



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 1

INFORME DE LABORATORIO (formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	Construcción de Software				
TITULO DE LA PRÁCTICA:	Informe Final				
NÚMERO DE PRÁCTICA:	8	AÑO LECTIVO:	2025B	NRO. SEMESTRE:	VI (sexto)
FECHA DE PRESENTACIÓN	16/12/2025	HORA DE PRESENTACIÓN	15:30		
INTEGRANTE (s) -Betanzos Rosas Taylor Anthony -Garcia Valdivia Ronald Pablo -Zapana Romero Pedro Luis Christian				NOTA (0-20)	
DOCENTE(s): <i>Mg. Diego Alonso Iquira Becerra</i>					

RESULTADOS Y PRUEBAS

I. Enlaces de trabajo

Enlace al Repositorio Final:

https://drive.google.com/drive/folders/1gaBK7_5jrkBlrdVgtvqkusoKwfx91FC6

Link del repositorio Github:

<https://github.com/TAYTOS/Music-therapy>

Link del cronograma:

<https://www.canva.com/design/DAG0fDDCsE0/29lcpZ8O6aRdz1swFk9VYQ/edit?locale=es-CO&ui=eyJBjp7IkUiOnsiQSI6dHJ1ZX19fQ>

Link de Jira:

<https://unsa-team-esiyms20.atlassian.net/jira/software/projects/SCRUM/boards/1/backlog?atlOrigin=eyJpljoiOWYwMTg1NjdkNTAzNDA1MzlhNTIiYTlyZGFiMjViYjUiLCJwljoiaij9>

II. Desarrollo de la aplicación

En este apartado veremos una recopilación de los trabajos realizados anteriormente como base del proyecto:

Introducción.



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 2

En el presente proyecto queremos abarcar la problemática de ayuda a terapia de manera didáctica con un juego de movimiento rítmico utilizando las extremidades superiores utilizando música. Comparamos artículos que trataron temas similares, así mismo se estableció un cronograma para la creación del juego, haciendo por plazos pasos del código.

Propósito del proyecto.

El presente sistema tiene como propósito ayudar a tener una terapia más didáctica y disfrutable para los distintos pacientes que sufren de un daño en los miembros superiores del cuerpo. Para algunas personas las terapias pueden considerarse tediosas, lo cual les quita las ganas de seguir participando o no los motiva a continuar con la terapia en sus hogares. Sin embargo, por medio del ritmo de las diferentes canciones se puede dejar de visualizar como una terapia en sí, sino como una actividad amena qué se puede realizar en los momentos de ocio.

Objetivos.

El proyecto tiene como objetivo principal brindar una experiencia cómoda para el usuario durante sus sesiones de terapia por medio de la precisión en la detección de la mano y sus movimientos en tiempo real, la integración fluida entre los movimientos que se realizan y la jugabilidad musical. Asimismo se busca conseguir pruebas positivas de usabilidad, interfaces y habilidad intuitiva.

Herramientas

Para llevar a cabo el desarrollo del presente proyecto, se seleccionaron herramientas tecnológicas que permitieran una implementación eficiente y funcional del sistema, centrado en la interacción rítmica a través del movimiento de las manos. Estas herramientas fueron escogidas con base en su compatibilidad con los objetivos del proyecto, su facilidad de integración y su eficacia comprobada en el desarrollo de aplicaciones interactivas.

- *Python: Se emplea como lenguaje principal de programación, dada su versatilidad y la amplia disponibilidad de librerías orientadas al desarrollo multimedia y la visión por computadora.*
- *OpenCV: Utilizada para la captura y procesamiento de imágenes en tiempo real. Esta herramienta permitirá detectar los movimientos de la mano mediante la cámara web, elemento clave para la interacción del usuario con el sistema.*
- *CVZone: Librería basada en OpenCV, que facilita la detección de gestos y la creación de interfaces visuales dinámicas, simplificando así el proceso de desarrollo.*
- *Pygame: Servirá como plataforma para la creación del entorno visual del juego, permitiendo la integración de elementos gráficos, musicales y de interacción con el usuario.*
- *GitHub: Se empleará como sistema de control de versiones, permitiendo una gestión colaborativa del código y un seguimiento ordenado de los avances en el desarrollo.*
- *Jira: Será utilizada como herramienta de gestión de tareas y seguimiento del proyecto. A través de su metodología basada en tableros Kanban y sprints, Jira permitirá organizar las actividades del equipo, priorizar funcionalidades y asegurar el cumplimiento de los plazos establecidos en el cronograma.*



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 3

Estas herramientas permitirán no solo cumplir con los requisitos técnicos del sistema, sino también garantizar una experiencia fluida y accesible para los usuarios durante sus sesiones de terapia.

Catálogo de Requisitos

Requisitos Funcionales (Agrupar los requisitos según módulos)

