Hit It!

Stephen Pupa, Shreya Ramesh, George Whitfield

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract* — Rhythm games on the market either do not incorporate portable hardware or have obscure music that the player does not enjoy. Furthermore, games have predetermined beat maps, limiting song choices. Hit It! bridges this gap, by introducing an involved game interface and an active hardware apparatus for users to hit in time to their favorite songs. A beat map will be automatically generated at the user's request. The user will then begin an immersive experience where they hit the drums based on the beats on the monitor and receive either a higher or lower score based on their performance.**

***Index Terms* — Beat Maps, Digital Signal Processing, Embedded Systems, Entertainment, Graphical User Interface, FSR**

I. Introduction

"Hit It!" is a drum-based rhythm game that offers a middle ground between other rhythm games on the market. Currently, the market is filled with rhythm games that either do not involve hardware or rely on a cumbersome and expensive setup that is not widely available. Furthermore, these games use predetermined songs, limiting the demographic it appeals to based on the songs they offer. Applications outside of the game-based realms do not accurately include the entertainment factor that is desired by many demographics.

Our project, "Hit it!", offers a small and portable drum apparatus that is easily carried around as well as a feature for users to input their own song. The game will create a customized beat map, allowing the user to play to the rhythm of their favorite songs. By incorporating the creation of beat maps based on user-inputted songs, the demographic can extend to all ages.

II. Use-Case Requirements

While planning our goals going forward, we've identified 6 use case requirements that have helped us better define success in meeting our objectives.

A. *Compactness*

The hardware should fit into a standard 15-liter backpack. The ability to fit within standard carrying devices (backpacks) is crucial for enabling users to transport the system with ease. The convenience of transportation also factors into the user's impressions of the game outside of their experience with the game itself.

B. *Effective Beat mapping*

The game should play user provided songs and generate beatmaps to go along with them. The beatmaps are accurate to the input song; an accuracy of at least 80% will feel true to the provided song. The duration of beat mapping relates back to initial impressions and encumbrances to the user. If the game requires a lengthy and tedious setup process when trying to use certain in-game features, users will avoid that feature altogether. The beat mapping also needs to be accurate, otherwise it would be dishonest to call it a beatmap, as an inaccurate beatmap defies the whole point of having the beatmap to start with.

C. Frame Rate

The game should run faster than 30 frames per second. Frame Rate is an important factor relating to user enjoyment, as a slow frame rate will lead to "choppy" visuals and a worsened user experience. This factor will likely conflict directly with graphical quality, as higher quality graphics will require more time to modify each frame.

D. Latency

The latency between the user hitting the drum and the game recognizing the input should be less than 70 ms. Latency is a combined criteria that assesses the total delay within both hardware and software from the time a user provides input to the time that input is displayed back to the user. This is an extremely important criteria as rhythm games are fundamentally reliant on timing, so delays in those timings will be very noticeable

E. Ease of Setup

The user should be able to set up the game in less than one minute. The ease of setup characterizes the user's initial impressions of our game. A cumbersome and tedious setup process will establish a negative opinion of new users from the start.

F. Drum Recognition Accuracy

The hardware should correctly identify user drum hit inputs 99% of the time. Drum recognition accuracy is the number of times the drum module registers a hit and passes that information to the computer divided by the total number of times the drum is hit. This criterion most accurately defines the "consistency" of our system by assessing how often the system performs as it has been defined to.

III. Architecture and/or Principle of Operation

There are two main components of the software as illustrated by the block diagram (Appendix Figure. 6): beat tracking in Python and gameplay code in C++. For the C++ portion, Open Graphics Library will be used to create the gameplay code. OpenGL is a relatively low level tool used for creating video games that will add technical complexity to the development process. This loop is what will be called when the game begins.

When the game is executed, the Python code will analyze all of the songs that have been placed into the *songs* folder, and export their beatmap data as JSON files. Then, when the player selects a song to play, the corresponding JSON will be read by the C++ code and loaded into the game.

The primary goal of the beat tracking Python script is to export a series of timestamps where the supposed beats are detected. In order to do this, the audio file is converted into an uncompressed wav file, from which the time domain representation of the audio file can be extracted. The time stamps, the BPM, and a random determination of which button the specific beat maps to are then entered into a JSON. This will then be sent back into the game loop.

The Hardware portion of this design is focused upon a series of "Piezoresistive Sensor Drum" modules which will be 3D printed. A 3D printed prototype of the modular housing module is depicted in Figure 1.

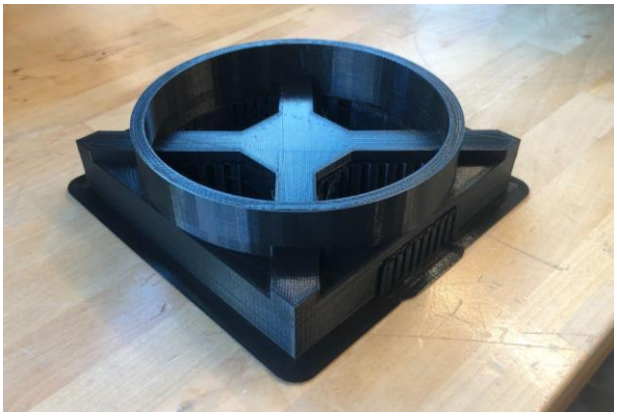

Fig 1. 3D printing of prototype drum pad module.

In addition to anywhere between 2 to 4 drum pads, there will also be a separate "ESP32 – Development Board" module that houses the ESP32 microcontroller, which will serve as the interface between the drum pads and the computer running the game program. This module will be much simpler in design, as it won't have any need for the circular opening that houses the drum's rubber pad. However, it will have more openings along the base of the module, as each opening corresponds to a drum pad connection, and each drum pad will be connected to the ESP32 module. The ESP32 module will also be slightly smaller in height while retaining the same width and length of the base, which will enable easy storage of the modules together.

The circuitry of the "Piezoresistive Sensor Drum" module is depicted in Appendix Figure. 7. This module contains the FSR and its supporting circuitry which converts the user provided force into an electrically readable analog voltage change.

