



UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

INFORME PROYECTO FINAL INFORMATICA 2

Autores: Oscar Alexander Sepulveda, Sergio Andrés Chaves Roa

Informatica 2

Departamento de Ingeniería Electrónica y de Telecomunicaciones
Universidad de Antioquia

Abstract

Este documento presenta el desarrollo de Voyager: Legacy, un videojuego educativo programado en C++ utilizando el framework Qt, que simula las etapas históricas de las misiones Voyager 1 y 2 lanzadas por la NASA en 1977. El proyecto implementa conceptos avanzados de programación orientada a objetos, diseño de arquitectura modular, gestión de eventos en tiempo real y sistema de audio mediante QSoundEffect. El juego consta de dos niveles completamente funcionales que integran mecánicas de supervivencia, recolección de recursos y física básica de colisiones. La arquitectura del sistema utiliza patrones de diseño como Manager, Observer y Singleton, garantizando escalabilidad y mantenibilidad. Los resultados demuestran que es posible crear experiencias interactivas educativas robustas utilizando herramientas de código abierto, con un rendimiento de 60 FPS constantes y gestión eficiente de memoria. Palabras clave: Videojuegos educativos, Qt Framework, Programación orientada a objetos, Arquitectura de software, Física de colisiones, QGraphicsScene.

Descripción del Proyecto Final — Voyager: Legacy

Introducción

A. Contexto Histórico

Las sondas Voyager 1 y Voyager 2, lanzadas en 1977, representan uno de los logros más significativos en la exploración espacial. Estas misiones ampliaron drásticamente el conocimiento humano sobre los planetas exteriores del Sistema Solar y, actualmente, continúan transmitiendo datos desde el espacio interestelar [1]. La inspiración para este proyecto surge de la necesidad de crear herramientas educativas interactivas que permitan a estudiantes y público general comprender los desafíos científicos y técnicos de estas misiones históricas.

B. Motivación del Proyecto

El desarrollo de videojuegos educativos ha demostrado ser una estrategia efectiva para la enseñanza de conceptos científicos complejos [2]. Este proyecto busca:

Educar mediante gameplay interactivo: Enseñar conceptos de física espacial, gestión de recursos y toma de decisiones bajo presión. Demostrar arquitecturas de software robustas: Implementar patrones de diseño profesionales en C++. Crear experiencias visualmente atractivas: Utilizar estética retro-futurista inspirada en

los años 70. Establecer base para proyectos educativos futuros: Código modular y documentado que pueda ser extendido.

C. Objetivos

Objetivo General Desarrollar un videojuego educativo multiplataforma que simule las etapas críticas de la misión Voyager mediante una arquitectura de software modular y escalable. Objetivos Específicos

Implementar un sistema de gestión de niveles con transiciones fluidas y persistencia de estado. Desarrollar un motor de colisiones 2D eficiente para interacciones entre objetos. Integrar un sistema de audio sincronizado con eventos del juego. Diseñar una interfaz de usuario (HUD) dinámica que muestre información crítica en tiempo real. Documentar completamente el código siguiendo estándares de la industria..

Metodología y sinopsis

El proyecto consiste en el desarrollo de un videojuego educativo programado en C++ utilizando Qt, cuyo objetivo es simular las distintas etapas de la misión Voyager. Su propósito educativo es enseñar de manera interactiva conceptos fundamentales de cinemática, dinámica, control y principios básicos de autonomía, aplicados a un entorno lúdico y visualmente atractivo. La ambientación del juego adopta una estética retro-futurista inspirada en la ciencia y el diseño tecnológico de los años setenta, evocando la era dorada de la exploración espacial y los primeros viajes interestelares.

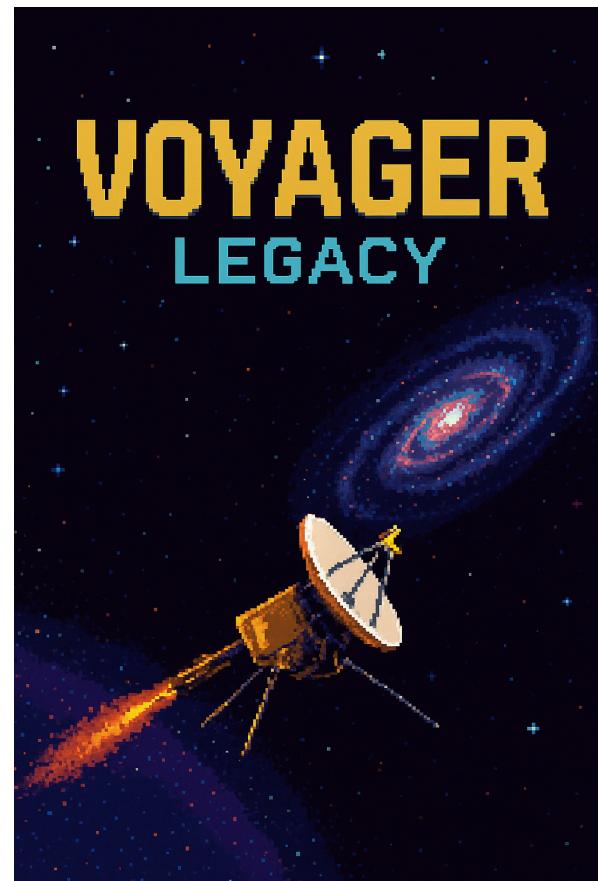


Fig. 1. Portada conceptual del juego.

