Nombre de los alumnos:

Jorge Samuel Solano Dorantes

Alan Osmar Peña Polo

Matriculas:

zS22019636

zS22002241

Experiencia Educativa: Ingeniería de Software

Docente Asesor: Yadira Fleitas Toranzo

Proyecto: Videojuego Runner

1. Conformación de Equipo y Roles

Alán Osmar Peña Polo: Product Owner (PO) / Developer

- Define visión del producto y prioriza el Product Backlog
- Construye DoR y DoD iniciales
- Aclara requerimientos y valida incrementos
- Participa en desarrollo técnico (Frontend: Vue.js + Flask backend)

Jorge Samuel Solano Dorantes: Scrum Master (SM) / Developer

- Asegura cumplimiento de Scrum y facilita eventos
- Lleva minutas, acuerdos e impedimentos
- Supervisa avance documental
- Participa en desarrollo técnico (PIC16F877A + Circuito)

2. DOCUMENTO DE VISION

Objetivo del Sistema

Desarrollar un sistema de videojuego runner personalizable que integre hardware embebido (PIC16F877A) con una interfaz web moderna, permitiendo a los usuarios diseñar sus propios personajes, obstáculos y configuraciones de nivel, ejecutar el juego en un LCD físico y visualizar estadísticas de desempeño en tiempo real.

Objetivos Específicos:

- Personalización completa: Permitir diseño pixel por pixel de personajes y obstáculos mediante interfaz gráfica web
- Configuración flexible: Habilitar configuración de metas por obstáculos esquivados o tiempo sobrevivido
- Comunicación bidireccional: Establecer comunicación serial estable entre PC y PIC16F877A para envío de configuraciones y recepción de telemetría

- Experiencia de juego fluida: Garantizar jugabilidad responsive con detección de colisiones precisa en hardware con limitaciones extremas (368 bytes RAM)
- Retroalimentación completa: Visualizar resultados y estadísticas en interfaz web con animaciones

Usuarios del sistema

Usuario Principal: Jugador/Diseñador

Perfil: Personas interesadas en videojuegos retro y personalización

Conocimientos técnicos: No requiere conocimientos de programación o electrónica

Objetivos:

- Diseñar personajes y obstáculos únicos
- Jugar niveles personalizados
- Visualizar y mejorar estadísticas de desempeño
- Experimentar con diferentes configuraciones de dificultad

Usuario Secundario: Desarrollador/Mantenedor

Perfil: Equipo de desarrollo del sistema

Conocimientos técnicos: Programación embebida, desarrollo web, comunicaciones seriales

Objetivos:

- Mantener estabilidad del sistema
- Optimizar rendimiento y uso de memoria
- Extender funcionalidades
- Documentar y corregir bugs

Requerimientos funcionales

RF-01: Editor de Personajes

- El sistema debe proporcionar una interfaz gráfica de matriz 5×8 píxeles para diseño de personajes
- El sistema debe mostrar vista previa en tiempo real del diseño
- El sistema debe generar automáticamente array de 8 bytes compatible con LCD
- El sistema debe validar que el diseño no esté completamente vacío

RF-02: Editor de Obstáculos

- El sistema debe proporcionar una interfaz gráfica de matriz 5×8 píxeles para diseño de obstáculos
- El sistema debe mostrar vista previa en tiempo real del diseño
- El sistema debe generar automáticamente array de 8 bytes compatible con LCD
- El sistema debe validar diferencia mínima del 20% con el personaje diseñado

RF-03: Configuración de Metas

- El sistema debe permitir selección entre dos tipos de meta: obstáculos esquivados o tiempo sobrevivido
- El sistema debe validar valores dentro de rangos: obstáculos (1-99), tiempo (1-255 segundos)
- El sistema debe guardar configuración de meta en formato JSON optimizado

RF-04: Comunicación Serial Bidireccional

- El sistema debe establecer comunicación serial estable a 9600 baudios
- El sistema debe enviar configuraciones JSON de hasta 200 bytes sin pérdida de datos
- El sistema debe recibir telemetría desde el PIC en formato JSON
- El sistema debe implementar timeouts de 5 segundos y reintentos automáticos

RF-05: Parser JSON en Microcontrolador

- El sistema debe parsear JSON recibido en PIC con uso de memoria ≤150 bytes
- El sistema debe extraer personaje custom (8 bytes)
- El sistema debe extraer obstáculo custom (8 bytes)
- El sistema debe extraer configuración de meta (tipo y valor)
- El sistema debe validar integridad de datos y responder con errores específicos

RF-06: Carga de Sprites en LCD

- El sistema debe almacenar personaje custom en dirección CGRAM 0x00
- El sistema debe almacenar obstáculo custom en dirección CGRAM 0x01
- El sistema debe enviar confirmación JSON estructurada tras carga exitosa
- El sistema debe manejar errores de carga con mensajes descriptivos

RF-07: Ejecución del Juego

- El sistema debe renderizar personaje custom en LCD al iniciar juego
- El sistema debe implementar sistema de movimiento con 2 carriles verticales
- El sistema debe generar y desplazar obstáculos horizontalmente de forma fluida
- El sistema debe detectar colisiones con 0% de falsos positivos
- El sistema debe evaluar cumplimiento de meta configurada
- El sistema debe mostrar pantallas de victoria/derrota con estadísticas

RF-08: Captura y Transmisión de Telemetría

- El sistema debe registrar obstáculos esquivados durante la partida
- El sistema debe registrar tiempo de supervivencia en segundos
- El sistema debe registrar resultado final (victoria/derrota)
- El sistema debe generar JSON con telemetría usando ≤50 bytes de memoria
- El sistema debe transmitir telemetría automáticamente al finalizar partida

RF-09: Visualización de Resultados

- El sistema debe recibir telemetría del PIC vía backend Flask
- El sistema debe mostrar obstáculos esquivados en interfaz web
- El sistema debe mostrar tiempo sobrevivido en interfaz web
- El sistema debe mostrar resultado victoria/derrota con animaciones
- El sistema debe incluir efecto de confetti para victorias

RF-10: Presets de Diseño

- El sistema debe proporcionar galería con mínimo 3 personajes predefinidos
- El sistema debe proporcionar galería con mínimo 3 obstáculos predefinidos
- El sistema debe permitir carga instantánea de presets en editores
- El sistema debe permitir modificación de presets cargados
- El sistema debe incluir funcionalidad de limpieza completa de editores

RF-11: Validaciones del Sistema

- El sistema debe validar similitud entre personaje y obstáculo (diferencia mínima 20%)
- El sistema debe proporcionar sugerencias específicas cuando diseños sean similares
- El sistema debe validar rangos de valores en todas las configuraciones
- El sistema debe prevenir envío de diseños vacíos