The analog voltage outputs from those modules will then each feed into the "ESP32 – Development Board" module also shown in Appendix Figure. 7. This module has access to an internal Analog to Digital Converter, allowing the analog voltages received from the drum pads to be transformed into digital signals. The results of this conversion are then transmitted out of the ESP32 chip using the UART serial communication protocol into a UART-USB interface chip, which then sends that information to the connected computer using the USB serial communication protocol.

IV. Design Requirements

Each of the use-case requirements specified earlier has an associated quantitative metric that defines what it means for this project to "succeed" at satisfying the user requirements. However, the design side of the project's implementation provides additional restrictions and specifications on those requirements:

A. Latency

The goal duration between drum hit (input) and GUI response (output) is no more than 70 milliseconds. For the software stream, the frames per second goal of 30 has the consequence of increasing the software latency to 33 milliseconds by default, as any inputs received after a frame update cannot be displayed through the GUI until the next frame update, which at worst could occur 33 milliseconds later (33 milliseconds = 1/30 seconds). 33 milliseconds is much higher than the expected runtime of the gameplay loop, so this stream's latency is constrained to 33 millisecond as long as the FPS remains 30.

The hardware stream will therefore have the remaining 37 milliseconds to work with, which will be divided up between the FSR, the ESP32's ADC, and the UART-USB interface. Based on the hardware specifications of the FSR,[5] latency should be negligible as its rise time is on the order of microseconds. Therefore, the ADC and UART-USB interface need to be under those 37 milliseconds when combined.

B. Drum Recognition Accuracy

The Drums should recognize over 99% of valid drum hits. This requires a definition of "valid hits," which when coming from a user perspective should be anything the user wanted to portray as a hit, but nothing the user didn't want to portray as a hit. Based on a study of forces on drumsticks,[17] 10 N appears to be a good definition of a valid hit. The reason we chose this value is because within said study, they provide a graph of force vs. time while striking a drum, and following the initial force peak, there are several smaller force peaks caused by vibrations. None of these secondary force peaks reach 10 N, so considering all forces above 10 N to be valid should maximize our recognition rate as it maximizes what we can consider as hits without getting any false positives.

The metric of over 99% is most heavily correlated to the recognition reliability of our FSR and the design of our drum pad modules. Assuming our FSR works to the specifications provided in its datasheet,[5] there should be no issues with it recognizing forces so long as they are properly applied. This means that properly designing our drum modules will be the greater challenge in achieving this metric.

C. Ease of Setup and Compactness

The system should take no longer than 1 minute to set up. This duration starts from when the user touches the modules and finishes when the game is operating and interactable for the user. The system should also be small for ease of portability.

There are two main aspects to this setup time, physical and electrical. The physical setup time is primarily defined by factors such as untangling wires and connecting modules together. One possible solution to this is bundling the cables together, reducing the likelihood of cords becoming tangled. Another possible solution is to have locking pin headers at the connection terminals for each of our modules. This type of connector will securely lock connections in place and make it easier for users to know when their cables are connected properly. Furthermore, setup time depends on a small portability of the hardware apparatus, which is currently planned to be 576 cubic cm for a single module. With the five expected modules (four drums and the microcontroller), this is well under the use case requirements.

Regarding the electrical setup time, the primary bottleneck for this area will be the hardware boot time. Once powered, the ESP32 module will need to fully boot and connect to the computer before the game runs. The ESP32 has a boot time on the millisecond to second range,[3] which is far quicker than the minute goal we have from the use case requirements.

D. Effective Beat mapping

The game should be capable of transforming provided songs into playable beatmaps. These beatmaps should take no longer than the song's duration to produce and should match sample beatmaps with over 80% accuracy.

The first challenge is the time it will take to create a beatmap from the user's song. Having it take several hours would severely decrease the user's enjoyment of the game, so the goal is to create a beat map from a user-inputted song in the length of the audio clip or one iteration over the song. To achieve this, an amplitude threshold-based onset beat detection algorithm will be used. The suggested algorithm is to determine the consistent noise spikes and consider the beat amplitude. However, user inputs may have to be restricted with the usage of this algorithm. Coupled with dynamic programming to lessen the recalculation time for similar calculations, there is only a need to loop through the audio file

once. These factors should assist in ensuring the time it takes to create the beat map is at or below the length of the song.

The second goal is having the beat map accuracy over 80% when compared to the manually created beat maps. There are several challenges with this. There is a high potential to detect beats where there are none, and the algorithm may also miss some beats. To meet this criterion, the aforementioned algorithm will also be used. By also using the BPM to ensure that counted beats roughly have the expected timings, the beat accuracy should be above 80%.

E. Frame Rate

The game should run at 30 frames per second (FPS). 30 FPS was the target goal set by what potential users have come to expect from the gaming market. However, from a design perspective, this means that the gameplay loop of the main codebase, including the communication with the user's hardware, must take no more than 33 milliseconds. This shouldn't be a difficult goal to attain under the right circumstances, however certain other game design criteria could potentially limit the system's ability to reach this goal. The most concerning tradeoff is likely going to be that between frame rate and the graphical quality/intensity.

Due to how the GUI displays images to the user, creating more visually engaging graphics often requires greater code complexity and as a result more calculations to be performed by the code each cycle of the gameplay loop. As such, there may come a point where further increasing the visual quality of the game results in the framerate dropping below the 30 FPS target. To mitigate this, the graphics will need to be improved in an intelligent fashion. Techniques such as 2.5D graphics (or pseudo-3D) graphics are a good example of this, as they can provide similarly engaging graphics to true 3D graphics without as much computational cost.

V. Design Trade Studies

A. Beat Mapping

The main package that the beat mapping algorithm will depend on is Librosa, an audio-processing package tool. The primary use of this package is to determine the beats per minute (BPM) of the audio file.[14] This package was used instead of a manual beats per minute determination due to latency concerns. Because a majority of BPM tracking algorithms depend on a combination of Fourier transformations and loops into the transformations, the expected latency would have doubled the time for the total beat mapping computation.

Determining the beat mapping algorithm requires the calculation of the average energy throughout the audio file [2]. This is calculated through the summation of the squared intensities over the audio file. This is then normalized and compared to an instantaneous energy. This does require looping throughout the wav files.

