Instituto Politecnico Nacional
Escuela Superior de Computo

Materia: Ingeneria de software para sistemas inteligentes.
Grupo: 6BM2

Profesor: Carreto Arellano Chadwick
Periodo: 2025/01

Documentación del Proyecto:

Shoot em up procedural con manejo de dificultad adaptativo al usuario

Realizado por:
Andrade Granados Daniela
Carrillo Barreiro José Emiliano
Huerta Villanueva Oscar

Escuela Superior de Computo
Fecha: 16 de octubre de 2024

Índice

1 Propuesta de Proyecto. 3
- 1.1 Introducción . . . 3
- 1.2 Problemática . . . 3
- 1.3 Objetivo . . . 3
- 1.4 Solución Propuesta . . . 3
- 1.5 Alcance del Proyecto . . . 4

2 Planeación del Proyecto 4
- 2.1 Requerimientos Funcionales . . . 4
  - 2.1.1 Generación Procedural de Niveles . . . 4
  - 2.1.2 Ajuste Dinámico de Dificultad . . . 4
  - 2.1.3 Algoritmo de Aprendizaje Automático . . . 4
  - 2.1.4 Sistema de Feedback Visual y Sonoro . . . 4
  - 2.1.5 Múltiples Tipos de Enemigos y Patrón de Ataques . . . 5
  - 2.1.6 Sistema de Puntuación y Métricas . . . 5
  - 2.1.7 Manejo de Guardado de Datos . . . 5
- 2.2 Requerimientos No Funcionales . . . 5
  - 2.2.1 Rendimiento . . . 5
  - 2.2.2 Escalabilidad del Algoritmo de IA . . . 5
  - 2.2.3 Usabilidad . . . 5
- 2.3 Calculo del proyecto . . . 6
- 2.4 Cronograma de actividades . . . 7

3 Analisis del Sistema 7
- 3.1 Matriz de procesos . . . 7
- 3.2 Diagrama de Actividades . . . 8
- 3.3 Diagramas de procesos detallado. . . . 9
  - 3.3.1 Diagrama de Actividades de Generación Procedural de Niveles . . . 9
  - 3.3.2 Entrenamiento de la IA . . . 10
  - 3.3.3 Gestión de Enemigos y Patrones de Ataque . . . 11
  - 3.3.4 Registro y análisis de puntuación . . . 12
  - 3.3.5 Optimización del Rendimiento . . . 13
  - 3.3.6 Escalabilidad del Algoritmo de IA . . . 14
  - 3.3.7 Interfaz de Usuario y Usabilidad . . . 15
  - 3.3.8 Ajuste dinámico de la dificultad . . . 16
  - 3.3.9 Manejo de guardado de datos . . . 17
  - 3.3.10 Sitema de feedback visual y sonoro . . . 18
- 3.4 Diccionario de Datos . . . 18
  - 3.4.1 Entidad: Jugador . . . 18
  - 3.4.2 Entidad: Partida . . . 18
  - 3.4.3 Entidad: Nivel . . . 19
  - 3.4.4 Entidad: Enemigo . . . 19
  - 3.4.5 Entidad: Ataque . . . 19
  - 3.4.6 Entidad: IA_Modelo . . . 19
  - 3.4.7 Entidad: Estadísticas . . . 20
- 3.5 Relaciones . . . 20
  - 3.5.1 Relación: Juega . . . 20
  - 3.5.2 Relación: Posee . . . 20
  - 3.5.3 Relación: Dispara . . . 20
  - 3.5.4 Relación: Modifica . . . 21
  - 3.5.5 Relación: Genera . . . 21
- 3.6 Diagrama Entidad-Relación (E-R) . . . 21
- 3.7 Generación Procedural de Niveles . . . 21
  - 3.7.1 Vista Funcional . . . 21
  - 3.7.2 Vista Lógica . . . 22
  - 3.7.3 Vista de Secuencia . . . 22

3.7.4 Vista de Despliegue . . . 23
3.7.5 Vista de Estados . . . 23
3.8 Entrenamiento de IA . . . 25
3.8.1 Vista Lógica . . . 25
3.8.2 Vista de Infraestructura . . . 25
3.8.3 Vista de Desarrollo . . . 25
3.8.4 Vista de Ejecucion . . . 26
3.8.5 Vista Funcional . . . 27
3.9 Gestión de Enemigos y Patrones de Ataque . . . 27
3.9.1 Vista Funcional . . . 27
3.9.2 Vista de Comportamientos . . . 28
3.9.3 Vista de Infraestructura . . . 28
3.9.4 Vista de Ejecución . . . 29
3.9.5 Vista de Integración . . . 29
3.9.6 Vista de Verificación y Validación . . . 30
3.9.7 Vista de Desempeño y Escalabilidad . . . 30
3.10 Registro y análisis de puntuación . . . 31
3.10.1 Diagrama de Vistas del Registro y Análisis de Puntuación . . . 31
3.11 Optimización de rendimiento . . . 31
3.11.1 Diagrama de Vistas de la Optimización del Rendimiento . . . 31
3.12 Escalabilidad del algoritmo . . . 32
3.12.1 Diagrama de Vistas de la Escalabilidad del Algoritmo de IA . . . 32
3.13 Ajuste Dinámico de la dificultad . . . 33
3.13.1 Diagrama de Vistas del ajuste dinámico de la dificultad . . . 33
3.14 Sistema de feedback visual y sonoro . . . 33
3.14.1 Diagrama de Vistas del sistema de feedback visual y sonoro . . . 33
3.15 Manejo de guardado de datos . . . 34
3.15.1 Diagrama de Vistas del Manejo de guardado de datos . . . 34

4 Diseño del Sistema **34**
4.1 Diagrama de Arquitectura. . . . 34
4.2 Diagrama de Repositorios/BD . . . 35
4.2.1 Diagrama de Flujo de Información . . . 35
4.3 Diagrama de interfaces/Interfaces tentativas . . . 36
4.4 Diagrama de interfaces . . . 36
4.5 Posibles interfaces. . . . 36
4.6 Pseudocodigos. . . . 38
4.6.1 Algoritmos del proceso de *Generación procedural de niveles.* . . . 38
4.6.2 Algoritmos del proceso de *Ajuste Dinamico de Dificultad.* . . . 41
4.6.3 Algoritmos del proceso del *modelo de IA* . . . 42
4.6.4 Algoritmos del proceso de *Retroalimentacion Visual y Sonora en Tiempo Real.* . . . 45
4.6.5 Algoritmos del proceso d*Gestión de enemigos y patrón de ataques* . . . 48
4.6.6 Algoritmos del proceso de *Gyardar y Cargar datos y Configuraciones de IA* . . . 50
4.6.7 Optimización de rendimiento . . . 52
4.6.8 Escalabilidad del algoritmo de IA . . . 53
4.6.9 Interfaz de Usuario y Usabilidad . . . 53

Índice de figuras

1 Cronograma de tiempo *Milestone* . . . 7
2 Diagrama de Actividades General . . . 8
3 Diagrama de Actividades de Generación Procedural de Niveles . . . 9
4 Diagrama de Actividades del Entrenamiento de IA . . . 10
5 Diagrama de Actividades de Gestión de Enemigos y Patrones de Ataque . . . 11
6 Diagrama de Actividades del Registro y Análisis de Puntuación . . . 12
7 Diagrama de Procesos para la Optimización del Rendimiento . . . 13
8 Diagrama de Procesos para la Escalabilidad del Algoritmo de IA . . . 14
9 Diagrama de Procesos para la Interfaz de Usuario y Usabilidad . . . 15
10 Diagrama de Actividades del ajuste dinamico de la dificultad . . . 16
11 Diagrama de Actividades del manejo de guardado de datos . . . 17
12 Diagrama de Actividades del sistema de feedback visual y sonoro . . . 18
13 Diagrama de entidad-relación de ejemplo . . . 21
14 Vista Funcional . . . 22
15 Vista Lógica . . . 22
16 Vista de Secuencia . . . 22
17 Vista de Despliegue . . . 23
18 Vista de Estados . . . 24
19 Vista Lógica del Sistema . . . 25
20 Vista de Infraestructura del proceso . . . 25
21 Vista de Desarrollo . . . 25
22 Vista de Ejecucion . . . 26
23 Vista Funcional . . . 27
24 Vista Funcional . . . 27
25 Vista de Comportamientos . . . 28
26 Vista de Infraestructura . . . 28
27 Vista de Ejecución . . . 29
28 Vista de Integración . . . 29
29 Vista de Verificación y Validación . . . 30
30 Vista de Desempeño y Escalabilidad . . . 30
31 Vista de Desarrollo del Registro y Análisis de Puntuación . . . 31
32 Vista de Desarrollo de la Optimización del Rendimiento . . . 31
33 Vista de Desarrollo de la Escalabilidad del Algoritmo de IA . . . 32
34 Diagrama de vistas del ajuste dinámico de la dificultad . . . 33
35 Diagrama de vistas del sistema de feedback visual y sonoro . . . 33
36 Diagrama de vistas del manejo de guardado de datos . . . 34
37 Diagrama de la arquitectura de la aplicación. . . . 34
38 Diagrama de Flujo de Información del Proyecto . . . 35
39 Diagrama de flujo de usuario . . . 36
40 Primera pantalla . . . 36
41 Ingreso de usuario . . . 36
42 Registro de usuario . . . 37
43 Posible fonde de nivel . . . 37
44 posible fondo de jefe . . . 37
45 Final de partida . . . 37

Índice de cuadros

1 Costos y periodicidad por ubicación . . . 6
2 Detalles Financieros de ganancias . . . 6
3 Matriz de procesos . . . 7
4 Entidad Jugador . . . 18
5 Entidad Partida . . . 19
6 Entidad Nivel . . . 19
7 Entidad Enemigo . . . 19
8 Entidad Ataque . . . 19

9 Entidad Modelo IA 20
10 Entidad Estadísticas 20
11 Relación Juega entre Jugador y Partida 20
12 Relación Posee entre Partida y Nivel 20
13 Relación Dispara entre Jugador/Enemigo y Ataque 21
14 Relación Modifica entre IA_Modelo y Enemigo 21
15 Relación Genera entre IA_Modelo y Estadísticas 21

1 Propuesta de Proyecto.

1.1. Introducción

Los juegos *shoot 'em up* son un subgénero de los videojuegos de acción donde se controla a un personaje o vehículo que dispara proyectiles a enemigos en hordas o individualmente que van apareciendo mientras esquiva disparos. En la actualidad, los videojuegos de tipo *shoot 'em up* han experimentado una gran evolución, desde mecánicas básicas hasta experiencias enriquecidas con gráficos avanzados y complejas estrategias. Sin embargo, uno de los desafíos más persistentes es el balance de la dificultad, que puede llevar a la frustración o aburrimiento del jugador si no está bien calibrada, además que usualmente se basan en memorizar niveles y patrones de repetición. Este proyecto propone un videojuego *shoot 'em up* procedural, con una inteligencia artificial adaptativa que ajuste la dificultad a las habilidades y estilo de juego del usuario, proporcionando una experiencia personalizada y dinámica en cada partida.

1.2. Problemática

Muchos videojuegos actuales ofrecen niveles de dificultad predefinidos que no siempre satisfacen a los jugadores, especialmente cuando estos tienen diferentes niveles de habilidad o estilos de juego. La falta de una dificultad ajustable personalizada para el jugador puede generar experiencias frustrantes para algunos usuarios, mientras que para otros, el juego puede volverse demasiado predecible y monótono. Además, la dificultad incremental tradicional no siempre garantiza un desafío adecuado, ya que no considera el rendimiento histórico del jugador ni sus preferencias. Esta situación crea una necesidad de sistemas más avanzados que puedan adaptar la experiencia de juego en función del comportamiento del usuario.

1.3. Objetivo

Desarrollar un videojuego *shoot 'em up* procedural que ajuste dinámicamente su dificultad mediante una inteligencia artificial basada en aprendizaje automático. El sistema buscará equilibrar la experiencia de juego en función de las habilidades y preferencias del jugador, aumentando la rejugabilidad y asegurando que cada partida ofrezca un reto adecuado. La IA se entrenará con los datos recopilados de las últimas 15 partidas del usuario, utilizando una combinación de aprendizaje por refuerzo y redes neuronales para ajustar los parámetros del juego.

1.4. Solución Propuesta

La solución consiste en un videojuego con las siguientes características:

- **Generación Procedural:** Los escenarios, enemigos y jefes serán generados de manera procedural, ofreciendo una experiencia diferente en cada partida.
- **Dificultad Adaptativa:** La dificultad se ajustará en tiempo real mediante una IA que analizará el rendimiento del jugador en sus últimas partidas. Esto permitirá una experiencia personalizada para cada jugador.
- **Retroalimentación Multisensorial:** El juego contará con retroalimentación visual y sonora para indicar la progresión del jugador. Los *sprites* de los enemigos cambiarán según su nivel, y el tono al derrotar a un jefe reforzará el sentido de logro.
- **Curva de Dificultad Gradual:** La dificultad aumentará gradualmente, especialmente durante las fases de los jefes, para mantener un balance entre el desafío y la frustración del jugador.
- **Heurística y Aleatoriedad:** El juego utilizará heurísticas para añadir elementos de aleatoriedad, evitando que el jugador memorice patrones de ataque o comportamiento de los enemigos, incrementando la rejugabilidad.

1.5. Alcance del Proyecto

El proyecto abarcará las siguientes fases:

1. **Desarrollo del Motor Procedural:** Crear un sistema de generación procedural de escenarios y enemigos.
2. **Implementación de la IA Adaptativa:** Desarrollar y entrenar una red neuronal que ajuste la dificultad en tiempo real, basada en el análisis del rendimiento del jugador.
3. **Retroalimentación Visual y Sonora:** Implementar un sistema que comunique al jugador su progreso a través de cambios visuales y auditivos.
4. **Curva de Dificultad y Balance:** Definir una curva de dificultad que sea accesible para jugadores principiantes y desafiante para jugadores avanzados.
5. **Pruebas y Ajustes:** Realizar pruebas de usuario para recoger *feedback* y ajustar los parámetros del juego, mejorando la experiencia de juego final.

Este proyecto tiene el potencial de innovar en la forma en que se gestionan los niveles de dificultad en los videojuegos, ofreciendo una experiencia de juego más atractiva, dinámica y equilibrada.

2 Planeación del Proyecto

2.1. Requerimientos Funcionales

2.1.1. Generación Procedural de Niveles

- El sistema debe generar un nivel único y dinámico cada vez que el jugador comience una nueva partida.
- La generación debe incluir la disposición de enemigos, obstáculos y otros elementos del entorno.

2.1.2. Ajuste Dinámico de Dificultad

- La inteligencia artificial debe ajustar la dificultad del juego en tiempo real, en función del rendimiento del jugador durante la partida.
- El ajuste de la dificultad debe tener en cuenta variables como la precisión, número de vidas perdidas, y puntuación obtenida.

2.1.3. Algoritmo de Aprendizaje Automático

- El sistema debe recopilar datos de las últimas 15 partidas del jugador para entrenar el modelo de IA.
- El modelo debe ser capaz de evolucionar según las mejoras o retrocesos en las habilidades del jugador.

2.1.4. Sistema de Feedback Visual y Sonoro

- El sistema debe proporcionar feedback visual y sonoro para indicar cambios en la dificultad (por ejemplo, un aumento de velocidad en los enemigos o más poder de fuego en el jugador).

2.1.5. Múltiples Tipos de Enemigos y Patrón de Ataques

- El juego debe contar con una variedad de enemigos con diferentes patrones de ataque, que se adapten en dificultad según la experiencia del jugador.
- Los patrones de ataque de los enemigos se deben ajustar a correspondencia del nivel de dificultad y tipo de enemigo.

2.1.6. Sistema de Puntuación y Métricas

- El juego debe registrar estadísticas detalladas (puntuación, enemigos eliminados, tiempo jugado, etc.) que puedan ser utilizadas para ajustar la dificultad.
- Debe mostrar estas estadísticas al jugador al final de cada partida.

2.1.7. Manejo de Guardado de Datos

- El sistema debe ser capaz de guardar y cargar datos del progreso del jugador, así como las configuraciones de IA para partidas futuras.

2.2. Requerimientos No Funcionales

2.2.1. Rendimiento

- El juego debe funcionar de manera fluida, con un frame rate mínimo de 60 FPS en condiciones estándar.

2.2.2. Escalabilidad del Algoritmo de IA

- Debe permitir la integración de nuevas variables de juego sin necesidad de rediseñar el sistema.

2.2.3. Usabilidad

- El juego debe ofrecer una interfaz intuitiva para que los jugadores de cualquier nivel puedan entender los controles y mecánicas del juego.
- El sistema de ajuste de dificultad debe ser transparente para el jugador, sin necesidad de intervención manual.

