VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi: 590 018

A CGIP Mini Project Report (21CSL66)

on

" Rocknet : Hand Gesture RPS "

Submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Engineering
in
Computer Science & Engineering

Submitted by

ADITI DAS	**1AY21CS014**
ANJALI K S	**1AY21CS029**
HARSHINI GAJANAN BHAT	**1AY21CS063**
HEMAVATHI D P	**1AY21CS065**

Under the guidance of

Mr. Gururaj P

Assistant Professor

Department of Computer Science & Engineering

Department of Computer Science & Engineering

Acharya Institute of Technology

Soladevanahalli, Bangalore – 560090

2023-2024

ACHARYA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belgaum)
Soladevanahalli, Bangalore – 560 107

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that Computer Graphics and Image Processing mini project entitled **"Rocknet: Hand Gesture RPS"** is a bonafide work carried out by **ADITI DAS (1AY21CS014)** , **ANJALI K S (1AY21CS029)** , **HARSHINI GAJANAN BHAT (1AY21CS063) and HEMAVATHI D P (1AY21CS065)** of 6th semester in partial fulfillment for the award of degree of **Bachelor of Engineering in Computer Science & Engineering of the Visvesvaraya Technological University**, **Belagavi**, during the year **2023-2024.** It is certified that all corrections/ suggestions indicated for internal assessments have been incorporated in the Report deposited in the departmental library. The Mini Project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the **Bachelor of Engineering Degree**.

Signature of Guides **Signature of H.O.D**

Name of the examiners **Signature with date**

1.

2.

ABSTRACT

RockNet presents an innovative and engaging approach to playing Rock-Paper-Scissors through the use of real-time hand gesture recognition. This project leverages advanced computer vision techniques and Python libraries, such as OpenCV and MediaPipe, to capture and analyze hand movements, providing an immersive and interactive gaming experience. By accurately detecting hand poses corresponding to rock, paper, and scissors, RockNet ensures precise and responsive gameplay. The game adheres to the traditional rules of Rock-Paper-Scissors, determining the winner or declaring a draw based on the gestures of the players. The intuitive user interface enhances the experience by displaying a real-time camera feed, allowing players to see their gestures and receive immediate feedback. RockNet's seamless integration of computer graphics and image processing not only demonstrates the practical application of these technologies but also offers a fun and engaging way to interact with them. The project showcases how real-time hand pose estimation can be effectively combined with game development principles to create an interactive application. Whether for educational purposes, entertainment, or demonstrating the capabilities of computer vision, RockNet provides a compelling example of how traditional games can be revitalized with modern technology. This project exemplifies the potential of combining AI and computer vision to create new, engaging user experiences.

ACKNOWLEDGEMENT

We express our gratitude to our institution and management for providing us with good infrastructure, laboratory, facilities and inspiring staff whose gratitude was of immense help in completion of this seminar successfully.

We express our sincere gratitude to our principal in charge, **Dr .Rajeswari** and Vice principal **Prof. Marigowda C K** for providing required environment and valuable support for developing this mini project.

Our sincere thanks to **Dr. Ajith Padyana,** Head of the Department, Computer Science and Engineering, Acharya Institute of Technology for his valuable support and for rendering us resources for this mini project work.

We express our gratitude to **Mr. Gururaj P**, **Mrs. Varalakshmi B D, Mr. Jawahar Jonathan,** Assistant Professors, Dept. of Computer Science and Engineering, Acharya Institute of Technology who guided us with valuable suggestions in completing this mini-project at every stage.

Our gratitude thanks is be rendered to many people who helped us in all possible ways.

ADITI DAS	**(1AY21CS014)**
ANJALI K S	**(1AY21CS029)**
HARSHINI GAJANAN BHAT	**(1AY21CS063)**
HEMAVATHI D P	**(1AY21CS065)**

TABLE OF CONTENTS

Abstract (i)

Acknowledgement (ii)

Table of contents (iii)

List of figures (iv)

1. Introduction

1.1 Introduction to Computer Graphics and Image Processing 1

1.2 Open GL 1

2. System Requirements Specification

2.1 Software Requirements 3

2.2 Hardware Requirements 3

2.3 Functional requirements 3

3. About the Project

3.1 Introduction 5

3.2 Objective 6

3.3 Built-in functions 6

3.4 User defined functions 7

3.5 Special Functions 8

3.6 Data flow diagram 9

3.7 Source Code/Pseudo Code 11

3.8 Code Explanation 13

4. Results 16

5. Conclusion & Future Work 19

References 21

LIST OF FIGURES

SL.NO	Figure Name	Page Number
01	Fig 3.1 Hand Landmarks	6
02	Fig 3.2 Data Flow Diagram	9
03	Figure 4.1: Home Page of ROCKNET – Hand Recognition with RPS	16
04	Figure 4.2: Opponent selection page	16
05	Figure 4.3: Play with Computer page of ROCKNET – Hand Recognition with RPS	17
06	Figure 4.4: Play with friend window of ROCKNET – Hand Recognition with RPS	17
07	Figure 4.5: About page of ROCKNET – Hand Recognition with RPS	18

CHAPTER - 01

INTRODUCTION

1.1 INTRODUCTION TO COMPUTER GRAPHICS AND IMAGE PROCESSING

Computer graphics is a sub-field of computer science focused on digitally synthesizing and manipulating visual content, crucial for applications like RockNet, a real-time Rock-Paper-Scissors game developed using Django. This project demonstrates the power of combining two and three-dimensional image creation with advanced computer vision techniques. Over time, hardware and algorithms have evolved to improve the speed and quality of image generation, essential for real-time applications like RockNet. The game involves creating and storing models and images of hand gestures, leveraging computer graphics for accurate visualization.

Computer graphics encompasses all aspects of image creation, including imaging, rendering, modeling, and animation. These principles are applied in RockNet to provide a seamless and engaging user experience, allowing players to see their gestures in realtime and receive immediate feedback.