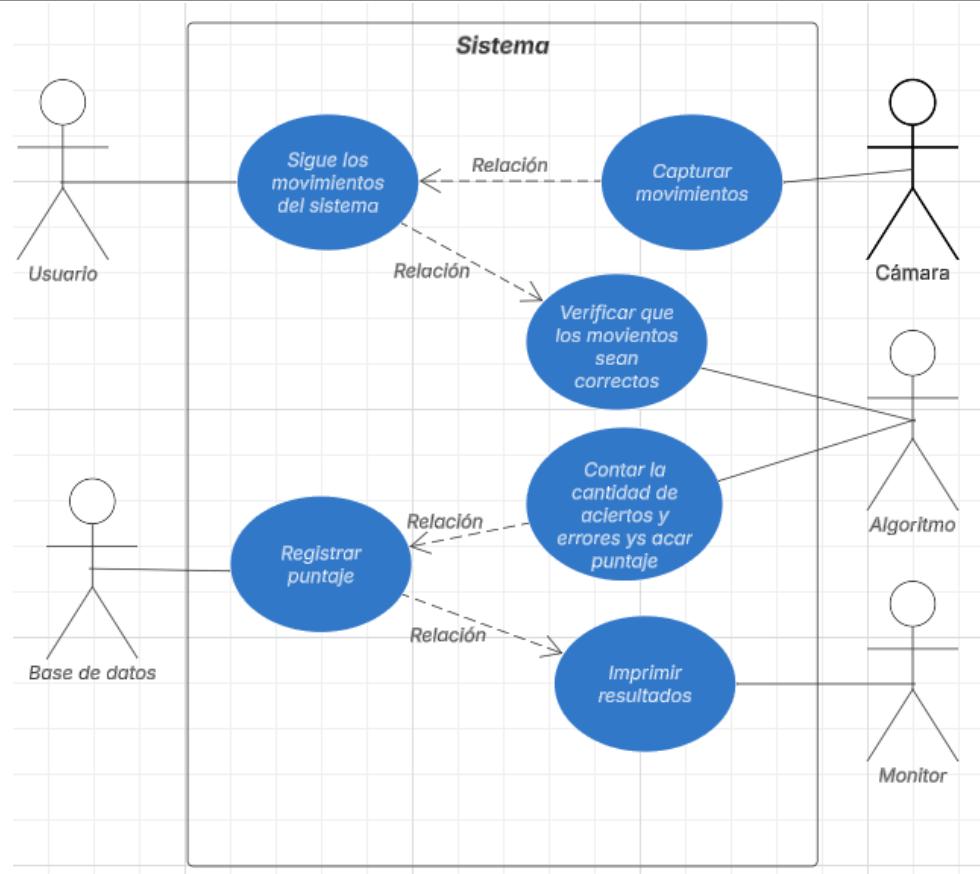
- *El sistema debe capturar los movimientos de la mano en tiempo real mediante la cámara web.*
- *El sistema debe reconocer gestos básicos de la mano (puño cerrado) y sus direcciones (arriba, abajo, izquierda, derecha).*
- *El sistema debe mostrar en pantalla flechas o patrones visuales.*
- *El sistema debe reproducir ritmo musical de fondo*
- *El sistema debe calcular puntajes basados en la precisión y sincronización de los movimientos del usuario.*
- *El sistema debe registrar el puntaje final de cada sesión y mostrarlo en una pantalla de resultados.*

Requisitos No Funcionales

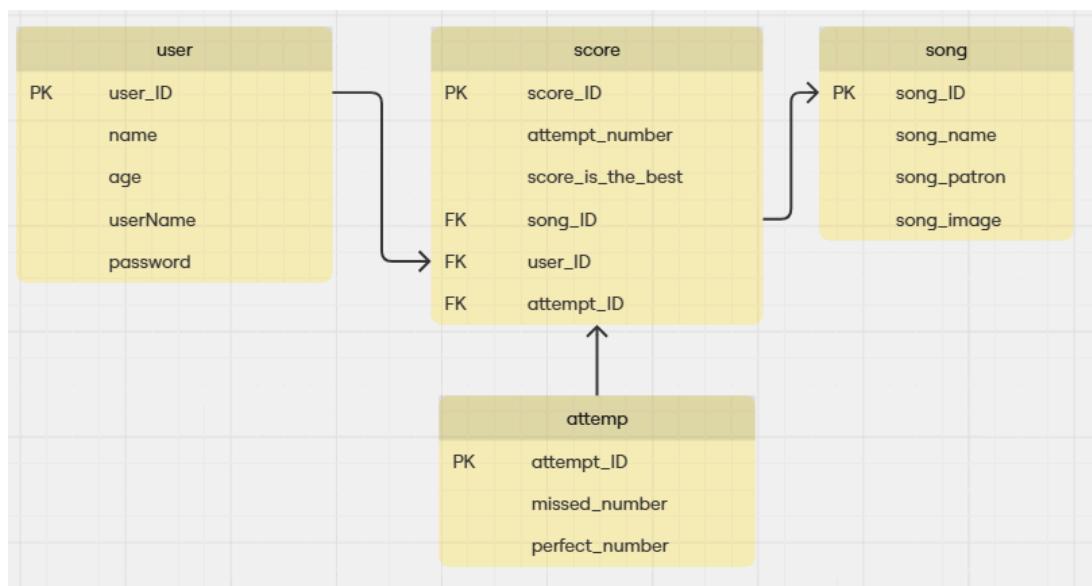
- *La detección de la mano debe tener una latencia baja.*
- *La interfaz debe ser intuitiva y accesible.*
- *El sistema debe permitir ser ejecutado en un equipo estándar sin requerir hardware especializado.*
- *El código debe ser modular para facilitar futuras mejoras (nuevos gestos, canciones, niveles).*
- *El sistema debe funcionar en tiempo real sin interrupciones perceptibles durante la ejecución del juego.*
- *El diseño visual debe ser atractivo y motivador, incorporando elementos gráficos y colores que fomenten la atención del usuario.*
- *El consumo de recursos (CPU y memoria RAM) debe ser eficiente, garantizando que otras aplicaciones puedan ejecutarse en paralelo sin afectar el rendimiento.*

