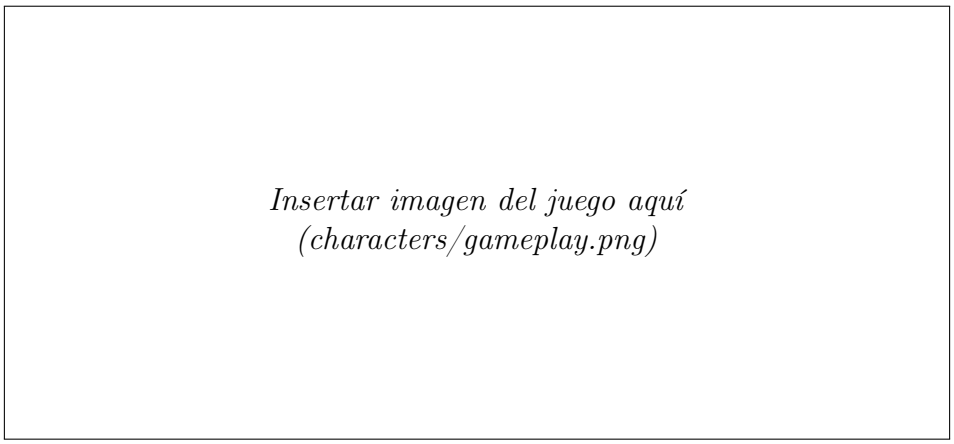
VISION WALL

Sistema Interactivo de Evaluación de Posturas en Tiempo Real



Proyecto Final

Asignatura: Visión por Computador

Autores:

Nicolás Rey Alonso
Wafa Azdad Triki

Tecnologías:

Godot 4.x • MediaPipe • Blender • GDScript

7 de enero de 2026

Índice

1. Introducción

Este proyecto implementa un sistema interactivo de evaluación de posturas corporales en tiempo real inspirado en el famoso programa de televisión japonés "Hole in the Wall". El sistema utiliza detección de pose mediante MediaPipe y un motor gráfico 3D (Godot) para crear una experiencia de juego inmersiva.

1.1. Objetivo General

Desarrollar un sistema que evalúe posturas corporales completas en tiempo real, permitiendo a los jugadores intentar atravesar paredes con agujeros en forma de figuras humanas. El sistema debe:

- Detectar la postura del jugador en tiempo real
- Generar paredes con agujeros en diferentes formas humanas
- Validar si la postura del jugador coincide con el agujero
- Proporcionar retroalimentación visual e interactiva
- Mantener un sistema de puntuación y vidas

2. Tecnologías Utilizadas

2.1. Motor Gráfico: Godot 4.x

Godot es un motor de juegos de código abierto que proporciona:

- Soporte para Node3D y MeshInstance3D para gráficos 3D
- Sistema de materiales y shaders avanzados
- Gestión de escenas y nodos jerárquicos
- GDScript para scripting en el motor

2.2. Detección de Pose: MediaPipe

MediaPipe es un framework de Google para visión por computadora que:

- Detecta 33 puntos de referencia en el cuerpo humano
- Opera en tiempo real con bajo latency
- Se comunica mediante WebSocket en localhost:8765
- Proporciona coordenadas 3D de cada articulación

2.3. Modelado 3D: Blender

Se utilizó Blender para:

- Crear y rigear 4 personajes diferentes (Saw, ET, Eleven, Homer Simpson)
- Exportar modelos en formato .glb con esqueleto Armature
- Aplicar materiales y texturas

2.4. GDScript

Lenguaje de scripting nativo de Godot utilizado para:

- Lógica del juego
- Comunicación WebSocket con MediaPipe
- Generación de geometría de paredes
- Control de animaciones

3. Arquitectura del Sistema

3.1. Estructura del Proyecto

```
BrainWallGodot/
 brain-wall/
    project.godot
    scenes/
       character_select.gd
       mainscript.gd
       PoseEvaluator.gd
       WallGenerator.gd
       PoseReceiver.gd
       GameHUD.gd
       main_menu.tscn
       mainScene.tscn
    assets/
        models/
        shaders/
        music/
        SoundEffects/
```

3.2. Flujo de Datos

1. **Captura de Video**: Cámara web captura al jugador
2. **MediaPipe**: Procesa el video y detecta 33 puntos de pose
3. **WebSocket**: Envía datos de pose a Godot (localhost:8765)

4. **PoseReceiver**: Recibe datos y los procesa
5. **Skeleton Animation**: Anima el personaje con los datos de pose
6. **WallGenerator**: Genera paredes y valida colisiones
7. **GameHUD**: Muestra puntuación, vidas y postura actual

4. Componentes Principales

4.1. 1. character_select.gd

Sistema de selección de personajes para 2 jugadores.