Image processing, on the other hand, uses algorithms and mathematical models to process and analyze digital images. In RockNet, image processing techniques are employed to enhance the quality of the captured hand gestures, extract meaningful information, and automate the detection of rock, paper, and scissors gestures. By integrating computer graphics and image processing, RockNet can create realistic textures for hand poses while adjusting lighting and removing noise from the camera feed.

Understanding both computer graphics and image processing unlocks the potential to create and manipulate the visual world. RockNet exemplifies this by providing an interactive and immersive gaming experience, showcasing the practical application of these technologies within a Django framework.

1.2 OPEN GL

Open Graphics Library (OpenGL) is a cross-language (language independent), cross platform (platform-independent) API for rendering 2D and 3D Vector Graphics (use of polygons to represent image). OpenGL API is designed mostly in hardware.

Design: This API is defined as a set of functions which may be called by the client program. Although functions are similar to those of C language but it is language independent.

Development: It is an evolving API and Khronos Group regularly releases its new version having some extended feature compare to previous one. GPU vendors may also provide some additional functionality in the form of extension.

Associated Libraries: The earliest version is released with a companion library called OpenGL utility library. But since OpenGL is quite a complex process. So, in order to make it easier other library such as OpenGL Utility Toolkit is added which is later superseded by free glut. Later included library were GLEE, GLEW, and gliding.

Implementation: Mesa 3D is an open-source implementation of OpenGL. It can do pure software rendering and it may also use hardware acceleration on BSD, Linux, and other platforms by taking advantage of Direct Rendering Infrastructure.

CHAPTER - 02

SYSTEM REQUIREMENTS SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- **Programming language** – Python, Html, CSS , Javascript , Django
- **Operating System** – Windows OS
- **Libraries** – OpenCV, MediaPipe

2.2 HARDWARE REQUIREMENTS

- Dual Core Processor
- Webcam
- 4GB RAM
- 40 GB Hard disks

2.3 FUNCTIONAL REQUIREMENTS

1. Hand Gesture Detection:

- Accurate Detection: The system must utilize MediaPipe's hand landmarks detection module to accurately detect and interpret the player's hand gestures. This involves identifying specific points on the hand to determine whether the gesture represents rock, paper, or scissors.
- Real-time Processing: The detection must happen in real-time to ensure the game responds promptly to the player's hand movements. This requires efficient processing to minimize latency and provide a seamless gaming experience.

2. Random Computer Move Generation:

- Random Choice Mechanism: The game should use Python’s random.choice function to generate a move for the computer. This function will randomly select one of the three possible moves: "rock," "paper," or "scissors."
- Equal Probability: Each move (rock, paper, scissors) should have an equal probability of being chosen by the computer to maintain fairness in the game.

3. Game Logic and Outcome Determination:

- Comparison Logic: The system must implement the logic to compare the player’s move with the computer’s move. It should determine the outcome based on the rules of rock-paper-scissors: rock beats scissors, scissors beats paper, and paper beats rock.
- Handling All Outcomes: The game should cover all possible scenarios: a tie (both moves are the same), a player win (player’s move beats the computer’s move), and a computer win (computer’s move beats the player’s move).

4. Real-time Feedback Display:

- Move Display: The game should display the moves made by both the player and the computer. This includes updating the screen with messages like "Player played rock" and "Computer played scissors."
- Outcome Display: The system must immediately show the result of the game after both moves are made, indicating whether it’s a tie, a player win, or a computer win. This feedback should be clear and prominent to enhance user engagement.

5. User Interface:

- Start and Instructions: The game interface should include a start button to initiate the game and clear instructions on how to play. It should guide the player on how to position their hand for gesture detection.
- Visual Appeal: The interface should be visually appealing, with appropriate graphics, animations, and text styles to make the game enjoyable. The design should be intuitive, allowing users to understand and interact with the game easily.

6. Error Handling and Robustness:

- Detection Errors: The system should handle cases where the hand gesture is not detected properly or when only partial hand landmarks are recognized. It should provide messages to the player to adjust their hand position or try again.
- 5tGraceful Degradation: In cases of temporary detection failure, the game should not crash but instead prompt the player to retry. This ensures robustness and enhances the overall user experience by handling errors gracefully.

CHAPTER - 03

ABOUT THE PROJECT

3.1 INTRODUCTION

A Computer Graphics and Image processing project based on "Rocknet : Hand Gesture RPS". RockNet is a real-time Rock-Paper-Scissors game built with Python libraries. It uses computer vision to detect hand gestures from your webcam and determine the winner based on the classic game rules. MediaPipe's hand pose estimation helps recognize gestures like rock, paper, and scissors. The program provides a user-friendly interface with a countdown timer and visual feedback, making RockNet an interactive way to play Rock-PaperScissors.

3.1.1 MEDIAPIPE MODELS

1. **Palm Detection Model**
   - Single-shot detector model optimized for mobile real-time uses.
   - Detects occluded and self-occluded hands with a large scale span.
   - Provides additional context like arm, body, or person features for accurate hand localization.
   - Trains a palm detector instead of a hand detector for simpler bounding boxes.
   - Uses non-maximum suppression algorithm for two-hand self-occlusion cases.
   - Models palms using square bounding boxes, reducing the number of anchors by 3-5.
   - Uses encoder-decoder feature extractor for bigger scene context awareness.
   - Minimizes focal loss during training to support large amount of anchors.
   - Achieves an average precision of 95.7% in palm detection.
2. **Hand Landmark Model**
   - Performs precise keypoint localization of 21 3D hand-knuckle coordinates via regression.
   - Learns a consistent internal hand pose representation.
   - Robust even to partially visible hands and self-occlusions.
   - Manually annotated ~30K real-world images with 21 3D coordinates.
   - Renders high-quality synthetic hand model over various backgrounds for additional supervision on hand geometry.

