

## **Project Scope**

Amateur grappling (wrestling, jiu-jitsu, and judo) competitions currently require 1-3 individuals to staff each mat in addition to the referee in order to operate the scoreboard, upload results, and notify the referee when time has expired. This introduces unnecessary expenses on the part of the event organizer. Additionally, communication issues between referees and table staff can hurt the integrity of competitions, in some cases even resulting in the wrong athlete being declared the winner.

We've developed a prototype for a wearable, ESP32 and Raspberry Pi based remote control system that enables a referee to operate the scoreboard, upload match results, and receive haptic feedback when time expires without the need for any additional staff and without impeding the referee from performing his or her officiating duties in any way.

The RefRemote system consists of three main components: a pair of ESP32 based wrist-wearable controllers, a Raspberry Pi based scoreboard program, and a Raspberry Pi based tournament manager program. The system architecture is designed such that any number of mats (each with a pair of controllers and scoreboard) can be run in parallel and managed by a single tournament manager node.

Each of the ESP32 based wrist wearable controllers delivers the following features:

- Intuitive three button layout for incrementing scores and controlling clock
- Haptic Feedback
- Battery Status Reading
- Gesture recognition mode for starting clock hands-free
- LED status indicators
- Interfaces with scoreboard node over BLE

Each Raspberry Pi based scoreboard node delivers the following features:

- Interfaces with controllers over BLE
- Displays match status details:
  - Score
  - Time
  - Period
  - Athlete Names
- Interfaces with tournament manager node over MQTT to upload results and retrieve matches from queue

The Raspberry Pi based tournament manager program delivers the following features:

- Interfaces with scoreboards over MQTT
- Allows user to initialize bracket with competitor names
- Builds match queue and assigns matches to mats.
- Receives results and updates bracket status and match queue

**Table 1: Bill of Materials**

| Component                                                                   | Quantity |
|-----------------------------------------------------------------------------|----------|
| ESP32-C3-Supermini Dev Board                                                | 2x       |
| Raspberry Pi (Desktop Configuration including keyboard, mouse, and monitor) | 2x       |
| 3.7V 250mAh LiPo Batteries                                                  | 2x       |
| ERM Haptic Motors                                                           | 2x       |
| LIS3DH 3-Axis Accelerometer Modules                                         | 2x       |
| Max17043 Fuel Gauge Modules                                                 | 2x       |
| Tactile push buttons                                                        | 6x       |
| RGB LED                                                                     | 2x       |
| White LED                                                                   | 2x       |
| Various Resistors                                                           | -        |

## System Architecture

The referee wears a red and green sweatband on their right and left wrist respectively with an insert containing an ESP32 and peripherals. The ESP32s interface with a Raspberry Pi scoreboard controller over BLE(Bluetooth Low Energy). The Raspberry Pi scoreboard node uploads match results to a second Raspberry Pi event manager used for data aggregation, bracket management, and match queueing over MQTT(Message Queuing Telemetry Transport). This device's software will be implemented such that it is able to scale up to interface with many mats in parallel, even if only a single mat is used in preliminary demos. *Figure 1* depicts a graphical representation of the system architecture.

