# DOCUMENTACIÓN TÉCNICA

## PROTOTIPO ModeX v1.5

Sistema Arcade Multijuego en Python

---

**Guía de Implementación y Arquitectura del Sistema**

ModeX Team · 01 / 03 / 2026

# Resumen Ejecutivo

ModeX v1.5 es un sistema de entretenimiento arcade que integra cuatro videojuegos originales bajo una plataforma unificada desarrollada en Python. El proyecto demuestra la viabilidad técnica de Python para el desarrollo de videojuegos 2D y presenta una arquitectura modular que facilita la replicación y extensión del sistema.

| Parámetro | Valor |
|---|---|
| Lenguaje principal | Python 3.x |
| Frameworks gráficos | Python Arcade 2.x / Pygame 2.5.x |
| Número de juegos | 4 (Racing, Platform, Shooter, Fight) |
| Patrón arquitectónico | MVC + Modular por capas |
| Rendimiento objetivo | 60 FPS estable |
| Consumo de memoria | ~187 MB en ejecución |
| Versión del prototipo | v1.5 (versión de investigación) |
| Plataformas soportadas | Windows 10/11, Ubuntu 22.04 |

# 1. Introducción

Este documento presenta la arquitectura técnica del sistema arcade ModeX v1.5. Está diseñado para permitir la replicación del proyecto por parte de desarrolladores, estudiantes autodidactas y cualquier persona interesada, ofreciendo explicaciones detalladas de cada componente, las decisiones de diseño adoptadas y su justificación técnica.

El objetivo principal de ModeX no es únicamente proporcionar entretenimiento, sino servir como plataforma de investigación que valida la hipótesis de que Python puede ser una alternativa viable para el desarrollo profesional de videojuegos 2D.

# 2. Stack Tecnológico

## 2.1 Selección de Python

Python fue elegido como lenguaje principal por su sintaxis clara, su ecosistema de bibliotecas especializadas y su adoptabilidad en entornos educativos. Si bien como lenguaje interpretado presenta menor rendimiento en operaciones de bajo nivel comparado con C++ o C#, las optimizaciones implementadas en ModeX permiten mantener 60 FPS estables.

## 2.2 Comparativa: Pygame vs Arcade

| Criterio | Pygame | Arcade |
|---|---|---|
| Edad del framework | Más de 20 años | Moderno (~2018) |
| API de sprites | Manual (sin abstracción) | Automatizada (SpriteList) |
| Detección de colisiones | Implementación manual | Integrada con motor de físicas |
| Dibujado por lotes (batch) | No nativo | Sí, mejora significativa en rendimiento |
| Gestión de escenas | Manual | Sistema de vistas (arcade.View) |
| Curva de aprendizaje | Media-alta | Baja-media |
| Uso en ModeX | Shooter, Fight | Racing, Platform, Menús |

## 2.3 Arquitectura Híbrida

ModeX utiliza ambas bibliotecas de forma complementaria. Arcade impulsa los juegos de Racing y Platform, así como todo el sistema de menús (covers/). Pygame impulsa los juegos de Shooter y Fight. Esta combinación demuestra que Python permite integrar múltiples frameworks en un mismo proyecto sin conflictos irresolubles, gracias al mecanismo de subprocesos (subprocess.run) para aislar contextos de ejecución.

# 3. Arquitectura General del Sistema

## 3.1 Estructura de Directorios

| Directorio / Archivo | Responsabilidad |
|---|---|
| assets/ | Recursos compartidos: imágenes, fuentes, audio |
| covers/ | Vistas de interfaz: portada, login, menú principal |
| games/racing/ | Juego de carreras (Neon Rush) — motor Arcade |
| games/platform/ | Juego de plataformas (Cosmic Odyssey) — motor Arcade |
| games/shooter/ | Juego de disparos (The Past Z) — motor Pygame |
| games/fight/ | Juego de peleas (The Ghost Tsushima) — motor Pygame |
| state/ | Persistencia de datos: SQLite, logros, respaldos |
| utils/ | Utilidades compartidas: rutas, colores, fuentes, botones |
| main.py | Punto de entrada del sistema |

## 3.2 Flujo de Ejecución General

El siguiente diagrama resume el ciclo de vida completo de una sesión en ModeX, desde el arranque hasta la salida del sistema.

