current emoji
  console.log('dominantEmoji ', dominantEmoji);
  console.log('emojis[currentIndex] ', emojis[currentIndex]);
  console.log('------------------------!');
  if (dominantEmoji === emojis[currentIndex]) {
    recordAttempt("success");
  }
}

```

In the ***updateGame*** function we simple check if the dominant emoji matches the given emoji. If there is a match the ***recordAttempt*** function is called to record a successful attempt.



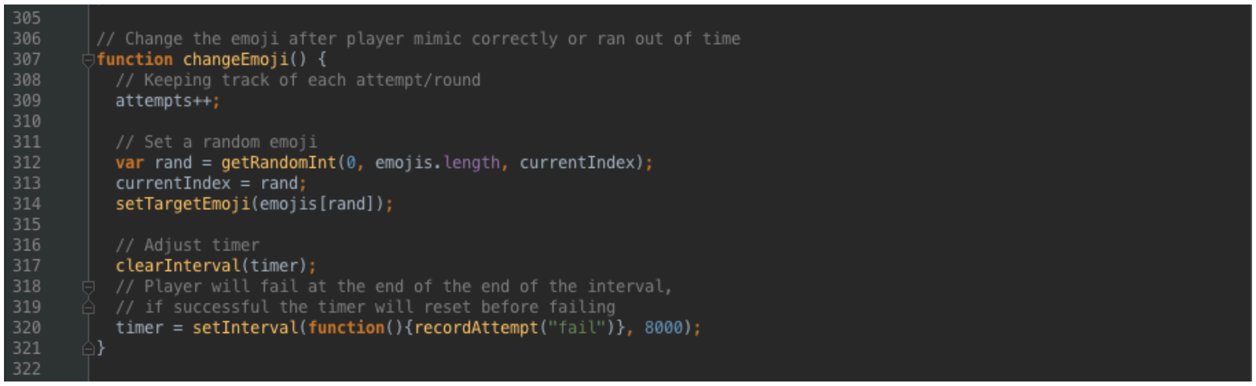
```javascript

// Record scores
function recordAttempt(result) {
  // If successful, add a point and provide feedback
  if (result === "success") {
    score++;
    $('#feedback').html('<span class="success-color">&#10004;</span>');
  } else {
    $('#feedback').html('<span class="error-color">next!</span> &#128073;');
  }
  $('#feedback-container').stop().fadeIn(200).delay(800).fadeOut(200);

  // Update score and get a new emoji
  setScore(score, attempts);
  changeEmoji();
}

```

The ***recordAttempt*** function is only invoked when:

a. there's a match between the dominant emoji and the given one, and

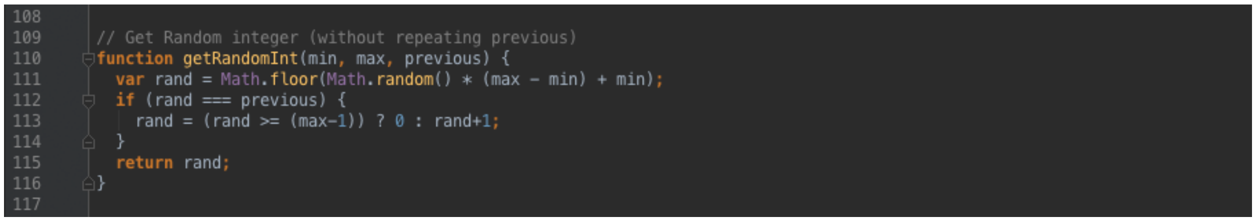
b. when the player runs out of time and fails to mimic a particular emoji (each round lasts 8 seconds.)

We update the ***score*** variable only if the attempt is successful, and then we proceed to give the player some feedback to let her know whether she correctly mimicked the given emoji or she ran out of time. Then, the ***setScore*** function is called to update the score board and a new round is started by calling the ***changeEmoji*** function.