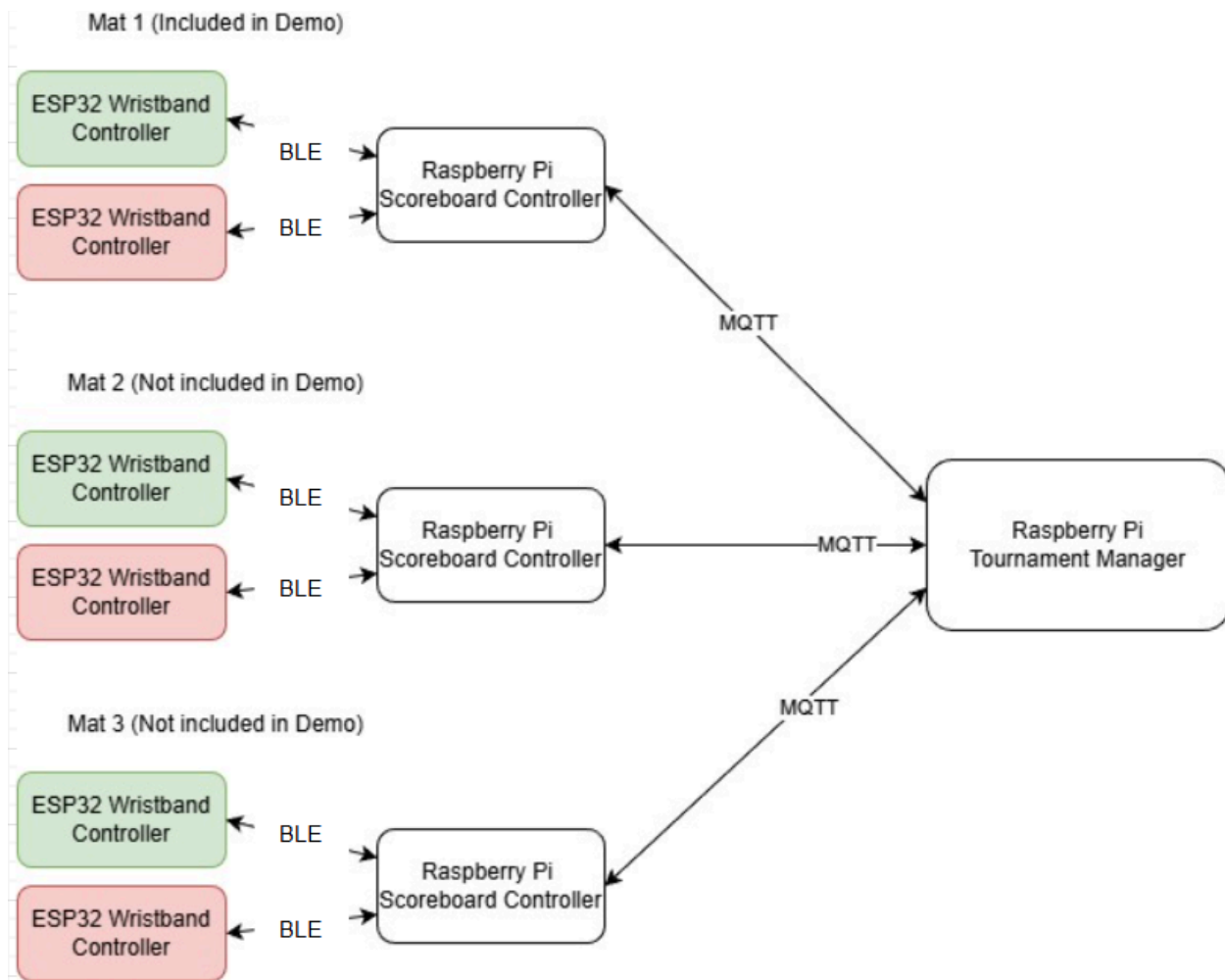


**Figure 1: System Architecture Diagram**

The RefRemote system leverages the MQTT topic structure to organize match assignments and results. The general topic structure used by the system can be summarized as:

- tournament\_name/bracket/match\_number/mat/action

An example of the topic structure used when assigning a match from the match queue to an available mat:

- capital\_classic/160lbs/match\_8/mat\_2/queue

An example of the topic structure used when uploading results from a completed match back to the tournament manager node:

- capital\_classic/160lbs/match\_8/mat\_2/result

According to MQTT conventions for subscribing to multiple topics, each scoreboard subscribes to brackets and matches queued up on its mat:

- capital\_classic/###/mat\_2/queue

The tournament manager node subscribes to all published mat results for the relevant tournament:

- capital\_classic/###/###/result

QOS level 3 is used for all match assignments and uploads of results to ensure successful transmission without duplicate uploads.

## Controller Design

The ESP32 controllers feature three sensor input types: button inputs with integrated debounce logic to handle timer and scoreboard control inputs, an accelerometer used for ML(Machine Learning) gesture recognition, and a fuel gauge circuit to read battery status. Actuator outputs on the ESP32 controller include: an RGB LED battery life indicator, a white LED indicating whether gesture listener mode is activated, and a haptic motor to notify the referee when time expires. The Raspberry Pi scoreboard controller also acts as an actuator as it displays score, time, period, and other relevant match parameters. *Figure 2* depicts a functional block diagram of the ESP32 controller design including pin level interface connections with peripherals and power pin connections, although note the diagram is not an exhaustive schematic of electronic components and connections.