2.3. Calculo del proyecto

Métricas	Detalles	Periodicidad	Costo	Subtotal
Computadoras	Primera	única ocasión	$ 15,000.00	$ 15,000.00
	Segunda	única ocasión	$ 20,000.00	$ 20,000.00
	Tercera	única ocasión	$ 14,000.00	$ 14,000.00
Luz	Primer Lugar	2 meses	$ 401.00	$ 802.00
	Segundo Lugar	2 meses	$ 399.00	$ 798.00
	Tercer Lugar	2 meses	$ 402.00	$ 804.00
Internet	Primer Lugar	3 meses	$ 850.00	$ 2,550.00
	Segundo Lugar	3 meses	$ 760.00	$ 2,280.00
	Tercer Lugar	3 meses	$ 900.00	$ 2,700.00
Personal	Desarrollador 1	3 meses	$ 24,000.00	$ 72,000.00
	Desarrollador 2	3 meses	$ 24,000.00	$ 72,000.00
	Desarrollador 3	3 meses	$ 24,000.00	$ 72,000.00
Licencias	Windows	1 año	$ 299.00	$ 299.00
	Office 365	1 año	$ 1,749.00	$ 1,749.00
	SteamWork	única ocasión	$ 1,930.00	$ 1,930.00
	Linux	2 OS	-	-
			TOTAL	$ 278,912.00

Cuadro 1: Costos y periodicidad por ubicación

Concepto	Valor
Ganancia Total	$320,748.80
Ganancia Neta	$41,836.80
Dividendo 1	$13,945.60
Dividendo 2	$13,945.60
Dividendo 3	$13,945.60
Precio de salida	$39.99
Porcentaje de Steam	5 %
Cantidad de copias para ganancia neta	$8,442.87

Cuadro 2: Detalles Financieros de ganancias

2.4. Cronograma de actividades

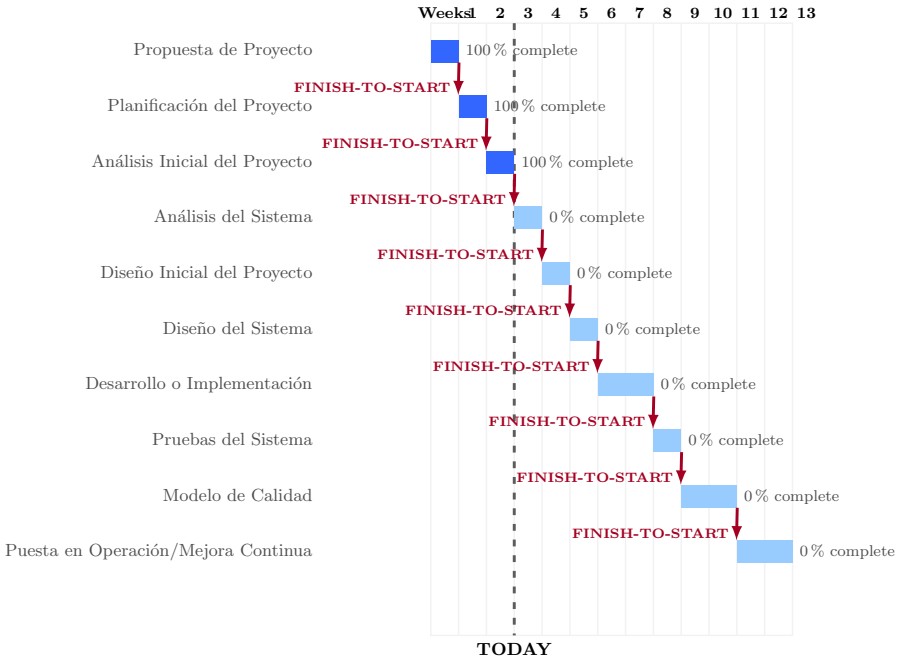

Figura 1: Cronograma de tiempo *Milestone*

3 Analisis del Sistema

3.1. Matriz de procesos

Proceso	Descripción	Entradas	Salidas	Actores
Generación Procedural de Niveles	Genera niveles dinámicos cada vez que el jugador comienza una nueva partida.	Parámetros de generación (enemigos, obstáculos, entorno).	Nivel único y dinámico con enemigos y obstáculos distribuidos proceduralmente.	Motor procedural, sistema de IA.
Ajuste Dinámico de Dificultad	Ajusta la dificultad del juego en tiempo real según el rendimiento del jugador.	Datos en tiempo real del jugador (vidas perdidas, precisión, puntuación).	Modificaciones en la dificultad (número de enemigos, velocidad, patrones).	Sistema de IA, motor de juego.
Entrenamiento del Algoritmo de IA	Entrena el modelo de IA basado en las últimas 15 partidas del jugador.	Datos históricos de las últimas 15 partidas (estadísticas, acciones).	Modelo de IA actualizado para ajustar la dificultad adaptativamente.	Sistema de IA, motor de aprendizaje.
Sistema de Feedback Visual y Sonoro	Proporciona retroalimentación visual y sonora durante la partida.	Eventos del juego (derrotas, cambios en dificultad, logros, avance).	Indicaciones visuales y sonoras de cambios en el juego (sprites, efectos).	Motor gráfico, motor de sonido.
Gestión de Enemigos y Patrones de Ataque	Gestiona la creación de enemigos con patrones de ataque adaptativos.	Variables de dificultad, tipo de enemigos, nivel del jugador.	Enemigos con patrones de ataque ajustados a la dificultad.	Sistema de IA, motor procedural.
Registro y Análisis de Puntuación	Registra las métricas y estadísticas del jugador durante la partida.	Estadísticas de la partida (puntuación, tiempo, enemigos eliminados).	Tabla de estadísticas mostrada al final de la partida.	Sistema de estadísticas, interfaz.
Manejo de Guardado de Datos	Guarda y carga los datos de progreso y configuraciones de IA.	Estado del jugador, configuración de IA.	Datos guardados para cargar en futuras partidas.	Sistema de almacenamiento.
Optimización del Rendimiento	Asegura un funcionamiento fluido del juego a 60 FPS o más.	Recursos del juego, carga del sistema, frame rate actual.	Juego fluido y sin caídas de frame rate.	Motor gráfico, sistema de rendimiento.
Escalabilidad del Algoritmo de IA	Permite añadir nuevas variables sin rediseñar el sistema.	Nuevas variables o métricas del juego.	Algoritmo de IA escalable que soporte nuevas variables.	Sistema de IA, motor de aprendizaje.
Interfaz de Usuario y Usabilidad	Proporciona una interfaz clara y fácil de usar para cualquier tipo de jugador.	Diseño de la interfaz, controles de usuario.	Interfaz intuitiva y accesible, ajustes automáticos de dificultad.	Sistema de interfaz de usuario.

Cuadro 3: Matriz de procesos

Explicación de la Matriz

- **Entradas:** Son los datos o parámetros que el sistema requiere para realizar cada proceso.
- **Salidas:** Son los resultados obtenidos después de completar el proceso.
- **Actores:** Identifican los sistemas, módulos o actores involucrados en ejecutar el proceso.