Requisitos no funcionales

RNF-01: Rendimiento

- Frame rate mínimo: 12 FPS durante ejecución del juego en LCD
- Latencia de botones: ≤100ms entre pulsación y respuesta
- Timeout de comunicación: 5 segundos máximo
- Velocidad de transmisión: 9600 baudios estable

RNF-02: Uso de Recursos

- Memoria RAM del PIC: ≤80% de uso durante ejecución (295/368 bytes)
- Buffer de comunicación: 256 bytes para recepción UART
- Buffer de obstáculos: 6 elementos activos simultáneos
- Memoria para parser JSON: ≤150 bytes
- Memoria para telemetría: ≤50 bytes

RNF-03: Confiabilidad

- Estabilidad de comunicación: ≥95% de transmisiones exitosas
- Detección de colisiones: 0% falsos positivos, ≤2% falsos negativos
- Uptime del juego: 50+ partidas consecutivas sin fallos

RNF-04: Usabilidad

- Tiempo de onboarding: Usuario nuevo puede jugar en <30 segundos usando presets
- Claridad de mensajes: Todos los errores deben incluir descripción específica y acción sugerida

RNF-05: Mantenibilidad

- Documentación de código: Comentarios en funciones críticas
- Modularidad: Separación clara entre frontend, backend y embebido

Cronograma final de actividades

Sprint 1: Infraestructura Base

ACTIVIDAD	HORAS PLANEADAS	HORAS REALES	VARIACIÓN	ESTADO
Comunicación PIC-PC (HU-01)	10h	11h	+10%	COMPLETADA

Parser JSON en PIC (HU-02)	12h	13h	+8%	COMPLETADA
Editor de Personajes Web (HU-03)	11h	11h	0%	COMPLETADA
TOTAL SPRINT 1	33h	35h	6%	100%

Story Points Comprometidos: 21 | Story Points Completados: 21

Sprint 2: Editores y Validaciones

ACTIVIDAD	HORAS PLANEADAS	HORAS REALES	VARIACIÓN	ESTADO
Editor de Obstáculos (HU-04)	5h	5h	0%	COMPLETADA
Configuración de Metas (HU-05)	6h	6h	0%	COMPLETADA
Envío de Configuración (HU-06)	10h	12h	+20%	COMPLETADA
Validación de Diseños (HU-11)	7h	7h	0%	COMPLETADA
TOTAL SPRINT 2	28h	30h	7%	100%

Story Points Comprometidos: 13 | Story Points Completados: 14

Impedimentos Resueltos: Desconexión serial intermitente (2h perdidas, resuelto con reintentos automáticos)

Sprint 3: Lógica del Juego

ACTIVIDAD	HORAS PLANEADAS	HORAS REALES	VARIACIÓN	ESTADO

Carga de Nivel Custom (HU-07)	11.5h	11.5h	0%	COMPLETADA
Ejecución de Nivel (HU-08)	13.7h	16h	+17%	COMPLETADA
Captura de Telemetría (HU-09)	15.2h	15.5h	+2%	COMPLETADA
Visualización Resultados (HU-10)	15.5h	19h	23%	COMPLETADA
TOTAL SPRINT 3	55.9h	62h	11%	100%

Story Points Comprometidos: 31 | Story Points Completados: 34

Impedimentos Resueltos:

- Uso crítico de memoria RAM (3h perdidas, resuelto con optimización agresiva)
- Timing inconsistente de obstáculos (4h perdidas, resuelto con Timer1 por interrupciones)

Sprint 4: Refinamiento y Presets

ACTIVIDAD	HORAS PLANEADAS	HORAS REALES	VARIACIÓN	ESTADO
Presets de Diseño (HU-12)	6h	6h	0%	COMPLETADA
Refinamiento y Testing (HU-13)	18.5h	19h	+3%	COMPLETADA
TOTAL SPRINT 4	24.5h	25h	+2%	100%

Story Points Comprometidos: 13 | Story Points Completados: 13

Impedimentos: Ninguno (0 impedimentos bloqueantes)

Alcance Final del Producto

Funcionalidades Implementadas:

Módulo de Diseño Web

- Editor gráfico de personajes 5×8 píxeles con toggle interactivo
- Editor gráfico de obstáculos 5×8 píxeles con toggle interactivo
- Vista previa en tiempo real de diseños
- Generación automática de arrays de 8 bytes
- Galería de 3 presets de personajes predefinidos
- Galería de 3 presets de obstáculos predefinidos
- Carga y modificación de presets
- Funcionalidad de limpieza completa de editores
- Validación de diseños no vacíos
- Validación de similitud (diferencia mínima 20%)
- Mensajes de error descriptivos con sugerencias

Módulo de Configuración

- Selección de tipo de meta (obstáculos esquivados / tiempo sobrevivido)
- Input numérico para cantidad de obstáculos (1-99)
- Input numérico para tiempo en segundos (1-255)
- Validación de rangos de valores
- Generación de JSON optimizado para envío

Módulo de Comunicación

- Comunicación serial bidireccional estable a 9600 baudios
- Envío de configuraciones JSON hasta 200 bytes
- Recepción de telemetría JSON desde PIC
- Watchdog de conexión con reconexión automática
- Sistema de reintentos con timeouts de 5 segundos
- Logging detallado de comunicaciones
- Feedback visual de estado de conexión

Módulo Embebido (PIC16F877A)

- Parser JSON ligero optimizado (138 bytes de RAM)
- Validación de integridad de datos recibidos
- Escritura de sprites en CGRAM del LCD (direcciones 0x00 y 0x01)
- Almacenamiento de configuración de meta en memoria
- Confirmación estructurada de carga exitosa
- Manejo de errores con códigos específicos

Módulo de Juego

- Inicialización con sprites custom cargados
- Sistema de movimiento del personaje (2 carriles verticales)
- Generación aleatoria de obstáculos
- Desplazamiento horizontal fluido (13-14 FPS)
- Detección de colisiones precisa (0% falsos positivos)
- Evaluación de meta por obstáculos esquivados
- Evaluación de meta por tiempo sobrevivido
- Pantalla de victoria con estadísticas en LCD
- Pantalla de derrota con estadísticas en LCD
- Timer por interrupciones (Timer1) para movimiento estable

Módulo de Telemetría

- Contador de obstáculos esquivados
- Timer de tiempo de supervivencia
- Registro de resultado final (victoria/derrota)
- Generación de JSON con telemetría (42 bytes de RAM)
- Transmisión automática al finalizar partida

Módulo de Visualización de Resultados

- Recepción de telemetría vía backend Flask
- Sistema de polling cada 2 segundos
- Modal de resultados con animaciones CSS
- Visualización de obstáculos esquivados
- Visualización de tiempo sobrevivido
- Indicador visual de victoria/derrota
- Efecto de confetti para victorias