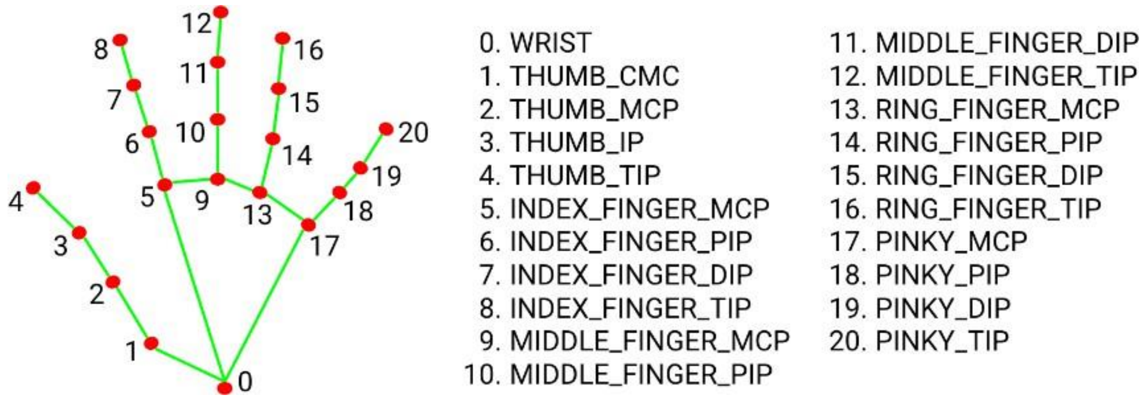

0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 3.1 : Hand Landmarks

3.2 OBJECTIVE

The objective of the project described in the text is to create a real-time RockPaperScissors game that uses computer vision techniques for hand pose estimation and gesture recognition. The project integrates these techniques with game development principles to create an interactive gaming experience.

3.3 BUILT-IN FUNCTIONS

- **cv2.cvtColor(frame,cv2.COLOR_BGR2RGB) and cv2.cvtColor (frame, cv2.COLOR_RGB2BGR):** These functions from the OpenCV library are used to convert the color space of an image between BGR (the default color space used by OpenCV) and RGB color spaces.
- **cv2.flip(frame, 1):** This function from the OpenCV library flips the image horizontally (1 means flipping horizontally, 0 means flipping vertically, and -1 means flipping both horizontally and vertically).
- **cv2.rectangle(frame, start_point, end_point, color, thickness):** This function from the OpenCV library draws a rectangle on the specified image (frame) with the given start and end points, color, and thickness.
- **cv2.putText(frame, text, position, font, font_scale, color, thickness, line_type):** This function from the OpenCV library draws text on the specified image

(frame) at the given position, with the specified font, font scale, color, thickness, and line type.

- **cv2.imshow('frame', frame):** This function from the OpenCV library displays the specified image (frame) in the window created earlier with cv2.namedWindow().
- **cv2.getWindowProperty('frame', cv2.WND_PROP_VISIBLE):** This function from the OpenCV library retrieves the specified property (cv2.WND_PROP_VISIBLE) of the window named ‘frame’. In this case, it's used to check if the window is still open (< 1 means the window is closed).
- **cv2.waitKey(delay):** This function from the OpenCV library waits for a key press event for the specified delay (in milliseconds). If the 'q' key is pressed, the script exits.
- **cv2.destroyAllWindows():** This function from the OpenCV library destroys all the windows created by the script.

3.4 USER DEFINED FUNCTIONS

1. gen(camera):

- Generates a video frame from the provided camera instance for streaming.
- Parameters: camera (an instance of VideoCamera or ComputerCamera).
- Yields: A video frame in a format suitable for streaming (multipart/x-mixed-replace).

2. index(request):

- Renders the index page of the game.
- Parameters: request (Django request object).
- Returns: An HttpResponse rendering the game/index.html template.

3. friend_game(request):

- Renders the game page for playing against a friend.
- Parameters: request (Django request object).
- Returns: An HttpResponse rendering the game/play.html template with the mode set to 'friend'.

4. computer_game(request):

- Renders the game page for playing against the computer.

- Parameters: request (Django request object).
- Returns: An HttpResponse rendering the game/play.html template with the mode set to 'computer'.

5. video_feed_friend(request):

- Provides a video feed for playing against a friend.
- Parameters: request (Django request object).
- Returns: A StreamingHttpResponse with a video feed generated from VideoCamera.

6. video_feed_computer(request):

- Provides a video feed for playing against the computer.
- Parameters: request (Django request object).
- Returns: A StreamingHttpResponse with a video feed generated from ComputerCamera.

7. select_opponent(request):

- Renders a page for selecting the opponent (friend or computer).
- Parameters: request (Django request object).
- Returns: An HttpResponse rendering the game/buttons.html template.

8. about(request):

- Renders the about page of the game.
- Parameters: request (Django request object).
- Returns: An HttpResponse rendering the game/about.html template.

3.5 SPECIAL FUNCTIONS

- **cv2.VideoCapture(0):** This function from the OpenCV library is used to initialize ,the video capture from the default camera (0 represents the default camera). It returns a VideoCapture object that can be used to read frames from the camera.
- **cv2.namedWindow('frame', cv2.WINDOW_NORMAL):** This function from the OpenCV library creates a window with the specified name ('frame') and displays it with normal properties (cv2.WINDOW_NORMAL).

- **getHandMove(hand_landmarks):** Determines the hand gesture ("rock", "paper", or "scissors") based on the provided hand landmarks. Parameters: hand_landmarks (MediaPipe hand landmarks). Returns: A string representing the hand gesture ("rock", "paper", or "scissors").

