Avance del Proyecto

Juego

Los Bien Producciones

Benjamín Contreras
Anselmo Díaz
Ignacio Ruiz

31 de octubre de 2025

ÍNDICE GENERAL

INTRODUCCIÓN

Este documento presenta el Avance GM del proyecto **"NOT HOTLINE MIAMI"**, un videojuego arcade 2D de supervivencia por oleadas, desarrollado sobre el framework **LibGDX**.

El informe no solo describe la funcionalidad y la experiencia visible para el usuario, sino que profundiza en la **arquitectura de software** y las decisiones de diseño que estructuran el proyecto. El objetivo es demostrar la aplicación de los principios de la Programación Orientada a Objetos para construir un sistema desacoplado, mantenible y extensible, capaz de gestionar la lógica del juego, sus armas, colisiones y múltiples estados de pantalla.

Para ello, el documento se organiza de la siguiente manera: Comienza con la perspectiva del jugador (GM1.1) y el análisis de ingeniería que detalla la evolución desde el prototipo base (GM1.2). Luego, se adentra en el diseño técnico, presentando el diagrama de dominio UML completo (GM1.3). Se analizan en detalle las abstracciones clave del sistema: el uso de **clases abstractas** como `Weapon` (Patrón Strategy) y `BaseScreen` (Patrón Template Method) (GM1.4), y la implementación de la **interfaz** `INavigableOption` para la gestión de menús (GM1.5). Finalmente, se consolida la aplicación de los principios OO (GM1.6) y se provee la evidencia del control de versiones (GM1.7).

AVANCE DEL PROYECTO: JUEGO

GM1.1: Explicación del Juego

Resumen

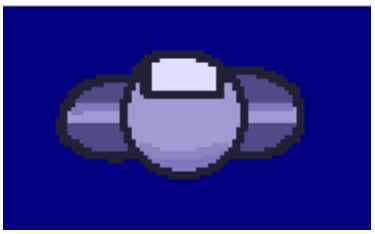
Arcade 2D de supervivencia por oleadas donde el jugador toma el rol de un personaje customizable (`Player`), moviéndose con física de inercia y enfrentando oleadas de enemigos empleando armas de disparo, rayos o melee. Cada ronda incrementa la dificultad; el HUD refleja recursos, estados y progreso. Al agotar la vida del personaje, aparece la pantalla de *Game Over* permitiendo un reinicio rápido o el volver al menu principal.

Objetivos de juego

- Superar cada ronda eliminando las amenazas entrantes o sobreviviendo bajo condiciones de tiempo o daño.
- Gestionar el arsenal: cambiar de arma entre opciones a distancia, recta o cuerpo a cuerpo, manteniendo la capacidad de ataque al quedar sin munición (conmutación automática a melee).
- Mejorar el desempeño: buscar `score` y `high score`, que se presentan en la interfaz; reiniciar y adaptarse tras cada derrota.

Elementos y feedback

- Protagonista: `Player`, un personaje animado y personalizable, con rotación, aceleración, fricción, estado de daño visual (parpadeo y sonido), y rebote controlado en los límites del escenario.



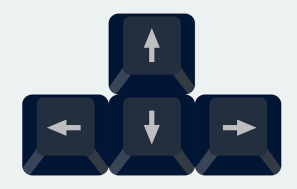
- Amenazas: Enemigos; reaccionan física y lógicamente al ser impactados por proyectiles, rayos o ataques cuerpo a cuerpo.

- Arsenal y proyectiles:
  - Proyectiles balísticos (`Bullet`) orientados al ángulo de disparo; eficaz en rango medio.

  - Rayos (`LaserBeam`) sostenidos o pulsados; barren la zona frente al personaje.
  - Ataques melee (`Swing`): golpe corto en arco, útil para defenderse a corta distancia.
- Interfaz: HUD informa vidas, ronda, puntaje, récord y estado de arma. El diseño incluye pantallas separadas para Inicio, Configuración de volúmenes, Personalización de aspecto, Tutorial guiado, Pausa con overlay y Game Over.

Controles

- Flechas izquierda/derecha: rotan el personaje; flecha arriba: aceleran la marcha; se desacelera por fricción al soltar.



- Tecla `Z`: dispara o activa el arma actual (puede ser bala, rayo, melee según la seleccionada).
- Tecla `ESC`: entra en Pausa; desde allí se puede continuar o ir al Menú Principal.

Maquetas y pantallazos

A continuación, se presentan las principales pantallas de la interfaz de usuario (UI) que componen el flujo de navegación del juego.

