Capstone 2 Project Final Report
Department of Electrical and Computer Engineering, Northeastern University

Project Title:
Playing Pokemon With the Brain

Team Members:
Sultan Alzaabi, Peter Zeqo, Michael Zeleznik, Chris Zarba, and Matthew Heim

Advisor:
John Kimani

Table of Contents

Abstract

The Pokemon BCI team has designed and built a system for brain-controlled inputs to a video game that, using sufficiently accurate EEG data, can allow a user to control the inputs for the game Pokemon Red with their mind and eyes. This technology is useful for people with physical disabilities, like ALS, who cannot move muscles voluntarily except for the eyes. The game is displayed in the center of the monitor and is surrounded by visual cues representing each of the possible inputs to the game. Each visual cue is a checkerboard pattern flashing at a different frequency. The visual cortex emulates the frequency of the flashing pattern it observes at a given moment. This phenomenon is called visually evoked potential (VEP) that allows the system to classify visual cues as game inputs based on the frequency of the VEP signal. While we were not able to achieve high accuracy with our EEG headset, our simulated EEG data with noise was able to play the game with great accuracy.

1 - Introduction

1.1 Problem Formation

In an era where the pervasiveness of gaming in popular culture is constantly increasing, the existing technology for disabled people to play is shockingly sparse. Because video games have a virtual environment rather than sports that require complex body functions, they can be played by using the mind itself as a controller. Video games do not necessarily require the use of a standard controller that depends on the functioning of the user's hands. A popular device for "locked-in" gamers, like people who suffer from ALS, is the breath controlled device, shown in Figure 1. Users perform certain inputs by breathing, allowing them to make real-time inputs to a game. We considered the possibility of a device that isn't overly invasive, uncomfortable, and only requires the user to sit and gaze at the screen. Our team aims to help make video games accessible to people who otherwise don't have an easy way to express or create movement themselves.

Currently, BCIs are not used for gaming because they are clunky, inaccurate, and not fun to use for gamers with disabilities. Part of the issue is that current technology is not accurate or fast enough to easily create devices that fill this need. Another part of the issue is that existing devices or interfaces that could fill this need are not enjoyable to use, and so would not be marketable. Our team focused on creating a preliminary version of a product that would be enjoyable to use and would pave the way for advancements towards a marketable BCI gaming device.

1.2 Problem Solution

Our objective with this system is to provide the software to allow people to play any video game just by looking at the screen. We chose Pokemon as an ideal game for this system for its slower-paced gameplay and popularity. Pokemon battles are turn based and there is never a need for quick reaction time like in a first-person shooter game. The user wears a headset that reads EEG signals and sends them to our software. The signals are processed and filtered, and then a machine learning algorithm finds their correlations to our possible target values. Then our input classification model chooses the most appropriate target for the data and sends an input to the game state. The game state is displayed on the monitor, which is surrounded by the visual cues which the user looks at to select their next action. There are two types of technology that allows this project to work. The one we made work is steady-state visually evoked potential, or SSVEP. SSVEP means that there is a constant flickering frequency. The other is c-VEP (code-VEP) which means there is a pseudorandom binary sequence representing the on and off states. C-VEP has lower latency but the straightforwardness of SSVEP made it easier to work with.In the research paper “Dynamic time window mechanism for time synchronous VEP-based BCI” by Gembler, Felix, et al., both systems were tested for converting brain signals into letters for a typing system. Final results showed that Code-VEP “system (mean word spelling ITR 92.7 bpm) outperformed the SSVEP system (ITR 75.1 bpm)” [1]. This higher rate of data transfer might allow for less latency between command selection from the user and command identification within the BCI system

Figure 1: A child using a breath-controlled gaming device

1.3 New Knowledge, Technology, or Technique Learned and Applied

We learned a lot through trial and error during this project. In every aspect of the design, we tried multiple options and observed the outcomes. For example, we moved towards using the Psychopy library in Python for reliability in our data acquisition. When we did this, we reused a lot of our existing Python code. The main technology learned in this project was working with an EEG headset and analyzing both offline and online data. We learned about using Jupyter Notebooks for analyzing the offline data. Also, we learned about building an experiment in PsychoPy and altering the code it generated to fit our needs. A statistical technique we learned about was canonical correlation analysis, or CCA. Since the documentation on this technique was lacking, we had to figure out how to use it ourselves. Generally, we learned more about applying Python, Numpy, Pandas, and other Python libraries to our problem.

1.4 Related Works

The technology at the core of this project has been developed in some form before. Existing setups have been created to "detect a level of attention and... relaxation"[3] in a player's brain waves as inputs for a video game. This is achieved through electroencephalogram (EEG) technology, a similar approach to our chosen hardware. Our deliverable differs, however, in that we plan to make the user experience tailored to Pokemon, whereas the previous EEG products have prioritized breadth over specialization. As such, in our research, we were unable to find a product tailored to an existing game, let alone a game as large as Pokemon.

It is worth noting that Pokemon is a property of video game giant Nintendo, a company that has pursued using a BCI setup to affect gameplay. As of 2020, patent US8704879B1 gives Nintendo the rights to eye-tracking technology that immerses the player in an illusory 3D environment [6]. The technology, seen in Figure 2 below, is non-invasive and can be considered a unique method of user input, as our project aspires to be. Ultimately, our project merely affects controls, while this IP affects the gaming experience altogether. Moreover, it is unspecified whether this technology is designed for a particular game as the Capstone project is.

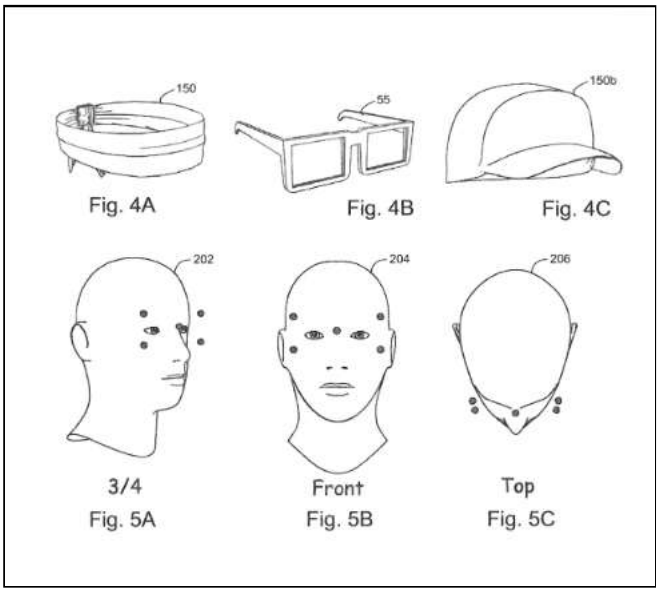


Figure 2: Nintendo's proposed eye-tracking product aimed at player immersion.

Another eye tracker that has similar objectives to our system is the Tobii Eye Tracker that uses an adaptive camera and an illuminator. This device is primarily used for gaming. However, our device is aimed at giving physically disabled people functionality and entertainment, while Tobii is aimed at improving competitive gameplay and streaming experience.

A study in 2018 by Hooman Nezamfar, et al. compared Eye Tracking against Code-VEP systems for solving a simple maze. On average, subjects reported that they preferred the Code-VEP system, found it easier to learn / interact with, and thought it responded faster to their commands [5].

The UI of our application contains the game screen in the middle and checkerboard patterns on the sides and corners of the screen. This is similar to an existing UI of an SSVEP-based spelling application shown in Figure 3. There has been extensive research in BCI spelling applications and it is close to our product idea. We rely on the same technology of a speller, but our application of the technology is different.

Figure 3: Example of UI used for spelling application.

2 - Design

2.1 Overview and Objectives

The system we have created acts as an interface between the Emotiv EPOC and the video game. The block diagram illustrating the flow of data is shown below in Figure 4. The EEG headset sends its data in offline mode to a csv file, to be analyzed. In online mode, the EEG sends data of a certain number of seconds known as the stimulus length: usually 1, 2, or 3 seconds, to our processing algorithms directly. The data for one stimulus length is filtered and a prediction is made on which target the user is gazing at. The selected command is then sent to the game, and the process continues. We used 4 channels in the occipital and parietal regions: P7, O1, O2, and P8. The location of these electrodes can be seen in Figure 5.