3.2. Diagrama de Actividades

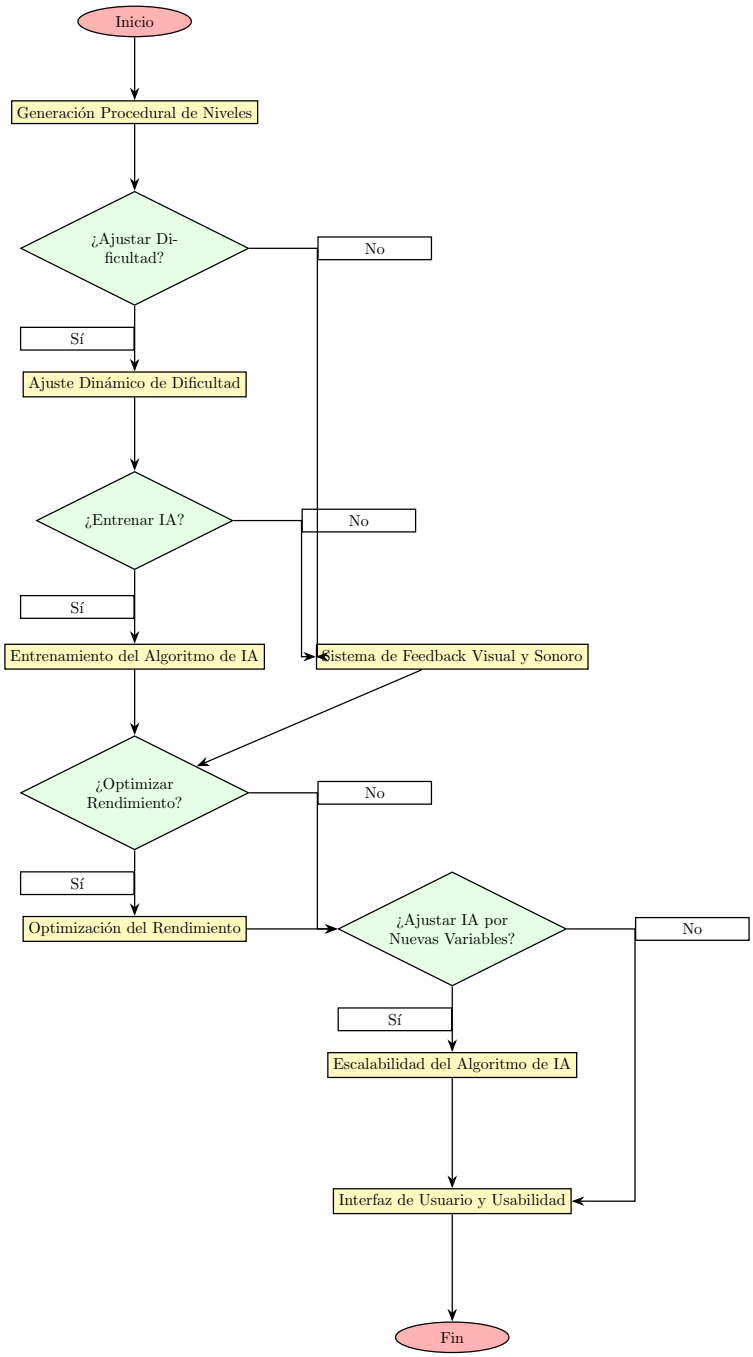

Figura 2: Diagrama de Actividades General

3.3. Diagramas de procesos detallado.

3.3.1. Diagrama de Actividades de Generación Procedural de Niveles

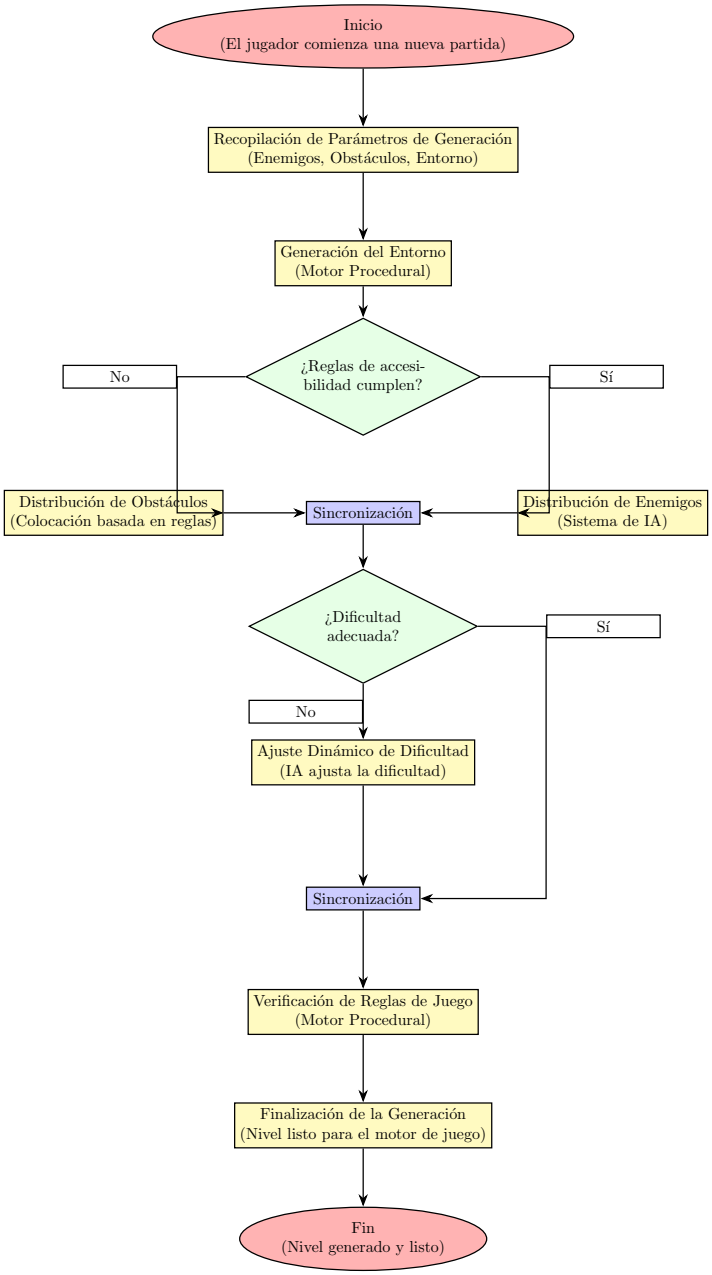

Figura 3: Diagrama de Actividades de Generación Procedural de Niveles

3.3.2. Entrenamiento de la IA

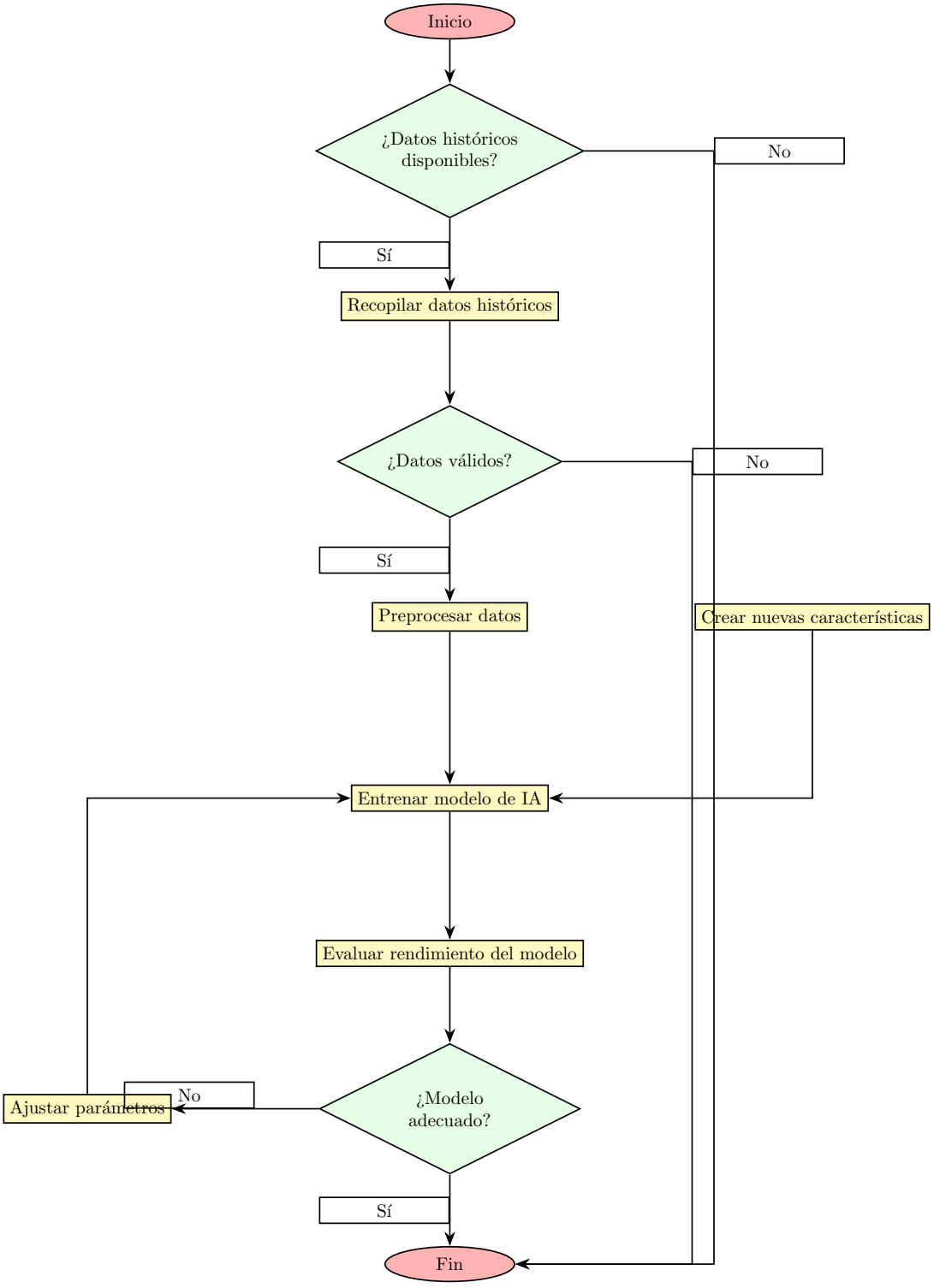

Figura 4: Diagrama de Actividades del Entrenamiento de IA

3.3.3. Gestión de Enemigos y Patrones de Ataque

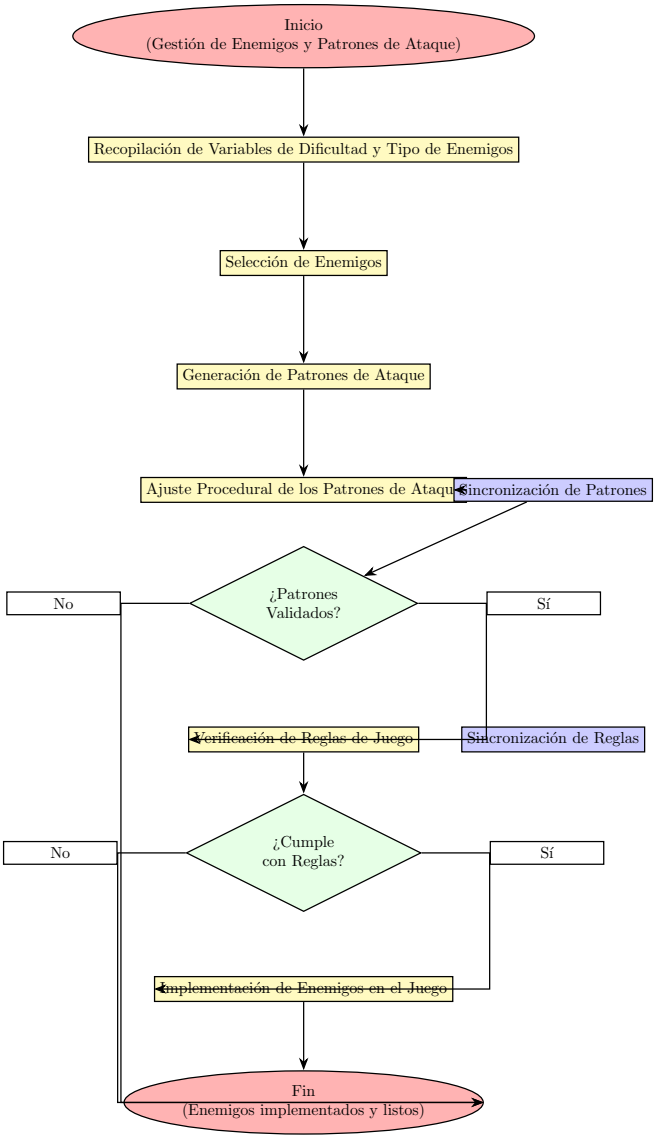

Figura 5: Diagrama de Actividades de Gestión de Enemigos y Patrones de Ataque

3.3.4. Registro y análisis de puntuación

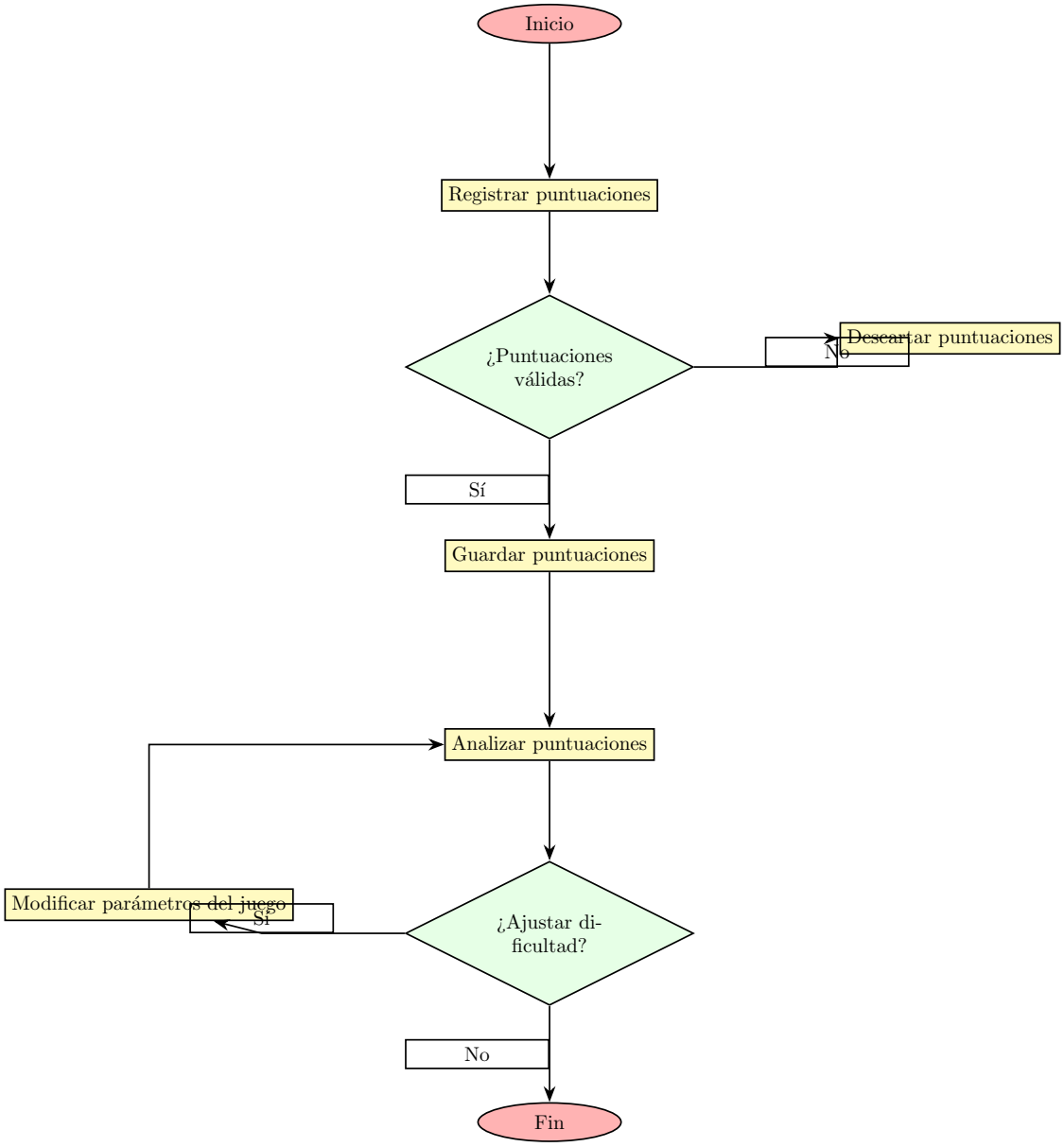

Figura 6: Diagrama de Actividades del Registro y Análisis de Puntuación

3.3.5. Optimización del Rendimiento

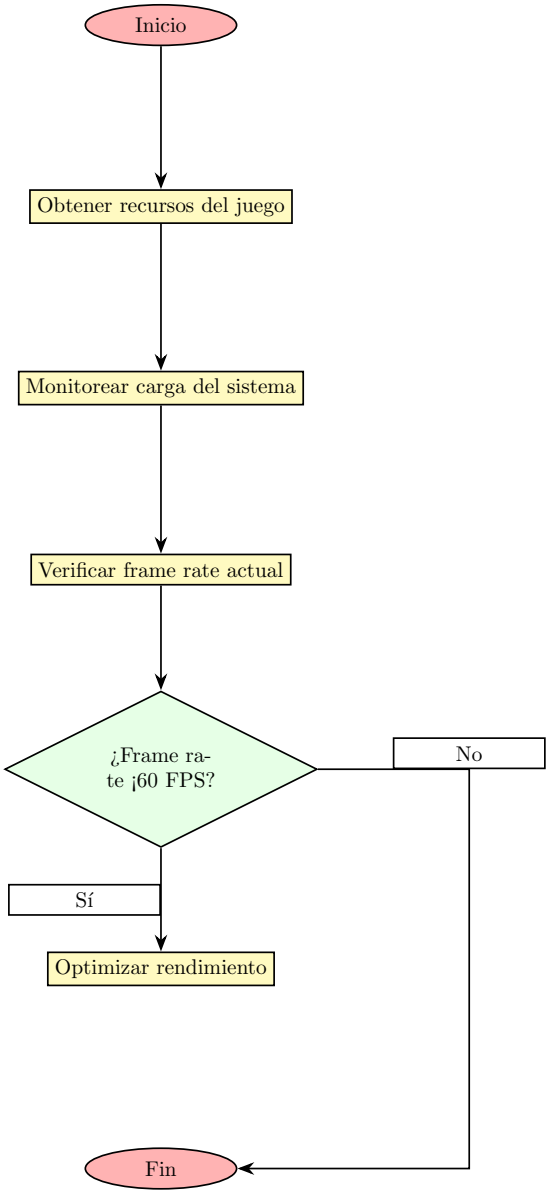

Figura 7: Diagrama de Procesos para la Optimización del Rendimiento

3.3.6. Escalabilidad del Algoritmo de IA

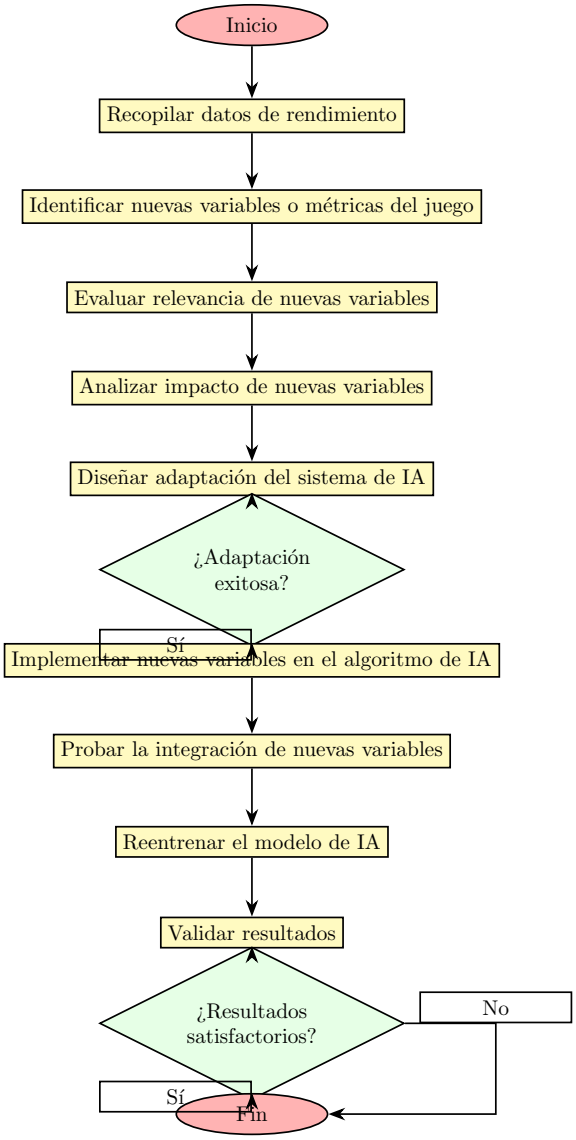

Figura 8: Diagrama de Procesos para la Escalabilidad del Algoritmo de IA

3.3.7. Interfaz de Usuario y Usabilidad

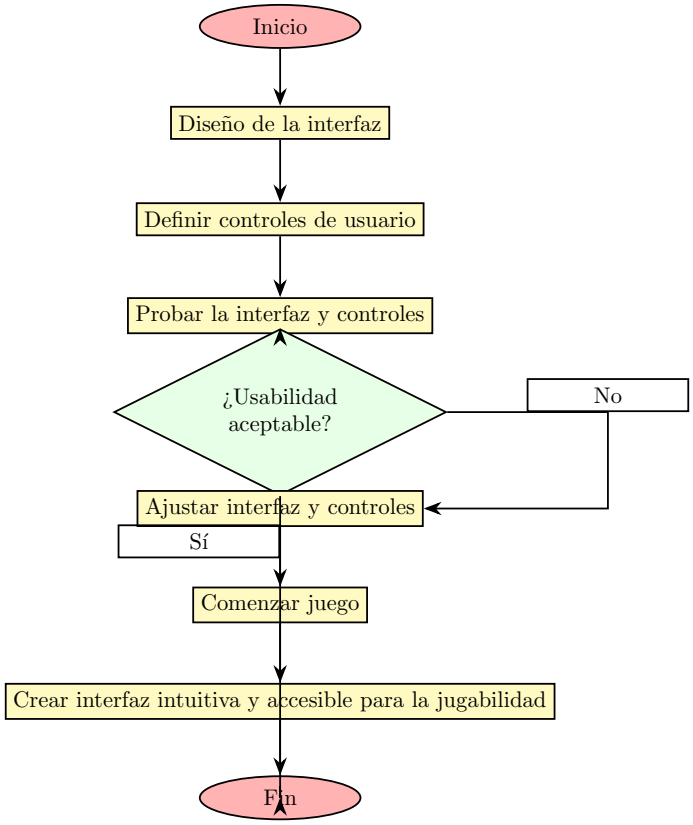

Figura 9: Diagrama de Procesos para la Interfaz de Usuario y Usabilidad

3.3.8. Ajuste dinámico de la dificultad

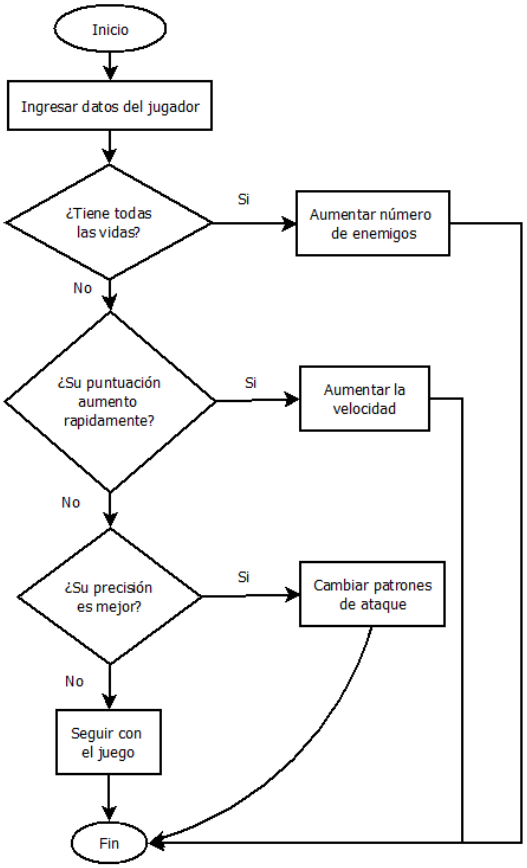

Figura 10: Diagrama de Actividades del ajuste dinamico de la dificultad

3.3.9. Manejo de guardado de datos

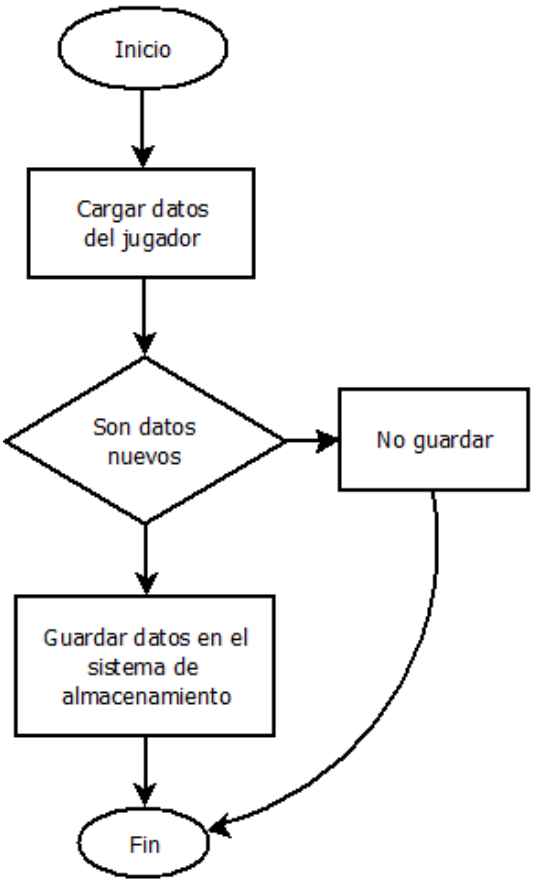

Figura 11: Diagrama de Actividades del manejo de guardado de datos

3.3.10. Sitema de feedback visual y sonoro

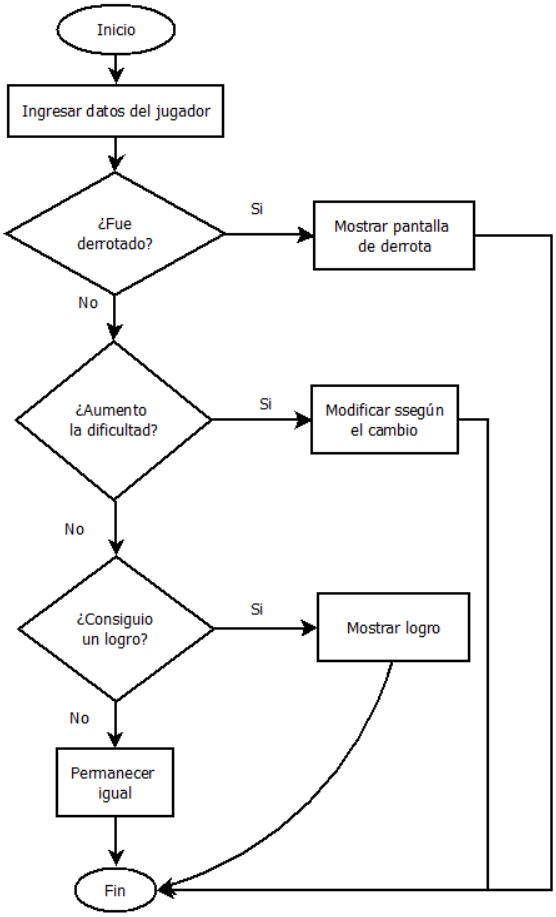

Figura 12: Diagrama de Actividades del sistema de feedback visual y sonoro

3.4. Diccionario de Datos

3.4.1. Entidad: Jugador

Esta entidad representa a los jugadores en el sistema, almacenando información básica sobre su identidad y desempeño en el juego.

Atributo	Tipo de Dato	Descripción
J_ID	`int`	Identificador único del jugador
Nombre	`varchar`	Nombre del jugador
Precision	`float`	Precisión del jugador durante la partida

Cuadro 4: Entidad Jugador

3.4.2. Entidad: Partida

Esta entidad representa una sesión individual de juego, almacenando datos específicos de cada partida jugada.

Atributo	Tipo de Dato	Descripción
ID	`int`	Identificador único de la partida
Pje_P	`int`	Puntaje obtenido en la partida
HP_Lost	`int`	Total de vidas perdidas durante la partida
T_Jgd	`float`	Tiempo jugado en minutos

Cuadro 5: Entidad Partida

3.4.3. Entidad: Nivel

Esta entidad define las características de cada nivel dentro del juego, como su dificultad, enemigos y obstáculos generados.

Atributo	Tipo de Dato	Descripción
ID	`int`	Identificador único del nivel
Pts_Obt	`int`	Puntos obtenidos en el nivel
E_Gen	`int`	Enemigos generados en el nivel
E_Del	`int`	Enemigos eliminados en el nivel
Obs_Gen	`int`	Obstáculos generados en el nivel
N_Difi	`int`	Dificultad del nivel
Vel_Scroll	`float`	Velocidad de desplazamiento del nivel

Cuadro 6: Entidad Nivel

3.4.4. Entidad: Enemigo

Esta entidad describe los enemigos que aparecen en el juego, incluyendo su tipo y cantidad de vida.

Atributo	Tipo de Dato	Descripción
ID	`int`	Identificador único del enemigo
T_En	`varchar`	Tipo de enemigo
E_Enmy	`int`	Vida del enemigo

Cuadro 7: Entidad Enemigo

3.4.5. Entidad: Ataque

Esta entidad almacena los datos relativos a los ataques en el juego, como el patrón, frecuencia y potencia del ataque.

Atributo	Tipo de Dato	Descripción
ID	`int`	Identificador único del ataque
Ptn_Atq	`json`	Patrón de ataque
Freq_Atq	`int`	Frecuencia de ataque
Pow_Atq	`int`	Potencia del ataque

Cuadro 8: Entidad Ataque

3.4.6. Entidad: IA_Modelo

Esta entidad contiene los datos relacionados con el modelo de inteligencia artificial utilizado para ajustar la dificultad del juego.

