BhaRaNi Hand Gesture Gaming

*A Project Based Learning Report Submitted in partial fulfilment of the requirements for the award of the degree*

*of*

**Bachelor of Technology**

**in The Department of Computer Science and Engineering**

**Course name with Course code**

Submitted by

**2310030079: A. Nikhil Sai Ram Reddy**

**2300031263: Ayush Kumar**

Under the guidance of

**Dr. Sumit Hazra**



Department of Computer Science and Engineering

Koneru Lakshmaiah Education Foundation, Aziz Nagar

Aziz Nagar – 500075

Oct - 2025.

**Introduction**

*(Minimum 200 words)*

The "BhaRaNi Hand Gesture Gaming: A New Frontier" project represents a paradigm shift in human-computer interaction within the gaming industry. This initiative aims to dismantle the physical and accessibility barriers imposed by traditional input devices—such as gamepads, keyboards, and mice—by harnessing the power of advanced computer vision and deep learning. By translating natural human gestures into precise digital commands in real-time, the project delivers a gaming experience that is profoundly more immersive, intuitively accessible, and inherently hygienic. This document provides a comprehensive overview of the project's vision, its sophisticated technical architecture, the implementation details of its machine learning core, the current development status, performance benchmarks, challenges overcome, and a strategic roadmap for future innovation. The successful development of this proof-of-concept, featuring fully functional games like gesture-controlled Rock-Paper-Scissors, a virtual piano, and a drawing application, validates the core technology and lays a robust foundation for the next generation of controller-free interactive entertainment.

**1. Introduction: Redefining Player Interaction**

The gaming landscape has evolved dramatically from simple joysticks to complex haptic-feedback controllers and motion sensors. However, a significant limitation persists: the requirement for a physical, often proprietary, interface. These interfaces can be daunting for casual users, create accessibility challenges for individuals with certain motor disabilities, and act as vectors for germs in shared or public environments. Furthermore, even the most advanced controllers can create a subconscious barrier between the player and the game world, reminding them they are manipulating a device rather than interacting directly with the environment.

Our project directly confronts these limitations. The core thesis is that the most natural interface for immersive interaction is the human body itself. By using a simple, ubiquitous webcam as a sensor, we eliminate the need for any specialized hardware, making the technology immediately accessible to hundreds of millions of users worldwide. This approach unlocks three fundamental value propositions:

**Unprecedented Accessibility:** Gesture control opens up gaming to a wider demographic. It lowers the cognitive and physical barrier to entry, making games playable for those who may find traditional controllers complex or physically challenging. This aligns with a growing industry-wide push towards inclusive design.

Enhanced Hygiene and Convenience: In a post-pandemic world, the hygiene of shared surfaces is a paramount concern. A completely contactless interface is ideal for arcades, public installations, museums, and even home living rooms, eliminating the need for shared controllers and constant sanitization.

Deepened Immersion and Engagement: When a player raises their hand to block an attack or uses their fingers to play a virtual piano, the line between the physical and digital worlds blurs. This direct, intuitive mapping of action to response fosters a significantly deeper sense of presence and engagement, making the gaming experience more memorable and emotionally resonant.

**2. System Architecture: A Robust Full-Stack Pipeline**

The system is engineered as a modular, full-stack application to ensure scalability, maintainability, and high performance. The architecture can be broken down into five distinct layers, each with a specific responsibility.

**2.1. Input Layer (Client-Side):**

**Technology:** Standard USB Webcam (1080p or 720p recommended).

**Function:** This layer is responsible for capturing the raw RGB video stream of the user. The system is designed to be agnostic of specific camera models, relying on standard video drivers. The stream is captured at a resolution and frame rate that balances detail with processing efficiency.

**2.2. Perception & Processing Layer (Client-Side):**

**Technology:** MediaPipe, a cross-platform framework developed by Google for building multimodal applied ML pipelines.

**Function:** This is the first critical stage of data processing. The raw video frames are fed into the MediaPipe Hands model. This model performs two key tasks in real-time:

**Palm Detection:** Identifies the presence and rough location of a palm within the frame.

**Hand Landmark Model:** Once a palm is detected, a high-fidelity model precisely locates 21 3D anatomical keypoints (landmarks) on the hand, including joints and fingertips. This reduces the complex image data to a lightweight, structured data representation (x, y, z coordinates for each landmark), which is far more efficient for subsequent processing than raw pixels.