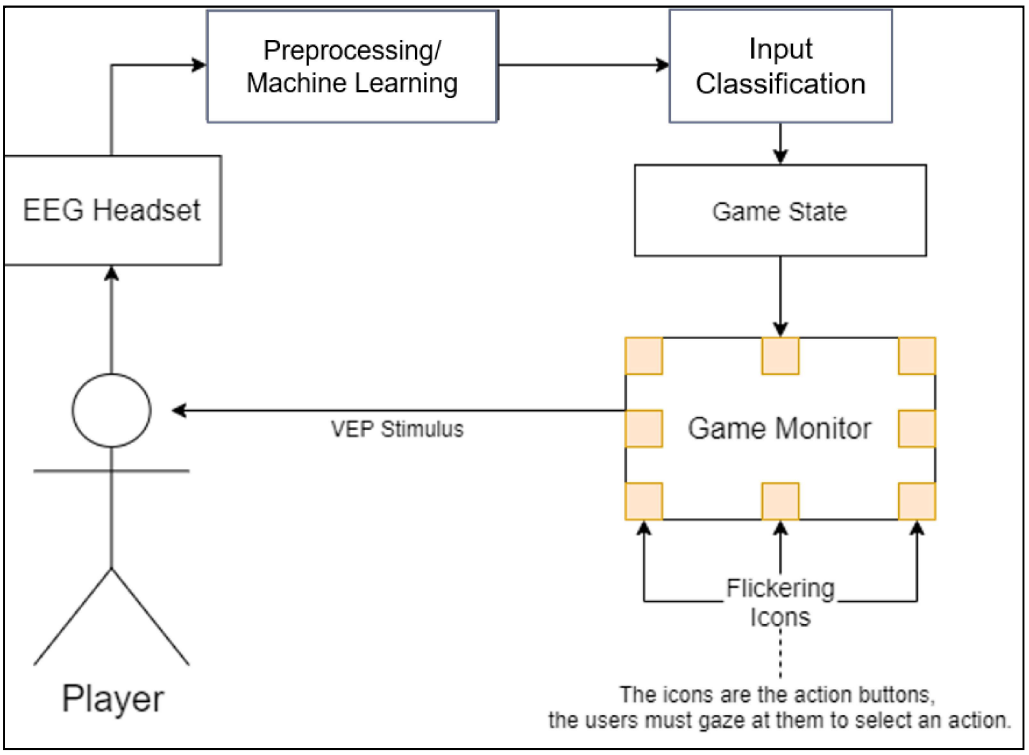


Figure 4: Block Diagram of BCI System

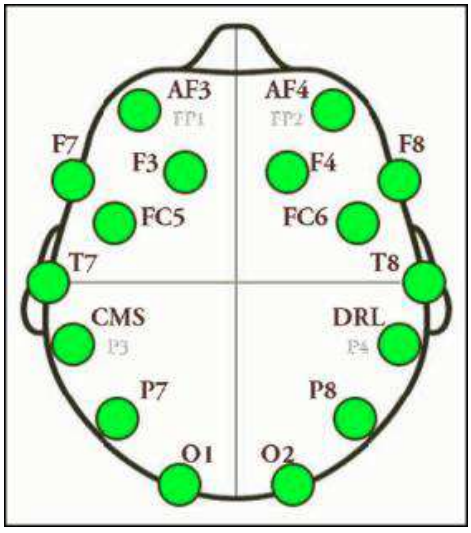


Figure 5: Emotiv EPOC electrode map

Our requirements for this project were to have as high accuracy as possible in making predictions with a 3 second or less stimulus length. Having a lower accuracy would result in more unintended commands being sent which makes the game less playable. An accuracy above 90% was desired. Other requirements for the interface itself were that it should be clear to use, and relatively easy and comfortable for the user.

2.2 Offline and Online Programs

In order to train models we also have to implement a system for collecting and exporting data to use as training and validation sets for model creation. This meant that we would need an interface that could both collect data and save it (offline), or collect data and periodically feed it to a pretrained model to simulate "live mode" for actually playing the game (online). The process diagram that we would model our code after shown below for each of the programs.

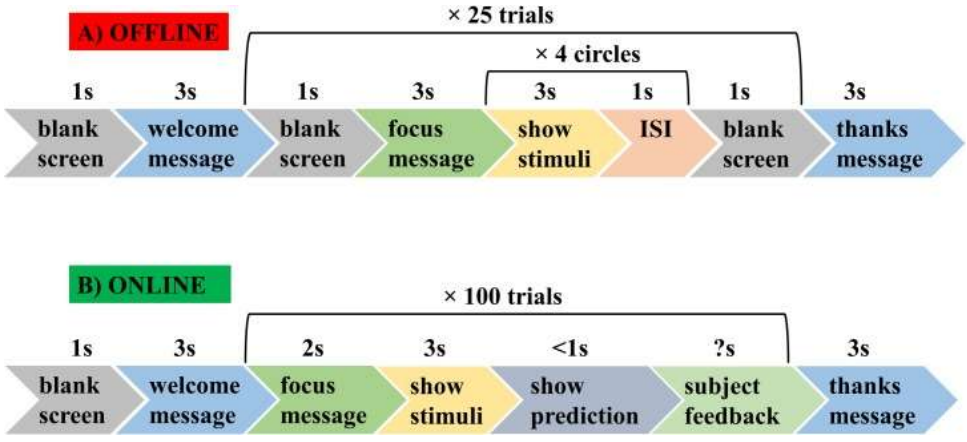


Figure 6: Offline and Online Process Diagram

2.3 Analysis and Synthesis

One of the approaches we took to identifying commands was to use neural networks. Although we have access to 14 sensors for data, we determined that only the 2 occipital and 2 parietal sensors would be needed as they measure visual data most closely. We used Tensorflow in order to implement both of the neural networks architectures we tested to classify data.

The first neural network we tried is a Long-Short-Term memory network that has recurrent gates. This allows the network to recognize and classify sequences of data. This means that it can input the entire length of 1-3 seconds of EEG data that we recorded when the user was looking at one of the flicker icons.

Our other attempted method was a standard Dense Feed Forward network in which we used power band data so that we work with input arrays integers (instead of data sequences) which are compatible with this network architecture. For this we focused on a split of the low and high power band data in both the alpha and beta wave ranges since this is where we would expect to see the VEPs.

Another approach for accurate predictions was canonical correlation analysis, or CCA. This was initially used to make predictions of which target the user was gazing at. We later modified its purpose to generate features that the k-nearest neighbors algorithm, or KNN, would classify. CCA finds linear relationships in multivariate data and is useful in decomposing and analyzing a multi-channel EEG signal.

We generated reference signals for each target and its corresponding frequency. These reference signals served as sources of truth to compare a measured EEG signal against. One reference signal was composed of sine and cosine waves at the target frequency and its second harmonic. Examples of reference signals are shown in Figure 7. In this analysis of actual EEG data, we used 4 targets for simplicity.

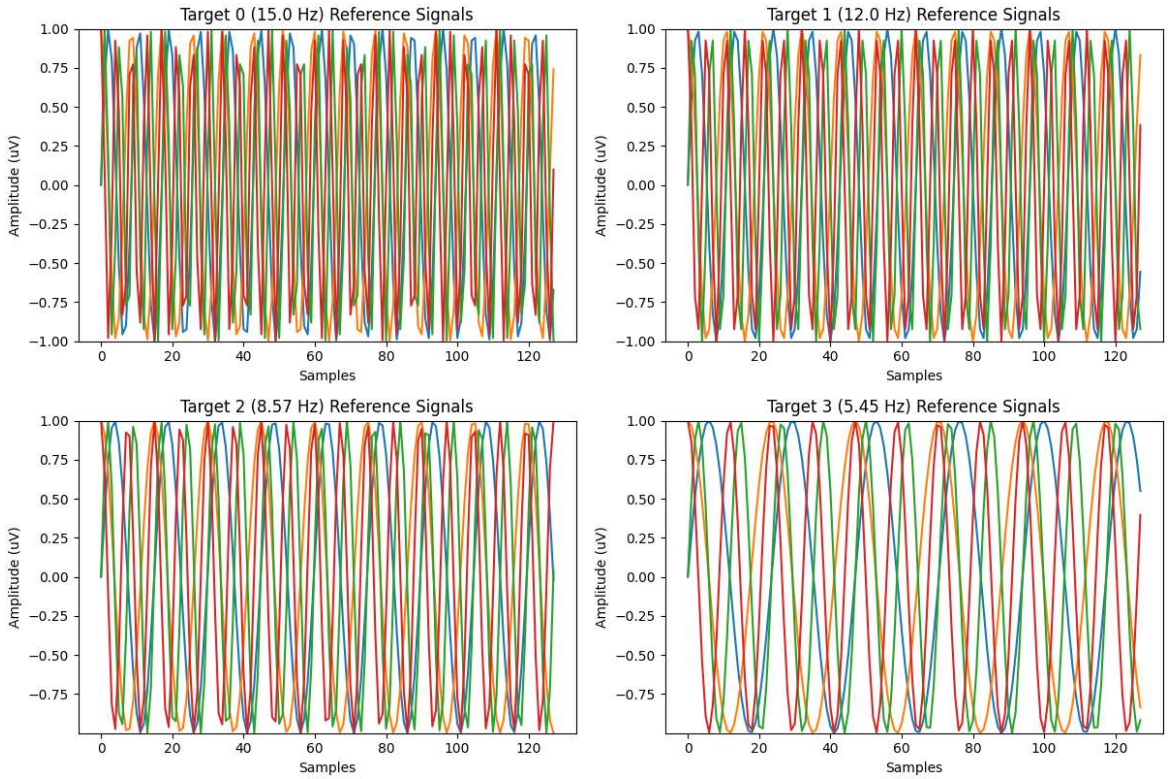


Figure 7: Reference signals for 4 targets over a 1 second window.

One second samples of EEG data can be seen in Figure 8. After using a band-pass filter, the filtered signals look like the ones shown in Figure 9.

