A Major project on

Rock Paper Scissors game using OpenCV

submitted in partial fulfillment for award of the degree of Bachelor of Technology in Information Technology by

Nishitha Eerishetty	**21321A1262**
Maneesha Gaddam	**21321A1250**
Manaswini Karre	**21321A1248**

Under the Esteemed Guidance of Internal Guide

B Anitha

Associate Professor, Information Technology

Bhoj Reddy Engineering College for Women

Department of Information Technology

(Sponsored by Sangam Laxmibai Vidyapeet, Accredited by NAAC with A Grade, Approved by AICTE and Affiliated to JNTUH) Vinaynagar, IS Sadan Crossroads, Saidabad, Hyderabad – 500 059, Telangana. www.brecw.ac.in

Ph: +91 -40-2453 7282; Email: principal@brecw.ac.in

AY 2024 - 25

Ref No: BRECW/IT Dept/Project 2024-25/PJ-A-21 Date: 11 - 06 - 2025

CERTIFICATE

This is to certify that the Major Project entitled **"Rock Paper Scissors Game using OpenCV"** is a Bonafide work carried out by

Nishitha Eerishitty	**21321A1262**
Maneesha Gaddam	**21321A1250**
Manaswini Karre	**21321A1248**

In partial fulfillment for award of the degree of Bachelor of Technology in Department of Information Technology from Bhoj Reddy Engineering College for Women, Hyderabad affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH).

B Anitha
Internal Guide

Dr C Murugamani
Head of the Department

External Examiner

ACKNOWLEDGEMENT

It is our pleasure to express our whole hearted thanks to our internal guide **B Anitha, Assistant Professor, Department of Information Technology,** for her extreme guidance and support in completing this project successfully.

We are thankful to our project coordinator **Tasneem Rahath, Assistant Professor, Department of Information Technology,** for her dynamic valuable guidance and constant management.

We express thanks and gratitude to **Dr C Murugamani, Professor & Head of the Department, Department of Information Technology,** for his encouragement and guidance in carrying out the major project presentation.

We would also like to thank **Dr J Madhavan, Professor & Principal of Bhoj Reddy Engineering College for Women** for encouragement in carrying out our major project successfully. We are also thankful to the staff members of Information Technology department, my friends and to our parents who helped us in completing this major project successfully.

By

Nishitha Eerishetty	(21321A1262)
Maneesha Gaddam	(21321A1250)
Manaswini Karre	(21321A1248)

Index

List of Figures

List of Tables

Abstract

The AI-Powered Rock-Paper-Scissors Game leverages real-time hand gesture recognition to create an interactive gaming experience using computer vision and artificial intelligence. Developed in Python with OpenCV and CvZone libraries, the system captures webcam input and utilizes the Hand Tracking Module for precise finger detection. Based on the number of fingers raised, the player's move (rock, paper, or scissors) is interpreted and compared against a randomly generated AI move, displayed using dynamic overlays on a custom-designed game background. The game initiates on a virtual keypress ('S') and features a countdown timer, intuitive graphical interface, and live score tracking. The integration of transparent PNG overlays and real-time video processing highlights the application's graphical fidelity and responsiveness. This project showcases the fusion of gesture control with classical game logic, emphasizing the utility of hand tracking in interactive applications. It holds potential in the realms of computer vision education, human-computer interaction, and gesture-based gaming interfaces.

Keywords: Computer Vision, Hand Tracking, Real-Time Gesture Recognition, Human-Computer Interaction, OpenCV, Cv Zone, Interactive Gaming, AI-Powered Game.

1. Introduction

1. Introduction

Gesture-based gaming, a subdomain of human-computer interaction (HCI), has gained widespread attention due to its ability to foster intuitive and immersive experiences without physical controllers. Central to this advancement is OpenCV, a robust open-source computer vision library extensively utilized for real-time image processing and object detection. Patel and Desai [1] present a comprehensive overview of OpenCV, highlighting its role in enabling image enhancement, object tracking, machine learning, and facial and gesture recognition. These capabilities make OpenCV an ideal foundation for interactive applications such as the Rock-Paper-Scissors (RPS) game.

Kumar and Singh [2] further reinforce OpenCV's practical utility in real-time image processing, emphasizing its modularity and extensive support for algorithmic prototyping. Their insights illustrate how OpenCV simplifies the implementation of complex vision-based systems, particularly in the context of gesture recognition.

Several studies have successfully applied OpenCV to the development of AI-powered RPS games. Mathur and Gupta [3] demonstrated an early implementation in which image datasets of hand gestures were captured and classified using OpenCV. This laid the groundwork for integrating gesture-based input into gameplay mechanics. Harish et al. [4] extended this concept by incorporating Media Pipe alongside OpenCV, creating a robust gesture-based interface that enhances responsiveness and user engagement.

Kumar and Singh [5] addressed critical real-world challenges in gesture recognition, such as inconsistent lighting conditions and varying webcam resolutions. Their work underscores the importance of preprocessing and adaptive detection methods, both of which are integrated into our system to ensure stable performance across diverse environments.

In the broader context of object detection, Verma and Kaur [6] analyzed a range of OpenCV techniques designed to improve detection accuracy and speed. Their review supports the effective segmentation and tracking of hands, essential for identifying RPS gestures in real-time.

Building on these foundations, Sharma and Verma [7] focused on data-driven modeling by capturing a comprehensive dataset of hand gestures for rock, paper, and

scissors. Machine learning techniques were then applied for gesture classification, enabling more robust and intelligent game behavior. Khan and Ali [8] echoed these findings and highlighted the importance of low-latency performance in real-time gesture recognition, especially in interactive games where visual feedback must be immediate.

Finally, Sharma and Gupta [9] emphasize the broad application range of OpenCV in computer vision tasks, including gesture control, augmented reality, and mobile robotics. Their work underscores OpenCV's position as a de facto standard for vision-based application development, and its relevance in designing responsive, AI-enhanced user interfaces.

Inspired by these contributions, our RPS project integrates OpenCV, CvZone, and MediaPipe to create a seamless, real-time gaming experience. The system captures hand gestures via webcam, classifies them using finger pattern recognition, and pits the user against an AI opponent whose choices are visualized using transparent overlays. With built-in timers, score tracking, and graphical feedback, the application delivers a complete gaming interface rooted in state-of-the-art computer vision and gesture tracking techniques.

Moreover, the integration of gesture recognition into traditional games like Rock-Paper-Scissors not only modernizes the gameplay experience but also demonstrates the versatility of vision-based interaction systems in education, entertainment, and assistive technologies. By leveraging the hand tracking capabilities of the HandTrackingModule from CvZone, built on top of OpenCV, this project captures and interprets finger patterns ([0,0,0,0,0], [1,1,1,1,1], [0,1,1,0,0]) to represent the three primary gestures—rock, paper, and scissors, respectively. The gesture classification is performed in real-time, with the system dynamically rendering the AI's response using PNG overlays.

This is further enhanced with an intuitive graphical user interface (GUI) featuring a custom-designed background, live webcam feed, countdown timer, and score display ensuring a seamless and engaging user experience.

This project also addresses several technical challenges reported in existing literature, such as gesture ambiguity, camera angle sensitivity, and frame consistency. By resizing and cropping the video feed before analysis, and timing each round with precision, the application ensures consistency in gesture detection. The game's initiation is controlled via a virtual 'S' key trigger, after which gameplay is managed through a state-based logic

system, separating gesture capture from result computation and display. As supported by the findings of Khan and Ali [8] and Kumar and Singh [5], such design strategies are vital for maintaining performance across diverse environments. Through the synthesis of existing research and practical implementation, this project exemplifies how OpenCV-powered systems can evolve simple games into sophisticated interactive experiences, bridging the gap between human intuition and machine perception.

1.1 Purpose of the Project:

The primary objective of this project is to develop an interactive Rock-Paper-Scissors (RPS) game that uses gesture recognition via computer vision to replace traditional input methods with real-time hand gestures. By employing OpenCV, CvZone, and MediaPipe, the system captures and analyzes hand movements from a webcam to identify the user's gesture. This provides an engaging and touchless interface where the user competes against an AI opponent, which randomly selects its own move. The game operates through intuitive visual feedback, including overlays of AI gestures, score tracking, and a countdown timer, enhancing user interaction. The choice of Rock, Paper, or Scissors is detected using finger pattern recognition, mapped to logical decisions in the gameplay. The purpose is not just entertainment, but to demonstrate a novel way of controlling digital applications through natural gestures.

Another core purpose of this project is to showcase the real-world applicability of computer vision and AI-based gesture tracking for interactive system design.

By simulating a familiar game, the project provides an accessible platform to explore human-computer interaction (HCI) using only a webcam and Python-based tools. It highlights how traditional game mechanics can be enhanced through vision-based interfaces, removing the need for physical buttons, mouse clicks, or touchscreen inputs. This not only modernizes the user experience but also creates a base model that can be adapted to other gesture-controlled applications. Additionally, the implementation reinforces key concepts in computer vision, including region of interest extraction, frame preprocessing, image classification, and decision logic using machine learning principles.

The project also aims to address and overcome several challenges commonly encountered in gesture-based systems, such as variations in lighting conditions, inconsistent hand positioning, and background noise. These real-world issues were identified in prior research and are managed in this implementation by resizing and cropping the video feed, using consistent detection zones, and timing the gesture capture with a visual countdown. By ensuring consistent detection regardless of environmental variation, the project serves as a demonstration of robust gesture recognition under practical constraints. Such robustness is crucial for extending similar systems into more complex environments like smart classrooms, physical therapy, or virtual learning platforms where precision and reliability are essential.

Beyond technical objectives, the broader purpose of the project is to encourage the adoption of AI-powered interactive systems across both educational and recreational domains. The contactless nature of gesture-based control has clear advantages in today's digital landscape—especially in a post-pandemic world where touchless interfaces are preferred for hygiene and accessibility. Ultimately, this Rock-Paper-Scissors game not only offers a playful and educational introduction to computer vision but also lays the groundwork for future innovations in gesture-based applications, making technology more natural, inclusive, and intuitive for users of all backgrounds.

1.2 Existing System:

The currently available Rock-Paper-Scissors (RPS) systems, whether in physical or digital format, offer only limited interactivity. Traditional versions rely entirely on human judgment, which is prone to error, while digital adaptations typically depend on static inputs like mouse clicks or keyboard controls. These input methods, although functional, fail to deliver an engaging or immersive experience. According to the document, many existing systems that attempt to integrate gesture recognition often do so using outdated or inaccurate algorithms, leading to poor recognition rates and incorrect outcomes. As a result, players are frequently frustrated by mismatched interpretations and lack of visual feedback. These systems generally lack real-time responsiveness, making the gameplay rigid and repetitive.

Furthermore, existing gesture-based RPS games suffer from low visual appeal and minimal AI integration. As discussed in the document, static interfaces with little to no animation or dynamic overlays diminish user engagement, failing to replicate the excitement of real-time gameplay. Most existing platforms do not support adaptive feedback, countdowns, or gesture-based state management—features which are essential for a modern interactive experience. From a technical perspective, challenges like environmental lighting changes, poor hand tracking, and fixed camera dependencies remain unresolved. This often results in inconsistent gesture detection and limits the platform’s usability across different devices and user environments. Such limitations reduce replayability and user satisfaction, especially among tech-savvy players accustomed to fluid and intelligent interfaces.

In contrast, our project overcomes these limitations by implementing a real-time hand detection and classification system using OpenCV and CvZone’s HandDetector. Our code initiates the game using a virtual 'S' key trigger, creating a seamless transition into gameplay with a dynamic timer, score tracking, and visual overlays of AI gestures. Unlike existing systems, our solution resizes, crops, and preprocesses the webcam feed to maintain detection consistency regardless of lighting or resolution variations.

This not only improves gesture recognition accuracy but also enhances user experience with fluid feedback and animated responses. The AI opponent adds unpredictability, while the well-structured game states—timed gesture input, result evaluation, and real-time display—ensure a smooth and immersive experience. Thus, our implementation directly addresses the shortcomings of traditional and existing gesture-based RPS games through an intelligent, vision-driven approach.

1.3 Disadvantages of Existing System:

- Limited Interactivity
- Static Interfaces
- Lack of AI Integration
- Low User Engagement
- Outdated or Inaccurate Gesture Recognition
- No Real -Time feedback
- Hardware Dependency

1.4 Proposed System:

The proposed system is an interactive, AI-driven Rock-Paper-Scissors game that utilizes real-time hand gesture recognition to enhance user engagement and provide a seamless gameplay experience. It leverages computer vision techniques via the OpenCV library combined with the cvzone framework's hand tracking module to accurately detect and interpret the player's hand gestures representing rock, paper, or scissors.

At the core, the system captures live video feed from a webcam and applies a hand detector that identifies the number of fingers raised to classify the player’s move. The game starts only upon the user pressing the 'S' key, allowing for controlled initiation. Once started, a 3-second timer countdown is displayed prominently, providing the player with sufficient preparation time before the move detection phase.

The AI opponent randomly selects its move after the countdown. The game then compares the player's gesture against the AI’s choice to determine the winner based on classic Rock-Paper-Scissors rules. The system dynamically overlays the AI's move icon and the live video feed onto a customized background, enriching the visual appeal. Scores for both the AI and the player are tracked and updated in real-time, displayed clearly on the game interface.

This design emphasizes responsiveness and user experience by ensuring that the gameplay occurs only after deliberate user input, reducing false starts. The gesture recognition approach allows intuitive, touchless interaction, making the game accessible and engaging. Additionally, modular code structure facilitates future enhancements such as adding multiplayer modes, varying difficulty levels, or integrating machine learning-based move prediction.

1.5 Advantages of Proposed System:

- Interactive Gameplay Experience
- Hands-Free Input with Hand Detection
- Real-Time Feedback and Scoring
- User–Friendly Interface
- Cross – Platform Compatibility
- Efficient and Lightweight Implementation

2. Related Work

2. Related Work

2.1 Survey:

The evolution of computer vision has accelerated dramatically with the widespread adoption of libraries such as OpenCV, which provides a rich set of tools for real-time image processing, object detection, and gesture recognition. These capabilities have paved the way for interactive systems that can interpret human movements with high accuracy and responsiveness. As reviewed by Patel and Desai [1], OpenCV has become a cornerstone in the development of vision-based applications due to its open-source nature, performance efficiency, and extensive community support. Kumar and Singh [2] also underscore the role of OpenCV in simplifying complex computer vision workflows across numerous domains.

A compelling use case of OpenCV is the development of gesture-controlled games such as Rock-Paper-Scissors (RPS). This game serves as a prime example of how vision-based interaction can replace traditional input methods. Mathur and Gupta [3] introduced a framework where hand gestures captured through a webcam are interpreted using OpenCV to drive the gameplay. Their approach utilizes edge detection, contour analysis, and convexity defects to distinguish between the three gestures—rock, paper, and scissors—demonstrating real-time interaction and engagement.

Enhancements in gesture recognition were further explored by Harish et al. [4], who focused on building a gesture-based application that adapts to variations in background and lighting. Their system used robust pre-processing techniques to stabilize hand detection and maintain consistent classification accuracy. The inclusion of feedback mechanisms in their implementation made the interaction more user-friendly and dynamic.

Kumar and Singh [5] extended the functionality of the RPS game by integrating artificial intelligence, allowing the system to behave as a responsive game opponent. Using decision logic and tracking of previous moves, the AI was able to simulate strategic thinking. This convergence of OpenCV and AI showcases the growing capability of vision systems not only to detect inputs but also to make informed decisions.

The foundational element of such gesture recognition systems is accurate object detection. Verma and Kaur [6] reviewed OpenCV's object detection features, emphasizing techniques such as Haar cascades and color segmentation. These methods serve as the groundwork for hand tracking, especially in dynamic environments. Sharma and Verma [7] developed a gesture detection model using HSV masking, image thresholding, and contour properties, allowing the system to differentiate gestures even under inconsistent lighting conditions or partially occluded views.

Gesture classification remains a key challenge in computer vision applications, and Khan and Ali [8] proposed improvements for gesture recognition in lightweight applications. Their emphasis on efficient processing pipelines and mobile compatibility reflects a broader movement toward deploying gesture-based systems in constrained environments. Additionally, Sharma and Gupta [9] demonstrated how OpenCV's image analysis tools are applicable in various domains, reinforcing its adaptability from games to surveillance and robotics.

Building on this foundation, the current project presents an OpenCV-powered Rock-Paper-Scissors game that captures and classifies hand gestures in real time using a webcam. By combining background removal, contour detection, and gesture classification logic, the system offers an engaging user experience that responds to motion instantly. A scoring mechanism and AI-based opponent logic enrich the gameplay by simulating competitive interaction.