```
// Change the emoji after player mimic correctly or ran out of time
function changeEmoji() {
  // Keeping track of each attempt/round
  attempts++;

  // Set a random emoji
  var rand = getRandomInt(0, emojis.length, currentIndex);
  currentIndex = rand;
  setTargetEmoji(emojis[rand]);

  // Adjust timer
  clearInterval(timer);
  // Player will fail at the end of the end of the interval,
  // if successful the timer will reset before failing
  timer = setInterval(function(){recordAttempt("fail")}, 8000);
}
```

Here, we update the ***attempts*** variable; since this function is invoked every time a new round begins, it seems to be a good place to do this. Then we get a random number and make sure it is not the same as the previous index, simply to avoid repeating the same emoji in consecutive rounds. The utility function ***getRandomInt*** was created for this purpose.

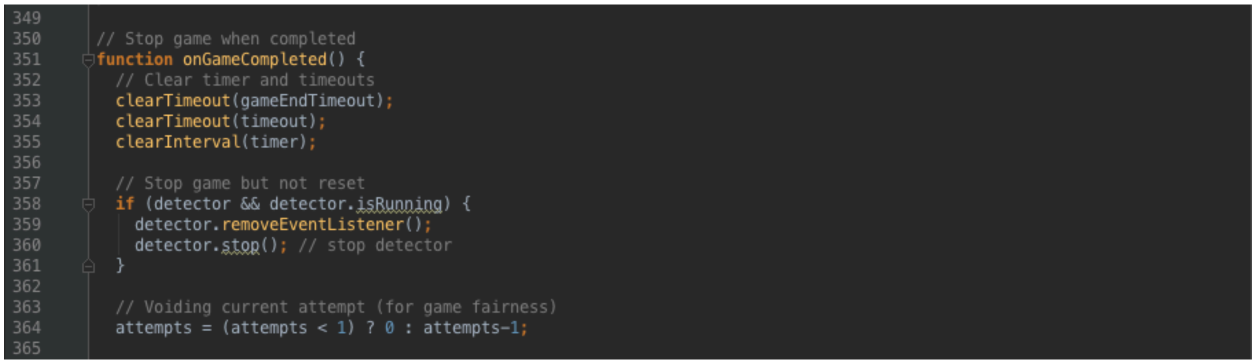


```
// Get Random integer (without repeating previous)
function getRandomInt(min, max, previous) {
  var rand = Math.floor(Math.random() * (max - min) + min);
  if (rand === previous) {
    rand = (rand >= (max-1)) ? 0 : rand+1;
  }
  return rand;
}
```

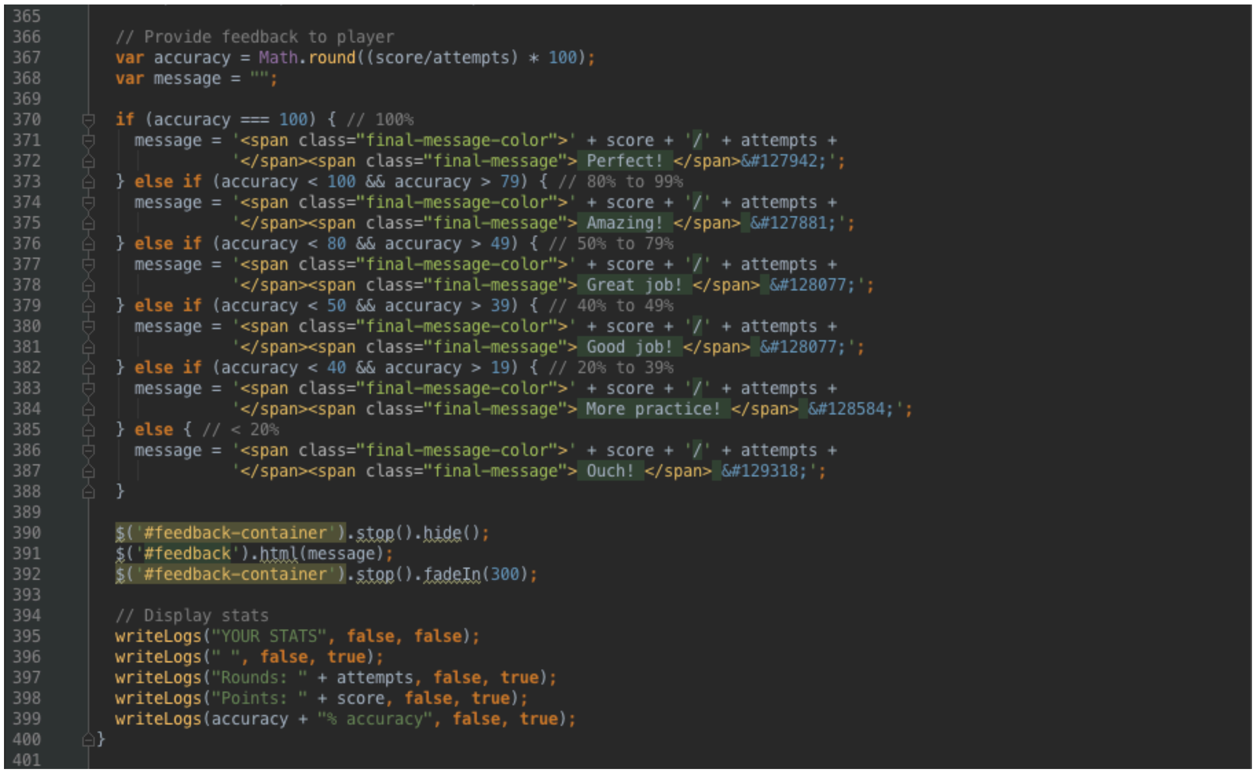
Then, the ***currentIndex*** variable that tracks the index of the current emoji is updated with the new random value and ***setTargetEmoji*** is called to update the view. We also reset the ***timer*** to start a new 8 second round.

Note that the callback function for the ***timer*** will record a failing attempt, which means the player will fail at the end of the interval. However, if an attempt is successful, the ***timer*** will reset before the callback function is invoked.

5. Ending the Game