Because the algorithm for determining the time stamps of the beats is a loop based approach, one of the tradeoffs in this algorithm is accuracy for speed. Ordinarily, these strategies should be avoided due to the desire for a lesser time of computation. However, because a majority of these values within the loop are recalculated, dynamic programming can be used to lessen the computation time. Accuracy was prioritized over speed because the inclusion of the dynamic programming should allow for a faster computation time than otherwise. While this does sacrifice efficiency in space, the use case requirements for the project do not set a limit for drive space. This approach maximized time savings as well as accuracy.

B. Graphics Programming Tradeoffs

The gameplay for *Hit It!* will be implemented in C++ with OpenGL. Video games are typically created nowadays using game engines such as Unity or Unreal. OpenGL is a more low-level tool for displaying video game graphics compared to game engines, which adds technical complexity to our project. Although OpenGL stands for "Open Graphics Library", it is not actually a library but rather a specification for the interface between the software and GPU.

There are several alternatives that we could have used instead of OpenGL for graphics displaying. One popular alternative is Pygame, which is a Python library for making games. Pygame also comes with built-in sound support, which OpenGL does not. We also could have used DirectX or Metal, which are graphics interfaces for Microsoft and Apple platforms. OpenGL was chosen over these alternatives because 1) the team has experience with OpenGL, 2) there are numerous cross-platform OpenGL libraries available (this is not the case for DirectX and Metal because they are OS-specific), 3) game programming in C++ with OpenGL faster than Pygame (due to Python being a slower language), and 4) C++ is common for development in the games industry.

C. Hardware Trade Studies

For the hardware portions of this project, there were several comparisons that needed to be made. However, the most important decisions were centered in four areas, the drum pad material, the FSR, the microcontroller, and the drum housing material.

Neoprene vs. Natural Rubber

Regarding the choice of drum pad, our team looked at several options with good elasticity and eventually settled between two materials, those being Neoprene and Natural Rubber. Neoprene is a synthetic version of rubber known for its resilience and elasticity, which makes it an excellent material for products like bouncy balls. Natural rubber has many of the same positive qualities regarding its elasticity, however it doesn't have the same resilience as Neoprene, although this isn't an issue since it will not be subjected to extreme temperatures or conditions for our purposes. Our team's original goal was to purchase samples of both, allowing their differing functionalities to be tested in a prototype drum firsthand, however, the cost ended up being the deciding factor here, as purchasing natural rubber sheets through Amazon was an order of magnitude higher than obtaining Neoprene rubber sheets. This circumstance made the decision between the two quite easy, as our team couldn't justify spending half of our budget on a material we may or may not end up using in the final product.

Flexiforce Pressure Sensor vs. Interlink Electronics FSR

The main qualities needed in the FSR for this project were response time and reliability, which derive from the design requirements of latency and drum recognition accuracy respectively. Our team's two main FSR candidates, those being the Flexiforce Pressure Sensor and the Interlink Electronics FSR 402, both satisfied these criteria, however there were also important secondary considerations.[4-5] Specifically, the Flexiforce Pressure Sensor had the benefits of high precision and a large force sensing range of approximately 0 to 445 Newtons, whereas the Interlink Electronics FSR 402 had the benefits of being much cheaper as well as having a larger sensing area. However, the FSR 402 has a much smaller sensing range than the Flexiforce sensor, as it can only reliably measure from 0.2 to 20 N, and in case the project needed to use different levels of force during a post-MVP stage of the project, our team opted to prioritize the Flexiforce sensor. However, after performing some preliminary tests on both sensors, it became clear that the user's force was dampened to such an extent through a layer of Neoprene that the vast majority of the Flexiforce sensor's measuring range was going unused. This eliminated the primary reason for selecting the Flexiforce sensor, and as such our team decided to focus our further efforts on the FSR 402 sensor instead since it performed similarly for a much cheaper price.

ESP32 vs. Arduino

Regarding the selection of microcontroller, our team narrowed down the choices to two options. The first was the ESP32 designed by Espressif and the second was the Arduino Zero made by Arduino. These two boards share many similarities, as both are 32-bit microcontrollers that operate at 3.3 Volts, and which are programmable using the extremely accessible and user-friendly Arduino IDE. However, there were a few defining factors which ultimately solidified the ESP32 as the preferable microcontroller.[1,3] Firstly, the Arduino Zero uses an ATSAMD21G18 which gives it a clock rate of 48 MHz, whereas the ESP32 uses a Tensilica Xtensa LX6 microprocessor chip, giving it a clock rate between 160 MHz and 240 MHz depending on the operating mode. Due to how having low latency is a core design requirement of this project, the ESP32 is the preferable option here. Secondly, the ESP32 is overall much cheaper to purchase at around $11 for some development boards whereas the Arduino Zero is closer to $40. There are certainly cheaper Arduino options available, but even the cheapest Arduino boards tend to be around $20, which is still higher than the ESP32 development board. Lastly, our team also has great deal of experience working with the ESP32 microcontroller, which would result in time savings of the course of the project. These three factors are what ultimately solidified the ESP32 microcontroller as the preferred option for this project.

3D Printing vs. PVC Piping

One tradeoff that our team experienced was between 3D printing and using available materials such as PVC pipes for developing our drum module housings. Initially, the team wanted to use 3D printing for all of the drum modules and the ESP32 module, however, the first round of prototype printing for the drum modules was far more expensive than anticipated. To 3D print a single prototype drum module, it cost approximately $55, which when expanded to 4 more drum modules and an ESP32 module would be an additional $275 expenditure on our team's budget. This is theoretically an expenditure our project can afford, as not much else within the design requires much spending, however, it certainly has prompted the exploration of alternatives. One possible alternative is to use PVC piping as the circular holder for our drum modules. This would be an order of magnitude less expensive, however the only downside is that the team would need to use power tools and adhesives to achieve similarly

designed drum pad modules. The primary tradeoff here is price vs. time and effort, as 3D printing requires far less time and effort whereas using other materials such as PVC pipe would save on costs. Currently, the team is still leaning towards the 3D printing route, because as mentioned previously there are few other expenditures we will have. However, should any unexpected design changes or post-MVP goals be created in the future, the option of PVC piping and other similar alternatives will likely become the preferable option.