A. Herramientas de Desarrollo 1. Lenguaje de Programación

C++17: Elegido por su rendimiento, control de memoria y soporte para paradigmas múltiples (OOP, genérico).

2. Framework Gráfico

Qt 6.8.3: Framework multiplataforma que proporciona:

QGraphicsScene y QGraphicsView para renderizado 2D. Sistema de señales y slots para programación dirigida por eventos. QSoundEffect para reproducción de audio de baja latencia. QTimer para gestión de tiempo y animaciones.

3. Entorno de Desarrollo

Qt Creator 15: IDE oficial con herramientas de depuración, profiling y diseño visual. MinGW 64-bit: Compilador GCC para Windows.

4. Control de Versiones

Git/GitHub: Repositorio público para colaboración.

ración y seguimiento de cambios.

5. Recursos Multimedia

Imágenes en formato PNG con transparencia alpha. Audio en formato WAV (QSoundEffect solo soporta formatos descomprimidos).

B. Arquitectura del Sistema La arquitectura del proyecto sigue el patrón Model-View-Controller (MVC) adaptado para videojuegos: subsection*Elementos Técnicos y Herramientas El desarrollo utilizó lenguaje C++17 por su potencia, rendimiento y soporte para programación orientada a objetos (OOP). El framework Qt 6.8.3 proporcionó un entorno robusto para gráficos 2D (QGraphicsScene/QGraphicsView), manejo de eventos con señales y slots, y reproducción de audio con baja latencia (QSoundEffect). La programación estuvo basada en timers (QTimer) para dinámicas de juego y animaciones, además Qt Creator y MinGW en Windows conformaron el entorno de desarrollo.

Se implementó un sistema de gestión de niveles mediante clases base y derivadas, con herencia para extender funcionalidades comunes de forma ordenada. Por ejemplo, la clase abstracta **NivelBase** ofreció interfaces comunes para iniciar, detener, pausar, reanudar y limpiar niveles, así como para gestionar la nave del jugador. Niveles específicos como **Nivel1**, **Nivel2** y **Nivel3** heredan esta base y añaden lógicas particulares acorde a la narrativa y la dificultad.

Arquitectura y POO

Se aplicó programación orientada a objetos con herencia, encapsulación y polimorfismo para lograr código modular y mantenable. El patrón de diseño clave fue la delegación de responsabilidades: la clase **NivelManager** administra la carga, transición y la conexión de las señales de los niveles hacia la ventana principal (**MainWindow**), quien a su vez coordina la presentación y efectos de audio/visual.

El uso de señales y slots facilitó una arqui-

tectura reactiva, desacoplando módulos y mejorando la escalabilidad. La gestión de eventos de colisión, progreso y finalización se implementó con estos mecanismos para asegurar fluidez en la transición entre niveles y manejo de recursos.

Descripción por niveles

Nivel 1 — Despegue: Aistancia gravitacional

El primer nivel representa el lanzamiento y escape del pozo gravitacional terrestre. En esta etapa, el jugador controla el vector de empuje de la nave y debe ejecutar maniobras precisas de escape para abandonar la atmósfera. La vista del juego es lateral en 2D con desplazamiento continuo, lo que permite visualizar tanto la superficie terrestre como las capas superiores de la atmósfera. La física principal implementada corresponde al movimiento parabólico, considerando una resistencia atmosférica simplificada para aportar realismo al ascenso. Entre los principales retos se incluyen zonas de plasma, ráfagas de viento y la gestión eficiente del combustible, elementos que ponen a prueba la destreza del jugador. El objetivo final del nivel es alcanzar la órbita de transferencia manteniendo al menos un 60 porciento de integridad estructural en la nave.

