S.T.E.P.S. - Style Tracking Expressive Pad System

Group 8: Christopher Solanilla, Jani John Lumibao, Anderes Abrams, Kaila Peeples

Department of Electrical and Computer Engineering, and College of Optics and Photonics, University of Central Florida, Orlando, Florida, 32816

***Abstract* - This paper presents S.T.E.P.S. (Style Tracking Expressive Pad System), a nine-panel rhythm game platform that fuses traditional timing-based foot input with real-time, camera-based pose evaluation. The system integrates custom modular hardware, a computer-vision subsystem, and a game engine that computes both accuracy and a *Style Score* from predefined full-body poses. We detail the mechanical, electrical, optical, and software design choices; discuss communication and power architectures; and report integration results and lessons learned. The prototype demonstrates low-latency sensing and robust tracking under varied lighting, offering an extensible approach for interactive entertainment, exercise-gaming, and overall human-computer interaction.**

I. INTRODUCTION

Commercial rhythm games such as DDR, Pump It Up, and StepManiaX popularized timing-based foot input on four to five panels. S.T.E.P.S. expands this paradigm with a nine-panel pad (cardinals, diagonals, and center) and introduces a Style Score that rewards expressive, full-body movement during designated moments. This dual-metric design targets two goals: 1. preserve the "hit on time" skill loop that competitive players expect, and 2. explicitly value unique gestures with the upper body so casual and performance-oriented players feel recognized.

Achieving this required cross-disciplinary engineering: embedded sensing for reliable foot detection, a modular electronics architecture to simplify wiring and debugging; a vision stack tolerant of variable lighting, and a game engine that synchronizes chart timing, HID input, and pose prompts.

II. SYSTEM COMPONENTS

The system is best understood by examining its primary hardware components and how they interact to form the complete design. This section provides a semi-technical overview of each module, outlining their functions and roles within the overall system. These components establish the foundation for data acquisition, processing, and communication across the architecture of the design.

A. Microcontrollers

Our design uses an ATmega32U4 microcontroller for the Master PCB and nine ATmega328P microcontrollers for the Input Tile PCBs. These chips were chosen because they're reliable, power-efficient, and easy to program using the Arduino IDE. Both provide the necessary processing performance and memory for our system, while also supporting USB communication for smooth data transfer between the dance pad and the computer.

B. Dance Pad Sensors

Each input tile uses a Force-Sensing Resistor (FSR) to detect when a player steps on it. We selected the Interlink [2] Electronics FSR 408 model because it's durable, affordable, and provides consistent sensitivity. The FSRs convert foot pressure into analog signals that are read by the microcontrollers, allowing the system to accurately capture and respond to each player's movements.

C. Communication Protocol

I^{2}C communication was selected for its efficiency and scalability, allowing all nine Tile Boards to interface with the Master Controller through only two data lines and power connections. This minimizes wiring complexity, conserves I/O pins, and maintains sufficient data speed for real-time sensor polling and LED control, making it the most suitable protocol for the system's modular architecture.

D. Power Supply

A 12V DC, 7A external switch-mode power supply (SMPS) was chosen to provide stable and efficient power delivery across the nine-tile system. The 12 V output meets the LED voltage requirements, while the 7 A rating offers sufficient overhead beyond the calculated 6 A load. The SMPS architecture ensures high efficiency and tight voltage regulation for both high-current lighting and low-power microcontrollers. Electrical noise is effectively mitigated through onboard filtering and decoupling networks integrated into the Power Hub and Master Controller PCBs. This configuration delivers a safe, reliable, and compact power solution that supports the modular design and overall system performance.

E. Voltage Regulator

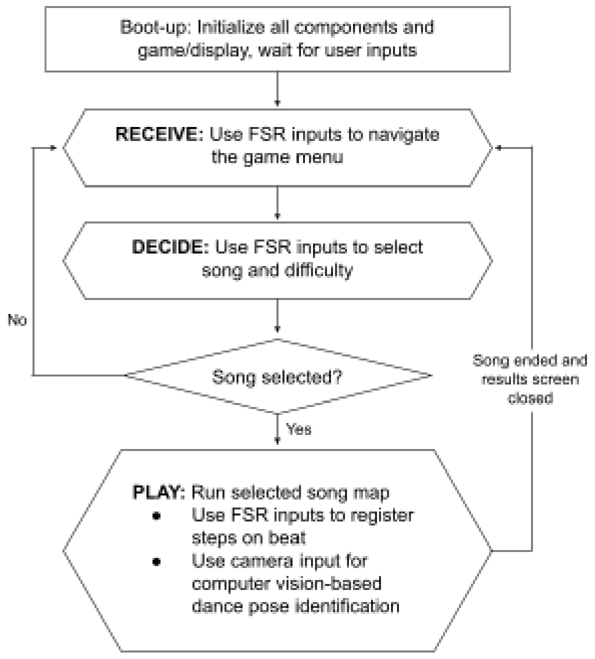
The system's power subsystem is centered around a TPS56628, a 4.5V-18V, 6A synchronous buck converter. This regulator steps down the 12V input from an external power adapter to a stable 5V rail, which powers both the main ATmega32U4 controller and a 3x3 grid of nine ATmega328PB tiles. To mitigate voltage drop and manage the cumulative current load across the grid, the 5V supply is split into three distinct power lines. Each line is dedicated to servicing one column of the 3x3 PCB array, ensuring all tiles receive a stable operating voltage.