4.1.1. Funcionalidades

- Interfaz de selección visual para 4 personajes
- Selección secuencial: Jugador 1 → Jugador 2
- Almacenamiento de selecciones en metadatos de escena
- Auto-transición a escena principal tras completar selección

4.1.2. Flujo

1. Mostrar 4 opciones de personajes
2. Jugador 1 selecciona personaje
3. Jugador 2 selecciona personaje
4. Cargar escena principal (mainScene.tscn)

4.2. 2. mainscript.gd

Script controlador principal del juego.

4.2.1. Variables Globales

```
- game_over: bool
- score: int
- lives: int
- current_pose_type: String
- ws: WebSocketPeer
- skeleton_nodes: Array (2 personajes)
```

4.2.2. Funciones Principales

- **_ready()**: Inicialización de escena, plataforma, cámara, luces, personajes
- **_process(delta)**: Loop principal del juego
- **connect_websocket()**: Conexión a localhost:8765
- **update_player_animation()**: Animar personajes con datos de pose
- **update_wall_system()**: Gestionar movimiento de paredes y colisiones
- **check_wall_collision()**: Validar si jugador pasó a través del agujero
- **create_platform()**: Crear plataforma de juego

4.2.3. Posiciones en Escena

- Jugador 1: (-2, 1, 0) - Verde
- Jugador 2: (2, 1, 0) - Magenta
- Cámara: (0, 2, 8)
- Plataforma: y=0, dimensiones 10×0.5×5

4.3. 3. WallGenerator.gd

Sistema de generación y movimiento de paredes con agujeros en forma de figuras humanas.

4.3.1. Variables de Exportación

```
@export var wall_spawn_interval: float = 3.0
@export var wall_speed: float = 5.0
```

4.3.2. Posturas Disponibles

1. **T_POSE**: Brazos extendidos horizontalmente
2. **ARMS_UP**: Brazos levantados verticalmente
3. **ARMS_DOWN**: Brazos bajados
4. **LEFT_ARM_UP**: Brazo izquierdo arriba
5. **RIGHT_ARM_UP**: Brazo derecho arriba
6. **SQUAT**: Agachado
7. **JUMP**: Saltando

4.3.3. Generación de Paredes

Cada pared es un Node3D que contiene:

- **Posición inicial**: (0, 2, 20)
- **Movimiento**: -5.0 unidades/segundo en eje Z
- **Geometry**: Malla generada con SurfaceTool
- **Material**: StandardMaterial3D azul transparente (0.2, 0.2, 0.8, 0.8)
- **Limpieza**: Se elimina cuando z <-30

4.3.4. Geometría de Paredes

Las paredes se generan usando SurfaceTool que:

- Crea rectángulos sólidos en posiciones específicas
- Deja agujeros vacíos donde el personaje debe pasar
- Utiliza la función add_rect() para construir la geometría
- Genera normales automáticamente con generate_normals()

4.4. 4. PoseReceiver.gd

Recibe datos de pose del servidor MediaPipe via WebSocket.

4.4.1. Estructura de Datos

Los datos de MediaPipe incluyen:

```
{
  "pose_landmarks": [
    {"x": float, "y": float, "z": float, "visibility": float},
    ... (33 puntos de referencia)
  ],
  "pose_world_landmarks": [...],
  "pose_type": "T_POSE" | "ARMS_UP" | ... (7 tipos)
}
```

4.4.2. Puntos de Referencia Utilizados

Se utilizan 8 puntos clave del esqueleto:

1. Cuello (Neck) - promedio de hombros
2. Hombro izquierdo (Left Shoulder)
3. Hombro derecho (Right Shoulder)
4. Codo izquierdo (Left Elbow)

5. Codo derecho (Right Elbow)
6. Muñeca izquierda (Left Wrist)
7. Muñeca derecha (Right Wrist)
8. Cadera (Hip) - promedio de caderas

4.5. 5. GameHUD.gd

Sistema de interfaz de usuario que muestra:

- Puntuación actual
- Vidas restantes
- Postura actual detectada

5. Sistemas de Juego

5.1. Sistema de Puntuación

- **Puntos por pared**: +10 puntos si se pasa correctamente
- **Penalización**: -1 vida si se choca con la pared
- **Vidas iniciales**: 3 vidas por jugador
- **Game Over**: Cuando las vidas llegan a 0

5.2. Sistema de Colisión

Las paredes se validan cuando:

- Posición Z de la pared está entre 1.0 y -1.0
- Sistema verifica si el jugador está dentro del área del agujero
- Si está dentro: +10 puntos
- Si está fuera: -1 vida

5.3. Animación de Personajes

Cada personaje tiene un Skeleton3D con 8 huesos que se animan:

- Se capturan rotaciones de MediaPipe
- Se interpolan las rotaciones con slerp() para suavidad
- Se actualiza cada frame según los datos de pose
- Los huesos se mapean a puntos de referencia de MediaPipe

5.4. Generación de Paredes

- **Intervalo**: Cada 3 segundos se genera una nueva pared
- **Postura**: Se selecciona aleatoriamente de las 7 disponibles
- **Posición inicial**: (0, 2, 20) - frente al jugador
- **Movimiento**: -5.0 unidades/segundo hacia el jugador
- **Limpieza**: Se elimina cuando sale del rango visible (z <-30)

6. Detalles Técnicos

6.1. Comunicación WebSocket

```
# Cliente: localhost:8765
# Protocolo: WebSocket con JSON
# Frecuencia: 30 FPS (aproximadamente)
# Datos: Pose 3D del cuerpo, tipo de postura detectada

Ejemplo de mensaje recibido:
{
  "pose_landmarks": [
    {"x": 0.5, "y": 0.2, "z": -0.1, "visibility": 0.98},
    ...
  ],
  "pose_type": "T_POSE"
}
```

6.2. Mapeo de Huesos

```
MediaPipe → Skeleton3D
11 (Left Shoulder) → bone_0
12 (Right Shoulder) → bone_1
13 (Left Elbow) → bone_2
14 (Right Elbow) → bone_3
15 (Left Wrist) → bone_4
16 (Right Wrist) → bone_5
23 (Left Hip) → bone_6
24 (Right Hip) → bone_7
```

6.3. Generación de Geometría

Las paredes se generan con SurfaceTool:

```
func add_rect(st: SurfaceTool, x1, y1, x2, y2):
    Crea dos triángulos formando un rectángulo
    - Vértice 1: (x1, y1, z)
    - Vértice 2: (x2, y1, z)
```

```
- Vértice 3: (x2, y2, z)
- Vértice 4: (x1, y2, z)

Triángulo 1: 1-2-3
Triángulo 2: 1-3-4
```

6.4. Material de Paredes

```
StandardMaterial3D:
  - albedo_color: (0.2, 0.2, 0.8, 0.8) - Azul transparente
  - transparency: TRANSPARENCY_ALPHA
  - cull_mode: CULL_DISABLED
```

7. Flujo de Ejecución

7.1. Inicio del Juego

1. Usuario abre aplicación
2. Selecciona 2 personajes (character_select.gd)
3. Se carga mainScene.tscn
4. mainscript.gd inicializa:
   - Crea plataforma
   - Posiciona cámara y luces
   - Carga y posiciona personajes
   - Conecta a WebSocket (localhost:8765)
   - Inicializa WallGenerator
   - Crea HUD

7.2. Loop Principal (_process)

1. Procesar eventos WebSocket
2. Actualizar animación de personajes con pose recibida
3. Mover paredes hacia jugadores
4. Validar colisiones con paredes
5. Actualizar HUD (puntuación, vidas, postura)
6. Generar nuevas paredes cada 3 segundos
7. Limpiar paredes fuera de rango

7.3. Validación de Colisión

1. Pared se acerca al jugador (Z disminuye)
2. Cuando pared.z está entre 1.0 y -1.0:
3. Verificar si jugador está dentro de los límites del agujero
4. Si SÍ: Otorgar +10 puntos
5. Si NO: Restar -1 vida
6. Desmarcar pared como validada

8. Configuración de Exportación

8.1. Exportables

```
WallGenerator.gd:
  @export wall_spawn_interval: float = 3.0
  @export wall_speed: float = 5.0

Personajes:
  - Posición 1: (-2, 1, 0)
  - Posición 2: (2, 1, 0)
```