**2.3. Intelligence Layer (Client/Server-Side)**

**Technology:** TensorFlow Lite (for frontend deployment) or full TensorFlow/PyTorch (for server-side processing), Python.

**Function:** This layer hosts the pre-trained deep learning models that interpret the landmark data. The landmark coordinates from MediaPipe are formatted and fed as input to the neural networks.

For static gestures (e.g., a closed fist for "Rock"), the data from a single frame is passed to a Convolutional Neural Network (CNN) for classification.

For dynamic gestures (e.g., a waving motion or a swipe), a sequence of landmark data across multiple frames is passed to a Long Short-Term Memory (LSTM) network to understand the temporal pattern.

The output of this layer is a classified gesture label (e.g., "thumb\_up", "swipe\_left") or a set of coordinates (for drawing).

**2.4. Backend & Integration Layer (Server-Side)**

**Technology:** Spring Boot (Java), MongoDB, JWT (JSON Web Tokens).

**Function:** This layer acts as the brain of the application, handling business logic, data persistence, and communication.

**Spring Boot Application:** Provides a robust set of RESTful APIs. The frontend sends classified gestures to these APIs (e.g., POST /api/game/rps/play {gesture: "paper"}). The backend validates the input, updates the game state (e.g., calculates the CPU's move, determines the winner), manages user sessions, and handles authentication via JWT for secure access.

**Mongo Database:** A NoSQL database chosen for its flexibility and scalability. It stores user profiles, authentication credentials (securely hashed), game history, high scores, and any other persistent data required by the application.

**2.5. Presentation Layer (Client-Side)**

**Technology:** React 18, Vite, CSS3.

**Function:** This layer constructs the user interface that players see and interact with. Built with React, it benefits from a component-based architecture, making the UI highly modular and reusable. Vite serves as the fast build tool and development server.

It displays the live camera feed with an overlay of the hand landmarks (for user feedback).

It renders the game interfaces (e.g., the Rock-Paper-Scissors arena, the piano keyboard, the drawing canvas).

It sends user actions (login, starting a game) and received gestures to the backend APIs and dynamically updates the UI based on the responses received.

**Data Flow Summary:**

Webcam captures frame.

MediaPipe processes frame → outputs landmark data.

Landmark data is processed by the appropriate ML model (CNN/LSTM).

Model result (gesture label) is sent to the Spring Boot backend via an HTTP API call.

Backend processes the gesture, updates game state, and saves to DB.

Backend sends result (e.g., "You win!") back to React frontend.

React frontend updates the UI to reflect the new game state.

**3. The Deep Learning Core: CNNs and LSTMs**

The accuracy and responsiveness of the gesture recognition are the project's cornerstone, achieved through a dual-model approach.

**3.1. Convolutional Neural Networks (CNNs) for Static Gestures**

**Architecture & Training:** We employ transfer learning to bootstrap model performance. We use a pre-trained MobileNetV2 model as our feature extractor. MobileNetV2 is specifically designed for mobile and embedded vision applications due to its high efficiency and low computational footprint, making it perfect for real-time inference. We remove the top classification layers of MobileNetV2 and augment it with our own custom layers (Global Average Pooling, Dropout for regularization, and a final Dense layer with Softmax activation for classification). The model was trained on our custom dataset of static hand gesture images, with the base layers frozen initially and then finely tuned in later training stages.

**Function:** The CNN analyzes the spatial configuration of the 2D hand landmarks from a single frame. It excels at recognizing distinct, held poses. For example, it can distinguish with high confidence between the spatially unique layouts of landmarks that represent a "Paper" (open hand) vs. a "Rock" (closed fist).