F. NIR LED Control

Control of the Near-Infrared (NIR) LED array is achieved using a low-side N-channel MOSFET switch (IRLR7843TRPBF). This logic-level MOSFET is driven directly by a PWM signal from the microcontroller. The 5V signal from the MCU pin fully exceeds the MOSFET's 2.3V threshold, turning it completely on and allowing current to flow through the LEDs. This PWM-based switching enables fine-grained brightness control and significantly reduces overall power consumption[5].

III. SYSTEM CONCEPT

To understand the complete system, a system flowchart would be helpful.



Boot-up: Initialize all components and game/display, wait for user inputs
RECEIVE: Use FSR inputs to navigate the game menu
DECIDE: Use FSR inputs to select song and difficulty
No
Song selected?
Song ended and results screen closed
Yes
PLAY: Run selected song map
Use FSR inputs to register steps on beat
Use camera input for computer vision-based dance pose identification

Fig. 1. Overall system flowchart showing the three main operational states (Receive, Decide, and Play) and how user inputs progress through each stage of the dance pad system.

In the *receive* portion of the cycle, inputs from the FSRs are detected and processed by the Master Controller. This occurs at two possible times: when the system is idle on the game menu, or when a song is actively being played. During the idle state, FSR inputs are primarily used for navigating through the menu options or selecting game modes. However, during gameplay, FSR inputs function as step detections on beat with the music. The receive state therefore serves as both an input detection phase and an interrupt listener, depending on the current stage of operation. When the system is idle, FSR readings are continuously monitored until a valid input is registered, prompting a transition to the *decide* state. During active gameplay, the same inputs are instead passed directly to the *play* state logic, where they are compared to the timing data of the selected song map.

In the *decide* portion of the cycle, the system interprets the FSR inputs received during the menu phase to determine which song and difficulty level the player has selected. Once a valid selection is confirmed, the Master Controller communicates the corresponding song data and configuration to the computer running the game software. This transition is notated by the “Song selected?” decision diamond on the flowchart. If no song selection is made, the system remains in the *receive* state, waiting for additional input. Upon a valid selection, the system advances to the play state, where the actual gameplay is initiated. The *decide* state is therefore responsible for verifying input, determining gameplay parameters, and establishing the transition from idle interaction to active performance.

In the *play* portion of the cycle, the selected song map is executed, and the system actively monitors FSR inputs along with camera-based computer vision data. The FSR inputs are used to detect step accuracy in real time, while the external camera sends positional data for pose tracking and movement analysis. These two input streams are processed together to evaluate player performance throughout the song. The play state is the longest and most dynamic portion of the cycle, functioning as both a continuous input-processing state and a monitoring state for end-of-song events. When the song concludes, the system transitions back to the receive state for menu navigation and result viewing, completing the full operational loop of the dance pad system.

A. System Hardware Concept

The following block diagrams present each primary hardware module, highlighting the data and control flow between them.