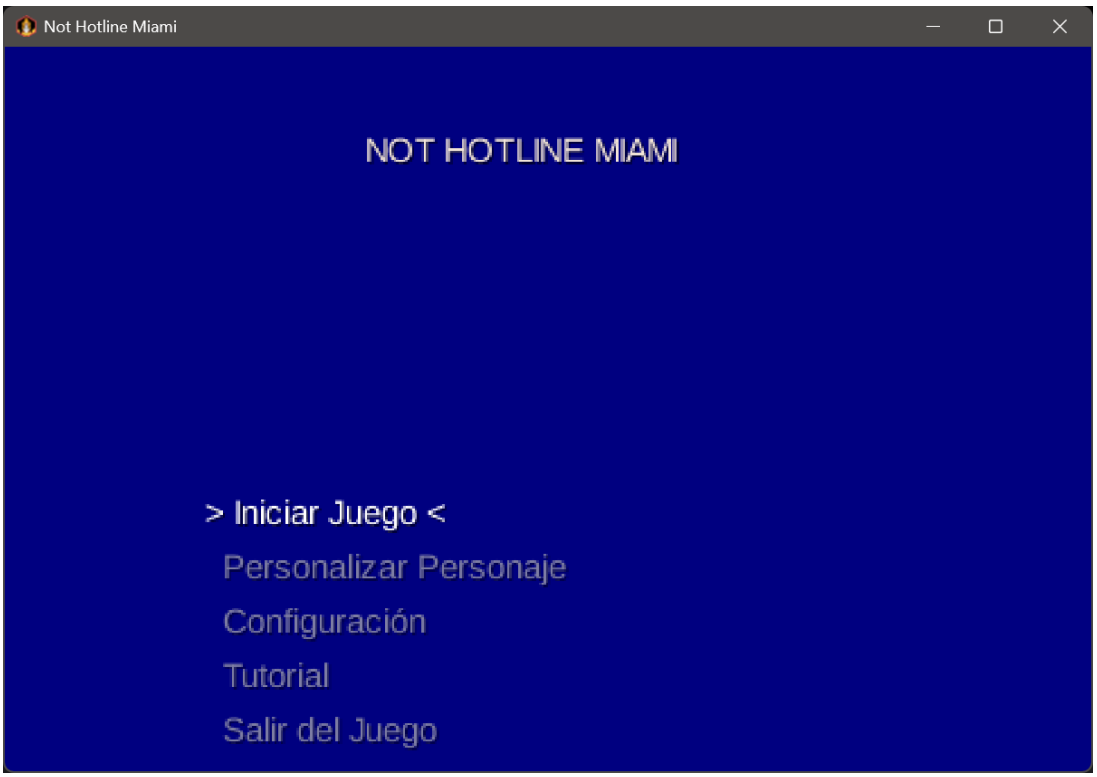


Figura 1: Pantalla de inicio (*MainMenuScreen*), punto de entrada al juego y los distintos sub-menús.

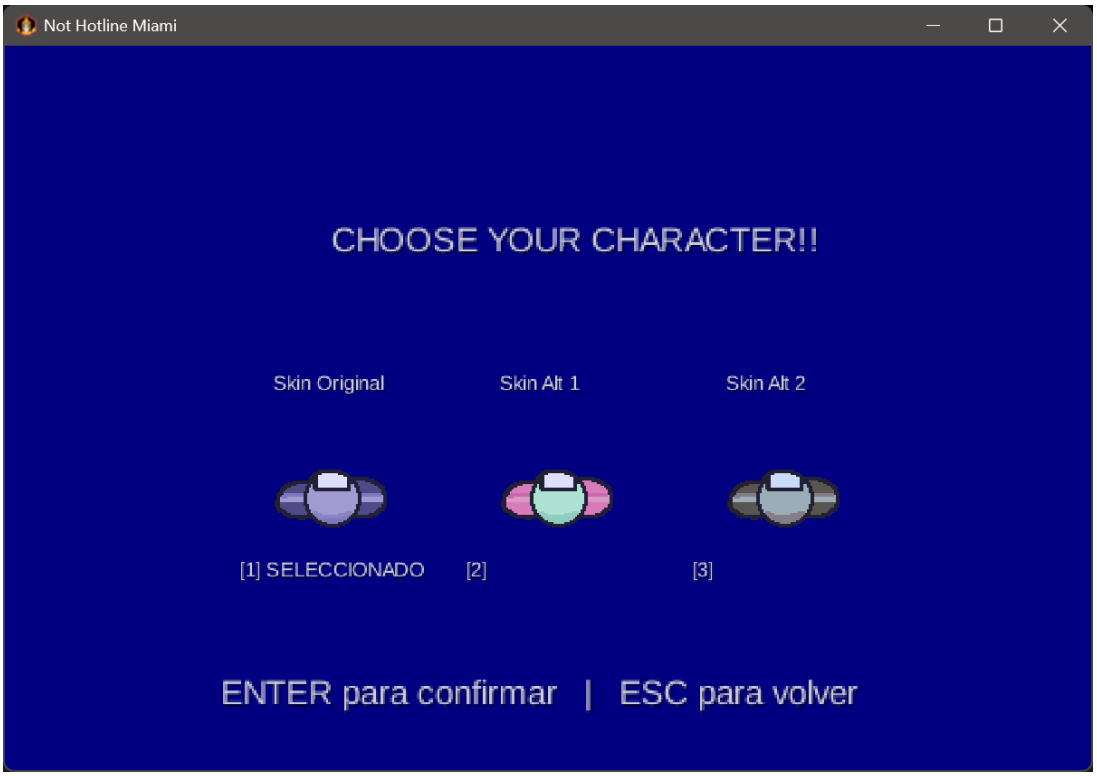


Figura 2: Pantalla de selección de personaje (*CustomizationScreen*), donde el jugador elije su apariencia.

Figura 3: Pantalla de configuración (*ConfigurationScreen*), permite el ajuste de los volúmenes de audio.