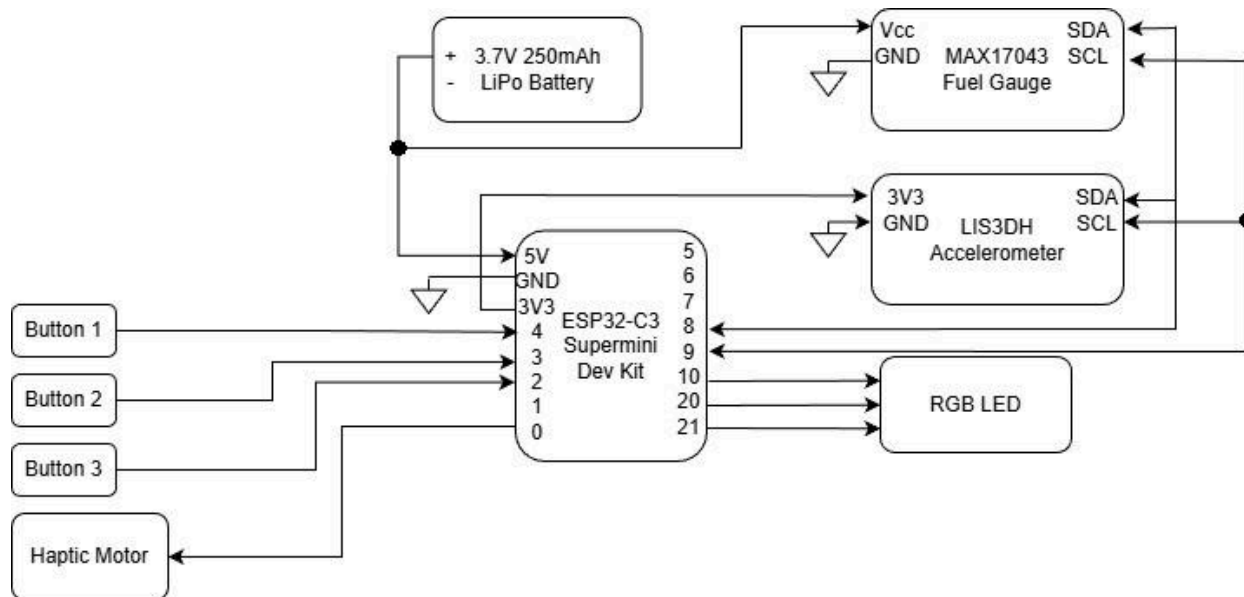


**Figure 2:** ESP32 Controller Functional Block Diagram

## Machine Learning Implementation

The goal of our machine learning implementation with respect to this project was to deliver a reliable gesture recognition feature capable of starting the clock hands free when the referee performs a distinctive “flick” gesture with their arm. The approach to delivering this feature was training a logistic regression based classifier model to categorize accelerometer data into either “idle”, “miscellaneous movement”, and “flick” categories. Datasets for the idle and miscellaneous movement categories were collected by implementing a custom ESP32 firmware to continuously poll the accelerometer and transmit the data over bluetooth to a python script