Diagramas

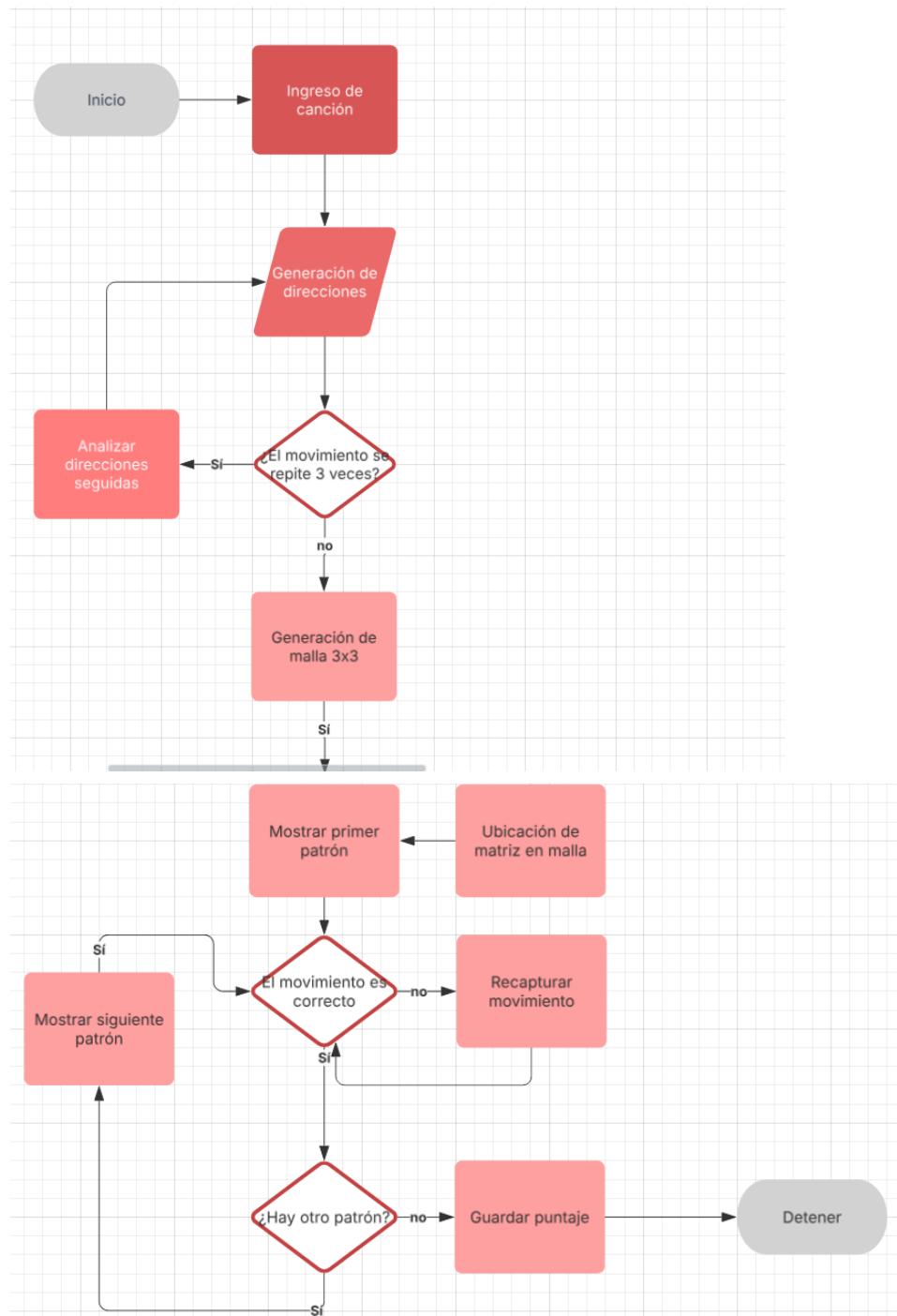
- A. *El sistema debe registrar el puntaje final de cada sesión y mostrarlo en una pantalla de resultados.*



B. El sistema debe registrar el puntaje final de cada sesión y mostrarlo en una pantalla de resultados.



C. Diagrama de flujo de la generación de patrones en pantalla



Diseño de la arquitectura de sistema

Estilo arquitectónico elegido: MVC (Modelo-Vista-Controlador)



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 6

- *Se adopta MVC porque separa la lógica del negocio, la presentación y los datos, facilitando mantenimiento y ampliaciones.*
- *Modelo: maneja los puntajes, sesiones y configuraciones del usuario.*
- *Vista: interfaz y elementos gráficos (menús, animaciones) creados en Pygame.*
- *Controlador: usa OpenCV y Mediapipe para interpretar los movimientos y coordinar la jugabilidad.*

Justificación del diseño

- *Aísla la lógica de detección de gestos de la interfaz visual.*
- *Permite agregar nuevos gestos o niveles sin reescribir el resto del sistema.*
- *Facilita pruebas y depuración por módulos.*

III. Funciones

En esta sección hablaremos de las principales clases que intervienen en el proyecto, así también se mostrará un esquema general de las clases que intervienen en el trabajo que en este caso son 5 clases:

A. vista.py — Vista (Interfaz gráfica)

Responsabilidad:

- *Se encarga de mostrar toda la información visual del sistema sobre el video en tiempo real.*

Funcionamiento:

- *Dibuja la interfaz del juego/terapia: puntaje, tiempo, combo, estados, instrucciones y un indicador visual del beat musical. Usa OpenCV para gráficos y una clase auxiliar (TextoUnicode) para soportar caracteres especiales.*

Métodos importantes:

- *dibujar_interfaz(): renderiza el panel superior e inferior con datos del sistema.*
- *dibujar_indicador_beat(): muestra el pulso rítmico sincronizado con la música.*
- *mostrar(): presenta el frame en pantalla.*

Python

```
import cv2
from utils.texto_unicode import TextoUnicode

class Vista:
    def __init__(self):
        self.texto_renderer = TextoUnicode()
        self.colores = {
            "fondo_panel": (20, 20, 40),
            "texto_principal": (255, 255, 255),
            "texto_score": (0, 255, 100),
            "texto_tiempo": (100, 200, 255),
```

```
        "borde": (100, 100, 200),
        "combo": (0, 255, 255),
    }

def dibujar_interfaz(
    self,
    frame,
    score,
    tiempo_restante,
    estado,
    num_puntos,
    bpm,
    musica_activa,
    beats_restantes,
    combo,
    max_combo,
    precision Ritmo,
):
    """Dibuja la interfaz del juego sobre el frame"""
    h, w, _ = frame.shape

    # Crear overlay semi-transparente para el panel superior
    overlay = frame.copy()

    # Panel superior con información (más grande para más datos)
    cv2.rectangle(overlay, (0, 0), (w, 120), (30, 30, 60), -1)
    frame = cv2.addWeighted(frame, 0.6, overlay, 0.4, 0)

    # Borde decorativo
    cv2.rectangle(frame, (5, 5), (w - 5, 115), self.colores["borde"], 3)

    # FILA 1: Puntaje y Combo
    cv2.putText(
        frame,
        f"Puntaje: {score}",
        (20, 40),
        cv2.FONT_HERSHEY_SIMPLEX,
        1.0,
        self.colores["texto_score"],
        3,
    )

    # Combo con efecto visual
    if combo > 0:
        color_combo = self.colores["combo"]
        tamano_combo = 1.0 + (min(combo, 20) * 0.02) # Crece con el combo
        cv2.putText(
            frame,
```



```
f"COMBO x{combo}!",
(w - 280, 40),
cv2.FONT_HERSHEY_SIMPLEX,
tamano_combo,
color_combo,
3,
)

# FILA 2: Tiempo/Beats restantes
if tiempo_restante > 5:
    color_tiempo = (0, 255, 0)
elif tiempo_restante > 2:
    color_tiempo = (0, 255, 255)
else:
    color_tiempo = (0, 0, 255)

cv2.putText(
    frame,
    f"Tiempo: {tiempo_restante}s",
(20, 75),
cv2.FONT_HERSHEY_SIMPLEX,
0.8,
color_tiempo,
2,
)

# Precisión de ritmo
color_precision = (
    (0, 255, 0)
    if precision Ritmo >= 80
    else (0, 255, 255) if precision Ritmo >= 60 else (0, 0, 255)
)

# FILA 3: Info musical y max combo
color_musica = (0, 255, 0) if musica_activa else (100, 100, 100)
cv2.putText(
    frame,
    f"Max Combo: {max_combo}",
(20, 105),
cv2.FONT_HERSHEY_SIMPLEX,
0.7,
color_musica,
2,
)

# Instrucciones según el estado
instrucciones = {
```