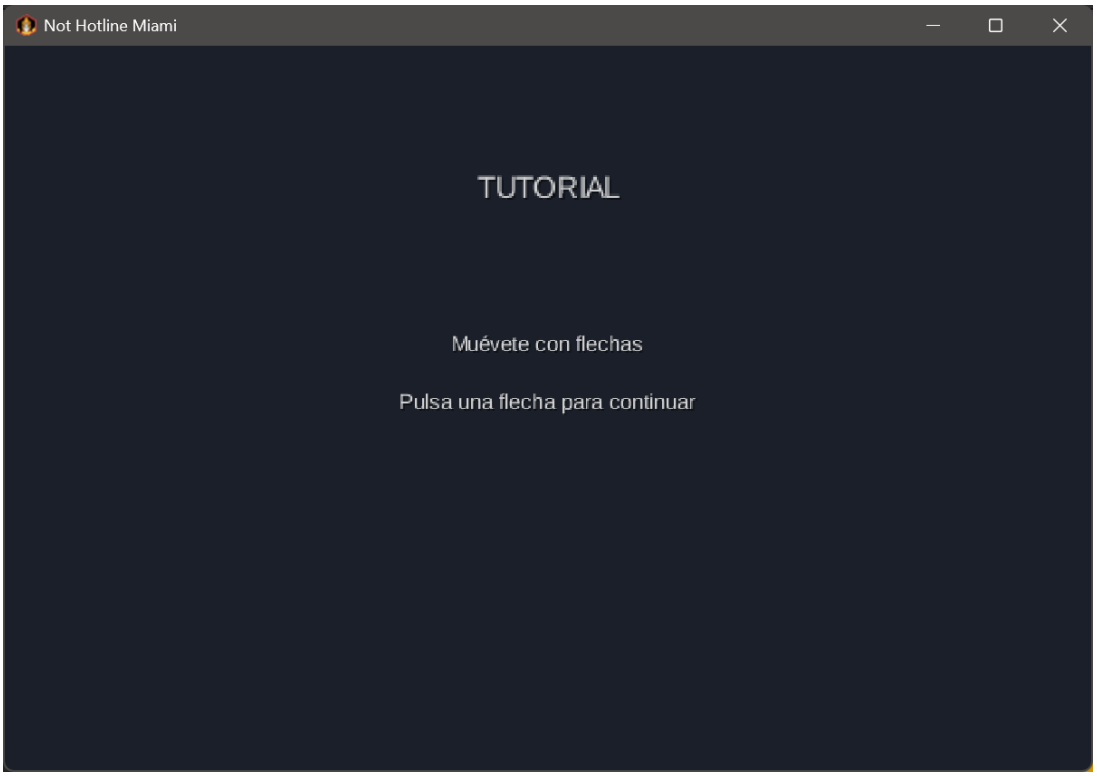


Figura 4: Pantalla de tutorial (*TutorialScreen*), que guía al jugador en los controles básicos.

Se han ilustrado las pantallas de navegación y configuración. En cuanto a la experiencia de juego en sí, las pantallas de *GameScreen* (jugabilidad y HUD), *PauseScreen* (superposición de pausa), y *GameOverScreen* (pantalla de muerte) se dejan para que el evaluador las descubra durante la ejecución del proyecto.

GM1.2: Análisis del Juego

Juego Base y Modificaciones

El proyecto toma como punto de partida un prototipo funcional basado en el clásico arcade ***Asteroids***, que incluía una nave (`Player`) con movimiento sin fricción, asteroides como únicos enemigos y un sistema de disparo con proyectiles infinitos.

Sobre esta base, se realizaron **modificaciones estructurales y de funcionalidad** clave para reorientar el juego a una experiencia de supervivencia arcade más moderna, similar al estilo de "Hotline Miami":

- **Física del Jugador:** Se descartó el movimiento original. En su lugar, se implementó un sistema de física con inercia controlada, donde el `Player` ahora acelera en una dirección (método `accelerate`) y se detiene mediante la aplicación de fricción (método `applyFriction`), como se evidencia en la clase `Player.java`.
- **Sistema de Armas:** El concepto de "proyectiles infinitos"se eliminó. Fue reemplazado por un **sistema de estrategia** (Patrón Strategy) encapsulado en la clase abstracta `Weapon.java`.
    - Cada arma ahora gestiona su propia munición (`municion` y `municionMax`) y

cadencia.
    - Esto permite al `Player` cambiar de arma y añade la necesidad de un arma de último recurso, el ataque `Melee` (basado en `Swing.java`), para cuando la munición se agota.
- **Estructura de Juego:** El prototipo original de una sola pantalla fue reemplazado por un flujo de aplicación robusto gestionado por el `ScreenManager`, permitiendo múltiples pantallas como menús, configuración y pausa.

Arquitectura y flujo

- El juego inicia en `MainGame`, donde se crea el batch de gráficos, la fuente y el viewport, se cargan recursos audiovisuales (Singleton `AssetManager`), y se instala el `ScreenManager` para navegación fluida entre menús y juego.
- El ciclo principal está en `GameScreen`, que instancia el `GameWorld`: allí se actualiza el personaje, managers de proyectiles, rayos, melee y oleadas, y se resuelven colisiones. Pausa y Game Over se manejan como pantallas, usando `EScreenType` y el ciclo intercambiable de `ScreenManager`.
- Los menús usan `NavigableScreen` junto a `OptionNavigator`, con opciones (`INavigableOption`) que ejecutan acciones (`IScreenAction`) para cambiar de pantalla o realizar operaciones.

Decisiones clave

- **Jerarquía de armas** (`Weapon` abstracta): cada arma define su lógica de disparo y crea proyectiles específicos (bala, rayo, melee); los managers administran activas y eliminan destruidas.
- **Sistema de colisiones** (`CollisionManager`): detecta interacciones entre el personaje, proyectiles y enemigos (`Enemy`); reproduce sonidos y suma puntaje según eventos.
- **Recursos y personalización**: `ESkinJugador` y `EGameSound` son enums; configuración de volúmenes en menú de opciones, con persistencia temporal (`volumenTemporal`) hasta confirmación.

GM1.3: Diseño UML y Dominio

Paquetes y Jerarquías del Dominio

Esta sección desglosa el diseño UML del sistema por paquetes, mostrando las principales clases y sus relaciones en diagramas de clases específicos para cada área funcional.

