**GestureUnity:**

**Webcam-Powered Control for Games**

Petrea Ana-Maria

Universitatea Dunărea de jos galați

Facultatea de: Automatică, Calculatoare, Inginerie Electrică și Electronică

Specializarea: Calculatoare și Tehnologia Informației

Email: pam130903@yahoo.com

Popa Adelina-Maria

Universitatea Dunărea de jos galați

Facultatea de: Automatică, Calculatoare, Inginerie Electrică și Electronică

Specializarea: Calculatoare și Tehnologia Informației

Email: popa\_adelyna56@yahoo.com

Radu Andreea-Denisa

Universitatea Dunărea de jos galați

Facultatea de: Automatică, Calculatoare, Inginerie Electrică și Electronică

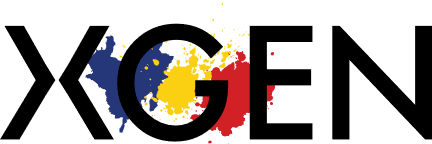
Specializarea: Calculatoare și Tehnologia Informației

Email: radu.andreea9@yahoo.com

**Abstract**

This paper presents a Unity game where hand gestures, detected via webcam and proccesed by Python, control the main character. Designed for players with motor difficulties, it eliminates the need for complex keyboard inputs, enhancing accessibility. By simplifying gameplay mechanics and making the characher move by only pointing like to the camera and letting the AI to do the rest, it offers an inclusive gaming experience without requiring game modifications. This innovative design opens up gaming experiences to a broader audience, particularly those with motor impairments, by eliminating barriers to gameplay.

**Keywords:** Unity, Python, Gesture Recognition, Accessibility



## Introduction

In a contemporary landscape marked by widespread advocacy for universal equality, it is imperative to acknowledge the essential role of equity as a precursor to its realization. Despite the global population of 8.1 billion inhabitants, humanity is almost extict. In the areas where equity is mandatory, for example schools or even working enviorments, in a world where equality is a basic human right, a considerable portion of individuals with disabilities are deprived of this fundamental entitlement. To address this disparity, our initiative endeavors to illustrate the simplicity of inclusivity through a Unity-based 2D game. Utilizing Python for gesture recognition, this interactive platform aims to demonstrate to individuals of all ages the feasibility of fostering inclusion within digital environments.

This idea emerged from a simple desire: to show people how effortless it can be to include others through an enjoyable experience. By incorporating machine learning elements often found in virtual reality games, we aim to turn this desire into reality. While browsing the internet, we encountered YouTube content creators and gamers who face the challenge of modifying games just to play them. In many instances, this task proves to be impossible or barely doable, requiring significant time and effort. These individuals often struggle to enjoy simple games without making modifications, leading to frustration and a time-consuming process. To address this issue and enhance the gaming experience, our solution involves controlling the main character through four distinct gestures: palm, rock, thumbs-up, and three-fingers up, each corresponding to a different direction. Utilizing Python, the camera detects these gestures and communicates with the Unity editor to determine the timing and direction of the character's movements.

## First Steps-Research

Unity serves as a versatile engine and framework for crafting scenes in 2D, and 3D environments, accommodating not only game development but also applications such as training simulators and business-focused tools. With Unity, interaction occurs not just through code but also via visual components, facilitating export to major mobile platforms and more, with a free version available. Integration with major 3D applications, diverse audio formats, and even Adobe Photoshop files streamlines asset assembly. Unity's hallmark feature, the Unity Asset Store, provides a rich repository of game components and assets, bolstering development with plugins, artwork, animations, audio, and more. The interface's scriptable nature allows seamless integration of third-party plugins, enhancing development workflows.

Our project embarked with a quest to streamline development solely within Unity, minimizing reliance on external tools. Fortuitously, we discovered Unity assets integrating OpenCV, a pivotal library for computer vision and machine learning. OpenCV's role in real-time operations, enabling object and facial recognition, greatly intrigued us. Integration with Python simplified analysis, leveraging its extensive libraries like NumPy for image processing and feature identification. This unified approach eliminated the need for separate Python environments, enhancing organization and workflow efficiency.

Initially, we experimented with integrating OpenCV to enable facial detection within Unity, laying the foundation for our subsequent exploration into gesture-based gameplay. Pursuing simplicity and accessibility for players, we opted for hand gestures as the primary control mechanism. After encountering challenges in initial attempts, we opted to integrate a Python editor, PyCharm, to facilitate deeper scripting capabilities and streamline development.