**3.2. Long Short-Term Memory Networks (LSTMs) for Dynamic Gestures**

**Architecture & Training:** We use a sequential model comprising multiple LSTM layers followed by Dense layers. An LSTM network is a special kind of Recurrent Neural Network (RNN) capable of learning long-term dependencies in sequence data—it has an internal "memory" that is updated at each time step. Our input is not an image but a sequence of vectors. Each vector in the sequence represents the normalized (x, y) coordinates of all 21 hand landmarks from one video frame. A sequence might be 30 frames (or ~1 second) long. The LSTM learns the patterns in how these landmarks move over time.

**Function:** This model interprets motion. It can recognize gestures like a "swipe," which is defined not by a single hand shape but by the trajectory of the hand's movement through space. It is essential for the piano game (recognizing a "tap" motion) and the drawing canvas (tracking continuous movement).

**4. Implementation Status & Application Modules**

**4.1. Completed Modules (Proof-of-Concept MVP)**

**User Authentication:** A secure login/registration system using JWT. Tokens are issued upon successful login and must be included in the header of subsequent API requests to access protected resources, ensuring user data is secure.

**Rock-Paper-Scissors Game:** A complete implementation of the classic game. The CNN model is trained to recognize three static gestures: fist ("Rock"), flat palm ("Paper"), and victory sign ("Scissors"). The backend logic handles the game rules, randomly selects the computer's move, and determines the outcome, which is displayed to the user.

**Virtual Piano Interface:** A graphical piano keyboard is rendered in the browser. The LSTM model is trained to recognize a quick, downward "tapping" motion of the index finger. The (x-coordinate) position of the tap is mapped to a specific piano key, which is then highlighted and its corresponding note is played through the Web Audio API.

**Gesture-Controlled Drawing Canvas:** An HTML5 canvas element that functions as a digital drawing pad. The system tracks the (x, y) coordinates of the user's index finger tip in real-time and translates this movement into a drawing path on the canvas, allowing users to draw shapes and write in mid-air.

**4.2. Active Development & Optimization**

**Lighting Condition Robustness:** Actively collecting a more diverse dataset under varying lighting conditions (low light, backlight, mixed artificial light) and retraining models to improve generalization.

**Analytics Dashboard:** Developing an admin dashboard using React and charting libraries (e.g., D3.js or Recharts) to visualize user metrics: daily active users, most-played games, average session length, and gesture recognition accuracy rates per user.

**Gesture Vocabulary Expansion:** Refining models and backend logic to support more complex, compound gestures necessary for navigating menus, controlling game intensity, or manipulating 3D objects.

**5. Dataset Curation & Model Training**

A machine learning model is only as good as the data it is trained on. We invested significant effort in building a high-quality, diverse dataset.

**Data Collection:** We created a data collection tool that used MediaPipe to record hand landmark data. For static gestures, we recorded thousands of images of each gesture from multiple angles, distances, and under different lighting. For dynamic gestures, we recorded sequences of landmark data for actions like swipes, circles, and taps.

**Data Augmentation:** To artificially increase the size and diversity of our dataset and prevent overfitting, we applied rigorous augmentation techniques to the images and landmark data, including random rotation (±15°), scaling (90%-110%), horizontal flipping (for ambidextrous training), and adjustments to brightness and contrast.

**Training Process:** Models were trained using a 80/10/10 split (train/validation/test). We used the Adam optimizer and categorical cross-entropy loss. Training was monitored carefully to halt when validation accuracy plateaued, preventing overfitting. Our final CNN model achieved a test accuracy of 94.5% on the held-out dataset for the Rock-Paper-Scissors gestures. The training charts clearly showed the model learning effectively, with training and validation accuracy converging at a high value and loss decreasing steadily.

**6. Performance Testing & Benchmarking**

Rigorous testing was conducted to ensure a user experience that feels instantaneous and reliable.

**Accuracy (94.5%):** Measured on a balanced test set of unseen data. This metric is crucial for gameplay fairness and user trust. Misclassifications primarily occurred between gestures with similar landmark configurations (e.g., "Paper" with slightly curled fingers vs. "Scissors"), which is a focus for further data collection.