Jerarquía de Pantallas

Agrupa las clases que gestionan la interfaz de usuario. La clase abstracta `BaseScreen` (paquete `pantallas`) sirve como base para todas las pantallas, incluyendo `GameScreen`, `PauseScreen` y `GameOverScreen` (paquete `pantallas.juego`), así como `MainMenuScreen`, `ConfigurationScreen` y `CustomizationScreen` (paquete `pantallas.menus`). La navegación se maneja a través del `ScreenManager`.

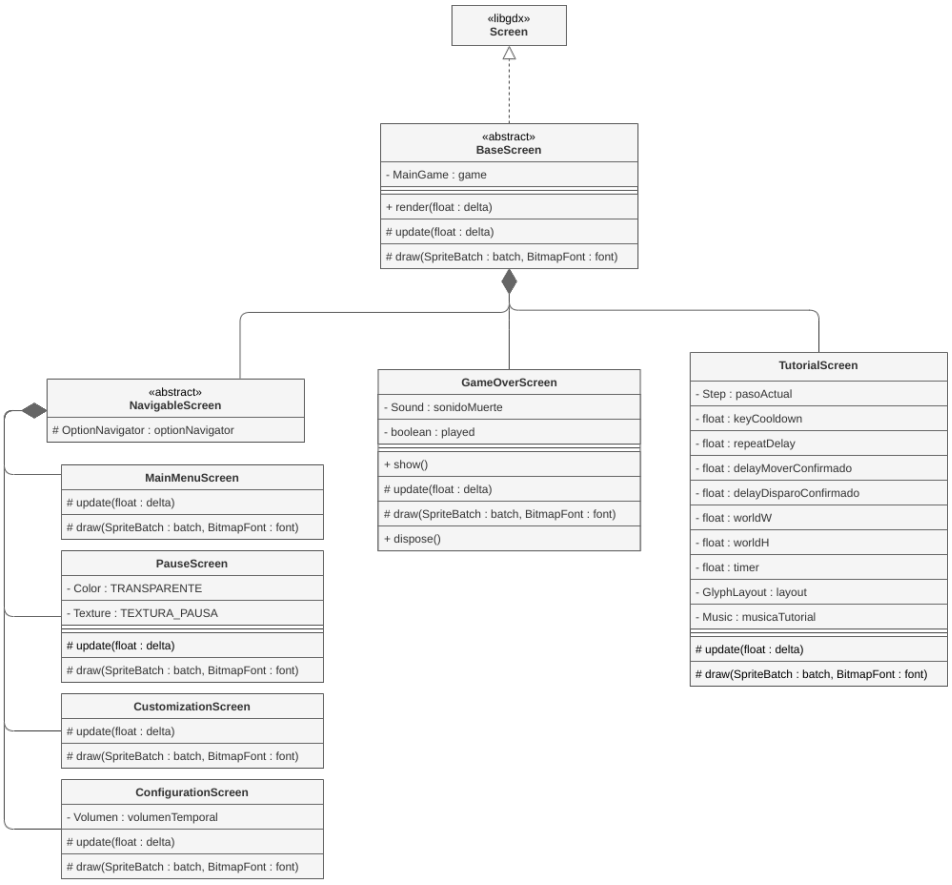


Figura 5: Diagrama de Clases UML del paquete `pantallas` y sus sub-paquetes.

Lógica del Juego

Este paquete contiene las clases centrales que orquestan el comportamiento principal: `MainGame` (punto de entrada), `GameWorld` (lógica activa), `OptionNavigator` (menús) y clases de utilidad. Destacan `AnimationFactory` (para animaciones) y la nueva `EnemyFactory` (Patrón Factory para la creación de instancias de `Enemy`).

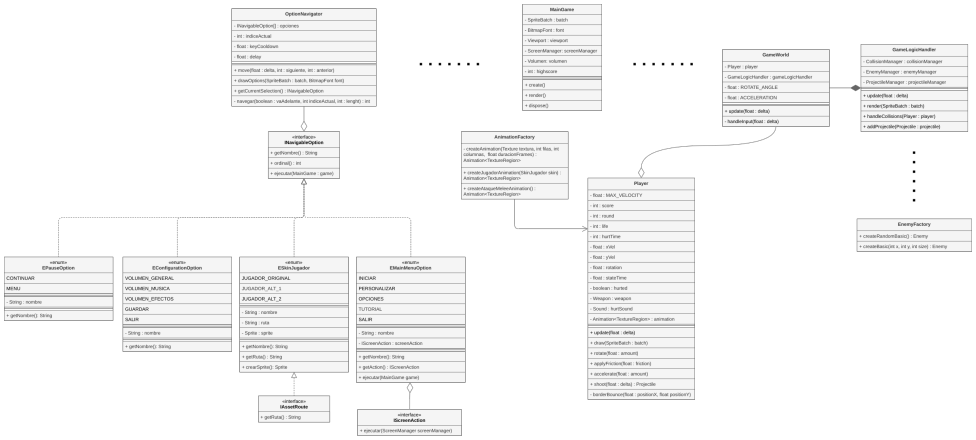


Figura 6: Diagrama de Clases UML del paquete `logica`.

Gestores (Managers)

Este grupo incluye los gestores encargados de las entidades y sistemas. `GameLogicHandler` coordina al nuevo `EnemyManager` (responsable de las oleadas de enemigos), `ProjectileManager` y `CollisionManager`. El paquete también incluye los Singletons `AssetManager` (recursos) y `ScreenManager` (navegación).