## Python Gesture Recognition

Python's attributes, including its object-oriented nature and dynamic semantics, align with our development goals, offering rapid application development and facilitating code maintenance. PyCharm, as a dedicated Python Integrated Development Environment (IDE), augments our workflow with essential tools for Python, web, and data science development, aligning seamlessly with Unity's capabilities and our project objectives.

For this to work, we needed OpenCV for video capturing from webcam and we used an other library for capturing and proccesing the gestures, called MediaPipe.

MediaPipe is a handy tool created by Google for making custom machine learning models that work with live and streaming media. It's still in the early stages, but it's great for building things like computer vision systems. In these systems, you can do things like figure out where objects are in a video or detect key points on a face. It works like a pipeline, where you put in video data, process it, and get useful information out the other end. We used this to help us detect when a certain gestures was made.

We based the game on four gestures: „Like”, „Open Palm”, „Three-Fingers-Up” and „Rock”.

import cv2  
import mediapipe as mp  
  
# Initialize Mediapipe Hands  
mp\_hands = mp.solutions.hands  
hands = mp\_hands.Hands(static\_image\_mode=False, max\_num\_hands=2, min\_detection\_confidence=0.5)  
mp\_drawing = mp.solutions.drawing\_utils  
  
# Initialize OpenCV's VideoCapture to start capturing from the webcam  
cap = cv2.VideoCapture(0)  
  
# Gesture labels  
gesture\_labels = {0: 'Like', 1: 'Palma', 2: 'Three Fingers Up', 3: 'Rock'}  
  
while cap.isOpened():  
 # Read each frame from the webcam  
 success, frame = cap.read()  
 if not success:  
 break  
  
 # Convert the BGR image to RGB  
 rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
  
 # Process the frame with Mediapipe Hands  
 results = hands.process(rgb\_frame)  
  
 if results.multi\_hand\_landmarks:  
 for hand\_landmarks in results.multi\_hand\_landmarks:  
 # Draw landmarks  
 mp\_drawing.draw\_landmarks(frame, hand\_landmarks, mp\_hands.HAND\_CONNECTIONS)  
  
 # Extract landmark positions  
 landmark\_positions = []  
 for landmark in hand\_landmarks.landmark:  
 landmark\_x = int(landmark.x \* frame.shape[1])  
 landmark\_y = int(landmark.y \* frame.shape[0])  
 landmark\_positions.append((landmark\_x, landmark\_y))  
  
 # Recognize gesture based on landmark positions  
 # Example: Recognize specific gestures based on finger positions  
 if len(landmark\_positions) >= 21: # Assuming at least 21 landmarks present for a full hand  
 thumb\_tip\_x, thumb\_tip\_y = landmark\_positions[4]  
 index\_finger\_tip\_x, index\_finger\_tip\_y = landmark\_positions[8]  
 middle\_finger\_tip\_x, middle\_finger\_tip\_y = landmark\_positions[12]  
 ring\_finger\_tip\_x, ring\_finger\_tip\_y = landmark\_positions[16]  
 little\_finger\_tip\_x, little\_finger\_tip\_y = landmark\_positions[20]  
  
 # Check if three fingers are up  
 three\_fingers\_up = index\_finger\_tip\_y < thumb\_tip\_y and \  
 middle\_finger\_tip\_y < thumb\_tip\_y and \  
 ring\_finger\_tip\_y < thumb\_tip\_y and \  
 little\_finger\_tip\_y > thumb\_tip\_y  
  
 # Recognize gesture based on finger positions  
 if three\_fingers\_up:  
 recognized\_gesture = gesture\_labels[2] # Three Fingers Up  
 elif thumb\_tip\_y > index\_finger\_tip\_y and \  
 thumb\_tip\_y > middle\_finger\_tip\_y and \  
 thumb\_tip\_y > ring\_finger\_tip\_y and \  
 thumb\_tip\_y > little\_finger\_tip\_y:  
 recognized\_gesture = gesture\_labels[1] # Like  
 elif thumb\_tip\_y < index\_finger\_tip\_y and \  
 thumb\_tip\_y < middle\_finger\_tip\_y and \  
 thumb\_tip\_y < ring\_finger\_tip\_y and \  
 thumb\_tip\_y < little\_finger\_tip\_y:  
 recognized\_gesture = gesture\_labels[0] # Palma  
 else:  
 recognized\_gesture = gesture\_labels[3] # Rock  
  
 # Print recognized gesture to console  
 print( recognized\_gesture)  
  
 # Display the frame  
 cv2.imshow("Hand Gesture Recognition", frame)  
  
 # Check for exit key  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release the VideoCapture and close all windows  
cap.release()  
cv2.destroyAllWindows()

This code is a Python script that uses the MediaPipe library to detect hand landmarks from a live video feed and then recognizes hand gestures based on the positions of those landmarks.