3.6 DATA FLOW DIAGRAM

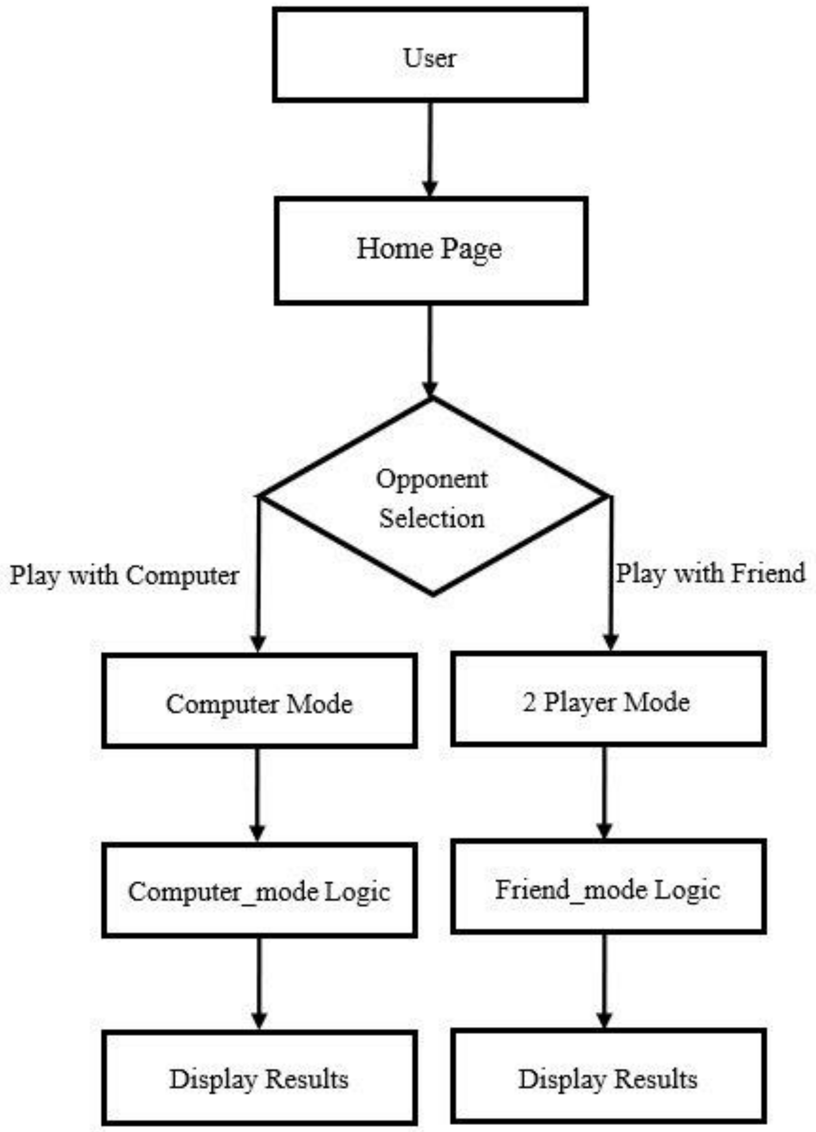


Figure 3.2 : Data flow diagram

The data flow diagram (DFD) presented outlines the sequence of actions and interactions a user undergoes when engaging with a hand gesture recognition game in a Django application. It starts with the user accessing the Home Page, which serves as the entry point to the game. From the Home Page, the user is prompted to make a crucial decision regarding the mode of gameplay, which is depicted as the “Opponent Selection” stage. This stage is a decision node where the user chooses between playing against the computer or another human player, thereby bifurcating the flow into two distinct paths.

In the first path, “Play with Computer”, the user engages in a single-player mode. This path leads to the “Computer Mode”, where the application initiates the logic specific to playing against the computer. This involves capturing the user's hand gestures through their webcam using MediaPipe and OpenCV. Simultaneously, the computer generates its move randomly or based on a predefined algorithm. The logic here is designed to compare the user's gesture against the computer's gesture to determine the outcome of each round. After processing the inputs, the results are displayed to the user, showcasing whether they won, lost, or if it was a draw.

The second path, “Play with Friend”, is designed for multiplayer interaction. Here, the user proceeds to the “2 Player Mode”, where both players’ gestures need to be captured and analysed. This mode requires real-time synchronization between the two players’ webcams to ensure the game flow is smooth and accurate. The “Friend_mode Logic” handles the simultaneous gesture recognition for both players, comparing their inputs to determine the winner of each round. This logic is more complex as it must account for the possibility of latency and ensure fairness in gesture recognition. After each round, the results are displayed to both players, maintaining transparency and engagement.

The display of results in both modes is a critical part of the user experience. In the “Display Results” stages for both the computer and friend modes, the application presents the outcome of the game rounds in a user-friendly manner. This may include visual representations of the gestures played, the winner of the round, and the overall score. The results display not only provides immediate feedback to the players but also enhances the interactive aspect of the game by making it visually appealing and informative.

Overall, the data flow diagram provides a clear and structured overview of the game's functionality. It ensures that each step from opponent selection to result display is logically connected and that the transition between different stages is seamless. This structured approach helps in maintaining the integrity of the game's logic and provides a smooth and engaging user experience, whether playing against the computer or another player. By clearly defining the flow and logic of the game, the DFD aids developers in understanding the system requirements and aids in the implementation and troubleshooting process.

3.7 SOURCE CODE / PSEUDO CODE

1.Setting Up MediaPipe Hands: First, initialize MediaPipe Hands and related drawing utilities.

```
import cv2 as cv
import mediapipe as mp
import threading
import random
# Initialize MediaPipe Hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
```

Below code snippet demonstrates the initialization of essential modules for a hand gesture recognition game using OpenCV and MediaPipe in Python. It imports necessary libraries: ‘cv2’ for computer vision tasks, ‘mediapipe’ for hand gesture recognition, ‘threading’ for handling concurrent execution, and ‘random’ for generating random values, likely for the computer's move in the game. The MediaPipe hands module is initialized with ‘mp_drawing’ for drawing utilities, ‘mp_drawing_styles’ for drawing styles, and ‘mp_hands’ for hand tracking and recognition functionality. This setup prepares the application to capture and process hand gestures effectively.