Atributo	Tipo de Dato	Descripción
ID	`int`	Identificador único del modelo de IA
Metricas	`json`	Métricas utilizadas por el modelo de IA
Tasa_Ajst	`float`	Tasa de ajuste de dificultad
Tm_Adapt	`float`	Tiempo de adaptación del modelo
DesvEst	`float`	Desviación estándar de la dificultad
Heuristica	`json`	Datos heurísticos utilizados por la IA

Cuadro 9: Entidad Modelo IA

3.4.7. Entidad: Estadísticas

Esta entidad almacena estadísticas globales y específicas del jugador, como puntajes y precisión, para el análisis del rendimiento.

Atributo	Tipo de Dato	Descripción
ID	`int`	Identificador único de las estadísticas
M_Sc	`int`	Mejor puntaje alcanzado
Ptj_Ptj	`int`	Puntaje obtenido en la partida
Best_Ptj	`int`	Mejor puntaje registrado
Ttl_Tiempo	`float`	Tiempo total jugado
Tm_media	`float`	Tiempo medio por partida
PrecisionJdr	`float`	Precisión promedio del jugador
EstrategiaMov	`json`	Estrategia de movimiento utilizada por el jugador

Cuadro 10: Entidad Estadísticas

3.5. Relaciones

3.5.1. Relación: Juega

Representa la relación entre un jugador y las partidas que ha jugado. Un jugador puede jugar varias partidas.

Entidad 1	Entidad 2	Cardinalidad
Jugador	Partida	1:N

Cuadro 11: Relación Juega entre Jugador y Partida

3.5.2. Relación: Posee

Define la relación entre una partida y los niveles jugados. Una partida puede tener varios niveles.

Entidad 1	Entidad 2	Cardinalidad
Partida	Nivel	1:N

Cuadro 12: Relación Posee entre Partida y Nivel

3.5.3. Relación: Dispara

Representa la relación entre un jugador o un enemigo y los ataques que realizan. Un jugador o enemigo puede realizar varios ataques, y un ataque puede ser utilizado por varios enemigos.

Entidad 1	Entidad 2	Cardinalidad
Jugador	Ataque	1:N
Enemigo	Ataque	N:M

Cuadro 13: Relación Dispara entre Jugador/Enemigo y Ataque

3.5.4. Relación: Modifica

Define la relación entre el modelo de IA y los enemigos, indicando que el modelo de IA puede modificar varios enemigos, y un enemigo puede ser modificado por varios modelos de IA.

Entidad 1	Entidad 2	Cardinalidad
IA_Modelo	Enemigo	1:N

Cuadro 14: Relación Modifica entre IA_Modelo y Enemigo

3.5.5. Relación: Genera

Representa la relación entre la IA y las estadísticas generadas a partir del comportamiento de los jugadores. El modelo de IA puede generar varias estadísticas.

Entidad 1	Entidad 2	Cardinalidad
IA_Modelo	Estadísticas	1:N

Cuadro 15: Relación Genera entre IA_Modelo y Estadísticas

3.6. Diagrama Entidad-Relación (E-R)

Figura 13: Diagrama de entidad-relación de ejemplo

Diagramas de vistas.

3.7. Generación Procedural de Niveles

3.7.1. Vista Funcional

Este diagrama representa la Vista de Caso de Uso para la Generación Procedural de Niveles. Describe las interacciones entre los actores principales del sistema y el proceso de generación de niveles.

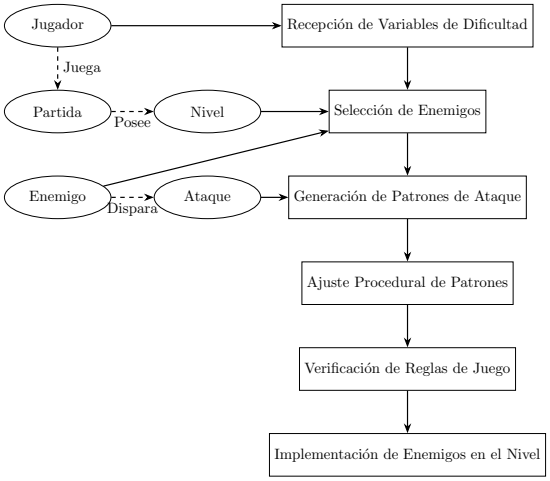

Figura 14: Vista Funcional

3.7.2. Vista Lógica

Este diagrama representa la Vista Lógica (Diagrama de Clases) para el sistema de generación procedural de niveles, mostrando las principales clases y sus interacciones. A su ves proporciona una visión clara de cómo las distintas clases y componentes del sistema interactúan entre sí en el contexto de la generación procedural de niveles.

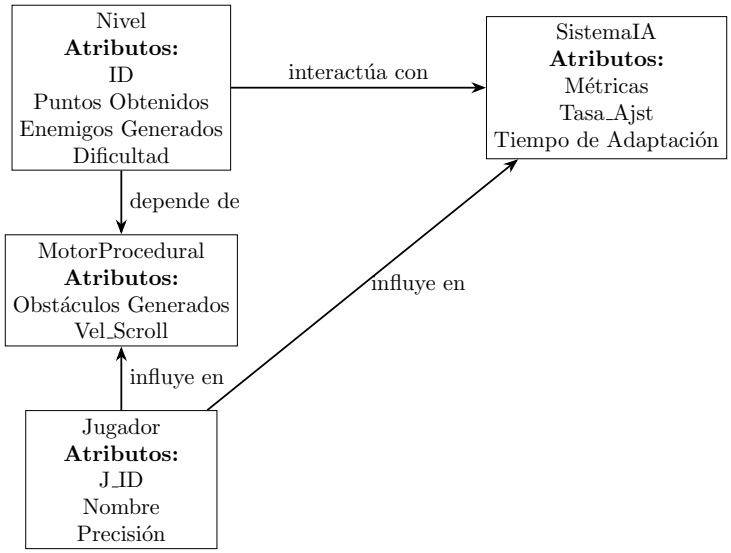

Figura 15: Vista Lógica

3.7.3. Vista de Secuencia

Este Diagrama de Secuencia muestra la interacción y el flujo de mensajes entre los actores principales y los componentes del sistema durante la generación de un nivel procedimental. Además de mostrar cómo las interacciones temporales entre los componentes del sistema aseguran que el nivel generado sea jugable y esté ajustado adecuadamente en cuanto a dificultad y distribución de enemigos.

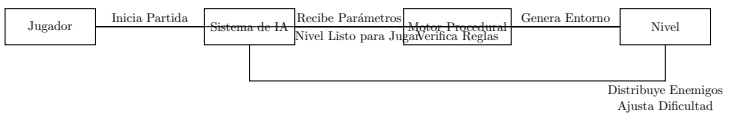

Figura 16: Vista de Secuencia

3.7.4. Vista de Despliegue

Este diagrama asegura que los roles y responsabilidades de los componentes estén claramente separados, con el servidor encargado de la lógica pesada y el cliente enfocado en la interacción con el jugador.

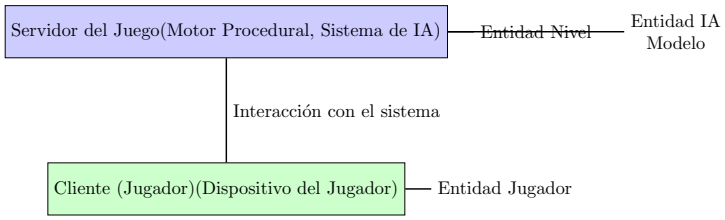

Figura 17: Vista de Despliegue

3.7.5. Vista de Estados

Este Diagrama de Estados muestra los estados por los que pasa un nivel durante su generación procedural en el sistema.

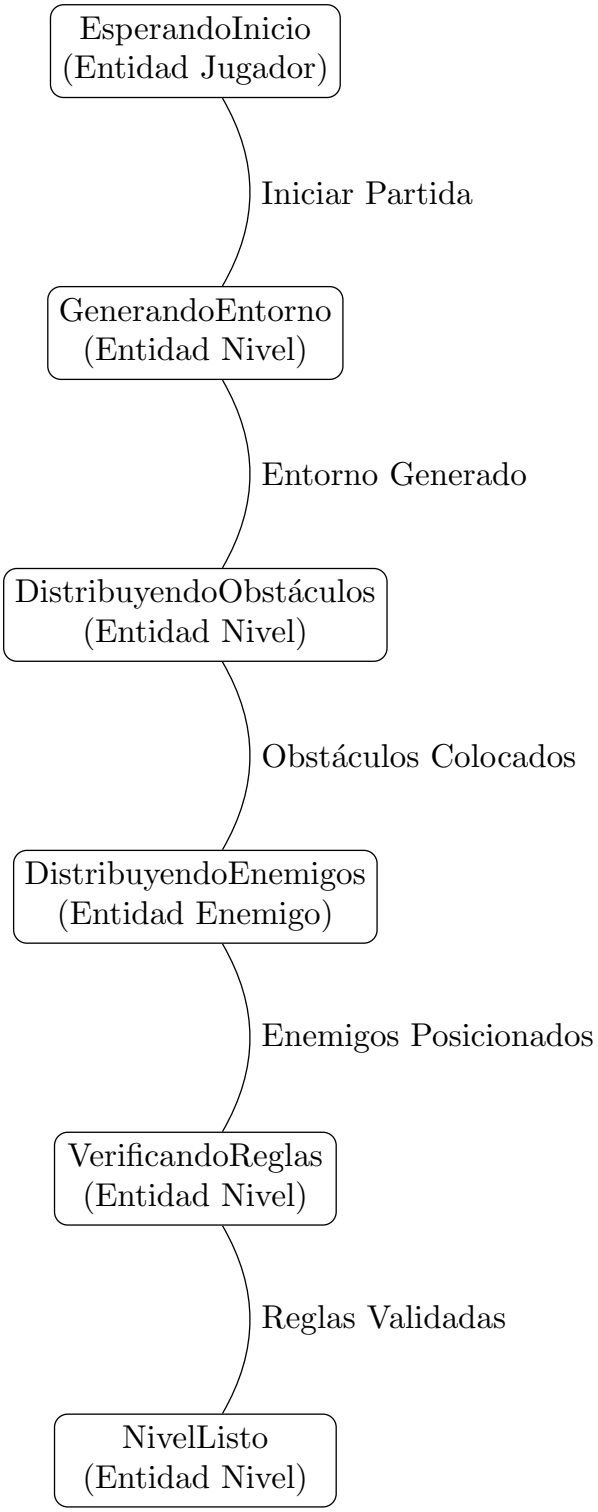

Figura 18: Vista de Estados

3.8. Entrenamiento de IA

3.8.1. Vista Lógica

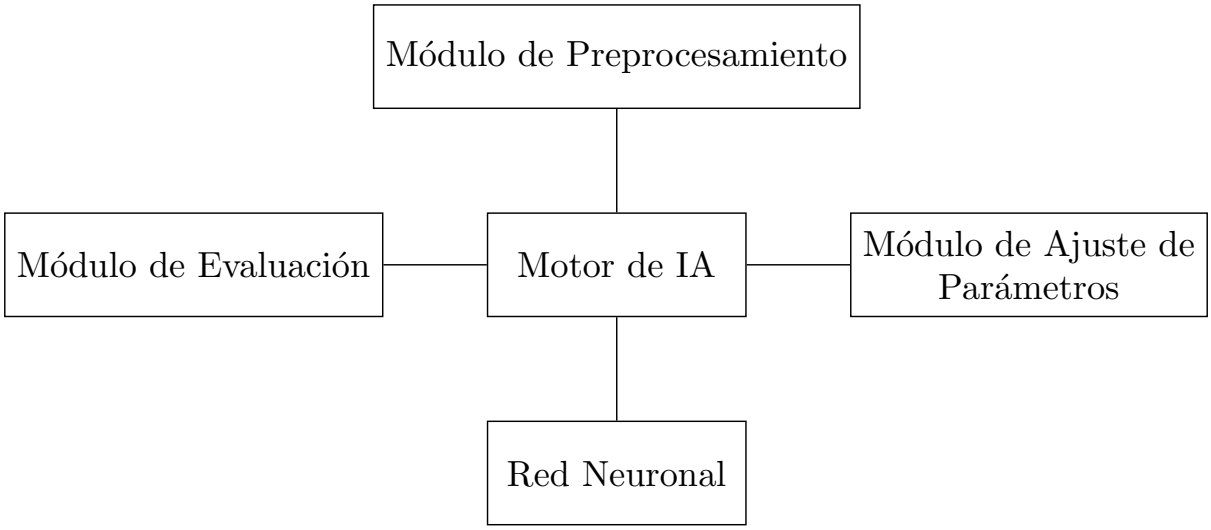

Figura 19: Vista Lógica del Sistema

3.8.2. Vista de Infraestructura

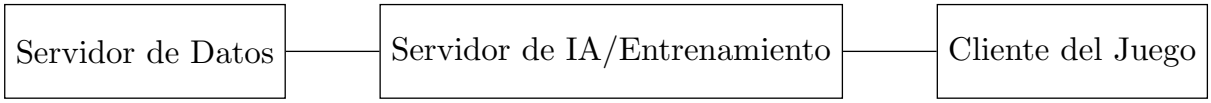

Figura 20: Vista de Infraestructura del proceso

3.8.3. Vista de Desarrollo

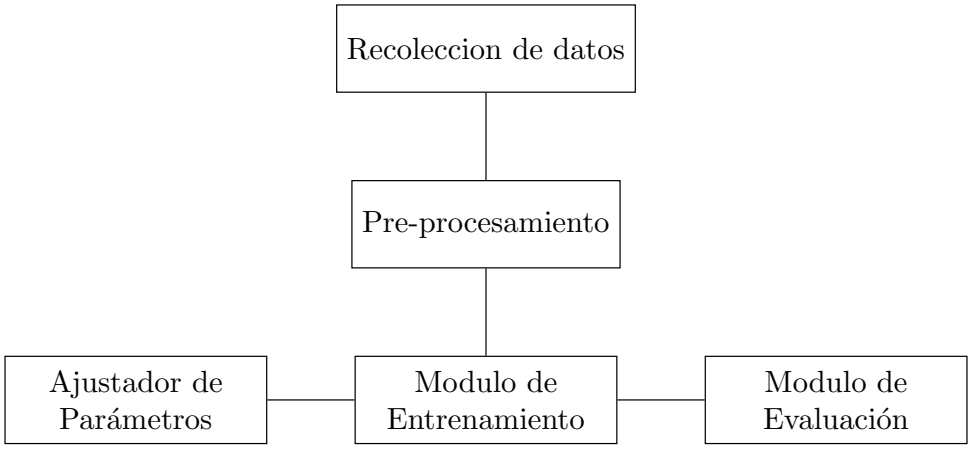

Figura 21: Vista de Desarrollo

3.8.4. Vista de Ejecucion

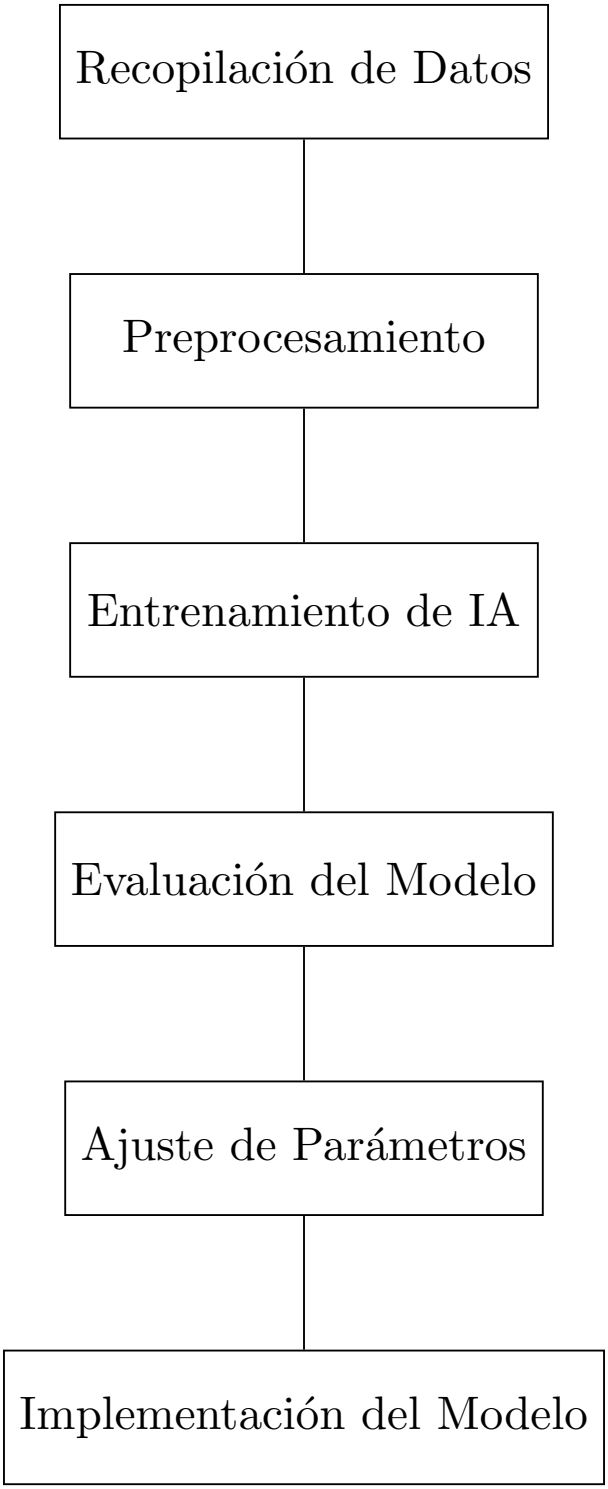

Figura 22: Vista de Ejecucion

3.8.5. Vista Funcional

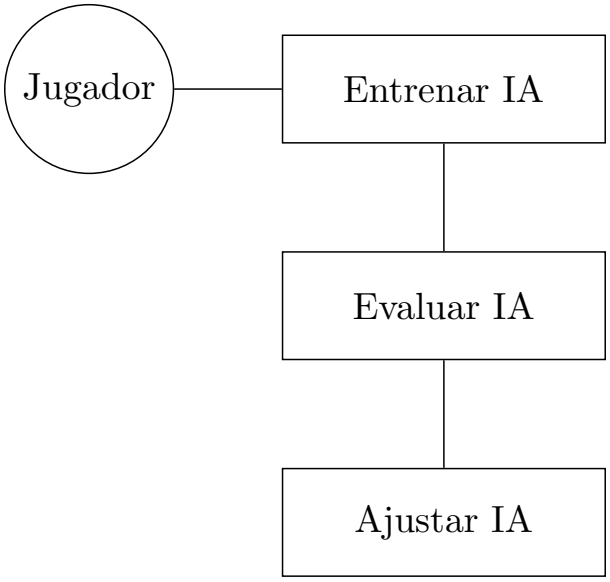

Figura 23: Vista Funcional

3.9. Gestión de Enemigos y Patrones de Ataque

3.9.1. Vista Funcional

Este diagrama muestra cómo las entidades y procesos interactúan para gestionar la dificultad y ajustar los patrones de ataque de los enemigos en tiempo real, de acuerdo con el progreso del jugador.