VI. System Implementation

Hardware System Implementation

As mentioned within Section III, the "Piezoresistive Sensor Drum" circuitry will be placed within a 3D printed drum housing depicted in Figure 1. The way this module will work is by having the FSR fixed to the central raised platform (as depicted in Figure 2).

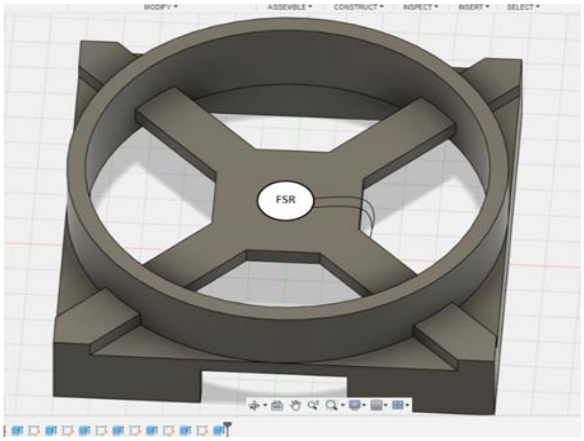

Fig 2. Top-down view of 3D rendered drum module

This will bring the sensor much closer to the force input that it is meant to be measuring. However, between the FSR and the user there will also be 1-2 sheets of Neoprene rubber, which are meant to slightly dampen the force, giving the drum a more satisfactory feeling when hit while also protecting the FSR from direct contact with the user.

The FSR forms a voltage divider with the 1 MOhm resistor also housed within this module, so that when the FSR receives the force and its resistance drops, the voltage at the node between the FSR and the resistor will change drastically. This changing analog voltage, as well as the GND and VCC (3.3 Volt) nodes which connect with the 1 MOhm resistor and the FSR respectively, will be passed through the opening in the modules base, as depicted below.

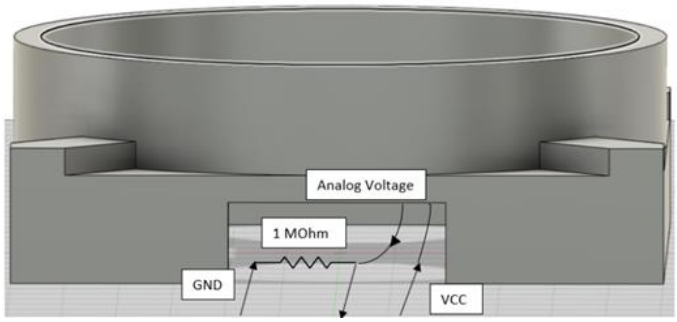

Fig 3. Side view of 3D rendered drum module

As the various voltage nodes are passed through this opening in the module's base, they will also be fed through a locking pin header, which will reduce the possibility of suddenly disconnecting with a module during use.

The analog voltages from the drum modules will then feed into the ESP32 development board module through its GPIO pins, specifically those which can be used with the ESP32's internal Analog to Digital Converter. This allows the ESP32 to read said pins and get their respective voltages represented as digital values. From here, the ESP32 will then package that information and transmit it using the UART serial communication protocol to the UART-USB interface chip present on the development board, which allows for the message to be converted to the USB communication protocol before being forwarded through the board's USB-C port to the connected computer. The only component in this stream that requires programming is the ESP32 microcontroller chip, which will be programmed using the Arduino IDE due to the ease of use and convenient helper functions that IDE provides.

Regarding potential plans for once MVP is reached, our team has discussed several times the possibility of adding LEDs to the drum pads, which would light up upon the user hitting the drum. This will require slight modification to the drum pad modules, as currently there is no opening for those LEDs to be situated upwards towards the user. Furthermore, modifications would also need to be made to the Piezoresistive Sensor Drum circuitry, however this wouldn't require any additional lines to be fed from the ESP32 module, as the VCC power line and the changing resistance of the FSR would be enough on their own to trigger these LED sequences.

Software System Implementation

Beat Tracking

Multiple packages will be used to create the beat tracking script. Firstly, Librosa is an audio-processing software package written in Python.[14] This package will be used to obtain the beats per minute (BPM) of the audio file. A majority of the rest of the digital signal processing will be done with relatively simple python packages, including Numpy, Scipy, and MatPlotLib [10,13]. The BPM can then be used to determine the approximate amount of time between beats in the audio file. The expected amplitude will be determined using an amplitude threshold-based onset beat detection algorithm. Currently, the assumption is that the audio files have a static BPM. By determining the average energy over the entire file and comparing it to the instantaneous energy, beat energy can be calculated as a larger change between intensities when compared to other points in the audio file. This can then be mapped to the time [2]. One issue with this method is the implicit assumption that the BPM is static, as well as the constant scalar value by which average energy is multiplied by that the instantaneous energy must be greater than. However, a predictive algorithm can be implemented in which the genre of the audio file can be determined, allowing for a more accurate scalar value. As discussed in the design trade studies section, the loop based structure of the algorithm has proven to take an excessive amount of time in computation. This violates the design requirements of the time it takes to compute the beat map. To remedy this, dynamic programming will be used. By using memory to recall the previous calculations, a significant number of loop calculations will be discarded, significantly decreasing the computation time of the beat mapping algorithm. In order to send the timestamps of the beat tracking algorithm to the main gameplay loop, the time stamps themselves will be exported into a JSON format that is able to be read by the game loop.

Post MVP, the current plan is to implement a version of melody tracking. Based on previous work in beat tracking, this will likely involve the determination of a consistent high frequency pattern. By isolating this pattern in the FFT, this can then be mapped to the already present JSON file to be read through the game play loop.

Gameplay Code

The gameplay will be implemented using C++. CMake will be used to compile and link source files, as well as all of the external libraries used. The graphics will be created using OpenGL, which is a popular interface for rendering graphics. Additional libraries are used for reading files and playing audio. C++ was chosen as the language for the gameplay code because it is common in the video game industry, and the members of the team are personally interested in OpenGL.