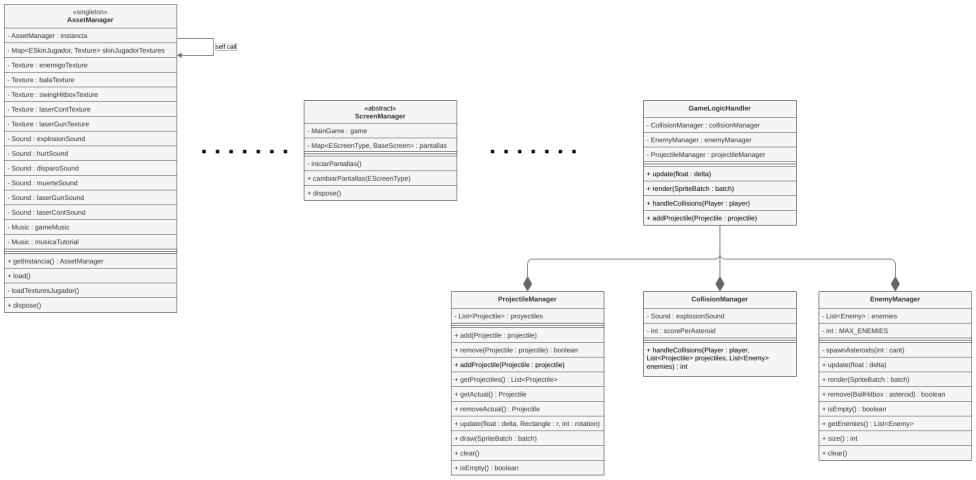


Figura 7: Diagrama de Clases UML del paquete `managers`.

Personajes

Este paquete define las entidades de juego. Contiene al `Player` (controlado por el usuario) y la nueva clase `Enemy` (controlada por la IA). Ambas clases heredan de una base de colisión: `Player` hereda de `Hitbox`, mientras que `Enemy` hereda de `BallHitbox`, especializando la lógica de colisión circular.

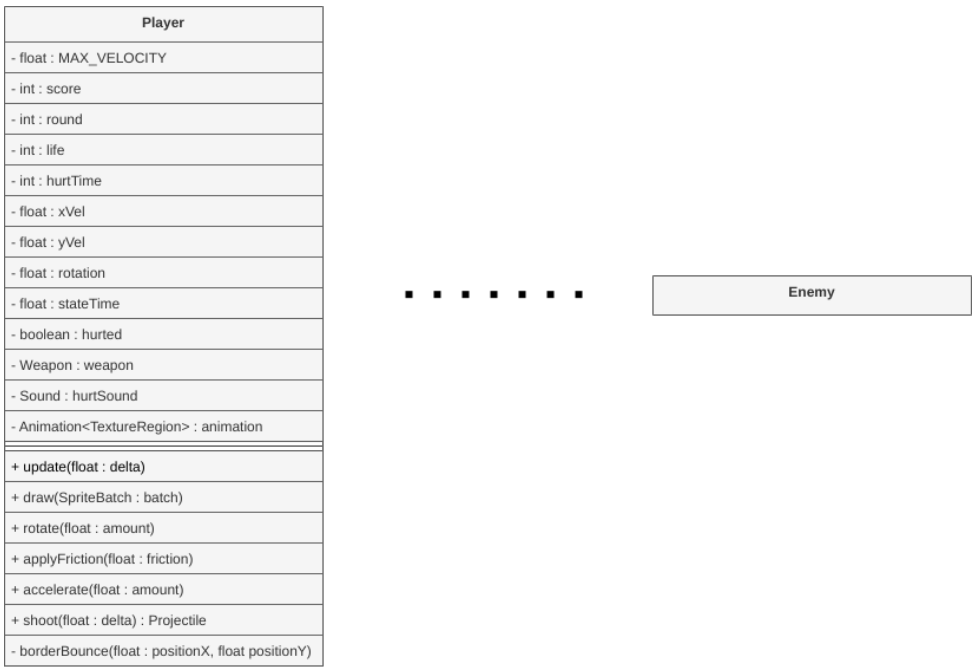


Figura 8: Diagrama de Clases UML del paquete `personajes`.

Armas y Proyectiles

La clase abstracta `Weapon` (paquete `armas`) es la base para todas las armas, incluyendo `HeavyMachineGun`, `Shotgun`, `LaserCannon`, `Melee`, etc. Estas armas crean instancias de proyectiles que heredan de `Projectile` (paquete `armas.proyectiles`), tales como `Bullet`, `LaserBeam` y `Swing`.

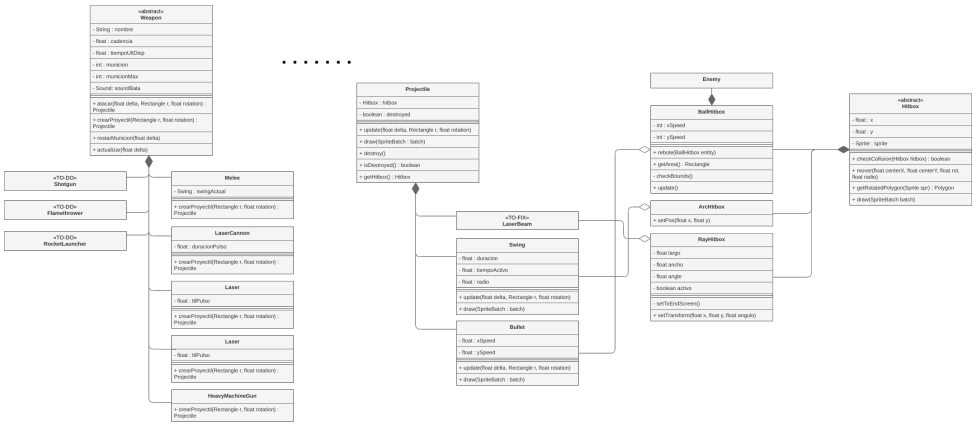


Figura 9: Diagrama de Clases UML de los paquetes `armas` y `armas.proyectiles`.