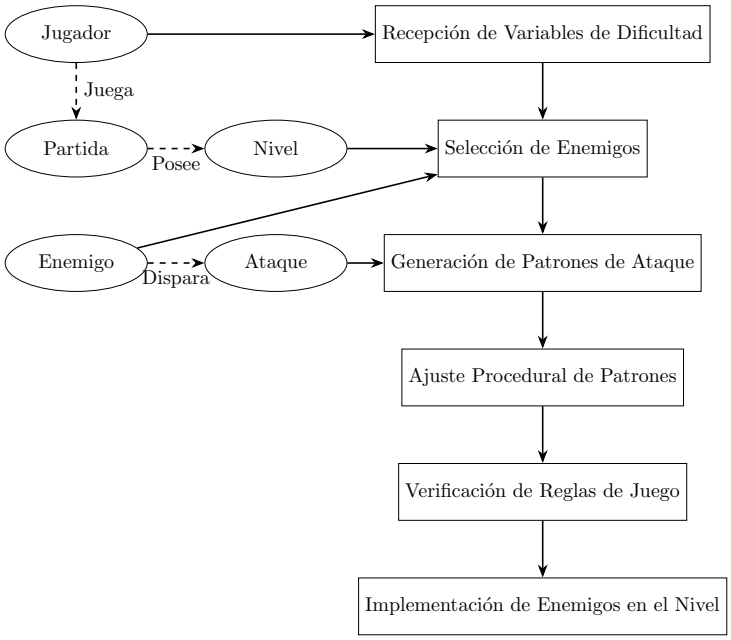

Figura 24: Vista Funcional

3.9.2. Vista de Comportamientos

Este diagrama destaca cómo el sistema de IA responde al rendimiento del jugador y ajusta dinámicamente tanto los enemigos como los patrones de ataque para mantener la jugabilidad equilibrada.

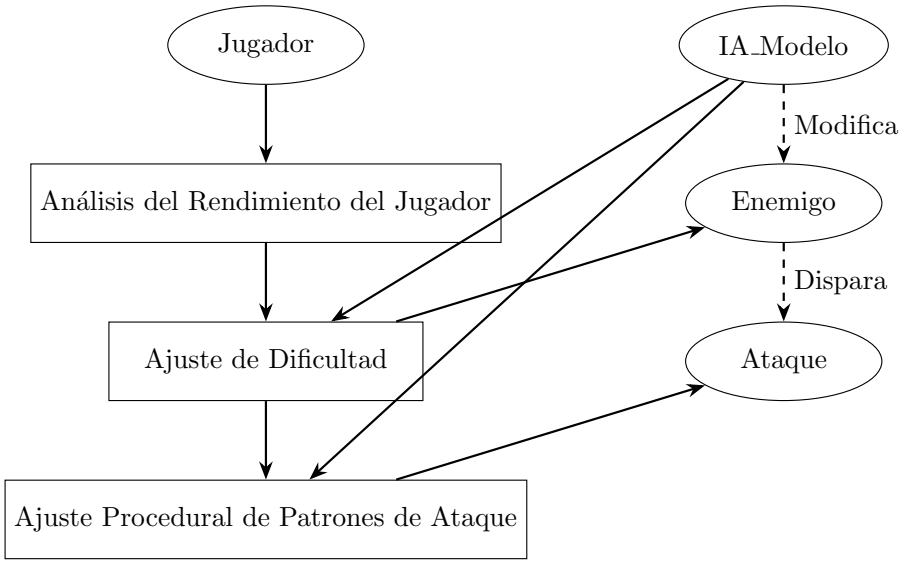

Figura 25: Vista de Comportamientos

3.9.3. Vista de Infraestructura

Este diagrama representa cómo se distribuye el sistema en la infraestructura y cómo interactúan los componentes físicos.

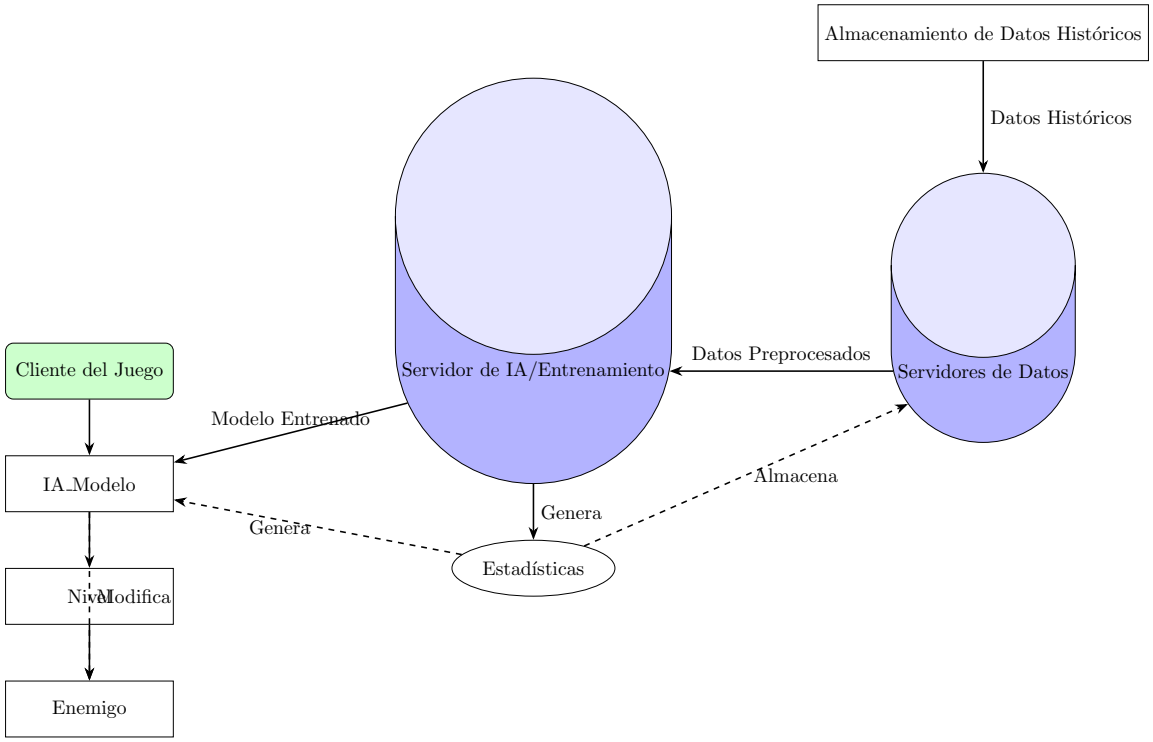

Figura 26: Vista de Infraestructura

3.9.4. Vista de Ejecución

Este diagrama muestra los pasos que sigue el sistema para el entrenamiento del modelo de IA, centrado en el flujo de trabajo y la interacción entre las diferentes entidades involucradas.

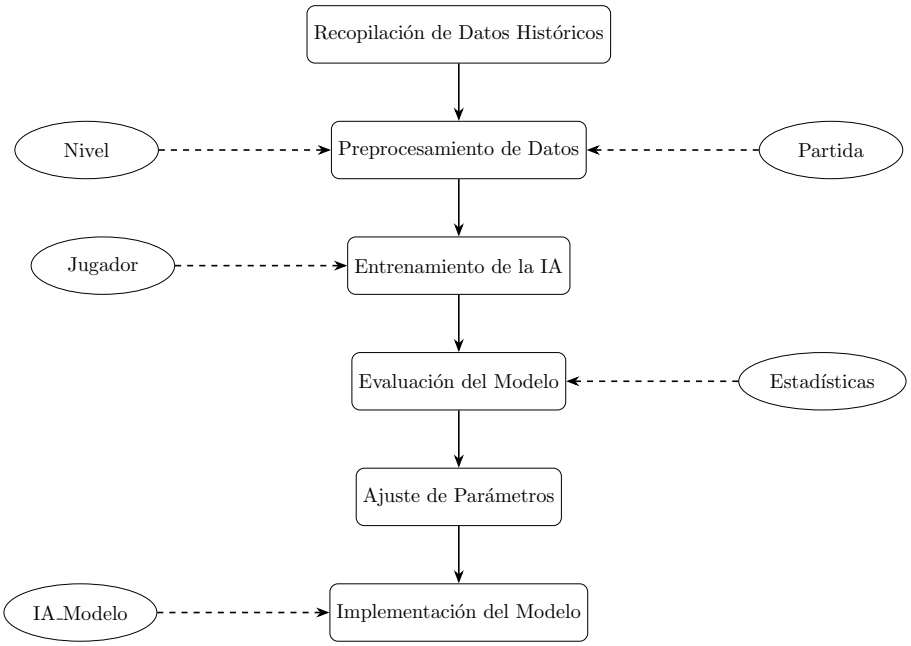

Figura 27: Vista de Ejecución

3.9.5. Vista de Integración

Este diagrama muestra cómo el proceso de Gestión de Enemigos y Patrones de Ataque se integra con otros procesos importantes dentro del sistema para garantizar una experiencia de juego cohesiva y adaptativa.

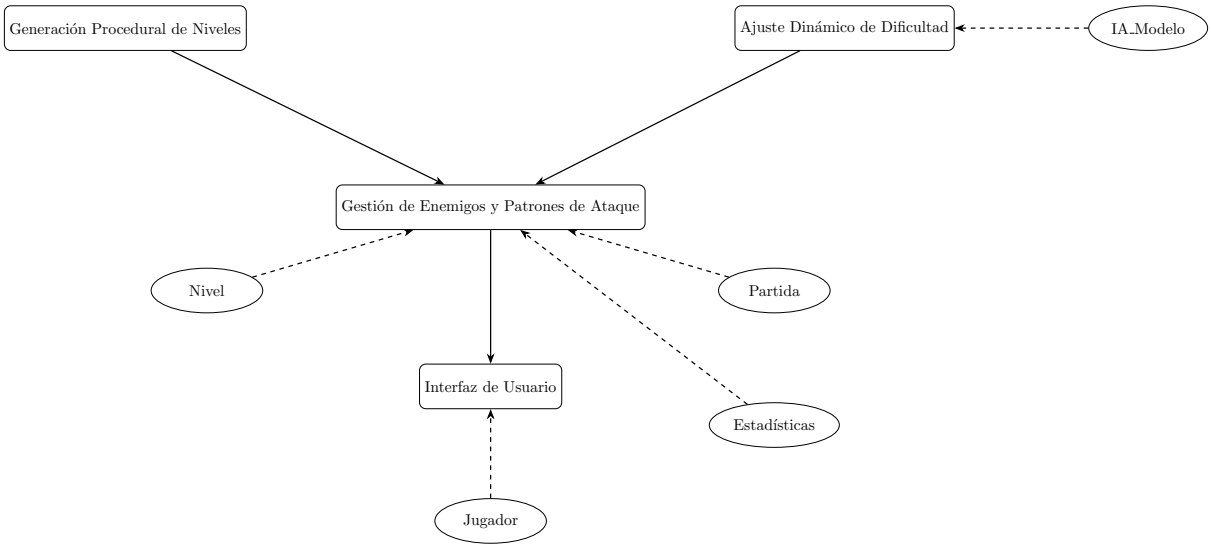

Figura 28: Vista de Integración

3.9.6. Vista de Verificación y Validación

Este diagrama ilustra el proceso de verificación de las reglas dentro del sistema de juego, para asegurarse de que los patrones de ataque y los enemigos seleccionados sean viables y cumplan con las normas de jugabilidad.

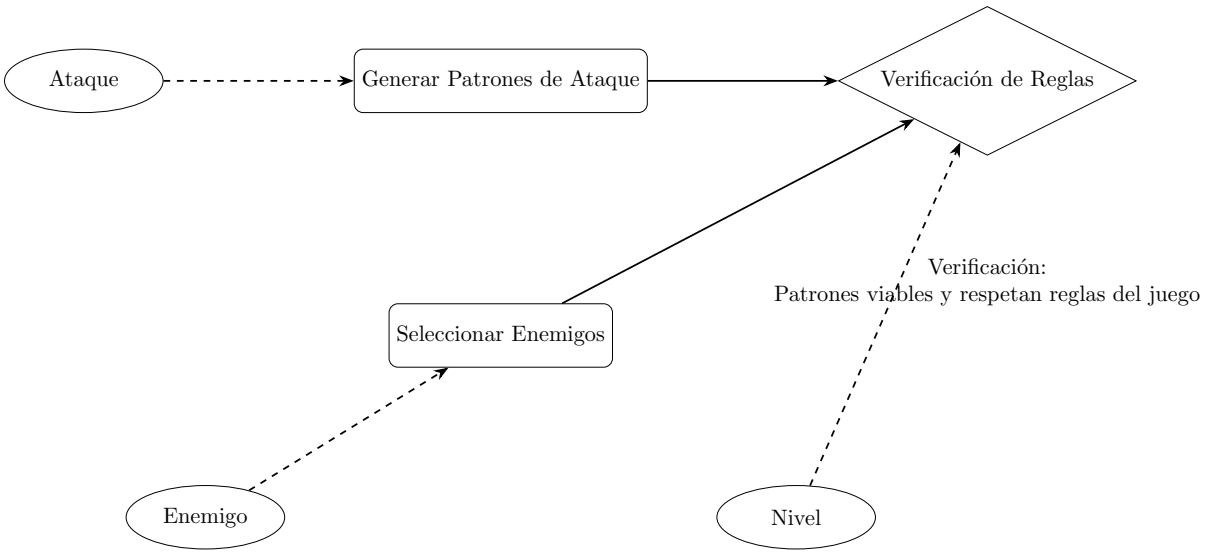

Figura 29: Vista de Verificación y Validación

3.9.7. Vista de Desempeño y Escalabilidad

Este diagrama representa la Vista de Desempeño o Escalabilidad del sistema, destacando los procesos clave que aseguran un rendimiento adecuado en tiempo real sin afectar la fluidez del juego.