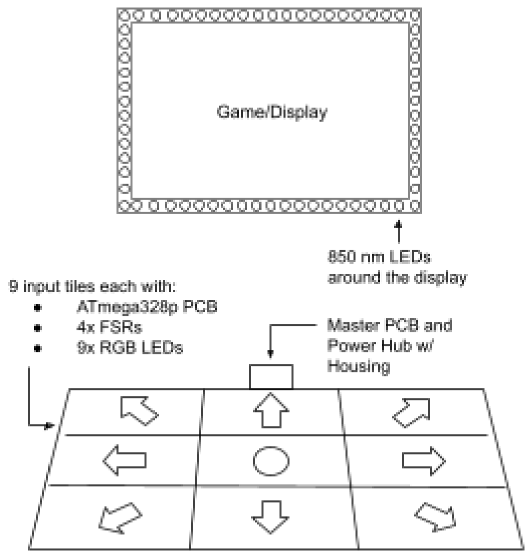


Fig. 2. High level hardware visual of overall design

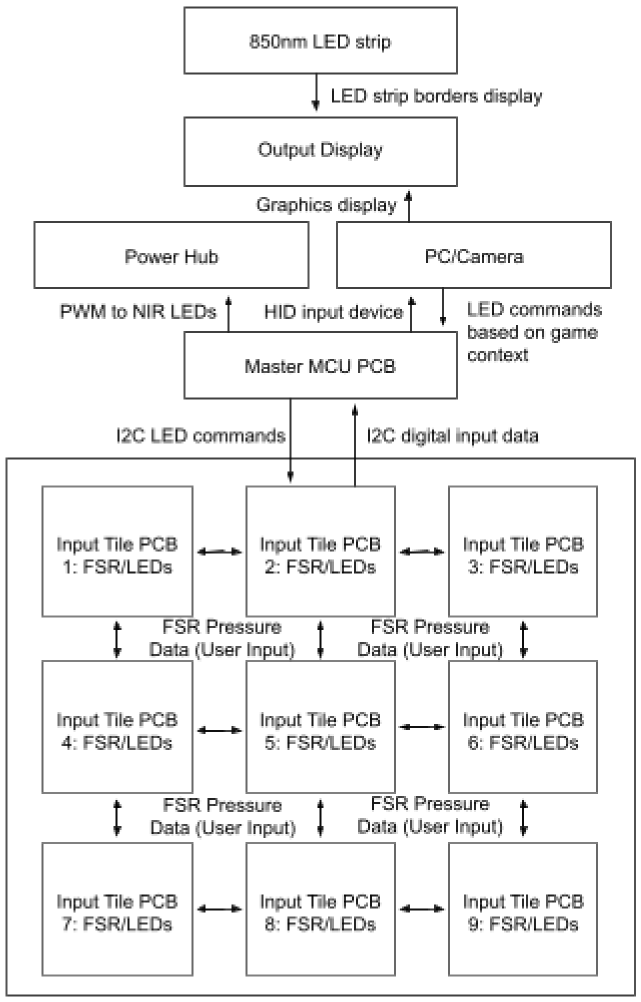

850nm LED strip
LED strip borders display
Output Display
Graphics display
Power Hub
PC/Camera
PWM to NIR LEDs
HID input device
LED commands based on game context
Master MCU PCB
I2C LED commands
I2C digital input data
Input Tile PCB 1: FSR/LEDs
Input Tile PCB 2: FSR/LEDs
Input Tile PCB 3: FSR/LEDs
FSR Pressure Data (User Input)
FSR Pressure Data (User Input)
Input Tile PCB 4: FSR/LEDs
Input Tile PCB 5: FSR/LEDs
Input Tile PCB 6: FSR/LEDs
FSR Pressure Data (User Input)
FSR Pressure Data (User Input)
Input Tile PCB 7: FSR/LEDs
Input Tile PCB 8: FSR/LEDs
Input Tile PCB 9: FSR/LEDs

Fig. 3. Hardware block diagram of the design's major system components, shown in a data I/O flowchart

IV. HARDWARE DETAIL

While all hardware components have been outlined and briefly explained in the past sections, the following section explains the hardware components of the design in technical detail, with the exception of the power hub, as it was already explained in enough detail in Section II.

A. Hardware Components

1) ATmega32u4 Microcontroller (Master)

Upon researching for the most reliable master microcontroller unit, we ultimately decided on using the ATmega32u4 chip, which is commonly found in Arduino Leonardo development boards. The ATmega32u4 natively supports USB HID[9], which allows it to act as a keyboard or joystick, extremely crucial for our game's real-time step detection. The ATmega32u4 has just enough analog inputs for our design by the way we have it wired out. Overall, it has the right balance between functionality, ease of use, and affordability, making it a practical and reliable core controller for our dance pad system, especially for single-player setups with limited hardware demands.

2) ATmega328pb Microcontroller (Slave Input Tile)

For our nine slave input tiles, we needed a microcontroller chip that could do the bare minimum as it would only be responsible for LED functionality and connecting to the FSRs. Additionally, to stay within our budget, we needed an affordable MCU. As a result, we decided on the ATmega328pb chip, which is commonly found in Arduino Uno development boards. The ATmega328pb chip is very compatible with ATmega32u4, so communication between the two through Arduino IDE is very simple to work with. In terms of design, each of our dance pad's input tiles will have four FSRs connected to them and four I^2C connection headers[4] to be able to connect with other input tiles. ATmega328pb can satisfy all design requirements while being very cost-efficient.

TABLE I

KEY FEATURES OF THE DESIGN'S MCUS

Feature	ATmega32u4	ATmega328pb
Clock Speed	16 MHz	16 MHz
Flash Memory	32 KB	32 KB
PWM Channels	7	6
ADC Inputs	12	6
USB Comm.	native USB	serial-to-USB
USB HID support	included	none
Cost	~$4	~$3

3) Force Sensing Resistors (FSRs)

The FSR Model 408 from Interlink Electronics was selected for its reliability, consistency, and ease of integration. Its 300 mm flexible strip design provides a large sensing area with uniform pressure response, ensuring accurate detection across all tiles. The sensor's pre-calibrated nature eliminates the need for manual tuning and minimizes issues such as drift or dead zones that can affect gameplay precision. Its slim form factor allows seamless installation beneath each pad surface, while its durable construction withstands repeated impacts over time. Overall, the Model 408 delivers the stability, responsiveness, and plug-and-play simplicity needed for a performance-grade dance pad system.