Hit It! Will use several external libraries for writing graphics code:

- GLAD - Implementation OpenGL functions
- glm - Math utility library for graphics programming
- GLFW - Cross platform window creation

For file I/O and playing audio, the following libraries are used:

- libaudiodecoder - reading WAV and MP3 files
- jsoncpp - reading beatmap files encoded as JSON
- PortAudio - cross platform real time sound rendering

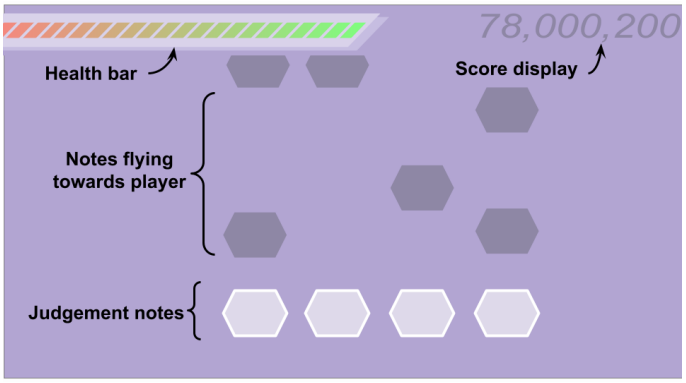

Fig 4. Mockup of gameplay GUI.

The GUI for the song selection and gameplay was heavily inspired by *Dance Dance Revolution* (DDR). The current design for the song selection screen is very similar to the song selection GUI from DDR. The health bar in our gameplay mockup is placed at the same location as the health bar in DDR.

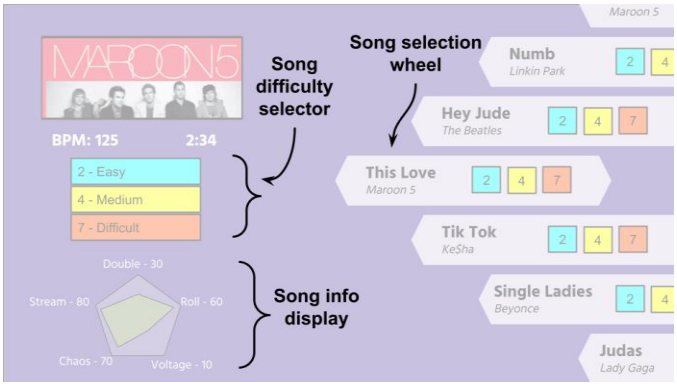

Fig 5. Mockup of song selection GUI

VII. Test, Verification and Validation

Most of the use case requirements and design requirements outlined previously can be assessed through some form of quantitative metric. Depending on whether the system succeeds or fails in reaching each of these criteria, as well as the extent to which it succeeds or fails, there may be a need to re-evaluate some of the design choices currently in place. Rectifying these issues will be discussed further in the mitigation plans (VIII. D).

Tests for Latency

To test the system's latency, the current plan is to use an iPhone camera to record the entire system so that the hit to the drum and the GUI response are both within frame. iPhones typically have a "slow-mo" option when recording which allows for up to 240 fps, which will be sufficiently precise for testing values on the magnitude of 70 ms. Additionally, this same method can be used to examine the individual latencies throughout the system as they aren't much smaller at around 30 to 40 ms (provided different start and end conditions of course).

Tests for Drum Recognition Accuracy

Drum recognition accuracy is the number of times the drum module registers a hit and passes that information to the computer out of the total times it is hit. The sample size of total hits will need to be at least 1000 strikes, since that is an order of magnitude larger than the percentage that defines success or failure in this test. Furthermore, although any force above 10 N will be considered valid from a design perspective, the accuracy value shouldn't be based on the lowest acceptable value, as accuracy is likely going to be greatest near that cutoff threshold. Instead, the accuracy should be based on expected "Average" force levels, which can be assessed easily by recording the force created by random participants when they are asked to hit the drum, and then averaging the level of their forces to get an "Average" force level to use for the further 1000 testing iterations.

Tests for Ease of Setup

To test this criterion, the setup times of random, uninvolved participants, will be recorded and averaged. This will give an indication of what new users of the system will likely experience, as these users will by design have no prior familiarity with the system, and therefore also no experience setting it up. For the number of trials, the aim is for anywhere between 10-20 participants.

Tests for Compactness

To test the compactness metrics, the dimensions of the system (or just the individual module sizes taken from their design schematics) need to be measured and summed. However, for a more practical implementation of this test, the system should also be placed within a variety of backpacks. This will demonstrate success in reaching the original qualitative goal in addition to success in reaching the quantitative measurement goal.

Tests for Beat Tracking Timing and Accuracy

To test the beat mapping accuracy, the beat overlap between beats generated by the system's beat map and those from a sample beat map will be compared, and as long as the two are in sync for 80% of the song, that will be considered successful. To test the duration, the time between the script receiving a WAV file to the time it outputs the JSON file will need to be measured. These points mark the start and end of the beat mapping script's runtime respectively, so the distance between the two is the script's total runtime.

Frame Rate

To test that the system is achieving the goal of 30 FPS, a timer will be added to the gameplay render loop in the C++ code. This timer will be able to record and output the time between frames, which therefore allows for the average frame rate over a long period of play to be easily calculated.

VIII. Project Management

A. Schedule

The proposed schedule for this semester can be seen in Appendix Figure. 8. At the end, there is a period for slack time in case an extra week is needed. Earlier on, the planning phases largely surrounded the proposal presentation. After the feedback from the proposal presentation, the work was divided into three main categories: signals processing, graphics creation, and hardware. Each member worked to test out the different packages and tools that may be used. After the design review, the final packages will be determined. Because some work was already done before the design review, the individual portions of the project should be created before Spring Break. The week after Spring Break will largely be dedicated to integration between the members' parts. The expected MVP will be obtained on March 21st. Furthermore, the extra levels and aesthetic changes made before the final deadline with a week of slack time in case of issues in the creation process.

Throughout this entire process, beta testing will be used to ensure that the game is enjoyable to the target audiences.