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 9

```
"esperando": "Coloca el PUÑO en el circulo VERDE para comenzar a bailar",
"trazando": "Mueve tu mano AL RITMO de los beats - NO pierdas 3 beats",
"completado": ";PERFECTO! Bailaste con ritmo",
"fallido": "Tiempo agotado - Intenta otra vez",
"fuera Ritmo": ";Perdiste el ritmo! Nuevo patron",
}

texto_estado = instrucciones.get(estado, "")

# Panel inferior con instrucciones
overlay2 = frame.copy()
cv2.rectangle(overlay2, (0, h - 70), (w, h), (30, 30, 60), -1)
frame = cv2.addWeighted(frame, 0.6, overlay2, 0.4, 0)

# Color del texto según el estado
if estado == "completado":
    color_texto = (0, 255, 0)
elif estado == "fuera Ritmo":
    color_texto = (0, 100, 255)
elif estado == "fallido":
    color_texto = (0, 0, 255)
elif estado == "trazando":
    color_texto = (0, 255, 255)
else:
    color_texto = self.colores["texto_principal"]

# Usar TextoUnicode para dibujar con soporte de ñ
self.texto_renderer.put_text(
    frame, texto_estado, (20, h - 50), tamano=22, color=color_texto, grosor=2
)

return frame

def dibujar_indicador_beat(
    self, frame, progreso_beat, hay_beat, en_sincronizacion, combo
):
    # Dibuja un indicador visual del beat con feedback de sincronización #
    h, w, _ = frame.shape

    # Posición del indicador (esquina superior derecha, más grande)
    centro_x = w - 80
    centro_y = 160
    radio_base = 35

    # Efecto de pulso cuando hay beat
    if hay_beat or progreso_beat < 0.2:
        factor_pulso = 1.8 - (progreso_beat * 3)
        radio = int(radio_base * factor_pulso)
```

```
intensidad = 255
else:
    radio = radio_base
    intensidad = int(100 + (155 * (1 - progreso_beat)))

# Color según sincronización
if en_sincronizacion:
    color = (0, 255, 0) # Verde: ¡Perfecto!
    grosor_borde = 4
elif progreso_beat < 0.3 or progreso_beat > 0.7:
    color = (0, 255, 255) # Cian: Cerca del beat
    grosor_borde = 3
else:
    color = (0, intensidad, 0) # Verde oscuro: Espera el beat
    grosor_borde = 2

# Dibujar círculo del beat
cv2.circle(frame, (centro_x, centro_y), radio, color, -1)
cv2.circle(
    frame, (centro_x, centro_y), radio + 2, (255, 255, 255), grosor_borde
)

# Barra de progreso circular (cuenta regresiva al próximo beat)
angulo_inicio = -90
angulo_fin = int(-90 + (360 * progreso_beat))
cv2.ellipse(
    frame,
    (centro_x, centro_y),
    (radio_base + 12, radio_base + 12),
    0,
    angulo_inicio,
    angulo_fin,
    (100, 200, 255),
    4,
)
)

# Texto "BEAT!" cuando hay beat
if hay_beat or progreso_beat < 0.15:
    cv2.putText(
        frame,
        "BEAT!",
        (centro_x - 35, centro_y + 8),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        (255, 255, 255),
        2,
    )
```



```
# Indicador de combo debajo del beat
if combo > 0:
    cv2.putText(
        frame,
        f"x{combo}",
        (centro_x - 20, centro_y + 50),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.9,
        (0, 255, 255),
        2,
    )

return frame

def mostrar(self, frame):
    """Muestra el frame en pantalla"""
    cv2.namedWindow("Terapia de Rehabilitacion", cv2.WINDOW_NORMAL)
    cv2.imshow("Terapia de Rehabilitacion", frame)
```

B. *modelo.py — Modelo (Lógica de detección de mano)*

Responsabilidad:

- Gestiona la detección de la mano y del gesto de puño.

Funcionamiento:

- Utiliza MediaPipe para detectar landmarks de la mano y determina si los dedos están cerrados, identificando un puño como interacción principal del usuario.

Métodos importantes:

- HandDetector
- detectar_mano(): detecta y dibuja la mano.
- detectar_puño(): evalúa si la mano está cerrada (puño).

Python

```
import mediapipe as mp
import cv2

class HandDetector:
    def __init__(self, max_hands=1, detection_confidence=0.7):
        self.hands_module = mp.solutions.hands
        self.hands = self.hands_module.Hands(
            max_num_hands=max_hands, min_detection_confidence=detection_confidence
        )
        self.drawer = mp.solutions.drawing_utils
```



```
def detectar_mano(self, frame):
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = self.hands.process(rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            self.drawer.draw_landmarks(
                frame, hand_landmarks, self.hands_module.HAND_CONNECTIONS
            )
    return hand_landmarks
return None

def detectar_punto(self, landmarks, frame):
    h, w, _ = frame.shape
    puntos = [(int(p.x * w), int(p.y * h)) for p in landmarks.landmark]
    dedos = [8, 12, 16, 20]
    abierto = 0

    for d in dedos:
        if puntos[d][1] < puntos[d - 2][1]:
            abierto += 1
    if abierto == 0:
        return True
    return False
```

C. controlador.py — Controlador (Lógica principal del sistema)

Responsabilidad:

- Coordina todo el sistema y controla el flujo del programa.

Funcionamiento:

- Administra estados (esperando, trazando, completado, fallido), evalúa el ritmo del usuario, calcula puntajes, controla combos y sincroniza interacción, patrones y música dentro del bucle principal.

