UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

MEMORIA DESCRIPTIVA DEL PROYECTO

Trabajo presentado por:

- William Herderson Choquehuanca Berna
- Juan Sergio Zeballos Perez
- Wilson Josue Turpo Huanca

Curso:
TECNOLOGÍA DE OBJETOS

Fecha de entrega:
Hasta el lunes 1-12-2025
a las 18:00 horas

AREQUIPA-PERÚ

2025

ÍNDICE

MEMORIA DESCRIPTIVA DEL PROYECTO

1. Introducción y contexto

1.1 Contexto

El aprendizaje de la Lengua de Señas Peruana (LSP) requiere recursos didácticos accesibles y prácticas frecuentes para su correcta adquisición. Las tecnologías de visión por computadora y detección de landmarks (p. ej. MediaPipe) permiten crear herramientas que detecten gestos en tiempo real usando hardware común (webcams). Este proyecto refactoriza un sistema previo (srlsp-game) hacia una arquitectura modular, orientada a reutilización y fácil extensión, integrando reconocimiento gestual con actividades lúdicas (juegos serios) para facilitar el aprendizaje de la LSP.

1.2 Problemática

- Escasez de herramientas educativas accesibles para aprender LSP.
- Las soluciones existentes requieren hardware especializado o no ofrecen retroalimentación interactiva.
- Necesidad de un sistema modular que permita: captura estable de video, detección en tiempo real, desacoplamiento entre sensor/procesamiento/UI y reutilización de la lógica de juego.

1.3 Objetivos

1.3.1 Objetivo general

Refactorizar y consolidar un sistema modular que permita capturar y reconocer señas del alfabeto manual peruano en tiempo real e integrarlas en juegos educativos, garantizando rendimiento, mantenibilidad y escalabilidad.

1.3.2 Objetivos específicos

1. Diseñar una arquitectura desacoplada con EventBus para comunicación entre módulos.

2. Implementar módulos multihilo para captura (camera) y procesamiento (detector) que no bloqueen la UI.

3. Encapsular la lógica de cada juego en clases reutilizables (clase_* + juego_*).

4. Crear GUI central (MainWindow) que administre hilos, preview de cámara y lanzadores de juegos.

5. Persistir resultados y métricas en SQLite.

6. Proveer tests básicos y herramientas de diagnóstico (quick_test_debug.py, tests/test_detector.py).

1.4 Alcance

Incluye:

- Captura por webcam, procesamiento en tiempo real usando MediaPipe/OpenCV, reconocimiento de letras del alfabeto.

- Tres juegos integrados (AH, LC, LADRILLOS) con lógica separada.

- GUI principal que administra cámara y lanza juegos.

- Persistencia de puntuaciones (SQLite).

Excluye (por ahora):

- Reconocimiento de palabras continuas o frases.

- Entrenamiento de modelos desde cero (se usa clasificador preexistente).

- Soporte multiplataforma intensivo (solo pruebas en entorno de desarrollo Windows).

2. Marco teórico

2.1 Visión por computadora y landmarks

- OpenCV: captura y preprocesamiento de frames (BGR→RGB, escalado).

- MediaPipe Hands: extracción de landmarks de mano (21 puntos por mano), robusto y eficiente en CPU.

- Normalización: convertir landmarks relativos a caja de mano para generar features invariables a escala/traslación.

2.2 Clasificación / Inferencia

- Uso de un clasificador (wrapper ClasificadorSenia → DetectorWrapper) que toma vectores derivadas de landmarks y devuelve letras (A–Z / Ñ) y anotaciones visuales.

2.3 Arquitectura concurrente

- Hilos: CaptureThread (producer), ProcessingThread (consumer / detector).

- Comunicación: Queue para frames y EventBus para notificar eventos (frame_captured, hand_detected).

2.4 Ingeniería del software aplicada

- Principios SOLID básicos: separación de responsabilidades (capture, processing, detector, UI, lógica de juego).