B. Team Member Responsibilities

Shreya
- Implementation of musical analysis algorithms for beatmap generation
- Exporting beatmap to JSON

George
- Implementation of gameplay: placement of notes on screen, handling player input, calculating player's score, playing audio, song selection, etc.
- Game design of gameplay

Stephen
- Design and Manufacture of Drum pad/ESP32 modules
- Design of FSR circuitry
- Programming of ESP32

C. Bill of Materials and Budget

The budget for our project can be seen in Appendix Figure. 9. The most noticeable feature is that it is mostly dominated by the 3D printing costs of our individual drum pad and ESP32 modules, which will cost around $55 each based on recent 3D prints. However, even accounting for six 3D printed modules (1 prototype, 4 for drum pads, 1 for ESP32), that still will only total half the available $600 budget, and of the remaining $300 there likely won't be too much expenditure after what has already been purchased, as no more ESP32 development boards or Flexiforce sensors will likely be needed.

D. Risk Mitigation Plans

If the OpenGL library fails to obtain the results needed or the interface proves too difficult to be understood, Pygame will be used in place of this. Because the team has used Pygame in more basic programming classes before, Pygame serves as a useful backup that can still create an engaging experience for gameplay. However, if certain advanced features of the gameplay fail (e.g. audio failures or latency errors), then these will simply be accounted for and incorporated into the game as a whole.

If the beat tracking does not have an accuracy that is playable by the user, then predetermined beat maps will be used in place of a user-inputted song. However, if there is some degree of accuracy but with a significant number of false positives or negatives, then these errors will be included in the final beatmap. Because these false readings are likely indicative of a melody or other part of the song, the inclusion of these false readings should not create a discrepancy in the "mood" of the song.

For the hardware, if issues with detection between the microcontroller and the graphical interface do occur, this is likely because of mechanical failures in the equipment. Again, this can be remedied through the creation of more robust hardware. However, if the problems with the microcontroller and the sensors are not able to be diagnosed, then buttons will be used to replicate the sensors. This should have the ability to function in the same manner as the sensors, but they will be simpler to install.

IX. Related Work

One game which shares a lot of similarities with our product is the franchise rhythm game Rock Band. Specifically, both our design and Rock Band have controllers modeled after drums, in which there are 4 drum pads that the user is meant to hit in rhythm with the notes appearing on the screen.

However, while our designs share the same number of drum pads, the Rock Band controller also has an additional input in the form of a bass pedal. This bass pedal adds extra complexity and difficulty to the game as users will need to split their focus between their foot on the bass pedal and the drums they hit with the drumsticks in their hands. Our design can take inspiration from this style of user engagement, as having a secondary mode of user input may be an interesting goal to have post-MVP. While a foot-based pedal may be somewhat difficult due to the lengthy cords it would inevitably require, we could have some non-drum form of input that the user must also manage, such as a switch or dial that they must adjust periodically, which would help divide their attention and add an additional challenge for experienced users.

One notable downside of Rock Band however is the lack of song flexibility their product has, as although they provide a large and varied list of popular songs, such as "Caught Up In You" by .38 Special and "I Bet My Life" by Imagine Dragons, their website also has a tab where users can request songs to be put in the game.[9] The presence of this tab highlights the major limitation of their product which our design is aiming to solve, as when a user wants to experience a specific song in game, the best they can do is message the developers of the product themselves and hope that they'll be able to obtain the rights. The fact that Rock Band felt the need to accept this input also proves the importance of our system's beat mapping functionality, as by avoiding this process altogether our product will save users the significant and unreliable hassle of messaging developers directly.

X. Summary

In summary, *Hit It!* is a rhythm game where the user hits drums to the beat of a song. The game design is inspired by popular rhythm games such as *Rock Band* and *Dance Dance Revolution*. The technical complexity of our project comes from our implementation of the drum hardware, signal processing algorithms, and use of low level graphics libraries.

The user interest in this game is primarily because of the active and engaging hardware and software experience. Furthermore, because users are able to input their own songs into the game, the game creates a more personalized experience. Because some of these design traits are currently lacking in the market, the introduction of this fun, personalized experience will allow users of all ages to play the game.

Throughout the design process and determining the distribution of labor, we have created a communicative process between team members. Clear communication between each of the team members will be key to integration, especially because each member is working on separate components of the game. We have not started integration yet, but we plan to make it as smooth as possible by using email and Discord (a collaborative messaging application) regularly. Moving forward and post MVP, our plan is to add more aesthetic features such as flashing lights on the drum heads and better game GUI. This seems to largely depend on the increased complexity of the software and the increased aesthetics of the hardware.

Glossary of Terms

Beatmap - the sequence of notes that fly towards the user during the game

Rhythm game - games (typically video games) which require the player to move their body in time with music

Beats - The groove of a song. These are the musical rhythmic hooks that the listener feels when listening to a piece of music.

BPM - Beats per minute. This is also referred to as the tempo. The term “beat” as used in BPM does not refer to the “beats” as defined above, but rather it refers to the steady musical pulse that the listener feels when hearing music.

Judgment Notes - In rhythm games, the “judgment notes” are the notes on the GUI that the beatmap notes will fly towards; the player must strike the game hardware when the beatmap notes align with the judgment notes.

FSR - Acronym for Force Sensitive Resistor. These resistors are usually made of piezoresistive materials, which give them the property of having different resistances under different levels of pressure/force. Generally, the resistance of FSRs decreases as more force is applied.

ADC - Acronym for Analog to Digital Converter, which converts an analog signal into a digital signal.

References

[1] “Arduino Zero.” Arduino Online Shop, Arduino, <https://store-usa.arduino.cc/products/arduino-zero.>

[2] “Beat Detection Algorithms.” *GameDev.net*, http://archive.gamedev.net/archive/reference/programming/features/beatdetection/index.html.