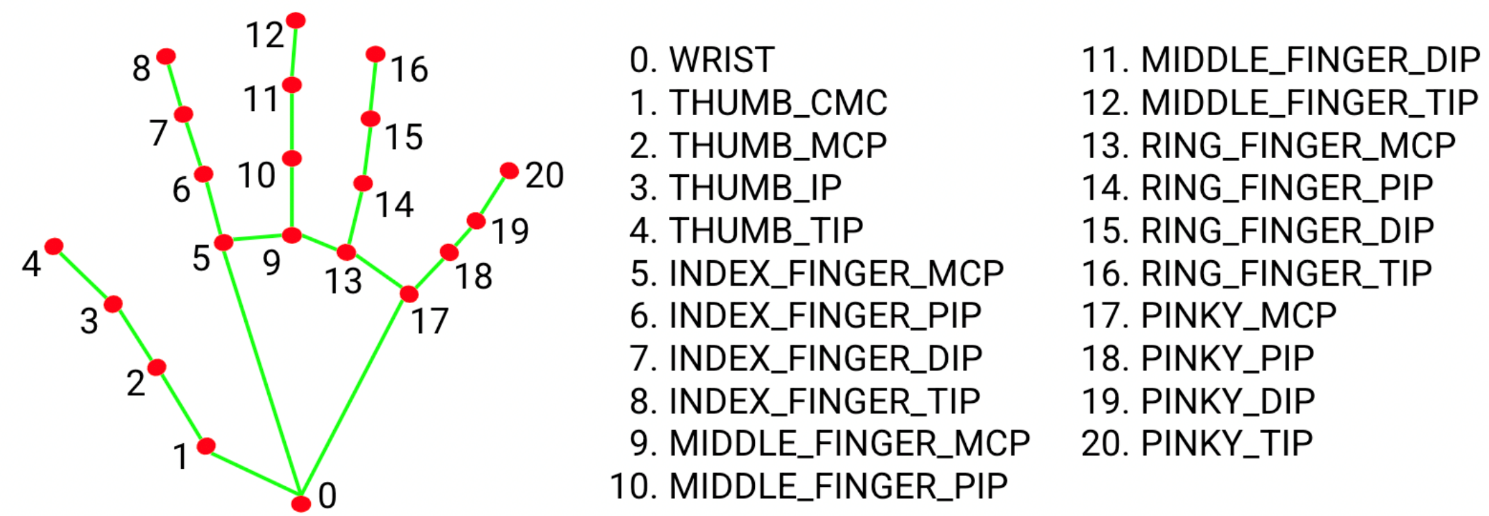
The MediaPipe library is used to detect and locate specific landmarks on a hand captured in a live video feed. These landmarks represent key points such as fingertips, knuckles, and the base of the hand. The script calculates the (x, y) coordinates of each detected landmark relative to the dimensions of the video frame. This means that the position of each landmark is represented as a fraction of the total width and height of the frame.

Fig. 1. The definition of the 21 landmarks

Autor: Google AI for Developers

Breakdown of the code:

* Drawing Landmarks: The script first draws landmarks on the detected hand in the video feed, making it easier to check exactly what it detects.
* Extracting Landmark Positions: It then extracts the positions of each landmark detected on the hand. These positions are stored as (x, y) coordinates relative to the dimensions of the frame.
* Recognizing Gestures: Next, the script checks the positions of specific landmarks corresponding to the thumb, index finger, middle finger, ring finger, and little finger. It uses these positions to determine the hand gesture.
* Three Fingers Up: This gesture is recognized when the tips of the index, middle, and ring fingers are positioned below the tip of the thumb, and the tip of the little finger is positioned above the thumb. This configuration suggests that three fingers are raised while the thumb is lowered, resembling the gesture of holding up three fingers.
* Like Gesture: If the position of the thumb tip is higher than all other finger tips, it recognizes the gesture as a "Like" gesture.
* Palm Gesture: If the position of the thumb tip is lower than all other finger tips, it recognizes the gesture as a "Palm" gesture.
* Rock Gesture: If none of the above conditions are met, it recognizes the gesture as a "Rock" gesture.
* Printing Recognized Gesture: Based on the relative positions of landmarks, the script categorizes the detected hand gesture into one of these predefined categories. Finally, the recognized gesture is printed to the console, allowing for real-time feedback on the detected hand gesture.