- Patrones de diseño: Observer (EventBus), Singleton (DBManager), Strategy/Template (posibles extensiones de procesamiento y GameBase).

3. Análisis y diseño

3.1 Requisitos funcionales (RF)

1. RF1 — Capturar frames de la webcam con tasa configurable (FPS).

2. RF2 — Procesar frames en tiempo real y detectar la letra señalada.

3. RF3 — Publicar eventos de detección (hand_detected) y frames (frame_captured).

4. RF4 — Mostrar preview de cámara en la GUI principal.

5. RF5 — Ejecutar juegos que consuman eventos de detección y actualicen su estado.

6. RF6 — Guardar puntuaciones por usuario/juego en la base de datos.

7. RF7 — Iniciar/detener captura desde GUI sin cerrar la aplicación.

8. RF8 — Proveer tests para verificación básica del detector y pipeline.

3.2 Requisitos no funcionales (RNF)

1. RNF1 — El sistema debe procesar al menos 10 FPS en hardware común (condicional).

2. RNF2 — Diseñado para tolerar pérdida ocasional de frames (cola con tamaño limitado).

3. RNF3 — La GUI debe permanecer responsive (no bloquear hilos de captura/procesamiento).

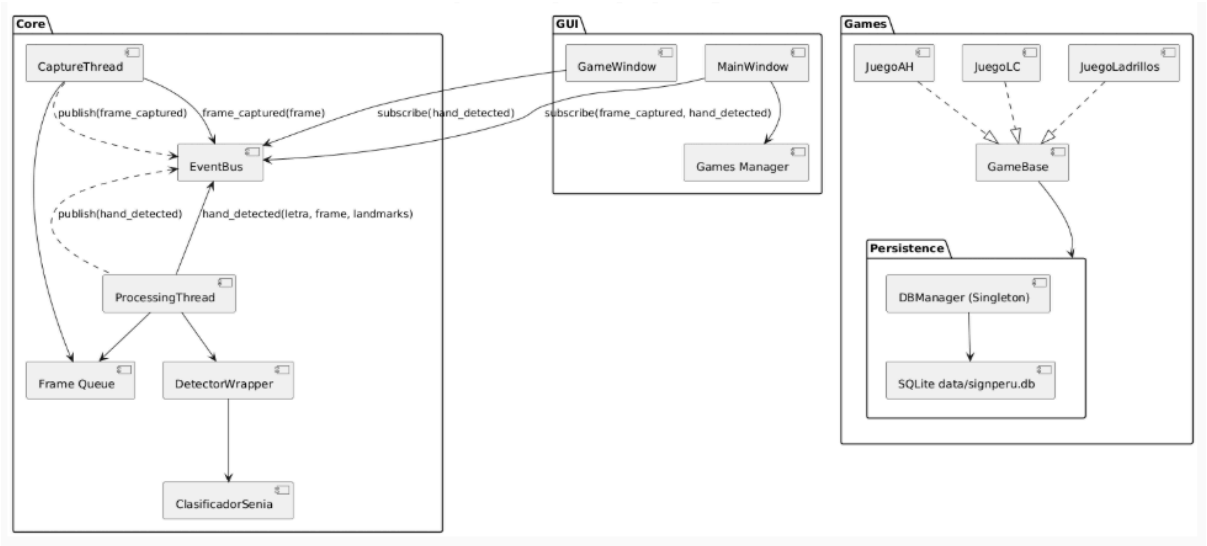
4. RNF4 — Código modular, documentado en español y con tests básicos.