```
// Stop game when completed
function onGameCompleted() {
  // Clear timer and timeouts
  clearTimeout(gameEndTimeout);
  clearTimeout(timeout);
  clearInterval(timer);

  // Stop game but not reset
  if (detector && detector.isRunning) {
    detector.removeEventListener();
    detector.stop(); // stop detector
  }

  // Voiding current attempt (for game fairness)
  attempts = (attempts < 1) ? 0 : attempts-1;
```



```

    // Provide feedback to player
    var accuracy = Math.round((score/attempts) * 100);
    var message = "";

    if (accuracy === 100) { // 100%
      message = '<span class="final-message-color">' + score + '/' + attempts +
                '</span><span class="final-message"> Perfect! </span>&#127942;';
    } else if (accuracy < 100 && accuracy > 79) { // 80% to 99%
      message = '<span class="final-message-color">' + score + '/' + attempts +
                '</span><span class="final-message"> Amazing! </span> &#127881;';
    } else if (accuracy < 80 && accuracy > 49) { // 50% to 79%
      message = '<span class="final-message-color">' + score + '/' + attempts +
                '</span><span class="final-message"> Great job! </span> &#128077;';
    } else if (accuracy < 50 && accuracy > 39) { // 40% to 49%
      message = '<span class="final-message-color">' + score + '/' + attempts +
                '</span><span class="final-message"> Good job! </span> &#128077;';
    } else if (accuracy < 40 && accuracy > 19) { // 20% to 39%
      message = '<span class="final-message-color">' + score + '/' + attempts +
                '</span><span class="final-message"> More practice! </span> &#128584;';
    } else { // < 20%
      message = '<span class="final-message-color">' + score + '/' + attempts +
                '</span><span class="final-message"> Ouch! </span> &#129318;';
    }

    $('#feedback-container').stop().hide();
    $('#feedback').html(message);
    $('#feedback-container').stop().fadeIn(300);