**INICIO — main.py**

▼ Configuración y optimizaciones iniciales

**TitleCoverView — Pantalla de portada**

▼ ENTER / Clic en "Iniciar"

**LoginView — Validación de usuario (4 letras + 2 números)**

▼ Usuario válido confirmado

**MainMenuView — Selector de juego principal**

| Tecla | Juego | Motor | Mecanismo de lanzamiento |
|---|---|---|---|
| 1 | Neon Rush (Racing) | Arcade | window.show_view(RacingTitleView) |
| 2 | Cosmic Odyssey (Platform) | Arcade | window.show_view(PlatformTitleView) |
| 3 | The Ghost Tsushima (Fight) | Pygame | subprocess.run(fight_game.py) |
| 4 | The Past Z (Shooter) | Pygame | subprocess.run(shooter_game.py) |

▼ Al terminar la partida

| **Regreso a MainMenuView o salida del sistema** |
|---|

# 4. Módulo de Utilidades (utils/)

El módulo utils/ concentra todos los componentes reutilizables del sistema. Su diseño garantiza que la lógica compartida esté en un único lugar, evitando duplicación de código y asegurando coherencia visual y funcional en todos los juegos.

| Archivo | Función principal | Patrón / Técnica |
|---|---|---|
| singleton.py | Metaclase SingletonMeta: garantiza una sola instancia por clase | Patrón Singleton + __slots__ |
| fonts.py | Gestión centralizada de fuentes tipográficas (Press Start 2P, Pixel Emulator) | Atributos de clase + carga única |
| colors.py | Paleta de ~40 constantes RGBA: colores neón, interfaz y transparencias | Módulo de constantes |
| paths.py | Construcción de rutas a assets + caché de texturas y texturas volteadas | lru_cache + pathlib.Path |
| draw.py | Efecto de texto neón en 3 capas: resplandor, contorno y texto principal | Renderizado por capas |
| button.py | Componente de botón con estados hover/press/flash y colores derivados automáticos | Componente reutilizable + __slots__ |

## 4.1 Sistema de Caché de Texturas (paths.py)

Para evitar cargas repetidas desde disco, paths.py implementa tres niveles de caché mediante diccionarios globales: _TEXTURE_CACHE (texturas originales), _FLIPPED_TEXTURE_CACHE (texturas volteadas horizontalmente) y _GET_TEXTURES_CACHE (conjuntos de frames de animación). La clave de cada entrada es la ruta del archivo o el id() del objeto textura.

## 4.2 Componente Button

La clase Button centraliza toda la lógica de los botones de interfaz, eliminando aproximadamente 300 líneas de código duplicado que existían en las versiones 1.0–1.2. A partir de un color base, el componente calcula automáticamente tres variantes: _outline_color (base oscurecido 10 unidades), _pressed_color (oscurecido 30) y _hover_color (aclarado ~30), garantizando coherencia visual sin configuración adicional.

# 5. Módulo de Persistencia (state/)

El módulo state/ gestiona toda la persistencia de datos del sistema: sesiones de juego, logros, estadísticas de jugadores y respaldos de la base de datos. Su arquitectura en capas separa los datos puros (models.py), el acceso a datos (db.py) y la lógica de negocio (manager.py).

## 5.1 Arquitectura del Módulo

| Componente | Archivo | Responsabilidad |
|---|---|---|
| Modelos de datos | models.py | Dataclasses inmutables: Player, Session, GameStats |
| Acceso a datos | db.py | Clase Database: conexión SQLite, CRUD, integridad |
| Lógica de negocio | manager.py | DBManager: login, logout, guardado de sesiones |
| Respaldos | backup.py | DBBackup: respaldos automáticos y restauración |
| Logros — Definiciones | achievements/definition.py | AchievementDefinition (frozen dataclass) |
| Logros — Registro | achievements/registry.py | AchievementRegistry con @cache |
| Logros — Verificación | achievements/checker.py | AchievementChecker con umbrales por juego |
| Constantes | constants.py | Metadatos de cada juego (IDs, nombres, tipos) |

## 5.2 Flujo de Recuperación de la Base de Datos

La clase Database implementa una estrategia de recuperación ante fallos en 3 niveles que prioriza la disponibilidad del sistema sobre la preservación total de datos históricos.