While this implementation demonstrates the potential of OpenCV for interactive applications, it struggles with diverse hand shapes, backgrounds, and camera quality. Future improvements could involve CNNs or transformer-based models to enhance gesture recognition accuracy. Sharma and Gupta (2021) emphasize the importance of virtual keyboards in enabling intuitive, touchless human-computer interaction, extending applications beyond gaming into education, healthcare, and accessibility. Integrating these advances may broaden the usability and robustness of gesture-based systems.

3. Requirement Analys

3. Requirement Analysis

3.1 Functional Requirements:

The functional requirements for the Rock-Paper-Scissors Game with Real-Time Gesture Recognition ensure the system delivers an interactive and engaging experience. Here are the key functional requirements:

- **Real-Time Gesture Recognition:** The system shall accurately detect and interpret hand gestures (rock, paper, scissors) in real-time using webcam input and the HandTrackingModule.
- **AI Move Generation:** The system shall generate a random move (rock, paper, or scissors) for the AI opponent.
- **Immediate Outcome Display:** The system shall display the outcome of each round (win, loss, or tie) immediately after both player and AI moves are determined.
- **Interactive Interface:** The game shall feature an intuitive graphical interface with dynamic overlays for player and AI moves, and a custom-designed game background.
- **Result calculation:** The system shall correctly determine the winner of each round based on the classic Rock-Paper-Scissors rules.
- **Score Tracking and Display:** The system shall track and display the live scores for both the player and the AI.
- **Game Initiation:** The game shall initiate upon a virtual keypress (specifically, 'S').
- **Countdown Timer:** The system shall incorporate a countdown timer before each round begins.
- **Video Processing and Overlays:** The system shall integrate transparent PNG overlays and perform real-time video processing to enhance graphical fidelity and responsiveness.

3.2 Non-Functional Requirements:

Non-functional requirements for the Rock-Paper-Scissors Game with Real-Time Gesture Recognition ensure the system performs efficiently and provides a seamless user experience. Here are the key non-functional requirements:

- **Performance:** The system shall process hand gestures and update the game state in real-time with minimal latency to ensure a smooth gaming experience.
- **Accuracy:** The hand gesture recognition module shall accurately identify the player's moves with a high degree of precision.
- **Reliability:** The system shall consistently operate without crashes or significant errors during gameplay.
- **Compatibility:** The system shall be compatible with standard webcams and common operating systems (e.g., Windows, macOS, Linux, given Python and OpenCV's cross-platform nature).
- **Efficiency:** The system shall utilize computational resources (CPU, GPU, memory) efficiently to avoid system slowdowns.
- **Maintainability:** The code should be well-structured, modular, and documented to facilitate future updates and maintenance.
- **Resource Optimization:** The application should be optimized to minimize resource consumption, especially important for real-time video processing.
- **User-Friendly Interface:** The graphical interface should be intuitive and easy for users to understand and interact with.

3.3 Computational Resource

3.3.1 Software Requirements:

Runtime Environment	: Windows 10/11 , macOS , or Linux
AI Library	: Python 3.7 or higher
Front-End Framework	: PyCharm
Styling	: Open CV , cvzone , NumPy , random , and time
Development Enviroment	: Python and pip for dependency management

3.3.2 Hardware Requirements:

Development Machine	: Personal computer or laptop
GPU	: Integrated CPU
Processor	: Intel Core i3 or higher
Memory	: 4GB RAM(8GB recommended)
Storage	: 1GB free space

4. Design

4. Design

4.1 Architecture:

4.1.1 System Architecture:

The System Architecture diagram provides a high-level, abstract view of the major components of your Rock-Paper-Scissors game and how they interact. It focuses on the logical flow of data and control within the system.

1. **User Interface Module:**

   - **Purpose:** This is the visible part of your application that the user interacts with. It's responsible for displaying all game-related information.
   - **Inputs:** Receives user commands (e.g., "Press 'S' button").
   - **Outputs:** Sends game state information (like scores, countdowns, game outcomes) to the user.
   - **Initiates game:** When the user presses 'S', it signals the Timer and State Management Module to start the game.
   - **Receives updates:** It receives score updates from the Game Logic Module with AI generated and game state (e.g., countdown, player move, AI move, outcome) from various modules to display them visually.

2. **Timer and State Management Module:**

   - **Purpose:** This module is the central orchestrator of the game flow, managing the game's state (e.g., pre-game, countdown, round in progress, round end) and timing.
   - **Inputs:** Receives the "start game" signal from the User Interface Module.
   - **Outputs:** Triggers the Camera Input Module to start processing video for gesture detection. Updates the User Interface Module with countdown information. Informs the Game Logic Module with AI generated when a round begins.

- **Interactions:** Game Start/Restart: Initiates the game when the 'S' button is pressed.
- **Countdown:** Manages the 3-second countdown, which is crucial for player readiness and gesture timing.
- **Round Control:** Determines when a round begins and ends, effectively synchronizing the camera input, hand detection, and game logic.

3. Camera Input Module:

- **Purpose:** Responsible for capturing live video feed from the webcam.
- **Inputs:** Triggered by the Timer and State Management Module to start video capture.
- **Outputs:** Provides continuous video frames (video feed) to the Hand Detection Module.
- **Interactions:** This module acts as the bridge between the physical world (webcam) and the digital processing of the game. It continuously streams frames, which are then analyzed.

4. Hand Detection Module:

- **Purpose:** The core computer vision component. It processes the raw video feed to identify and track hands, and then interpret specific hand gestures.
- **Inputs:** Receives video feed from the Camera Input Module.
- **Outputs:** Identifies the player's move (Rock, Paper, or Scissors) and sends this information to the Game Logic Module with AI generated.
- **Gesture Interpretation:** Utilizes libraries like OpenCV and CvZone's HandTrackingModule to detect key points on the hand and infer the gesture (e.g., counting fingers).
- **Provides player move:** Once a gesture is confidently detected, it translates it into a game move.

5. Game Logic Module with AI generated:

- **Purpose:** This is the brain of the game. It orchestrates the actual game rules,

generates AI moves, determines outcomes, and manages scores.
Receives the player's move from the Hand Detection Module.
Receives signals from the Timer and State Management Module about round progression.

- **Outputs:** Generates an AI move.
  Calculates the round outcome (win/loss/tie).
  Updates the scores.
  Communicates the game outcome and scores to the User Interface Module for display.
- **Interactions:**
  AI Decision: Randomly generates AI's move for each round.
  Rule Enforcement: Compares the player's move with the AI's move based on Rock-Paper-Scissors rules.
  Score Management: Increments scores for the winner and maintains the overall game score
- **Modularity**: Highlight how each module is self-contained and performs a specific function.
  If one module needs changes (e.g., improving hand detection accuracy), it ideally shouldn't affect other modules significantly.
- **Clear Interfaces**: Discuss the well-defined interfaces between modules (e.g., Camera Input provides video feed to Hand Detection).
  This reduces coupling and makes the system more robust.
- **Scalability & Extensibility:** Explain how this architecture allows for future enhancements.
  For instance, adding more complex AI, different game modes, or alternative input methods would ideally only require changes to specific modules rather than a complete system overhaul.
- **Abstraction:** Emphasize that this diagram hides implementation details, focusing solely on what each component does and how it interacts at a high level.

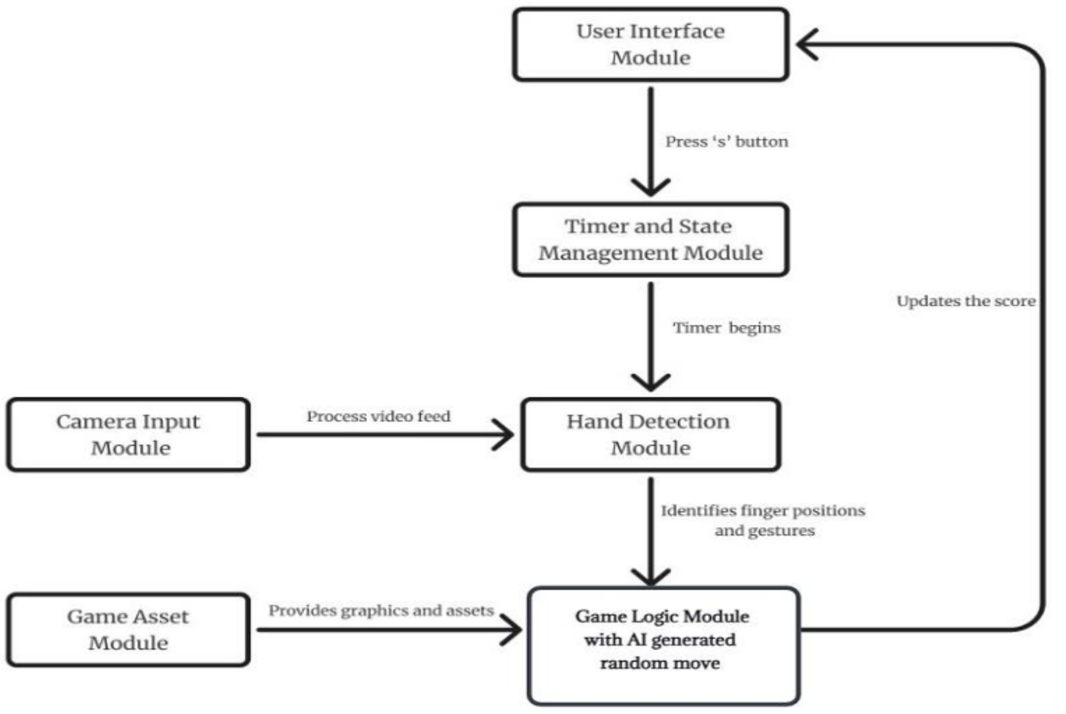

User Interface Module
Press 's' button
Timer and State Management Module
Timer begins
Updates the score
Camera Input Module
Process video feed
Hand Detection Module
Identifies finger positions and gestures
Game Asset Module
Provides graphics and assets
Game Logic Module with AI generated random move

Fig. 4.1.1 System Architecture

4.1.2 Technical Architecture:

The Technical Architecture diagram provides a more granular, implementation-focused view, showing the flow of information and specific technologies or components involved. It's less about abstract modules and more about the concrete data path.

1. User Interface (User Desktop & Keyboard):

- Component: Represents the physical user interaction point (desktop/laptop screen and keyboard).
- Flow: The user presses the 'S' key on the keyboard. This input Starts the game.
- Role: The entry point for user interaction to begin a game.

2. 'S' Button:

- Component: Represents the virtual 'S' button or the event triggered by pressing 'S'.
- Flow: Sends a signal to the Timer and State Management Module.

3. Timer and State Management Module:

- Component: Same as in System Architecture, but here it's depicted as part of the operational flow.

- Receives "Starts the game" signal.
- Initiates a 3 sec countdown.
- Communicates with the Camera Input Module to Start Stream.
- Communicates with the Game Logic Module to signal round start.
- Receives updates from the Game Logic Module to Calculates and updates the score.

4. Camera (Video stream):

- **Component**: The physical webcam device.
- **Flow:** Generates a continuous Video stream.
- **Role:** Captures the raw visual data of the player's hand.

5. Camera Input Module:

- **Component:** The software component responsible for interacting with the webcam.
- **Flow:** Receives Video stream from the Camera. Passes this stream to the Hand Detection Module.
- **Role:** Driver for the camera.

6. Hand Detection Module (depicted with hand gestures: Rock, Paper, Scissors):

- **Component:** The image processing and computer vision core.
- Receives Video stream from the Camera Input Module.
- Makes a gesture (interprets the hand shape into Rock, Paper, or Scissors).
- Sends the interpreted player's move to the Game Logic Module.
- **Role:** Translates visual input into game-relevant data. This is where OpenCV and CvZone's HandTrackingModule would be actively processing frames.

7. Game Logic Module:

- **Component:** The central processing unit for game rules and AI.
- Receives the player's gesture from the Hand Detection Module.
- Compares player and AI moves, determines the outcome.
- Communicates the outcome to the Game Asset Module for display.
- Sends score updates to the Timer and State Management Module (which then updates the UI).
- **Role:** Executes the game rules and integrates AI decisions.

8. AI:

- **Component**: Specifically highlights the Artificial Intelligence aspect within the Game Logic Module.
- **Flow:** When triggered by the Game Logic Module, it generates a random move.
- **Role:** Provides the opponent's move.

9. Game Asset Module:

- **Component:** The repository for visual elements.
- Receives game outcomes (e.g., "Rock wins", "Paper covers Rock") from the Game Logic Module.
- Shows graphics and assets (like overlays of the chosen moves, winner/loser text).
- This is the module that interfaces with the UI to render the visual feedback.
- **Role:** Provides all the visual cues for the game, ensuring a rich graphical experience using transparent PNG overlays.

10. UI (User Interface):

- **Component:** The displayed game window on the user's screen.
- **Flow:** Receives Calculates and updates the score from the Timer and State Management Module and receives Shows graphics and assets from the Game Asset Module.
- **Role:** Renders all game elements, including the webcam feed with overlays, scores, countdown, and game outcomes.
- **Data Flow:** Trace the path of data from the user's action (keypress) to the final visual output on the screen. This demonstrates a deep understanding of the system's operational flow.
- **Technology Stack Integration:** Explicitly mention where specific technologies like OpenCV, CvZone, and Python fit into this architecture. For example, Hand Detection Module relies heavily on OpenCV and CvZone.
- **Real-time Aspects:** Detail how the Camera Input Module and Hand Detection Module work in tandem to achieve real-time gesture recognition, which is critical for an interactive game. Discuss potential challenges like latency and how the system aims to minimize it**.**
- **Hardware-Software Interaction:** Show how the software modules (Camera Input Module) interact with physical hardware (Camera).
- **Event-Driven Nature:** Point out how the system reacts to events, such as a keypress triggering the game start, or a hand gesture being detected triggering a game logic calculation.
- **Visual Feedback Loop:** Explain the complete loop from gesture detection, through game logic, to updating the Game Asset Module and UI to provide immediate visual feedback to the player.

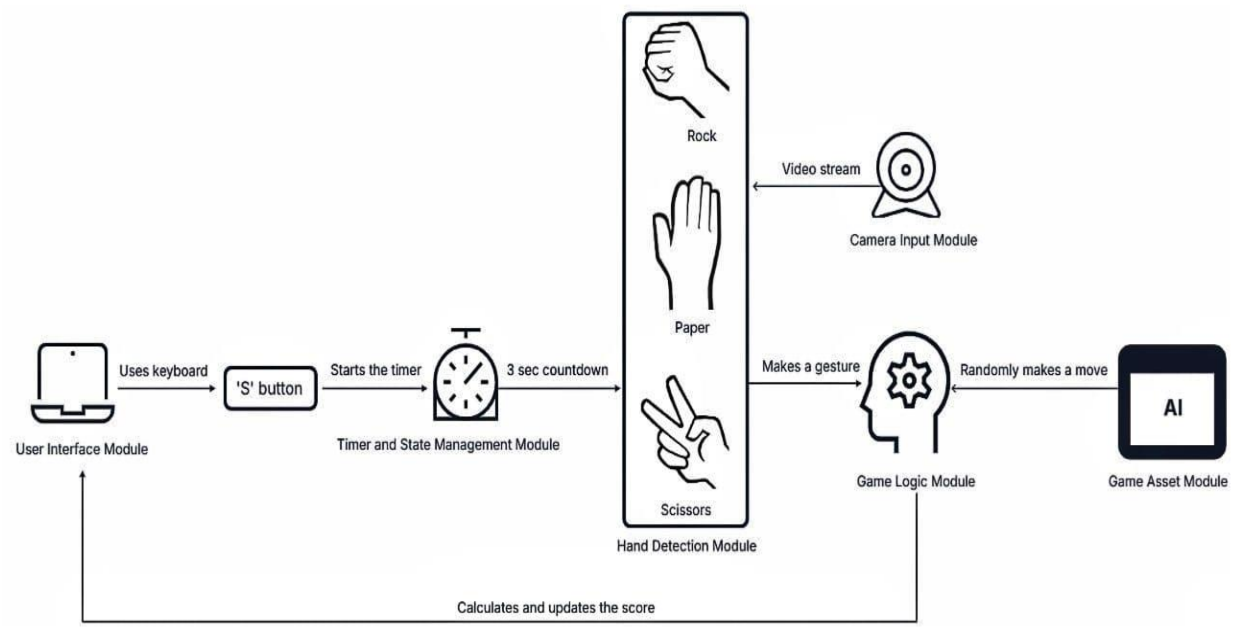


Fig.4.1.2 Technical Architecture

4.2 Modules

A module is a part of a program. Programs are composed of one or more independently developed modules. A module description provides detailed information about a module and its supported components. The included description is available directly by making environment check for supported components. The modules are

- Video Capture and Preprocessing
- Camera Input Module
- User Interface Module
- Hand Detection Module
- Time Management and State Module
- Game Logic Module
- Game Asset Module

1. Camera Input Module:

- **Camera Initialization**

  The camera initialization process involves connecting the application to the webcam using cv2.VideoCapture(0), where the 0 parameter ensures the default webcam is selected. This step sets up the hardware-software interaction, which is essential for capturing real-time video feed. Additionally, error-handling mechanisms are implemented to verify that the camera is successfully accessed, avoiding crashes due to hardware or permission issues. Without proper initialization, the entire gameplay dependent on live gesture input would not function.