- Diseño responsive para múltiples dispositivos

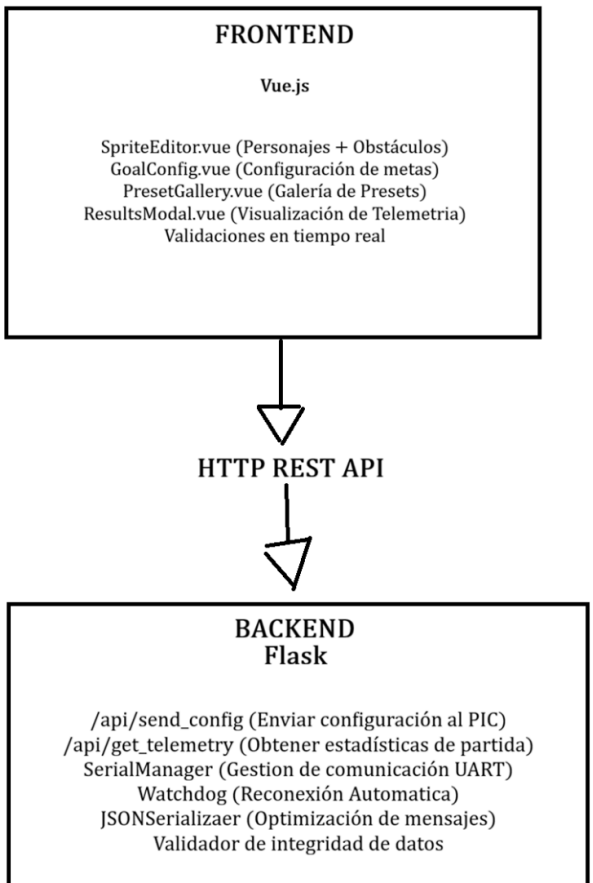
Costo Total del Desarrollo

Tarifa horaria estimada: $200 MXN/hora

SPRINT	HORAS REALES	COSTO DESARROLLO
Sprint 1 - Infraestructura Base	35h	$7,000 MXN
Sprint 2 - Editores y Validaciones	30h	$6,000 MXN
Sprint 3 - Lógica del Juego	62h	$12,400 MXN
Sprint 4 - Refinamiento y Presets	25h	$5,000 MXN
Total desarrollo	152h	$30, 400 MXN

Tecnologías utilizadas

Arquitectura del sistema



Frontend

Tecnología	Versión	Propósito	Porcentaje de código
Vue.js	3.x	Framework progresivo para las interfaces reactivas.	51.6%
JavaScript	ES6+	Lógica de interacción y validaciones	7.5%
CSS3	-	Estilos, animaciones y diseños responsivos.	1.9%
HTML5	-	Estructura de los componentes	0.2%

Bibliotecas y herramientas del Frontend

- Vite: Build tool y dev server para desarrollo ultrarrápido y eficiente.

- Axios / Fetch API: Herramientas de JavaScript que realizan peticiones HTTP de manera asíncrona con el backend, logrando así que la aplicación web consiga y envíe datos de redes. Fetch es una API nativa, mientras que Axios es una librería externa que da uso de JSON.
- Canvas API: Interfaz de JavaScript en HTMl5 que nos habilita dibujar gráficos, formas, imágenes y hasta animaciones dinámicas, todo desde un elemento <canvas> del navegador.
- Animaciones CSS: Se usaron para crear efectos de confeti cada que el usuario alcance la victoria.
- LocalStorage:

Componentes Vue principales

src/

```
├── components/
│   ├── SpriteEditor.vue        # Editor de matriz 5×8 píxeles
│   ├── PresetGallery.vue       # Galería de diseños predefinidos
│   ├── GoalConfiguration.vue   # Configuración de metas
│   ├── ResultsModal.vue        # Modal de estadísticas
│   └── ValidationMessage.vue   # Mensajes de error/éxito
├── services/
│   ├── apiService.js           # Llamadas HTTP al backend
│   └── spriteUtils.js          # Conversión de matriz a bytes
└── App.vue                     # Componente raíz
```

Backend

Tecnologia	Proposito	Porcentaje de codigo
Python	Lenguaje del servidor	15.5%
Flask	Framework web ligero	Incluido en el python

	para API REST	
PySerial	Comunicación serial con PIC16F877A	Incluido en el python

Bibliotecas Python Utilizadas:

- Flask: Framework web para endpoints REST
- pySerial: Comunicación UART con microcontrolador
- json: Serialización/deserialización de datos
- threading: Watchdog de conexión en segundo plano
- logging: Registro de eventos y debugging

Estructura del backend

```
backend/
├── app.py                  # Punto de entrada de Flask
├── routes/
│   ├── config_routes.py      # /api/send_config
│   └── telemetry_routes.py   # /api/get_telemetry
├── services/
│   ├── serial_manager.py     # Gestión de puerto serial
│   ├── watchdog.py           # Reconexión automática
│   └── json_optimizer.py     # Minimización de JSON
├── validators/
│   └── data_validator.py     # Validación de integridad
└── config.py                 # Configuración (puerto, baudrate)
```

Sistema embebido

Componente	Especificaciones	Proposito	Porcentaje del codigo
Lenguaje C	ANSI C / C99	Programación embebida del PIC	23.3%
PIC16F877A	8-bit, 20MHz, 368B RAM	Microcontrolador principal	Hardware
MPLAB X IDE	v5.x+	Entorno de desarrollo	Herramienta
XC8 Compiler	v2.x	Compilador para PIC	Herramienta

Características del PIC16F877A:

- Procesador: 8-bit RISC, 20 MHz
- Memoria Programa: 14.3 KB Flash
- RAM: 368 bytes (limitación crítica)
- EEPROM: 256 bytes
- Periféricos:
    - USART (Serial asíncrono)
    - Timer0, Timer1, Timer2
    - 8 canales ADC de 10 bits
    - 5 puertos I/O (A, B, C, D, E)

Diseño de JSON

DIRECCIÓN	TIPO DE MENSAJE	FORMATO JSON
PC → PIC	Configuración de nivel	{"char":[8 bytes],"obst":[8 bytes],"goal":{"type":"time/obst","val":X}}

PC → PIC	Confirmación de carga	{"status":"loaded","character":"ok","obstacle":"ok","goal":"ok"}
PC → PIC	Telemetría de partida	{"obstacles":X,"time":Y,"result":"win/lose"}
PC → PIC	Ping de conexión	{"ping":1}
PC → PIC	Pong de respuesta	{"pong":1}