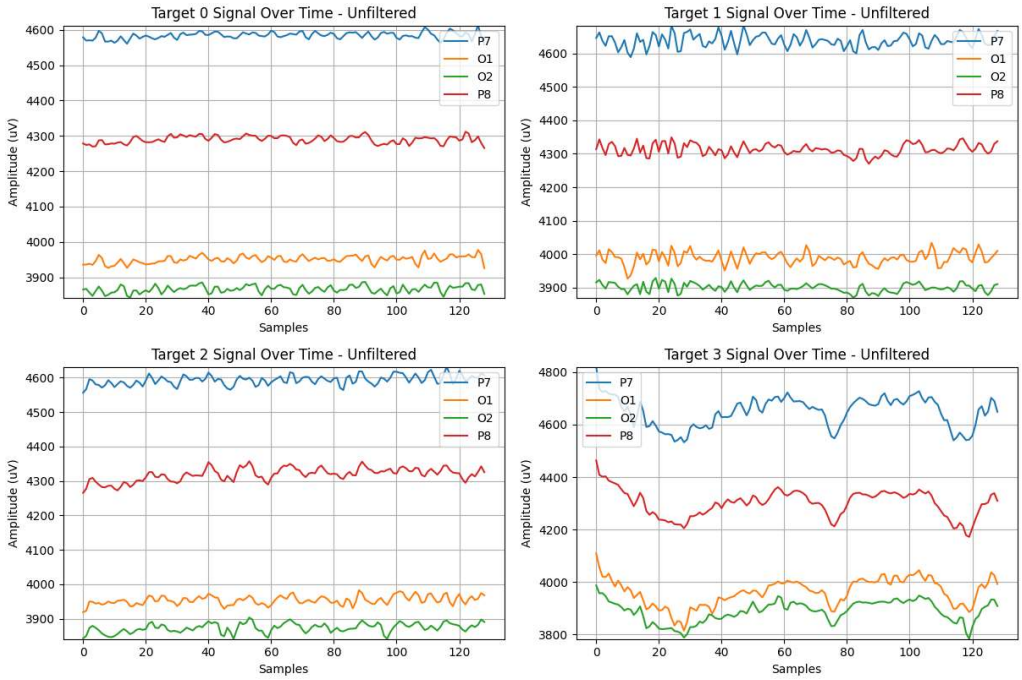


Figure 8: Recorded, unfiltered EEG data over a 1 second window.

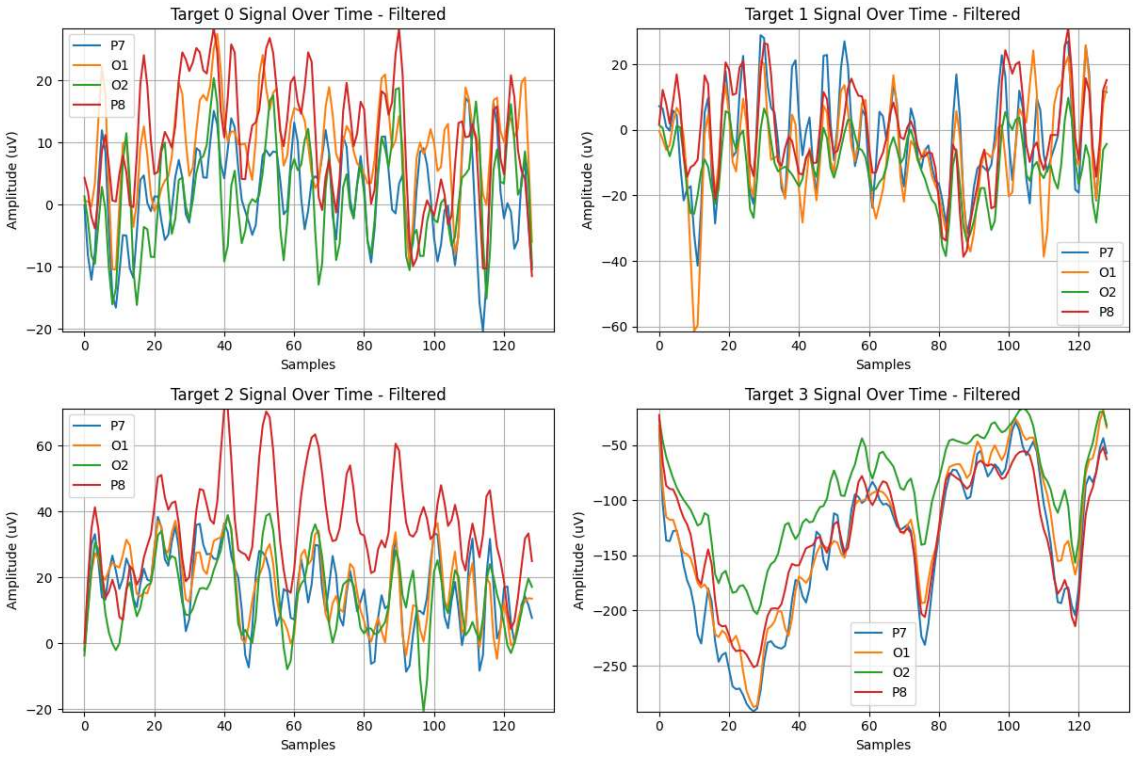


Figure 9: Recorded, filtered EEG data over a 1 second window.

3 - Software Implementation and Results

3.1 Data Collection

Training Neural networks would require us to have subjects go through an “offline” process in which they stare at each target in the game space for a set amount of second, a total of 25 times for each target. During this time, our data collection process should be constantly collecting the EEG and Power Band data from the Emotiv. This data should be accumulated and then exported to CSV files at the end of the process so that we can later read them in and treat them as training data in model creation.

This data collection process was implemented by creating two different versions of our software, one that runs the offline testing process and exports data, and one that collects data and uses the pre-trained models in live time to interact with the game of choice.

Both versions of the code share commonalities in displaying the targets around a central focus image (instructions during testing or the game we’re playing). On each tick or frame that the process is running, we use the Cortex API provided by Emotiv to request the EEG and Power Band data from the headset, and store the values in a Numpy Array associated with the target the user is looking at. Each of our targets on the screen has their own unique array to keep track of which data corresponds to the user looking at which target. The testing process we followed is the one presented in the Design section.

3.2 PyArcade Version

Our initial approach was to use the python library, PyArcade, to create a UI that draws the stimulus and game at the same time. In order to run the GameBoy ROMs we implemented the PyBoy library, and worked with the Dev’s to take image information from the emulator and draw it into our PyArcade window.

To create visual stimuli, we drew checkerboard icons around the game in the corners of our window that the user could look at to pick a command. The checkerboards were chosen based on proof of them working in past studies, and would flicker by having their colors inverted on each flip.

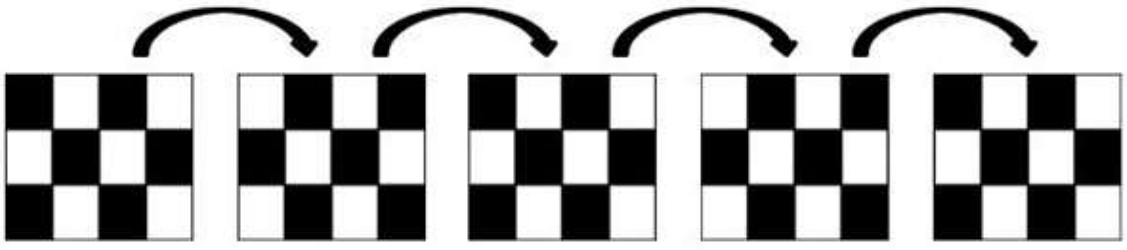


Figure 10: Checkerboard Flickering Pattern

.

We realized that we would need to create 2 Arcade Window subclasses, a data collection window to get training data for our models (offline program), and a live mode window (online program) where we can test the models we create and play the game. The data collection window mimicked the live mode without the game ROM loaded and running in the center, instead being replaced by text instructing the user through the offline process.

Figure 11: PyArcade Online Program Window

Playing the game in Live mode was handled through the use of pretrained models that were loaded into a Command Handler object. This object would also instantiate a PyBoy Controller, which would hold a reference to the PyBoy object which runs the game, and has a mapping of commands to button events (ex. Up, down, A) that the PyBoy class can take in and use to simulate button presses in the game. The Command handler is given EEG or Power Band data periodically, filters it, passes it to the model to predict which square the user is looking at, and passes the prediction to the Controller. The controller converts the prediction to PyBoy button events and passes them to the PyBoy object which then simulates the button presses and plays the game.

Unfortunately, there were unforeseen issues with this library. The timinging between frames was inconsistent. The expected time for each frame would be (1/FramesPerSecond), as each frame should take an equal amount of time, and we should process a set amount of Frames each second. From our experimentation we observed some frames taking up to 2 or 3 times as long as they should. Since PyArcade couldn't deliver consistent times between each frame we were unable to collect data in a consistent manner.

Without good clean data we can't capture the VEPs in our brains caused by the flickering icons, and this bad data means that no classification models would be able to make accurate predictions. This results in the game being nearly unplayable.

While this version of the program was unsuccessful in the end, it did lead to code being written for handling PyBoy to run the game, interacting with the game, and creating, training, and loading in models for use on data collected from the Emotiv. All of this would go on to be re-used in the final version of the program written in PsychoPy.

3.3 PsychoPy Version

To handle the inconsistencies of PyArcade, we had to switch to a different library. We were recommended PsychoPy by Prof. Erdogmus. Our main concern here was the time it would take to recreate the whole project in PsychoPy. However, PsychoPy was designed to be very easy to use and specifically to run a wide range of experiments in the behavioral sciences. Therefore, we utilized its Builder GUI and Coder IDE to quickly put back our project together, shown in figures 12 and 13.

We made some major changes to our project as we transitioned to using PsychoPy in order to step towards our goal of a proof-of-concept. First, we changed the checkerboards to simply be flickering white icons. Second, the look of the UI was a little bit different. We also went down from 8 commands to 4 commands only; however, when tested with simulated EEG data, the system can also work with 8 commands.

The Builder GUI made it very easy to build the new project by utilizing routines and loops. Then, we compiled the Builder project into a python script. From there, we simply added our old code and modified some smaller features, and then the project was ready for testing!