**Intento de conexión a ModexArcade.db**

▼

**¿Pasa PRAGMA integrity_check?**

| Resultado | Acción |
|---|---|
| ✓ Sí — BD íntegra | Usar base de datos existente normalmente |

| Resultado | Acción |
|---|---|
| ✗ No — BD corrupta | Mover corrupta a /corrupted/ con timestamp |
| Backup disponible | Restaurar desde backup más reciente (Nivel 2) |
| Sin backups válidos | Crear base de datos nueva desde cero (Nivel 3) |

## 5.3 Principios de Diseño del Módulo state/

| Principio | Implementación | Beneficio |
|---|---|---|
| Singleton | Database, DBManager, registros de logros | Una sola conexión a BD; sin estados divergentes |
| Caché agresivo | @lru_cache, @cache, diccionarios internos | Minimiza consultas a disco en el game loop |
| Inmutabilidad | dataclass(frozen=True) en modelos y definiciones | Previene modificaciones accidentales de datos |
| Transaccionalidad | BEGIN IMMEDIATE + rollback automático | Integridad ante fallos en operaciones compuestas |
| Separación SRP | models → db → manager (3 capas) | Testabilidad, reutilización y claridad de código |

# 6. Juego de Carreras — Neon Rush (games/racing/)

Neon Rush es un juego de carreras en perspectiva cenital (top-down) donde el jugador compite contra un vehículo controlado por IA a través de tres circuitos progresivamente más desafiantes. El módulo está construido sobre Python Arcade con una arquitectura orientada a objetos de cuatro capas.

## 6.1 Estructura de Capas

| Capa | Componentes principales | Responsabilidad |
|---|---|---|
| Ítems (items/) | Car, CPU, Track, Wall | Entidades físicas del mundo del juego |
| Estado (state/) | GameState, RaceManager | Lógica de carrera, árbitro y textos de UI |
| Recursos (resources/) | globals.py | Configuración centralizada por nivel |
| Vistas (views/) | RacingTitleView, PauseView, RacingGameEnd, RacingGameOver | Pantallas e interfaz de usuario |

## 6.2 Sistema de Físicas del Vehículo

| Parámetro | Valor | Descripción |
|---|---|---|
| MAX_SPEED_NORMAL | 300 px/s | Velocidad máxima sin turbo activo |
| MAX_SPEED_TURBO | 450 px/s | Velocidad máxima durante el turbo |
| ACCELERATION | 400 px/s² | Tasa de incremento de velocidad |
| FRICTION | 0.98 | Multiplicador de deceleración por frame (normal) |
| TURBO_FRICTION | 0.993 | Fricción reducida durante el turbo |
| TURBO_DURATION | 2.0 s | Duración del impulso de turbo |
| TURBO_COOLDOWN | 5.0 s | Recarga antes de poder usar el turbo de nuevo |
| ROTATION_SPEED | 300 °/s | Velocidad de giro del vehículo |

## 6.3 Inteligencia Artificial del CPU

El vehículo CPU navega mediante un sistema de waypoints predefinidos por pista. En cada frame, calcula el ángulo hacia el waypoint activo y aplica una corrección suave mediante interpolación lineal (lerp) para producir giros fluidos. Incluye tres mecanismos avanzados:

| Mecanismo | Descripción |
|---|---|
| Control de velocidad por curvatura | Reduce la velocidad objetivo proporcionalmente a la cercanía del siguiente waypoint; fórmula cuadrática que respeta curvas moderadas pero frena agresivo en curvas cerradas. |
| Rubber-banding | Ajusta dinámicamente la velocidad máxima del CPU según la distancia al jugador, manteniendo la carrera competitiva en todas las dificultades. |
| Recuperación anti-atasco | Si el CPU no avanza más de 10 unidades en 120 frames, activa un ciclo de recuperación de 3 etapas: retroceder (60 frames) → girar (30 frames) → avanzar lento (60 frames). |