2. Hand Gesture Detection Function: This function determines the gesture based on the landmarks of the detected hand.

```
def getHandMove(hand_landmarks):
    landmarks = hand_landmarks.landmark
    # Check for "rock"
    if all([landmarks[i].y < landmarks[i+3].y for i in range(9, 20, 4)]):
        return "rock"
    # Check for "scissors"
    elif landmarks[13].y < landmarks[16].y and landmarks[17].y < landmarks[20].y:
        return "scissors"
    # Default to "paper"
    else:
        return "paper"
```

The ‘getHandMove’ function determines the gesture of a hand (rock, paper, or scissors) based on the landmarks provided by the MediaPipe hands module. It first assigns the ‘landmarks’ variable to the ‘landmark’ attribute of the ‘hand_landmarks’ object. To identify the “rock” gesture, it checks if specific landmarks (9, 13, 17, and 20) are all below their respective preceding landmarks (13, 17, 21, and 24) in the y-coordinate, indicating a closed fist. For the “scissors” gesture, it checks if the y-coordinates of landmarks 13 and 17 are below those of landmarks 16 and 20, respectively, which suggests an extended index and middle finger. If neither condition is met, it defaults to the “paper” gesture, implying an open hand. This function plays a critical role in interpreting hand gestures for the game.

3. Game Logic – 2 Player Mode

```
hls = results.multi_hand_landmarks
if hls and len(hls) == 2:
    self.p1_move = getHandMove(hls[0])
    self.p2_move = getHandMove(hls[1])
if self.p1_move == self.p2_move:
    self.gameText += " Game is tied."
 elif self.p1_move == "paper" and self.p2_move == "rock":
    self.gameText += " Player 1 wins."
elif self.p1_move == "rock" and self.p2_move == "scissors":
    self.gameText += " Player 1 wins."
elif self.p1_move == "scissors" and self.p2_move == "paper":
    self.gameText += " Player 1 wins."
else:
    self.gameText += " Player 2 wins!"
```

The above code snippet shows the logic for ‘2 Player Mode’. It begins by checking if two hands are detected (hls) by MediaPipe's hand landmarks detection module. If exactly two hands are detected, it retrieves the moves for Player 1 and Player 2 using the getHandMove function, which presumably interprets the hand landmarks to determine whether the gesture is rock, paper, or scissors. This ensures that the game can only proceed when both players’ moves are detected, maintaining the integrity of the two-player game mechanics.

The code then compares the moves of both players to determine the outcome of the game. If both players make the same move, the game is declared a tie. Otherwise, it checks the winning conditions for Player 1, such as “paper” beating “rock”, “rock” beating “scissors”, and “scissors” beating “paper”. If none of these conditions are met, it concludes that Player 2 wins. This logical flow ensures that all possible outcomes of the game are covered, and appropriate messages are appended to self.gameText to indicate the result, enhancing the game's interactivity and feedback to the players.

4. Game Logic – 1 Player Mode

```
if hls and len(hls) == 1:
    self.player_move = getHandMove(hls[0])
    self.computer_move = random.choice(["rock", "paper", "scissors"])
self.gameText = f"Player played {self.player_move}. Computer played {self.computer_move}."
if self.player_move == self.computer_move:
    self.gameText += " Game is tied."
elif self.player_move == "paper" and self.computer_move == "rock":
    self.gameText += " Player wins."
elif self.player_move == "rock" and self.computer_move == "scissors":
    self.gameText += " Player wins."
elif self.player_move == "scissors" and self.computer_move == "paper":
    self.gameText += " Player wins."
else:
    self.gameText += " Computer wins!"
```

The code snippet provided here is designed for a hand gesture recognition game where a single player competes against the computer. When the hand landmarks (‘hls’) detected by MediaPipe indicate only one hand, the game recognizes the player's move using the ‘getHandMove’ function. Simultaneously, the computer randomly selects a move from “rock”, “paper”, or “scissors” using Python's ‘random.choice’ function. The game's text output (‘self.gameText’) is updated to display the moves of both the player and the computer, providing immediate feedback on the choices made.

Following this, the code determines the outcome of the game by comparing the player's move with the computer's move. If both moves are the same, it results in a tie. The code then checks various win conditions for the player: “paper” beats “rock”, “rock” beats “scissors”, and “scissors” beats “paper”. If the player's move does not meet any of these winning conditions, the computer is declared the winner. This logical sequence ensures all possible game outcomes are addressed and the result is clearly communicated to the player through the updated game text.

3.8 CODE EXPLAINATION:

The provided code snippets are part of a larger hand gesture recognition game project, where the player competes against a computer opponent in a game of rock-paper-scissors. This game leverages the MediaPipe library to detect hand gestures and determine the player’s move, while the computer’s move is generated randomly. The outcome of each game round is then evaluated and displayed to the player.

The game starts by checking if MediaPipe has detected any hand landmarks (‘hls’). Specifically, the code looks for cases where exactly one hand is detected, as indicated by the condition ‘if hls and len(hls) == 1’. This ensures that the game proceeds only when a single hand gesture is captured, which corresponds to the player's move.

Once a single hand is detected, the ‘getHandMove’ function is called with ‘hls[0]’ as its argument. This function is responsible for interpreting the hand landmarks and determining the player’ gesture, which could be “rock”, “paper”, or “scissors”. The result is then stored in the ‘self.player_move’ variable. Simultaneously, the computer's move is generated randomly from the list [“rock”, “paper”, “scissors”] using Python’s ‘random.choice’ function. The randomly chosen move is stored in the ‘self.computer_move’ variable.