Hitboxes

Este paquete define las diferentes formas de colisión. La clase abstracta `Hitbox` sirve como base para `BallHitbox`, `RayHitbox` y `ArcHitbox`. Como se mencionó, `Player` hereda de `Hitbox`.

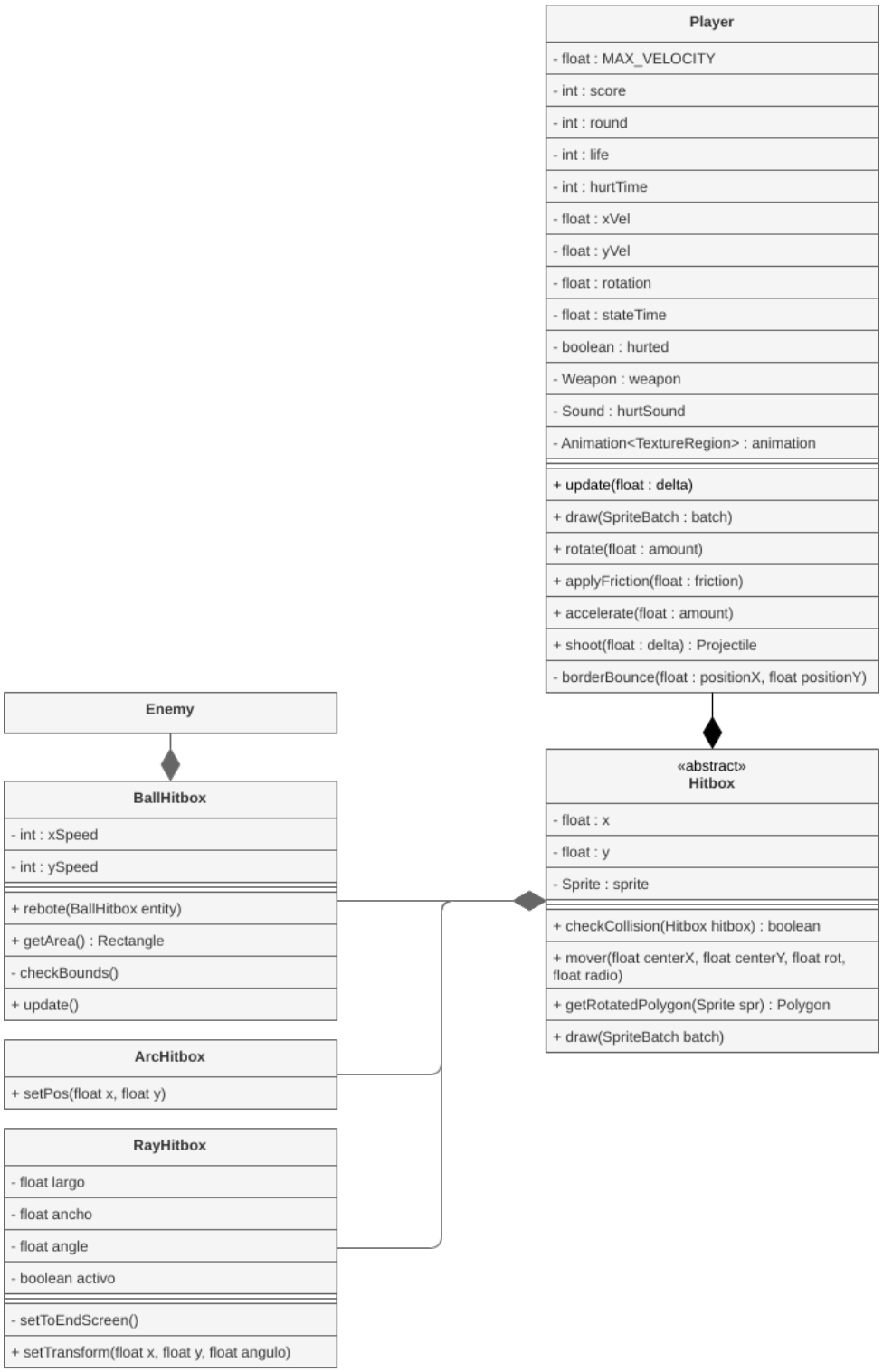


Figura 10: Diagrama de Clases UML del paquete `hitboxes`.

Enumeradores e Interfaces

Estos paquetes definen los contratos y tipos de datos clave. `interfaces` provee `INavigableOption` y `IAssetRoute`. `enumeradores` define tipos como `EScreenType`, `ESkinJugador` y las opciones de menú (`EMainMenuOption`, etc.).