## 6.4 Dificultades del CPU

| Dificultad | corner_brake_factor | offset_magnitude | Comportamiento |
|---|---|---|---|
| Fácil | 0.85 (frena poco) | 18° (zig-zag notorio) | Impreciso; comete errores visibles en curvas |
| Medio | 0.65 | 10° | Conducción fluida con imperfecciones moderadas |
| Difícil | 0.50 (frena agresivo) | 4° (casi perfecto) | Trazado óptimo; muy difícil de superar |

## 6.5 Flujo de Pantallas — Racing

**RacingTitleView — Portada Neon Rush**

▼ ENTER

**RacingCarMenu — Selección de vehículo (1/2/3 o clic)**

▼ Selección confirmada

**RacingGame — Nivel 1 (Física · Colisiones · Checkpoints)**

| Condición | Transición |
|---|---|
| race_state = "finished" (Nivel 1 o 2) | Cargar siguiente nivel (_load_level) |
| race_state = "finished" (Nivel 3) | Navegar a RacingGameEnd (Victoria) |
| race_state = "game_over" | Navegar a RacingGameOver (Derrota) |
| ENTER durante carrera | Activar PauseView (Singleton) |

# 7. Juego de Disparos — The Past Z (games/shooter/)

The Past Z: Defense & Cleanup es un videojuego de acción en vista lateral desarrollado con Pygame. El jugador debe sobrevivir oleadas de zombis gestionando munición, puntuación y mejoras de armas. Soporta modalidad de un jugador y dos jugadores simultáneos.

## 7.1 Modos de Juego

| Modo | Nombre en pantalla | Objetivo | Archivo |
|---|---|---|---|
| defense | Last Stand | Proteger un muro central de los ataques zombis hasta que sea destruido | last_stand_mode.py |
| cleanup | Defensa Clásica | Sobrevivir oleadas de zombis que atacan directamente al jugador | defense_mode.py |

## 7.2 Tipos de Zombis

| Tipo | Velocidad | HP | Daño/s | Disponible desde |
|---|---|---|---|---|
| Common | 80 px/s | 100 | 10 | Oleada 1 |
| Runner | 200 px/s | 80 | 15 | Oleada 1 |
| Cryo | 60 px/s | 200 | 20 | Oleada 1 (raro) |
| Explosive | 80 px/s | 90 | — (explota) | Oleada 1 (muy raro) |
| Toxic | 100 px/s | 180 | 8 | Oleada 1 (raro) |
| Electric | 88 px/s | 200 | 12 | Oleada 5 |
| Psionic | 72 px/s | 150 | — (efecto) | Oleada 8 |
| Alpha | 72 px/s | 300 | 35 | Oleada 10 |
| Sentinel | 100 px/s | 500 | 25 | Oleada 12 |
| Prototype | 80 px/s | 2500 | 50 | Oleada 15 (jefe) |

## 7.3 Sistema de Power-Ups

| Power-Up | Duración | Efecto |
|---|---|---|
| Salud (heal) | Instantáneo | Recupera 30 HP al instante |
| Munición (maxammo) | Instantáneo | Rellena todos los cargadores al máximo |
| Velocidad (speed) | 5 s | Duplica la velocidad de movimiento |
| Disparo rápido (rapidfire) | 15 s | Divide el fire_rate entre 2; activa disparo triple con dispersión |
| Escudo (shield) | 8 s | Añade 100 puntos de escudo que absorben daño antes que el HP |
| Puntos dobles (doublepoints) | 15 s | Multiplica ×2 todos los puntos obtenidos |
| Congelar tiempo (timefreeze) | 10 s | Los zombis dejan de moverse completamente |
| Muerte instantánea (instakill) | 15 s | Daño del jugador = 1000 (mata de un disparo) |
| Daño aumentado (damage) | 10 s | Duplica el daño base del jugador |
| Invisibilidad (ghost) | 8 s | El jugador no recibe daño ni es detectado |
| Bomba nuclear (nuke) | Instantáneo | Elimina todos los zombis en pantalla |