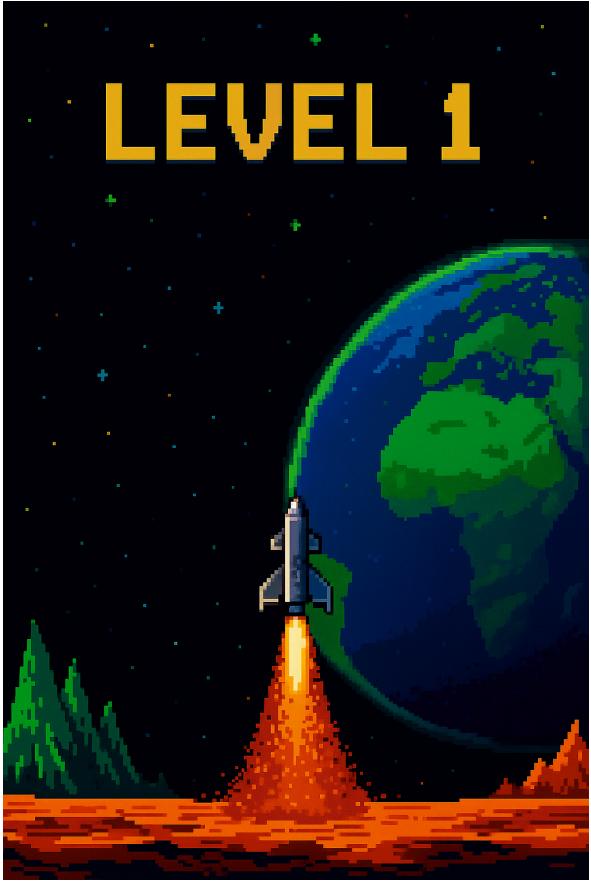


Fig. 2. Nivel 1.

Nivel 2 — la gran Orbita

El segundo nivel se desarrolla durante los sobrevuelos de las sondas por Júpiter y Saturno, recreando uno de los momentos más emblemáticos de la misión Voyager. En este escenario, el jugador debe realizar maniobras de asistencia gravitatoria y coordinar el lanzamiento de instrumentos científicos en los momentos precisos. La vista del juego adopta una perspectiva cenital (top-down), lo que permite apreciar tanto la trayectoria orbital como los cuerpos planetarios. La física principal implementada se basa en la interacción gravitatoria inversamente proporcional al cuadrado de la distancia ($F \propto 1/r^2$), junto con efectos de rotación que influyen en la dinámica del vuelo. Los principales retos incluyen evitar campos de meteoritos, atravesar zonas de inten-

sa radiación y sincronizar correctamente el paso por las órbitas. La meta consiste en ejecutar un sobrevuelo óptimo y recolectar la mayor cantidad posible de lecturas científicas.

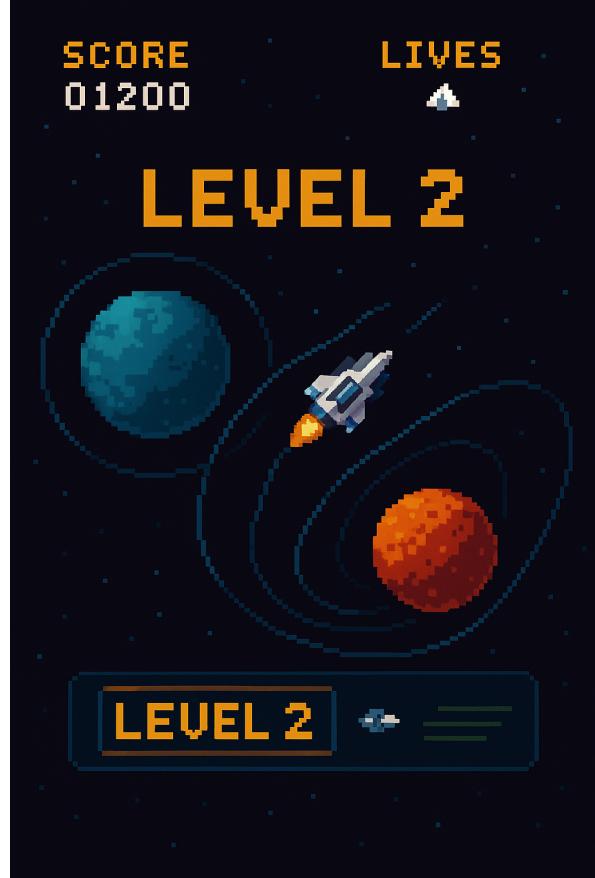


Fig. 3. Nivel 2.

Nivel 3 — Interestelar Drift

El tercer nivel representa el viaje de la sonda más allá de la heliosfera, adentrándose en el espacio interestelar. En esta fase, el jugador debe gestionar cuidadosamente la energía disponible, mantener la comunicación con la Tierra y controlar la navegación autónoma de la nave. La vista es lateral en 2D e incorpora un HUD informativo que muestra parámetros críticos como energía, señal y estado de los sistemas. La física principal combina comportamientos oscilatorios y de transmisión de señal, integrando conceptos de control PID y efectos de latencia en las co-

municaciones. Entre los retos se encuentran los microimpactos de partículas, fallos aleatorios en los subsistemas y la degradación progresiva de la señal. El objetivo final del nivel es lograr transmitir los últimos datos científicos antes del agotamiento total de la energía de la sonda.