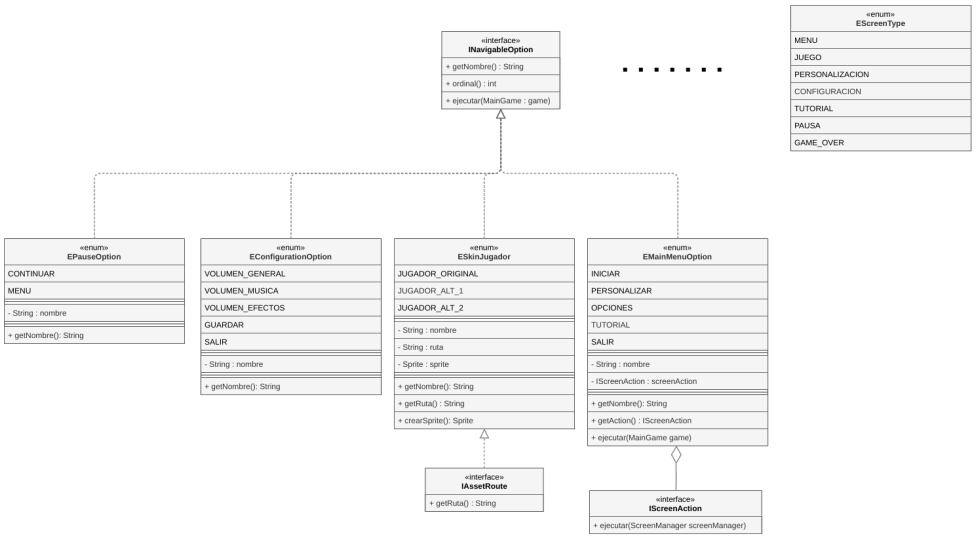


Figura 11: Diagrama de Clases UML de los paquetes `enumeradores` e `interfaces`.

Relaciones Clave del Dominio

Aquí se detallan las interacciones y dependencias fundamentales entre los diferentes componentes del sistema:

- **Composición y Estrategia**: El personaje (`Player`) **es un** `Hitbox` y **tiene una** `Weapon` (Patrón Strategy). La `Weapon` activa crea un `Projectile`.
- **Encapsulamiento**: Cada `Projectile` **tiene una** `Hitbox` específica (ej. `Bullet` usa `BallHitbox`).
- **Coordinación**: `GameLogicHandler` coordina a los `managers`, que a su vez gestionan listas de entidades (ej. `ProjectileManager` gestiona `Projectile`).
- **Navegación**: El `ScreenManager` **usa** `EScreenType` para intercambiar instancias de `BaseScreen`. Las pantallas de menú (`NavigableScreen`) **usan** un `Option-Navigator`, que gestiona un array de `INavigableOption`.

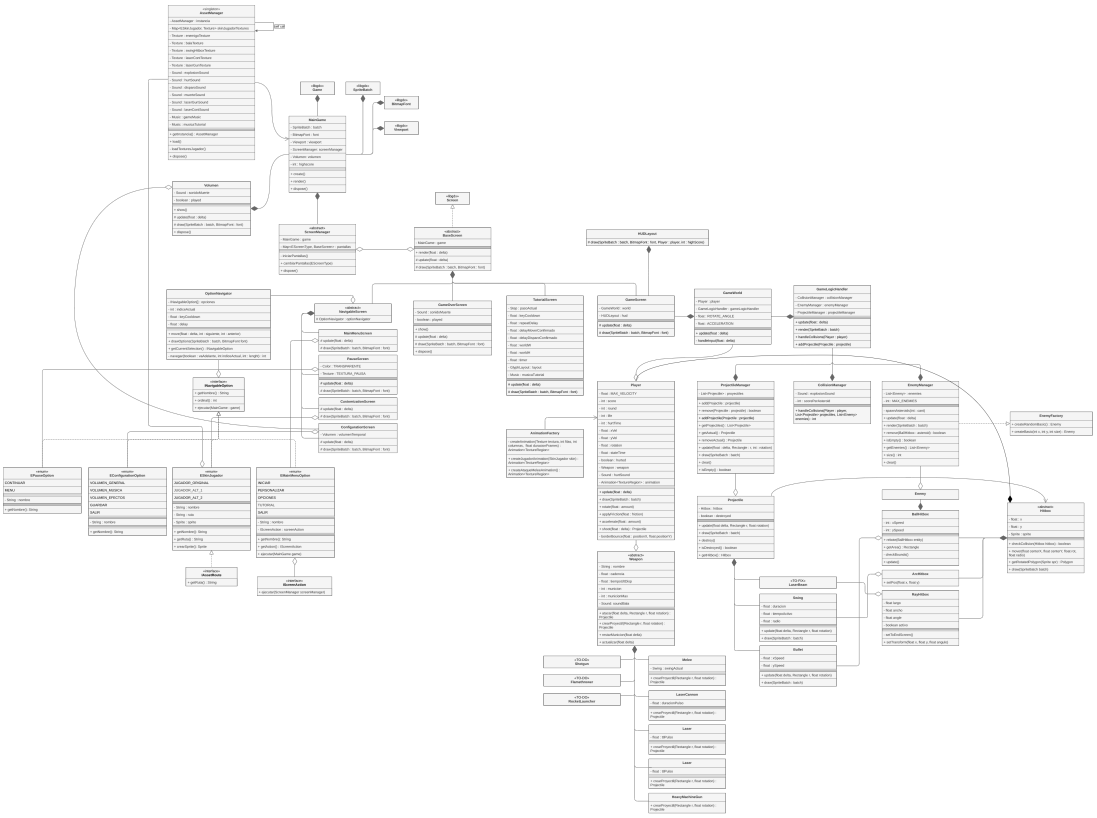


Figura 12: Diagrama UML general de la arquitectura del dominio, mostrando las relaciones clave entre paquetes.

GM1.4: Clase Abstracta

Clase Abstracta (1) `Weapon`. Define el contrato común para todos los tipos de armas, implementando la lógica compartida de cadencia y munición, y forzando a las clases hijas a implementar el método abstracto `crearProyectil`. Esto ejemplifica el **Patrón Strategy**.