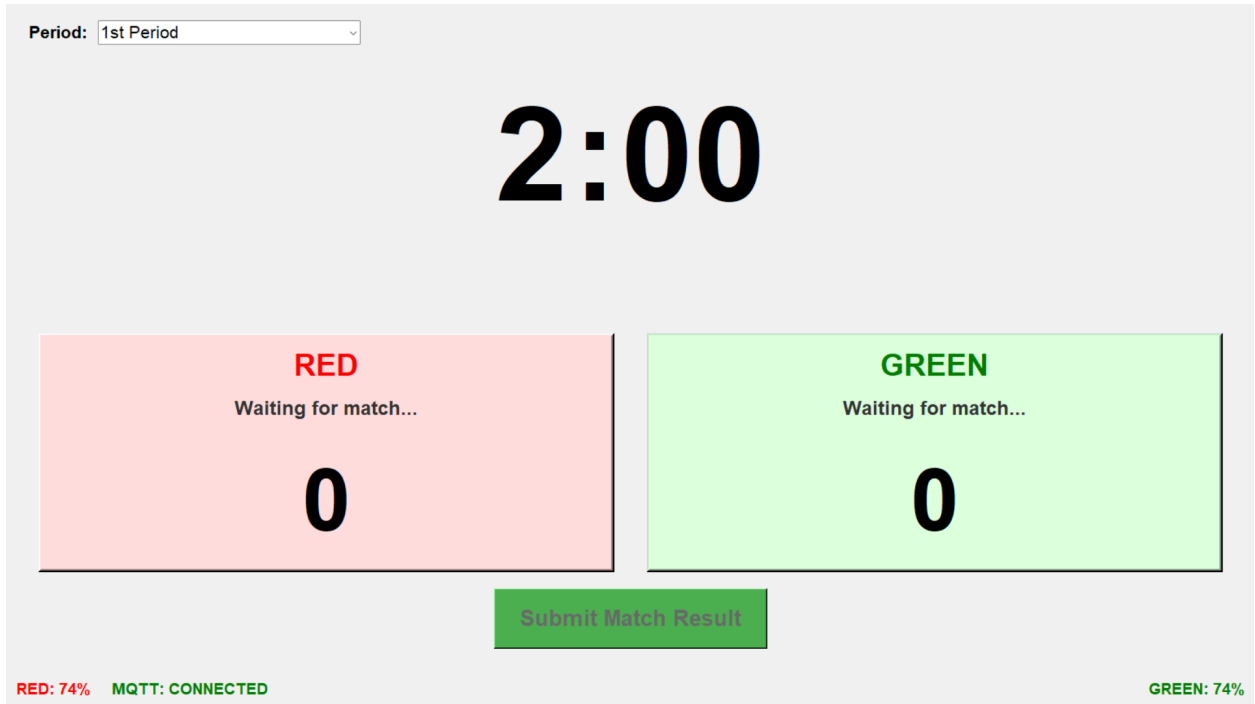
which saved the data as a .csv file. Data collection for each of these categories was performed over 2 minutes, in which the accelerometer was left to sit still for the idle category and continuously moved around the space in a variety of movements excluding a deliberate upward flick for the miscellaneous movement category. Since the flick gesture being targeted could not be performed continuously over the course of 2 minutes, a different strategy was employed to capture the dataset for that gesture. The ESP32 firmware was modified to collect and transmit accelerometer data for a 1 second window every 2 seconds, during this active window, an LED connected to the controller would become illuminated. Using this method, we were able to collect 100 samples of accelerometer data of each second when the target gesture was being performed, while excluding the time when resetting between each gesture.

The combined dataset was segmented into short, overlapping time windows to mimic how the controller would behave in real use. Because the accelerometer is sampled at 50 Hz, a 250 ms window corresponds to 12 samples. Windows were generated with 50% overlap to increase the number of training examples and improve temporal resolution. For each window, the model input wasn't time series data, but a compact set of 14 engineered features: mean, standard deviation, minimum, and maximum of X/Y/Z acceleration, plus mean and standard deviation of overall acceleration magnitude. This feature design was chosen to retain the "shape" and intensity of a flick while keeping the model lightweight. Before training, all features were normalized using Z-score standardization (subtract mean and divide by standard deviation), which improves numerical stability and makes the logistic regression weights more meaningful across differently scaled features.

Logistic regression was chosen because it produces a very small parameter set, trains quickly, and performs inference using fewer operations than many other machine learning algorithms, making it ideal for deployment on a microcontroller. After training, the learned parameters were exported and embedded directly into the ESP32 firmware. On-device inference runs continuously over a sliding window of the most recent 12 accelerometer samples. The same 14 features are computed, normalized using the stored parameters, and class scores are computed. The highest scoring class is selected. A cooldown period is applied after any detected flick to prevent repeated triggers from a single motion.

Testing revealed that the model performs well at distinguishing between when the accelerometer is at rest, when it is being moved regularly, and when it is suddenly accelerated and decelerated quickly. However, the model struggles to distinguish between sudden movements in different directions, presenting a risk of accidental triggering. This risk is mitigated by only running inference of accelerometer data when the user activates "gesture listener mode". This mode only becomes active when the clock is not currently running and only listens for the gesture on the specific controller for which it was activated. While active, an LED indicator on the controller lights up and a message is displayed on the scoreboard. When the gesture is detected, gesture listener mode is deactivated, the clock is started, and the ESP32 sends a haptic buzz to notify the user that the event was triggered successfully.