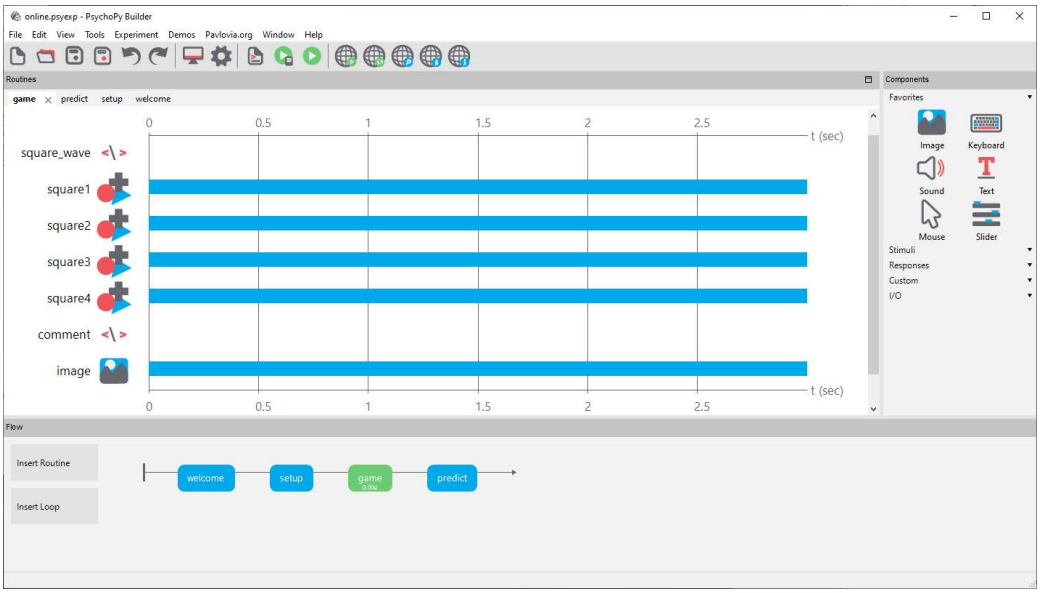


Figure 12: PsychoPy's builder GUI.

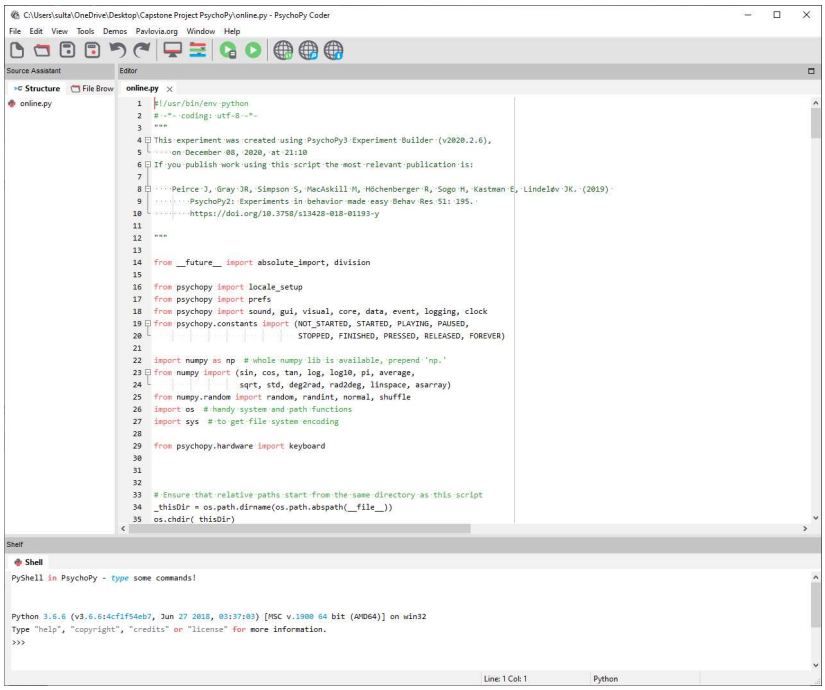


Figure 13: PsychoPy's coder IDE.

With that, the inconsistencies caused by PyArcade were fixed by PsychoPy. The following figures are screenshots of some parts from our project:

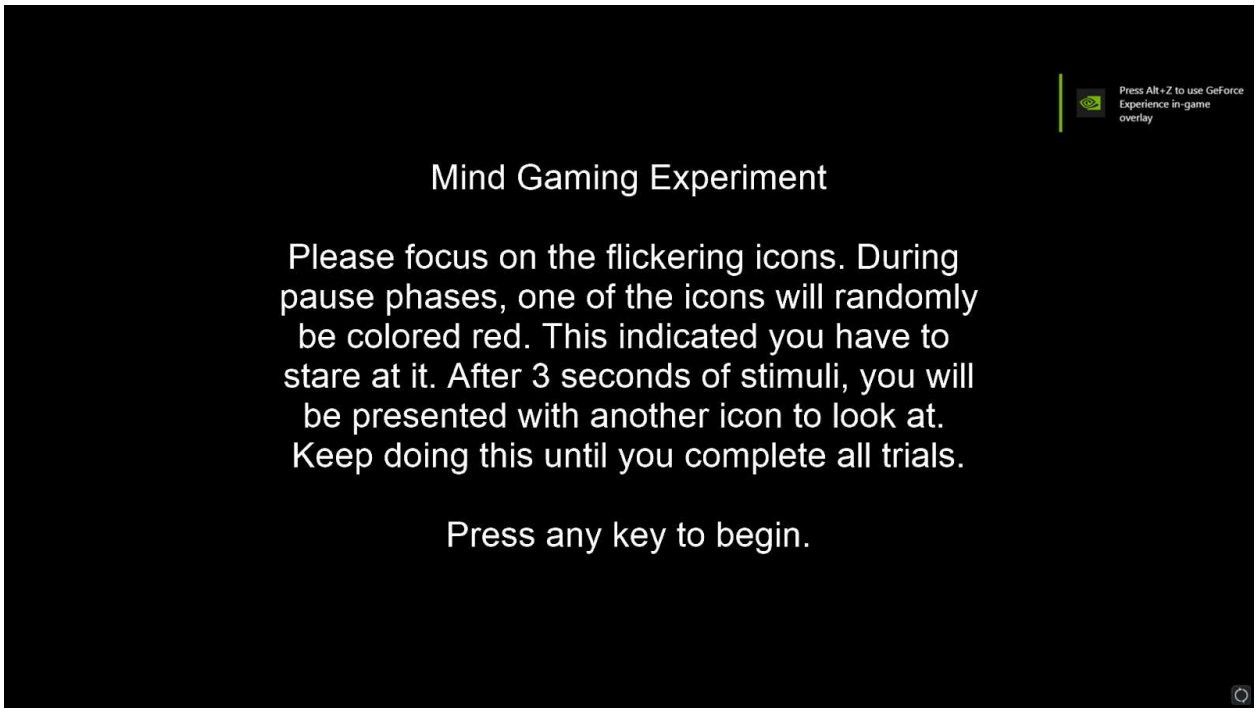


Figure 14: Instruction scene from our software.

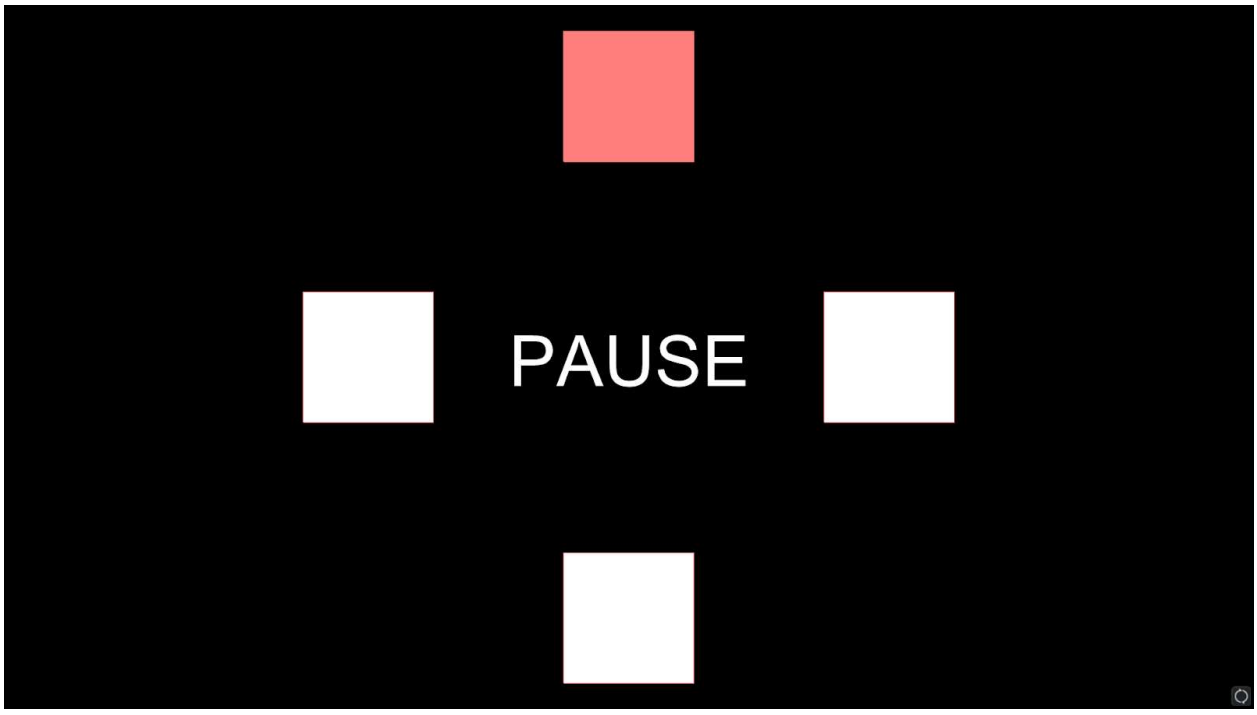


Figure 15: Focus message scene, it tells the user which icon to look at next.