Hijas (7) Se implementan múltiples subclases concretas que heredan de `Weapon`. Cada una define un comportamiento de ataque único (balas, rayo, melee) al implementar `crearProyectil` de forma diferente:

- `HeavyMachineGun`, `Melee`, `Laser`, `LaserCannon`
- (Pendientes: `Shotgun`, `Flamethrower`, `RocketLauncher`)

Contexto de uso La clase `Player` (el contexto) mantiene una referencia a `Weapon` (`private Weapon weapon`). Cuando el jugador dispara, `Player` simplemente delega la llamada al método `disparar()` de su arma activa, sin necesidad de saber qué tipo de arma es.

Clase Abstracta (2) `BaseScreen`. Define el esqueleto para todas las pantallas del juego (menús, pantalla de juego, etc.), implementando la interfaz `Screen` de LibGDX. Define el **Patrón Template Method** en su método `render()`, que llama a los métodos abstractos `update()` y `draw()` que las hijas deben implementar.

Hijas (6) Prácticamente todas las pantallas del juego heredan de esta clase, proveyendo su propia lógica de actualización y dibujado:

- `GameScreen`, `TutorialScreen`, `GameOverScreen`

- `NavigableScreen` (que a su vez es abstracta y es padre de `MainMenuScreen`, `ConfigurationScreen`, etc.)

Contexto de uso El `ScreenManager` (el contexto) gestiona un mapa de instancias de `BaseScreen`. Al llamar a `cambiarPantalla(EScreenType tipo)`, el manager intercambia la pantalla activa del juego por la instancia de `BaseScreen` correspondiente.

GM1.5: Interfaz

Interfaz `INavigableOption`. Define el contrato de una opción navegable de menú (requiere `getNombre()`, `ordinal()`, y opcionalmente `ejecutar()`), permitiendo tratar distintas opciones de forma uniforme.

Implementaciones (2) Múltiples enumeraciones implementan esta interfaz. Cada una provee su nombre visible y, en el caso de `EMainMenuOption`, una acción lambda a ejecutar:

- `EMainMenuOption`, `EConfigurationOption`
- `EPauseOption`, `ESkinJugador`

Contexto de uso `NavigableScreen` utiliza un `OptionNavigator` (el contexto) para recorrer colecciones de `INavigableOption` según el input del jugador. El contexto (ej. `MainMenuScreen`) luego llama a `opcionActual.ejecutar(getGame())` para realizar la acción correspondiente, logrando que los menús compartan mecánica sin acoplarse a tipos concretos de opción.

GM1.6: Encapsulamiento y Principios OO

Encapsulamiento El estado sensible del personaje, armas y managers se declara privado; el acceso ocurre mediante getters/setters y operaciones de alto nivel (`update`, `draw`, `handleCollisions`), evitando la exposición directa de detalles internos.

Abstracción y polimorfismo Las clases abstractas `Weapon`, `Hitbox`, `BaseScreen` y la interfaz `INavigableOption` definen contratos que permiten tratar variantes (armas, geometrías de colisión, pantallas y opciones) de forma uniforme sin cambiar el código cliente.

Responsabilidad única (SRP) Cada manager y pantalla atiende un ámbito concreto. Por ejemplo, `ProjectileManager` gestiona proyectiles, `EnemyManager` gestiona las oleadas de `Enemy`, `EnemyFactory` se encarga de su creación, `CollisionManager` las colisiones, y `ConfigurationScreen` solo los volúmenes.

Inversión de dependencias (DIP) La navegación depende de las abstracciones `INavigableOption`/`IScreenAction` en lugar de tipos concretos de opción; los recursos se obtienen vía `AssetManager` centralizado en lugar de cargas ad-hoc por clase.

Abierto/Cerrado (OCP) Se pueden agregar armas (`Weapon` hijas) o formas de colisión (`Hitbox` hijas) sin modificar `Player`, `GameWorld` ni los managers, aprovechando el polimorfismo y la composición.

GM1.7: Utilización de GitHub

Para la gestión, seguimiento y control de versiones del proyecto, se utilizó **Git** como sistema principal. Todo el código fuente se centralizó en un repositorio remoto de **GitHub**, lo cual facilitó el trabajo colaborativo y un registro riguroso de la evolución del desarrollo.

El repositorio completo, junto con el historial de cambios, está disponible públicamente en el siguiente enlace:

https://github.com/honaisu/ProyectoVideojueguil

CONCLUSIÓN

El presente Avance GM ha culminado con la implementación de un núcleo de juego funcional y una arquitectura de software robusta para el proyecto **"NOT HOTLINE MIAMI"**. Se ha establecido un bucle de juego completo (gestionado por `GameWorld` y `GameLogicHandler`) y un sistema de navegación por estados (manejado por `ScreenManager`), ambos fundamentados sólidamente en principios de Programación Orientada a Objetos.

La aplicación de la abstracción ha sido el pilar de este avance. El diseño de la clase abstracta `Weapon` (Patrón Strategy) y la clase `Projectile` permite al `Player` utilizar un arsenal variado sin acoplarse a implementaciones concretas. De forma análoga, la clase abstracta `BaseScreen` (Patrón Template Method) y la interfaz `INavigableOption` han facilitado la creación de un sistema de menús y pantallas uniforme y desacoplado.

El resultado es un sistema que cumple exitosamente con el **Principio de Abierto/Cerrado**: es posible añadir nuevas armas (hijas de `Weapon`), nuevos proyectiles (hijos de `Projectile`) y nuevas formas de colisión (hijos de `Hitbox`) sin necesidad de modificar el código central del jugador o del gestor de colisiones.

Con esta arquitectura modular ya validada, los próximos hitos se enfocarán en la expansión de contenido (balance de armas, variedad de enemigos, efectos visuales) y el pulido de la experiencia, manteniendo y aprovechando las buenas prácticas OO establecidas en este GM.