Team A5 - *Hit it!* - Use Case

Hit It! is a drum-based rhythm game that aims to serve as a middle-ground between other rhythm gaming alternatives.

The Rhythm gaming market:

- Few rhythm games involve hardware
- Limited demographic appeal

Hit it! solves these issues:

- Small and portable hardware
- User inputted songs

ECE Areas:

- Hardware Systems
- Software Systems
- Signals and Systems

Use Case Requirements

Portable

Should be small enough to fit within an average backpack (<15 Liters)

Plays User's Music

Hit It! should be capable of generating beatmaps from user provided songs

Accurate Beatmapping

Generated beatmaps should align well with (>80% accurate) external beatmap software or our hand-calculated beatmaps

Latency

Latency between player input and game reaction should be < 70 ms

Ease of Use

Setup should be quick and easy (<1 min)

Input Recognition

The Drums used for player input should recognize the overwhelming majority (>99%) of hits by the player

Solution Approach-Hardware

- ESP32 Microcontroller Development Board
  - Converts analog voltage inputs into USB communication protocol
    - Analog Voltage > ADC > UART to USB interface > USB protocol
  - High clock speed (240 MHz) to reduce latency
- FlexiForce Pressure Sensor
  - Converts physical force (user input) to analog voltage
    - Allows for creation of voltage divider
  - High Precision (< 3% deviation)
- Integration
  - Microcontroller sends drum hit voltages to C++ program over USB-C port
  - Determines when the user has hit the apparatus

