Tongue-computer Interface for the browser

This project is for Prof. Jacob's OOP class focusing on non-WIMP("windows, icons, menus, pointer") user interfaces, by Woodbury Shortridge (woodbury.shortridge@tufts.edu) and Jose Lemus (Jose.Lemus@tufts.edu). We worked in collaboration on both the concept, feasibility testing and development of the interface and game.

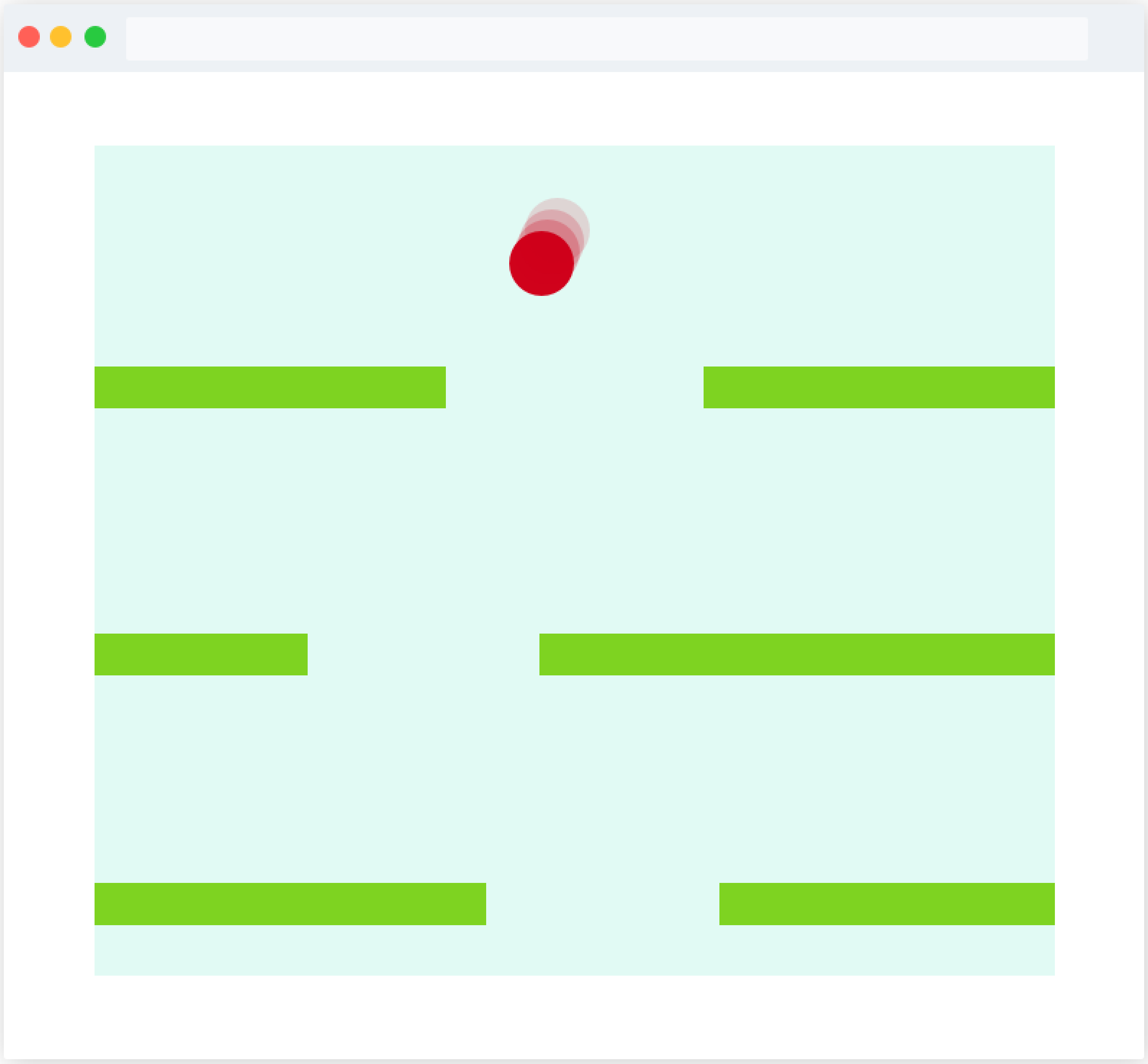
Premise

Tongue-computer interfaces are not new, there are many explorations in HCI research using various embedded sensor methods such as infrared, microphones, intraoral electrode arrays or even glossokinetic potential. And, it's not only fun and games: tongue-computer interfaces can enable severely disabled users such as those with upper body impairments. From browsing the web to flying a drone, the tongue's dynamic mobility offers wide potential for interaction design.

Our project

We propose a new, hardware-independent, approach to tongue-computer interaction. Using client-side JavaScript and a user's own device webcam we developed a method for input with tongue movements. Our proof-of-concept application is a simple JavaScript game that requires LEFT and RIGHT controls. The interface leverages the newly released JavaScript implementation of Tensorflow to classify the player's tongue movements without relying on sensors or even server-side computation.

Prototype:



Technologies

We developed a simple JavaScript clone of Flappy Bird, a “side-scroller” where the player controls a bird, attempting to fly between columns of green pipes without hitting them. Our clone takes a vertical viewpoint, meaning the avatar (a ball) progresses downwards requiring the player to move LEFT or RIGHT to avoid collisions. Our program design exploits the features of object-oriented programming, using JavaScript classes to encapsulate graphical objects (see Game, Ball and Wall classes), interface controls (see Interface, FaceFind, and Training classes) and data needed for callback and drawing routines.