- **Frame Capture**

  The frame capture mechanism employs cap.read() to continuously retrieve video frames. The process involves separating the returned data into two parts: a 8oolean value that indicates the success of the frame retrieval and the actual frame data. By continuously looping this process, the application maintains a seamless flow of real-time frames, allowing for uninterrupted gesture recognition. Any delay or lag in this module can adversely affect gameplay responsiveness, emphasizing the importance of optimized frame capture.

- **Frame Display**

  To ensure consistent user experience across devices, each captured frame is resized to predefined dimensions using cv2.resize(). This resizing ensures that all subsequent image processing operations are uniform, regardless of the original resolution of the webcam feed. Moreover, resizing the frame reduces computational load, as smaller frame sizes require fewer resources for processing. This balance between performance and quality is critical for real-time applications like this game.

2. User Interface Module

- **Text Display**

  Dynamic text updates, such as displaying scores and instructions, are achieved through cv2.putText(). This function allows developers to fine-tune text properties, including the font type, size, and positioning, to ensure visibility and readability. By updating the text dynamically, the game keeps players informed of the current status, creating an immersive experience. Proper placement and design of text overlays are vital, as they directly impact the clarity and usability of the interface.

- **Image Overlay**

  The visual elements of the game, such as rock, paper, and scissors icons, are integrated into the frame using image overlay techniques like cvzone.overlayPNG(). These graphical assets provide a visual representation of game moves, enriching the overall experience. The images are carefully blended with the video feed, ensuring they do not obscure key elements of the frame, such as the player’s gestures. This seamless integration adds a layer of polish and professionalism to the game.

- **Window Display**

  The processed video frames, complete with text and image overlays, are displayed in a dedicated game window using cv2.imshow(). This window acts as the primary interface for the player, presenting all the game elements in real time. Additionally, mechanisms are in place to allow the player to exit the game gracefully, typically by detecting a key press like q.

3. Hand Detection Module

- **Hand Detection Initialization**

  The hand detection system leverages pre-trained models like MediaPipe Hands, designed to identify hand landmarks efficiently. This initialization process includes setting detection thresholds and configuring the model to recognize multiple hands if needed. Accurate initialization ensures that the system is ready to detect gestures reliably under varying lighting and background conditions. Robust detection is crucial for gameplay accuracy, as misdetections can lead to incorrect outcomes.

- **Finger Position Tracking**

  Once hands are detected, their positions are analyzed using the coordinates of specific hand landmarks. By determining whether fingers are extended or folded, the system identifies the player’s hand configuration in real-time. This data is used to decode gestures accurately, forming the backbone of gesture-based gameplay. The ability to track finger positions even in motion highlights the sophistication of the detection model.

- **Gesture Recognition**

  Gesture recognition combines the detected hand configuration with predefined rules to classify gestures into rock, paper, or scissors. These classifications are achieved by analyzing the relative positions of the fingers and comparing them with known patterns. For instance, a closed fist corresponds to rock, an open palm corresponds to paper, and two extended fingers mimic scissors. The recognition accuracy is crucial, as errors directly affect the fairness and enjoyment of the game.

4. Time Management and State Module

- **Timer Initialization**

  A timer system is incorporated to divide gameplay into distinct phases, such as the player's input time and result display time. This ensures that the game progresses at a steady pace, preventing stalling or rushed interactions. The timer is implemented using Python's time module, allowing for precise countdowns.

- **State Transition Management**

  The game consists of multiple states, including gesture input, AI move generation, and result display. State transitions are handled programmatically to maintain a logical flow between phases. For example, once the player's gesture is detected, the system transitions to generating the AI's move before calculating the result. These transitions are timed to ensure smooth gameplay and prevent abrupt changes that might confuse the player.

- **Countdown Display**

  To enhance user engagement, a countdown is displayed during the gesture input phase. This countdown is shown on the screen using cv2.putText() and updates in real-time. The visual representation of time remaining adds an element of urgency, making the game more dynamic. The countdown display also ensures that players are aware of the game's pace and can act within the allotted time.

5. Game Logic Module

- **Random AI Move Generation**

  The computer's move is generated randomly using Python's random.choice() function, ensuring unpredictability in gameplay. This randomness simulates the behavior of a real opponent, keeping the game fair and challenging. The chosen move is represented visually on the screen, adding excitement and suspense for the player. By incorporating randomness, the system ensures that no two games are the same, increasing replayability.

- **Result Determination**

  Once both moves are made, the system compares them using predefined rules to determine the winner. For example, rock beats scissors, scissors beats paper, and paper beats rock. This comparison is implemented using a series of conditional statements that cover all possible outcomes. The result is then displayed on the screen, providing immediate feedback to the player.

- **Score Management**

  Scores are tracked throughout the game, updating after each round based on the outcome. The system maintains separate scores for the player and the computer, ensuring transparency. These scores are displayed prominently on the screen, allowing players to monitor their progress. Proper score management adds a competitive element, motivating players to perform better.

6. Game Asset Module

- **Image Loading**

  Game assets, such as gesture icons and background images, are preloaded into memory using cv2.imread(). These assets are stored in a dedicated folder, allowing for easy retrieval and organization. Preloading ensures that the assets are readily available during gameplay, minimizing delays. High-quality images enhance the visual appeal of the game, making it more engaging for players.

- **Resource Management**

All assets, including images, fonts, and configurations, are systematically organized within the project directory. This organization simplifies development and ensures that all resources are easily accessible. Proper resource management also reduces the risk of runtime errors caused by missing files. Efficient handling of resources contributes to the overall stability and performance of the application.

- **Asset Positioning**

  The positioning of game assets is carefully planned to create a clean and intuitive user interface. Pixel coordinates are used to place each element precisely, ensuring they are visible and do not overlap with other components. This meticulous placement enhances the visual appeal of the game and improves usability. Clear and well-organized UI elements contribute to a seamless player experience.

4.3 UML Diagrams

4.3.1 Use Case Diagram:

Purpose: The Use Case Diagram illustrates the high-level functionality of the system and how users (actors) interact with it. It defines what the system does from an external point of view.

Actors:

- **Player:** The human user interacting with the game.
- **Administrator:** Likely a user with elevated privileges to manage game settings or data.

Main Use Cases (Functions):

The diagram seems to categorize the use cases, possibly indicating different modules or stages of interaction.

User Input:

- "Q" to Restart: Allows the player to restart the game.

- "S" to Start: Initiates the game.
- "Q" to Exit: Allows the player to quit the game.
- **Enter Score Limit:** Player provides the target score to win.
- **Enter Player Name:** Player inputs their name.
- **Enter AI Name:** Player inputs the AI's name.visible services that the system provides in the context of its environment

Game Management:

- **Score and Time Management:** Handles updating and displaying scores and the game timer.
- **Game States Management**: Controls the different phases of the game (e.g., initial setup, active play, round end, game end).
- **Detect Winner:** Determines if a player has reached the score limit and won.
- **Display Overall Winner:** Shows the winner of the entire game.
- **Hand Detection:** (This is where the computer vision aspect comes in)
- **Hand Detection:** Identifies the presence of hands in the camera feed.
- **Finger Position Tracking:** Tracks the position of fingers.
- **Hand Gestures:** Recognizes specific hand gestures (e.g., rock, paper, scissors).
- **Player Move:** Translates hand gestures into a game move for the player.

AI Management:

- **Random AI move generation:** The AI randomly selects a move.
- **Display AI Move:** Shows the AI's chosen move.

System Interactions (Implicit/Supporting):

- **Camera Input:** The system needs to access and process camera feed.
- **Frame Capture:** Captures individual frames from the camera.

- **Frame Processing:** Processes the captured frames (e.g., for hand detection)
- **Game Assets:** Likely refers to images or resources for displaying moves (e.g., Resources/1.png).
- **Display Assets:** Displays the visual elements of the game.

How to use it for project work:

- **Scope Definition:** Clearly understand what features your game will have. Each use case represents a distinct functional requirement.
- **Requirement Gathering:** Convert each use case into detailed functional requirements. For example, "Enter Score Limit" would lead to requirements like: "The system shall allow the user to input a positive integer as the score limit."
- **Test Case Generation:** Each use case can be a basis for testing. For "Player Move," you'd test if different hand gestures correctly translate to rock, paper, or scissors.
- **Module Design:** The grouping of use cases suggests potential modules in your system (e.g., a "Game Management" module, a "Hand Detection" module).
- **User Manual Creation:** The use cases directly map to actions a user can perform, providing a good outline for a user manual.

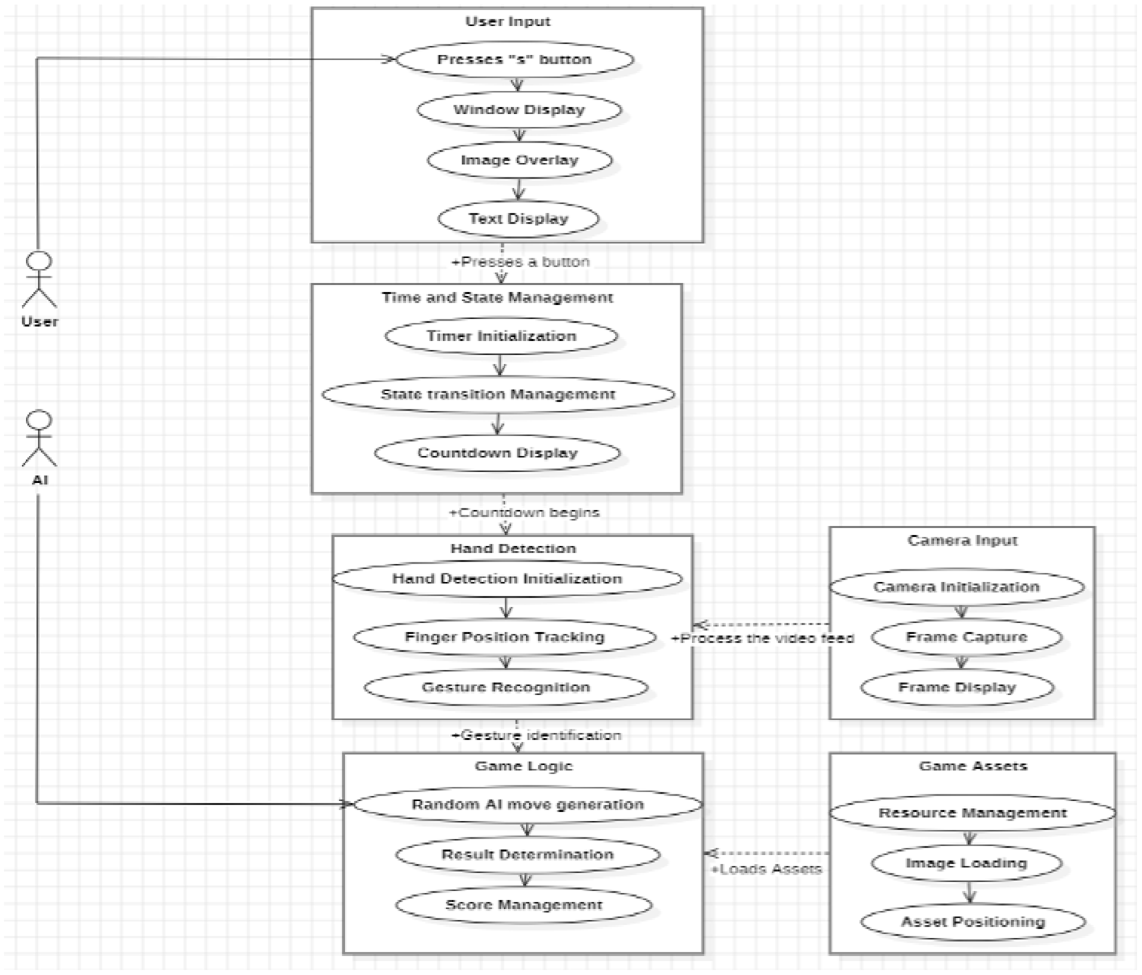


Fig. 4.3.1 Use Case diagram

4.3.2 Class Diagram:

Purpose: The Class Diagram illustrates the static structure of the system, showing the classes, their attributes (data), operations (methods/functions), and the relationships between them.

Key Classes (Based on common patterns in such games and vision systems):

Virtual Keyboard:

- **Attributes:** Potentially key List (list of keys), current Keys (layout, e.g., alpha or numeric).
- **Operations:** create Key List(), draw Key board(), check Key Press(). This class encapsulates all virtual keyboard logic.

- **Hand Detector:** (Likely from cvzone)
- **Attributes:** detection Con, max Hands.
- **Operations:** find Hands(), fingers Up(). This class is responsible for hand and finger tracking.

Game Logic (or Game Manager):

- **Attributes:** player Name, AI Name, score Limit, scores, state Result, initial Time, timer, game In Progress, round Completed, winner Declared, winner Name, player Move, AI Move, AIImage.

Operations:

- Game Flow: startGame(), restartGame(), quitGame().
- Round Logic: determineWinner(), updateScores().
- State Management: updateGameState().
- AI Logic: generateAIMove().
- DisplayManager (or integrated into GameLogic): drawKeyboard(img), drawPurpleButton(img).

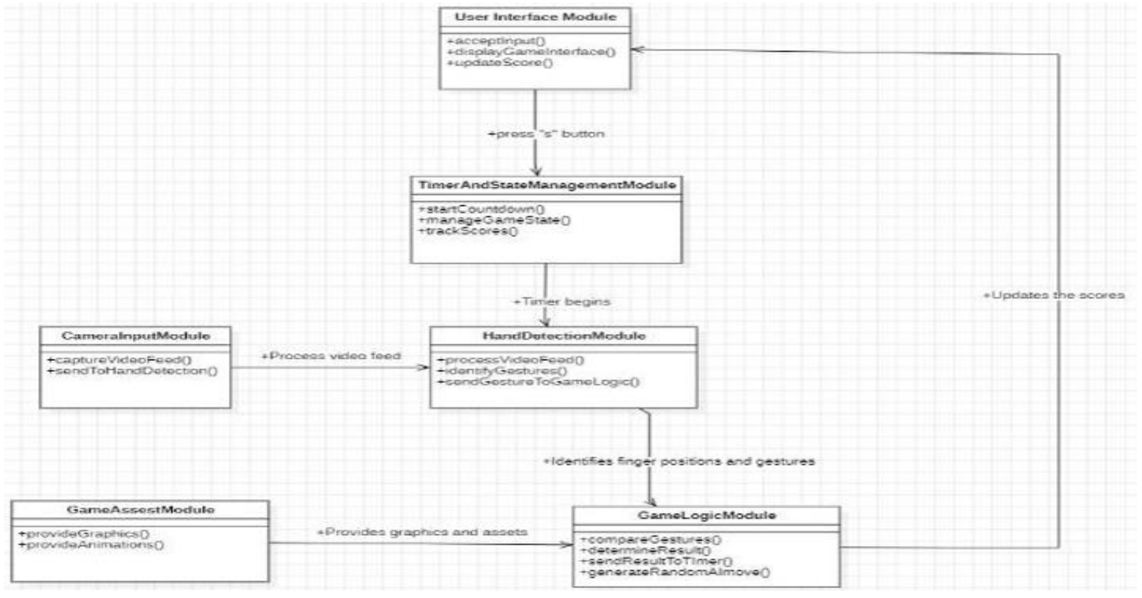


Fig. 4.3.2 Class diagram

4.3.3 Sequence Diagram:

Purpose: The Sequence Diagram shows the interactions between objects in a time-ordered sequence. It visualizes the flow of messages (method calls) between different objects to achieve a specific functionality. It's excellent for understanding the dynamic behavior of a system and for debugging interactions.