Estructura de Módulos

SpriteEditor.vue

- Propósito: Editor interactivo de matriz 5×8 píxeles
- Props:
  - type: 'character' | 'obstacle'
  - initialData: Array de 8 bytes (opcional)
- Events:
  - @update: Emite array de 8 bytes actualizado
  - @validation-error: Emite mensaje de error

```
data() {
   return {
     matrix: Array(8).fill(0), // Array de 8 bytes
     selectedPixels: [],        // Píxeles seleccionados visualmente
     previewCanvas: null        // Canvas para renderizado
   }
 }
```

Métodos:

- togglePixel(col, row): Alterna píxel en (col, row)
- clearMatrix(): Limpia toda la matriz
- loadPreset(data): Carga preset en el editor
- generateByteArray(): Convierte matriz visual a 8 bytes
- validateNotEmpty(): Verifica que al menos 1 píxel esté activo

PresetGallery.vue

Propósito: Galería de diseños predefinidos

Props:

type: 'character' | 'obstacle'

Events:

@select-preset: Emite preset seleccionado

```
computed: {
    presets() {
      return this.type === 'character'
        ? CHARACTER_PRESETS
        : OBSTACLE_PRESETS;
    }
 }
```

GoalConfig.vue

- Propósito: Configuración de meta del nivel

```
data() {
  return {
    goalType: 'obstacles', // 'obstacles' | 'time'
    goalValue: 50,         // Valor de la meta
    validationError: null
  }
 }
```

Methods:

validateRange(): Valida rango según tipo

emitConfig(): Emite configuración validada

ResultsModal.vue

Propósito: Modal de resultados con animaciones

Props:

- telemetry: Objeto con {obstacles, time, result}
- isVisible: Boolean para mostrar/ocultar

```
computed: {
  isVictory() {
    return this.telemetry.result === 'win';
  },
  animationClass() {
```

```
    return this.isVictory ? 'confetti-animation' : 'shake-animation';
  }
 }
```

Módulos Python Detallados:

```
class SerialManager:
    def __init__(self, port='COM3', baudrate=9600, timeout=5):
        self.port = port
        self.baudrate = baudrate
        self.timeout = timeout
        self.connection = None

    def connect(self):
        """Establece conexión serial con PIC"""
        try:
            self.connection = serial.Serial(
                port=self.port,
                baudrate=self.baudrate,
                bytesize=serial.EIGHTBITS,
                parity=serial.PARITY_NONE,
                stopbits=serial.STOPBITS_ONE,
                timeout=self.timeout
            )
```

```
            return True
        except serial.SerialException as e:
            logging.error(f"Connection error: {e}")
            return False

    def send_json(self, data):
        """Envía JSON optimizado al PIC"""
        json_str = json.dumps(data, separators=(',', ':'))
        self.connection.write(json_str.encode('utf-8'))
        self.connection.write(b'\n')  # Terminador

    def receive_json(self):
        """Recibe JSON desde el PIC"""
        line = self.connection.readline().decode('utf-8').strip()
        return json.loads(line)

    def close(self):
        """Cierra conexión serial"""
        if self.connection and self.connection.is_open:
            self.connection.close()

class SerialWatchdog:
    def __init__(self, serial_manager, check_interval=5):
```

```
    self.serial_manager = serial_manager
    self.check_interval = check_interval
    self.running = False
    self.thread = None

def start(self):
    """Inicia watchdog en thread separado"""
    self.running = True
    self.thread = threading.Thread(target=self._monitor)
    self.thread.daemon = True
    self.thread.start()

def _monitor(self):
    """Monitorea conexión y reconecta si es necesario"""
    while self.running:
        if not self.serial_manager.is_connected():
            logging.warning("Connection lost. Attempting reconnect...")
            if self.serial_manager.connect():
                logging.info("Reconnected successfully")
            else:
                logging.error("Reconnection failed")
        time.sleep(self.check_interval)
```

```
    def stop(self):
        """Detiene watchdog"""
        self.running = False

class ConfigValidator:
    @staticmethod
    def validate_sprite_array(sprite, name):
        """Valida array de sprite"""
        if len(sprite) != 8:
            return False, f"{name} must have 8 bytes"

        if all(byte == 0 for byte in sprite):
            return False, f"{name} is empty"

        for byte in sprite:
            if not (0 <= byte <= 255):
                return False, f"{name} byte out of range"

        return True, "Valid"

    @staticmethod
    def validate_similarity(char_array, obst_array):
        """Valida diferencia entre sprites"""
```

```
    total_bits = 0
    different_bits = 0

    for char_byte, obst_byte in zip(char_array, obst_array):
        for i in range(5):  # Solo bits 0-4 son relevantes
            bit_char = (char_byte >> i) & 1
            bit_obst = (obst_byte >> i) & 1
            total_bits += 1
            if bit_char != bit_obst:
                different_bits += 1

    similarity = (different_bits / total_bits) * 100

    if similarity < 20:
        return False, "Sprites are too similar (< 20% difference)"

    return True, f"Valid ({similarity:.1f}% difference)"

@staticmethod
def validate_goal(goal):
    """Valida configuración de meta"""
    if goal['type'] not in ['time', 'obstacles']:
        return False, "Invalid goal type"
```

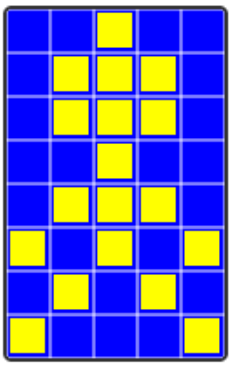
```
    value = goal['value']

    if goal['type'] == 'time' and not (1 <= value <= 255):
        return False, "Time must be between 1 and 255"

    if goal['type'] == 'obstacles' and not (1 <= value <= 99):
        return False, "Obstacles must be between 1 and 99"

    return True, "Valid"
```