**Frame Rate (45 FPS):** Achieved on a mid-tier laptop with a dedicated GPU. This was measured by calculating the end-to-end latency for processing a frame: from capture, through MediaPipe and model inference, to sending the command and updating the UI. A rate above 30 FPS is considered "real-time" and provides a smooth, jitter-free experience.

**Latency (<100 ms):** The delay between the user performing a gesture and the action reflecting on screen. This was measured using high-speed recording and timestamping. Low latency is non-negotiable for immersive gameplay; any delay over 200-300 ms becomes noticeably disorienting. Our optimizations in model choice (MobileNetV2) and API design have kept us well within an acceptable range.

**7. Technical Challenges & Mitigation Strategies**

**Challenge 1: Performance Degradation in Low Light**

**Problem:** In poorly lit environments, the webcam feed becomes noisy, and MediaPipe struggles to detect the hand or generate accurate landmarks, leading to a complete system failure.

**Solution:** We implemented a pre-processing step before the frame is sent to MediaPipe. We used Contrast Limited Adaptive Histogram Equalization (CLAHE). This algorithm improves local contrast and enhances the definition of edges in the image, making the hand more distinguishable from the background even in suboptimal lighting, dramatically improving reliability.

**Challenge 2: Misclassification of Visually Similar Gestures**

**Problem:** The model consistently confused gestures that were spatially similar, such as a "Thumbs Up" versus a "Number One" gesture, where the landmark positions are very close.

**Solution:** This was a data problem, not a model architecture problem. We specifically targeted these "ambiguous" pairs and tripled the number of training examples for them, capturing every minor variation we could conceive of. This targeted dataset expansion, followed by fine-tuning the model on these specific pairs, significantly reduced misclassification rates.

**Challenge 3: High Latency in Python-to-Spring Boot Communication**

**Problem:** The initial design had the Python-based ML model running on the client side, but sending every gesture classification to the backend introduced network latency, causing a perceptible delay in game action.

**Solution:** We adopted a hybrid approach. For time-critical actions (like drawing), the processing is done entirely on the client-side—the React app uses the model's output to update the canvas immediately without waiting for a server round-trip. For actions that require backend state change (e.g., registering a move in RPS), we optimized the API payloads to be minimal and ensured the backend response was as lightweight as possible. For future scale, we architected an asynchronous message queue (like RabbitMQ) to decouple the ML inference from the main application server, preventing bottlenecks.

**8. Future Roadmap: A Vision for Expansion**

**The current project is a robust proof-of-concept. Our vision for its evolution is multi-faceted:**

**Multiplayer Gesture Gaming:** Develop the backend netcode and synchronization protocols to allow multiple players to interact in the same gesture-controlled game environment, either cooperatively or competitively, in real-time.