    // Display stats
    writeLogs("YOUR STATS", false, false);
    writeLogs(" ", false, true);
    writeLogs("Rounds: " + attempts, false, true);
    writeLogs("Points: " + score, false, true);
    writeLogs(accuracy + "% accuracy", false, true);
}

```

At the beginning of the game, when the ***startNewGame*** function is invoked, a 90 seconds timeout *(**gameEndTimeout**)* is set. At the end of the interval ***onGameCompleted*** is called, where we take all the necessary steps to end the game:

- Clearing timer and timeouts.
- Stopping the ***detector.***
- Adjusting the ***attempts*** count: We do this by subtracting 1 from the total number of attempts. This is necessary because the last attempt to mimic the emoji will never be successful, and that's unfair to the player. This ensures the player will have the chance to achieve a perfect score.
- Providing feedback to the player: In this step we update the view to give the player some information about their performance (stats) and a few words based their accuracy percentage.

6. Additional Notes

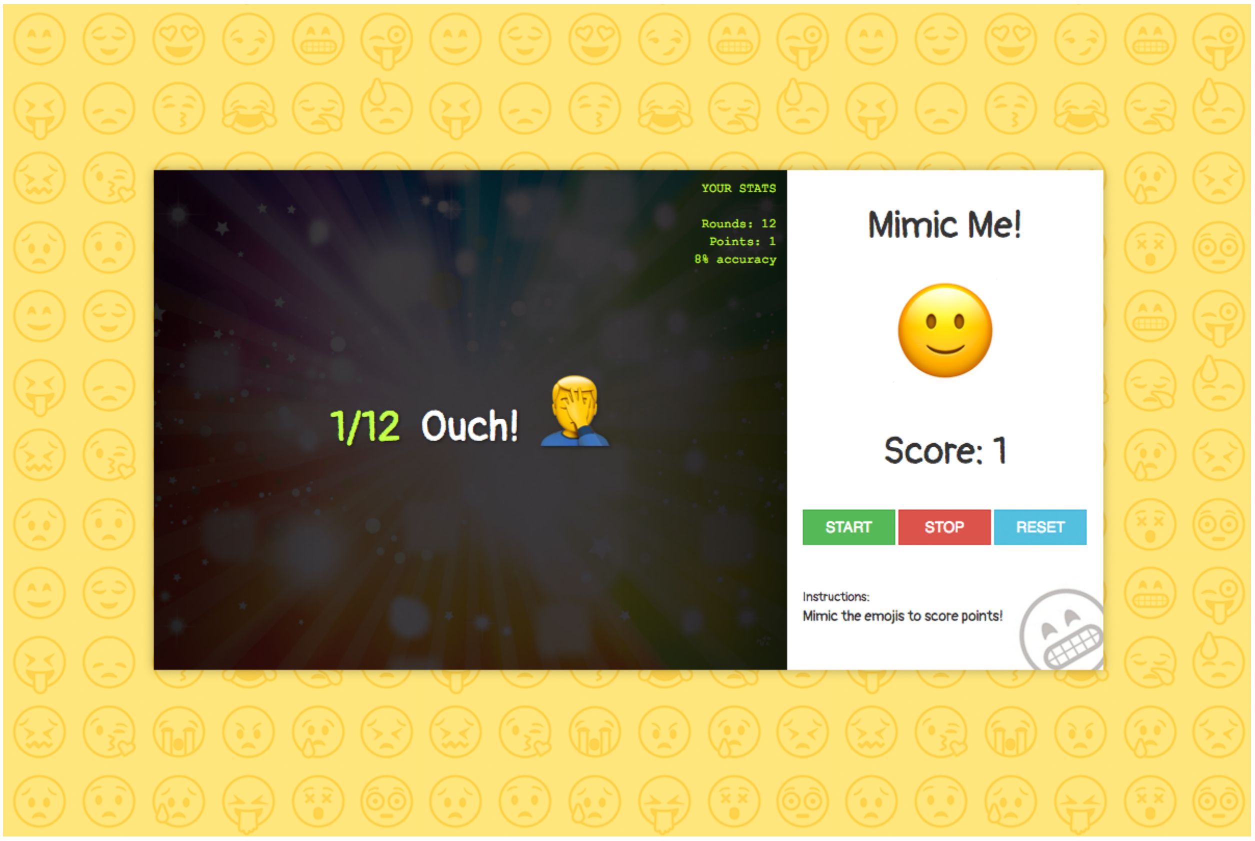
Some adjustments were made to the provided code in order to enhance the overall experience, such as:

- Emoji 128524 was removed from the game due to difficulties during gameplay. This particular emoji requires players to close their eyes, which makes it quite hard to play with.

- The ***writeResults*** function was refactored in order to make the game more user-friendly. The player does not need to see all the information available, so only a portion of that information is shown (the emotional information is displayed on the left side of the video feed.)

- A ***resetView*** function was added to take care of reseting all the visual elements.

- Feedback was included throughout the game to aid the player and make the game more friendly and fun.



Game is available online at ***https://middlechild.github.io/AIND-CV-Mimic-Emoji***