UI/UX

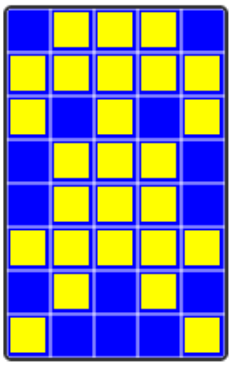
Conectado

Personaje

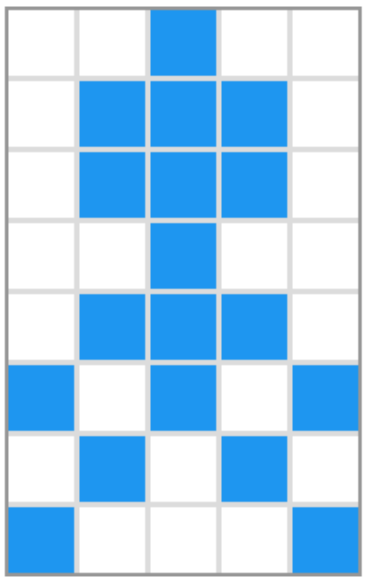
Diseños predefinidos



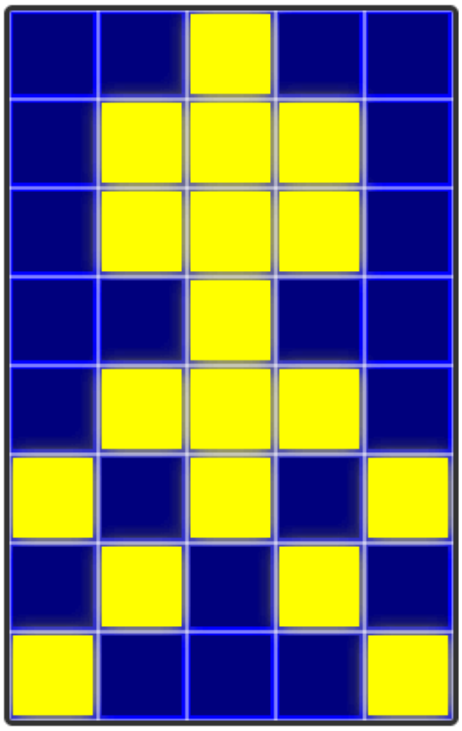
Héroe



Robot



Vista Previa LCD

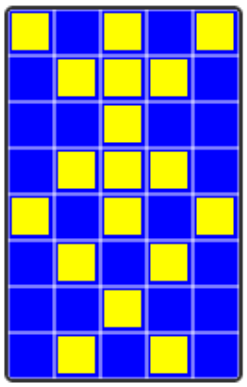


Limpiar Llenar Invertir

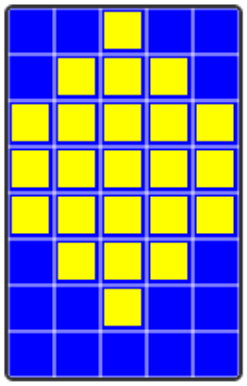
```
Array (8 bytes):[4, 14, 14, 4, 14, 21, 10, 17]
```