Key Elements and Interactions:

- **Lifelines (Vertical Dashed Lines):** Represent individual objects or instances participating in the interaction over time. The diagram shows:
  1. User
  2. Virtual Keyboard
  3. Game Manager (likely the central control class)
  4. Hand Detector
  5. Camera Manager
  6. Game Assets (representing image files, etc.)
  7. AI Module
- **Activation Bars (Rectangles on Lifelines):** Indicate the period during which an object is performing an action (its method is active).
- **Messages (Horizontal Arrows):** Represent method calls or signals passed between objects.
  1. User -> Game Manager: startGame(), restartGame(), quitGame().
  2. User -> Virtual Keyboard: typeInput() (representing keyboard interaction).
  3. Virtual Keyboard -> Game Manager: sendPlayerName(), sendAIName(), sendScoreLimit().
  4. Game Manager -> Camera Manager: initializeCamera(), captureFrame().

5. Game Manager -> Hand Detector: detectHands(frame), recognizeGesture(handData).
6. Game Manager -> AI Module: generateAIMove().
7. Game Manager -> Game Assets: loadAIMoveImage(aiMove).
8. Game Manager -> Display Manager (implied): displayUI(), displayPlayerMove(), displayAIMove(), displayScores(), displayWinner().

Flow of a Game Round (inferred):

- **Game Setup:** User types in game parameters via the Virtual Keyboard, which sends them to Game Manager.
- **Start Game:** User sends startGame() to Game Manager.
- **Round Initialization:** Game Manager might initializeCamera() via Camera Manager.

Game Loop:

- Camera Manager captureFrame().
- Hand Detector detectHands() and recognizeGesture().
- AI Module generateAIMove().
- Game Manager receives player's move and AI's move.
- Game Manager loads aiMoveImage from Game Assets.
- Game Manager performs determineWinner() and updateScores().
- Game Manager sends messages to Display Manager to update the screen.

End of the Game: If a winner is detected, Game Manager displays the winner and possibly waits for a restart/quit.

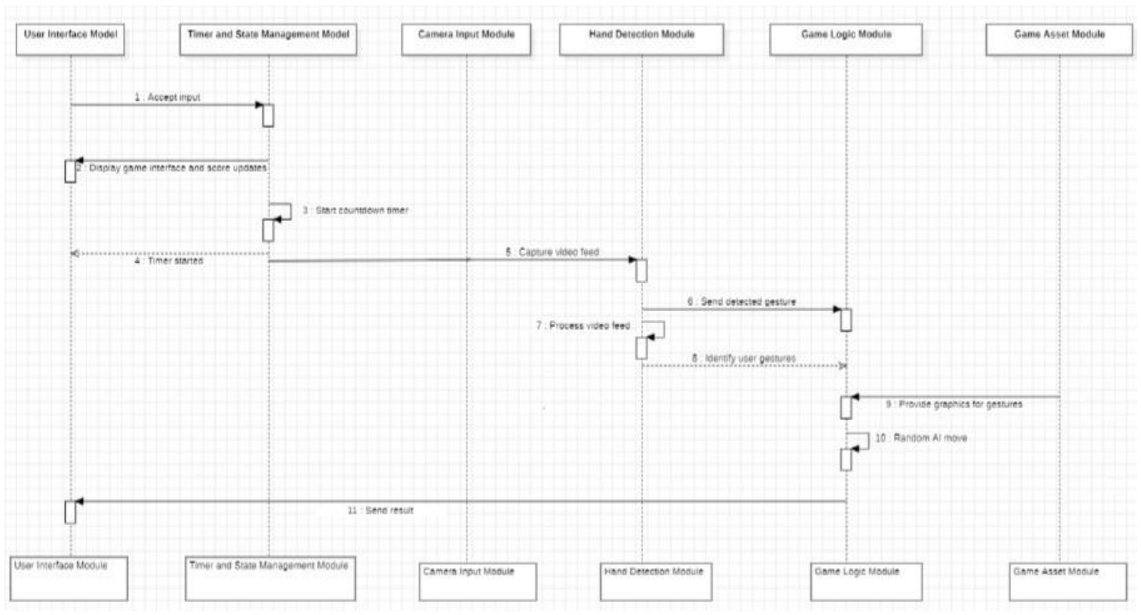


Fig. 4.3.3 Sequence diagram

4.3.4 Activity Diagram:

Purpose: The Activity Diagram depicts the flow of activities within a process or use case. It's similar to a flowchart, showing the sequence of actions and decisions that occur. It's particularly useful for modeling business processes, workflows, or the internal logic of a complex operation.

Breakdown of the Diagram:

- **Start Node (Black Circle):** Represents the beginning of the activity.
- **User Input:** The initial state where the system waits for player interaction (e.g., entering names, starting the game).
- **Start Timer:** The game timer begins.
- **AI generates random move:** The AI decides its move.

Right Branch (Camera/Vision System):

- **Initialize Camera:** Sets up the webcam.
- **Capture Frame:** Captures a single image from the camera.

- **Display Frame:** Shows the captured image on the screen.
- **Hand Detection:** Identifies hands in the frame.
- **Recognize Gesture:** Interprets the hand's posture as a Rock, Paper, or Scissors move.
- **Synchronized Activities (Join):** The two parallel branches (AI move generation and hand gesture recognition) converge, indicating that both inputs are needed before proceeding.
- **Perform game logic:** This is a high-level activity that encompasses the core game rules.
- **Determine the score:** Compares the player's and AI's moves to decide the winner of the round.
- **Update Score:** Increments the score of the winning party.
- **End Node (Bullseye):** Represents the completion of the activity flow.

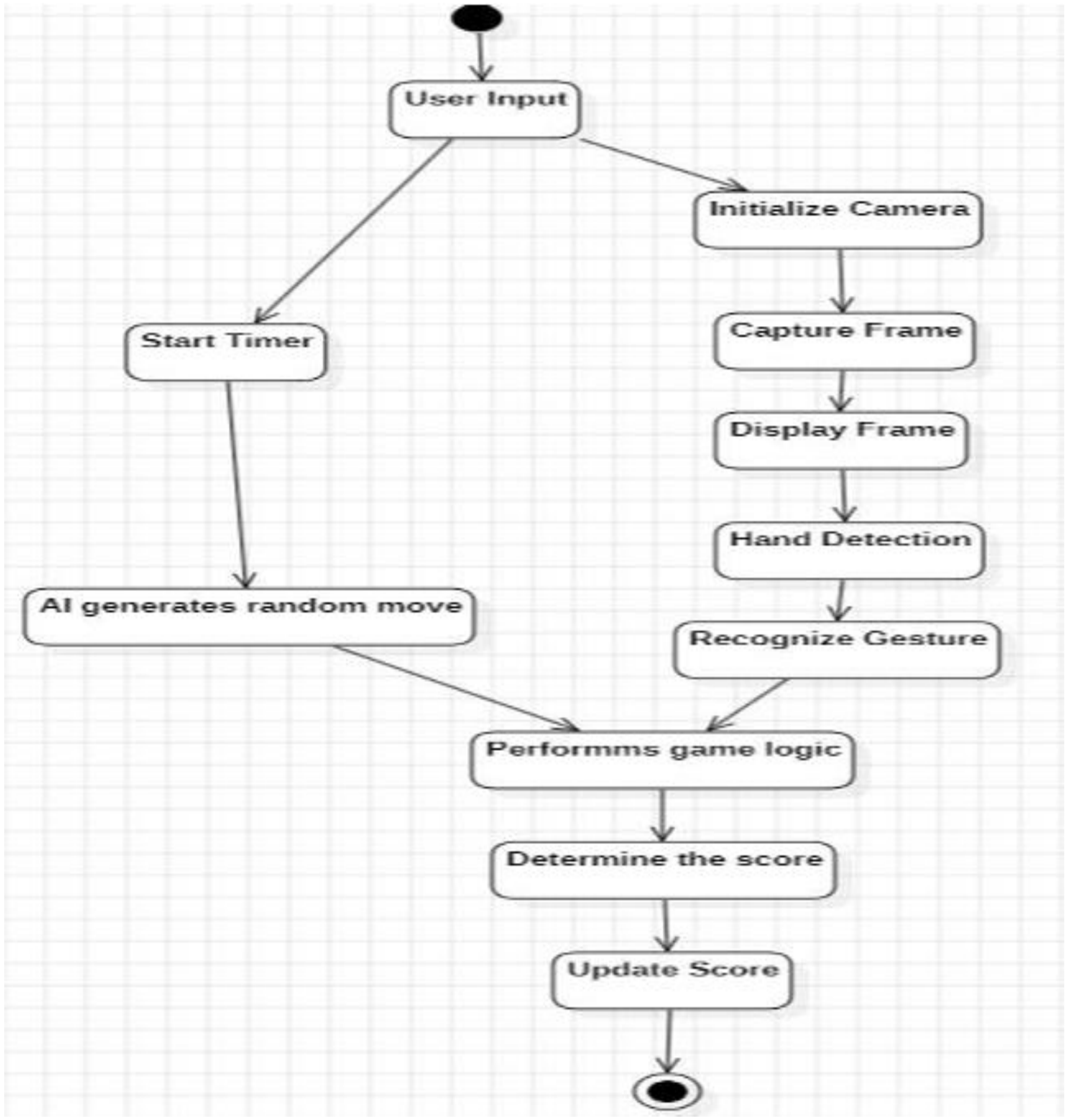


Fig. 4.3.4 Activity diagram

5. Implementation

5. Implementation

5.1 Libraries

5.1.1 cv2(OpenCV)

OpenCV is an open-source computer vision and image processing library used for real-time image and video capture and manipulation.

- **cv2.VideoCapture(0)** – Captures video feed from the default webcam.
- **cap.set(3, 1280)** – Sets the width of the video frame to 1280 pixels.
- **cap.set(4, 720)** – Sets the height of the video frame to 720 pixels.
- **cv2.rectangle(img, (x, y), (x + w, y + h), color, thickness)** – Draws a rectangle on the image.
- **cv2.putText(img, text, position, font, scale, color, thickness)** – Adds text overlay on the image.
- **cv2.imread(path, flag)** – Reads an image from the specified path.
- **cv2.imshow(windowName, image)** – Displays an image in a window.
- **cv2.waitKey(delay)** – Waits for a key press for the specified delay in milliseconds.
- **cv2.destroyAllWindows()** – Closes all OpenCV windows.

5.1.2 Cvzone

Cvzone is a wrapper library around OpenCV that simplifies common computer vision tasks and provides higher-level interfaces.

- **cvzone.overlayPNG(img, overlayImg, pos)** – Overlays a transparent PNG image over another image at the given position.

5.1.3 cvzone.HandTrackingModule.HandDetector

A part of the cvzone library, HandDetector simplifies hand detection using MediaPipe and provides landmark-based hand tracking.

- **HandDetector(detectionCon=0.8, maxHands=1)** – Initializes hand detector with confidence threshold and max hands to track.
- **detector.findHands(img)** – Detects hands and returns the processed image and hand landmark list.
- **detector.fingersUp(hand)** – Returns a list of booleans (0 or 1) indicating which fingers are up.

5.1.4 Time

The time module is used for tracking time, delays, and intervals in seconds.

- **time.time()** – Returns the current time in seconds since the epoch.
- **time.sleep(seconds)** – Pauses the program execution for the specified number of seconds.

5.1.5 Random

The random module provides functions to generate random numbers and selections.

- random.randint(a, b) – Returns a random integer N such that a <= N <= b, used to simulate AI's random move.

.

5.2 Pseudo code

```
#importing necessary libraries:
import cv2
import cvzone
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import time
import random
cap = cv2.VideoCapture(0)
cap.set(3, 1280)
cap.set(4, 720)
detector = HandDetector(detectionCon=0.8, maxHands=1)
# Game variables
playerName = ""
cursorVisible = True
lastCursorToggle = time.time()
invalidInput = False
invalidTimer = 0
aiName = ""
scoreLimit = 0
finalText = ""
nameStage = 0  # 0 = Score Limit, 1 = AI Name, 2 = Player Name
keyboardVisible = True
startGame = False
stateResult = False
scores = [0, 0]
initialTime = 0
```

```
timer = 0
gameInProgress = False
waitForNextRound = True
aiMove = None
aiImage = None
roundCompleted = False  # Prevents continuous moves until 'S' is pressed again
winnerDeclared = False  # To show winner message once and quit after
# Keyboard layouts
alphaKeys = [["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"],
             ["A", "S", "D", "F", "G", "H", "J", "K", "L"],
             ["Z", "X", "C", "V", "B", "N", "M", "<", "SPACE", "ENTER"]]
numKeys = [["7", "8", "9"],
           ["4", "5", "6"],
           ["1", "2", "3"],["0", "<", "ENTER"]]
currentKeys = numKeys  # Start with number keys for score limit
def createKeyList(keys):
    keyList = []
    for i in range(len(keys)):
        row = keys[i]
        x_offset = 50
        for key in row:
            if key == "SPACE":
                w = 170  # Wider for SPACE
            elif key == "ENTER":
                w = 130  # Wider for ENTER
            else:
                w = 85
            keyList.append({'pos': (x_offset, 100 * i + 50), 'text': key, 'size': (w, 85)})
        x_offset += w + 15  # Add gap between keys
    return keyList
keyList = createKeyList(currentKeys)
def drawKeyboard(img):
    for key in keyList:
        x, y = key['pos']
        w, h = key['size']
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 255), cv2.FILLED)
        text_offset_x = 20 if key['text'] not in ["SPACE", "ENTER"] else 10
        cv2.putText(img, key['text'], (x + text_offset_x, y + 55),
                    cv2.FONT_HERSHEY_PLAIN, 2.5, (255, 255, 255), 3)
    return img
def checkKeyPress(fingerPos):
    for key in keyList:
        x, y = key['pos']
        w, h = key['size']
        if x < fingerPos[0] < x + w and y < fingerPos[1] < y + h:
            return key['text']
    return None
def drawPurpleButton(img):
    x, y, w, h = 1100, 50, 150, 80
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 255), cv2.FILLED)
    cv2.putText(img, "KB", (x + 10, y + 55), cv2.FONT_HERSHEY_PLAIN, 3, (255, 255,
255), 3)
    return (x, y, w, h)
def isInButton(pos, buttonRect):
    x, y, w, h = buttonRect
    return x < pos[0] < x + w and y < pos[1] < y + h
```

```
while True:
    success, img = cap.read()
    #img = cv2.flip(img, 1)
    imgScaled = cv2.resize(img, (0, 0), None, 0.875, 0.875)
    imgScaled = imgScaled[:, 80:480]
    hands, img = detector.findHands(img)
    # Update keyboard layout based on stage
    if keyboardVisible:
        if nameStage == 0:
            if currentKeys != numKeys:
                currentKeys = numKeys
                keyList = createKeyList(currentKeys)
        else:
            if currentKeys != alphaKeys:
                currentKeys = alphaKeys
                keyList = createKeyList(currentKeys)
        img = drawKeyboard(img)
        if hands:
            lmList = hands[0]["lmList"]
            indexFinger = lmList[8]
            middleFinger = lmList[12]
            keyPressed = None
            for key in keyList:
                x, y = key['pos']
                w, h = key['size']  # Use actual key size
                if x < indexFinger[0] < x + w and y < indexFinger[1] < y + h:
                    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), cv2.FILLED)
                    text_offset_x = 20 if key['text'] not in ["SPACE", "ENTER"] else 10
                    cv2.putText(img, key['text'], (x + text_offset_x, y + 55),
                                cv2.FONT_HERSHEY_PLAIN, 2.5, (255, 255, 255), 3)
                    if abs(indexFinger[1] - middleFinger[1]) < 30:
                        keyPressed = key['text']
                        time.sleep(0.4)
                        break
            if keyPressed is not None:
                if keyPressed == "<":
                    finalText = finalText[:-1]
                elif keyPressed == "SPACE":
                    finalText += " "
                elif keyPressed == "ENTER":
                    if nameStage == 0:
                        if finalText.isdigit() and int(finalText) > 0:
                            scoreLimit = int(finalText)
                            finalText = ""
                            nameStage = 1
                            invalidInput = False
                        else:
                            invalidInput = True
                            invalidTimer = time.time()
                    elif nameStage == 1:
                        if finalText.strip():
                            aiName = finalText.strip()
                            finalText = ""
                            nameStage = 2
                            invalidInput = False
                        else:
                            invalidInput = True
                            invalidTimer = time.time()
                    elif nameStage == 2:
                        if finalText.strip():
                            if finalText.strip().lower() == aiName.strip().lower():
                                invalidInput = True
                                invalidTimer = time.time()
                                finalText = ""  # Optional: clear input so user retypes
                            else:
                                playerName = finalText.strip()
                                finalText = ""
                                nameStage = 3
                                keyboardVisible = False
                                invalidInput = False
                        else:
                            invalidInput = True
                            invalidTimer = time.time()
                else:
                    if nameStage == 0:
                        if keyPressed.isdigit():
                            finalText += keyPressed
                    else:
                        finalText += keyPressed
    else:
        if not keyboardVisible:
            buttonRect = drawPurpleButton(img)
            if hands:
                lmList = hands[0]["lmList"]
                indexFinger = lmList[8]
                middleFinger = lmList[12]
                if isInButton(indexFinger, buttonRect):
                    if abs(indexFinger[1] - middleFinger[1]) < 30:
                        # Draw pressed button in green
                        x, y, w, h = buttonRect
                        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), cv2.FILLED)
                        cv2.putText(img, "KB", (x + 10, y + 55), cv2.FONT_HERSHEY_PLAIN, 3,
(255, 255, 255), 3)
                        keyboardVisible = True
                        time.sleep(0.5)
    if nameStage == 0:
        promptText = "Enter SCORE LIMIT and press ENTER"
        displayText = finalText
    elif nameStage == 1:
        promptText = "Enter AI NAME and press ENTER"
        displayText = finalText
    elif nameStage == 2:
        promptText = "Enter YOUR NAME and press ENTER"
        displayText = finalText
    else:
        promptText = "Press 'S' to star the game, 'R' to restart and 'Q' to quit the game"
        displayText = f"{playerName} VS {aiName} | Score to win: {scoreLimit}"
    if invalidInput and time.time() - invalidTimer < 2:
        if nameStage == 2 and finalText == "":
            cv2.putText(img, "AI and Player name cannot be the same!", (300, 400),
cv2.FONT_HERSHEY_PLAIN, 2.5,
                        (0, 0, 0), 3)
        else:
            cv2.putText(img, "Enter valid input!", (500, 100), cv2.FONT_HERSHEY_PLAIN,
2.5, (0, 0, 0), 3)
    cv2.putText(img, promptText, (50, 50), cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 0), 3)
    cv2.rectangle(img, (40, 440), (1240, 500), (0, 0, 0), cv2.FILLED)
    # Blinking cursor logic
    if time.time() - lastCursorToggle > 0.5:
        cursorVisible = not cursorVisible
        lastCursorToggle = time.time()
    # Append cursor if visible
    displayTextWithCursor = displayText + "|" if cursorVisible else displayText
    cv2.putText(img, displayTextWithCursor, (60, 490), cv2.FONT_HERSHEY_PLAIN, 3,
(255, 255, 255), 3)
    if startGame:
        cv2.putText(img, f"{playerName}: {scores[0]}", (50, 650),
cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)
        cv2.putText(img, f"{aiName}: {scores[1]}", (800, 650),
cv2.FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 3)
    if startGame:
        if not stateResult:
            timer = time.time() - initialTime
            cv2.putText(img, f"Timer: {int(timer)}", (500, 650),
cv2.FONT_HERSHEY_PLAIN, 3, (0, 255, 255), 3)
            if timer > 3:
                stateResult = True
                roundCompleted = True  # Mark round as done
                playerMove = None
                if hands:
```