[3] “ESP32 Series Datasheet.” Rev. 3.8. Espressif Systems. Pg. 8-14. <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>

[4] “FlexiForce® Standard Model A201.” ZFLEX A201-100. Rev. A. Tekscan. Pg. 1-2. <https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/FLX-A201-A.pdf>

[5] “FSR® 400 Series Data Sheet.” PDS-10004-C. Rev. 2. Interlink Electronics. Pg. 2-3. <https://cdn2.hubspot.net/hubfs/3899023/Interlinkelectronics%20November2017/Docs/Datasheet_FSR.pdf>

[6] glad used for implementation of OpenGL functions. Feb 2022 https://github.com/Dav1dde/glad

[7] GLFW used for creating OpenGL context and window. Feb 2022 https://www.glfw.org/

[8] glm used for mathematical helper functions with OpenGL. Feb 2022 https://github.com/g-truc/glm/releases/tag/0.9.9.8

[9] “Harmonix Music Systems, Inc..” Rock Band Rivals, Harmonix Music Systems, Inc., <https://www.rockband4.com/.>

[10] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

[11] jsoncpp used for reading JSON beatmap files. Feb 2022 https://github.com/open-source-parsers/jsoncpp

[12] libaudiodecoder used for decoding music files. Feb 2022 https://github.com/asantoni/libaudiodecoder

[13] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[14] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17(3), 261-272.

[15] PortAudio used for audio playback. Feb 2022 http://www.portaudio.com/docs/v19-doxydocs/index.html

[16] Vries, Joey de. *Learn Opengl - Graphics Programming Learn Modern Opengl Graphics Programming in a Step-by-Step Fashion*. Kendall & Welling, 2020. Feb 2022.

[17] Wagner, Andreas. “Analysis of Drumbeats: Interaction between Drummer, Drumstick and Instrument.” *Master's thesis at the Department of speech, music and hearing*, Kunglia Tekniska Högskolan, 2006, pp. 19–23.