3.4 Data Filtering

With the signals from the headset acquired, the excess noise necessitated that we perform pre-procedures on our data before passing it into our classification models. First, we considered several methods of signal filtration. We discovered the signals the human brain sends during visual stimuli are in the neighborhood of 7-25 Hz [2], so we set this range as our area of focus. Signals of these frequencies were to be amplified for further analysis.

In order to isolate the essential frequencies, we implemented a Python module for a Butterworth bandpass filter wherein we set our cutoff frequencies at our desired boundaries. We also found success with a Finite Impulse Response (FIR) filter module that would act similarly with minimal feedback. The resulting filtered signals underwent Fast Fourier Transforms (FFTs) as part of testing. The FFTs would produce the frequency response to our filtered signals and allow us to determine if unwanted frequencies remained in our data.

Beyond standard hard-coded filtering, we also considered an AI-based filtering solution. Principal Component Analysis (PCA) is typically used as an unsupervised method of dimensional reduction in data processing [4], but it can also be used as a means of noise reduction, as in our use case. For an EEG signal composed of multiple channels (four in our case), a PCA filter can analyze the "principal components" of the data and recreate the original signal as if every component were to perform as well as the principal component(s). Figure 16 illustrates the effectiveness of PCA in reducing noise for the O1 channel.

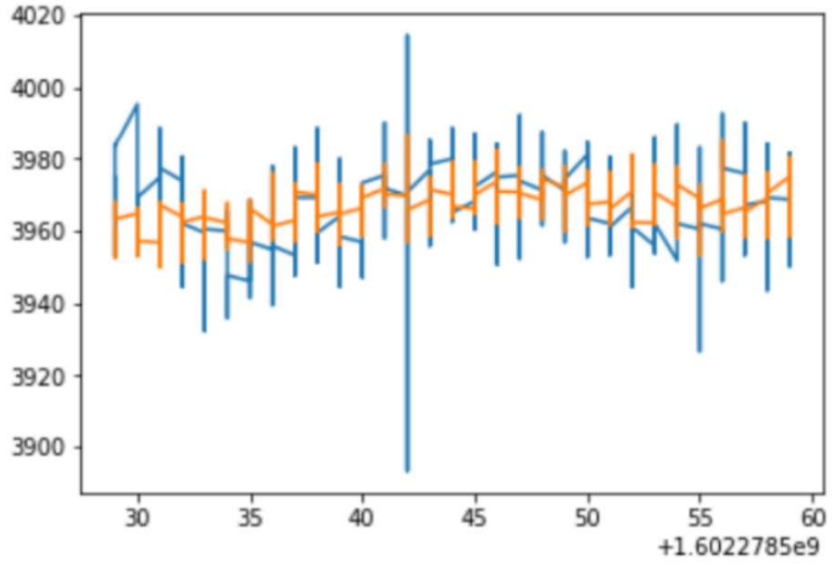


Figure 16: An O1 channel signal before (blue) and after (orange) PCA filtering

Each method of filtering ultimately was included in a Python filter class designed to simply receive EEG data as an input and produce a clean signal as an output via the selected method of filtration. For all filtration methods that required a Nyquist frequency, we programmed a value of half our sampling rate. Thus, a default sampling rate of 128 Hz would result in a Nyquist frequency of 64 Hz.

3.5 Machine Learning

Both Dense Feed Forward Networks (DFFN) and Long-Short-Term memory (LSTM) models were created through Tensorflow to try and accurately predict the checkerboard that a user was looking at. For the showed nearly random guessing results most likely due to data issues not network architecture.

Our DFFN was created with 2 hidden layers, each with a width of 100 nodes. Given power band data, an array of floats representing the intensity in each brain wave frequency band as shown in Figure 17, the models achieved nearly random accuracy.

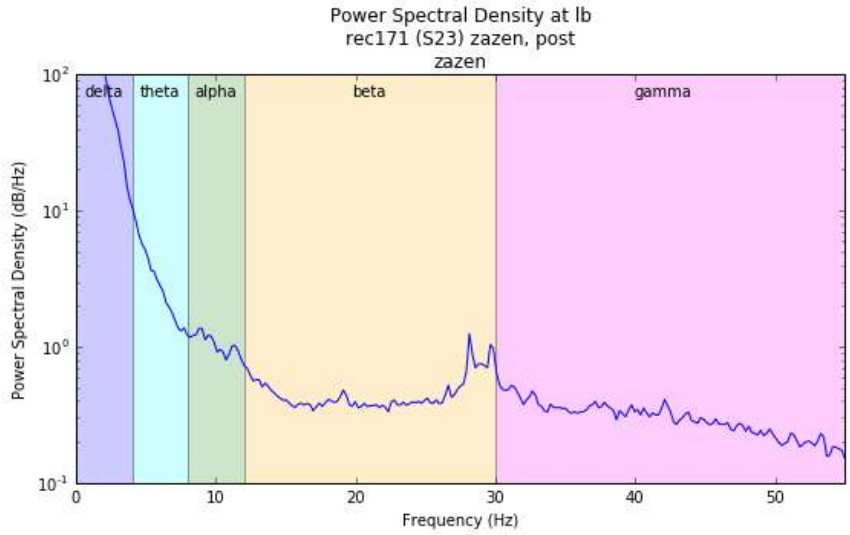


Figure 17: Example of Power Spectrum Data for EEG Data

This is likely due to the data issues we experienced due to the Emotiv headset. The lack of detected VEP's leaves both models with little to nothing to detect, so random guessing was the expected outcome knowing this now.

The LSTM model is similar to a DFFN, but has a memory component to the model so it can process a sequence of data. This means that each node has a memory of previous data seen, and as a result this model can process sequences of data instead of an int/float array for a set of features. Due to this, the model is capable of training on and predicting the user's intended target from the entire EEG data sequence collected during the "gazing period".

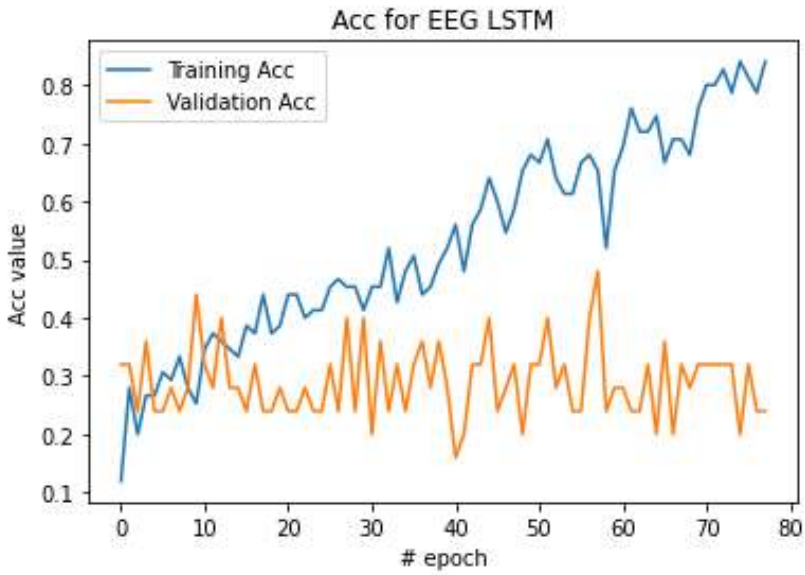


Figure 18: LSTM Model Training and Validation Accuracy on EEG Data

From Figure 18 we can show that the model is guessing nearly randomly on the validation (unseen) dataset, while learning the training data quite well. This is likely due to overfitting in which the model is remembering the examples it's seen too well due to the pool of training data being quite small. Once again, due to the data issues this model was also searching for meaning in data that didn't have any, so it's performance was nearly random and accurate command prediction was not achieved.

Multiple techniques were used to try to enhance the performance of both models. We performed Grid Search, the process of testing all subsets of a given possible set of feature values, to try to determine the optimal architecture of our models. These parameters included features such as the width in nodes of our layers, the number of hidden layers, the optimizers used, and the learning rates for our models. Grid Search tests each subset and returns the set of parameters that perform the best, but no model was capable of reliably predicting the commands on the validation dataset at a rate of greater than 35% (on 4 targets).

Aside from Grid Search, we also applied numerous data processing techniques such as PCA, Wavelet Decomposition, and Butter Bandpass Filters. Once more, none of this data preprocessing was able to create meaningful data for our models to achieve high accuracy due to the initial data being poor.