The next step involves updating the game text to reflect the moves made by both the player and the computer. The string ‘self.gameText’ is constructed to include the player’s move and the computer’s move, providing immediate feedback to the player. This is done using an f-string for string formatting, which results in a message like “Player played rock. Computer played scissors”.

To determine the outcome of the game, the code then compares the player’s move with the computer’s move. The first condition checks if the moves are the same, indicating a tie. If the condition ‘if self.player_move == self.computer_move’ is true, the game text is appended with “Game is tied”. This ensures that ties are handled correctly and communicated to the player.

For scenarios where the moves are different, the code evaluates various winning conditions for the player. It checks if the player's move beats the computer’s move based on the rules of rock-paper-scissors. Specifically, the conditions ‘elif self.player_move == “paper” and self.computer_move == “rock”’, ‘elif self.player_move == “rock” and self.computer_move == “scissors” ’, and ‘elif self.player_move == “scissors” and self.computer_move == “paper” ’ cover all possible win scenarios for the player. If any of these conditions are met, the game text is appended with “Player wins”.

If none of the player win conditions are satisfied, the code defaults to the conclusion that the computer has won. This is handled by the final ‘else’ statement, which appends “Computer wins!”

to the game text. This ensures that all possible outcomes of the game are covered, and the result is clearly communicated to the player.

The implementation also highlights the importance of real-time feedback and user interaction in a game setting. By updating the game text with the moves and outcomes, the code keeps the player informed and engaged throughout the game. Additionally, the use of random move generation for the computer ensures unpredictability, making the game more challenging and enjoyable.

In summary, these code snippets form a crucial part of the hand gesture recognition game, showcasing the integration of computer vision, randomization, and game logic to create an interactive and entertaining experience for users. It demonstrates how advanced technologies like MediaPipe can be leveraged to develop innovative applications in the field of gaming.

CHAPTER - 04

RESULTS

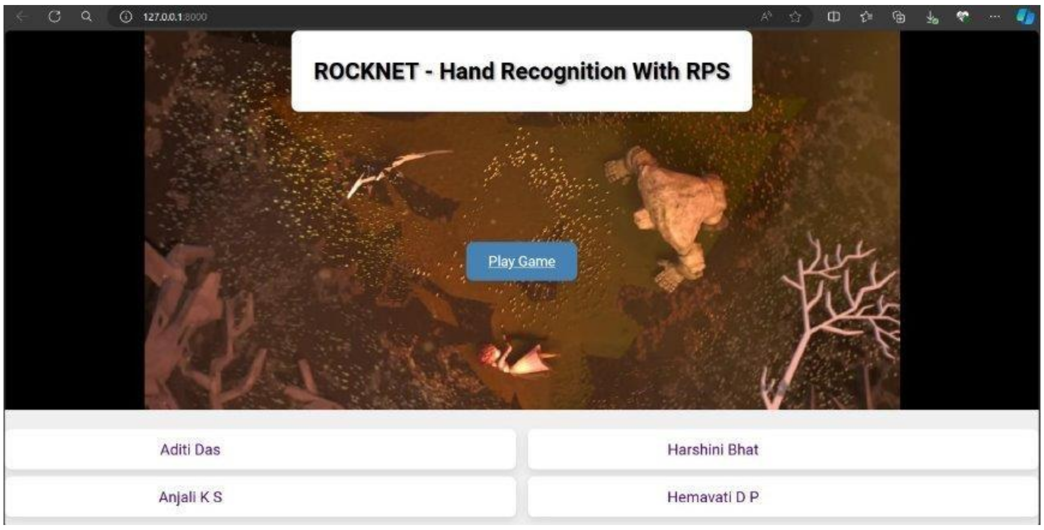
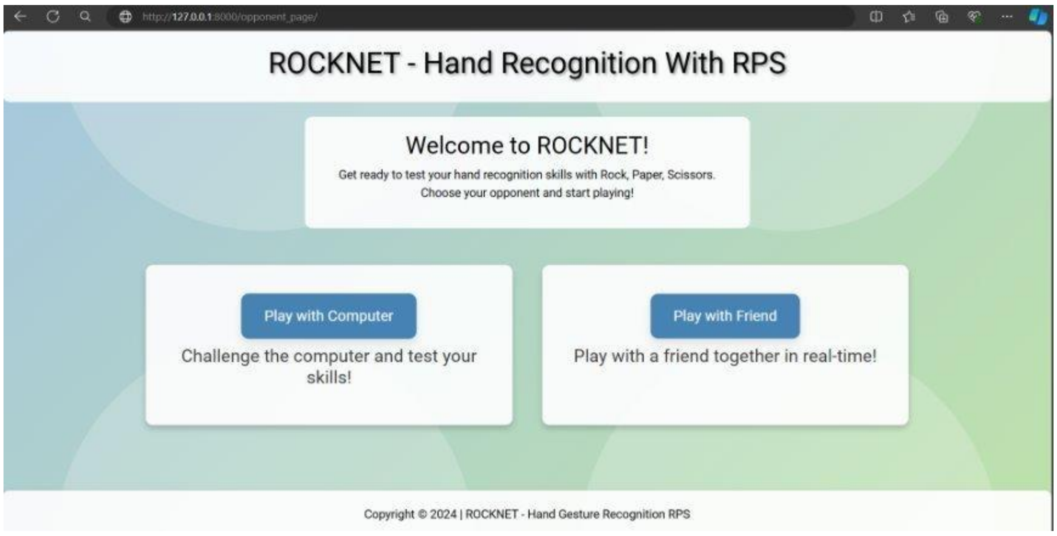


Figure 4.1: Home Page of ROCKNET – Hand Recognition with RPS


ROCKNET - Hand Recognition With RPS

Welcome to ROCKNET!