Clases / funciones importantes:

- main()

Loop principal de captura, procesamiento, evaluación de ritmo y renderizado.

Controla eventos de teclado y transiciones de estado.

Python

```
import cv2
import numpy as np
import time
from utils.camara import Camara
```



```
from utils.patrones import GeneradorPatrones
from utils.musica import GestorMusica
from vista import Vista

def main():
    cap = cv2.VideoCapture("http://192.168.1.37:4747/video")
    if not cap.isOpened():
        print("No se pudo acceder a la cámara IP.")
        return

    cam = Camara()
    vista = Vista()
    generador = GeneradorPatrones()
    musica = GestorMusica()

    score = 0
    estado = "esperando"

    rastro_usuario = []
    posicion_anterior = None

    combo ritmo = 0
    max_combo = 0
    fallos ritmo = 0
    MAX_FALLOS_RITMO = 3

    puntos_minimos = 30
    mensaje_temporal = ""
    tiempo_mensaje = 0

    musica.reiniciar_compas()

    while True:
        frame, is_fist, pos = cam.obtener_frame(cap)
        if frame is None:
            continue

        hay_beat = musica.actualizar()
        progreso_beat = musica.obtener_progreso_beat()
        en_ventana = musica.esta_en_ventana_sincronizacion()

        hay_movimiento = (
            is_fist
            and pos[0] is not None
            and posicion_anterior is not None
            and np.linalg.norm(np.array(pos) - np.array(posicion_anterior)) > 8
        )
```

```
en Ritmo, fallo_instantaneo = musica.verificar_sincronizacion(hay_movimiento)

# ===== RITMO (UN SOLO LUGAR) =====
perdio Ritmo = False

if en Ritmo:
    combo Ritmo += 1
    max_combo = max(max_combo, combo Ritmo)
    fallos Ritmo = max(0, fallos Ritmo - 1)
elif fallo_instantaneo:
    fallos Ritmo += 1
    combo Ritmo = max(0, combo Ritmo - 1)
    if fallos Ritmo >= MAX_FALLOS_RITMO:
        perdio Ritmo = True

tiempo_restante = int(musica.obtener_tiempo_restante_patron())
beats_restantes = musica.obtener_beats_restantes()

# ===== ESTADOS =====

if estado == "esperando":
    rastro_usuario.clear()
    combo Ritmo = 0
    fallos Ritmo = 0

    if is_fist and pos[0] is not None:
        if generador.calcular_distancia_a_inicio(pos) < 50:
            estado = "trazando"
            rastro_usuario.append(pos)
            musica.reiniciar_compas()
            musica.reiniciar_estadisticas()

elif estado == "trazando":
    if is_fist and pos[0] is not None:
        if not rastro_usuario or np.linalg.norm(
            np.array(pos) - np.array(rastro_usuario[-1]))
            ) > 5:
            rastro_usuario.append(pos)

    if perdio Ritmo:
        estado = "fuera Ritmo"
        mensaje_temporal = "¡Perdiste el ritmo!"
        tiempo_mensaje = time.time()
        musica.reproducir_error()

if len(rastro_usuario) >= 10:
    cobertura, llegado_final, _ = (
```

```
        generador.calcular_progreso_detallado(rastro_usuario)
    )

    if (
        cobertura >= 0.75
        and llegado_final
        and len(rastro_usuario) >= puntos_minimos
    ):
        estado = "completado"

        puntos = int(cobertura * 10)
        puntos += min(combo ritmo * 2, 20)
        puntos += int(musica.obtener_precision_ritmo() / 10)

        score += puntos
        mensaje_temporal = f";PERFECTO! +{puntos} pts"
        tiempo_mensaje = time.time()
        musica.reproducir_exito()

    if musica.es_momento_de_nuevo_patron() and estado == "trazando":
        estado = "fallido"
        mensaje_temporal = "Tiempo agotado"
        tiempo_mensaje = time.time()

    elif estado in ("completado", "fallido", "fuera Ritmo"):
        generador.gerar_patron()
        musica.reiniciar_compas()
        rastro_usuario.clear()
        fallos_ritmo = 0
        estado = "esperando"

# ===== DIBUJO =====

frame = generador.dibujar_patron_en_frame(
    frame, rastro_usuario if estado == "trazando" else []
)

# Mano (círculo restaurado)
if pos[0] is not None:
    if is_fist:
        color = (0, 255, 255) if en_ventana else (0, 150, 255)
        radio = 25 if hay_beat else 20
        cv2.circle(frame, pos, radio, color, -1)
        cv2.circle(frame, pos, radio + 3, (255, 255, 255), 3)
    else:
        cv2.circle(frame, pos, 12, (255, 0, 0), -1)
        cv2.circle(frame, pos, 16, (255, 255, 255), 2)
```

```
if musica.musica_activa:
    frame = vista.dibujar_indicador_beat(
        frame, progreso_beat, hay_beat, en Ritmo, combo Ritmo
    )

frame = vista.dibujar_interfaz(
    frame,
    score,
    tiempo_restante,
    estado,
    len(rastro_usuario),
    musica.bpm,
    musica.musica_activa,
    beats_restantes,
    combo Ritmo,
    max_combo,
    musica.obtener_precision_ritmo(),
)

vista.mostrar(frame)

if pos[0] is not None:
    posicion_anterior = pos

key = cv2.waitKey(1) & 0xFF
if key == ord(" "):
    break
elif key in (ord("m"), ord("M")):
    musica.desactivar_musica() if musica.musica_activa else
musica.activar_musica()
elif key in (ord("+"), ord("=")):
    musica.cambiar_bpm(musica.bpm + 10)
elif key in (ord("-"), ord("_")):
    musica.cambiar_bpm(musica.bpm - 10)

cap.release()
cv2.destroyAllWindows()

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

D. *patrones.py — Modelo de patrones de movimiento*

Responsabilidad:

- *Genera, evalúa y dibuja los patrones de movimiento que el usuario debe seguir.*

Funcionamiento:

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 17

- Crea distintos patrones geométricos (círculo, zigzag, espiral, etc.), verifica qué tanto el usuario los ha seguido y calcula métricas como cobertura y llegada al final.

Métodos importantes:

- `generar_patron()`: crea patrones secuenciales.
- `calcular_progreso_detallado()`: evalúa precisión del trazo.
- `dibujar_patron_en_frame()`: renderiza el patrón y el progreso.

Python