3.6 Canonical Correlation Analysis

CCA finds two canonical correlations between the measured stimulus and each reference. So, for 4 targets there are 8 canonical correlations. These canonical correlations are used as features for KNN (K-nearest neighbors). KNN is trained and tested on offline data with the correct target as a label. CCA can also be used to classify by choosing the target with the maximum correlation score. The functions that generate reference signals and perform CCA are

shown in Figure 18. The data that is sent in is all channels of EEG data for some predefined stimulus length as a number of seconds.

```
def getReferenceSignals(frequencies, num_seconds):
    ref_signals = []
    for freq in frequencies:
        ref_signals.append(generateRefSignal(128 * num_seconds, freq))
    return np.asarray(ref_signals)

def generateRefSignal(samples, freq):
    signals = []
    t = np.arange(0, (samples/128.0), step=1.0/128.0)
    signals.append(np.sin(np.pi*2*freq*t))
    signals.append(np.cos(np.pi*2*freq*t))
    signals.append(np.sin(np.pi*4*freq*t))
    signals.append(np.cos(np.pi*4*freq*t))

    return np.asarray(signals)

def findCorr(data, target):
    result = np.zeros([(target.shape)[0], 2]) # num_targets x number of canonical corrs
    cca = CCA(n_components=2)
    for i in range(0, (target.shape)[0]):
        cca.fit(data, np.squeeze(target[i,:,:]).T)
        a, b = cca.transform(data, np.squeeze(target[i,:,:]).T)
        # a and b shape = 128 x (n_components)
        corr_arr0 = np.corrcoef(a[:,0], b[:,0])
        corr_arr1 = np.corrcoef(a[:,1], b[:,1])
        corr = corr_arr0[0, 1]
        corr2 = corr_arr1[0, 1]
        result[i][0] = corr
        result[i][1] = corr2
    return result

def getChunkFeatures(data, ref_signals):
    corrs = findCorr(data, ref_signals)
    return corrs.flatten(order='F')
```

Figure 18: Code for reference signals and CCA

Using CCA to generate features for KNN, we only reached a maximum of 50% accuracy. Table 1 shows the accuracy of KNN on one of our members's recorded EEG data. The signals did not have artifacts or large disturbances, yet showed poor accuracy. We also expected accuracy to increase from one to three seconds for a stimulus length, which did not happen.

Stimulus Length	Accuracy
1 second	*19.6 %*
2 seconds	*50 %*
3 seconds	*42.1 %*

Table 1: KNN Accuracy with different stimuli length.

Looking at the power spectrum of our recorded EEG data shows that accurate predictions were unable to be made. Each dataset we created was a user gazing at one target for many stimulus cycles. We expected a high power in the target frequency corresponding to each dataset. However, the results showed a similar power spectrum for every target. With indiscernible data, getting accurate results was not possible.

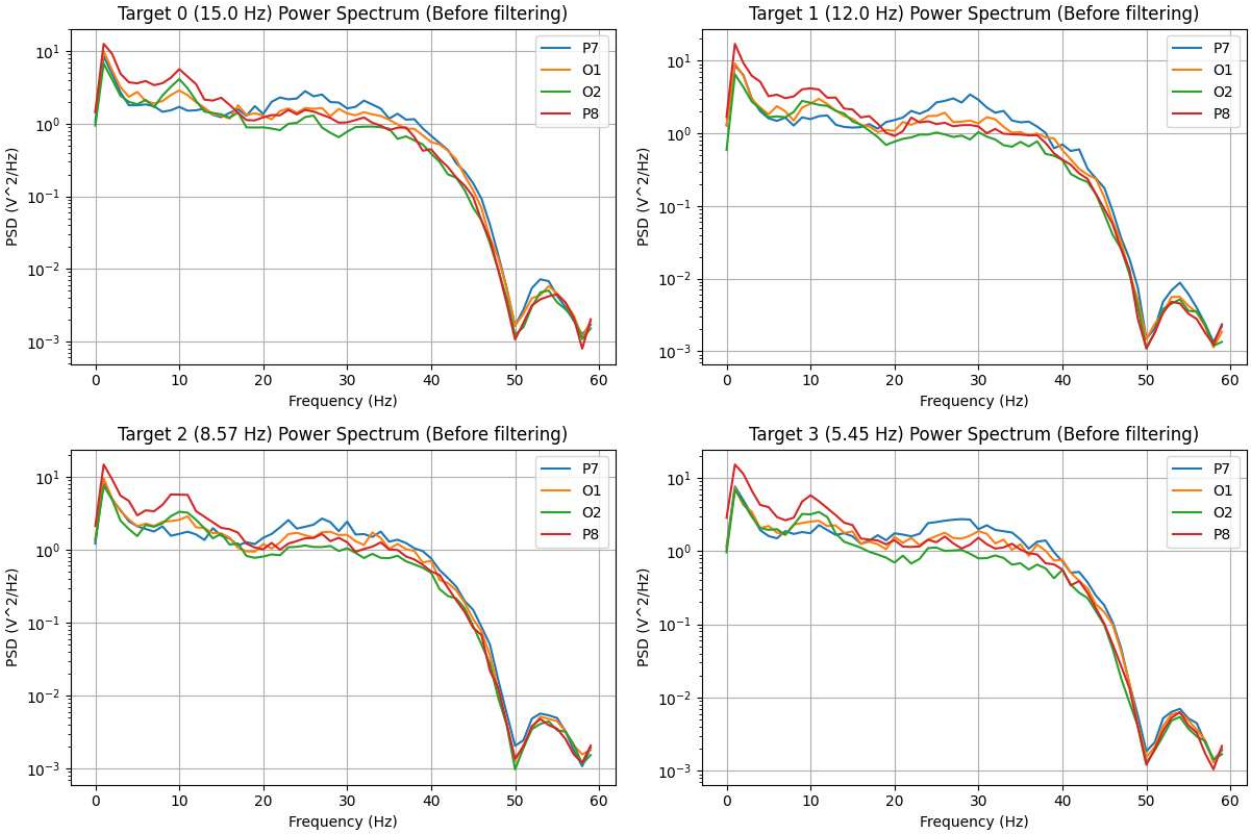


Figure 19: Power spectrum of recorded time data from EEG.

3.7 Evaluating Alternate Solutions Against Requirements

An alternate solution to this design was created when faced with the problem of inaccurate results. We decided to simulate good EEG data with added noise. In this analysis, we use 8 targets because we need more than 4 targets to gain complete control in the game. The resulting accuracy was very high, and KNN's accuracy is shown in Table 2.

Stimulus Length	Low Noise Accuracy	High Noise Accuarcy
1 second	98.7 %	64.4 %
2 seconds	100 %	97.5 %
3 seconds	100 %	100 %

Table 2: Accuracy of simulated data for low and high noise for different stimulus lengths.

The unfiltered simulated signals are shown below in Figure 20. For additive noise, there is noise across all channels, and a smaller randomness for each channel. Both of these were Gaussian noise. We also multiplied the target signal with Gaussian noise. We also added some low-frequency (0.1 to 1 Hz) noise by implementing band-limited noise. With the low noise settings, the number of sources of noise and the standard deviation of Gaussian noise were reduced.

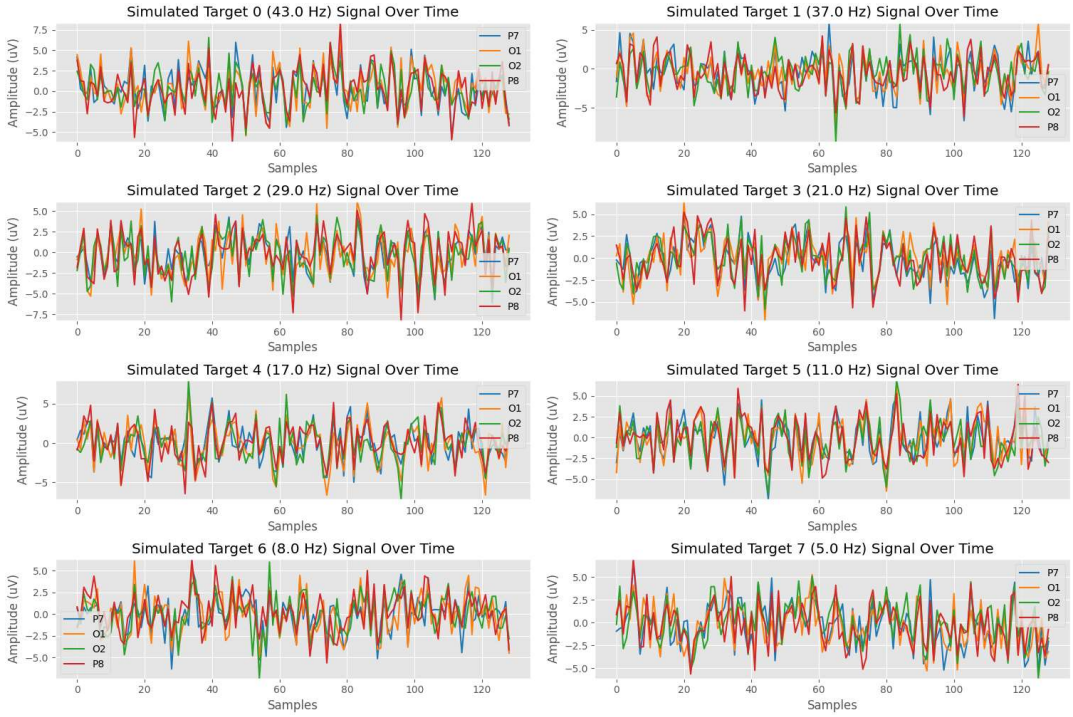


Figure 20: Simulated EEG data for 8 targets over a 1 second window (high noise setting).

The power spectrum of the simulated data, shown in Figure 21, was computed to compare against the actual EEG power spectra. In these plots, there are clear spikes at the desired target frequency. We recorded 100 seconds of simulated data, and these power spectra are taken from all samples.