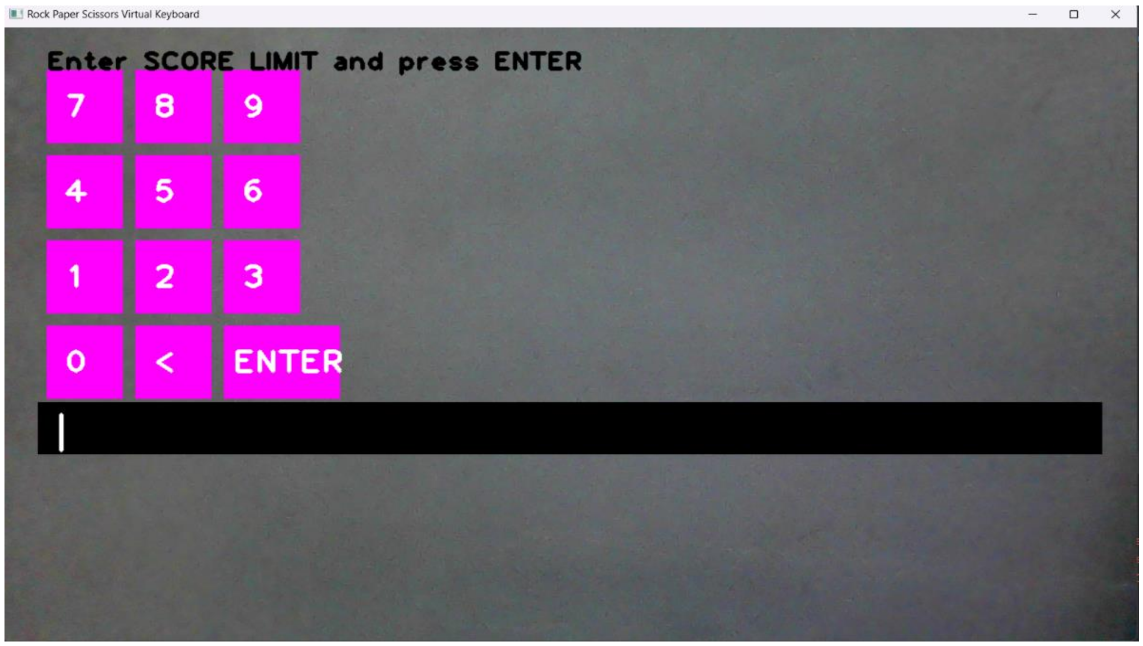
```
                hand = hands[0]
                fingers = detector.fingersUp(hand)
                if fingers == [0, 0, 0, 0, 0]:
                    playerMove = 1  # Rock
                elif fingers == [1, 1, 1, 1, 1]:
                    playerMove = 2  # Paper
                elif fingers == [0, 1, 1, 0, 0]:
                    playerMove = 3  # Scissors
            aiMove = random.randint(1, 3)
            aiImage = cv2.imread(f'Resources/{aiMove}.png',
cv2.IMREAD_UNCHANGED)
            # Only show AI move on left side if it's not the final winning round
            if not ((scores[0] + 1 >= int(scoreLimit)) or (scores[1] + 1 >= int(scoreLimit))):
                img = cvzone.overlayPNG(img, aiImage, (220, 200))
            if playerMove is None:
                scores[1] += 1  # No hand detected, AI gets the point
            else:
                if (playerMove == 1 and aiMove == 3) or (playerMove == 2 and aiMove == 1)
or (
                        playerMove == 3 and aiMove == 2):
                    scores[0] += 1
                elif (aiMove == 1 and playerMove == 3) or (aiMove == 2 and playerMove ==
1) or (
                        aiMove == 3 and playerMove == 2):
                    scores[1] += 1
            if scores[0] >= int(scoreLimit):
                winnerDeclared = True
                winnerName = playerName
            elif scores[1] >= int(scoreLimit):
                winnerDeclared = True
                winnerName = aiName
        else:


            if 'aiImage' in locals():
                img = cvzone.overlayPNG(img, aiImage, (990,400))  # Match new position
    if not keyboardVisible and not startGame:
        cv2.putText(img, "Press Purple KEYBOARD button to open virtual keyboard", (50,
690), cv2.FONT_HERSHEY_PLAIN, 2, (200, 0, 200), 2)
    if not keyboardVisible and startGame:
        cv2.putText(img, "Game running - Press virtual 'R' to restart, 'Q' to quit", (50, 690),
cv2.FONT_HERSHEY_PLAIN, 2, (200, 0, 200), 2)
    if keyboardVisible and nameStage > 2:
        if hands:
            lmList = hands[0]["lmList"]
            indexFinger = lmList[8]
            middleFinger = lmList[12]
            keyPressed = None
            for key in keyList:
                x, y = key['pos']
                w, h = key['size']  # Use actual key size
                if x < indexFinger[0] < x + w and y < indexFinger[1] < y + h:
                    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), cv2.FILLED)
                    text_offset_x = 20 if key['text'] not in ["SPACE", "ENTER"] else 10
                    cv2.putText(img, key['text'], (x + text_offset_x, y + 55),
                                cv2.FONT_HERSHEY_PLAIN, 2.5, (255, 255, 255), 3)
                    if abs(indexFinger[1] - middleFinger[1]) < 30:
```

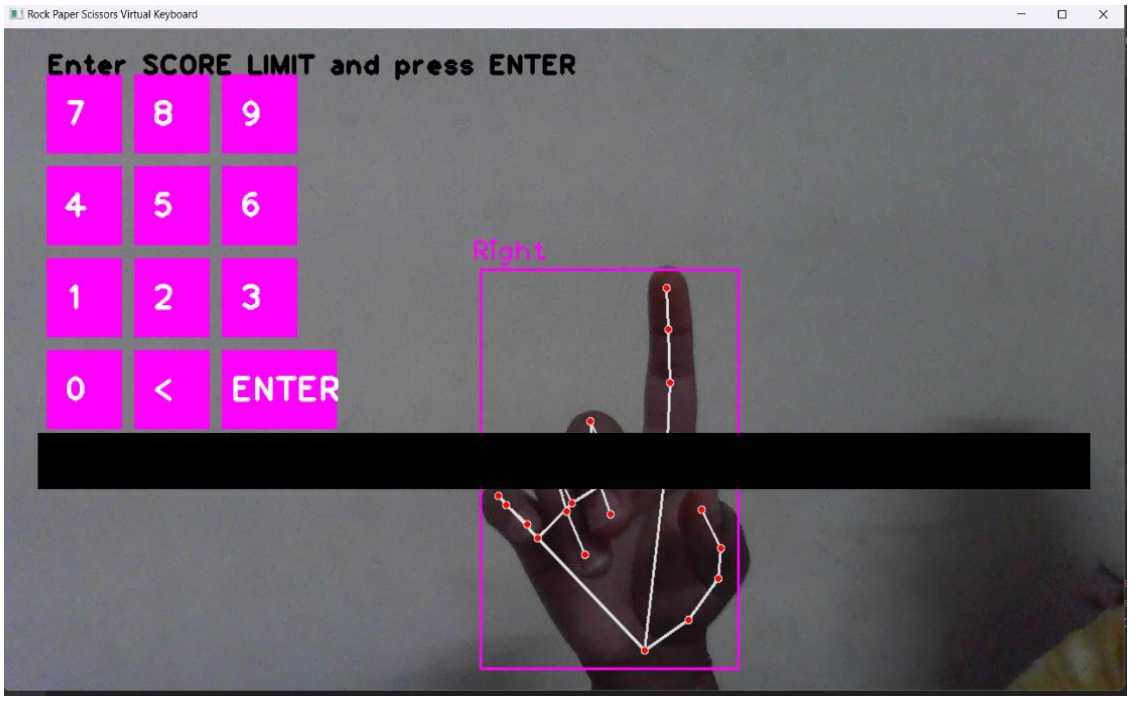
```
                keyPressed = key['text']
                time.sleep(0.4)
                break
        if keyPressed is not None:
            if keyPressed.lower() == 's':
                if not startGame:
                    startGame = True
                    roundCompleted = False
                    stateResult = False
                    initialTime = time.time()
                elif roundCompleted:
                    startGame = True
                    roundCompleted = False
                    stateResult = False
                    initialTime = time.time()
            elif keyPressed.lower() == 'r':
                startGame = False
                stateResult = False
                roundCompleted = False
                scores = [0, 0]
                playerName = ""
                aiName = ""
                scoreLimit = ""
                finalText = ""
                nameStage = 0
                currentKeys = numKeys
                keyList = createKeyList(currentKeys)
                keyboardVisible = True
            elif keyPressed.lower() == 'q':
                break
    if winnerDeclared:
        if aiImage is not None:
            img = cvzone.overlayPNG(img, aiImage, (950, 400))  # ← Show AI move at bottom
 right
        cv2.rectangle(img, (200, 250), (1000, 450), (0, 0, 0), cv2.FILLED)
        cv2.putText(img, f"{winnerName} has won!!", (300, 400),
                    cv2.FONT_HERSHEY_PLAIN, 5, (0, 255, 0), 5)
        cv2.imshow("Rock Paper Scissors Virtual Keyboard", img)
        cv2.waitKey(2000)  # Wait 2 seconds
        break # Exit game
    cv2.imshow("Rock Paper Scissors Virtual Keyboard", img)
    key = cv2.waitKey(1)
    if key == 27:
        break
 cap.release()
cv2.destroyAllWindows()
```

6. Screenshots

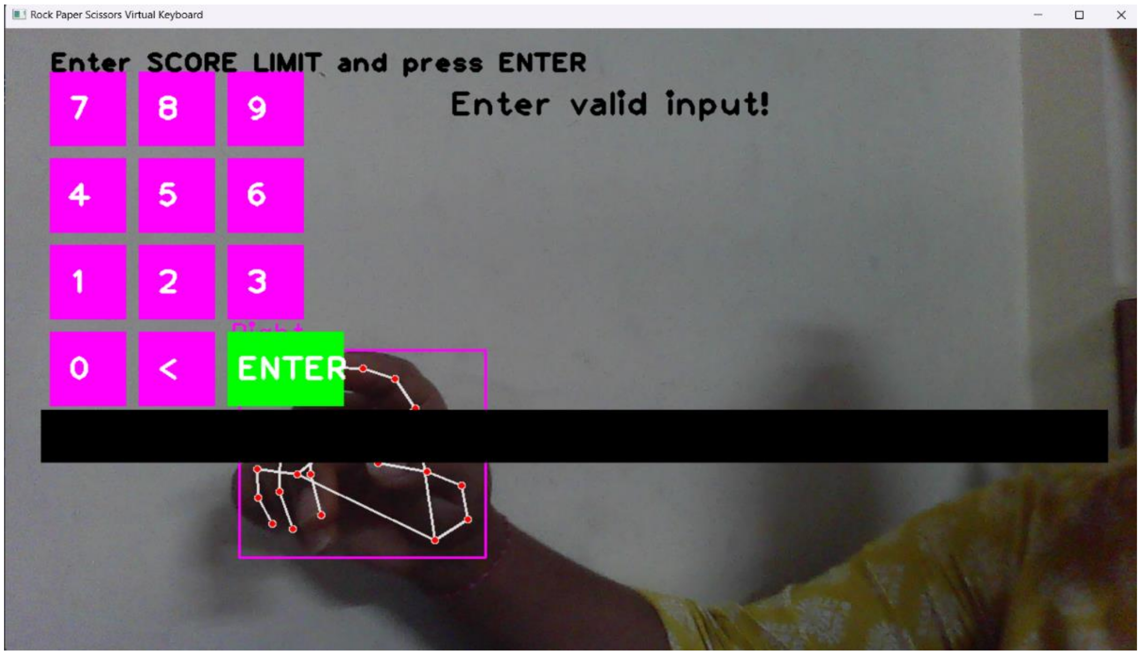
6. Screenshots


Rock Paper Scissors Virtual Keyboard
Enter SCORE LIMIT and press ENTER
7
8
9
4
5
6
1
2
3
0
<
ENTER

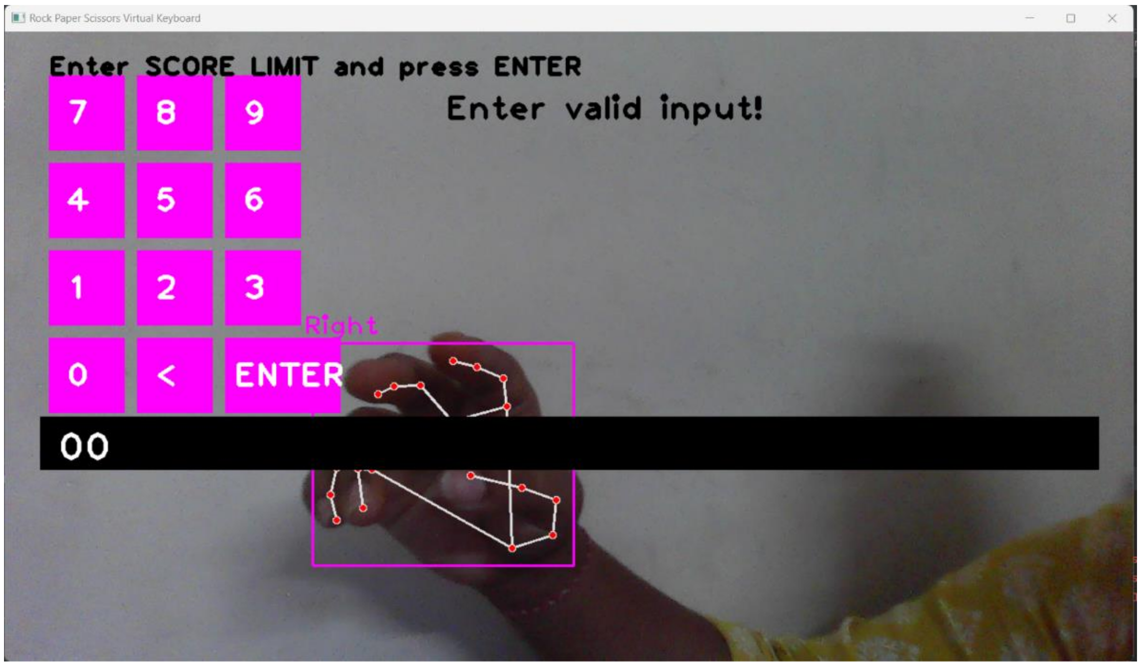
Screenshot 6.1 Score Limit Layout


Rock Paper Scissors Virtual Keyboard
Enter SCORE LIMIT and press ENTER
7
8
9
4
5
6
1
2
3
0
<
ENTER
Right