## Interface Designs



Period:

# 2:00

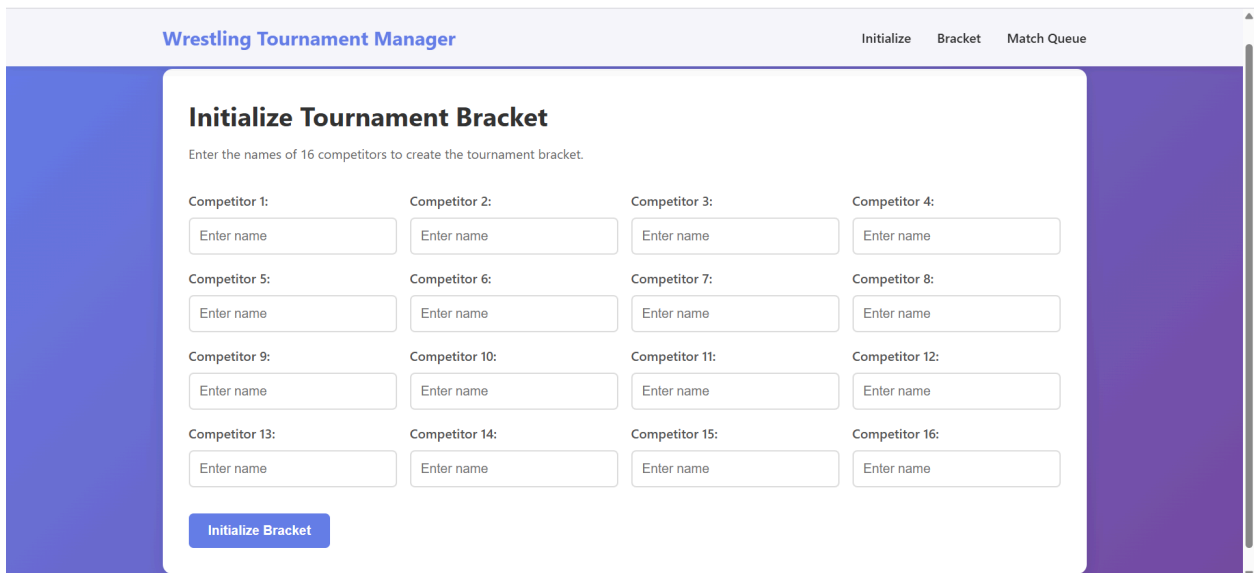
**RED**  
Waiting for match...  
**0**

**GREEN**  
Waiting for match...  
**0**

Submit Match Result

RED: 74% MQTT: CONNECTED GREEN: 74%

*Figure 3: Scoreboard Interface Design*



Wrestling Tournament Manager Initialize Bracket Match Queue

### Initialize Tournament Bracket

Enter the names of 16 competitors to create the tournament bracket.

|                                                           |                                                           |                                                           |                                                           |
|-----------------------------------------------------------|-----------------------------------------------------------|-----------------------------------------------------------|-----------------------------------------------------------|
| Competitor 1:<br><input type="text" value="Enter name"/>  | Competitor 2:<br><input type="text" value="Enter name"/>  | Competitor 3:<br><input type="text" value="Enter name"/>  | Competitor 4:<br><input type="text" value="Enter name"/>  |
| Competitor 5:<br><input type="text" value="Enter name"/>  | Competitor 6:<br><input type="text" value="Enter name"/>  | Competitor 7:<br><input type="text" value="Enter name"/>  | Competitor 8:<br><input type="text" value="Enter name"/>  |
| Competitor 9:<br><input type="text" value="Enter name"/>  | Competitor 10:<br><input type="text" value="Enter name"/> | Competitor 11:<br><input type="text" value="Enter name"/> | Competitor 12:<br><input type="text" value="Enter name"/> |
| Competitor 13:<br><input type="text" value="Enter name"/> | Competitor 14:<br><input type="text" value="Enter name"/> | Competitor 15:<br><input type="text" value="Enter name"/> | Competitor 16:<br><input type="text" value="Enter name"/> |