Our interface uses the new Tensorflow.js library and a method called “transfer learning”. Specifically, we implement the k-nearest neighbors algorithm (KNN) on top of MobileNet. MobileNet is a small, low-latency, and low-power convolutional neural network trained on imagenet data. It is is designed to run efficiently in the browser. Our program grabs the second to last layer in the neural network and feeds it into a KNN classifier. This allows us to rapidly train our own tongue-based classes. The classes are trained on user generated images it captures of people sticking their tongue to the left and right during the training phase. We implemented Tracking.js to find and create a bounding box around the user’s mouth. These coordinates are used to crop the video stream into focused images fed into our classifier.

Feasibility tests

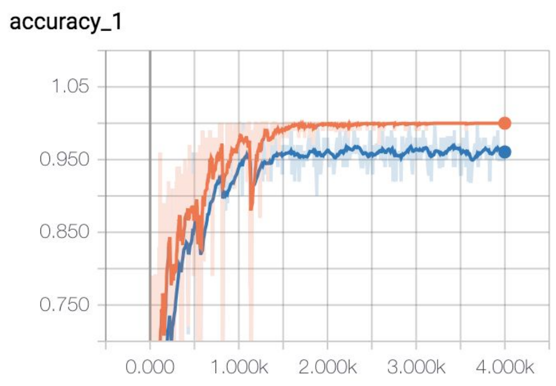
Our greatest technical risk was using Tensorflow.js to build a tongue classifier. Initially it was unknown if we could make reasonable inferences with a simple webcam. To test the feasibility, we created a test JavaScript app. Using Tensorflow.js, the app loads the MobileNet model and a KNN classifier (see `interface.js`).

Next, we made a simple OpenCV program in Python to collect training images (see `training/camera.py`). This program listens to keypresses and captures images of right, left or no tongue out. The training images are fed into our app by json data and added to the classifier using KNN.

To test feasibility, the app creates a canvas to which the webcam is streamed. Then, with the new model, we classify the each canvas frame. The prediction values are printed to html using the innerText method. We found our initial methods worked ok, but not well enough to actively play a game.