Get ready to test your hand recognition skills with Rock, Paper, Scissors. Choose your opponent and start playing!

Play with Computer

Challenge the computer and test your skills!

Play with Friend

Play with a friend together in real-time!

Copyright © 2024 | ROCKNET - Hand Gesture Recognition RPS

Figure 4.2: Opponent selection page

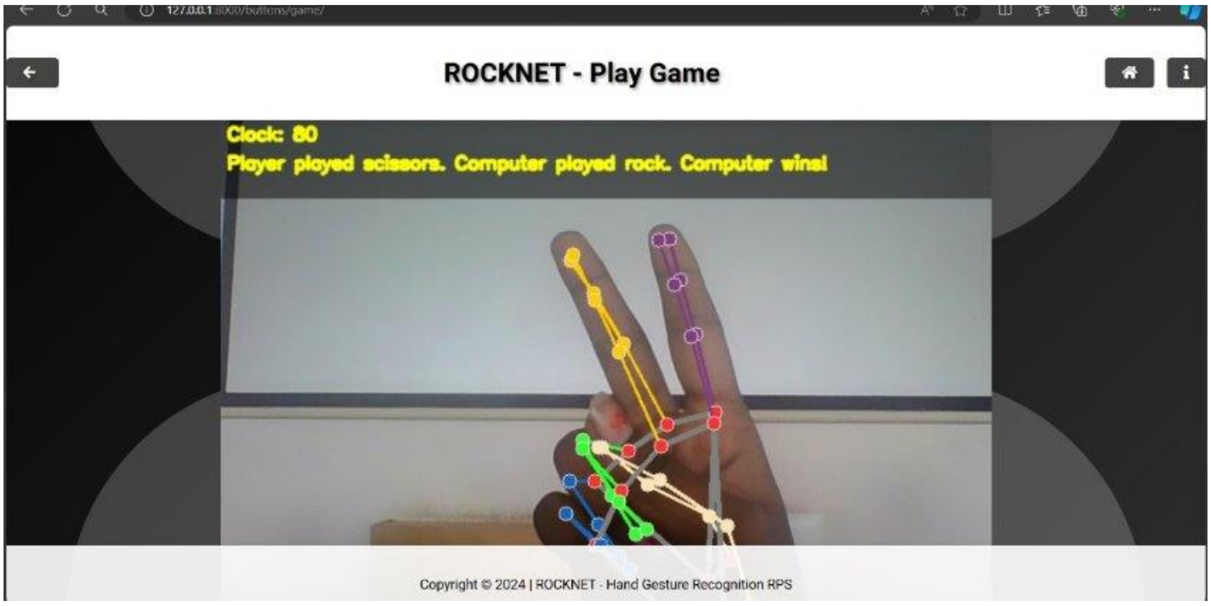

127.0.0.1:8000/buttons/game/
ROCKNET - Play Game
Clock: 80
Player played scissors. Computer played rock. Computer wins!
Copyright © 2024 | ROCKNET - Hand Gesture Recognition RPS

Figure 4.3: Play with Computer page of ROCKNET – Hand Recognition with RPS

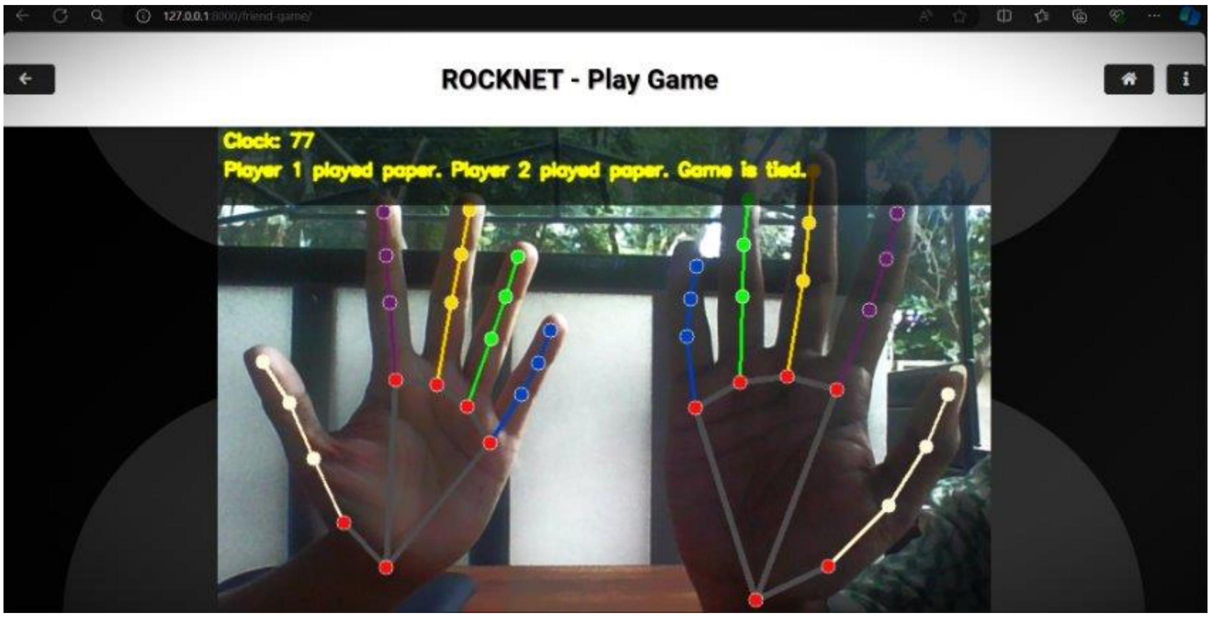
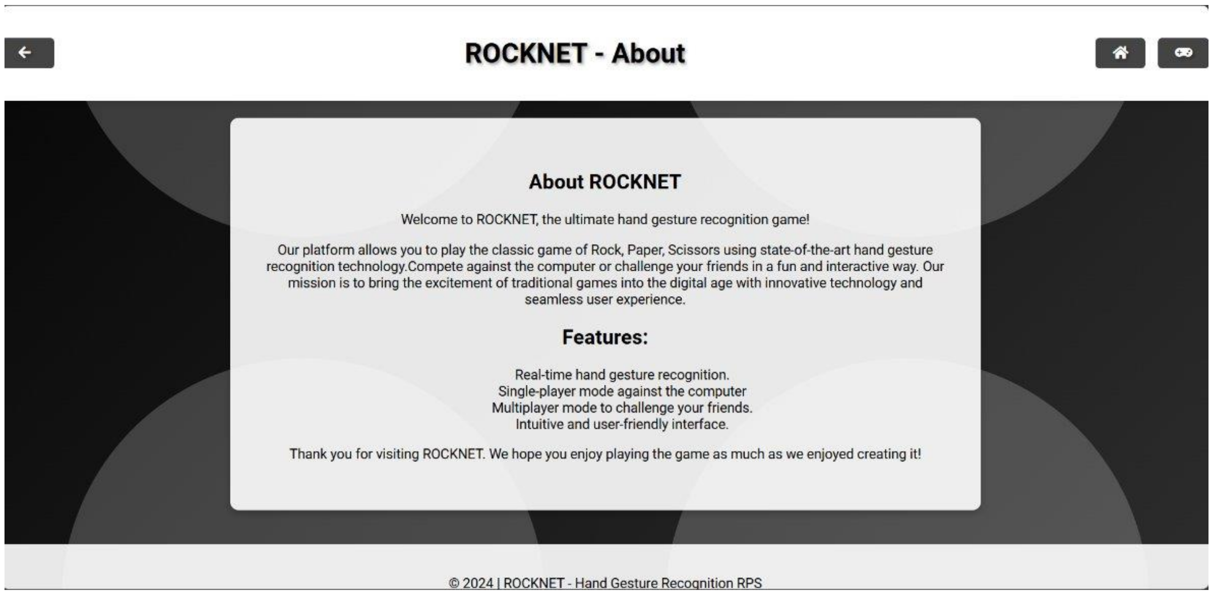