B. Hardware Testing

One of our key engineering specifications is the dance pad's input response time, therefore, we had to base our hardware testing on the overall speed it takes for the player to physically press on an input tile to the moment the game registers the press as input. While we have tested and found that the raw response time of the FSRs in communication with the game is almost instantaneous, we can't assume for sure the true speed until the overall system is established, which at this moment, it's not. However, once the dance pad system is fully integrated, we will then be able to demonstrate, with results, the dance pad's true input response time.

V. OPTICAL SYSTEM DETAIL

The optical subsystem of our project is what provides all the visual input needed for MediaPipe[1] to track the player's body in real time. It Includes an 850 nm near-infrared (NIR) illumination setup, a global-shutter RGB camera[3], and a wide angle aspherical M12 lens. The goal was to create an imaging setup that could provide consistent illumination and resolution across a 1.8m x 1.8 m play area.

After testing in several environments, it was clear that MediaPipe only requires around 40% illumination uniformity for accurate tracking. In bright spaces like the Senior Design Lab, the existing overhead lighting already provides enough visible light for the camera, so the additional NIR illumination isn't even necessary. The LEDs mainly help in darker rooms where ambient light is limited.

A. Optical Components

1) Imaging Lens

A 3.2mm f/2.3 M12 aspherical lens (CIL034-F2.3-M12ANIR) is being used because it gives a 97° diagonal field of view and has the ability to capture the 850 nm illumination well due to not having a IR cut filter. This lens, combined with the 1/ 2.6'' AR0234 sensor, covers the 1.8m x 1.8m play area at a 1.8m working distance and provides about 0.6 pixels per millimeter (≈1.7 mm pixel^{-1}) across the pad. While that isn't enough to resolve 1mm-sized features, it's more than enough for full-body pose tracking since joints and limbs span many pixels. MediaPipe consistently reached ≥ 95 % tracking accuracy with this configuration, so the 3.2 mm lens was kept for its wide coverage and simplicity. Depth-of-field analysis using a 0.005 mm circle of confusion confirmed that subjects between roughly 1.5m and 2.1 m stay in focus.

2) Camera Module

The SVPRO AR0234 global-shutter camera runs up to 120 fps @ 720 p with 3.45 µm pixels and strong near-infrared sensitivity. The global shutter helps prevent motion distortion during fast movements, which is critical for a dance-based system. The USB 3.0 interface enables low-latency data transfer between the camera and processing unit, and the camera integrates cleanly with the illumination and lens system to provide stable input for tracking.

3) Illumination System

The illuminations system uses one 116-inch, ~42W 850 nm LED strip mounted around the perimeter of the monitor frame to directly face the player area. The strips are powered by a 12 V constant-voltage supply and remain continuously on during tracking. Earlier tests were done using aluminum channels with diffusers, but the diffusers resulted in reduced brightness and lower measured illumination uniformity.The effect was likely caused by insufficient mixing distance and wavelength-dependent scattering, which caused uneven attenuation of the 850nm emission. Removing them produced a more uniform and higher-intensity NIR distribution across the play area. In bright rooms, the ambient lighting alone is sufficient for MediaPipe tracking, so the NIR LEDs primarily serve as supplemental illumination for darker testing environments. Figure 4 demonstrates that in the controlled environment, the illumination system achieves the 90% uniformity spec.

B. Optical Testing

Illumination uniformity was measured in MATLAB using the min-to-mean intensity ratio across a defined region of interest. In controlled dark-room testing, the system reached about 91 % uniformity. In more open or reflective environments, results ranged from 40% to 75% depending on wall color, ceiling height, and background reflections. Camera testing was performed alongside these trials to confirm that exposure, contrast, and frame stability were maintained across all lighting conditions. Even at the lower end of illumination uniformity, MediaPipe consistently maintained full tracking accuracy, confirming that roughly 40% uniformity is sufficient for proper operation.

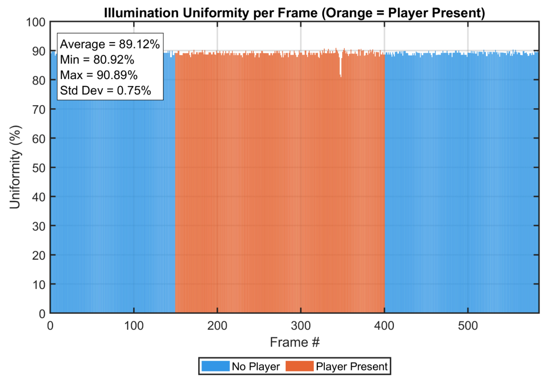


Fig. 4. Frame-by-frame illumination uniformity measurement. The orange region indicates frames where a player was present. Uniformity remained between 80 % and 91 % throughout testing, confirming stable illumination during motion.