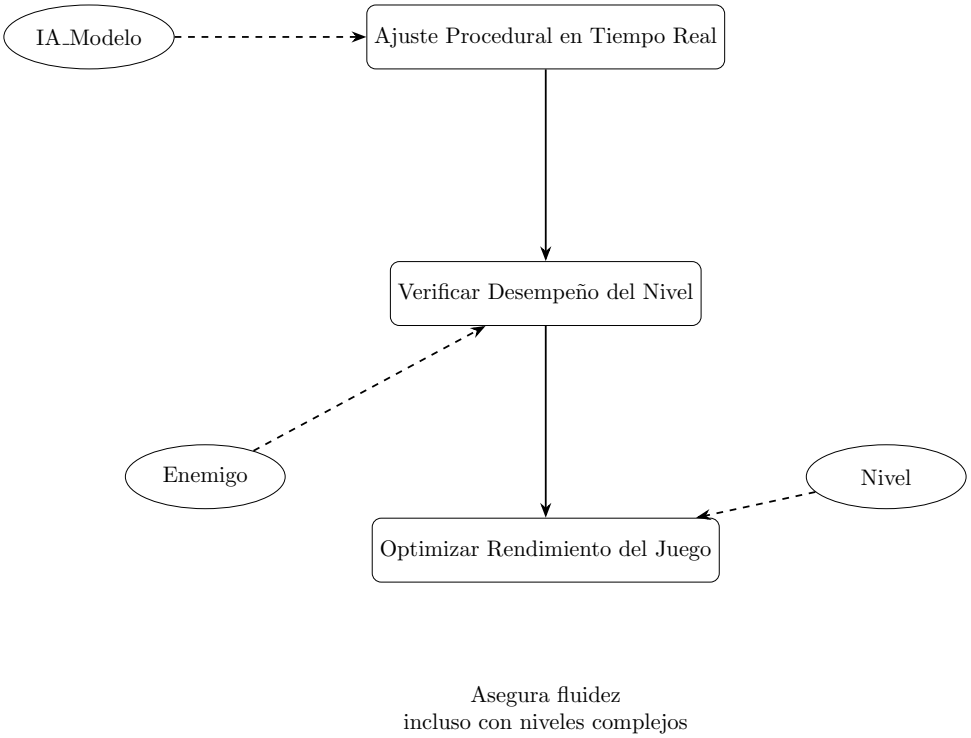

Figura 30: Vista de Desempeño y Escalabilidad

3.10. Registro y análisis de puntuación

3.10.1. Diagrama de Vistas del Registro y Análisis de Puntuación

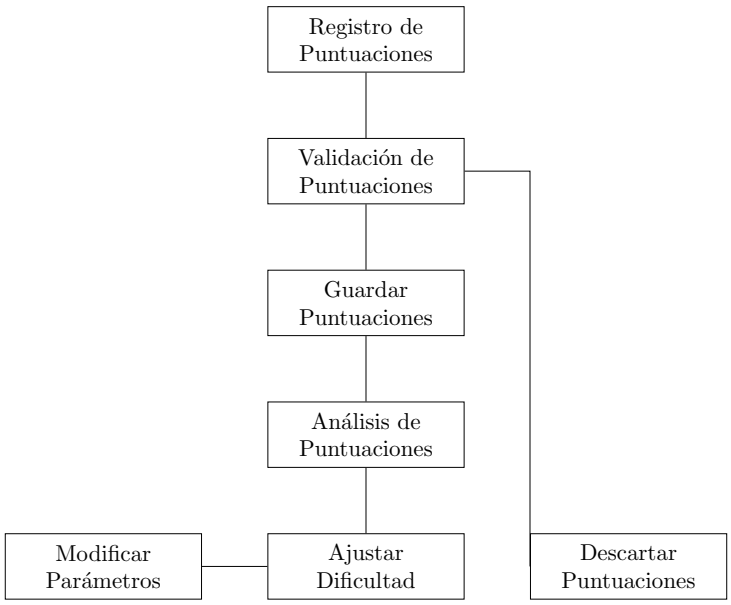

Figura 31: Vista de Desarrollo del Registro y Análisis de Puntuación

3.11. Optimización de rendimiento

3.11.1. Diagrama de Vistas de la Optimización del Rendimiento

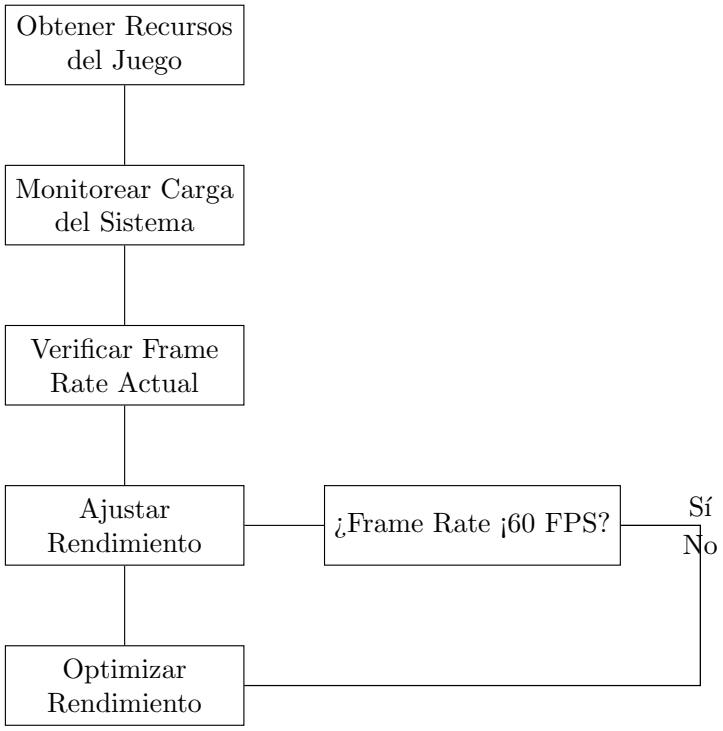

Figura 32: Vista de Desarrollo de la Optimización del Rendimiento

3.12. Escalabilidad del algoritmo

3.12.1. Diagrama de Vistas de la Escalabilidad del Algoritmo de IA

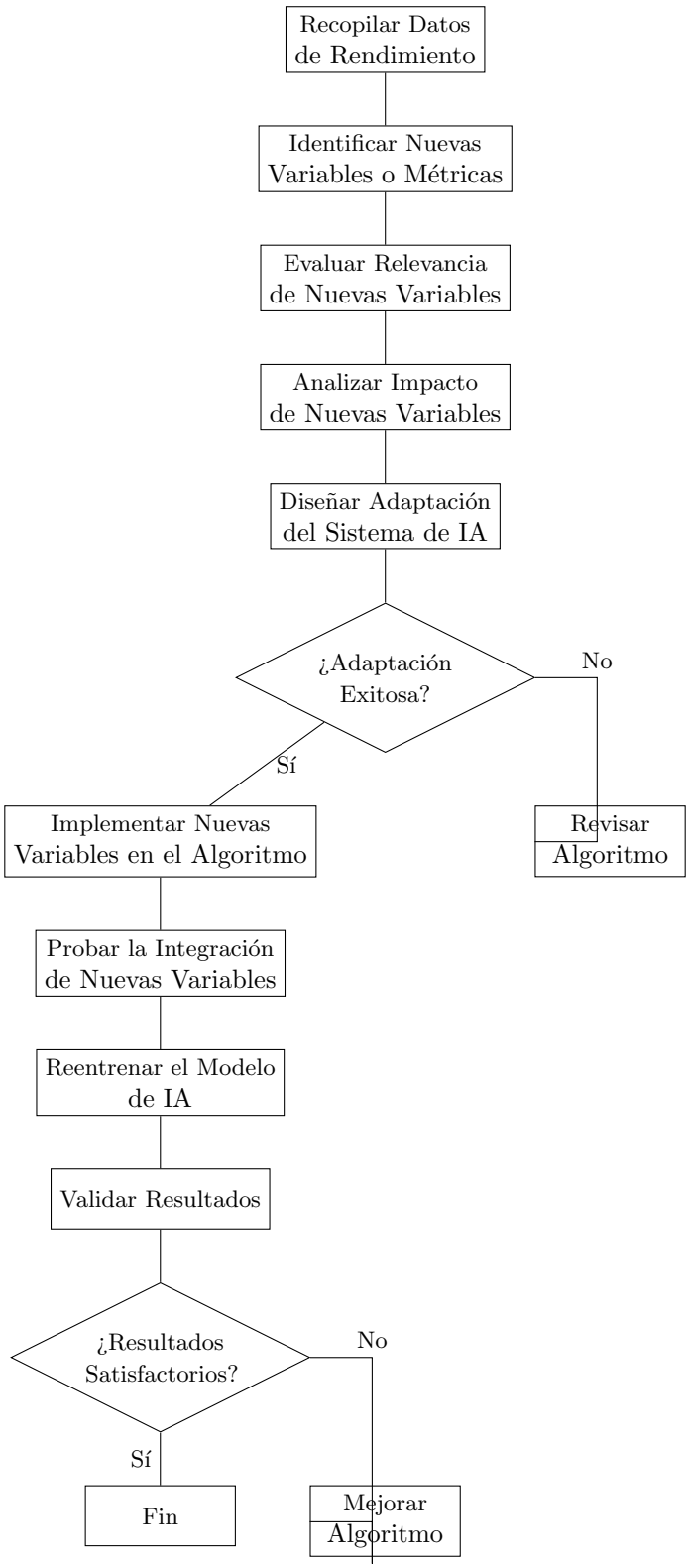

Figura 33: Vista de Desarrollo de la Escalabilidad del Algoritmo de IA

3.13. Ajuste Dinámico de la dificultad

3.13.1. Diagrama de Vistas del ajuste dinámico de la dificultad

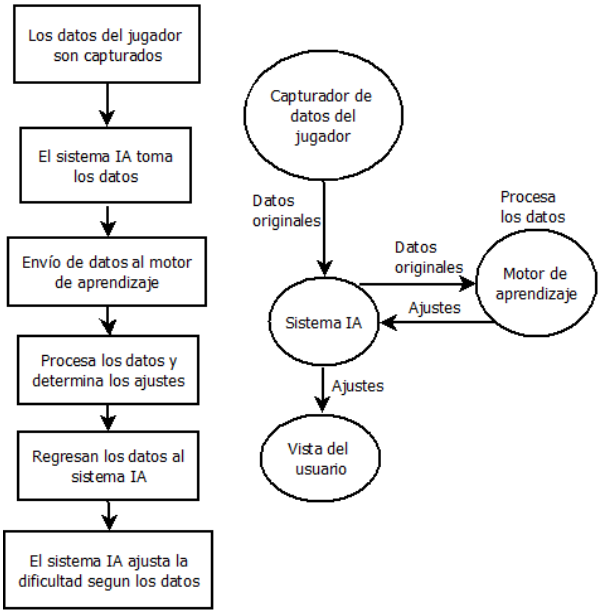

Figura 34: Diagrama de vistas del ajuste dinámico de la dificultad

3.14. Sistema de feedback visual y sonoro

3.14.1. Diagrama de Vistas del sistema de feedback visual y sonoro

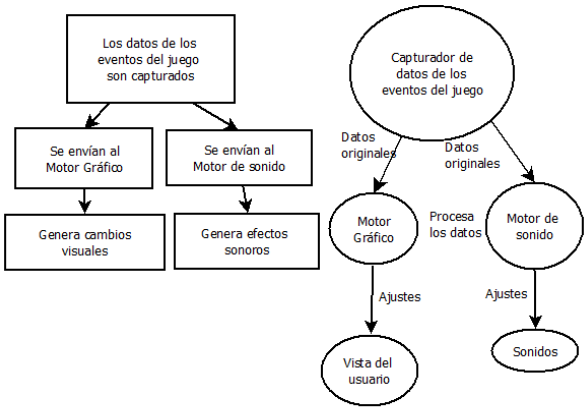

Figura 35: Diagrama de vistas del sistema de feedback visual y sonoro

3.15. Manejo de guardado de datos

3.15.1. Diagrama de Vistas del Manejo de guardado de datos

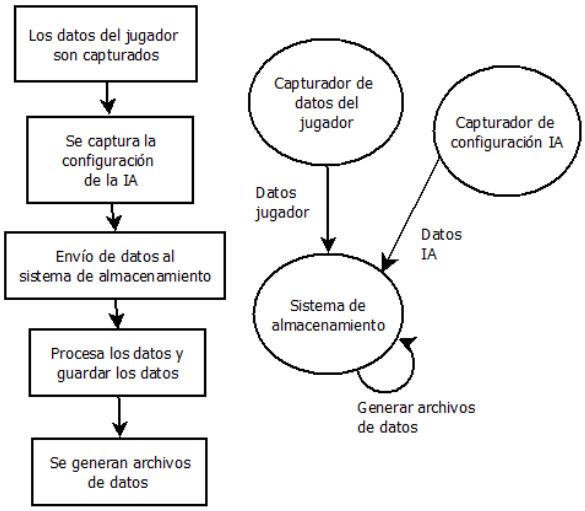

Figura 36: Diagrama de vistas del manejo de guardado de datos

4 Diseño del Sistema

4.1. Diagrama de Arquitectura.

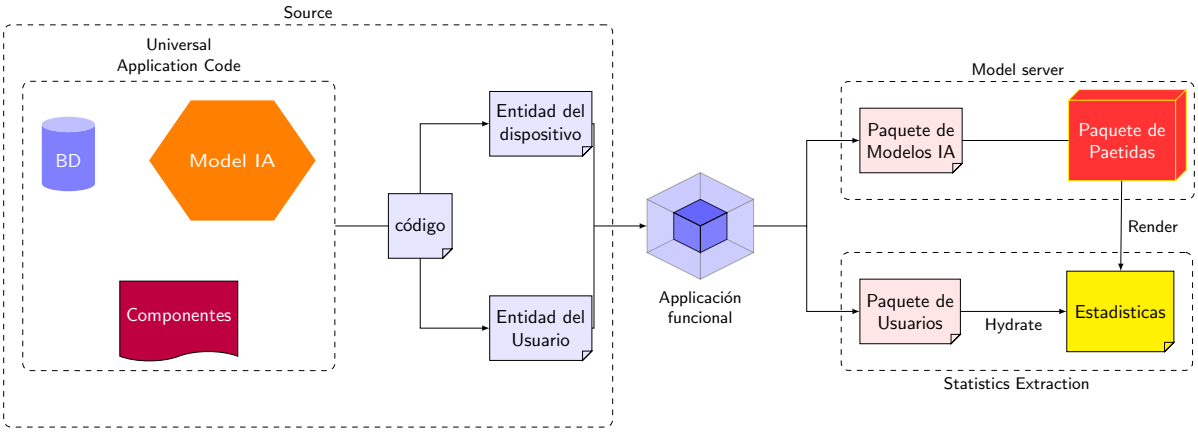

Figura 37: Diagrama de la arquitectura de la aplicación.

4.2. Diagrama de Repositorios/BD

4.2.1. Diagrama de Flujo de Información

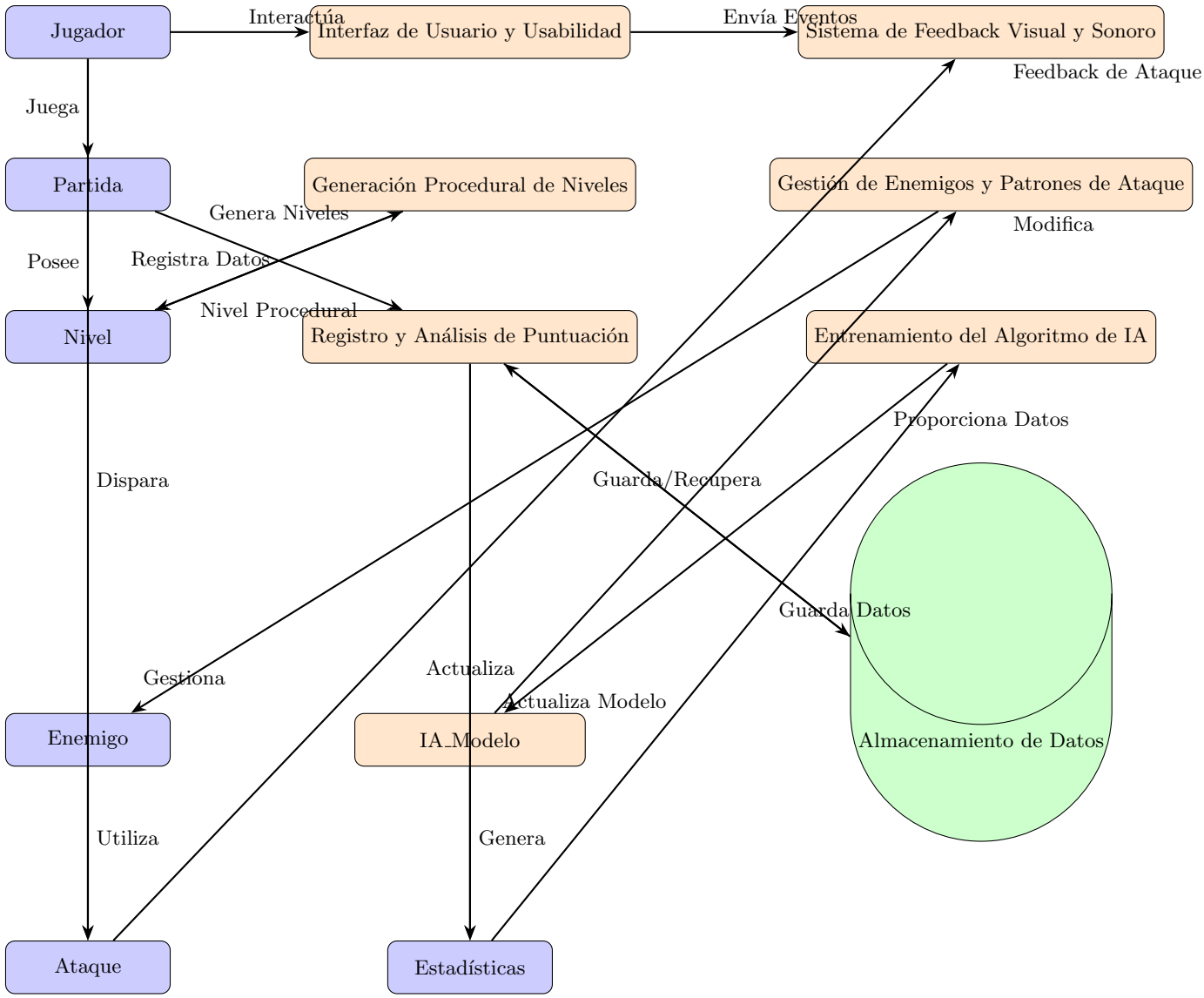

Figura 38: Diagrama de Flujo de Información del Proyecto

4.3. Diagrama de interfaces/Interfaces tentativas

4.4. Diagrama de interfaces

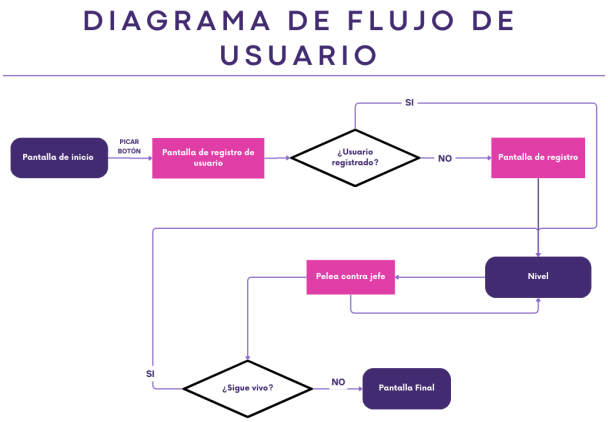

Figura 39: Diagrama de flujo de usuario

4.5. Posibles interfaces.

Figura 40: Primera pantalla

Figura 41: Ingreso de usuario

Figura 42: Registro de usuario

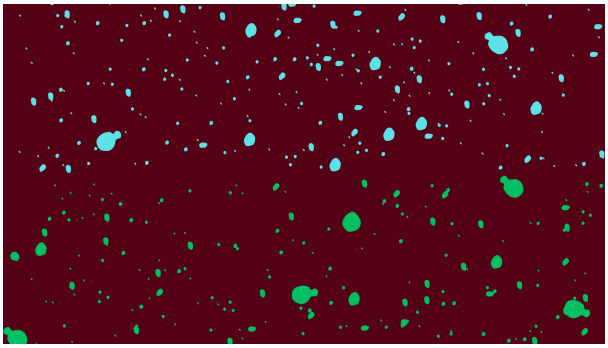

Figura 43: Posible fonde de nivel

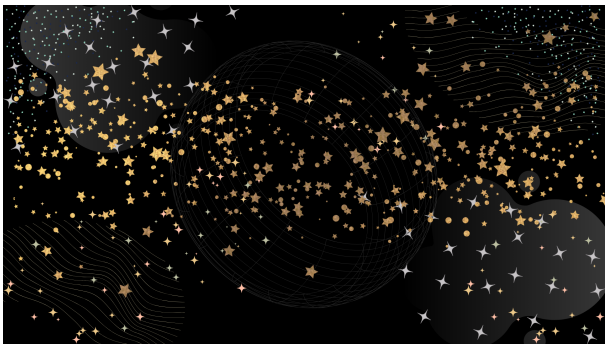

Figura 44: posible fondo de jefe

Figura 45: Final de partida

4.6. Pseudocodigos.

4.6.1. Algoritmos del proceso de *Generación procedural de niveles.*

Algoritmo de Generación Procedural de Entornos

Problemática: Crear entornos variados pero coherentes. Mantener un equilibrio entre la variedad y la jugabilidad puede ser un reto, ya que demasiada aleatoriedad puede generar niveles desequilibrados o imposibles de completar.