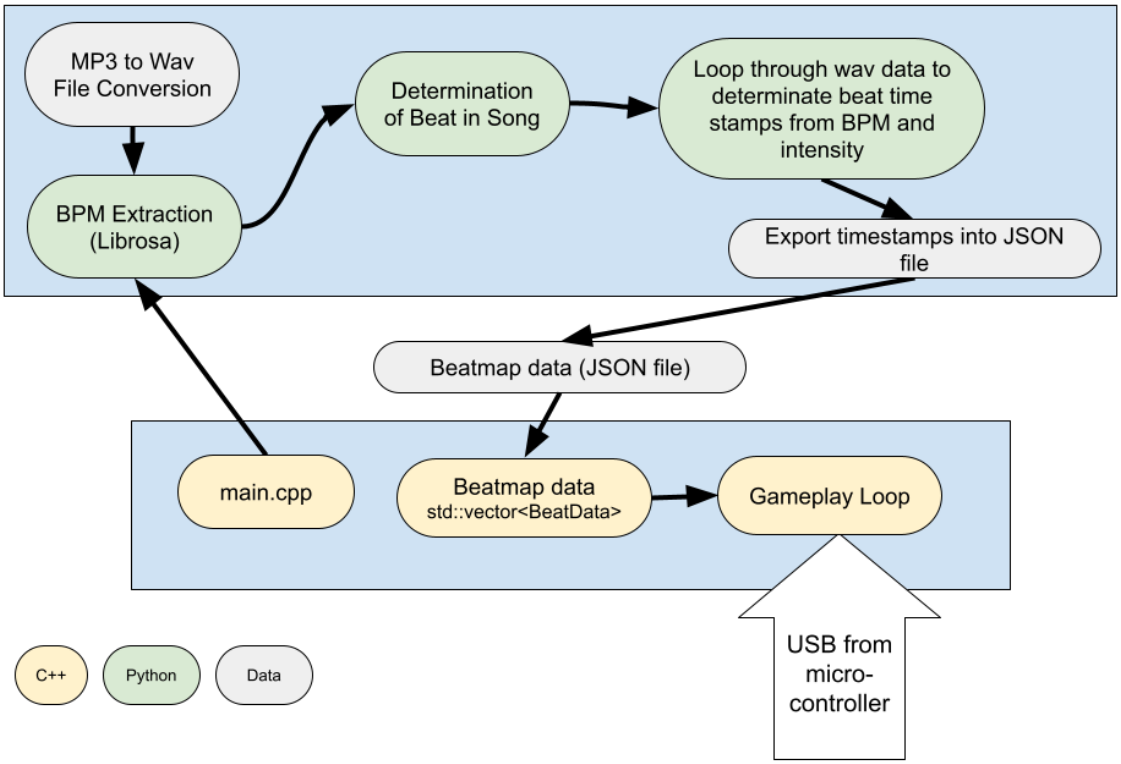

Fig 6. Block Diagram of the Software information stream

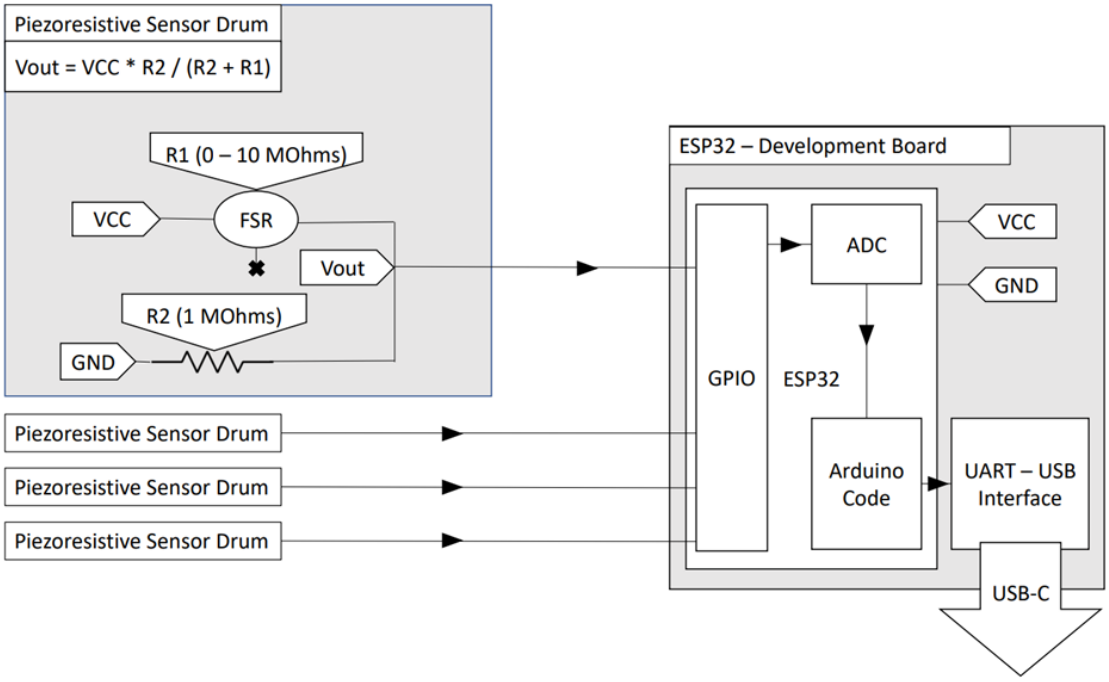

Fig 7. Block Diagram of the Hardware information stream

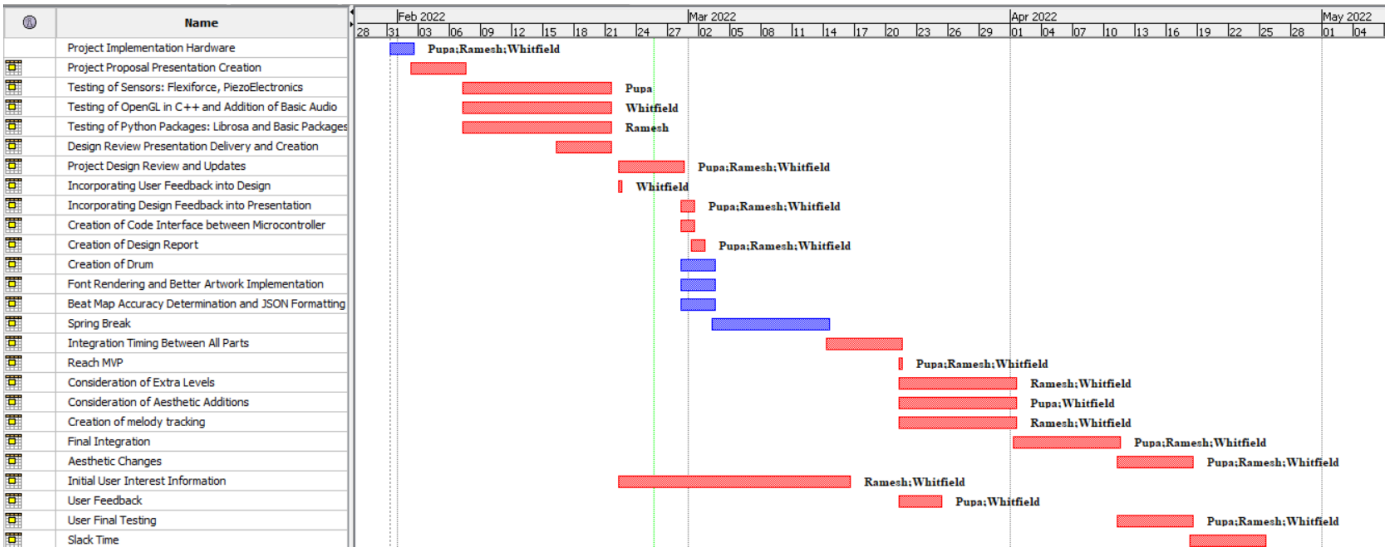

Fig 8. Gantt chart depicting the project's planned tasks

Hardware Testing Supplies (Items already purchased)

Qty	Item Name	Manufacturer	Description	Distributer	URL	Unit Price	Shipping + Tax	Total Price
1	FlexiForce Pressure Sensor - 100lbs.	FlexiForce	Piezoresistive force sensor	Sparkfun	https:/	$19.95	$13.68	$33.63
5	MCP6004-E/P	Microchip Technologies	Op Amp Quad Low Power Amplifier	Arrow Electronics	https:/	$0.63	$6.63	$9.80
1	DZS Elec 15PCS 35mm Piezo Disc Transducer	N/A	Cheap backup teasting piezo transducers	Amazon	https:/	$8.99	$0.00	$8.99
1	10-Pack 6mm Rubber Neoprene Square Padding Sheets	Genie Crafts	Neoprene rubber sheets for drum pad testing	Amazon	https:/	$9.99	$0.00	$9.99
1	HiLetgo ESP-WROOM-32 ESP32 ESP-32S Development Board	HiLetgo	ESP32 development board	Amazon	https:/	$10.99	$0.00	$10.99

Drum Module Supplies (Plan to Purchase/Need funding for)

Estimate Qty	Item Name	Manufacturer	Purpose	Distributer	URL	Estimate Unit Price	Total Estimate Price	Additional Notes
4 to 6	3D Printed Drum/ESP32 Modules	CMU 3D Printing Services	Isolating Circuitry from user and/or harm	CMU TechSpark	N/A	$55.00	$220 - $330	
3 to 5	Simple PCBs	Not Yet Known	Having custom PCBs will reduce chance of circuitry damage	N/A	N/A	Unknown	Unknown	Replacable by "Tiny Breadboards"
3 to 5	Tiny Breadboards	Adafruit	Mapping the internal circuitry of each module	Adafruit	https:/	$4.00	$12 to $20	Replacable by "Simple PCBs"
2 to 4	Amphenol FCI Clincher Connector (2 Position, Male)	Amphenol Communications Solutions	Making the IE FSR connections more reliable	SparkFun	https:/	$2.10	$4.20 to $8.40	
4 to 8	22-23-2037	Molex	Locking Pin Headers which will be placed at module interfaces	Mouser	https:/	$0.37	$1.48 to $2.96	
5	OY105KE	Ohmite	1 Mohm Resistors. Used for creating voltage dividers.	Mouser	https:/	$2.57	$12.85	

Non-purchased Items (Items owned previously)

Qty	Item Name	Manufacturer	Purpose
2	FSR 402	Interlink Electronics	Primary Force Sensitive Resistor
100+	Jumper Wires	SparkFun Electronics	Facilitating Electronic connections within indivudal modules
4	OY105KE	Ohmite	1 Mohm Resistors. Used for creating voltage dividers.

Fig 9. Bill of materials (BOM)