Solution Approach-Software

Integration of Beat Tracking Algorithm into Gameplay Code

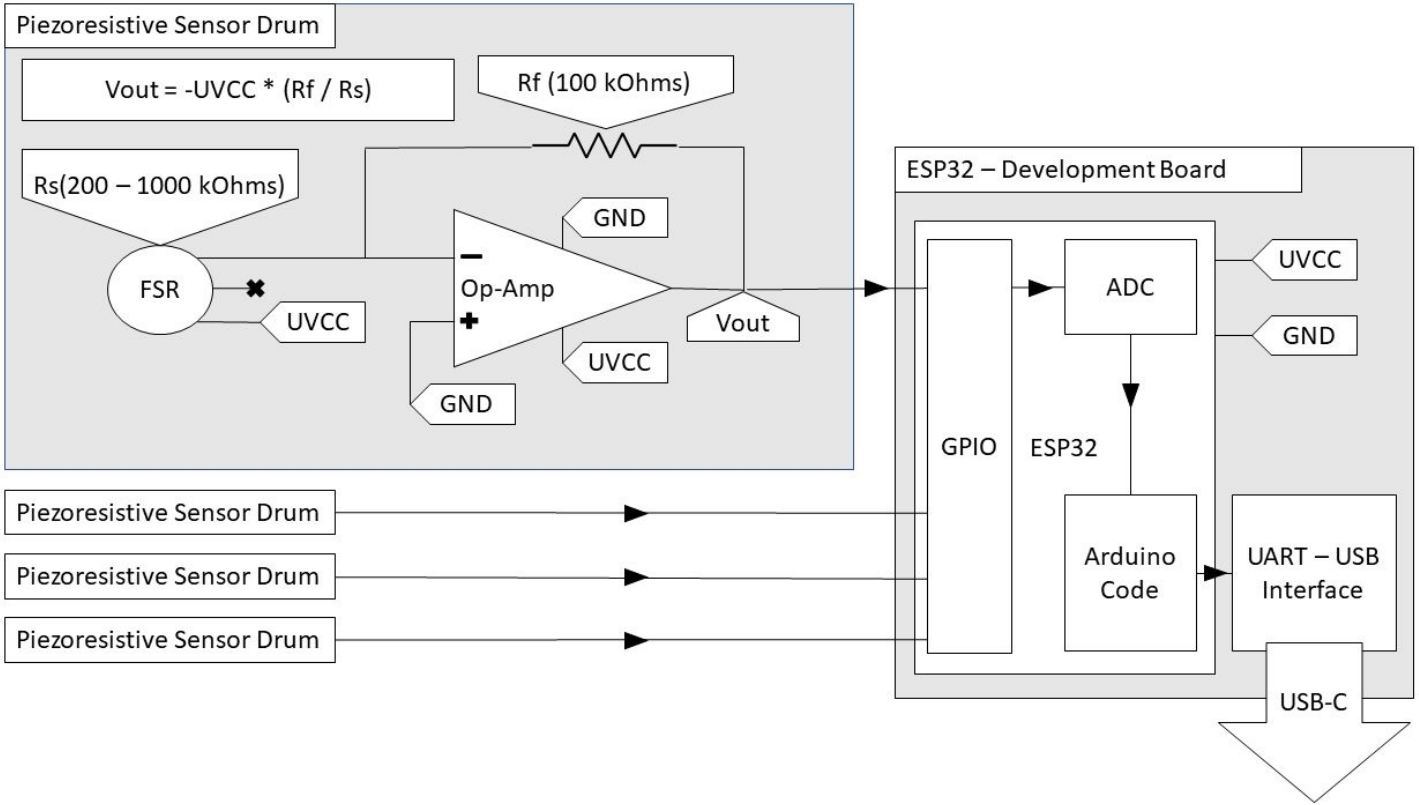
Python - Beat Tracking

- Use of Scipy, Numpy, Librosa and Matlab packages for BPM
- Output of time stamps of beats in JSON

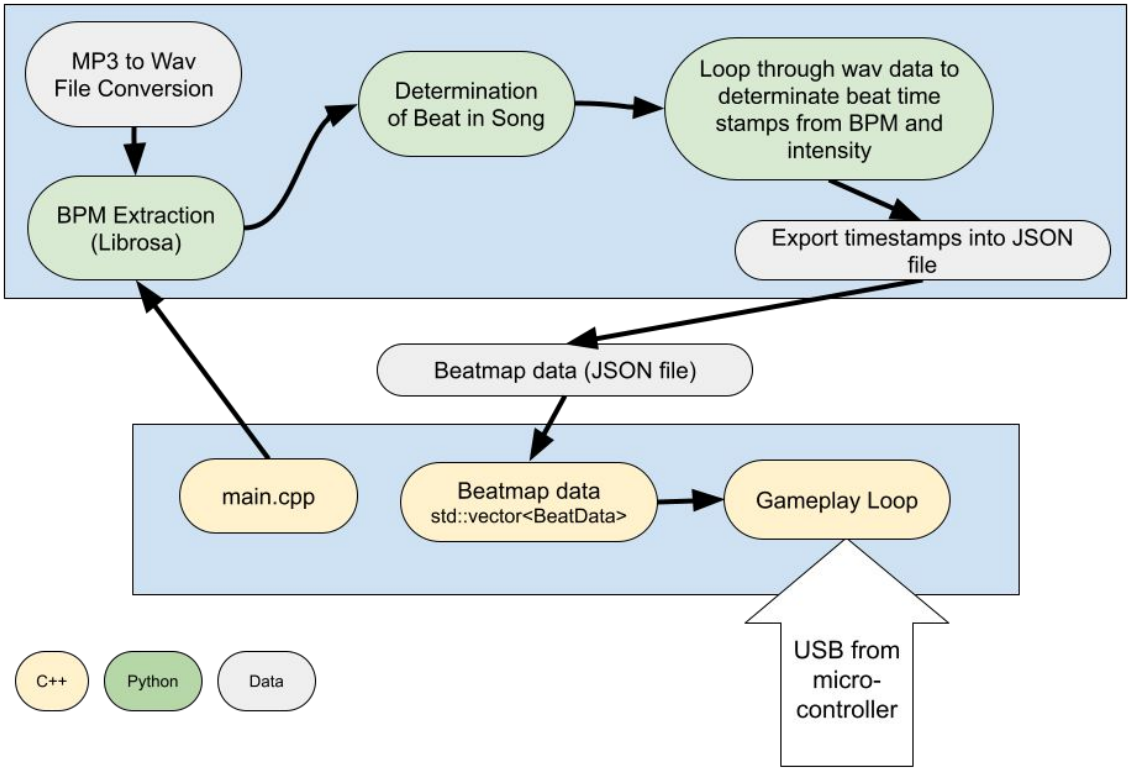
C++ - Gameplay Code

- OpenGL for game graphics
- SoLoud for game sound
- At start of game, main.cpp will execute the Beat Tracking Python code