## 7.4 Máquina Pack-a-Punch — Mejoras de Arma

| Estadística mejorada | Fórmula (nivel n) | Ejemplo (nivel 2) |
|---|---|---|
| Daño | base_damage × (5.0 + n × 0.90) | 20 × 6.8 = 136 de daño |
| Cadencia de fuego | base_fire_rate × (1.5 − n × 0.8), mín. 0.08 s | 0.3 s → 0.08 s (máximo) |
| Tamaño del cargador | base_magazine_size × (1.5 + n × 0.2) | 30 × 1.9 = 57 balas |
| Cargadores máximos | min(20, 12 + n) | 12 + 2 = 14 cargadores |

# 8. Juego de Peleas — The Ghost Tsushima (games/fight/)

The Ghost Tsushima es un videojuego de combate bidimensional (fighting game) desarrollado con Pygame. Enfrenta al Martial Hero (Jugador 1) contra el Skeleton (Jugador 2) en un escenario lateral con física de salto, sistema de animaciones por frames y combate 1 vs 1 en tiempo real.

## 8.1 Controles

| Acción | Jugador 1 (Martial Hero) | Jugador 2 (Skeleton) |
|---|---|---|
| Mover izquierda/derecha | Flechas ← → | Teclas A / D |
| Saltar | Flecha ↑ | Tecla W |
| Ataque básico (10 HP) | Tecla 0 / Clic izquierdo | Tecla G |
| Ataque medio (15 HP) | Tecla X / Clic derecho | Tecla H |
| Ataque fuerte (17 HP) | — | Tecla J |
| Pausar / Confirmar | ENTER | ENTER |
| Salir al menú ModeX | ESC | ESC |

## 8.2 Constantes de Física del Combate

| Constante | Valor | Descripción |
|---|---|---|
| GRAVITY | 6.0 px/frame² | Aceleración gravitacional aplicada cada frame cuando el personaje está en el aire |
| JUMP_SPEED | 36 px/frame | Velocidad inicial (hacia arriba) al saltar. Altura máxima ≈ 108 px |
| MOVE_SPEED | 14 px/frame (J1) / 10 (J2) | Velocidad de desplazamiento horizontal |
| GROUND_Y | SCREEN_H − 10 = 710 px | Posición Y del suelo; cuando y ≥ GROUND_Y el personaje aterriza |
| FPS objetivo | 60 frames/segundo | Velocidad del bucle de juego |

## 8.3 Jerarquía de Clases

El juego implementa una jerarquía de herencia de tres niveles: Actor (clase base con físicas, animaciones y sistema de daño) es la clase padre de Player (Jugador 1, control por delta-time) y

Skeleton (Jugador 2, control por conteo de ticks). Ambas clases sobrescriben los métodos de actualización y dibujado, y añaden lógica específica de sonido, colisión y sprites.

## 8.4 Tipos de Animación

| Animación | ¿Loopea? | Prioridad | Disparada por |
|---|---|---|---|
| quieto | Sí | Base | vel_x = 0, sin animación activa |
| correr | Sí | Baja | vel_x ≠ 0 |
| salto | No | Media | Tecla de salto (si en suelo) |
| ataque / ataque2 / ataque3 | No | Alta | Tecla de ataque; bloquea otros ataques |
| hit | No | Alta | apply_damage() con vida > 0 |
| muerte | No (último frame) | Máxima | apply_damage() con vida = 0; activa dying=True |

# 9. Juego de Plataformas — Cosmic Odyssey (games/platform/)

Cosmic Odyssey es un juego de plataformas de desplazamiento lateral donde el jugador avanza por tres niveles con enemigos, obstáculos y jefes al final de cada etapa. Utiliza Python Arcade y el editor de niveles Tiled Map Editor (archivos .tmx).