9. Personajes Disponibles

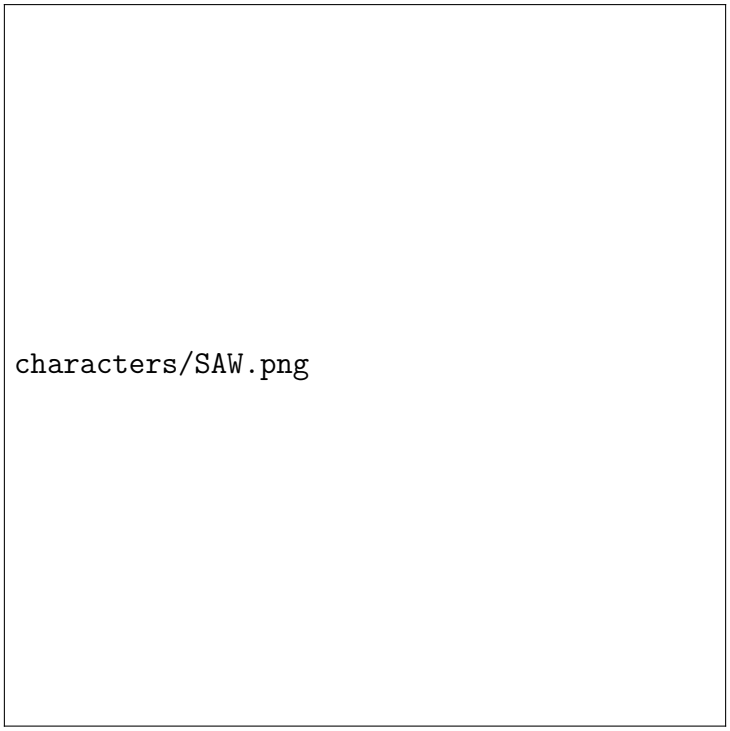
El juego incluye 4 personajes basados en series y películas, cada uno con su propio modelo 3D rigeado y animación personalizada. Fueron modelados y rigeados en Blender usando el complemento Rigify para asegurar un esqueleto completo.

9.1. Personaje 1: SAW

Descripción: Personaje inspirado en el villano de la película de terror "Saw" (2004). Representa un personaje oscuro y misterioso con características intimidantes y expresión seria.

Origen: Película de Cine de Horror - "Saw"

- **Archivo**: Saw.blend
- **Tipo de Modelo**: Humanoide
- **Características**: Expresión grave, indumentaria oscura

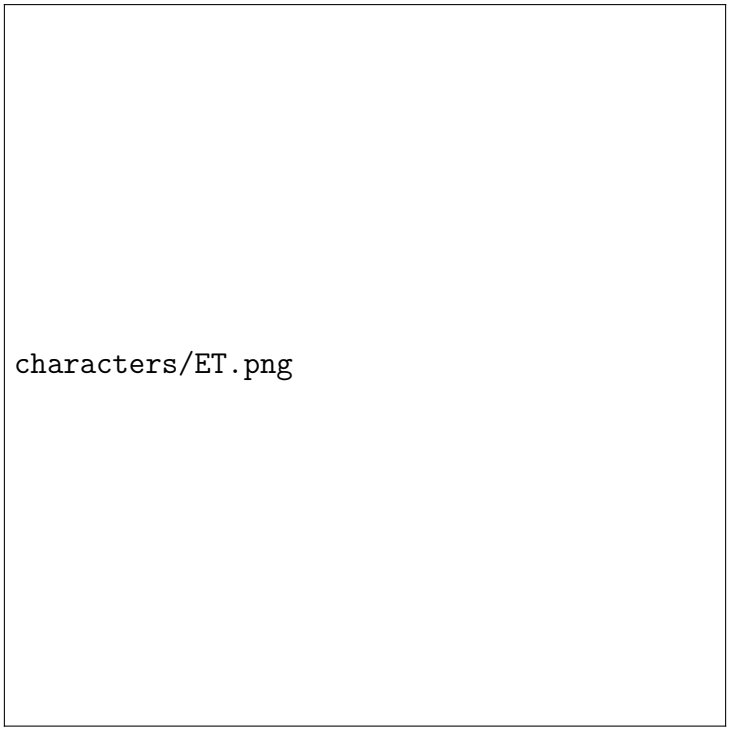


9.2. Personaje 2: ET

Descripción: Personaje alienígena inspirado en el famoso extraterrestre de la película clásica "E.T. the Extra-Terrestrial" (1982). Proporciona una experiencia visual única y fantástica.