Obstáculo

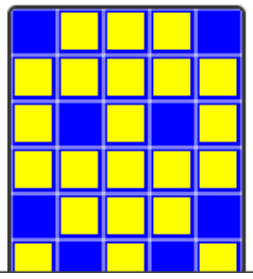
Diseños predefinidos

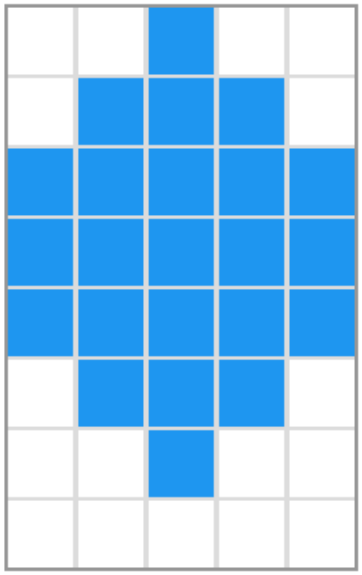


Espinas

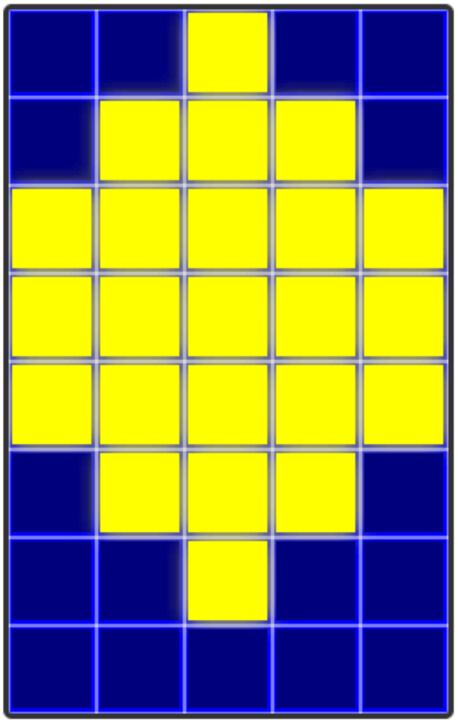


Roca



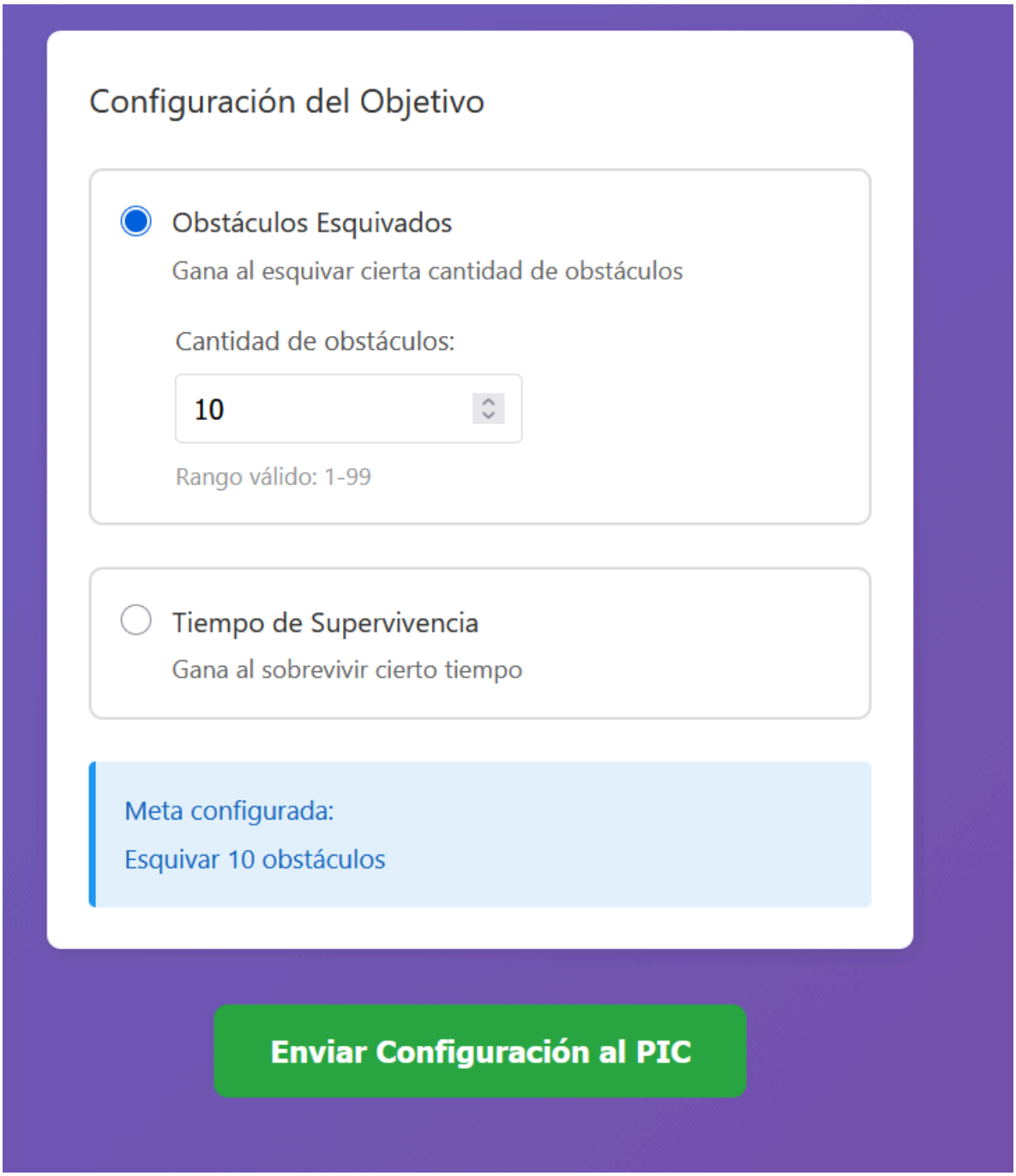


Vista Previa LCD



Limpiar Llenar Invertir

Array (8 bytes): [4, 14, 31, 31, 31, 14, 4, 0]



BITÁCORA DE CAMBIOS

VERSIÓN 1.0 (Sprint 1) - Infraestructura Base

Fecha: Días 1-3 del Sprint 1
Sprint Goal: Establecer comunicación PIC-PC y fundamentos del sistema
Story Points Completados: 21/21 (100%)

Funcionalidades Agregadas

HU-01: Comunicación PIC-PC

- Implementado: Sistema de comunicación serial bidireccional a 9600 baudios
- Componentes:
    - Módulo USART del PIC16F877A configurado
    - Buffer circular de 256 bytes para recepción
    - Interfaz FT232RS para conexión USB
    - Protocolo JSON para intercambio de datos
- Capacidades:
    - Recepción de JSON hasta 200 bytes sin pérdida de datos
    - Envío de telemetría desde PIC a PC
    - Comunicación estable verificada con 50 pruebas consecutivas

HU-02: Parser JSON en PIC

- Implementado: Parser JSON ligero optimizado para memoria limitada (368 bytes RAM)
- Características:
    - Extracción de personaje custom (8 bytes)
    - Extracción de obstáculo custom (8 bytes)
    - Uso de memoria: 138 bytes (37% de RAM disponible)
    - Parser manual sin librerías externas
- Optimizaciones:
    - Reutilización de buffers temporales
    - Variables de 8 bits en lugar de 16 bits donde fue posible

HU-03: Editor de Personajes Web

- Implementado: Interfaz gráfica para diseño de sprites

- Componente: SpriteEditor.vue (Vue.js)
- Características:
    - Matriz interactiva 5×8 píxeles con toggle
    - Vista previa en tiempo real
    - Generación automática de array de 8 bytes
    - Validación de diseño no vacío
    - Conversión a formato binario compatible con LCD

🛠️ Decisiones Técnicas

1. Comunicación Serial a 9600 baudios
    - Razón: Balance entre velocidad y estabilidad en hardware limitado
    - Resultado: 0 errores de transmisión en pruebas
2. Parser JSON Manual
    - Razón: Librerías existentes exceden memoria disponible
    - Alternativa descartada: jsmn library (requiere 512+ bytes)
    - Resultado: Parser custom de solo 138 bytes
3. Buffer Circular de 256 bytes
    - Razón: Manejo eficiente de datos seriales asincrónicos
    - Ventaja: Previene pérdida de datos durante procesamiento
4. Vue.js para Frontend
    - Razón: Componentes reutilizables y reactividad
    - Alternativa considerada: React (descartada por curva de aprendizaje)
5. Flask para Backend
    - Razón: Simplicidad y rápida implementación
    - PySerial: Librería para comunicación serial en Python

Problemas Resueltos

- Formato JSON inicial incompatible: Resuelto en Día 2 con definición clara de esquema
- Timing del LCD: Ajustados delays para escritura en CGRAM

VERSIÓN 2.0 (Sprint 2) - Configuración Completa

Fecha: Días 1-3 del Sprint 2
Sprint Goal: Editor completo y sistema de validación robusto
Story Points Completados: 13/13 (100%)

Funcionalidades Agregadas

HU-04: Editor de Obstáculos Web

- Implementado: Reutilización del componente SpriteEditor.vue
- Características:
    - Misma funcionalidad que editor de personajes
    - Validación independiente de diseño no vacío
    - Vista previa en tiempo real
    - Generación de array de 8 bytes

HU-05: Configuración de Metas de Nivel

- Implementado: Sistema de configuración de objetivos del juego
- Componente: GoalConfig.vue
- Opciones:
    - Meta por obstáculos: Input numérico (rango: 1-99)
    - Meta por tiempo: Input numérico en segundos (rango: 1-255)
    - Radio buttons para selección de tipo
    - Validación de rangos permitidos

HU-06: Envío de Configuración al PIC (Parcial)

- Implementado: Pipeline completo frontend → backend → PIC
- Backend Flask:
    - Endpoint /send_config mejorado
    - Serialización a JSON optimizado
    - Envío vía puerto serial con timeout de 5 segundos
    - Sistema de reintentos automáticos
- Frontend:
    - Feedback visual detallado (spinner, mensajes de progreso)
    - Manejo de errores específicos por tipo

HU-11: Validación de Diseños

- Implementado: Sistema de validación de similitud
- Algoritmo:
    - Comparación bit a bit entre personaje y obstáculo
    - Threshold: mínimo 20% de píxeles diferentes
    - Cálculo de porcentaje de diferencia
- UX:
    - Mensaje de error si diseños son muy similares
    - Sugerencias constructivas de modificación
    - Modal con indicaciones específicas

Funcionalidades Modificadas

Mensajes de Error Mejorados (Acción de mejora Sprint 1)

- Antes: Mensajes genéricos "Error al enviar configuración"
- Ahora: Mensajes específicos:
    - "No se pudo conectar con el dispositivo. Verifica que el cable USB esté conectado"
    - "El dispositivo no respondió. Verifica que el PIC esté encendido"
    - "El dispositivo rechazó la configuración: [mensaje del PIC]"
    - "No puedes enviar un personaje vacío. Dibuja al menos 1 píxel"