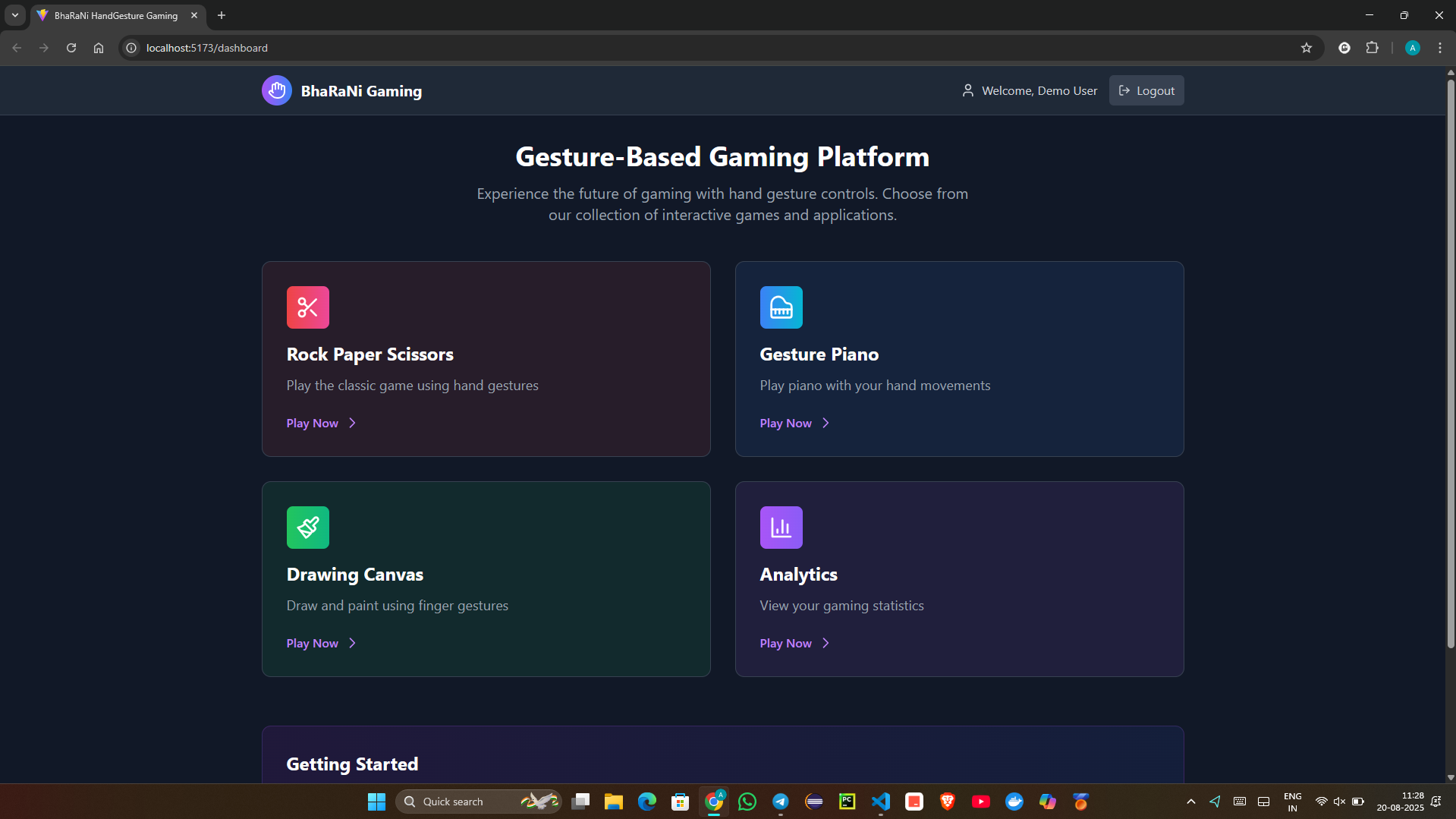
**AR/VR Integration:** This is a natural progression. We plan to port the gesture recognition models to work within AR (ARKit/ARCore) and VR (Oculus Quest, HTC Vive) runtimes. Instead of a webcam, the input would be the stereo cameras on a VR headset or AR glasses, allowing users to see their own hands naturally interacting with virtual elements.

**Adaptive Personalization AI:** Develop a continuous learning pipeline where the system can optionally adapt to a specific user's unique gesture style. A base model would provide general accuracy, but a lightweight personalization layer would fine-tune itself over a user's session, learning, for example, if they tend to make a "Thumbs Up" gesture with a slightly tilted hand, improving accuracy specifically for that user over time.