VI. SOFTWARE DETAIL

The game engine handles song selection, chart parsing, scroll timing, note judgments, combo/scoring logic, and UI overlays. The HID input path abstracts pad events as buttons, avoiding custom kernel drivers.

A. Software Components

1) Godot Game Engine

Inside the game engine of choice Godot, we have implemented multiple different menus that users can navigate throughout using the dancepads. Inside of the settings menu for example, changing the in game volume and the speed at which the notes fall in the game, is seamless by being able to hold down the dance pad arrows and change the slider.

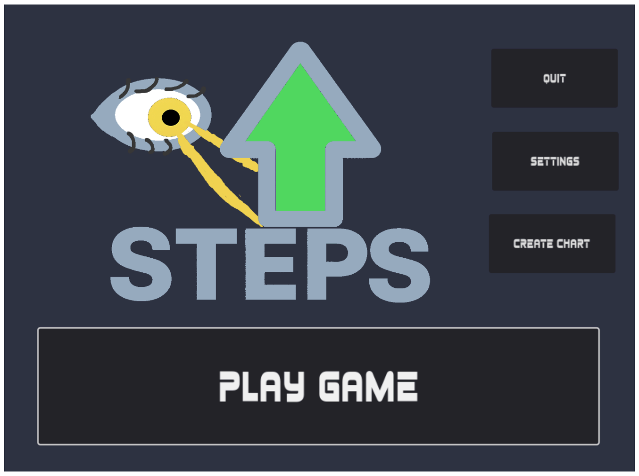
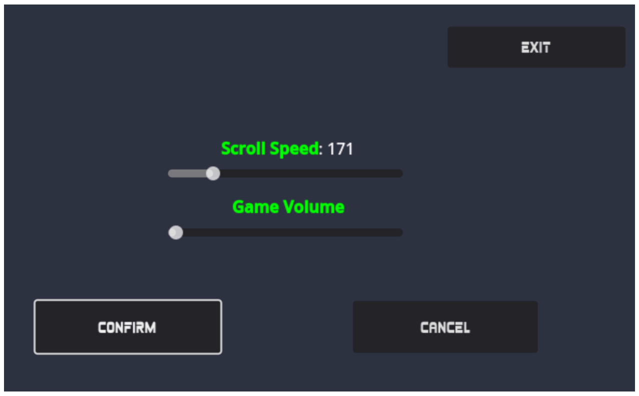


Fig. 5. Image of Main Menu and Start Screen of S.T.E.P.S

To start off our game, we are met with our main menu (as shown in figure 5 correctly reference it). Users have the ability to play the game, switch to the settings menu to change their game volume and the speed at which the notes are able to fall at, and lastly be able to create their own charts to play.



Scroll Speed: 171

Fig. 6. Image of the settings menu during the running of the game

Moving on to the settings menu, we are met with this UI. As one can see, this is a very simple settings menu where one can change the game volume and change the speed of the notes as mentioned before. Using additional coding logistics and manipulation of built in properties of assets, one is able to cancel the changes that they have made and go back to the defaulted settings, they can confirm their changes to which the settings will be saved globally and taken back to the main menu screen, or they are able to exit the settings menu when they would like and return back to the main menu screen.

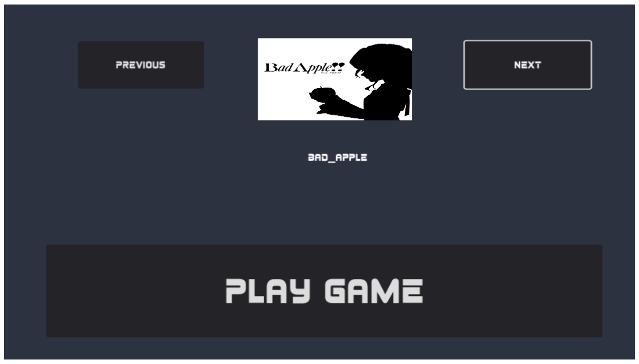


Fig. 7. Image of the song selection menu during the running of the game

Lastly before playing the chart, is our song selection menu where users are able to select their song of choice and play the map related to it. Using a list of songs and list of images, exists a linked list that will wrap around to the beginning once it reaches the end of the list.

2) Computer Vision Model

A lightweight RGB pose stack runs at camera frame rate; the engine computes angular deltas vs. per-pose acceptance bands and gates the Style award on temporal alignment with the music beat window. To improve reliability, we restrict recognition to a curated pose set rather than unconstrained freestyle.