Posible solución: Utilización de algoritmos como *Perlin Noise* o *Simplex Noise* para generar terrenos o plataformas con una coherencia visual y de estructura. También se pueden emplear *gramáticas de espacio* para la disposición de ambientes más estructurados.

Desafío técnico: Mantener un balance entre la generación aleatoria y las reglas del diseño de niveles. Es necesario verificar que los niveles generados sean accesibles y jugables (sin zonas inalcanzables).

Algorithm 1: Inicializar Generador de Terrenos (e.g., Perlin Noise);

```
foreach sector del nivel do
    Definir parámetros de terreno (altura, bioma, etc.);
    Generar altura usando ruido (Perlin Noise);
    if bioma == 'agua' then
        Establecer zona como agua;
    end
    else if bioma == 'montaña' then
        Establecer elevación adicional;
    end
end
Verificar que no haya zonas inalcanzables;
foreach punto de inicio y final do
    if distancia(punto inicio, punto final) no es accesible then
        Ajustar terreno o generar puente/plataforma;
    end
end
return entorno generado
```

Algoritmo de Distribución de Obstáculos

Problemática: La colocación de obstáculos debe respetar reglas de accesibilidad, sin hacer el nivel demasiado difícil ni demasiado fácil.

Posible solución: Algoritmos de *optimización espacial* como *temple simulado (simulated annealing)* o algoritmos genéticos pueden ser útiles para iterar sobre las colocaciones de obstáculos y ajustar la dificultad, evitando repeticiones no deseadas y respetando restricciones de espacio.

Desafío técnico: La necesidad de verificar constantemente que la disposición de los obstáculos no bloquee el progreso del jugador y que ofrezca un reto adecuado.

Algoritmo de Distribución de Enemigos

Problemática: Colocar enemigos de manera estratégica en el nivel, ajustando su comportamiento según la dificultad del jugador, puede volverse complicado en niveles más avanzados o con enemigos complejos.

Posible solución: Usar una *red neuronal o aprendizaje profundo* para ajustar dinámicamente la colocación y el comportamiento de los enemigos basándose en el estilo de juego del usuario (un enfoque de

Algorithm 2: Inicializar Lista de Obstáculos (paredes, trampas, bloques)

```
; Definir la cantidad máxima de obstáculos según la dificultad;
foreach sector del nivel do
    Seleccionar una ubicación aleatoria;
    Verificar que la ubicación sea accesible;
    Colocar obstáculo;
    Evaluar balance de dificultad;
    if dificultad ¿límite superior then
        Eliminar obstáculos o reubicar;
    end
end
Iterar hasta lograr una distribución óptima usando algoritmo de temple simulado;
Ajustar distribución según las restricciones (accesibilidad, dificultad);
return obstáculos distribuidos
```

dificultad adaptativa). Alternativamente, un enfoque de *clusterización jerárquica* podría agrupar enemigos según el espacio disponible o su dificultad para asegurarse de que no se concentren en un solo punto.

Desafío técnico: El cálculo eficiente y adaptativo del comportamiento enemigo en tiempo real, asegurando que los enemigos estén bien distribuidos pero sin predecir el comportamiento del jugador con demasiada precisión, lo que podría hacer el juego frustrante o repetitivo.

Algorithm 3: Inicializar Lista de Enemigos disponibles

```
; Definir cantidad y tipo de enemigos según la dificultad del jugador;
foreach sector del nivel do
    Seleccionar una posición estratégica aleatoria;
    Colocar enemigo en la posición;
end
Ajustar el comportamiento enemigo según la dificultad del jugador;
if jugador es avanzado then
    Aumentar agresividad y velocidad;
end
else if jugador es novato then
    Reducir agresividad y añadir enemigos más fáciles;
end
Verificar que los enemigos estén bien distribuidos;
if enemigos están demasiado cerca o concentrados then
    Reubicar algunos de ellos;
end
return nivel con enemigos colocados;
```

Algoritmo de Ajuste Dinámico de Dificultad

Problemática: Ajustar la dificultad en función del rendimiento del jugador sin hacer el juego ni demasiado fácil ni demasiado difícil.

Posible solución: Se puede usar un sistema de *aprendizaje reforzado* que ajuste los parámetros del juego (como la cantidad de enemigos, su agresividad o la cantidad de obstáculos) en función del comportamiento del jugador. Este sistema podría aprender del jugador en tiempo real y ajustar las condiciones del nivel en consecuencia.

Desafío técnico: El ajuste de dificultad debe realizarse de manera sutil para que el jugador no perciba cambios drásticos, lo que podría afectar la inmersión. También es crucial evitar la creación de "picos" de dificultad que hagan el juego injustamente complicado.

Algorithm 4: Inicializar parámetros de dificultad (agresividad, cantidad de enemigos, velocidad, etc.).

Monitorear el desempeño del jugador (e.g., precisión, tiempo de reacción);
foreach *nivel generado* **do**
  Ajustar dificultad dinámicamente;
  if *jugador progresa muy rápido* **then**
    Incrementar cantidad de enemigos o agresividad;
  end
  else if *jugador tiene dificultades* **then**
    Reducir la cantidad de enemigos o facilitar obstáculos;
  end
end
Aplicar los ajustes al nivel en tiempo real;
return *nivel ajustado con la dificultad adaptada*;

Algoritmo de Verificación de Reglas

Problemática: Asegurar que el nivel generado sea jugable y cumpla con las reglas establecidas (accesibilidad, distribución de enemigos y obstáculos, etc.).

Posible solución: Un *algoritmo de búsqueda* como *A* (A-star)* o *algoritmos basados en grafos* para verificar que cada parte del nivel sea accesible y jugable. El algoritmo puede validar la disposición de los obstáculos y enemigos y confirmar que el jugador puede completar el nivel sin quedar atrapado o encontrarse en situaciones imposibles.

Desafío técnico: Verificar las reglas sin impactar negativamente en el rendimiento del juego, especialmente en niveles muy complejos o con una gran cantidad de elementos.

Algorithm 5: Algoritmo de verificación de Reglas

foreach *sector del nivel generado* **do**
  Generar un grafo representando el mapa;
  Ejecutar el algoritmo A* para verificar rutas accesibles;
  if *ruta no es accesible* **then**
    Ajustar el terreno o reubicar obstáculos;
  end
  Verificar la distribución de enemigos y obstáculos;
  if *enemigos u obstáculos bloquean el progreso* **then**
    Reubicar elementos;
  end
end
if *el nivel cumple con las reglas de jugabilidad* **then**
  return *"Verificación exitosa"*;
end
else
  return *"Ajustes necesarios"*;
end

Algoritmo de Generación de Entornos Jugables

Problemática: Asegurarse de que los niveles generados sean divertidos, equilibrados y no repetitivos, mientras se ajustan a las reglas de diseño establecidas.

Posible solución: *Algoritmos basados en evolución procedural* (PCG - Procedural Content Generation) que usan *gramáticas generativas* o métodos como *Markov Chains* para crear niveles con ciertas estructuras reconocibles, pero que aún mantienen variedad.

Desafío técnico: Evitar la monotonía, asegurando que el sistema pueda generar suficientes variaciones interesantes en los niveles para que el jugador no perciba repetición en la experiencia.

Algorithm 6: Generación Procedural de un Nivel Jugable

```
Inicializar gramática procedural para generar estructuras reconocibles;
foreach sector del nivel do
    Generar secciones del nivel utilizando gramáticas;
    Validar la jugabilidad de cada sección (rutas, obstáculos, enemigos);
end
Evaluar el balance de dificultad y variedad;
if sección es muy repetitiva then
    Cambiar parámetros para generar una variación;
end
Verificar accesibilidad;
if alguna sección no es accesible then
    Ajustar su generación;
end
return Entorno jugable generado;
```

4.6.2. Algoritmos del proceso de *Ajuste Dinamico de Dificultad.*

Algoritmo de Ajuste Dinámico de Dificultad

Problemática: Ajustar la dificultad en función del rendimiento del jugador sin hacer el juego ni demasiado fácil ni demasiado difícil.

Posible solución: Se puede usar un sistema de *aprendizaje reforzado* que ajuste los parámetros del juego (como la cantidad de enemigos, su agresividad o la cantidad de obstáculos) en función del comportamiento del jugador. Este sistema podría aprender del jugador en tiempo real y ajustar las condiciones del nivel en consecuencia.

Desafío técnico: El ajuste de dificultad debe realizarse de manera sutil para que el jugador no perciba cambios drásticos, lo que podría afectar la inmersión. También es crucial evitar la creación de "picos" de dificultad que hagan el juego injustamente complicado.

Entradas:

- Datos del jugador:
  - vidas_perdidas
  - precision
  - puntuacion

Salidas:

- Modificaciones de la dificultad:
  - numero_enemigos
  - velocidad_enemigos
  - patrones_enemigos

Actores:

- Sistema de IA
- Motor de aprendizaje

Pseudocódigo:

Algorithm 7: Ajuste Dinámico de la Dificultad del Juego

Input: vidas_perdidas, precision, puntuacion
Output: numero_enemigos, velocidad_enemigos, patrones_enemigos

```
// Inicialización de parámetros
dificultad_base ← 1.0;
incremento_dificultad ← 0.1;
vidas_max ← 3;
precision_esperada ← 0.75;
puntuacion_esperada ← 1000;
while el juego esté en ejecución do
    // Leer datos del jugador
    leer(vidas_perdidas, precision, puntuacion);
    if vidas_perdidas ¿0 then
        dificultad_base ← dificultad_base - vidas_perdidas / vidas_max end                    end
        if precision ¡precision_esperada then
            dificultad_base ← dificultad_base - incremento_dificultad;
        end
        if puntuacion ≥ puntuacion_esperada then
            dificultad_base ← dificultad_base + incremento_dificultad;
        end
        // Aplicar límites a la dificultad
        dificultad_base ← max(0.5, min(dificultad_base, 2.0));
        // Modificar parámetros de enemigos según la dificultad
        numero_enemigos ← base_enemigos * dificultad_base;
        velocidad_enemigos ← base_velocidad * dificultad_base;
        patrones_enemigos ← seleccionar_patrones(dificultad_base);
        // Enviar parámetros ajustados al motor de juego
        enviar_parametros(numero_enemigos, velocidad_enemigos, patrones_enemigos);
        // Actualizar datos del jugador con retroalimentación del sistema IA y motor
           de aprendizaje
        actualizar_datos_jugador();
```

Explicación:

El sistema ajusta dinámicamente la dificultad del juego basándose en el rendimiento del jugador. Si el jugador pierde vidas o su precisión cae por debajo del umbral esperado, la dificultad se reduce. Si la puntuación supera la esperada, la dificultad aumenta. El ajuste se aplica a los enemigos, modificando su número, velocidad y patrones. El motor de aprendizaje recibe retroalimentación para mejorar la adaptación del sistema a largo plazo.

4.6.3. Algoritmos del proceso del *modelo de IA*

Algoritmo de Validación de Datos Históricos (Vista Lógica y Física)

Problemática: Verificar que los datos de las partidas sean válidos y consistentes antes de usarlos en el modelo de IA.

Algorithm 8: Validación de Datos Históricos

Input: Datos_Históricos
foreach *partida en Datos_Históricos* **do**
  if *partida.puntuacion ¡0* ***or*** *partida.precision ¡0* ***or*** *partida.tiempo_jugado ¡= 0* **then**
    Descartar partida;
  end
  if *partida.vidas_perdidas ¡0* ***or*** *partida.enemigos_eliminados ¡0* **then**
    Descartar partida;
  end
  if *datos inconsistentes encontrados* **then**
    Notificar error y corregir;
  end
end
return *Datos_Históricos_Validados*

Algoritmo de Preprocesamiento de Datos.

Problemática: Preparar los datos recopilados para ser utilizados en el entrenamiento del modelo de IA, mediante normalización y generación de nuevas características.

Algorithm 9: Preprocesamiento de Datos

Input: Datos_Históricos_Validados
Datos_Normalizados = Normalizar(Datos_Históricos_Validados);
Caracteristicas_Adicionales = Generar_Caracteristicas(Datos_Normalizados);
foreach *partida en Caracteristicas_Adicionales* **do**
  tendencia = Calcular_Tendencia(partida.puntuacion, partida.precision, partida.vidas_perdidas);
  Agregar tendencia a partida;
end
return *Datos_Preprocesados*

Algorithm 10: Normalización de Datos

Input: Datos
foreach *dato en Datos* **do**
  dato.normalizado = $\frac{dato.valor - min(dato)}{max(dato) - min(dato)}$;
end
return *Datos_Normalizados*

Algorithm 11: Generación de Características

Input: Datos
Crear nuevas características basadas en estadísticas (p. ej., mejora del rendimiento del jugador en cada partida);
return *Datos con características adicionales*

Algoritmo de Entrenamiento del Modelo.

Problemática: Entrenar un modelo de red neuronal basado en los datos preprocesados, ajustando los pesos mediante retropropagación.

Algorithm 12: Entrenamiento del Modelo

```
Input: Datos_Preprocesados
Inicializar modelo con pesos aleatorios;
foreach época do
    foreach entrada en Datos_Preprocesados do
        salida_predicha = Propagar_Adelante(entrada, modelo);
        error = Calcular_Error(salida_predicha, entrada.objetivo);
        Ajustar_Pesos(error, modelo);
        Retropropagar(error, modelo);
    end
end
return Modelo_Entrenado
```

Algorithm 13: Propagación Adelante

```
Input: entrada, modelo
foreach capa en modelo do
    entrada = Activacion(Suma_Pesos(entrada, capa.pesos));
end
return entrada como salida_predicha
```

Algorithm 14: Ajuste de Pesos

```
Input: error, modelo
foreach capa en modelo do
    Actualizar pesos basándose en error y tasa de aprendizaje;
end
```

Algorithm 15: Retropropagación

```
Input: error, modelo
Propagar el error hacia atrás ajustando pesos de capas anteriores;
```

Algoritmo de Evaluación del Modelo.

Problemática: Evaluar el rendimiento del modelo entrenado utilizando métricas clave.

Algorithm 16: Evaluación del Modelo

```
Input: Modelo_Entrenado, Datos_Validacion
métrica_precision = 0;
métrica_tiempo_respuesta = 0;
foreach entrada en Datos_Validacion do
    predicción = Modelo_Entrenado.Propagar_Adelante(entrada);
    métrica_precision += Comparar_Prediccion(predicción, entrada.objetivo);
    métrica_tiempo_respuesta += Calcular_Tiempo_Respuesta(predicción);
end
Calcular_Métricas_Finales(métrica_precision, métrica_tiempo_respuesta);
return Métricas
```

Algorithm 17: Comparación de Predicción

```
Input: predicción, objetivo
if predicción es correcta then
    return 1;
end
else
    return 0;
end
```

Algorithm 18: Cálculo del Tiempo de Respuesta

Input: predicción
Medir tiempo que tomó la predicción;
return *tiempo*

Algoritmo de Ajuste de Parámetros.

Problemática: Ajustar los parámetros del modelo si el rendimiento es subóptimo según las métricas evaluadas.

Algorithm 19: Ajuste de Parámetros del Modelo

Input: Modelo_Entrenado, Métricas
if *Métricas.precision ¡umbral* ***or*** *Métricas.tiempo_respuesta ¿limite* **then**
    Ajustar tasa_de_aprendizaje en Modelo_Entrenado;
    Aumentar número de iteraciones si es necesario;
    Reentrenar el modelo con los nuevos parámetros;
end
return *Modelo_Ajustado*

Algoritmo de Implementación del Modelo Actualizado.

Problemática: Integrar el modelo actualizado en el sistema de juego para ajustar la dificultad en tiempo real.

Algorithm 20: Implementación del Modelo

Input: Modelo_Ajustado
Sistema_Juego.Cargar_Modelo(Modelo_Ajustado);
while *juego esté activo* **do**
    Datos_Actuales = Recopilar_Datos_Juego();
    dificultad_ajustada = Modelo_Ajustado.Prediccion(Datos_Actuales);
    Sistema_Juego.Ajustar_Dificultad(dificultad_ajustada);
end

4.6.4. Algoritmos del proceso de *Retroalimentacion Visual y Sonora en Tiempo Real.*

Problemática

En muchos juegos, proporcionar retroalimentación inmediata al jugador es crucial para mejorar la inmersión y la experiencia de juego. Sin embargo, gestionar esta retroalimentación visual y sonora en tiempo real puede ser un desafío, especialmente cuando se debe reaccionar dinámicamente a varios eventos de juego, tales como:

- Derrotas.
- Cambios en la dificultad.
- Logros alcanzados.
- Avances en niveles.

Los principales problemas son:

- Desincronización entre los eventos del juego y la retroalimentación visual o sonora.
- Falta de personalización de la retroalimentación basada en la complejidad de los eventos.
- Sobrecarga de recursos si la retroalimentación no se gestiona eficientemente, lo que puede provocar retrasos o bloqueos.

Posible Solución

El algoritmo de retroalimentación visual y sonora en tiempo real tiene como objetivo sincronizar los eventos del juego con respuestas visuales (sprites) y sonoras (efectos) adecuadas. La solución consiste en:

- **Detectar eventos clave en el juego:** Lectura de eventos durante la ejecución, como derrotas, cambios en la dificultad, logros y avances.
- **Proporcionar retroalimentación visual:** Mostrar sprites en ubicaciones apropiadas, como explosiones al perder o íconos cuando aumenta la dificultad.
- **Proporcionar retroalimentación sonora:** Reproducir sonidos que refuercen la experiencia del jugador, como sonidos para logros o cambios de dificultad.
- **Sincronización con el motor de juego:** Asegurar que los sprites y efectos sonoros se envíen al motor gráfico y de sonido sin latencia perceptible.

Desafío Técnico

Implementar esta solución de retroalimentación visual y sonora en tiempo real presenta varios desafíos técnicos, incluyendo:

Sincronización en tiempo real El sistema debe reaccionar de forma instantánea a los eventos. Cualquier retraso en la detección de los eventos o en la representación visual/sonora afectaría negativamente la experiencia del jugador. El algoritmo debe ser lo suficientemente eficiente para evitar un lag perceptible.