5. RNF5 — Persistencia confiable (SQLite), con acceso concurrente seguro (patrón Singleton para conexión).

6. RNF6 — Portabilidad: Python 3.10+ en Windows/Linux.

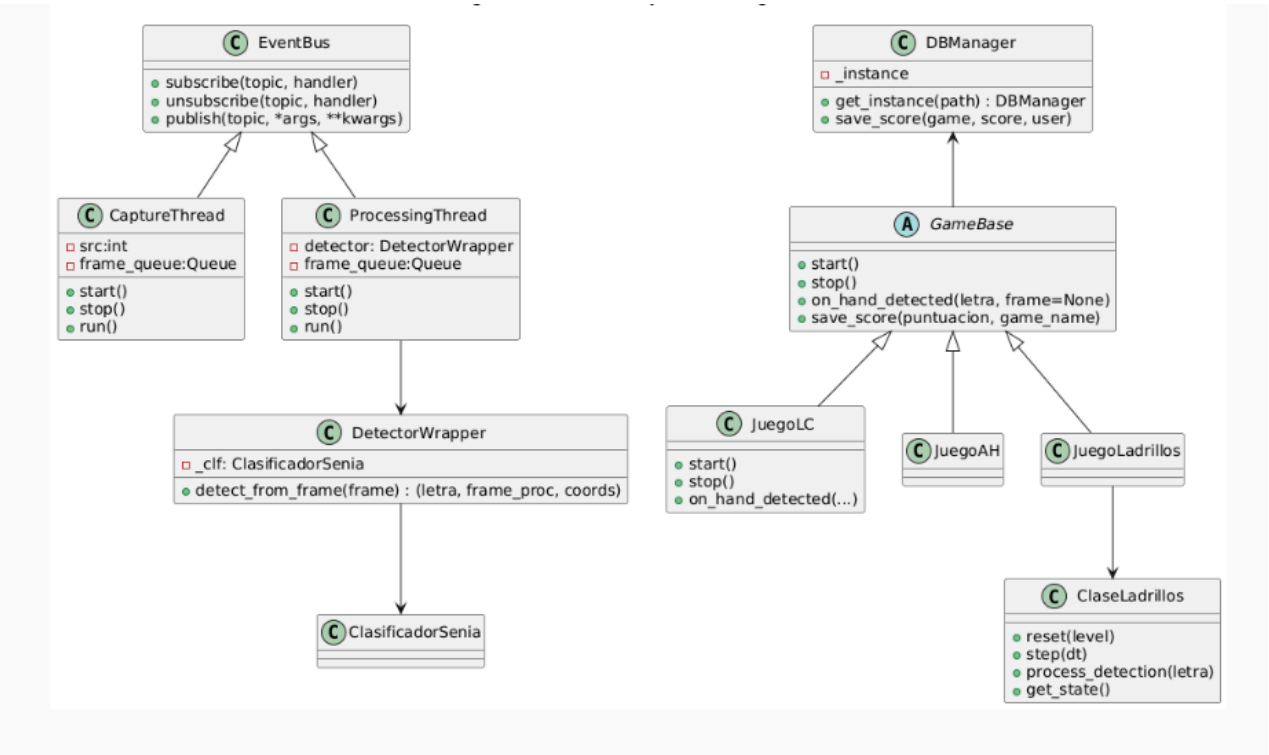
3.3 Arquitectura del sistema

El sistema aplica una arquitectura modular y basada en componentes ligeros (micro-componentes dentro de la app). El patrón central es pub/sub (Observer) mediante EventBus. Otras piezas: Producer/Consumer (Queue), Singleton (DBManager), Template Method para la base de juegos.
A continuación se entrega un diagrama de componentes (PlantUML) que describe módulos y relaciones:



3.4 Diagrama de clases

PlantUML simplificado con las clases clave. Puedes pegar este plantuml en cualquier editor compatible para visualizarlo.



3.5 Patrones de diseño utilizados

3.5.1 Observer / Pub-Sub

EventBus permite desacoplar productores (captura/procesamiento) y consumidores (UI/juegos).

3.5.2 Singleton

DBManager se instancia como singleton para controlar la conexión SQLite y evitar múltiples accesos simultáneos.