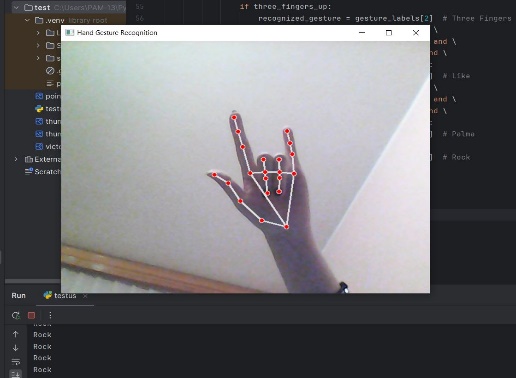
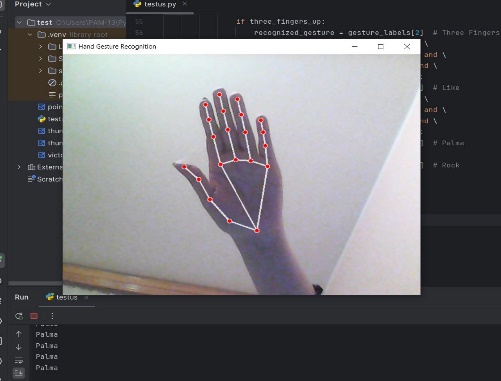
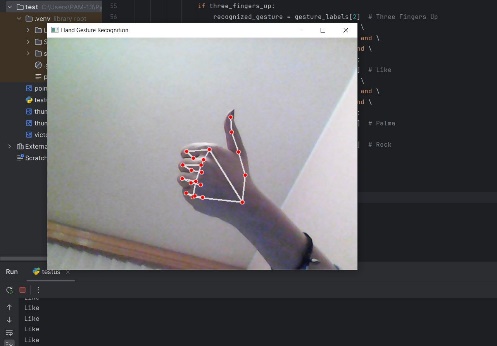
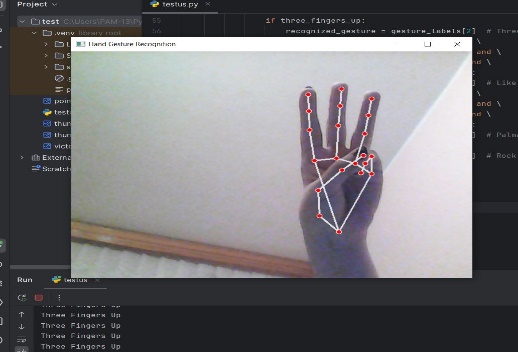


Fig. 2. The four gestures

Autor: Petrea Ana-Maria

In summary, this code leverages the relative positions of landmarks detected on a hand to infer the corresponding gesture being performed. By comparing these positions against predefined criteria for different gestures, the script determines the most likely gesture and provides feedback accordingly.

## Making the Game

In Unity, we've begun developing our game, which centers around a teddy bear named Orion navigating through a forest. During his journey, Orion discovers a mysterious coded message. Soon after, a robot named Vega, dispatched by the Universe to seek out the hero of life, appears to assist Orion in deciphering the message. This message holds the key to combating an evil alien entity known as Shadow, along with his spectral followers, the ghosts.During his quest to decode the message, Orion traverses the universe, encountering and aiding various civilizations along the way. Each civilization offers Orion valuable pieces of information or artifacts that contribute to unraveling the mystery encoded in the message. These encounters not only assist Orion in his mission but also provide opportunities for him to learn and grow as he interacts with diverse cultures and beings across the cosmos.