Gestión de múltiples eventos simultáneos El sistema debe ser capaz de manejar varios eventos ocurriendo al mismo tiempo. Por ejemplo, si un jugador pierde una vida mientras logra un objetivo, el sistema debe mostrar tanto la derrota como el logro sin sobrecargar el motor gráfico o de sonido.

Optimización de recursos Para evitar la sobrecarga del sistema, es necesario gestionar cuidadosamente las llamadas al motor gráfico y de sonido. Esto incluye evitar redundancias en los efectos de sonido y la superposición innecesaria de sprites.

Escalabilidad A medida que el juego se vuelve más complejo, con más eventos posibles, el sistema debe poder escalar sin comprometer el rendimiento. El motor de retroalimentación debe ser capaz de manejar desde juegos simples hasta títulos más complejos sin perder fluidez.

Entradas:

- Eventos del juego:
  - derrotas
  - cambios_dificultad
  - logros
  - avances

Salidas:

- Indicaciones visuales y sonoras:
  - sprites
  - efectos_sonido

Actores:

- Motor gráfico
- Motor de sonido

Pseudocódigo:

Algorithm 21: Retroalimentación Visual y Sonora en Tiempo Real

Input: derrotas, cambios_dificultad, logros, avances
Output: sprites, efectos_sonoros
while *el juego esté en ejecución* **do**
    `// Leer eventos del juego`
    leer(eventos);
    if *evento == derrota* **then**
        `// Retroalimentación para derrotas`
        mostrar_sprite(.ºexplosion", posicion_jugador);
        reproducir_sonido("derrota");
    end
    if *evento == cambios_dificultad* **then**
        if *dificultad aumenta* **then**
            `// Retroalimentación para aumento de dificultad`
            mostrar_sprite(ïcono_dificultad_up", pantalla);
            reproducir_sonido("dificultad_aumenta");
        end
        if *dificultad disminuye* **then**
            `// Retroalimentación para disminución de dificultad`
            mostrar_sprite(ïcono_dificultad_down", pantalla);
            reproducir_sonido("dificultad_disminuye");
        end
    end
    if *evento == logro* **then**
        `// Retroalimentación para logros`
        mostrar_sprite(ïcono_logro", centro_pantalla);
        reproducir_sonido("logro_desbloqueado");
    end
    if *evento == avance* **then**
        `// Retroalimentación para avances en el juego`
        mostrar_sprite(ïndicador_avance", interfaz);
        reproducir_sonido(.ªvanzar_nivel");
    end
    `// Enviar sprites y efectos sonoros al motor gráfico y de sonido`
    enviar_sprites_y_sonidos(motor_grafico, motor_sonido);
end

Explicación:

Este pseudocódigo gestiona los eventos del juego como derrotas, cambios en la dificultad, logros y avances. Dependiendo del evento, se generan indicaciones visuales (sprites) y sonoras (efectos de sonido) que son enviadas al motor gráfico y al motor de sonido. Por ejemplo, al perder una vida, se muestra un sprite de explosión y se reproduce un sonido específico de derrota. De igual manera, si hay un cambio en la dificultad o se alcanza un logro, la retroalimentación visual y sonora se adapta.

4.6.5. Algoritmos del proceso d*Gestión de enemigos y patrón de ataques*

Algoritmo de Selección de Enemigos y Patrones de Ataque

Problemática: Seleccionar enemigos adecuados y generar patrones de ataque en tiempo real basados en variables de dificultad.
Algoritmo: Se puede usar una *heurística de selección* combinada con un *algoritmo de búsqueda* como *algoritmo genético*. Los enemigos y sus patrones de ataque se seleccionan según la dificultad, iterando sobre un conjunto de posibles combinaciones y eligiendo las más optimizadas según el nivel actual.
Desafío: Adaptarse en tiempo real a los cambios en la dificultad y garantizar que los patrones generados no sean repetitivos ni imposibles.

Algorithm 22: Algoritmo Genético para Selección de Enemigos y Patrones de Ataque

Inicializar población con N configuraciones de enemigos y patrones;
foreach *configuración en la población* **do**
  Evaluar cada configuración en base a:
    - Dificultad deseada
    - Diversidad de patrones
    - Repetición de ataques

end
for *cada iteración hasta el número máximo de generaciones* **do**
  Seleccionar configuraciones más óptimas (por ejemplo, mediante torneo);
  Cruzar las configuraciones seleccionadas para generar nuevas combinaciones;
  Mutar aleatoriamente algunos patrones en las nuevas configuraciones;
  Evaluar nuevas configuraciones;
  if *se encuentra una configuración óptima o se alcanzan las generaciones máximas* **then**
    Salir del bucle;
  end
end
return *configuración más adecuada para la situación actual del juego*;

Algoritmo de Ajuste Dinámico de Ataques

Problemática: Ajustar el comportamiento de los enemigos y sus patrones de ataque en función del rendimiento del jugador en tiempo real.
Algoritmo: El uso de un *sistema basado en aprendizaje reforzado* o un *modelo predictivo* que ajuste patrones de ataque dinámicamente, aprendiendo del estilo de juego del usuario y modificando la dificultad de manera progresiva.
Desafío: Mantener una transición suave en los ajustes, evitando cambios bruscos que puedan desbalancear la jugabilidad.

Algorithm 23: Ajuste dinámico de ataques basado en Aprendizaje Reforzado

Inicializar un modelo de Aprendizaje Reforzado (Q-learning o similar);
Definir estados (niveles de rendimiento del jugador);
Definir acciones (ajustes en patrones de ataque: más agresivo, más defensivo, etc.);
Definir recompensas (basado en si el jugador mejora o empeora);
while *el juego esté en progreso* **do**
  Observar el rendimiento del jugador;
  Determinar el estado actual del jugador;
  Elegir la mejor acción basada en el modelo actual (exploración vs explotación);
  Ejecutar el ajuste de patrones de ataque;
  Observar el resultado (mejora o empeora la experiencia del jugador);
  Actualizar el modelo de aprendizaje reforzado basado en la recompensa obtenida;

end
Actualizar patrones de ataque dinámicamente según la acción seleccionada;

Algoritmo de Gestión de Variables de Dificultad

Problemática: Recopilar y procesar datos como estadísticas del jugador y variables de dificultad para generar ataques adecuados.
Algoritmo: Un *algoritmo de clasificación jerárquica* podría agrupar los datos del jugador (rendimiento, estadísticas) y decidir cómo influir en los parámetros de ataque. Por ejemplo, clasificar los jugadores en diferentes niveles de habilidad para ajustar la agresividad de los enemigos.
Desafío: Procesar los datos en tiempo real sin sobrecargar el sistema o causar retrasos en la jugabilidad.

Algorithm 24: Algoritmo de Clasificación Jerárquica para Variables de Dificultad

```
Inicializar datos de rendimiento del jugador (estadísticas como precisión, tiempo de reacción, etc.);
Agrupar jugadores en diferentes niveles de habilidad mediante Clasificación Jerárquica;
while el juego esté en progreso do
    Recoger datos de rendimiento del jugador en tiempo real;
    Actualizar la clasificación del jugador si es necesario;
    if el jugador pertenece al grupo de nivel bajo then
        Generar patrones de ataque menos agresivos;
    else if el jugador pertenece al grupo de nivel medio then
        Generar patrones balanceados;
    else if el jugador pertenece al grupo de nivel alto then
        Generar patrones más agresivos;
    end
end
Actualizar constantemente los ataques según la nueva clasificación;
```

Algoritmo de Control de Carga

Problemática: Garantizar que el ajuste dinámico y generación de enemigos/patrones de ataque no afecte negativamente el rendimiento del juego.
Algoritmo: Se puede implementar un *algoritmo de control de rendimiento* basado en la cantidad de enemigos y la complejidad del patrón de ataque. Técnicas como el *cálculo incremental* o el *ajuste procedural limitado* pueden ayudar a mantener el equilibrio.
Desafío: Lograr un rendimiento estable incluso en niveles muy complejos, con múltiples enemigos y patrones de ataque en pantalla.

Algorithm 25: Control de Carga para Manejo de Enemigos y Patrones

```
Definir límite máximo de enemigos y complejidad de patrones basado en los recursos del sistema;
Inicializar la cantidad de enemigos y la complejidad de patrones a un nivel básico;
while el juego esté en progreso do
    Medir uso de CPU y GPU del sistema;
    if el uso del sistema es bajo then
        Aumentar la cantidad de enemigos y la complejidad de los patrones;
    else if el uso del sistema está cerca del límite then
        Reducir la cantidad de enemigos o simplificar los patrones de ataque;
end
Ajustar dinámicamente la cantidad de enemigos y la complejidad según el rendimiento del sistema;
```

Algoritmo de Sincronización de Módulos

Problemática: Integrar la gestión de enemigos con otros sistemas como el motor procedural de niveles o el ajuste de dificultad adaptativa.
Algoritmo: Un *algoritmo de sincronización* basado en eventos podría coordinar las funciones de los diferentes módulos. Por ejemplo, cuando el nivel se genera, el sistema de enemigos debe ser notificado para colocar adecuadamente los enemigos.

Desafío: Asegurarse de que la comunicación entre los sistemas sea eficiente y sincrónica, sin causar cuellos de botella en el rendimiento.

Algorithm 26: Algoritmo de Sincronización de Módulos basado en eventos

when el nivel se genera **do** Notificar al sistema de gestión de enemigos;
Obtener la estructura del nivel generado;
foreach *región del nivel* **do**
  Asignar enemigos y patrones de ataque adecuados;
end
Sincronizar el sistema de IA con el entorno generado;
Notificar al sistema de ajuste dinámico de dificultad;
when los parámetros del jugador cambien **do** Ajustar los patrones de ataque en consecuencia;
Actualizar otros módulos según los nuevos parámetros (dificultad, enemigos);
Coordinar la comunicación entre módulos para asegurar consistencia;

Algoritmo de Verificación de Jugabilidad

Problemática: Asegurar que los patrones de ataque y enemigos generados sean jugables y no rompan las reglas del diseño del juego.
Algoritmo: Un *algoritmo de búsqueda A** o un *algoritmo basado en grafos* puede verificar la accesibilidad y jugabilidad de los patrones de ataque, asegurándose de que el jugador tenga suficiente espacio para esquivar y que los enemigos estén distribuidos de manera justa.
Desafío: Verificar todas las combinaciones de enemigos y patrones sin causar problemas de rendimiento.

4.6.6. Algoritmos del proceso de *Gyardar y Cargar datos y Configuraciones de IA*

Problemática

En los videojuegos que utilizan inteligencia artificial (IA) para ajustar la dificultad o adaptar la jugabilidad, es fundamental poder guardar el estado del jugador y las configuraciones de la IA para continuar la partida en el futuro. Si los datos del jugador y la IA no se guardan correctamente, se pierde todo el progreso realizado y la IA no puede ajustar adecuadamente su comportamiento en función de las partidas anteriores. Los principales problemas son:

- Pérdida de datos críticos del jugador o de la configuración de la IA.
- Incompatibilidad entre versiones de los datos guardados.
- Desincronización entre el progreso del jugador y el estado de la IA.

Posible Solución

El algoritmo de guardado y carga de datos del proceso y configuraciones de IA debe:

- **Guardar datos críticos del jugador:** Registrar el estado actual del jugador, como su puntuación, nivel alcanzado, vidas restantes y cualquier otro progreso relevante.
- **Guardar configuraciones de la IA:** Almacenar las configuraciones actuales de la IA, incluyendo los ajustes dinámicos realizados en la dificultad o patrones de comportamiento.
- **Cargar datos en futuras partidas:** Al iniciar una nueva sesión de juego, el sistema debe ser capaz de restaurar tanto el progreso del jugador como la configuración de la IA, para continuar el juego en el mismo estado.
- **Sincronización con el sistema de almacenamiento:** El algoritmo debe interactuar con el sistema de almacenamiento del juego para leer y escribir datos sin errores.

Desafío Técnico

Implementar un sistema eficaz para guardar y cargar datos del proceso y configuraciones de IA presenta varios desafíos técnicos:

Integridad de los datos Es esencial que el sistema de guardado y carga mantenga la integridad de los datos. Cualquier corrupción en los archivos de guardado o inconsistencia entre los datos del jugador y las configuraciones de la IA puede llevar a comportamientos inesperados o la pérdida de progreso.

Compatibilidad de versiones Cuando se actualiza un juego, es posible que la estructura de los datos guardados cambie. El sistema debe ser capaz de manejar estos cambios para asegurar que los datos guardados en versiones anteriores sean compatibles con versiones más nuevas del juego.

Rendimiento y tiempo de carga El proceso de guardar y cargar datos debe ser eficiente, de manera que no interrumpa la experiencia de juego. Cargar el estado del jugador y la IA debe ser rápido, especialmente en juegos donde los jugadores pueden cargar partidas frecuentemente.

Seguridad de los datos Es importante garantizar que los archivos de guardado no puedan ser manipulados por usuarios para obtener ventajas indebidas en el juego. Implementar medidas de seguridad, como encriptación o verificación de integridad, es crucial para prevenir trampas.

Escalabilidad En juegos más complejos, donde el estado del jugador y la IA pueden ser muy detallados, el sistema de guardado y carga debe escalar adecuadamente. Esto implica ser capaz de manejar grandes volúmenes de datos sin afectar el rendimiento.

Entradas:

- Estado del jugador:
  - nivel
  - puntuacion
  - vidas
  - progreso
- Configuración de la IA:
- dificultad$_a ctualparametros_I A(velocidad, númerodeenemigos, patrones)$

Salidas:

- Datos guardados:
  - estado_jugador_guardado
  - configuracion_IA_guardada

Actores:

- Sistema de almacenamiento

Pseudocódigo:

Algorithm 27: Guardar y Cargar Datos del Proceso y Configuraciones de IA

Input: estado_jugador, configuracion_IA
Output: datos_guardados

```
guardar_progreso() // Recopilar el estado del jugador
estado_jugador ← {nivel, puntuacion, vidas, progreso};
// Recopilar la configuración de la IA
configuracion_IA ← {dificultad_actual, parametros_IA};
// Abrir archivo para guardar los datos
archivo_guardado ← abrir_archivo("datos_guardados.dat", .escritura");
// Escribir datos en el archivo
escribir_archivo(archivo_guardado, estado_jugador, configuracion_IA);
// Cerrar el archivo
cerrar_archivo(archivo_guardado);
cargar_progreso() // Abrir archivo para cargar los datos guardados
archivo_guardado ← abrir_archivo("datos_guardados.dat", "lectura");
// Leer datos del archivo
datos_guardados ← leer_archivo(archivo_guardado);
// Cerrar el archivo
cerrar_archivo(archivo_guardado);
// Restaurar el estado del jugador
nivel ← datos_guardados[.estado_jugador"]["nivel"];
puntuacion ← datos_guardados[.estado_jugador"]["puntuacion"];
vidas ← datos_guardados[.estado_jugador"]["vidas"];
progreso ← datos_guardados[.estado_jugador"]["progreso"];
// Restaurar la configuración de la IA
dificultad_actual ← datos_guardados[çonfiguracion_IA"]["dificultad_actual"];
parametros_IA ← datos_guardados[çonfiguracion_IA"]["parametros_IA"];
// Aplicar el estado y configuración restaurados
actualizar_estado_jugador(nivel, puntuacion, vidas, progreso);
actualizar_configuracion_IA(dificultad_actual, parametros_IA);
```

Explicación:

El pseudocódigo cubre dos funciones principales: guardar el estado actual del jugador y la configuración de la IA en un archivo, y luego cargar esos datos para restaurar el progreso y las configuraciones en futuras sesiones. - El sistema de almacenamiento utiliza funciones para abrir, leer y escribir archivos, permitiendo la persistencia de datos. - Los datos incluyen tanto el estado del jugador (nivel, puntuación, vidas, etc.) como las configuraciones de IA (dificultad, velocidad de enemigos, etc.). - En la función de carga, los datos se restauran y aplican al sistema del juego para continuar desde donde se dejó.

4.6.7. Optimización de rendimiento

```
// Pseudocódigo para la optimización del rendimiento
void optimizarRendimiento() {
    float frameRate = obtenerFrameRate();

    if (frameRate < 60) {
        reducirCalidadGrafica();
        ajustarResolucion();
    }
    liberarRecursosNoUsados();
}
```

```
// Función de monitoreo
void monitorearSistema() {
    while (true) {
        optimizarRendimiento();
        esperar(100); // Esperar 100 ms antes de la siguiente comprobación
    }
}
```

4.6.8. Escalabilidad del algoritmo de IA

```
// Pseudocódigo para la escalabilidad del algoritmo de IA
class ModeloIA {
    vector<string> variables;
    map<string, float> parametros;

    void agregarVariable(string nuevaVariable) {
        variables.push_back(nuevaVariable);
    }

    void ajustarModelo() {
        for (string variable : variables) {
            // Ajuste de parámetros basados en el rendimiento actual
            parametros[variable] = calcularNuevoParametro(variable);
        }
    }
};

// Función para escalar el modelo de IA
void escalarIA(ModeloIA& modelo, vector<string> nuevasVariables) {
    for (string variable : nuevasVariables) {
        modelo.agregarVariable(variable);
    }
    modelo.ajustarModelo();
}
```

4.6.9. Interfaz de Usuario y Usabilidad

```
// Inicializar la ventana de la interfaz
sf::RenderWindow ventana(sf::VideoMode(800, 600), "Juego Procedural");

// Diseñar la interfaz y los controles
void inicializarInterfaz() {
    // Crear botones, texto y otros elementos de la interfaz
}

// Probar la interfaz para asegurar usabilidad
void probarInterfaz() {
    // Realizar pruebas de usabilidad y ajustar elementos según sea necesario
}

// Ajustar la interfaz y los controles
void ajustarInterfaz() {
    // Modificar la interfaz para mejor experiencia de usuario
}

// Ejecutar el bucle principal del juego
void ejecutarJuego() {
```

```
    while (ventana.isOpen()) {
        sf::Event evento;
        while (ventana.pollEvent(evento)) {
            if (evento.type == sf::Event::Closed)
                ventana.close();
        }

        ventana.clear();
        // Dibujar elementos de la interfaz aquí
        ventana.display();
    }
}
```