## 9.1 Entidades del Sistema

| Clase | Archivo | Descripción |
|---|---|---|
| Player | items/player.py | Personaje del jugador: movimiento con inercia, ataque (bala/melee), animaciones, invulnerabilidad temporal |
| Enemy | items/enemy.py | Enemigos con patrulla (A→B), detección de rango y sistema de vida configurable |
| Boss | items/boss.py | Jefe de nivel con máquina de estados de 5 fases según porcentaje de salud |
| Bullet | items/bullet.py | Proyectil con dueño, dirección, velocidad, daño y tiempo de vida máximo |
| Platform | items/platform.py | Plataforma estática o móvil (solo bloquea desde arriba) |
| Wall | items/wall.py | Pared sólida cargada desde capas .tmx (bloquea los 4 lados) |

## 9.2 Máquina de Estados del Jefe

| Estado | Condición de activación | Comportamiento |
|---|---|---|
| Idle | Al aparecer por primera vez | Espera un momento antes de comenzar |
| Patrol | Salud > 75% | Movimiento lento de lado a lado |
| Aggressive | Salud 40%–75% | Mayor velocidad y frecuencia de ataques |
| Enraged | Salud < 40% | Velocidad máxima, nuevos patrones de ataque |
| Dying | Salud = 0 | Animación de muerte; termina el nivel |

## 9.3 Sistema de Colisiones

| Tipo de colisión | Resultado |
|---|---|
| Jugador ↔ Pared/Suelo | Motor de físicas detiene el movimiento |
| Jugador ↔ Plataforma (desde arriba) | Se detiene sobre la plataforma |
| Jugador ↔ Plataforma (desde abajo) | La atraviesa (sin colisión) |
| Jugador ↔ Enemigo/Jefe | El jugador recibe daño si no está invulnerable |
| Bala del jugador ↔ Enemigo/Jefe | El enemigo recibe daño; la bala se destruye |
| Bala enemigo ↔ Jugador | El jugador recibe daño si no está invulnerable |
| Bala ↔ Pared/Plataforma | La bala se destruye al impactar |
| Jugador ↔ Coleccionable | Se recoge el ítem y se aplica su efecto |
| Jugador ↔ Zona de meta | Se activa la condición de victoria del nivel |

## 9.4 Flujo de Progresión — Platform

**PlatformTitleView — Portada Cosmic Odyssey**

▼ ENTER

**Menú de selección de personaje (menus/)**

▼ Personaje guardado en GameState

**PlatformGame — Nivel 1 (Física · Colisiones · Enemigos · Jefe)**

| Condición | Transición |
|---|---|
| Nivel completado (meta o jefe derrotado) | _load_level(load_next_level=True) |
| Todos los niveles completados | Navegar a PlatformGameEnd (Victoria) |
| Sin vidas o tiempo agotado | Navegar a PlatformGameOver (Derrota) |
| ENTER/ESC durante juego | Activar PauseView (Singleton) |

# 10. Módulo de Interfaz — Covers (covers/)

El módulo covers/ constituye la capa de presentación de entrada de ModeX. Contiene las vistas previas al juego y actúa como intermediario entre el sistema Arcade y los juegos Pygame, gestionando la identidad visual del proyecto y la autenticación del jugador.

## 10.1 Vistas del Módulo

| Vista | Clase | Función |
|---|---|---|
| Portada | TitleCoverView | Primera pantalla del sistema. Muestra imagen de portada ModeXCover.png con botón "Iniciar". |
| Login | LoginView | Captura y valida el nombre de usuario (formato: 4 letras + 2 números). Incluye lista de palabras prohibidas (frozenset). |
| Menú principal | MainMenuView | Selector de los 4 juegos. Gestiona transiciones Arcade→Arcade y lanzamiento Arcade→Pygame vía subprocess. |

## 10.2 Validación de Usuario — LoginView

| Condición de error | Mensaje mostrado | Duración |
|---|---|---|
| len(letras) ≠ 4 | DEBES INGRESAR 4 LETRAS | 3 segundos |
| len(números) ≠ 2 | DEBES INGRESAR 2 NÚMEROS | 3 segundos |
| Letras en FORBIDDEN_WORDS | PALABRA NO PERMITIDA | 3 segundos |