3.5.3 Template Method / Base Class

GameBase define la interfaz/flujo básico de un juego (start, stop, on_hand_detected, save_score). Las implementaciones concretas especializan la lógica.

3.5.4 Strategy (opcional)

ProcessingThread puede admitir distintas estrategias de procesamiento si se extiende para incorporar pre/post-procesamientos alternativos.

4. Implementación

4.1 Tecnologías utilizadas

- Lenguaje: Python 3.10+ (probado en 3.12).
- Librerías: OpenCV, MediaPipe, TensorFlow/TFLite (si aplica), NumPy, Pillow, pygame, customtkinter, imageio.
- Persistencia: SQLite (sqlite3) vía persistence/db_manager.py.
- Herramientas de desarrollo: Visual Studio Code, venv, pip.
- Testing: tests básicos en tests/test_detector.py.

4.2 Estructura del proyecto

```
ProyectoTO_senias/
├─ srlsp-game/
│  ├─ README.md
│  ├─ run.sh
│  ├─ requerimientos.txt
│  ├─ src/signperu/
│  │  ├─ app.py
│  │  ├─ config.py
│  │  ├─ init_db.py
│  │  ├─ data/signperu.db
│  │  ├─ clasificador/abecedario.py no cambia
│  │  ├─ core/detector.py
│  │  ├─ core/capture.py
│  │  ├─ core/processing.py
│  │  ├─ core/events.py
│  │  ├─ gui/main_window.py
│  │  ├─ gui/frames/login_frame.py
│  │  ├─ gui/frames/profile_frame.py
│  │  ├─ gui/frames/game_frame.py
│  │  ├─ gui/frames/settings_frame.py
│  │  ├─ games/game_base.py
│  │  ├─ games/clase_ah.py # lógica del juego ahorcado
│  │  ├─ games/clase_lc.py # lógica del juego letras caen
│  │  ├─ games/clase_ladrillos.py # lógica de Arkanoid
│  │  ├─ games/PalabraN_ah.py #palabra nueva
│  │  ├─ games/juego_AH.py
│  │  ├─ games/juego_LADRILLOS.py
│  │  ├─ games/juego_LC.py
│  │  ├─ games/PALABRAS #archivo de palabras
```

```
│ │ ├─ persistence/db_manager.py
│ │ ├─ persistence/models.sql
│ │ ├─ test/quick_test_debug.py # test de camara y dibujo frame
│ │ ├─ utils/audio.py
│ │ ├─ utils/logger.py
│ │ ├─ utils/helpers.py
│ │ └─ tests/test_detector.py
└─ venv/
```

4.3 Modelo de datos

Tablas principales (esquema simplificado):

- scores
    - id INTEGER PRIMARY KEY
    - user TEXT (opcional)
    - game_name TEXT
    - score INTEGER
    - date TIMESTAMP
- users (opcional)
    - id, username, email, created_at

Puedes ver persistence/models.sql para el SQL completo.

4.4 Persistencia de datos

DBManager implementa un patrón Singleton que expone métodos como save_score(game_name, score, user=None) y get_top_scores(game_name, limit=10). La base de datos es un archivo SQLite: src/signperu/data/signperu.db.