127.0.0.1:8000/friend-game/
ROCKNET - Play Game
Clock: 77
Player 1 played paper. Player 2 played paper. Game is tied.

Figure 4.4: Play with friend window of ROCKNET – Hand Recognition with RPS



About ROCKNET

Welcome to ROCKNET, the ultimate hand gesture recognition game!

Our platform allows you to play the classic game of Rock, Paper, Scissors using state-of-the-art hand gesture recognition technology.Compete against the computer or challenge your friends in a fun and interactive way. Our mission is to bring the excitement of traditional games into the digital age with innovative technology and seamless user experience.

Features:

Real-time hand gesture recognition.
Single-player mode against the computer
Multiplayer mode to challenge your friends.
Intuitive and user-friendly interface.

Thank you for visiting ROCKNET. We hope you enjoy playing the game as much as we enjoyed creating it!

© 2024 | ROCKNET - Hand Gesture Recognition RPS

Figure 4.5: About page of ROCKNET – Hand Recognition with RPS

CHAPTER - 05

CONCLUSIONS AND FUTURE WORK

5.1 CONCLUSION

RockNet serves as a notable demonstration of the application of computer vision in an interactive context. Utilizing MediaPipe's hand detection model, the system effectively identifies hand gestures through webcam input, allowing for real-time gameplay of Rock-Paper-Scissors. This project skillfully merges real-time hand pose estimation with game development principles, showcasing the potential and practicality of these advanced technologies. Through its seamless integration of traditional game mechanics and real-time user feedback, RockNet provides an engaging user experience and underscores the robust capabilities of contemporary computer vision tools in developing interactive applications.

5.2 FUTURE WORK

1. Multiplayer Online Mode

- **Develop an online multiplayer mode allowing players to compete remotely:** Implementing a multiplayer mode will enable users to connect and compete against each other over the internet, regardless of their physical location. This feature will leverage networking protocols to ensure seamless interaction and real-time gameplay between players.
- **Integrate real-time communication features such as voice and video chat:** Adding voice and video chat capabilities will enhance the multiplayer experience by allowing players to communicate directly with each other during gameplay. This integration will use WebRTC or similar technologies to ensure low-latency, high-quality communication.

2. Augmented Reality (AR) Integration

- **Incorporate AR elements to provide a more immersive gaming experience:** Integrating AR into the game will allow players to interact with virtual elements superimposed onto their real-world environment. This can be achieved using AR frameworks such as ARKit or ARCore, enhancing the overall user experience by blending the physical and digital worlds.

- **Allow players to see virtual game elements overlaid on their physical environment:** By overlaying virtual game components onto the player's surroundings, the game becomes more engaging and realistic. This feature will utilize the device's camera and sensors to accurately position and render virtual objects in real-time.

3. Enhanced Gesture Recognition

- **Implement advanced machine learning models for more accurate hand pose estimation and gesture recognition:** Utilizing sophisticated machine learning algorithms will improve the accuracy of detecting and interpreting hand gestures. This advancement will reduce errors and provide a smoother, more responsive gameplay experience.
- **Include support for additional gestures and hand poses to expand game functionality:** Expanding the repertoire of recognizable gestures will add depth to the game, allowing for more complex interactions and new gameplay mechanics. This can involve training models on a diverse dataset of hand movements to ensure robust recognition capabilities.

REFERENCES

1. https://chuoling.github.io/mediapipe/solutions/hands
2. https://www.geeksforgeeks.org/basic-transformations-opengl/
3. https://www.javatpoint.com/computer-graphics-tutorial
4. https://en.wikipedia.org/wiki/Matplotlib