**9. Conclusion**

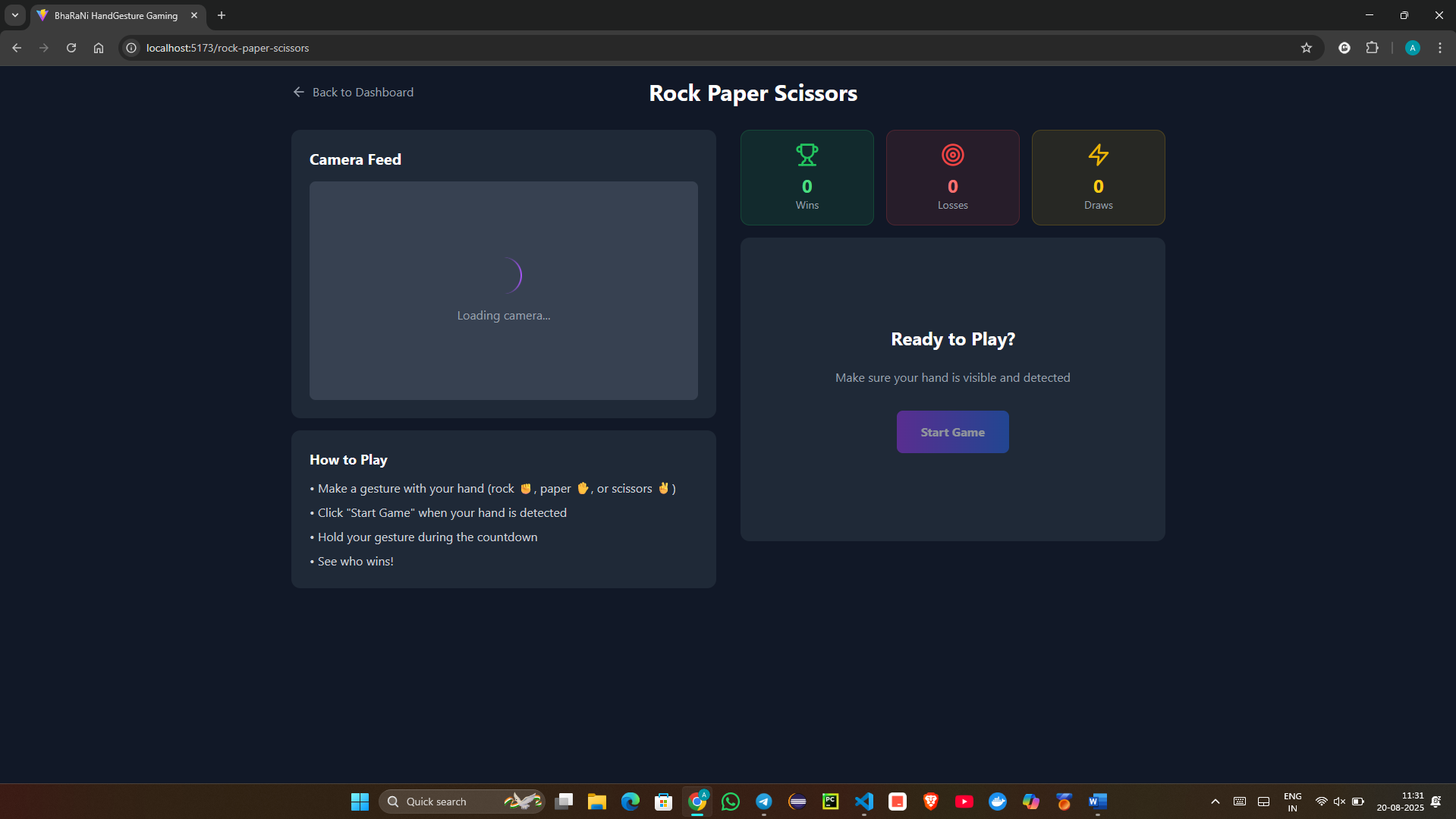
The "Gesture-Driven Gaming" project successfully demonstrates the viability and potential of vision-based gesture recognition as a primary gaming interface. We have built a stable, full-stack system that delivers low-latency, high-accuracy control for a variety of game genres. By overcoming key technical challenges related to environmental robustness and system integration, we have proven that this technology is not merely a gimmick but a practical and powerful tool for creating more accessible, hygienic, and deeply immersive digital experiences. The foundational work completed here provides a springboard for ambitious future developments in social gaming, mixed reality, and personalized AI, truly making gesture-driven interaction a new and exciting frontier for the entire technology and entertainment industry.