Initialize Bracket

*Figure 4: Tournament Manager Initialize Bracket Interface*

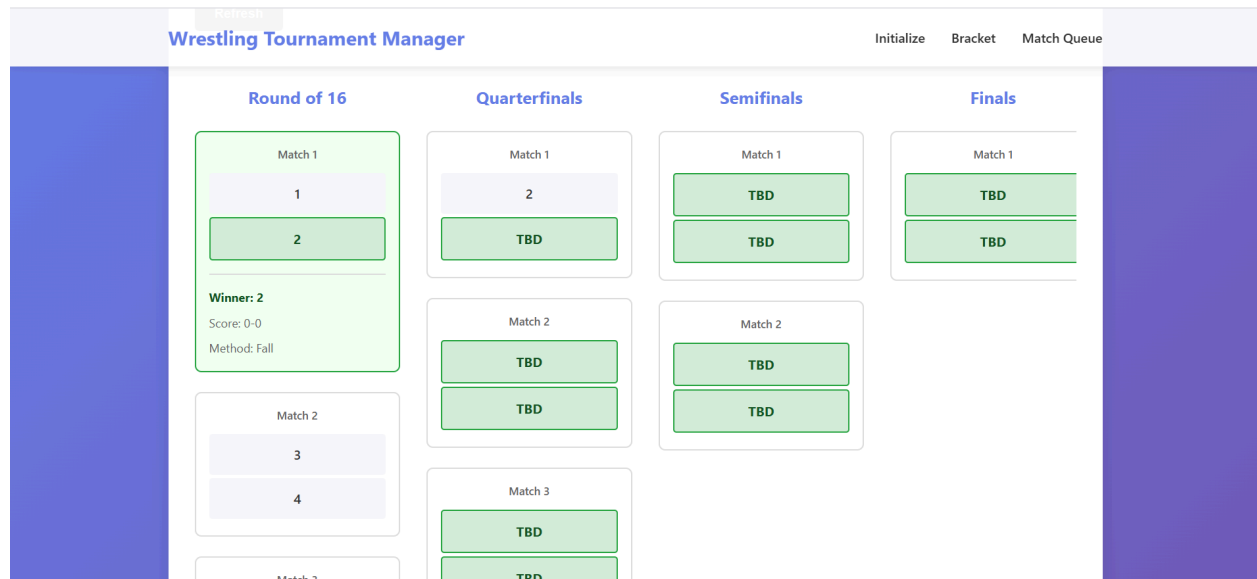


**Figure 5:** *Tournament Manager Bracket Status Interface*

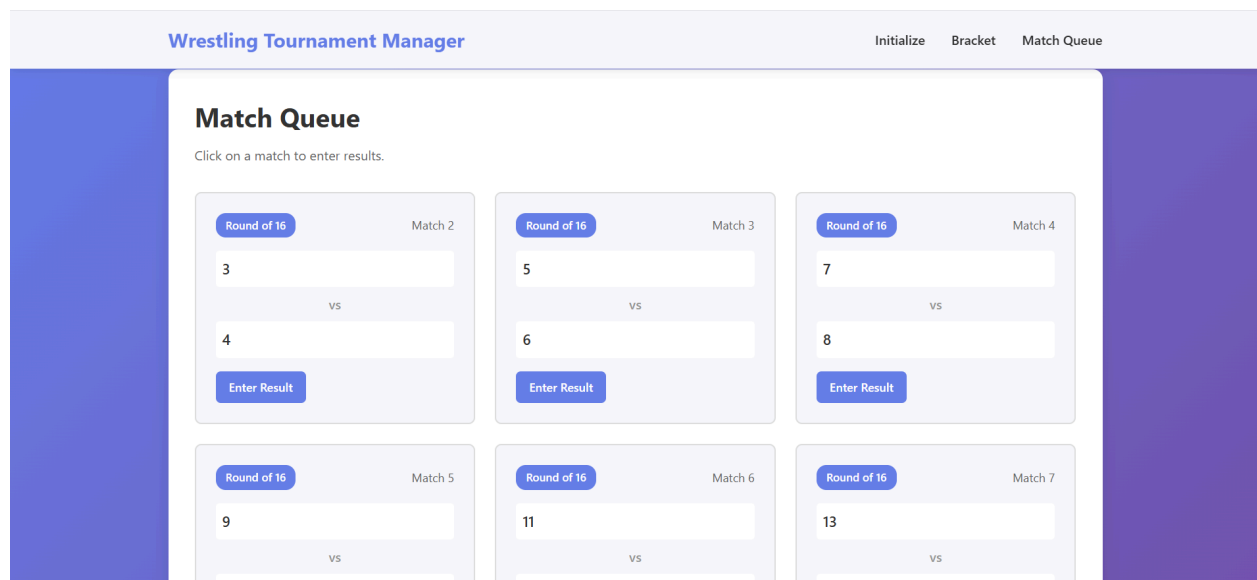


**Figure 6:** *Tournament Manager Match Queue Interface*

## Disclaimer

This submission contains confidential proprietary information intended solely for academic evaluation and is not authorized for public distribution.