Origen: Película de Ciencia Ficción - "E.T. the Extra-Terrestrial"

- **Archivo**: ET.blend
- **Tipo de Modelo**: Humanoide fantástico
- **Características**: Silueta única, proporciones exageradas, piel verdosa



9.3. Personaje 3: ELEVEN

Descripción: Personaje inspirado en la serie de Netflix “Stranger Things” (2016-presente). Representa un personaje joven con presencia cautivadora y características distintivas.

Origen: Serie de Televisión - “Stranger Things”

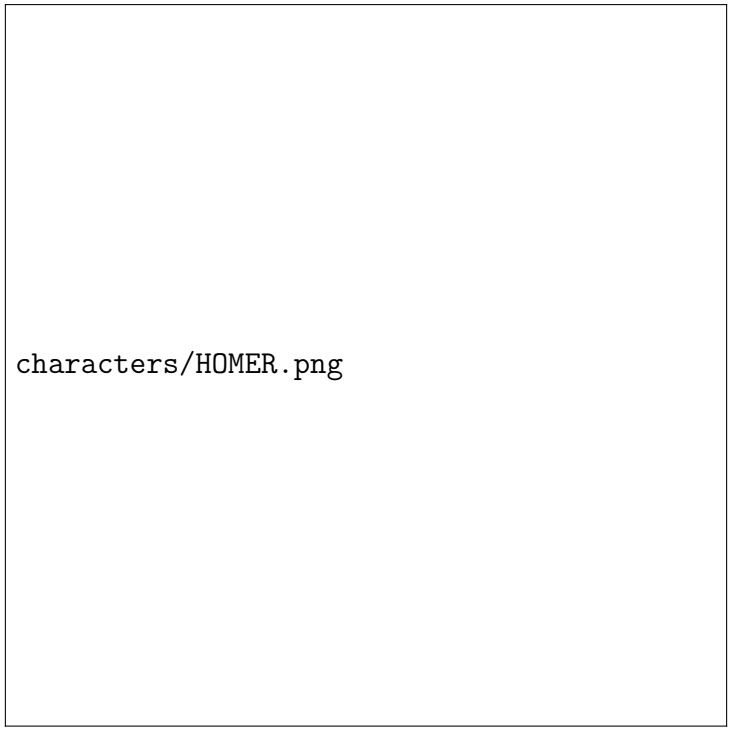
- **Archivo**: Eleven2.blend
- **Tipo de Modelo**: Humanoide
- **Características**: Proporciones juveniles, diseño visual moderno

9.4. Personaje 4: HOMER

Descripción: Personaje inspirado en el icónico "Homer Simpson" de la serie animada "The Simpsons" (1989-presente). Añade humor y familiaridad al juego con proporción cómica y humorística.

Origen: Serie de Televisión Animada - "The Simpsons"

- **Archivo**: HOMER_SIMPSON.blend
- **Tipo de Modelo**: Humanoide caricaturesco
- **Características**: Cuerpo rechoncho, color amarillo característico, proporciones cómicas



9.5. Sistema de Rigging

Todos los personajes fueron rigeados utilizando el siguiente proceso:

1. Importar modelo base en Blender
2. Aplicar complemento Rigify para generar esqueleto automático
3. Ajustar y refinar los pesos de deformación (skinning)
4. Exportar en formato .glb con Armature para compatibilidad con Godot

10. Sistema de Audio y Sonidos

El juego incluye un sistema completo de audio diseñado para mejorar la experiencia de juego mediante retroalimentación sonora inmersiva. Todos los efectos de sonido fueron creados manualmente.

10.1. Música de Fondo

10.1.1. Tema Principal

- **Archivo**: turiruriru-01.mp3
- **Duración**: Loop continuo durante el juego
- **Descripción**: Música electrónica/pop japonesa que crea una atmósfera divertida y energética
- **Formato**: MP3 comprimido para optimizar almacenamiento
- **Volumen**: Configurado para no sobrepasar la retroalimentación de sonidos de efecto

10.2. Efectos de Sonido

Todos los efectos de sonido fueron creados manualmente en el software de edición de audio MuseScore (formato .mmpz).