```
import cv2
import numpy as np
import math

class GeneradorPatrones:
    def __init__(self):
        self.patrones = []
        self.tipo_actual = ""
        self.puntos_clave = []
        self.generar_patron()

    def generar_patron(self):
        if not hasattr(self, "indice_patron"):
            self.indice_patron = 0

        orden_patrones = ["zigzag", "curva", "triangulo", "circulo", "espiral"]

        self.tipo_actual = orden_patrones[self.indice_patron]
        self.indice_patron = (self.indice_patron + 1) % len(orden_patrones)

        puntos = []
        puntos_clave = []

        if self.tipo_actual == "circulo":
            cx, cy = 320, 240
            radio = 120
            for i in range(0, 360, 8):
                ang = math.radians(i)
                x = int(cx + radio * math.cos(ang))
                y = int(cy + radio * math.sin(ang))
                puntos.append((x, y))
                if i % 90 == 0:
                    puntos_clave.append((x, y))

        elif self.tipo_actual == "triangulo":
            cx, cy = 320, 280
            size = 180
            vertices = [
```

```
(cx - size // 2, cy + size // 3),
(cx + size // 2, cy + size // 3),
(cx, cy - 2 * size // 3),
(cx - size // 2, cy + size // 3),
]

for i in range(len(vertices) - 1):
    x1, y1 = vertices[i]
    x2, y2 = vertices[i + 1]
    steps = 20
    for j in range(steps):
        t = j / steps
        x = int(x1 + (x2 - x1) * t)
        y = int(y1 + (y2 - y1) * t)
        puntos.append((x, y))

puntos_clave = vertices[:-1]

elif self.tipo_actual == "curva":
    for t in np.linspace(0, 2 * np.pi, 80):
        x = int(200 + t * 60)
        y = int(240 + 100 * math.sin(2 * t))
        puntos.append((x, y))
        if abs(math.sin(2 * t)) > 0.95:
            puntos_clave.append((x, y))

elif self.tipo_actual == "zigzag":
    x_inicio, y_inicio = 150, 240
    amplitud = 80
    for i in range(8):
        x = x_inicio + i * 60
        y = y_inicio + (amplitud if i % 2 == 0 else -amplitud)
        if i > 0:
            x_prev = x_inicio + (i - 1) * 60
            y_prev = y_inicio + (amplitud if (i - 1) % 2 == 0 else -amplitud)
            steps = 10
            for j in range(steps):
                t = j / steps
                px = int(x_prev + (x - x_prev) * t)
                py = int(y_prev + (y - y_prev) * t)
                puntos.append((px, py))
        puntos_clave.append((x, y))

elif self.tipo_actual == "espiral":
    cx, cy = 320, 240
    for i in range(0, 360 * 4, 10):
        ang = math.radians(i)
        radio = i / 10
```



```
x = int(cx + radio * math.cos(ang))
y = int(cy + radio * math.sin(ang))
puntos.append((x, y))
if i % 360 == 0:
    puntos_clave.append((x, y))

self.patrones = puntos
self.puntos_clave = (
    puntos_clave
    if puntos_clave
    else [puntos[0], puntos[len(puntos) // 2], puntos[-1]]
)

def calcular_distancia_a_inicio(self, pos):
    if not self.patrones or pos[0] is None:
        return float("inf")

    inicio = self.patrones[0]
    dx = pos[0] - inicio[0]
    dy = pos[1] - inicio[1]
    return math.sqrt(dx * dx + dy * dy)

def calcular_progreso_detallado(self, rastro_usuario):
    if not rastro_usuario or not self.patrones:
        return 0.0, False, 0.0

    umbral_cercania = 40

    puntos_patron_cubiertos = set()
    for idx, punto_patron in enumerate(self.patrones):
        for punto_usuario in rastro_usuario:
            dx = punto_usuario[0] - punto_patron[0]
            dy = punto_usuario[1] - punto_patron[1]
            if math.sqrt(dx * dx + dy * dy) < umbral_cercania:
                puntos_patron_cubiertos.add(idx)
                break

    cobertura = len(puntos_patron_cubiertos) / len(self.patrones)

    punto_final = self.patrones[-1]
    distancia_al_final = float("inf")
    for punto_usuario in rastro_usuario[-10:]:
        dx = punto_usuario[0] - punto_final[0]
        dy = punto_usuario[1] - punto_final[1]
        dist = math.sqrt(dx * dx + dy * dy)
        if dist < distancia_al_final:
            distancia_al_final = dist
```

```
llegado_al_final = distancia_al_final < 60

max_indice_alcanzado = 0
for idx, punto_patron in enumerate(self.patrones):
    for punto_usuario in rastro_usuario:
        dx = punto_usuario[0] - punto_patron[0]
        dy = punto_usuario[1] - punto_patron[1]
        if math.sqrt(dx * dx + dy * dy) < umbral_cercania:
            max_indice_alcanzado = max(max_indice_alcanzado, idx)

porcentaje_patron = (max_indice_alcanzado / len(self.patrones)) * 100

return cobertura, llegado_al_final, porcentaje_patron

def dibujar_patron_en_frame(self, frame, rastro_usuario=[]):
    if not self.patrones:
        return frame

    umbral = 40
    puntos_cubiertos = set()

    for idx, punto_patron in enumerate(self.patrones):
        for punto_usuario in rastro_usuario:
            dx = punto_usuario[0] - punto_patron[0]
            dy = punto_usuario[1] - punto_patron[1]
            if math.sqrt(dx * dx + dy * dy) < umbral:
                puntos_cubiertos.add(idx)
                break

    for i in range(len(self.patrones) - 1):
        color = (0, 255, 0) if i in puntos_cubiertos else (255, 200, 0)
        grosor = 3 if i in puntos_cubiertos else 4
        cv2.line(frame, self.patrones[i], self.patrones[i + 1], color, grosor)

    for i, punto in enumerate(self.patrones):
        if i % 8 == 0:
            color = (0, 255, 0) if i in puntos_cubiertos else (255, 150, 0)
            cv2.circle(frame, punto, 6, color, -1)
            cv2.circle(frame, punto, 8, (255, 255, 255), 1)

    for punto_clave in self.puntos_clave:
        if any(
            math.sqrt((p[0] - punto_clave[0]) ** 2 + (p[1] - punto_clave[1]) ** 2)
            < umbral
            for p in rastro_usuario
        ):
            color = (0, 255, 0)
        else:
```