5. Pruebas y validación

5.1 Plan de pruebas

1. Pruebas unitarias:
    - tests/test_detector.py: pruebas sobre detector y normalización de landmarks.

Python

```python
# src/signperu/test/quick_test_debug.py
import time
import cv2
from signperu.core.detector import DetectorWrapper

def main(duration=20, cam_src=0):
    det = DetectorWrapper()
    cap = cv2.VideoCapture(cam_src, cv2.CAP_DSHOW if hasattr(cv2,
"CAP_DSHOW") else 0)
    if not cap.isOpened():
        print("ERROR: no se pudo abrir la cámara.")
        return
    t0 = time.time()
    print("Prueba rápida con debug. Mira la cámara...")
    while time.time() - t0 < duration:
        ret, frame = cap.read()
        if not ret:
            continue
        letra, annotated, coords = det.detect_from_frame(frame)
        if letra:
            print("Señal detectada:", letra)
        if coords:
            print("Landmarks:", coords[:5], "... total:", len(coords))
        # mostramos feed anotado por si hay landmarks
        annotated_rgb = cv2.cvtColor(annotated, cv2.COLOR_BGR2RGB)
        cv2.imshow("Debug annotated (q para salir)", annotated_rgb)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
    print("Fin prueba.")

if __name__ == "__main__":
    main()
```

2. Pruebas de integración:
    - test/quick_test_debug.py: capture → processing → publish frame; verificar que preview y detecciones llegan a la GUI.
3. Pruebas de aceptación (manual):
    - Ejecutar MainWindow, iniciar cámara, probar cada juego (AH / LC / LADRILLOS) y validar comportamiento.

4. Pruebas de rendimiento:
    - Medición de FPS efectivo en la pipeline (Capture → Processing → UI) con diferentes resoluciones (720p, 480p).
5. Pruebas de stress:
    - Abrir/ cerrar cámara repetidas veces, ejecutar juegos en secuencia para validar liberación de recursos.
6. Pruebas de usabilidad:
    - Evaluar claridad del feedback (preview, caja de letra detectada, HUD del juego).

6. Resultados

6.1 Funcionalidades implementadas

- Captura de webcam y publicación de frames en EventBus.
- Pipeline de procesamiento que publica hand_detected con letra y frame anotado.
- GUI principal con preview, controles de cámara y lanzadores de juegos.
- Tres juegos integrados que reciben eventos de detección y actualizan su estado.
- Persistencia de puntuaciones en SQLite.
- Tests y utilidades de debug.

6.2 Métricas de código

Nota: para métricas más exactas use herramientas como cloc o sloccount. A continuación se dan estimaciones conservadoras basadas en la estructura actual:

Métrica	Estimación aproximada
Líneas de código (total .py)	6,000 – 9,000 LOC
Archivos fuente (Python .py)	40 – 70 archivos
Clases implementadas	~25 – 40 clases
Métodos públicos	~120 – 220
Patrones de diseño usados	Observer, Singleton, Template Method, Strategy (parcial)

Archivos de tests (.py)	3 – 8

7. Conclusiones

- La refactorización hacia una arquitectura modular y basada en eventos facilita la mantenibilidad y extensión del proyecto (añadir nuevos juegos o mejorar el detector).
- Separar la lógica de juego (clase_) de la UI (juego_) permite reutilizar código, testear la lógica sin UI y facilitar arreglos de bugs.
- El enfoque multihilo con Queue + EventBus mantiene la UI responsive y permite un pipeline de procesamiento robusto.
- Persistencia en SQLite es suficiente para registrar puntuaciones y estadísticas iniciales.

8. Recomendaciones y trabajo futuro

1. Medir y optimizar rendimiento: pruebas con diferentes resoluciones y optimizaciones en el preprocesamiento (ROI, reducción de resolución).
2. Recolectar dataset: ampliar ejemplos de señas LSP para mejorar el clasificador.
3. Mejorar UX: añadir calibración inicial de la cámara, tutorial interactivo y perfiles de usuario.
4. Logging y telemetría: agregar métricas de rendimiento y trazas más detalladas (módulo utils/logger.py).
5. Portabilidad: preparar build para Linux y considerar empaquetado (PyInstaller) o portar a web (WebRTC + WASM).
6. A/B tests y evaluación con usuarios reales: validar impacto pedagógico.

9. Referencias

- OpenCV — https://opencv.org
- MediaPipe — https://mediapipe.dev
- TensorFlow Lite — https://www.tensorflow.org/lite
- PlantUML — https://plantuml.com

- Documentación Python: threading, queue, sqlite3, tkinter, pygame