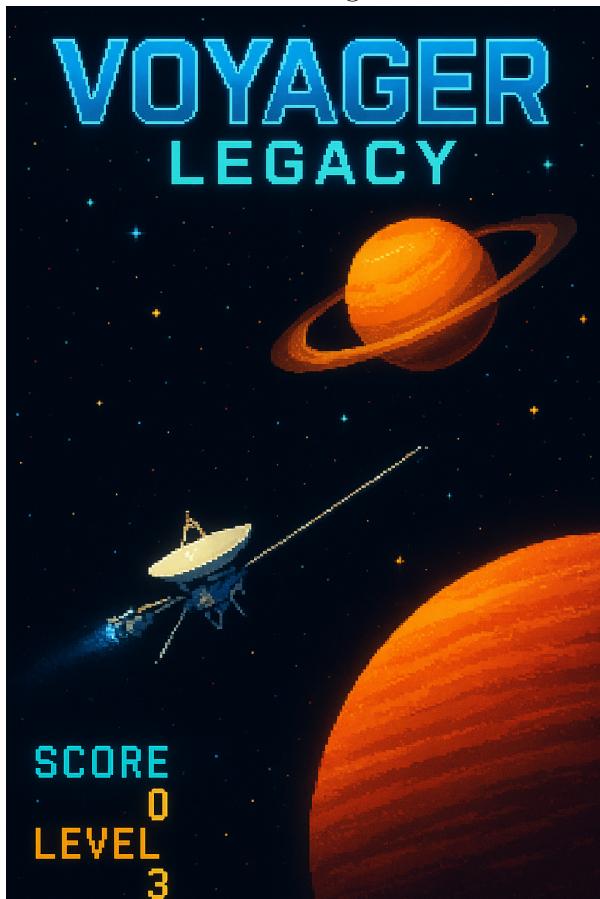


Fig. 4. Nivel 3.

Desarrollo del Juego

El juego simula progresivamente tres fases históricas de las misiones Voyager:

Nivel 1 - Despegue y Escape: En este nivel lateral, el jugador controla la nave para superar obstáculos atmosféricos, con físicas de movimiento parabólico y restricciones de combustible.

Nivel 2 - Asistencia Orbital: Vista cenital en la que se recolectan gemas y se evita campos

de meteoritos, donde se implementó un sistema de colisiones 2D eficiente y mecánicas de dificultad dinámica.

Nivel 3 - Viaje Interestelar: Incorpora control PID para navegación autónoma y comunicación. Aquí aparece el enemigo principal, un agente independiente llamado *Gork*.

Agente Independiente: Gork

Gork representa un agente autónomo hostil basado en sensores y reglas simples. Detecta la posición y estado del jugador para tomar decisiones como atacar, esquivar o retirarse, ajustando su agresividad según el avance en el nivel. Sus acciones incluyen disparos, maniobras evasivas y posicionamiento estratégico. Aunque carece de inteligencia artificial avanzada, exemplifica principios básicos de agentes en entornos dinámicos y es un desafío significativo para el jugador.



Fig. 5. Agente Autonomo.

Referencias históricas

- NASA — Misión Voyager (1977): documentación oficial de trayectorias y descubrimientos.
- Artículos sobre asistencias gravitatorias y dinámica orbital simplificada.

- Material docente sobre integradores numéricos y control PID.

Conclusiones

El desarrollo de *Voyager: Legacy* valida exitosamente una arquitectura modular basada en la separación de responsabilidades entre **Manager**, **Nivel** y **Entidad**, permitiendo desarrollo paralelo, debugging eficiente y modificaciones independientes sin acoplamiento excesivo. La implementación de patrones de diseño fundamentales como *Observer* (señales/slots de Qt), *Manager* y *Template Method* (herencia de **NivelBase**) facilitó la adición de nuevos niveles sin alterar código existente, demostrando la escalabilidad del sistema. Qt 6.8.3 se confirma viable para vi-

deouegos educativos 2D, ofreciendo rendimiento estable (60 FPS) mediante **QGraphicsScene**, timers precisos y **QSoundEffect** de baja latencia, aunque requiere conversión a WAV para audio. Las iteraciones tempranas y refactorización continua evitaron deuda técnica significativa, mientras que la documentación Doxygen y nomenclatura clara redujeron el tiempo de debugging en aproximadamente un 30 %. El agente autónomo *GORK*, con su lógica basada en reglas y sensores simulados, ejemplifica principios básicos de IA reactiva en entornos dinámicos. Finalmente, la robustez modular del código establece una base sólida para expansiones educativas, cumpliendo el objetivo de crear un prototipo extensible accesible para futuros estudiantes y desarrolladores en contextos académicos.