```
        color = (255, 100, 0)
        cv2.circle(frame, punto_clave, 12, color, 2)
        cv2.circle(frame, punto_clave, 15, (255, 255, 255), 1)

    if self.patrones:
        cv2.circle(frame, self.patrones[0], 20, (0, 255, 0), -1)
        cv2.circle(frame, self.patrones[0], 22, (255, 255, 255), 3)

    if len(self.patrones) > 1:
        cv2.circle(frame, self.patrones[-1], 20, (0, 0, 255), -1)
        cv2.circle(frame, self.patrones[-1], 22, (255, 255, 255), 3)

    return frame
```

E. camara.py — Entrada de video y reconocimiento gestual

Responsabilidad:

- Captura video en tiempo real y detecta la mano, el puño y su posición.

Funcionamiento:

- Lee frames desde la cámara, los procesa con MediaPipe, identifica si la mano está cerrada y calcula el centro de la mano para su seguimiento espacial.

Clases importantes:

- obtener_frame(): devuelve el frame procesado, el estado del puño y su posición (x, y).

Python

```
import cv2
import mediapipe as mp

class Camara:
    def __init__(self):
        self.mp_hands = mp.solutions.hands
        self.hands = self.mp_hands.Hands(
            max_num_hands=1,
            model_complexity=0,
            min_detection_confidence=0.7,
            min_tracking_confidence=0.7,
        )
        self.mp_draw = mp.solutions.drawing_utils

    # Estilos de dibujo personalizados
    self.drawing_styles = mp.solutions.drawing_styles

def obtener_frame(self, cap):
```

```
"""Captura frame, detecta la mano y devuelve si es puño y la posición"""
ret, frame = cap.read()
if not ret:
    print("Error: no se pudo leer frame del stream.")
    return None, False, (None, None)

# Voltar horizontalmente para efecto espejo
frame = cv2.flip(frame, 1)

# Convertir a RGB para MediaPipe
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = self.hands.process(rgb)

is_fist = False
cx, cy = None, None

if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        # Dibujar landmarks de la mano con estilo mejorado
        self.mp_draw.draw_landmarks(
            frame,
            hand_landmarks,
            self.mp_hands.HAND_CONNECTIONS,
            self.mp_draw.DrawingSpec(
                color=(0, 255, 0), thickness=2, circle_radius=3
            ),
            self.mp_draw.DrawingSpec(color=(255, 255, 255), thickness=2),
        )

    landmarks = hand_landmarks.landmark
    h, w, _ = frame.shape

    # Detectar si los dedos están cerrados (puño)
    # Verificar dedos índice, medio, anular y meñique
    fingers_closed = [
        landmarks[8].y > landmarks[6].y, # Índice
        landmarks[12].y > landmarks[10].y, # Medio
        landmarks[16].y > landmarks[14].y, # Anular
        landmarks[20].y > landmarks[18].y, # Meñique
    ]

    # Verificar pulgar
    thumb_closed = abs(landmarks[4].x - landmarks[3].x) < 0.05

    # Es puño si todos los dedos están cerrados
    is_fist = all(fingers_closed) and thumb_closed

    # Calcular centro de la mano (entre muñeca y centro de la palma)
```

```
cx = int((landmarks[0].x + landmarks[9].x) / 2 * w)
cy = int((landmarks[0].y + landmarks[9].y) / 2 * h)

# Dibujar indicador visual de puño
if is_fist:
    cv2.putText(
        frame,
        " ",
        (10, 30),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        (0, 255, 0),
        2,
    )
else:
    cv2.putText(
        frame,
        " ",
        (10, 30),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        (0, 165, 255),
        2,
    )

return frame, is_fist, (cx, cy)
```

F. Esquema general de la organización

```
None
Music-therapy/
└── controlador.py
└── vista.py
└── modelo.py
└── utils/
    ├── camara.py
    │   └── Clase: Camara
    └── patrones.py
        └── Clase: GeneradorPatrones
```