## 10.3 Mecanismo de Lanzamiento de Juegos Pygame

Arcade y Pygame no pueden ejecutarse en el mismo proceso simultáneamente ya que ambos frameworks intentan controlar la ventana del sistema operativo. La solución implementada en MainMenuView._launch_pygame_game() consiste en:

| Paso | Acción |
|---|---|
| 1 | Detener la música del menú Arcade |
| 2 | Ocultar la ventana Arcade (window.set_visible(False)) |

| Paso | Acción |
|---|---|
| 3 | Ejecutar el juego Pygame como subproceso bloqueante (subprocess.run) |
| 4 | Esperar a que el proceso Pygame termine y leer su código de salida |
| 5 | exit_code = 0 → mostrar ventana Arcade de nuevo (volver al menú) |
| 6 | exit_code = 99 → cerrar toda la aplicación (arcade.exit()) |

## 10.4 Patrón de Gestión de Audio

Las tres vistas implementan el mismo patrón de ciclo de vida para el audio: carga en __init__() (sin reproducción), reproducción en bucle en on_show_view(), y detención explícita antes de cada transición. Este patrón previene la reproducción de múltiples pistas superpuestas, que es el error de audio más común en sistemas multi-vista.

# 11. Patrones de Diseño Aplicados

| Patrón | Implementación en ModeX | Beneficio principal |
|---|---|---|
| Singleton | Database, DBManager, PauseView, AchievementRegistry, GameState, RaceManager | Una sola instancia global accesible; evita estados divergentes y costosas re-inicializaciones |
| Máquina de estados | boss_data/state.py (Boss), RaceManager (estados de carrera) | Comportamiento dinámico limpio y extensible sin lógica condicional anidada |
| Repository | Separación db.py (datos) vs manager.py (negocio) | Testabilidad mediante mocking; reutilización de la capa de datos |
| Componente reutilizable | Button en utils/button.py | Eliminación de ~300 líneas duplicadas; consistencia visual automática |
| Caché de recursos | _TEXTURE_CACHE, _FLIPPED_TEXTURE_CACHE, map_cache | Reducción de latencia de disco; rendimiento estable durante el game loop |
| Separación de responsabilidades | Capas items / state / resources / views en cada módulo de juego | Código cohesivo y de bajo acoplamiento; cambios localizados |
| Configuración centralizada | resources/globals.py en cada módulo de juego | Parámetros del juego modificables sin tocar la lógica central |
| Subproceso aislado | subprocess.run() para juegos Pygame | Coexistencia de dos frameworks incompatibles en la misma plataforma |

# 12. Evolución del Proyecto

## 12.1 Historial de Versiones

| Versión | Hito principal | Rendimiento | Memoria |
|---|---|---|---|
| v1.0 — Prototipo inicial | Sin caché de texturas, rutas como strings literales, sin optimizaciones GC | ~40 FPS | ~300 MB |
| v1.2 — Primera refactorización | Módulos independientes, introducción de utils/, implementación de __slots__ | ~48 FPS | ~240 MB |
| v1.4 — Arquitectura modular | Sistema de vistas unificado, caché agresivo en paths.py, gestión manual de GC, clase Button centralizada | 55–60 FPS | ~190 MB |
| v1.5 — Versión de investigación | Integración de PerformanceAnalyzer, refinamiento de optimizaciones, documentación técnica completa | 58 FPS promedio | ~187 MB |

## 12.2 Métricas Técnicas del Sistema

| Métrica | Valor |
|---|---|
| FPS estable (v1.5) | 58 FPS promedio |
| Consumo de memoria (v1.5) | ~187 MB en ejecución |
| Reducción de memoria (v1.0→v1.5) | 38% de reducción (~300 MB → ~187 MB) |
| Módulos de juego | 4 (Racing, Platform, Shooter, Fight) |
| Total de archivos Python | >45 archivos .py |
| Tipos de zombis (Shooter) | 10 tipos con estadísticas únicas |
| Power-ups (Shooter) | 11 tipos con efectos distintos |
| Logros (Shooter) | 12 logros (5 secretos) |
| Resoluciones soportadas | 6 resoluciones predefinidas con ajuste automático |
| Niveles por juego | 3 (Racing y Platform); oleadas infinitas (Shooter) |

| Métrica | Valor |
|---|---|
| Dificultades de CPU (Racing) | 3 (Fácil, Medio, Difícil) |
| Estados del jefe (Platform) | 4 estados dinámicos según % de salud |

