Carrera Espacial

RESUMEN Este documento presenta el diseño, desarrollo e implementación de un videojuego 2D titulado “Carrera Espacial”, realizado en C++ utilizando el framework Qt. El juego se estructura en tres niveles con mecánicas distintas: exploración y recolección de ítems, carrera con nave espacial y juego de disparos contra oleadas de enemigos. El proyecto se desarrolló con un enfoque orientado a objetos, utilizando herencia, composición y manejo de escenas gráficas mediante QGraphicsScene y QGraphicsView. Se describen la motivación, los requisitos, la arquitectura del sistema, la lógica interna, los procedimientos de prueba y los resultados obtenidos. Finalmente, se discuten las principales limitaciones y se proponen mejoras para trabajos futuros.

Palabras clave: videojuegos 2D, Qt, C++, programación orientada a objetos, Informática 2.

1. INTRODUCCIÓN El proyecto “Carrera Espacial” consiste en un videojuego 2D desarrollado en C++ con la biblioteca Qt, como trabajo académico para el curso Informática 2 de la Universidad de Antioquia. El juego combina diferentes mecánicas clásicas de los videojuegos tipo arcade (exploración, carrera y disparos) con el objetivo de integrar los conceptos vistos en clase, especialmente la programación orientada a objetos y el manejo de interfaces gráficas. El juego está compuesto por una ventana principal con menú y tres niveles independientes pero conectados entre sí mediante un sistema de desbloqueo progresivo. Cada nivel presenta reglas propias, metas específicas y elementos visuales diferenciados, lo que permite explorar diversas formas de interacción con el usuario.

2. CONTEXTO DEL PROYECTO, MOTIVACIÓN Y PÚBLICO OBJETIVO El proyecto se sitúa en el contexto académico de un curso universitario de programación, en el cual se busca pasar de ejercicios aislados a un software más completo, integrando clases, herencia y polimorfismo, manejo de eventos y señales/slots en Qt, y uso de escenas y elementos gráficos. La motivación principal es contar con un proyecto que sea tanto formativo como entretenido, aplicar conceptos teóricos en un caso práctico más cercano al desarrollo real de videojuegos, y desarrollar habilidades de diseño, análisis y resolución de problemas en un entorno interactivo. El público objetivo incluye estudiantes de programación que deseen entender la integración de C++ y Qt en un juego, así como usuarios interesados en juegos 2D simples, con partidas cortas y dificultad creciente.

3. PLANTEAMIENTO DEL PROBLEMA En muchos cursos de programación los ejercicios son fragmentados y carecen de un producto final integrador. El problema que se aborda es: ¿Cómo desarrollar un videojuego 2D didáctico, modular y extensible, que permita aplicar los conceptos de programación orientada a objetos y el uso de Qt en un

contexto práctico? Se requiere un juego que tenga varias mecánicas, sea modular, de forma que cada nivel esté encapsulado, y permita futuras extensiones (nuevas naves, enemigos, niveles, etc.).

4. NECESIDAD QUE CUBRE EL SOFTWARE Y JUSTIFICACIÓN DEL DESARROLLO El proyecto cubre la necesidad de disponer de un caso de estudio completo donde se integren diseño de clases con herencia (Personaje, Jugador, Enemigo, NaveEspacial), manejo de sprites, colisiones, temporizadores y eventos de teclado, y una arquitectura con una ventana principal que coordina escenas de juego. La justificación del desarrollo es tanto académica como práctica: permite afianzar contenidos del curso a través de un proyecto motivador, genera un software demostrable (se puede jugar y mostrar) y sirve como base para futuros proyectos de mayor complejidad.

5. DEFINICIÓN GENERAL Y OBJETIVOS 5.1 Definición general del software "Carrera Espacial" es un videojuego 2D por niveles cuyo flujo general es: Nivel 1: un personaje recorre un mapa recolectando cohetes mientras evita enemigos y gestiona un tiempo límite. Nivel 2: una nave espacial avanza lateralmente esquivando obstáculos y meteoritos en un escenario desplazable. Nivel 3: el jugador controla una mira y dispara a aliens que descienden en oleadas desde la parte superior de la pantalla.

5.2 Objetivo general Diseñar e implementar un videojuego 2D modular en C++/Qt que combine distintas mecánicas de juego y permita aplicar de forma integrada los conceptos de programación orientada a objetos, manejo de escenas y eventos.

5.3 Objetivos específicos Implementar una jerarquía de clases basada en una clase base Personaje. Desarrollar tres niveles jugables con mecánicas diferenciadas. Integrar un menú principal que gestione el acceso y desbloqueo progresivo de niveles. Implementar la lógica de colisiones, temporizadores y estados de victoria/derrota.

6. ALCANCE FUNCIONAL, METAS Y REQUISITOS PRINCIPALES 6.1 Alcance funcional El videojuego incluye: Un menú principal con opciones de acceso a Nivel 1, 2 y 3. Tres niveles jugables con jugador y/o nave controlables por teclado, elementos de entorno (obstáculos, meteoritos, aliens, cohetes), y condiciones de victoria y derrota definidas. Sprites de fondo y pantallas de "WIN" y "GAME OVER".