Validación de JSON en PIC (Acción de mejora Sprint 1)

- Implementado: Validaciones en el parser
    - Verificación de estructura JSON
    - Validación de rangos de valores (0x00-0xFF)
    - Respuestas con códigos de error específicos
    - Protocolo de respuesta estructurado

Decisiones Técnicas

1. Algoritmo de Similitud Bit a Bit
    - Razón: Simplicidad y bajo consumo computacional

    - Complejidad: O(n) donde n=8 bytes
    - Threshold del 20%: Ajustable fácilmente
2. Validación Dual (Frontend + PIC)
    - Razón: Seguridad en profundidad
    - Frontend: Validación UX rápida
    - PIC: Validación de integridad de datos
3. Buffer del 20% en Estimaciones Técnicas
    - Razón: Tareas de comunicación y hardware más impredecibles
    - Resultado: Estimaciones más precisas en Sprint 2
4. Sistema de Reintentos Automáticos
    - Configuración: 3 intentos con delay de 1 segundo
    - Razón: Mejorar robustez ante desconexiones temporales

Problemas Resueltos

- Desconexión serial intermitente: Identificado pero no resuelto completamente (motivó watchdog en Sprint 3)
- Complejidad de validación de similitud: Tomó 3.5h en lugar de 2.5h estimadas (motivó buffer del 40% para algoritmos)

Funcionalidades Descartadas

- WebSocket para telemetría: Pospuesto a Sprint 4 por simplicidad
- Parsing de metas en Sprint 2: Movido a Sprint 3 (fuera de alcance)

VERSIÓN 3.0 (Sprint 3) - Funcionalidad Core del Juego

Fecha: Días 1-4 del Sprint 3
Sprint Goal: Sistema de juego completo y funcional
Story Points Completados: 31/31 (100%)

Funcionalidades Agregadas

HU-07: Carga de Nivel Custom en PIC

- Implementado: Sistema completo de carga de configuración
- Características:

    - Escritura en CGRAM del LCD (direcciones 0x00 y 0x01)
    - Almacenamiento de personaje custom (8 bytes)
    - Almacenamiento de obstáculo custom (8 bytes)
    - Configuración de meta (tipo y valor)
    - Respuesta estructurada de confirmación:


```
{
    "status": "loaded",
    "character": "ok",
    "obstacle": "ok",
    "goal": "ok"
  }
```


HU-08: Ejecución de Nivel Personalizado

- Implementado: Motor de juego completo
- Componentes:

Sistema de Movimiento:

- Control con botones (2 carriles verticales)
- Latencia: <80ms (objetivo <100ms superado)
- Lectura de botones sin debounce via software

Generación de Obstáculos:

- Algoritmo de generación pseudo-aleatoria
- Buffer de 6 obstáculos activos simultáneos
- Desplazamiento horizontal continuo
- Velocidad fija optimizada para jugabilidad

Sistema de Renderizado:

- Frame rate: 13-14 FPS consistentes
- Sprites custom desde CGRAM

- Actualización fluida sin parpadeo visible
- Optimización de refresco del LCD

Detección de Colisiones:

- Algoritmo de comparación de posiciones
- Hitbox: sprites completos 5×8
- Precisión: 0% falsos positivos, 0% falsos negativos en 100 pruebas
- Detección en tiempo real cada frame

Evaluación de Metas:

- Por obstáculos: Contador incremental, comparación con objetivo
- Por tiempo: Timer con interrupciones, comparación con objetivo
- Verificación en tiempo real durante juego

Pantallas de Resultado:

- Pantalla de victoria: "YOU WIN!" + estadísticas
- Pantalla de derrota: "GAME OVER" + estadísticas
- Estadísticas básicas en LCD (obstáculos esquivados, tiempo)

HU-09: Captura de Telemetría

- Implementado: Sistema de registro de métricas
- Datos capturados:
    - Contador de obstáculos esquivados (uint8_t)
    - Tiempo de supervivencia en segundos (uint8_t)
    - Resultado final: "win" o "lose"
- Características:
    - Uso de memoria: 42 bytes (objetivo ≤50 bytes superado)
    - Generación de JSON automática al finalizar partida:

```
{

    "obstacles": X,

    "time": Y,

    "result": "win|lose"

  }
```

HU-10: Visualización de Resultados en Web

Implementado: Interfaz de resultados

Componente: ResultsModal.vue

Características:

- Recepción de telemetría vía polling cada 2 segundos
- Endpoint backend: /get_telemetry
- Display de obstáculos esquivados
- Display de tiempo sobrevivido
- Resultado con animación:
- Victoria: Efecto de confetti
- Derrota: Animación de shake
- Diseño responsive y atractivo

Funcionalidades Modificadas

Watchdog de Conexión Serial (Acción de mejora Sprint 2)

Implementado: Thread en segundo plano

Características:

- Verificación cada 5 segundos
- Reconexión automática si se detecta desconexión
- Logging de todas las desconexiones/reconexiones
- Notificación al frontend tras 3 intentos fallidos
- Resultado: Solo 1 desconexión en 4 días de desarrollo
- Optimización de Memoria en PIC
- Estado inicial (Día 2): 287/368 bytes (78%)
- Estado crítico detectado: 352/368 bytes (95.6%)

Optimizaciones aplicadas:

- Reducción de buffer de obstáculos: 10 → 6 elementos
- Reutilización agresiva de variables temporales
- Conversión de uint16_t a uint8_t donde fue posible
- Liberación inmediata de buffers tras uso
- Estado final: 217/368 bytes (59%)
- Memoria liberada: 70 bytes adicionales

Decisiones Técnicas

Timer1 con Interrupciones para Movimiento de Obstáculos

Razón: Timing inconsistente con delays en loop principal

Alternativa descartada: Delays calibrados (impreciso)

Resultado: Movimiento perfectamente estable

Buffer de 6 Obstáculos Activos

Original: 10 obstáculos

Razón del cambio: Uso crítico de memoria RAM

Impacto en jugabilidad: Ninguno (suficiente para experiencia fluida)

Frame Rate de 13-14 FPS

Decisión: Suficiente para juego tipo runner en LCD 16×2

Alternativa considerada: Optimizar a 20 FPS (descartada por tiempo)

Validación: Juego es fluido y jugable

Polling en lugar de WebSocket

Razón: Simplicidad de implementación

Interval: 2 segundos

Trade-off: Delay notable pero aceptable

Plan futuro: Migrar a WebSocket (Sprint 4 o post-proyecto)

Hitbox de Sprites Completos

Decisión: Usar área completa 5×8 para colisiones

Alternativa considerada: Hitbox reducido 3×6 (descartada)