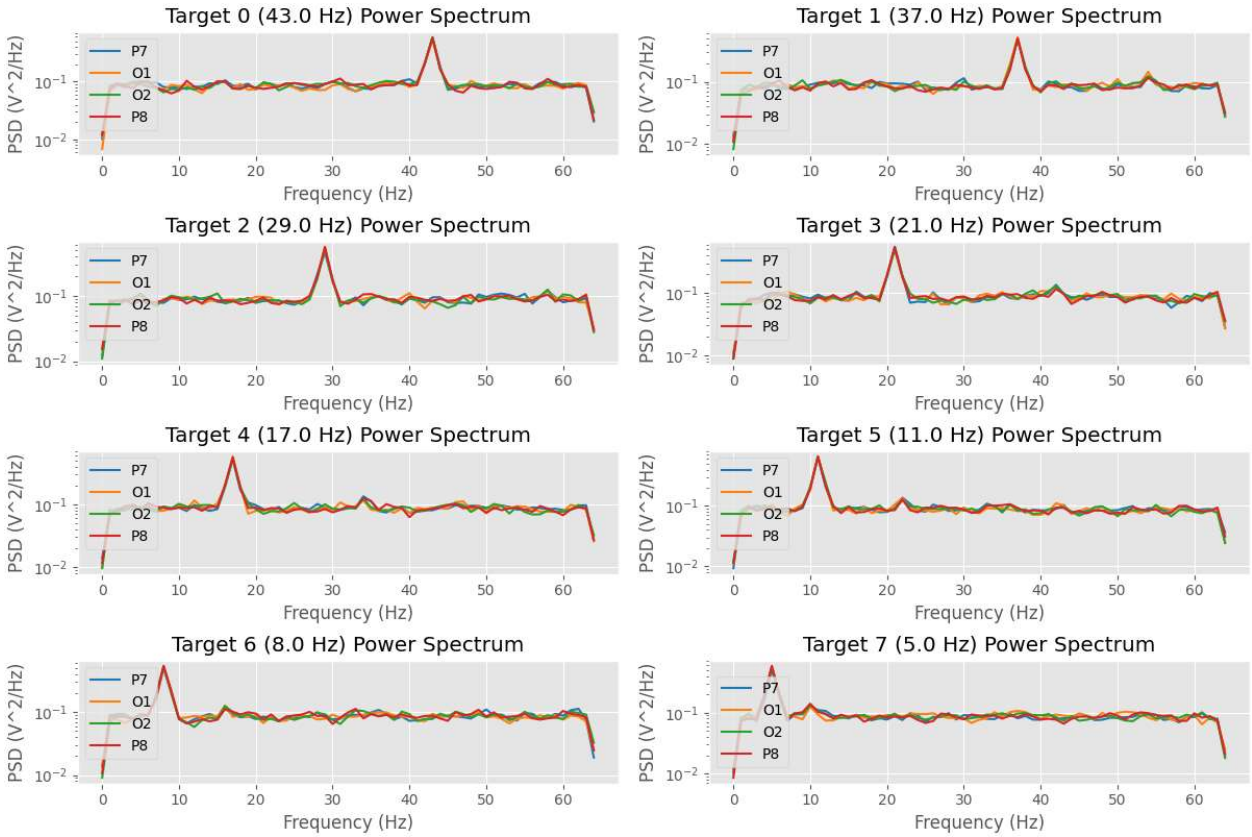


Figure 21: Simulated EEG power spectrum over all samples.

3.8 Simulated Results

For verification, we used the simulated data to play the game with the keyboard. While this was not truly playing with the EEG as we initially planned, it demonstrated the steps of our data pipeline. Once the data was updated, it was filtered using a band-pass filter allowing frequencies between 0.16 and 50 Hz. Our controls, and the order of targets, were the A button, B button, up, left, right, down, start, and select. The relatively prime frequencies we used were 43, 37, 29, 21, 17, 11, 8, and 5 Hz. If target frequencies were not relatively prime, their harmonics would overlap. Numbers 1-8 represented those buttons, and the arrow keys were mapped to the same targets as numbers 3,4,5,and 6. Each key represented a different target frequency, and pressing a key regenerated a 3 second duration of the VEP signal with random noise. Every frame, the 3 second VEP signal was fed into CCA which made a prediction and sent the appropriate command to the pyboy controller. To include non-action, we only selected a target if

there was a CCA score higher than a threshold of 0.3. We had high accuracy in this experiment, resulting in a smooth and complete experience of playing the game. A demonstration of playing the game with simulated signals is in the video presentation for this project.

3.9 Constraints

One constraint for this project is the latency. There may be up to 3 seconds of latency per command if high accuracy is needed. Accuracy is a higher priority than the stimulus length. Also, calibration is required for each new user. Another person is required to set up the EEG headset on a physically disabled user. Additionally, disturbing the EEG headset's placement on the user's head is a potential issue. Finally, this system relies on gazing for long amounts of time which makes eye strain another potential issue.

The constraints for this product also depend on the constraints of the EEG headset. For example, the best electrode placement was at Oz (located in between O1 and O2 at the center line), but our headset did not support that location. We had to use the two nearest locations of O1 and O2 instead, which may have led to decreased accuracy.

Another constraint to this project as far as its usability is the cost. It's not very easy for a disabled person to acquire a headset good enough to use the software that has been written, and if they did it would be pretty costly.

4 - Impact of Engineering Solution

4.1 Public Health, Safety and Welfare Consideration

In a safety sense, the team considered multiple options for a means of Brain-Computer Interface. While we ultimately chose the Emotiv EPOC headset, we were aware of the existing invasive technology that accomplishes the same goals. One example of current invasive BCI technology is Electrocorticography (ECoG): a pad of electrodes positioned just above the cortex. Because ECoG is closer to the source of activity, it has a higher spatial resolution (a denser set of sensors) and a clearer signal.

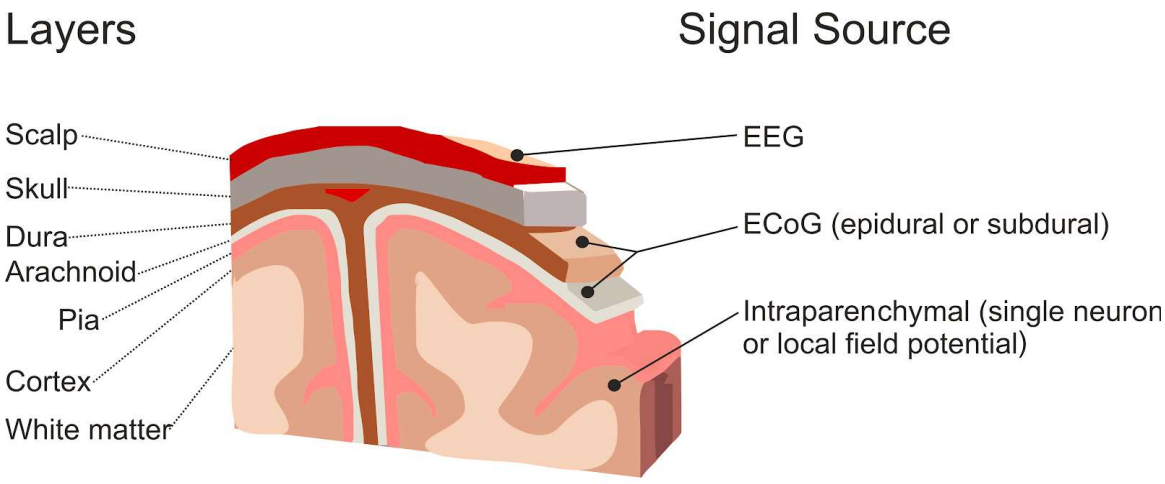


Figure 22: Representation of the inside layers of the brain and the corresponding sensing capabilities. An invasive BCI setup was considered and rejected.

4.2 Economic consideration

Given our decision to tailor our project for a Pokemon-style game, we needed to be conscious of the economic implications of our design and any pricing model we created. While standard console and handheld controllers work for all games on the system by definition, our control scheme works for only one. To charge for the use of this control scheme allows for every subsequent customized hands-free setup, ultimately de-incentivizing its use. Releasing our project as open-source software ultimately remedies this predicament. Additionally, The price of all the components combined is not significantly cheaper than buying a new gaming console.

4.3 Security Consideration

In creating profiles to store a user's brainwave signals for quality purposes, we recognize that this falls under collection of personal data. The items being collected, however, cannot readily be used against the user the way that sensitive information such as routing numbers can be. As such, we do not consider the profile to be a serious security risk. Also, because we are

using a non-invasive device, there is less risk associated with the device being hacked. We will nonetheless be sure to hold the information in a secure location and refuse to grant access to any third parties. Users should consent to their information being stored in a private way.

4.4 Environmental Impact

If there is one area where we may want to consider our large-scale impact, it is in the environmental effect of expanded use of non-invasive BCI technology. As a more sophisticated piece of technology, the Emotiv EPOC headset consumes more power than a traditional controller. Expanding its use would lead to an increase in power needed to play a game. Our remaining hardware will require minimal extra power to offset this issue.

In manufacturing, our system only depends upon existing parts and uses software to connect these parts. We do not foresee this technology drastically changing existing manufacturing of EEGs and PCs.