Figure 8 shows 6 possible poses that the player can be prompted to perform while in game. Running on a separate thread, there is a process constantly taking in input from the camera and attempting to map 32 landmarks to represent the bones on the player[7-8]. These landmarks include dots mapped onto the left and right wrists, shoulders, hips, and head. In each frame it is possible to get the coordinates of these landmarks which is particularly useful in relation to each of the other landmarks.

Figure 9 shows using landmarks in action to identify the Muscle Man pose specifically in testing. Identifying the different poses we take a tailored approach for each pose. For example, when it comes to the Muscle Man pose the main concern is the placement of both the wrists and elbows being above the shoulders, while also having the elbow bent at a certain angle range. Looking at figure 9, we can see an example of a code block that would determine if our pose would be a Muscle Man pose.

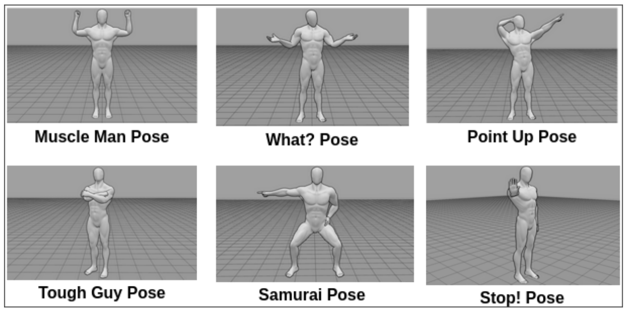


Fig. 8. Table of 6 different poses available in the game. Art made with PoseMy.Art

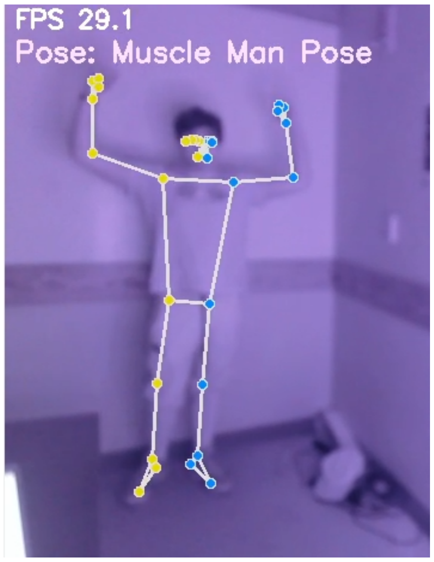
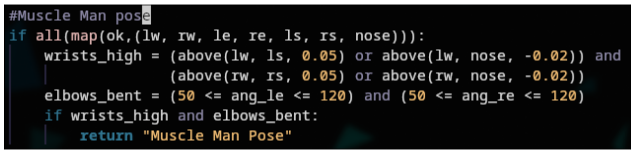


Fig. 9. Screenshot of running MediaPipe with debugging tools to demonstrate how using landmarks can accurately and efficiently identify poses.



```
#Muscle Man pose
if all(map(ok,(lw, rw, le, re, ls, rs, nose))):
    wrists_high = (above(lw, ls, 0.05) or above(lw, nose, -0.02)) and
                  (above(rw, rs, 0.05) or above(rw, nose, -0.02))
    elbows_bent = (50 <= ang_le <= 120) and (50 <= ang_re <= 120)
    if wrists_high and elbows_bent:
        return "Muscle Man Pose"
```

Fig. 10. Code block showcasing how one can easily determine a specific pose like the Muscle Man pose by calculating the distances and angles between landmarks.

3) UDP Sender

Our vision code runs outside of the main game loop, so we needed a fast and simple way to send pose data from the Python process to the game engine in Godot. Because both programs run on the same computer, we didn't need a complicated networking setup, just something lightweight that could move small packets of data between processes almost instantly. For this reason, we chose to use a local UDP socket on 127.0.0.1:54545. The "local" part means the data never actually leaves the machine or travels through any external network. It stays entirely within the operating system's memory, so the transfer time is effectively negligible. Figure 11 shows a clear visual understanding of the process on the high-level overview on how the vision process communicates with the game process.

We specifically went with UDP instead of TCP because of how the two protocols handle data. TCP guarantees that every packet arrives in order, but it does that by adding a lot of overhead and waiting for acknowledgments whenever something goes missing. That may be suited for something like web pages or file downloads, but considering our purposes it is not needed for real-time live camera feed pose deduction. In our case, we're sending updates 60 to 120 times per second, and losing one or two frames isn't a problem. In fact, the next one will be along in less than 20 milliseconds.

Using TCP would have introduced extra latency and possibly caused the game to stutter if even a single packet were delayed or had to be resent. UDP, on the other hand, is connectionless and doesn't care about perfect reliability. It just sends each packet as fast as possible and moves on.

That makes UDP perfect for this type of one-way, high-frequency data stream. We only need the most recent pose at any given moment, not a history of every pose since the start of the song. If a packet gets dropped, the game simply continues using the last one until a new one arrives a fraction of a second later. Because both processes are local, packet loss is extremely rare anyway, so we get the benefit of ultra-low latency without worrying about missed data.