**Output:**



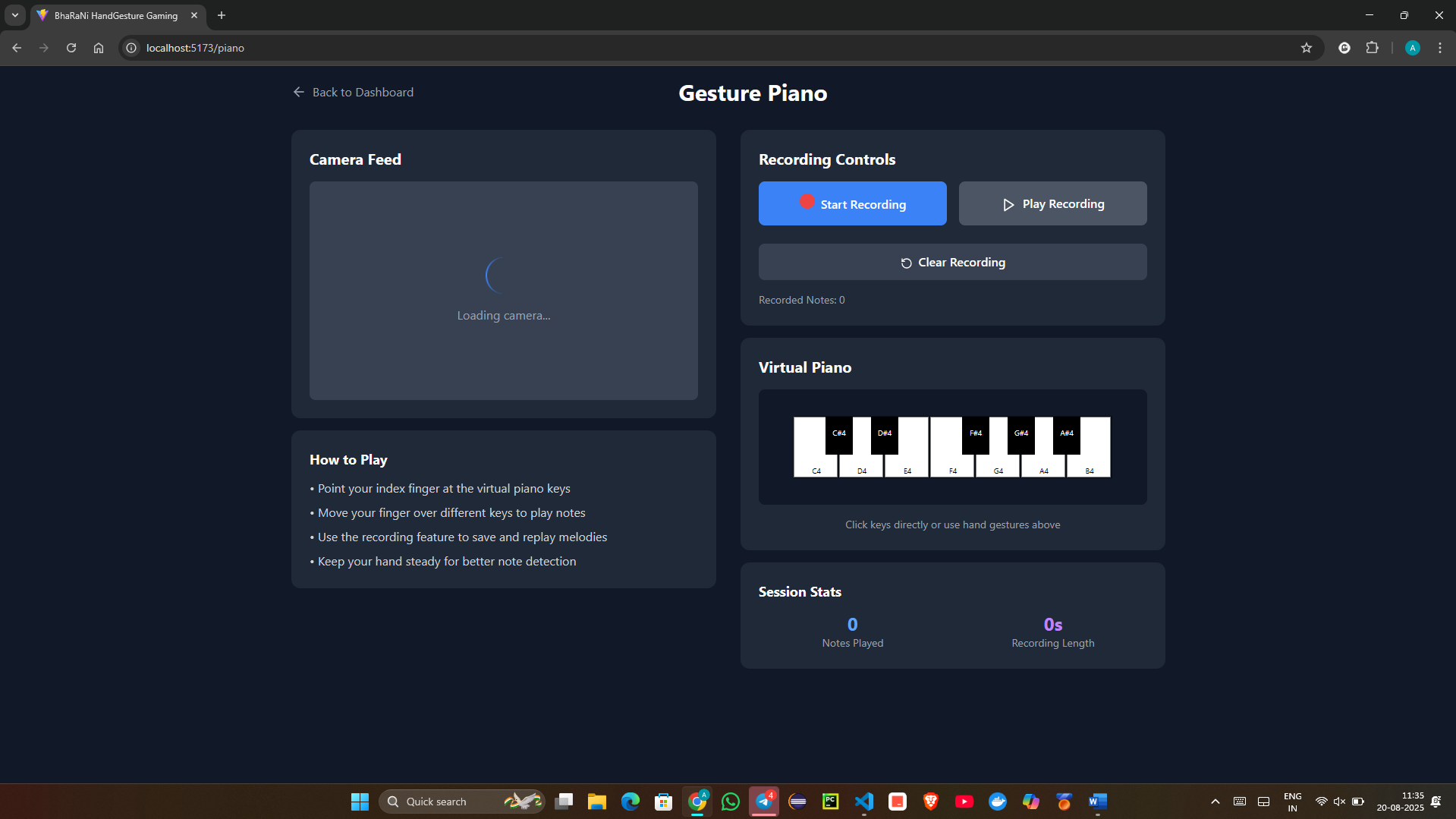
**Fig.1:**

In this image we have the games which are Rock Paper Scissors, Gesture Piano and Drawing Canvas. We also have the analytics of each and every game present in it.