10.2.1. Sonido de Click en Botón

- **Archivo**: soundEffect_buttonClick.ogg
- **Creación**: Generado manualmente en MuseScore
- **Uso**: Se reproduce al hacer click en cualquier botón del menú o interfaz
- **Duración**: 0.2 segundos
- **Descripción**: Sonido seco y percusivo que indica selección
- **Formato Original**: soundEffect_buttonClick.mmpz (MuseScore)
- **Archivo de Backup**: soundEffect_buttonClick.mmpz.bak

10.2.2. Sonido de Hover en Botón

- **Archivo**: soundEffect_buttonHover.ogg
- **Creación**: Generado manualmente en MuseScore
- **Uso**: Se reproduce al pasar el cursor sobre un botón
- **Duración**: 0.15 segundos
- **Descripción**: Sonido suave y ascendente que indica posibilidad de interacción
- **Formato Original**: soundEffect_buttonHover.mmpz (MuseScore)

10.3. Integración en Godot

Los archivos de audio se integran en el motor Godot mediante:

- **AudioStreamPlayer**: Nodo responsable de reproducir sonidos
- **Carpeta**: assets/music/ y assets/SoundEffects/
- **Importación automática**: Godot convierte automáticamente .mmpz a .ogg para optimización
- **Control de volumen**: Ajustable durante el gameplay

10.4. Estructura de Archivos de Audio

```
assets/
 music/
    turiruriru-01.mp3.import
 SoundEffects/
    soundEffect_buttonClick.ogg.import
    soundEffect_buttonHover.ogg.import

Archivos de edición (raíz del proyecto):
 turiruriru-01.mmpz
 soundEffect_buttonClick.mmpz
 soundEffect_buttonClick.mmpz.bak
 soundEffect_buttonHover.mmpz
```

10.5. Proceso de Creación de Sonidos

El flujo de creación de efectos de sonido fue:

1. Crear composición básica en MuseScore
2. Seleccionar instrumentos/tonos para cada efecto
3. Ajustar duración y volumen
4. Exportar como .ogg para compatibilidad con Godot
5. Guardar archivo .mmpz original para futuras ediciones
6. Crear backup (.bak) para seguridad
7. Importar en Godot y ajustar parámetros de audio

11. Galería del Juego

Esta sección incluye información sobre los elementos visuales del juego:

11.1. Escenas Principales

- **Menú de Selección de Personajes**: Interfaz visual para elegir entre 4 personajes
- **Gameplay Principal**: Vista 3D con 2 personajes en la plataforma
- **Paredes Dinámicas**: Diferentes formas de agujeros según la postura
- **Interfaz HUD**: Displays de puntuación, vidas y postura actual

11.2. Elementos Visuales

- **Plataforma**: Base gris oscuro 10x0.5x5 unidades
- **Paredes**: Azul transparente (RGBA: 0.2, 0.2, 0.8, 0.8)
- **Iluminación**: Sistema de luces 3D para ambiente realista
- **Cámara**: Posición fija en (0, 2, 8) para vista de frente

Nota: Las imágenes pueden insertarse aquí usando: \includegraphics[width=0.8\textwidth]{ruta/imag

12. Limitaciones y Mejoras Futuras

12.1. Limitaciones Actuales

- La detección de pose depende de la calidad de la cámara
- Requiere conexión WebSocket a servidor MediaPipe
- Las paredes tienen un tamaño fijo
- No hay validación de profundidad (solo área 2D)

12.2. Mejoras Futuras

- Integrar IA para adaptar dificultad
- Modo multijugador en red
- Tablas de puntuaciones persistentes
- Modelos de personajes más detallados
- Sistema de combos y multiplicadores
- Modos de juego adicionales (tiempo limitado, etc.)
- Más variedad de efectos de sonido
- Sistema de partículas para colisiones

13. Conclusión

Este proyecto demuestra la integración exitosa de:

- Detección de pose en tiempo real con MediaPipe
- Renderizado 3D con Godot
- Animación de personajes basada en datos capturados
- Sistemas de juego interactivos

- Comunicación cliente-servidor mediante WebSocket
- Audio personalizado creado manualmente
- Múltiples personajes con rigging avanzado

El sistema proporciona una experiencia inmersiva y divertida inspirada en el programa televisivo “Hole in the Wall”, permitiendo a los jugadores desafiar sus habilidades de contorsión mientras interactúan con paredes de formas diversas.