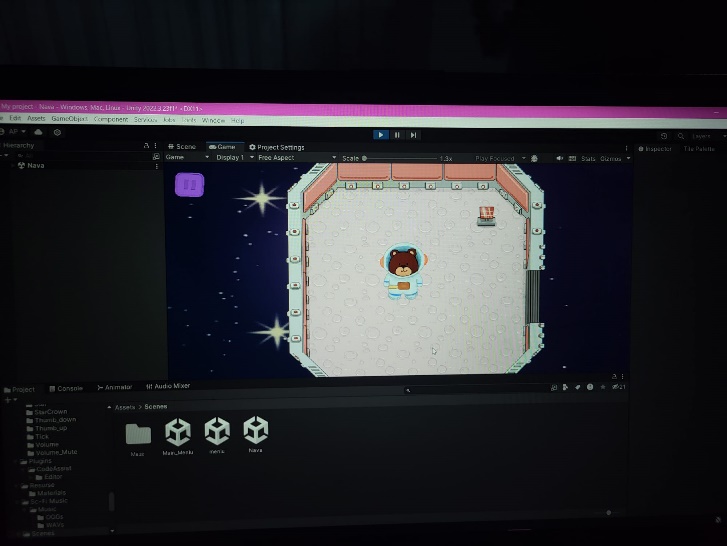


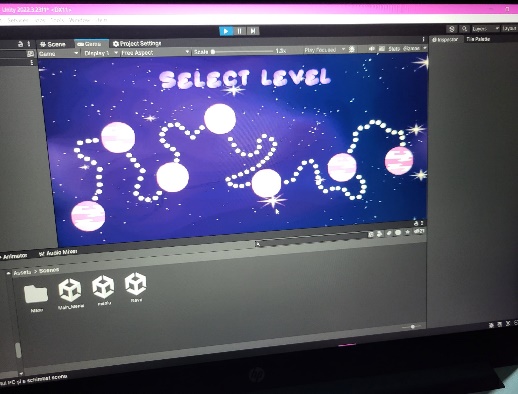
Fig. 3. A peek in the game

Autor: Popa Adelina

We propose a method to improve the gameplay experience of a Unity 2D game by introducing dynamically generated mazes as puzzles in each level. The main objective is to offer players unique and challenging puzzles that contribute to their overall enjoyment and engagement with the game. Our approach involves integrating maze generation algorithms into the level design process. These algorithms, such as Recursive Backtracking, Prim's Algorithm, and Kruskal's Algorithm, allow us to create maze layouts dynamically, ensuring that each level presents a fresh and unpredictable challenge for players. The generated mazes serve as the primary gameplay element, requiring players to navigate through them to progress in the game. We implement player interaction mechanics, including movement controls and obstacle avoidance, to facilitate navigation through the maze. Additionally, we populate the mazes with various obstacles, collectables and enemies to increase the challenge and keep players engaged. Overall, our approach demonstrates the effectiveness of dynamically generated mazes in enhancing the gameplay experience of a Unity 2D game. By adding depth, variety, and challenge to the puzzles, we aim to increase player engagement and enjoyment, ultimately contributing to the success of the game.

By integrating maze generation algorithms like Recursive Backtracking, Prim's Algorithm, and Kruskal's Algorithm into the level design process, each level offers a fresh challenge for players.

Recursive Backtracking, a recursive method, starts from a random cell, recursively exploring unvisited neighboring cells until all cells are visited, resulting in mazes with winding corridors and dead ends. Prim's Algorithm expands the maze from a starting cell, integrating frontier cells into the maze until all cells are connected, producing mazes with a minimum spanning tree structure. Kruskal's Algorithm, starting with independent cells, merges sets of neighboring cells until a single set remains, resulting in mazes with a more uniform distribution of corridors.

* Recursive Backtracking: It begins by selecting a random cell within the grid and marking it as visited. From this initial cell, the algorithm recursively explores neighboring cells that have not been visited yet. It continues this process until it reaches a dead end where no unvisited neighboring cells remain. At this point, the algorithm backtracks to the previous cell and explores other unvisited paths. This process repeats until all cells in the grid have been visited, resulting in mazes characterized by winding corridors and dead ends.
* Prim's Algorithm: It operates as a greedy algorithm for maze generation. It starts with a designated starting cell and gradually expands the maze outward. Initially, all cells except the starting cell are considered walls. The algorithm maintains a list of frontier cells, which are adjacent to the current maze but not yet included in it. Randomly selecting frontier cells, Prim's Algorithm integrates them into the maze by removing the walls between them and the existing maze. This process continues until all frontier cells have been assimilated, creating a fully connected maze with a minimum spanning tree structure.
* Kruskal's Algorithm: It begins with each cell as an independent set. It targets walls between neighboring cells and examines if these walls belong to different sets. If so, the walls are removed, and the sets are merged. This process repeats until only one set remains, resulting in a fully connected maze. Kruskal's Algorithm tends to produce mazes with a more uniform distribution of corridors, making it suitable for generating mazes with balanced and evenly distributed paths.