5 - Teamwork Plan and Division of Tasks

Task	Sultan	Matthew	Christopher	Michael	Peter
Epoc Headset configuration	X				X
Emulator/Unity mining			X	X	X
CVEP R&D	X	X	X	X	
SSVEP R&D					X
Signal Processing R&D		X		X	
UI R&D	X		X		
Machine Learning R&D		X			X
Psychopy R&D	X	X	X	X	X
CCA Model development		X	X		
Data compilation	X				X
Data filtration/manipulation		X	X	X	
Filter implementation			X	X	
KNN development		X			
Input Classification		X			
Emulator implementation	X				X
UI development	X				X
Power Spectrum analysis		X		X	
Synthetic Test Data generation		X		X	
Group Planning/Logistics	X				
Final Presentation & Report	X	X	X	X	X

6 - Budget, Cost Analysis, and Parts

The budget for this project was $600, spent exclusively on our Emotiv EPOC+ headset, commercially available to us and within our means. We agreed to spend the balance of our budget on this headset knowing it would be the only hardware component we would need for this project.

We considered multiple headsets with varying numbers of channels and other amenities for both performance and convenience. A cross-comparison of our options is included in Table 3 below. In our research, we found that similar pursuits in BCI also utilized Emotiv for their source of EEG signals. Additionally, the product was commercially available for around $600, so we felt comfortable buying it knowing that we wouldn't need to purchase any more significant materials. One of the initial draws of Emotiv was a well-documented Cortex API that would allow us to easily acquire data and format as we desired. It should also be noted, however, that in our discussions with Professor Erdogmus, he was not as optimistic about the Emotiv headset as we had been.

EEG Headset	Open Source API	Number of Channels	Pre-Assembled	Commercial Appearance	Cost of Unit
Emotiv Insight	✓	5	✓	✓	$299
Open BCI Mark IV	✓	8			$499
Emotiv Epoc	✓	14	✓	✓	$599

Table 3: Comparison of Commercial EEG Headsets

Conclusion

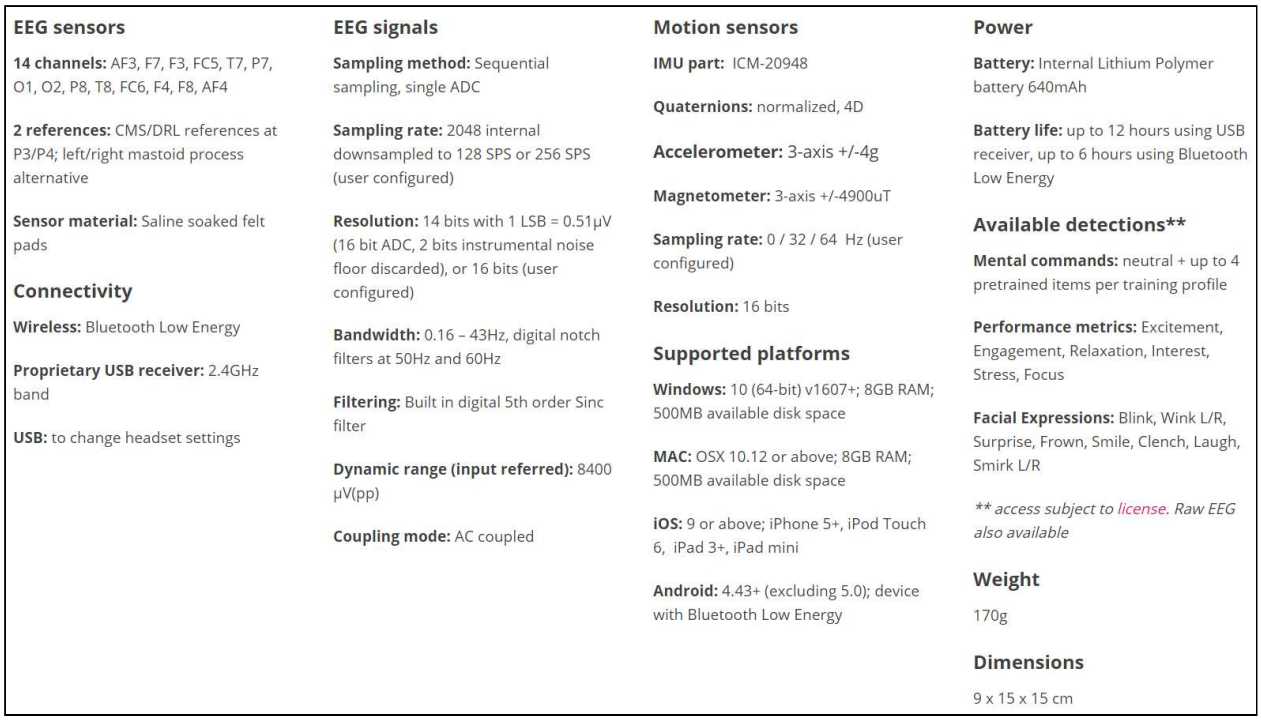
BCI technology is still at an early stage but may have a large impact once a high-speed connection between the brain and a computer is established. There is ongoing research in invasive BCIs that have higher spatial and temporal resolution than the non-invasive type we worked with. Non-invasive BCIs are easier to use and can be used by everyone without a high cost and risky procedure. This project focuses on using an EEG as a non-invasive BCI to know which command a user wants to send to a game. SSVEP is the technology that allows this to happen. There is a lot of research on SSVEP and c-VEP, but not with the application of playing a game. We think that a product to allow physically disabled people to play video games would greatly add to their life. Also, our system may be tested and used by anyone; all that is required is to wear an EEG headset. Our project is an effort to move BCI technology forward and also enrich people's lives. In the future, a key intersection of technology will be artificial intelligence and the BCI. Having a connection with the brain, we can learn more about optimizing our brain and body's functions, and we can better control prosthetic limbs or artificial body parts. The

team members expect to be interested and involved with further development of a high-speed brain-computer interface.

It is disappointing that we were not able to successfully use real EEG in playing a game with our software. However, we are pleased with the results we obtained from our simulated EEG data, as it proved our software is working properly. We were able to run a game in the window and have multiple lights constantly flickering around it, with no inconsistencies. In retrospect, we learned to do even more research before buying the EEG headset we required. Ultimately, it seems like this device was the main cause of all our problems, but it was the only device we can work with in our time frame.

Appendices

Appendix A: Emotiv EPOC+ Headset Technical Specifications



EEG sensors

14 channels: AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4

2 references: CMS/DRL references at P3/P4; left/right mastoid process alternative

Sensor material: Saline soaked felt pads

Connectivity

Wireless: Bluetooth Low Energy

Proprietary USB receiver: 2.4GHz band

USB: to change headset settings

EEG signals

Sampling method: Sequential sampling, single ADC

Sampling rate: 2048 internal downsampled to 128 SPS or 256 SPS (user configured)

Resolution: 14 bits with 1 LSB = 0.51μV (16 bit ADC, 2 bits instrumental noise floor discarded), or 16 bits (user configured)

Bandwidth: 0.16 – 43Hz, digital notch filters at 50Hz and 60Hz

Filtering: Built in digital 5th order Sinc filter

Dynamic range (input referred): 8400 μV(pp)

Coupling mode: AC coupled

Motion sensors

IMU part: ICM-20948

Quaternions: normalized, 4D

Accelerometer: 3-axis +/-4g

Magnetometer: 3-axis +/-4900uT

Sampling rate: 0 / 32 / 64 Hz (user configured)

Resolution: 16 bits

Supported platforms

Windows: 10 (64-bit) v1607+; 8GB RAM; 500MB available disk space

MAC: OSX 10.12 or above; 8GB RAM; 500MB available disk space

iOS: 9 or above; iPhone 5+, iPod Touch 6, iPad 3+, iPad mini

Android: 4.43+ (excluding 5.0); device with Bluetooth Low Energy

Power

Battery: Internal Lithium Polymer battery 640mAh

Battery life: up to 12 hours using USB receiver, up to 6 hours using Bluetooth Low Energy

Available detections**

Mental commands: neutral + up to 4 pretrained items per training profile

Performance metrics: Excitement, Engagement, Relaxation, Interest, Stress, Focus

Facial Expressions: Blink, Wink L/R, Surprise, Frown, Smile, Clench, Laugh, Smirk L/R

*** access subject to license. Raw EEG also available*

Weight

170g

Dimensions

9 x 15 x 15 cm

Appendix B: Github Repositories

Pyarcade repository (older): https://github.com/PeZeqo/capstone_pokemon_bci

Psychopy repository (newer): https://github.com/salzaabi/CapstoneJ8

References

[1] Gembler, Felix, et al. “Dynamic Time Window Mechanism for Time Synchronous VEP-Based BCIs—Performance Evaluation with a Dictionary-Supported BCI Speller Employing SSVEP and c-VEP.” *Plos One*, vol. 14, no. 6, 2019, doi:10.1371/journal.pone.0218177.

[2] Alnemari, Mohammed. “Integration of a Low Cost EEG Headset with The Internet of Thing Framework.” *EScholarship, University of California*, 15 Sept. 2017, escholarship.org/uc/item/0d90x267.

[3] Gaiba, Andrew., et al., inventors. Video game hardware systems and software methods using electroencephalography. 23 Sep. 2010. U.S. Patent 20100240458A1. *Google Patents*.

[4] Morán, A., Soriano, M.C., 2018. Improving the quality of a collective signal in a consumer EEG headset. PLOS ONE 13, e0197597.. doi:10.1371/journal.pone.0197597

[5] Nezamfar, Hooman, et al. “Code-VEP vs. Eye Tracking: A Comparison Study.” *Brain Sciences*, vol. 8, no. 7, 2018, p. 130., doi:10.3390/brainsci8070130.

[6] Cheng, Howard., et al., inventors. Eye tracking enabling 3D viewing on conventional 2D display. 22 Apr. 2014. U.S. Patent 8704879B1. *Google Patents*. 2