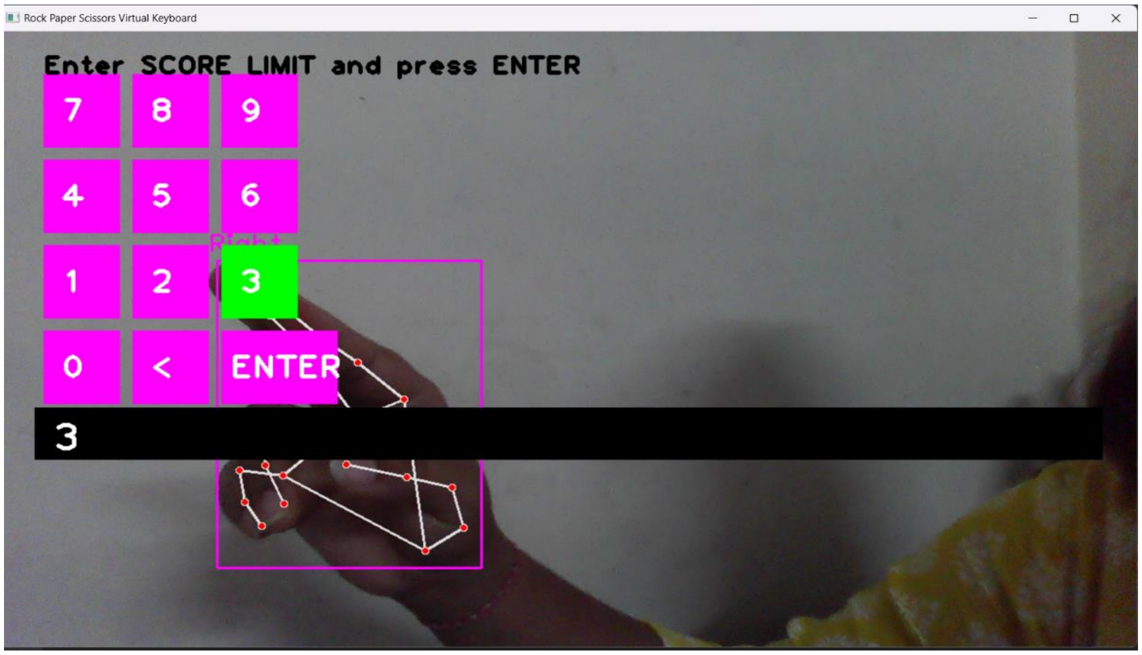
Screenshot 6.2 Hand identification


Rock Paper Scissors Virtual Keyboard
Enter SCORE LIMIT and press ENTER
Enter valid input!
7
8
9
4
5
6
1
2
3
0
<
ENTER

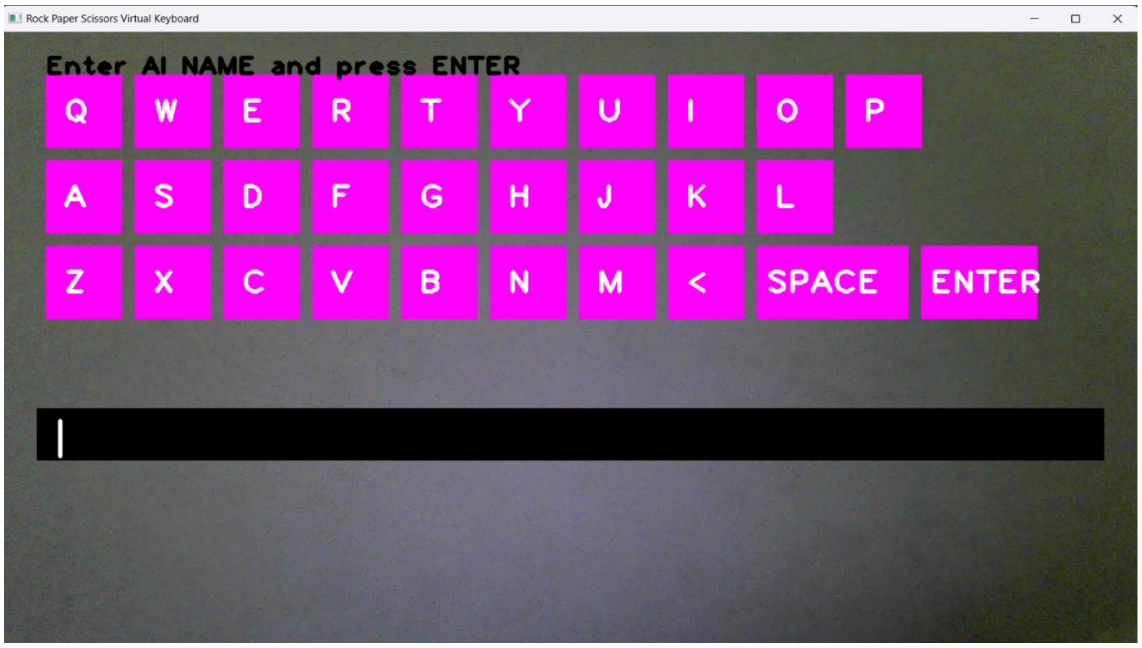
Screenshot 6.3 Empty Score limit


Rock Paper Scissors Virtual Keyboard
Enter SCORE LIMIT and press ENTER
Enter valid input!
7
8
9
4
5
6
1
2
3
Right
0
<
ENTER
00

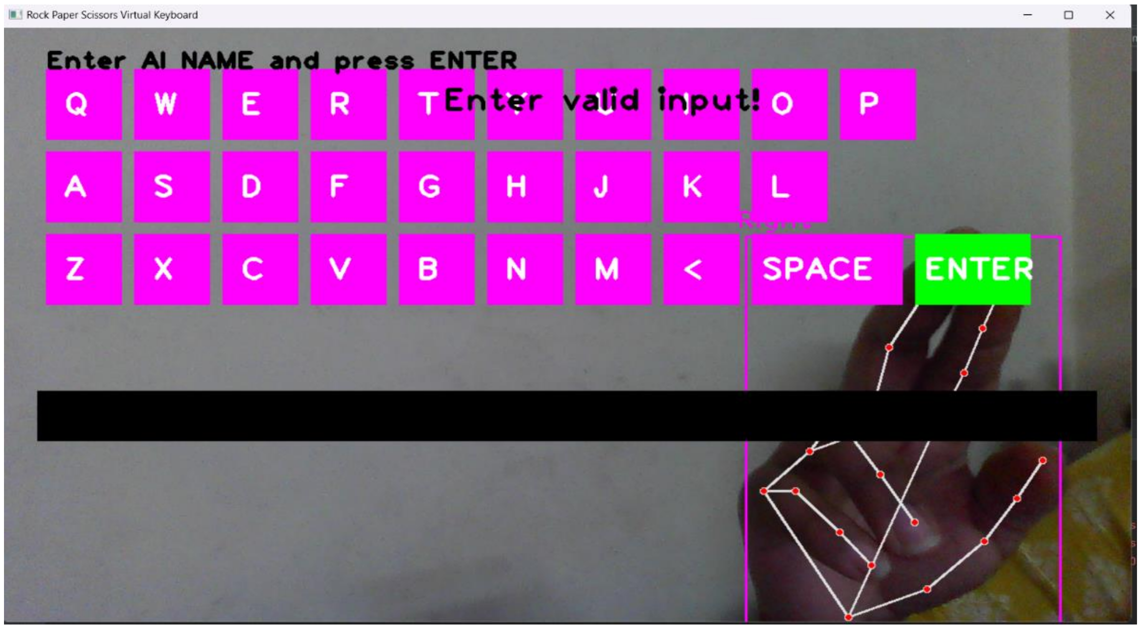
Screenshot 6.4 Zero Score Limit


Rock Paper Scissors Virtual Keyboard
Enter SCORE LIMIT and press ENTER
7
8
9
4
5
6
1
2
3
0
<
ENTER
3

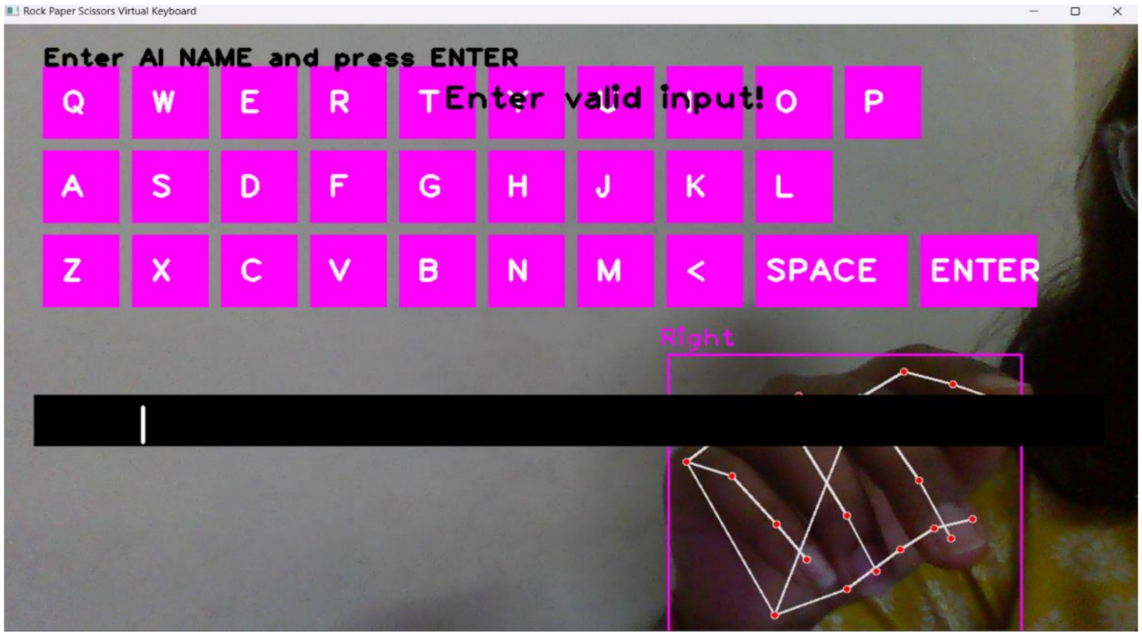
Screenshot 6.5 Valid Score Limit entry


Rock Paper Scissors Virtual Keyboard
Enter AI NAME and press ENTER
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER

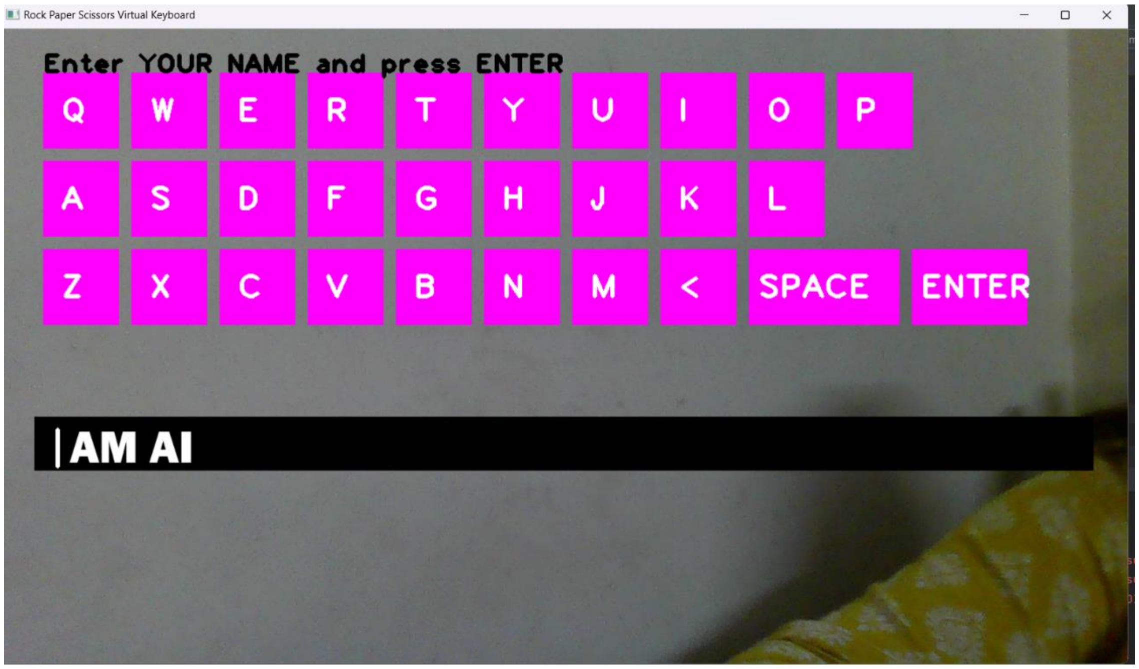
Screenshot 6.6 AI-name entry Layout


Rock Paper Scissors Virtual Keyboard
Enter AI NAME and press ENTER
Enter valid input!
Q W E R T O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER
Right

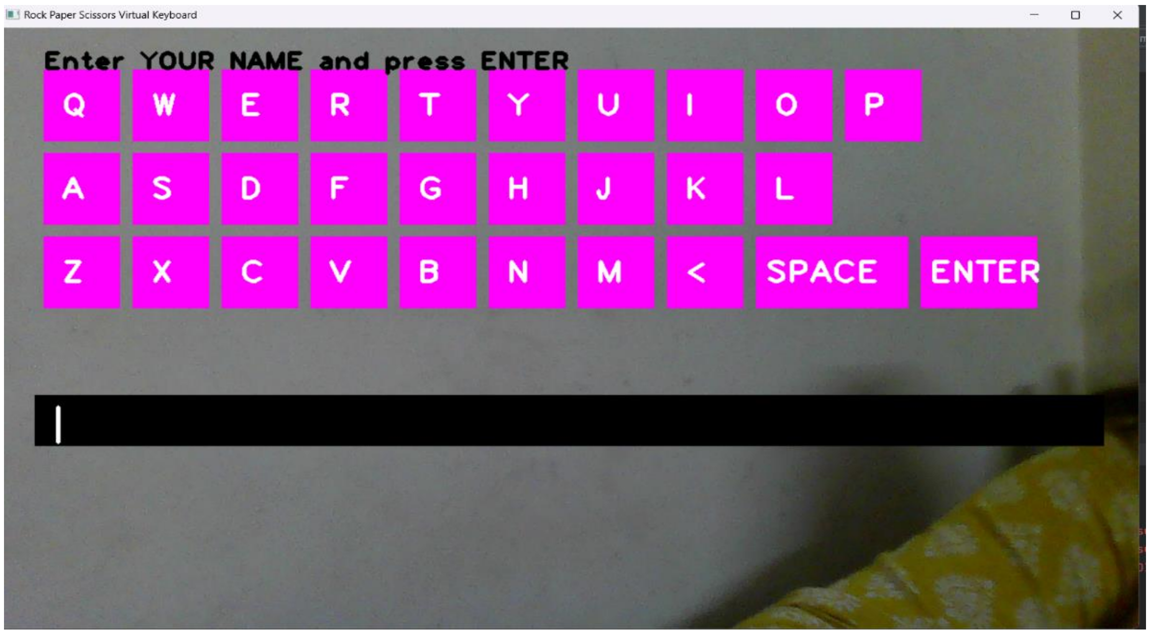
Screenshot 6.7 Empty AI-name


Rock Paper Scissors Virtual Keyboard
Enter AI NAME and press ENTER
Enter valid input!
Q W E R T O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER
Right