B. Software Diagrams

1) Menu Diagram

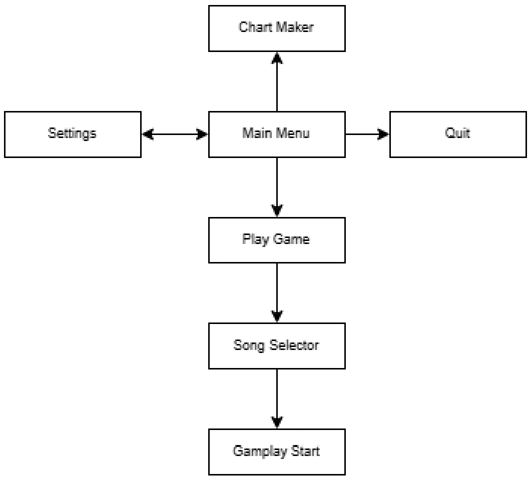


Fig. 11. Menu flow diagram

2) Gameplay Diagram

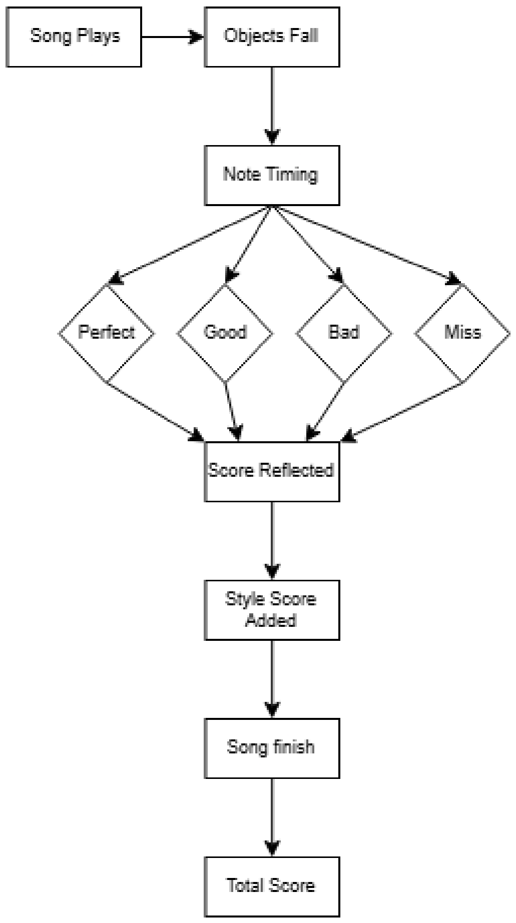


Fig. 12. Diagram of how the game operates

The gameplay diagram starts off with the song starting, to which objects fall to the beat of the song. Afterwards when users attempt to "hit" the note at the correct time, it is recorded and the score will reflect said score value. One example is a "Perfect" timing has a value of 10 while a "Miss" timing has a value of 0. During this time additionally the style score is added using computer vision calculations when pose prompts appear. This will happen continuously until the song ends to which the total score will be shown after all calculations have been conducted.

3) Computer Vision Diagrams

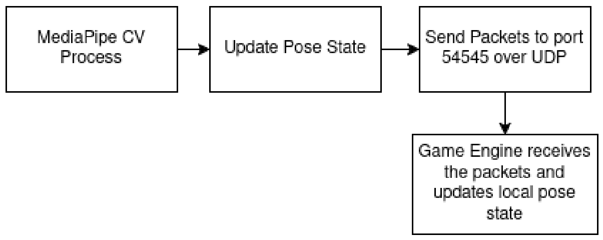

MediaPipe CV Process
Update Pose State
Send Packets to port 54545 over UDP
Game Engine receives the packets and updates local pose state

Fig. 13. Software Flow chart for how the game receives information from the computer vision sub-system.

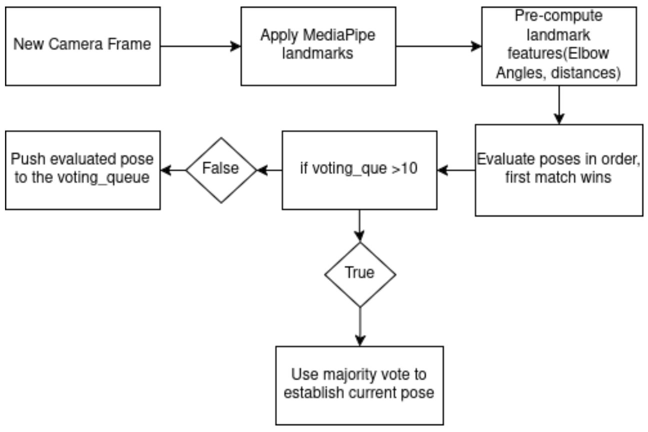

New Camera Frame
Apply MediaPipe landmarks
Pre-compute landmark features(Elbow Angles, distances)
Evaluate poses in order, first match wins
if voting_que >10
False
Push evaluated pose to the voting_queue
True
Use majority vote to establish current pose

Fig. 14. Software Flow chart for how frames are interpreted and processed to evaluate the player's pose.