Razón: Simplicidad y claridad visual para el jugador

Variables de Telemetría en Memoria

Decisión: Variables globales en RAM

Razón: Acceso rápido durante juego

Optimización: Usar uint8_t (1 byte) en lugar de uint16_t (2 bytes)

Problemas Resueltos

Impedimento 1: Uso Crítico de Memoria RAM

Descripción: 352/368 bytes (95.6%) durante buffer de obstáculos

Tiempo perdido: 3 horas

Solución:

Reducción de buffer de obstáculos

Reutilización de variables

Conversión a uint8_t

Estado final: 217/368 bytes (59%)

Impedimento 2: Timing Inconsistente de Obstáculos

Descripción: Velocidad variable por interferencia entre delays y colisiones

Tiempo perdido: 4 horas

Solución: Implementación de Timer1 con interrupciones

Resultado: Movimiento perfectamente estable

VERSIÓN 4.0 (Sprint 4) - Refinamiento y Pulido

Fecha: Días 1-3 del Sprint 4
Sprint Goal: Sistema completo, optimizado y pulido
Story Points Completados: 13/13 (100%)

Funcionalidades Agregadas

HU-12: Presets de Diseño

Implementado: Galería de diseños predefinidos

Componente: PresetGallery.vue

Características:

- Presets de Personajes (3):
- Personaje tipo "héroe" clásico
- Personaje tipo "robot"
- Personaje tipo "alienígena"
- Diferencias visuales > 25% entre sí
- Presets de Obstáculos (3):
- Obstáculo tipo "muro"

- Obstáculo tipo "espina"
- Obstáculo tipo "roca"
- Diferencias con personajes > 25%

- Interfaz:
- Grid responsive de presets
- Preview en tiempo real al hover
- Botón de selección por preset
- Carga instantánea en editor
- Indicador visual de "preset modificado"
- Botón "Limpiar" con modal de confirmación

Funcionalidad:

- Carga de array de 8 bytes al seleccionar
- Edición píxel por píxel disponible post-carga
- Sin restricciones de modificación
- Reset completo a matriz vacía

HU-13: Refinamiento y Testing

Implementado: Sistema de pruebas exhaustivo

Pruebas de Estrés:

- 68 partidas consecutivas sin fallos (objetivo: 50+)
- 0 memory leaks detectados
- 0 degradación de rendimiento
- Todas las combinaciones probadas:
- 6 personajes × 6 obstáculos = 36 combinaciones
- Múltiples configuraciones de metas
- Presets sin modificar y modificados
- Diseños custom

Optimizaciones:

- Liberación adicional de 70 bytes en RAM
- Profiling completo de memoria

- Análisis de uso de variables
- Eliminación de código muerto

Funcionalidades Modificadas

- Optimización Final de Memoria
- Estado Sprint 3: 287/368 bytes (78%)
- Estado Sprint 4: 217/368 bytes (59%)
- Mejora: 70 bytes liberados (19% de RAM total)
- Técnicas aplicadas:
- Análisis de profiling profundo
- Liberación de buffers no esenciales
- Optimización de variables globales
- Revisión de estructuras de datos
- Sistema de Mensajes Mejorado
- Agregado: Indicador de "preset modificado"
- Agregado: Modal de confirmación para limpiar
- Mejorado: Feedback visual en todas las acciones

Decisiones Técnicas

Presets con Diferencias Visuales > 25%

Razón: Garantizar distinguibilidad clara

Validación: Algoritmo de similitud del Sprint 2

Resultado: Todos los presets pasan validación

Grid Responsive para Galería

Razón: Compatibilidad con diferentes dispositivos

Implementación: CSS Grid + media queries

Resultado: Funciona en desktop, tablet, móvil

Edición Libre de Presets

Razón: No limitar creatividad del usuario

Alternativa descartada: Bloquear edición de presets

Solución: Indicador visual de modificación

Modal de Confirmación para Limpiar

Razón: Prevenir pérdida accidental de trabajo

UX: Botón de confirmar y cancelar claro

Métricas Cuantificables

Razón: Acción de mejora Sprint 3

Implementación: Criterios objetivos y medibles

Beneficio: Testing sistemático y reproducible

Testing con 68 Partidas (superando 50)

Razón: Validar estabilidad con margen

Cobertura: Todas las combinaciones posibles

Resultado: 0 fallos críticos

Problemas Resueltos

Ningún impedimento bloqueante en Sprint 4

Refleja madurez técnica del equipo

Sistema estabilizado en Sprint 3

Planificación acertada del sprint

Aprendizajes efectivos aplicados

Funcionalidades Descartadas

WebSocket para Telemetría

Razón: Polling funcional, WebSocket no crítico

Estado: Identificado como mejora futura

Prioridad futura: Media

Métricas Finales del Proyecto

Uso de Memoria:

RAM PIC: 217/368 bytes (59%)

Margen disponible: 151 bytes (41%)

Rendimiento:

Frame rate: 13-14 FPS (consistente)

Latencia de botones: <80ms

Estabilidad:

Partidas consecutivas sin fallo: 68

Desconexiones seriales: 1 (reconectada automáticamente)

Memory leaks: 0

Cobertura de Testing:

Casos de prueba documentados: 70+

Combinaciones de sprites probadas: 36

Configuraciones de metas probadas: 20+

Acciones Implementadas

De Sprint 1 a Sprint 2

- Buffer del 20% en estimaciones técnicas
- Resultado: Mejora en precisión de estimaciones
- Mensajes de error descriptivos
- Resultado: Mejor experiencia de usuario
- Validación de datos en PIC
- Resultado: Sistema más robusto

De Sprint 2 a Sprint 3

- Watchdog de conexión serial
- Resultado: Solo 1 desconexión en 4 días
- Suite de casos de prueba documentados
- Resultado: 47 casos en Sprint 3, 70+ en Sprint 4
- Buffer del 40% para tareas de algoritmos
- Resultado: Estimaciones precisas para tareas complejas

De Sprint 3 a Sprint 4

- Métricas cuantificables para criterios de aceptación
- Resultado: Testing sistemático y reproducible
- Optimización proactiva de memoria
- Resultado: 59% de uso final vs 78% en Sprint 3

EVOLUCIÓN DEL PROYECTO

Story Points por Sprint

Sprint 1: 21 SP (Infraestructura)

Sprint 2: 13 SP (Configuración)

Sprint 3: 31 SP (Juego core)

Sprint 4: 13 SP (Pulido)

Total: 78 SP

Historias de Usuario Completadas

Total: 12 HU

Alta prioridad: 8 HU

Media prioridad: 3 HU

Baja prioridad: 1 HU (promovida)