Block Diagrams- Hardware



Block Diagrams- Software



Implementation Plans

Hardware

- Piezoresistive Sensor Drum
  - This module is taken from the FSR's datasheet and purchases through commercial sources.
- ESP32 Development Board
  - The ESP32 development board combines several features into one simple/affordable board.

Software

- Beat Tracking Algorithm - Python
  - Use of package Librosa to extract BPM from wav file
  - Use of numpy and scipy to transform sampling rate into time stamps
- Gameplay Code - C++
  - Use OpenGL for displaying graphics
    - GLFW, glad, glm
  - Use SoLoud for playing sounds
  - Harfbuzz + freetype for font rendering

Bill of Materials

Hit It! Hardware Testing Supplies (Group A5 Round 1 BOM)

Distributer	Description	Distributer URL	Unit Price	Shipping + Tax	Total Price	Shipping Notes
Sparkfun	Piezoresistive force sensor	https://www.sparkfun.com	$19.95	$13.68	$33.63	2 day shipping option selected
Arrow Electronics	Op Amp Quad Low Power Amplifier F	https://www.arrow.com/en	$0.63	$6.63	$9.80	
Amazon	Cheap backup teasting piezo transdu	https://www.amazon.com/D	$8.99	$0.00	$8.99	Free with prime
Amazon	Neoprene rubber sheets for drum pad	https://www.amazon.com/1	$9.99	$0.00	$9.99	Free with prime
Amazon	ESP32 development board	https://www.amazon.com/H	$10.99	$0.00	$10.99	Free with prime
					$73.40	

Remaining Budget: $600 - $73.40 = $526.60

Testing, Verification, and Metrics - Hardware

Area	Metric(s)	Test Inputs/Outputs
Latency	37 ms (Overall for Hardware)	Inputs: Force to the FSR Outputs: Oscilloscope Voltages with timestamps
Portability/Compactness	15,000 cm^3 (Overall)	Inputs: Module sizes Outputs: Physical Dimensions (Measurements) and Practical Test Results (Backpack)
Drum Recognition Accuracy	>99% Recognition Rate	Inputs: Repeated force application to the FSR Outputs: Recognition in Software
Ease of Setup	< 1 minute Setup Time (Overall)	Inputs: Random Participants Outputs: Recorded Setup Times

Testing and Verification - Software

Area	Metric(s)	Test/Inputs/Outputs
Beat Tracking Timing	Time for Creation of Beat Mapping is Length of Audio File (at most)	Use Python to track code execution time Input: Code File Output: Timings
Beat Tracking Accuracy	Generated beatmaps are more than 80% accurate to testing/sample beatmaps	Beat overlap and Beat Closeness Inputs: Created Time Stamps for Map, Auto-Generated Time Stamps for Map Outputs: Percentage of time stamps where some overlap existed
Aesthetics (GUI)	> 30 FPS. Non-geometric graphics	Add timer to gameplay render loop, find average FPS Inputs: Code File Outputs: FPS

Risk Mitigation + Unknowns

Hardware:

- Latency:
  - Mitigation: Hardware selection
  - Fallback: Component replacement, code optimization, backup 100ms goal
- Compactness:
  - Mitigation: Goal > Expectation
  - Fallback: Module redesign, ESP32 wireless features
- Drum Recognition:
  - Mitigation: Hardware selection
  - Fallback: Commercially available buttons
- Ease of Setup:
  - Mitigation: Latch connectors
  - Fallback: ESP32 wireless features

Software:

- False Positives/Negatives
  - Mitigation: Use of BPM to limit false readings
  - Fallback: Incorporation of false readings into beat map
- Communication error with microcontroller
  - Mitigation: test driven development - run test cases for communication protocol
  - Fallback: drop communication if error is detected

Schedule & Division of Labor