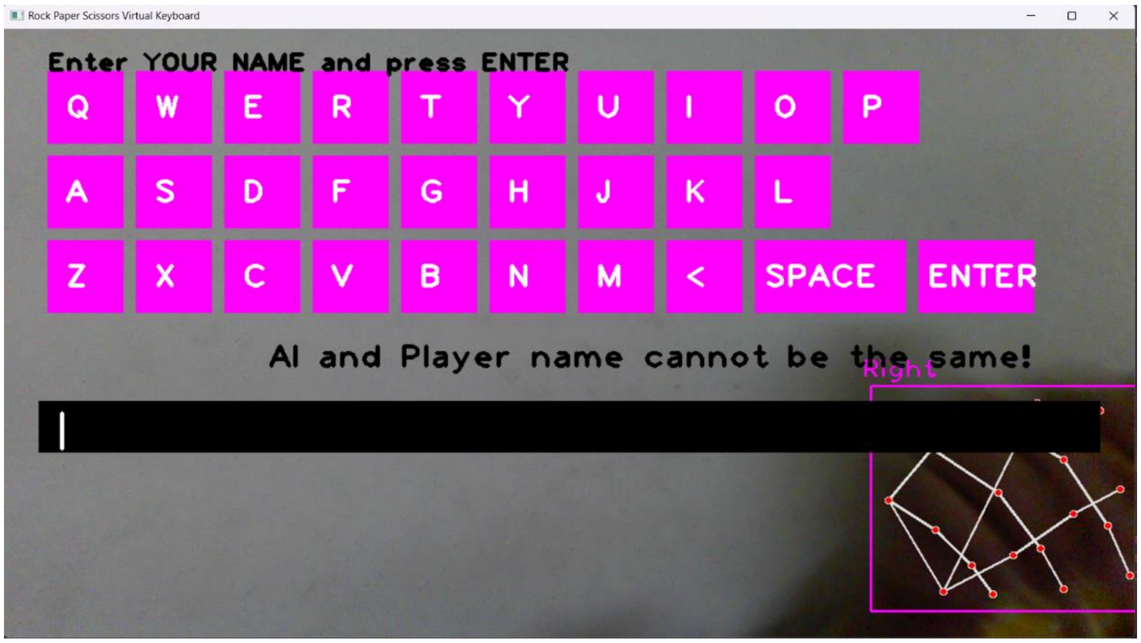
Screenshot 6.8 Spaces as AI-name


Rock Paper Scissors Virtual Keyboard
Enter YOUR NAME and press ENTER
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER
AM AI

Screenshot 6.9 Valid AI-name

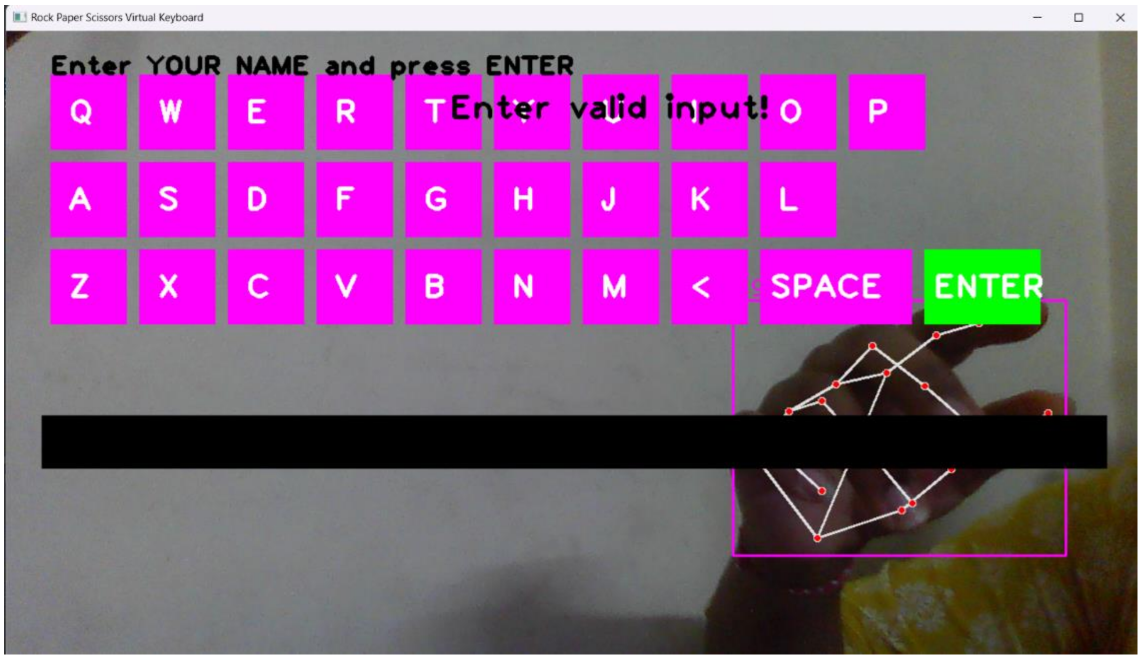

Rock Paper Scissors Virtual Keyboard
Enter YOUR NAME and press ENTER
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER

Screenshot 6.10 Player-name entry Layout


Rock Paper Scissors Virtual Keyboard
Enter YOUR NAME and press ENTER
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER
Right

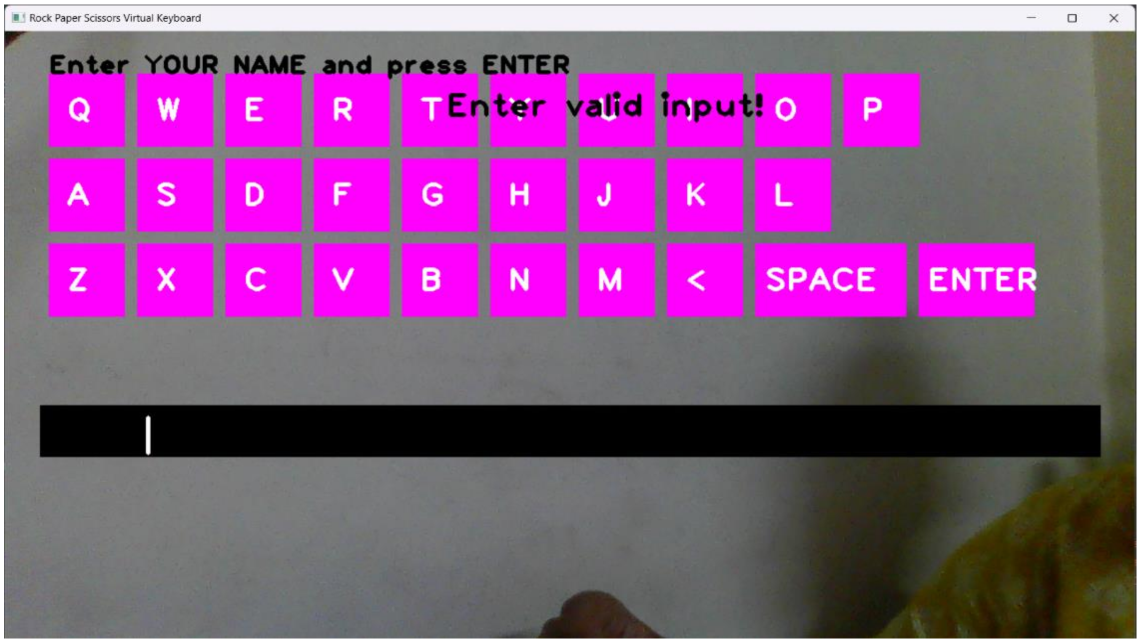
AI and Player name cannot be the same!

Screenshot 6.11 Same AI and Player-name

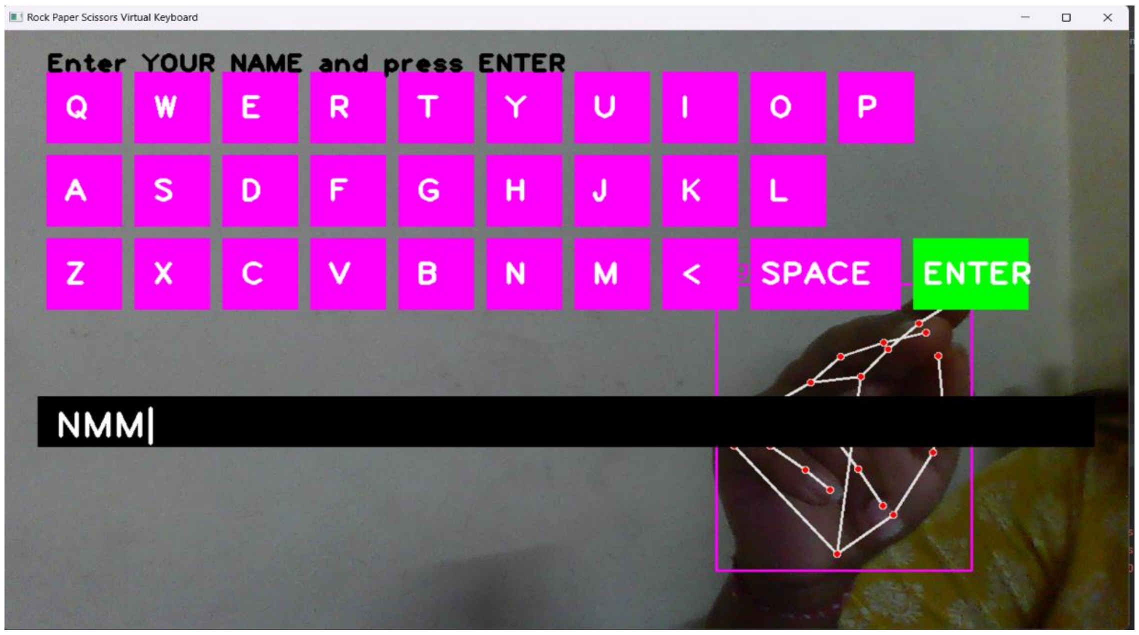

Rock Paper Scissors Virtual Keyboard
Enter YOUR NAME and press ENTER
Q W E R T O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER

Enter valid input!

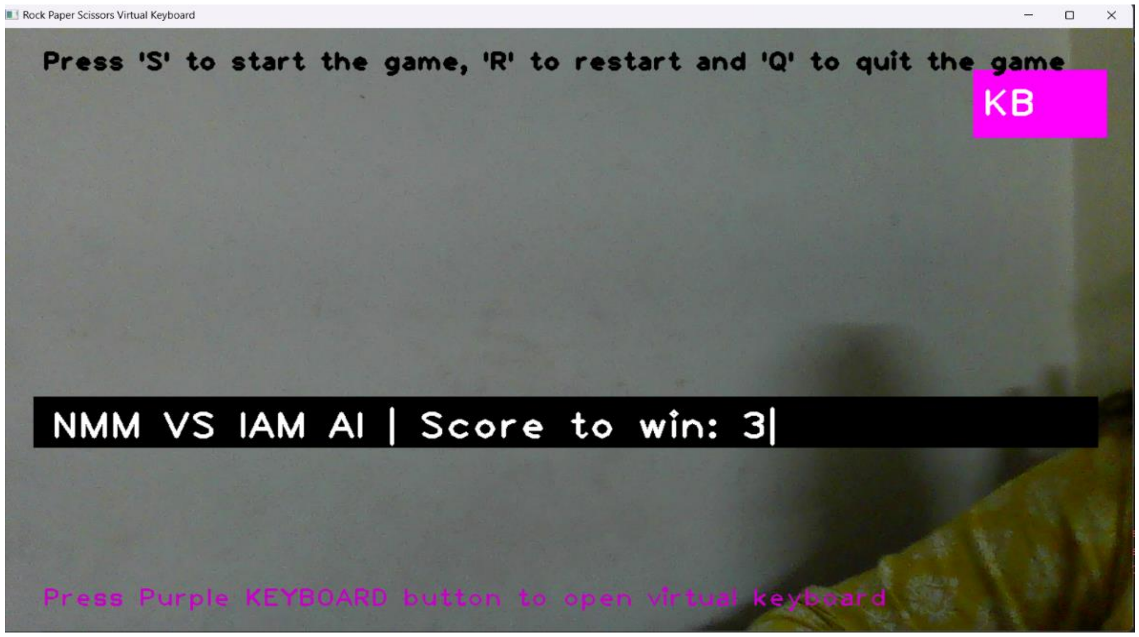
Screenshot 6.12 Empty Player-name


Rock Paper Scissors Virtual Keyboard
Enter YOUR NAME and press ENTER
Enter valid input!
Q W E R T O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER

Screenshot 6.13 Spaces as Player-name


Rock Paper Scissors Virtual Keyboard
Enter YOUR NAME and press ENTER
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER
NMM