```
└── musica.py
    └── Clase: GestorMusica

└── texto_unicode.py
    └── Clase: TextoUnicode
```

IV. Interfaces

Se hablará de la interfaz, como se puede ver en la imagen de abajo el área de jugabilidad es limpia, el punto amarillo es la representación del puño, el punto verde en la figura es el inicio de la forma y el rojo representa el final, el área de la parte de arriba contiene el tiempo que representa el tiempo para realizar al actividad, el puntaje en 0 que cambiará al terminar la figura, y el combo que representa los sin errores que hacemos.



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 25

Al finalizar se cargará otra figura para su realización y se actualizará el puntaje con el obtenido de la figura anterior



V. Mantenimiento a realizar

Para detectar errores se hicieron varios casos de prueba, así como la ayuda de herramientas como pylint para la detección de errores del código, las pruebas de software que se realizaron son las siguientes:

Pruebas unitarias

Número del caso de prueba	Función	Descripción de lo que se probará	Prerrequisitos
PU-001	Funcionamiento de cámara	Se probará el funcionamiento de la cámara, siendo capaz de mostrar una ventana que detecte lo que se encuentra frente a la cámara del usuario	Tener cámara en el dispositivo a usar



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 26

PU-002	Generación de patrones	<i>El sistema genera patrones almacenados de manera aleatoria</i>	<i>Tener almacenado patrones en el sistema</i>
PU-003	Inversión del frame de cámara	<i>La pantalla con la captura de video que se muestra en la pantalla invierte la izquierda con la derecha</i>	<i>La cámara debe funcionar correctamente y ejecutar el programa</i>
PU-004	Detección de mano	<i>El sistema detecta una sola mano mostrando un punto central en la palma y otros a lo largo de dedos</i>	<i>Poner una mano al frente de la cámara</i>
PU-005	Detección de puño	<i>El sistema reconocerá cuando se hace un puño a comparación de otras poses de la mano</i>	<i>Poner un puño frente la cámara</i>
PU-006	Generación Beat rítmico	<i>El sistema generará un beat para poder realizar un movimiento rítmico.</i>	<i>Configurar el ritmo</i>

Pruebas de Integración

Número del caso de prueba	Función	Descripción de lo que se probará	Prerrequisitos
PI-001	Integración de módulo de procesamiento de video con detección	<i>Se probará que el módulo de captura de video de CVZone se conecte correctamente con la función que detecta manos</i>	
PI-002	Integración de entrada de usuario con procesamiento	<i>Se probará que con ejecutar el usuario podrá correr los módulos necesarios</i>	

Pruebas Funcionales

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 27

Número del caso de prueba	Función	Descripción de lo que se probará	Prerrequisitos
PF-001	Confirmación de seguimiento de puño al patrón en pantalla	<i>El sistema debe confirmar de que el puño sigue el patrón</i>	<i>El usuario debe seguir el patrón en la pantalla</i>
PF-002	Contabilización del puntaje	<i>El sistema debe Mostrar el puntaje cuando el usuario hace el patrón correctamente</i>	<i>El usuario debe seguir el patrón en la pantalla de manera correcta</i>
PF-003	Aumento de combo	<i>El sistema debe mostrar en pantalla el combo que hace el usuario según qué tan correcto hace</i>	<i>El usuario debe seguir el patrón en la pantalla sin haber fallas seguidas.</i>
PF-004	Generación de patrón en pantalla	<i>El patrón se debe generar de manera aleatoria en pantalla</i>	<i>El usuario debe correr el programa</i>

Pruebas de Sistemas

Número del caso de prueba	Función	Descripción de lo que se probará	Prerrequisitos
PS-001	Detección de 2 manos	<i>Se probará que el sistema no detecta 2 manos a la vez</i> 	<i>Poner la mano frente a la pantalla</i>
PS-002	Detección de ninguna mano	<i>Se probará que no se detectará cosas que no sean de forma de mano humana</i>	<i>Poner la mano frente a la pantalla</i>

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 28

PS-003	Detección de posiciones de mano	<p><i>El sistema detectará las diferentes posiciones de la mano que estén frente a la pantalla</i></p>	<p><i>Poner la mano frente a la pantalla</i></p>
PS-004	Cumplimiento de patrón	<p><i>Al hacer un patrón correctamente el sistema mostrará otro patrón</i></p>	<p><i>El usuario debe haber completado de manera correcta el patrón en la pantalla</i></p>
PS-005	Incumplimiento de patrón	<p><i>El usuario no hace el patrón correctamente en el tiempo estimado, se mostrará un mensaje de que el usuario ha periodo el ritmo</i></p>	<p><i>No hacer el patrón completo y de manera correcta en el tiempo estimado</i></p>



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 29

Gracias a estas pruebas de software detectamos errores que hacían de nuestro software ineficiente para el usuario, se aislaron en los 3 tipos de errores de mantenimiento y posteriormente corregidos.

A. Mantenimiento Preventivo

- *Se usó la ayuda de las pruebas unitarias para esta parte.*
- *Con el uso de la herramienta pylint se eliminó parte del código que no se usaba.*

B. Mantenimiento Correctivo

- *Después de hacer pruebas de funcionalidad y por recomendación del profesor se corrigieron los mensajes en la pantalla que obstruía con la jugabilidad.*
- *Se corrigió la barra superior eliminando información innecesaria como el Bit y ritmo, dejando únicamente la información importante.*
- *Se mejoró la detección de figuras con la mano.*
- *Se eliminó el tiempo de espera al hacer correcta una figura.*
- *Se estableció un orden en cuanto a las figuras, de la más fácil de hacer al más difícil, estableciendo así un orden de niveles de dificultad para el usuario.*

C. Mantenimiento Evolutivo

- *Como mejoras futuras se podría hacer la implementación de nuevas funciones, no solo para las manos, sino para toda la parte superior.*
- *Desarrollar un programa que tenga distintas funciones para distintas terapias.*
- *Colocarlo en un servidor para hacerlo tipo competitivo.*

VI. Lecciones aprendidas

Como lección aprendida tenemos que el desarrollo de un programa aunque sea una versión de prueba o prototipo lleva su tiempo, pero si se le dedica la motivación necesaria y el tiempo necesario por muy pequeño que sea y resulta funcional es un gran avance, ya que puede servir como base para mejorarse en el futuro o servir de motivación para otros trabajos que sean similares.

A través de este trabajo nos hemos dado cuenta que no todos los proyectos se hacen de manera rápida, ya que hay cosas como los casos de prueba que nos permiten entender más cómo interactúa nuestro programa con el usuario y así comprender más sus necesidades.

Si hubiéramos hecho algo diferente desde el inicio del proyecto, de haberse podido es tener a un paciente que tenga problemas de movilidad con la mano.

Haber realizado pruebas de usabilidad desde el inicio.



Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLE-001

Página: 30

CONCLUSIONES

Es muy importante definir desde una etapa muy temprana todos los requerimientos de nuestro programa. No podemos ir avanzando y ver qué es lo que necesitamos a último momento, debemos investigar primero sobre qué tecnologías son las que nos ayudarán a conseguir nuestro proyecto, Además de haber escogido metodologías que nos ayudarán a cumplir de manera satisfactoria con las entregas de nuestro trabajo.

METODOLOGÍA DE TRABAJO

1. Cascada

Se usó para la planificación de requisitos y todo lo necesario para la etapa inicial del proyecto donde definimos todo aquello que necesitaremos.

2. Metodología Kanban

Se aplica a lo que sería el desarrollo en sí del proyecto ya hablando netamente la codificación, se escogió este modelo porque presenta avances aproximadamente cada 3 semanas, esto es perfecto ya que nuestro proyecto presentará grandes cambios para las entregas de cada unidad.

REFERENCIAS Y BIBLIOGRAFÍA

- [1] L. A. Quilindo y S. Salamanca, "Especificando una arquitectura de software," Revista TIA, vol. 11, n.º 2, pp. 69–81, 2023. [En línea]. Disponible: <https://revistas.udistrital.edu.co/index.php/tia/article/view/18076>
- [2] M. del R. Ramírez-Jiménez, K. Pulido-Hernández, C. E. Rivera-Orozco, N. A. Gómez-Torres, L. Serrano-Zúñiga y L. M. Orozco-Torres, "UML: Una manera de representar, interpretar, analizar y desarrollar el pensamiento computacional," RIDE Revista Iberoamericana para la Investigación y el Desarrollo Educativo, vol. 15, n.º 29, pp. —, 2024. DOI: 10.23913/ride.v15i29.2196