C. Software Testing

Our software testing centered on making the pose detection feel natural and reliable in real play, not just in unit tests. Traditional function-level checks are useful, but computer-vision behavior depends on people, lighting, and motion, so we emphasized iterative, human-in-the-loop evaluation.

We started with a rough pose classifier and then ran short test rounds with multiple participants. In each round, every participant performed all six poses ten times in sequence. For each attempt we logged whether the system produced the correct stable label within the acceptance window. After each round we adjusted thresholds, such as elbow angles, wrist-to-shoulder distances, the torso-scale factor, and the stability parameters (window size and cooldown), then repeated the protocol.

This process of testing into tweaking into re-testing loop continued until the classifier consistently matched how the poses should feel in game. Across multiple participants and poses, the final round reached 98.33% pose detection accuracy with some remaining errors concentrated in fast transitions between visually similar poses.

For the game itself, software testing focused mainly on ensuring smooth timing, reliable input, and stable gameplay. Rhythm games rely on precise synchronization, so even the smallest timing errors can break the experience. Testing involved verifying that the notes in each map synchronized with the beat of the music. To further prevent these kinds of issues, we introduced the ability to set a toggleable offset. Additional tests involved rapid input sequences and long play sessions to confirm consistent HID response without dropped inputs or frame stutters. Menu navigation and song selection were also tested to verify correct wrapping behavior and state transitions.

VII. CONCLUSION

S.T.E.P.S. demonstrates that hybrid scoring with timing accuracy and pose-based style can broaden rhythm-game appeal while remaining technically tractable on student-designed hardware. A modular three-board architecture, FSR-based tiles, an RGB global-shutter vision stack, and an HID-centric software path together yielded a responsive, engaging prototype. We hope this blueprint accelerates future teams building expressive, vision-aware game systems.

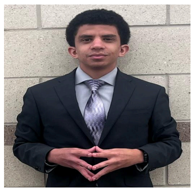
ACKNOWLEDGEMENT

The authors of the paper would like to acknowledge Kyle Ward, creator of StepManiaX[6], for his assistance in giving us insight as a successful person in the field. He has given us permission to show off his works for our class as examples and has given us useful advice on how to succeed.

BIOGRAPHY

Kaila Peeples is a photonic Science and engineering student at the University of Central Florida, with a minor in Materials Science and Engineering. She will receive her Bachelor of Science degree in December 2025. Kaila will pursue a career in applied optics and photonics, focusing on technologies that advance imaging, sensing, and other practical applications with positive real-world impact.

Jani Jon Lumibao is a Computer Engineering student at the University of Central Florida. He will receive his Bachelor of Science in Computer Engineering degree in December 2025. Jani Jon continues to look for ways to improve his skills in embedded systems, game design, and web development for a promising career in the future.



Andres Abrams is a Computer Engineering student at the University of Central Florida. After completing his Bachelors in Computer Engineering, he plans to further pursue his career in software development, additionally he has also developed an interest in Real Time Systems, embedded systems, and other hardware focused areas of development.

Christopher Solanilla is a Computer Engineering student at the University of Central Florida. He is the group member that came up with the idea of making an arcade machine due to being passionate about rhythm games in his offtime. While pursuing Computer Engineering, he has had multiple opportunities with game development and computer vision leading him to attempt to combine the two disciplines into an ambitious project. In the future, he plans to work in industry with an eventual goal to create his own software business.

Blake Whitaker is an Electrical Engineering student at the University of Central Florida. He serves as the hardware lead for this project, where he focuses on circuit design, PCB layout, and system integration. After graduation in December 2025 he plans to take the Fundamentals of Engineering (FE/EIT) exam in December 2025 as a step toward professional licensure while also pursuing a career in electrical engineering.

REFERENCES

[1] Google Research, "MediaPipe: Cross-platform, customizable ML solutions for live and streaming media."

[2] Interlink Electronics, "FSR® Force Sensing Resistors — Model 408 Datasheet."

[3] onsemi, "AR0234: 1/2.6-inch 2.3 MP Global Shutter Image Sensor."

[4] NXP, "I²C-bus specification and user manual."

[5] IEC 62471, "Photobiological safety of lamps and lamp systems."

[6] Step Revolution. (2024). StepManiaX. Kyle Ward

[7] Google Research. (2020). BlazePose: On-device, Real-time Body Pose Tracking

[8] Google AI. MediaPipe Pose Landmarker lightweight, real-time 33-landmark body pose detection.

[9] USB Implementers Forum, Device Class Definition for Human Interface Devices (HID)