Screenshot 6.14 Valid Player-name entry

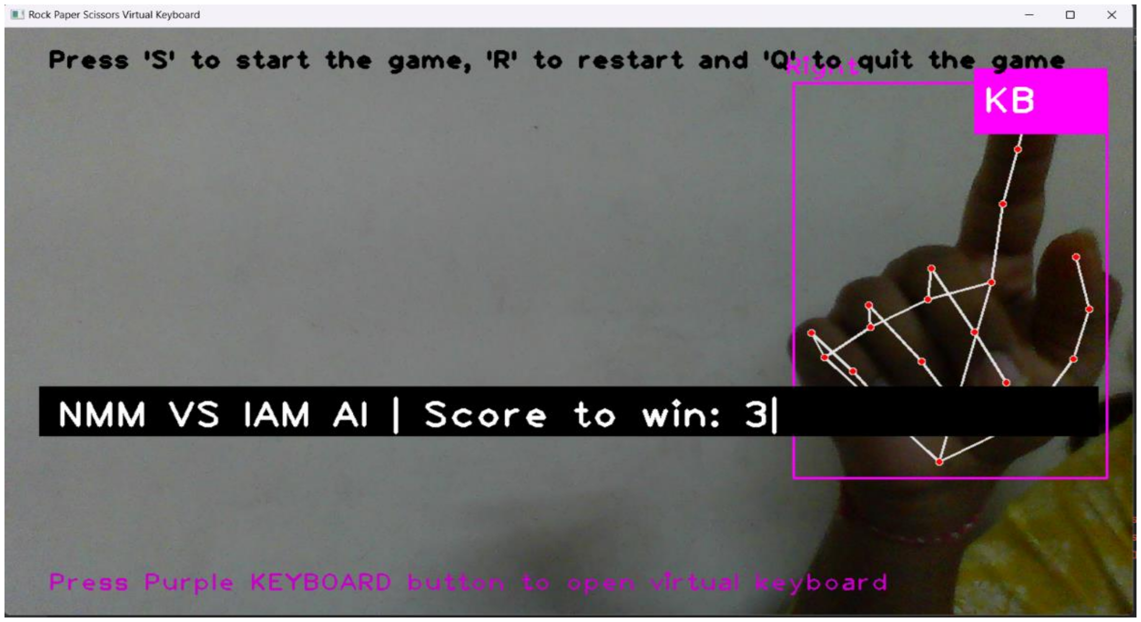


Press 'S' to start the game, 'R' to restart and 'Q' to quit the game

NMM VS IAM AI | Score to win: 3|

Press Purple KEYBOARD button to open virtual keyboard

Screenshot 6.15 Instructions

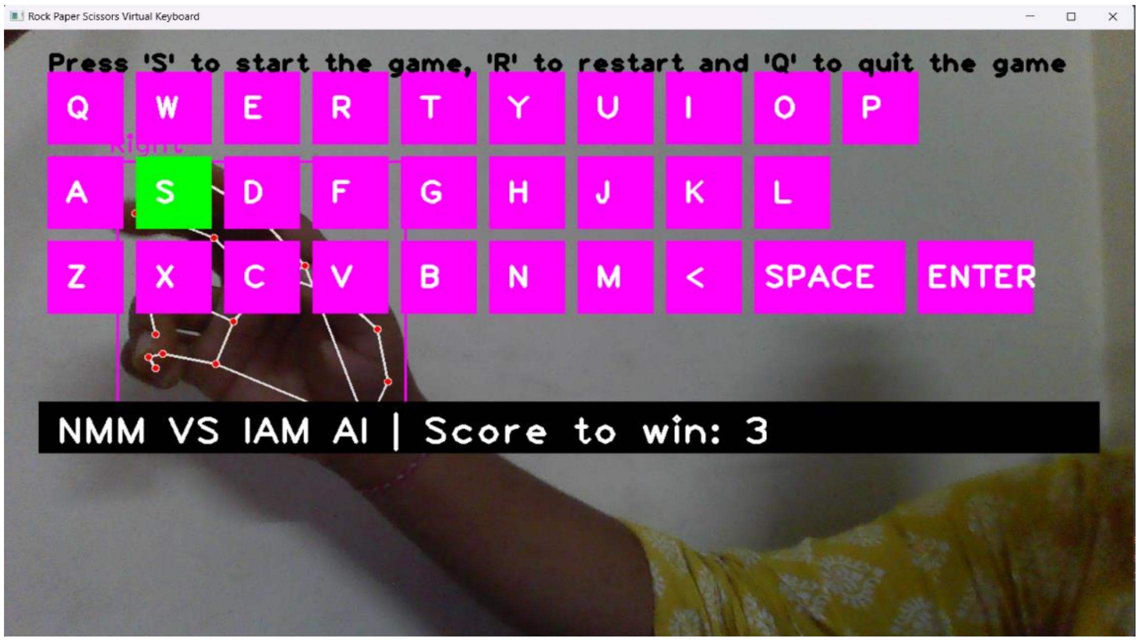


Press 'S' to start the game, 'R' to restart and 'Q' to quit the game

NMM VS IAM AI | Score to win: 3|

Press Purple KEYBOARD button to open virtual keyboard

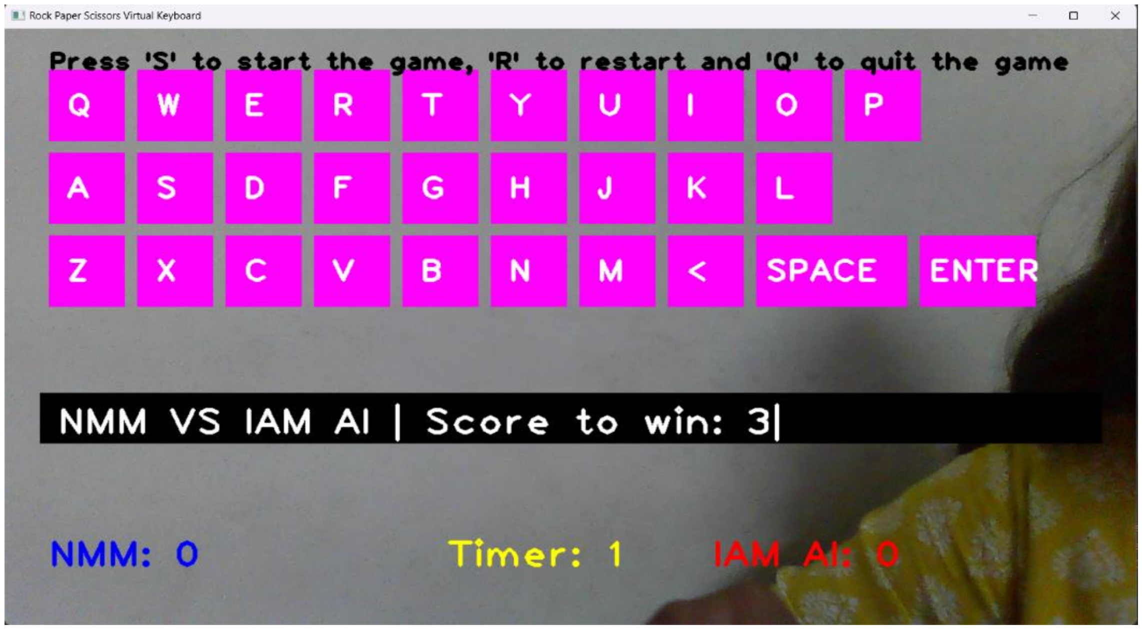
Screenshot 6.16 Re-opening of Virtual keyboard


Rock Paper Scissors Virtual Keyboard
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER
Right

Press 'S' to start the game, 'R' to restart and 'Q' to quit the game

NMM VS IAM AI | Score to win: 3

Screenshot 6.17 Start of Game

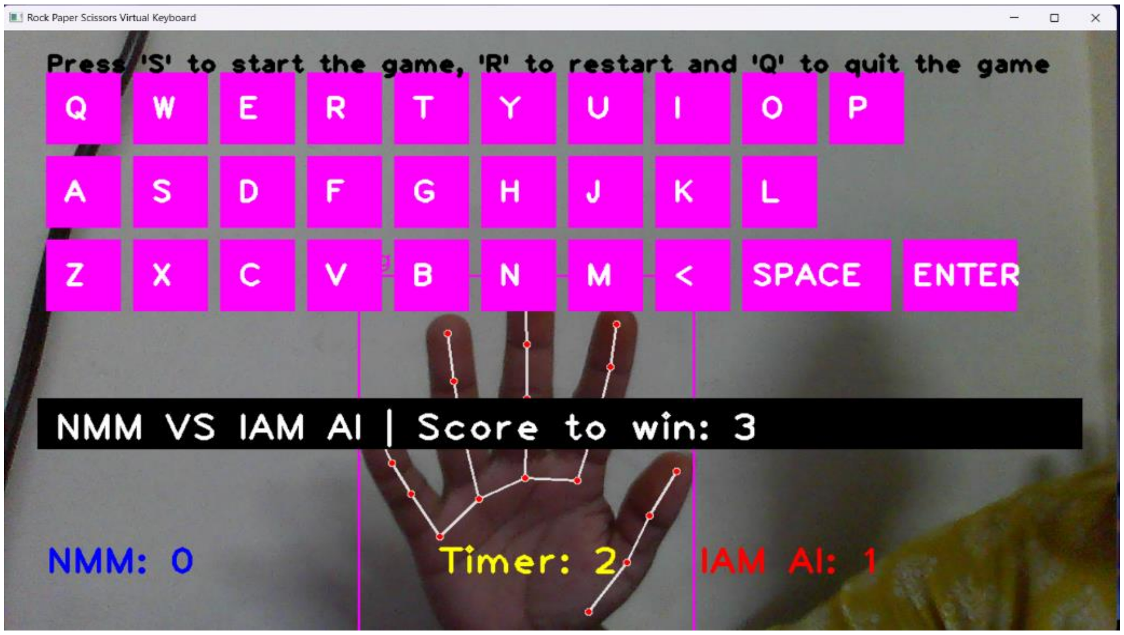

Rock Paper Scissors Virtual Keyboard
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER

Press 'S' to start the game, 'R' to restart and 'Q' to quit the game

NMM VS IAM AI | Score to win: 3|

NMM: 0 Timer: 1 IAM AI: 0

Screenshot 6.18 Timer begins


Rock Paper Scissors Virtual Keyboard
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M < SPACE ENTER

Press 'S' to start the game, 'R' to restart and 'Q' to quit the game

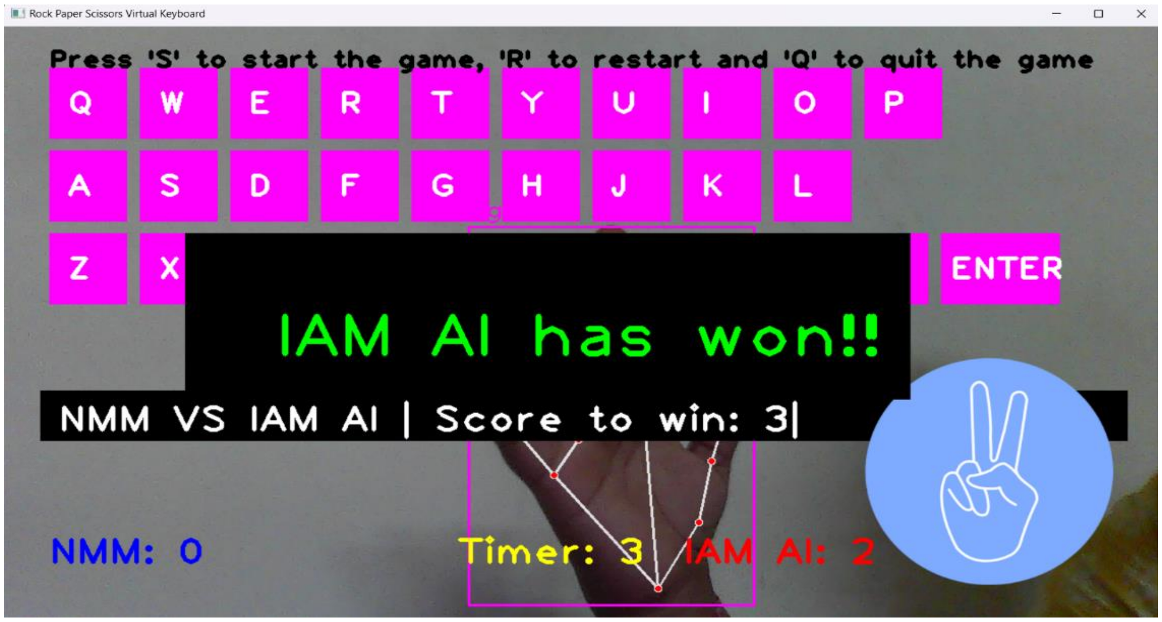
NMM VS IAM AI | Score to win: 3

NMM: 0

Timer: 2

IAM AI: 1

Screenshot 6.19 Score Updation



Press 'S' to start the game, 'R' to restart and 'Q' to quit the game

IAM AI has won!!

NMM VS IAM AI | Score to win: 3|

NMM: 0

Timer: 3

IAM AI: 2

Screenshot 6.20 Winner Declaration

7. Test Cases

7. Test Cases

7.1 Types of Testing

There are many types of testing methods are available in that mainly used testing methods are

- **Unit testing**

  Unit testing involves designing test cases that validate internal program logic, ensuring that program inputs produce expected outputs. In this project, unit tests were conducted on:

  - Gesture recognition logic using hand landmarks
  - Timer and game state transition logic
  - Result evaluation and score calculation
  - Input validation for player names and score limit
  - Each function was tested in isolation before integration

- **Integration testing**

  Integration testing ensured that all modules—camera input, gesture recognition, UI updates, and score handling—work together seamlessly. It validated that:

  - Recognized hand gestures are correctly matched against AI's gestures
  - UI updates (scores, results) sync correctly with gameplay
  - Inputs flow smoothly from capture to display and scoring

- **Functional test**

  Functional testing verified that the system functions as intended based on the requirements:

  - Valid inputs proceed to gameplay; invalid ones prompt error messages
  - S key starts the game, R restarts it, and Q exits cleanly
  - Hand gestures result in accurate scoring and winner display
  - Timer countdown behaves correctly within each round

- **System Test:**

  System testing evaluated the full, integrated game environment, including camera hardware and user interaction. It ensured:

- Smooth full-cycle gameplay from name input to game conclusion
- Proper rendering of camera feed on custom background
- Game responsiveness and timing remained consistent throughout multiple rounds

- **White Box Testing**

  White box testing involved examining the internal logic of key functions, particularly:

  - Finger state recognition logic
  - Timer state transitions
  - Conditional checks for determining round outcomes

  This ensured logical correctness and handled edge cases like tie scenarios or missing inputs.

- **Black Box Testing**

  Black box testing involved testing the system without internal code access:

  - Inputs like names, gestures, and commands were entered externally
  - Outputs (visual display, messages, and score) were verified for correctness
  - The system was treated as a closed box to evaluate user-facing behavior only

7.2 Test cases

Test strategy and approach

Field testing will be performed manually. All major functionalities such as hand gesture recognition, camera feed integration, UI rendering, and game logic will be thoroughly tested through both black box and white box approaches. Functional tests are written in detail to cover all scenarios.

Features to be tested

- Verify proper camera initialization and frame capture
- Validate accurate recognition of hand gestures (rock, paper, scissors)
- Ensure correct and randomized AI gesture display.

- Confirm correct winner logic and score updates after every round
- Ensure timer countdown functions correctly and resets between rounds
- Verify name input validation, score limit input validation, and command key handling (S, R, Q)
- Ensure proper display of UI elements such as scores, names, timer, and overlay images

The actual purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner.

Test Cases:

Test ID	Test Name	Inputs	Expected Output	Actual Output	Status
1	Camera Initialization	Open webcam	Live video feed starts successfully	Camera feed displayed in window	Succes s
2	Right hand identification	Show right hand in camera	Right hand identified and tracked	Right hand correctly tracked	Succes s
3	Left hand identification	Show left hand in camera	Left hand identified and tracked	Left hand correctly tracked	Succes s
4	Empty Score Limit Not Accepted	Leave score limit empty	"Enter valid input!" message	Error message shown	Succes s
5	Zero Score Limit Not Accepted	Enter '0' as score limit	Rectangle shape drawn when confirmed with pinky	"Enter valid input!" message	Succes s
6	Empty Player Name Not Accepted	Leave player name empty	"Enter valid input!" message	Error message shown	Succes s

7	Empty AI Name Not Accepted	Leave AI name empty	“Enter valid input!” message	Error message shown	Success
8	Space as Player Name Not Accepted	Enter space as player name	“Enter valid input!” message	Error message shown	Success
9	Space as AI Name Not Accepted	Enter space as AI name	“Enter valid input!” message	Error message shown	Success
10	Same AI and Player Name Not Allowed	Enter same name for both	“AI name cannot be same as Player”	Proper message shown	Success
11	Start Game on ‘S’ Key	Press 'S'	Game starts with camera and timer	Game starts correctly	Success
12	Restart Game on ‘R’ Key	Press 'R'	Game resets	Game resets properly	Success
13	Quit Game on ‘Q’ Key	Press 'Q'	Game exits	Game exits properly	Success
14	Player Wins Round	Player shows winning gesture	“Player_Name has won!!” displayed	Message displayed	Success

15	AI Wins Round	AI has winning gesture	“AI_Name has won!!” displayed	Message displayed	Success

Table 7.1: RPS game using OpenCV Test Cases

8. Conclusion

8. Conclusion

The AI-Powered Rock-Paper-Scissors game stands as a compelling demonstration of the transformative potential of computer vision and artificial intelligence in creating intuitive and engaging interactive experiences. By successfully integrating real-time hand gesture recognition through Python, OpenCV, and CvZone, the project showcases how classical game logic can be revitalized with cutting-edge technology. The ability to interpret player moves based on precise finger detection, combined with dynamic visual feedback and live score tracking, highlights significant advancements in human-computer interaction and real-time responsiveness. This project not only delivers an enjoyable gaming experience but also serves as a robust prototype for future applications. It underscores the practical utility of hand tracking in diverse domains, from immersive gaming to educational tools and accessibility solutions, ultimately paving the way for more natural and seamless interactions between humans and technology.

9. Future Scope

9. Future Scope

The future scope of AI-powered hand gesture recognition in interactive gaming and human-computer interaction is vast and transformative. Rock-Paper-Scissors game based on computer vision demonstrates the foundational capabilities that will evolve into more advanced systems. A primary focus will be enhancing recognition accuracy and robustness in real-world conditions, including variable lighting, background clutter, partial occlusion, and diverse hand characteristics, through the use of deep learning techniques such as Transformer-based and hybrid models. Gesture recognition will expand beyond static hand signs to dynamic and continuous motions, enabling more natural and nuanced controls in digital interfaces, real-time sign language interpretation, and hands-free operation of industrial equipment. The future will also see widespread adoption of multimodal interaction, combining gestures with voice commands, eye-tracking, haptic feedback, and brain-computer interfaces to deliver intuitive and immersive experiences. In gaming, seamless integration with AR and VR will allow players to interact with virtual environments using natural hand movements, enhancing realism and engagement. Beyond entertainment, applications will extend to telepresence, remote collaboration, surgical assistance, and smart environments, where gesture control can be used to operate devices without physical contact. Adaptive AI will personalize interactions by learning user behavior, adjusting difficulty levels, and dynamically generating content. Edge computing will play a vital role by enabling real-time gesture processing on local devices, reducing latency and dependency on cloud infrastructure. These advancements will pave the way for highly responsive, context-aware, and user-friendly systems, making human-computer interaction more intuitive and accessible than ever before.

10. References

10. References

[1] Patel, A., & Desai, R. (2021). OpenCV: A Comprehensive Review and Applications in Computer Vision. Journal of Computer Vision Research.

[2] Kumar, S., & Singh, P. (2020). A Review on OpenCV. International Journal of Image Processing.

[3] Mathur, S., & Gupta, S. (2021). Rock, Paper and Scissors Using OpenCV. Proceedings of the International Conference on AI and Computer Vision.

[4] Harish, H.K., et al. (2022). Rock Paper Scissors Using Gesture-Based Application. Journal of Interactive Technologies.

[5] Kumar, R., & Singh, A. (2021). Playing Rock-Paper-Scissors Using AI Through OpenCV. IEEE Access.

[6] Verma, A., & Kaur, J. (2020). A Review on Object Detection using OpenCV Method. International Journal of Computer Applications.

[7] Sharma, P., & Verma, D. (2021). Rock, Paper and Scissors Using OpenCV. International Journal of Computer Vision and AI.

[8] Khan, M., & Ali, S. (2022). Rock Paper Scissors Using Application. Gesture-Based Journal of Computer Science and Engineering.

[9] Sharma, R., & Gupta, L. (2021). OpenCV for Computer Vision Applications. Journal of Emerging Technologies.

[10] Mallik, B. et al. (2024). Virtual Keyboard using Hand Gesture Recognition and Mediapipe. Computer Systems Sci. Eng.