Next, we re-trained our own model using Tensorflow’s Python library. Our script ran 4,000 training steps. Each step grabbed 10 images at random from our LEFT RIGHT images (this time using OpenCV and haarcascades to crop images about the mouth) found their bottlenecks from the cache, and fed them into the final layer to get predictions. Then, we converted this frozen model into a tensorflow.js web model. (see `web_model`).

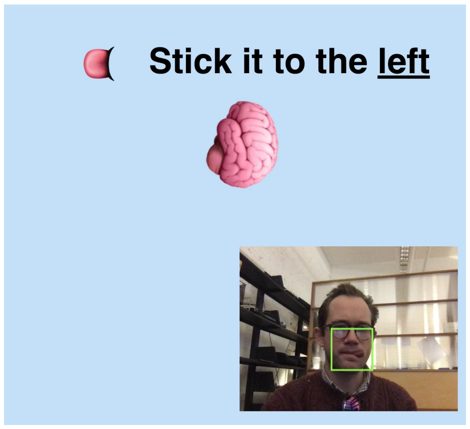
The new model improved our results, but we were still not satisfied with controlling the game. Our final and most reliable approach was to incorporates a training session with the user to capture user-generated images of tongue RIGHT and LEFT and feed them into our classifier. We implemented Tracking.js to create a bounding box and crop the webcam stream around the user's mouth in real time.



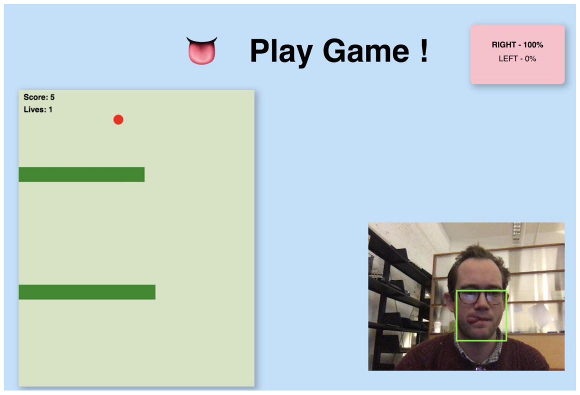
Design

Our design uses the graphical layout as a way to communicate information to the user. On the right side of the browser there is prediction values to give observability into the algorithm's inferences, grouped by a background color. Also on the right is the video stream featuring a bounding box. Instructions are placed in the middle of the screen with large letters. We chose to design the program to run with no keyboard or pointer interaction, tongue only. Therefore, for the training procedure we leverage animations and sound cues to help guide and give contextual awareness to the user flow. The game appears on the left side of the browser after training. We chose to use this progressive disclosure to keep visual attention on the relevant elements while training. The game itself features colors and shapes to denote game elements and displays the score and lives on the left upper corner of this canvas.

Training:



Game-play:



Demo

A web demo is available: https://woodburyshortridge.github.io/tongue-comp/index.html

The repo is also available: https://woodburyshortridge.github.io/tongue-comp

Serve from this root directory and visit your localhost.

I prefer to use Node.js http server, install with `npm install http-server -g` and run `http-server` at directory root.

MacOS

MacOS comes with PHP, try `PHP -S localhost:8000`

Windows

If you have Python, try `python -m SimpleHTTPServer 8000`

Develop

Install node if you don't have it: https://nodejs.org/en/download/

Install dependencies: `npm install`

Run the dev server: `npm run start`

Build production: `npm run build`

Train

Dependencies

Python 3: https://www.python.org/downloads/

OpenCV: `pip install opencv-python`

Capture photos

To take training photos, move to the training directory: `cd training`

Then, launch the Python program: `python camera.py`

With this program running, hit the 'r' key to take right tongue photos, the 'l' key to take left tongue photos, the 'n' key to take no tongue photos, and the 'q' key to quit the program.

Images will be saved to their respective folders.

To add new images to the json used to train the app, run `python data.py`