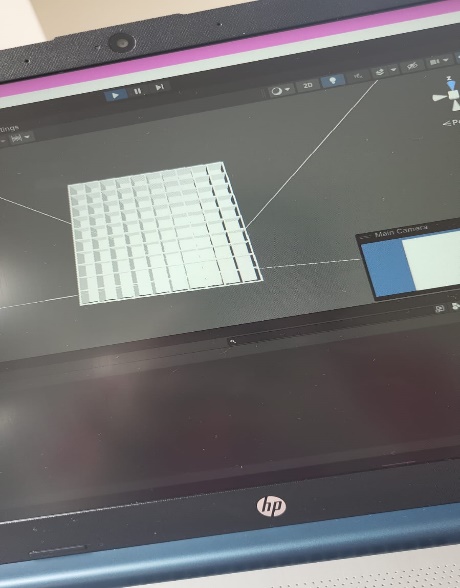


Fig. 4. The Maze and Level Selection

Autor: Popa Adelina

## The link between Unity and Python

Unity and Python communication can be achieved using network communication, specifically TCP/IP sockets. This allows communication between the two independently running applications, each written in a different programming language. In a typical scenario, Unity acts as the server, listening on a specific port for connections from other applications. Python can act as the client, connecting to this port and sending or receiving data from Unity.

What each one does:

Unity (server):

* A Unity application can create a TCP/IP socket server using the TcpListener class from the System.Net.Sockets namespace.
* The Unity server waits for connections from clients, usually in a separate thread to avoid blocking the main thread of the application.
* When a connection is established, the server receives and interprets the data received from the client.

Python (client):

* A Python application can create a TCP/IP socket client using the socket module.
* The Python client connects to the specified IP address and port where the Unity server is running.
* Once the connection is established, the client can send and receive data to and from the Unity server.

Once the connection is established between the two applications, they can exchange data using methods like send and recv, typically in string or byte string formats. This data can encompass various information or commands crucial for controlling or coordinating between the applications, such as recognized hand gestures, game data, or the states of different objects. This connectivity enables developers to seamlessly integrate Python functionalities into Unity projects and vice versa. Python's strengths in data processing and interaction with external devices complement Unity's capabilities in creating immersive 3D graphics and simulations. Together, this integration offers developers a broad spectrum of possibilities, empowering them to create interactive experiences that harness the strengths of both environments.

## Conclusions

Today, people everywhere are pushing for fairness and equal treatment for everyone. But some folks, especially those with disabilities, still don't get the same chances as others, like going to school or having a fair shot at work. To fix this, a group came up with a cool idea: they made a simple game using Unity and Python. This game helps show how easy it is to include everyone, no matter who they are.

They started by learning about Unity, which is great for making all kinds of games and simulations. They found out they could use Python with Unity to recognize hand movements, which is important for their game. Python is handy because it's easy to work with and can do things quickly. They used it along with a tool called MediaPipe to spot hand gestures in videos. The game they made is about a teddy bear named Orion exploring a forest. Along the way, he faces challenges, like solving puzzles in mazes. These mazes are made differently each time you play, so it's always a new challenge. To connect Unity and Python, they used a method called TCP/IP sockets. Think of Unity as the server, like the host of a party, and Python as the guest. They chat with each other over the internet to make the game work smoothly.

By combining Unity's cool graphics and Python's smarts, they made a game that's fun for everyone and shows how important it is to include everyone, no matter what.

## Webography & Bibliography

|  |  |
| --- | --- |
| https://ai.google.dev/edge/mediapipe/solutions/vision/gesture\_recognizer  https://viso.ai/computer-vision/mediapipe/  https://youtu.be/RQ-2JWzNc6k?si=NYBEij2rLBssuaIW  https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/  https://www.fortinet.com/resources/cyberglossary/tcp-ip  Forouzan, B.A. , TCP/IP Protocol Suite (3rd Ed), McGraw-Hill, 2000  Naugle, M., Illustrated TCP/IP, Wiley Computer Publishing, 1998  Tanenbaum, A.S., Reţele de calculatoare, Agora, 1998 |  |