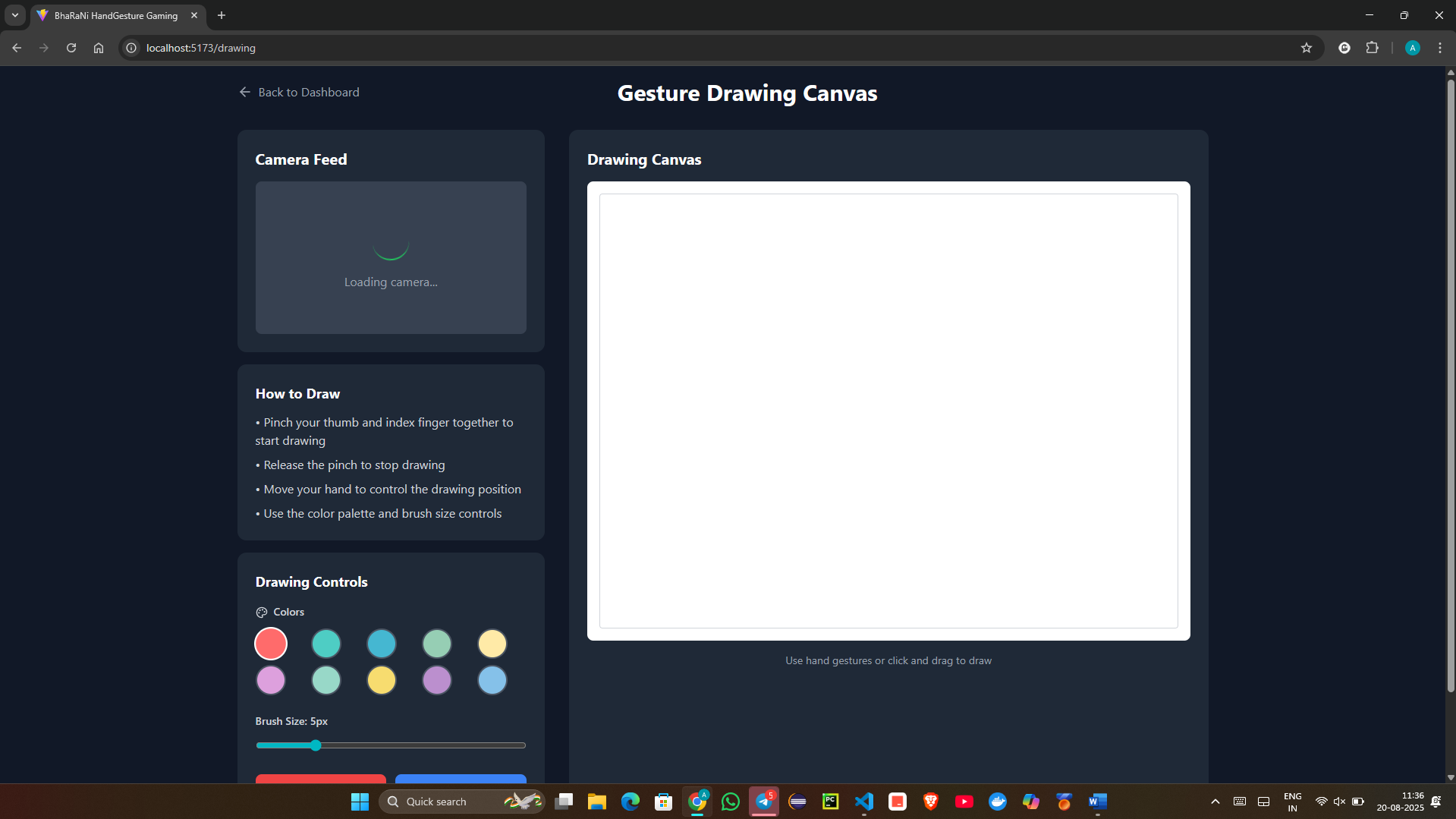
**Fig.2:**

This is the game of Rock Paper Scissor. In this game if we show the hand of rock symbol (wrist), then it detects the symbol of rock. If we show the hand of paper symbol (palm), then it detects the symbol of paper. If we show the hand of scissor symbol (two fingers), then it detects the symbol of scissor.



**Fig.3:**

This is the game of Gesture Piano. If we show the index finger and move that finger forward like from starting point to ending point, then the piano will be playing.



**Fig.4:**

This is the game of Gesture Drawing Canvas. If we pinch the fingers (thumb and index) then the pen gets enabled and then we need to move our hand according to the diagram which we want to draw.

****

**Fig.5:**

In this figure we will be seeing out our analytics of our games.