# 13. Guía de Replicabilidad

## 13.1 Requisitos del Sistema

| Componente | Versión recomendada | Notas |
|---|---|---|
| Python | 3.10 o superior | Testeado en Windows 10/11 y Ubuntu 22.04 |
| Python Arcade | 2.6.x | pip install arcade |
| Pygame | 2.5.x | pip install pygame |
| Almacenamiento | ~500 MB libres | Para assets (sprites, imágenes, sonidos) |
| RAM | Mínimo 512 MB (recomendado 2 GB) | Para evitar stuttering en bucles de juego |
| Resolución de pantalla | 1280×720 mínima | Resolución base del sistema |

## 13.2 Pasos de Instalación

| Paso | Comando / Acción |
|---|---|
| 1. Clonar el repositorio | git clone <url_del_repositorio> |
| 2. Crear entorno virtual (opcional) | python -m venv venv && source venv/bin/activate |
| 3. Instalar dependencias | pip install arcade pygame |
| 4. Verificar estructura de assets | Confirmar que existe la carpeta assets/ con subdirectorios imgs/, music/ y fonts/ |
| 5. Ejecutar el sistema | python main.py |

## 13.3 Consideraciones Importantes

Inversión de modos en Shooter: internamente, el código llama "defense" al modo que ejecuta Last Stand y "cleanup" al modo de Defensa Clásica. Esta inversión es intencional y debe preservarse al añadir funcionalidades.

Resolución fija en Shooter y Fight: ambos juegos están diseñados para 1280×720. Cambiar WIDTH/HEIGHT puede descolocar elementos de la interfaz.

Directorio de trabajo: los archivos shooter_game.py y fight_game.py cambian el directorio de trabajo a su propia carpeta al iniciarse. Todos los paths son relativos a esa carpeta.

Tolerancia a assets faltantes: Shooter y Fight generan superficies de color sólido como respaldo cuando faltan archivos gráficos. Solo los sprites del jugador son estrictamente necesarios para una experiencia visual aceptable.

# 14. Glosario de Términos Técnicos

| Término | Definición |
|---|---|
| Arcade (biblioteca) | Framework moderno de Python para videojuegos 2D que utiliza OpenGL para renderizado acelerado por hardware. |
| Batch rendering | Técnica que agrupa múltiples llamadas de dibujado en una sola instrucción a la GPU, reduciendo la sobrecarga de comunicación. |
| delta_time (dt) | Tiempo transcurrido entre dos frames consecutivos. Permite que la física sea independiente de la velocidad del hardware. |
| Frozen dataclass | Clase de datos inmutable en Python (dataclass(frozen=True)). Una vez creada, sus atributos no pueden modificarse. |
| FPS | Frames Per Second (fotogramas por segundo). Mide la fluidez de la animación; 60 FPS es el estándar de referencia. |
| Game loop | Bucle principal de un videojuego que ejecuta en cada frame: procesar entrada → actualizar lógica → dibujar pantalla. |
| Hitbox | Rectángulo o forma invisible que define el área de colisión de una entidad, generalmente más pequeña que su sprite visible. |
| lru_cache / @cache | Decoradores de Python que memorizan el resultado de una función para evitar recalcular con los mismos argumentos. |
| Pygame | Biblioteca Python para videojuegos basada en SDL, con más de 20 años de desarrollo. Proporciona ventana, eventos, gráficos y audio. |
| Singleton | Patrón de diseño que garantiza que una clase tenga exactamente una instancia durante toda la ejecución del programa. |
| Sprite | Imagen 2D (o conjunto de imágenes animadas) que representa una entidad visual en el juego. |
| SQLite | Base de datos relacional embebida, sin servidor, que almacena todos sus datos en un único archivo .db. |
| subprocess.run() | Función de Python que ejecuta un proceso hijo del sistema operativo y espera (bloqueante) a que termine. |
| TMX / Tiled | Formato de mapa XML y editor visual gratuito (Tiled Map Editor) para diseñar niveles de videojuegos por capas. |
| Waypoint | Coordenada (x, y) predefinida que sirve de punto de paso para la navegación autónoma de la IA del CPU. |