6.2 Metas principales Conseguir que los tres niveles sean jugables y funcionales. Lograr un rendimiento aceptable y fluido. Mantener una organización del código clara y extensible.

7. ESPECIFICACIÓN DE REQUERIMIENTOS 7.1 Requisitos funcionales RF1. Menú principal: la aplicación debe iniciar mostrando un menú con botones para cada nivel. RF2. Desbloqueo de niveles: el Nivel 2 solo se habilita tras ganar el Nivel 1; el Nivel 3 solo tras ganar el Nivel 2. RF3. Nivel 1: debe permitir mover al jugador, recolectar cohetes, evitar enemigos y terminar el nivel por victoria o derrota. RF4. Nivel 2: debe

permitir controlar la nave, generar obstáculos y meteoritos, y terminar por victoria o por colisión. RF5. Nivel 3: debe permitir mover la mira, disparar a aliens, gestionar oleadas y terminar por victoria o por exceso de aliens escapados. RF6. Mensajes de estado: al finalizar un nivel se debe mostrar una pantalla de “WIN” o “GAME OVER”.

7.2 Requisitos no funcionales RNF1. Rendimiento: el juego debe actualizarse de forma fluida, mediante temporizadores periódicos. RNF2. Usabilidad: los controles se realizan únicamente con el teclado, de forma intuitiva (WASD/flechas y barra espaciadora). RNF3. Mantenibilidad: la lógica de cada nivel y de cada tipo de personaje debe estar encapsulada en clases separadas.

7.3 Restricciones Uso obligatorio de C++ y Qt. Resolución fija de ventana y escena de juego. Control únicamente con teclado.

7.4 Dependencias Biblioteca Qt (widgets, escenas gráficas, temporizadores). Archivo de recursos de sprites (imágenes de fondos, personajes, etc.).

7.5 Usuarios destinatarios Estudiantes de programación de nivel intermedio. Usuarios generales que deseen jugar un minijuego arcade sencillo.

8. METODOLOGÍA Y PLANIFICACIÓN Se adoptó una metodología iterativa e incremental, similar a un enfoque ágil sencillo: Fase 1 – Configuración del proyecto: creación de la ventana principal (MainWindow), escena y vista. Fase 2 – Nivel 1: diseño del mapa, creación del jugador, enemigos, cohetes y lógica de tiempo/colisiones. Fase 3 – Nivel 2: implementación de la nave espacial, generación aleatoria de obstáculos y meteoritos, y desplazamiento del escenario. Fase 4 – Nivel 3: creación de la mira, aliens en oleadas y lógica de disparos. Fase 5 – Integración: conexión de señales entre niveles y menú, y pulido visual (sprites de estados, fondos).

9. DISEÑO Y ARQUITECTURA DEL SISTEMA 9.1 Arquitectura general La arquitectura se organiza en tres bloques: Interfaz y flujo principal: MainWindow, que contiene el QGraphicsView y gestiona el menú. Escenas de juego: clases Nivel (para niveles 1 y 3) y Nivel2 (para el segundo nivel). Entidades del juego: jerarquía derivada de Personaje: Jugador, Enemigo, NaveEspacial, además de Obstaculo y otros elementos.

9.2 Principales componentes Personaje: clase base que agrupa propiedades de posición, velocidad y aceleración, y funciones para actualizar el movimiento. Jugador: personaje del nivel 1 controlado por teclado, con animaciones según dirección y detección de colisiones con cohetes y obstáculos. Enemigo: enemigo con IA sencilla de persecución (nivel 1) y variante de alien descendente (nivel 3). NaveEspacial: personaje del nivel 2 que avanza horizontalmente y responde al movimiento vertical y a la aceleración. Obstaculo: elemento gráfico que representa objetos estáticos del escenario o meteoritos. Nivel: escena que contiene y administra todos los elementos necesarios para los niveles 1 y 3. Nivel2: escena específica para el nivel de carrera con

la nave espacial. MainWindow: administra los cambios de escena, el menú y la lógica de desbloqueo de niveles.

9.3 Flujo de clases y navegación El programa inicia creando MainWindow y mostrando el menú. Al seleccionar un nivel, MainWindow crea la escena correspondiente (Nivel o Nivel2) y la muestra en el QGraphicsView. La escena administra su propio temporizador, manejo de colisiones y estado interno. Al terminar, emite señales de victoria o derrota que MainWindow escucha para actualizar el menú y desbloquear niveles.

10. DEPENDENCIAS EXTERNAS Y BIBLIOTECAS UTILIZADAS Qt: QMainWindow, QGraphicsScene, QGraphicsView, QGraphicsPixmapItem, QTimer, QKeyEvent, entre otros. C++ estándar: tipos básicos, contenedores y funciones matemáticas para manejo de posiciones y velocidades. Recursos gráficos: imágenes de sprites, fondos, cohetes, aliens, meteoritos y pantallas de victoria/derrota, gestionadas con un archivo de recursos de Qt.

11. DESARROLLO E IMPLEMENTACIÓN 11.1 Lógica del Nivel 1 Creación del mapa con obstáculos y cohetes. El Jugador se desplaza con el teclado y recolecta cohetes al colisionar con ellos. Uno o varios Enemigo persiguen al jugador; si lo alcanzan, el nivel termina en derrota. Un temporizador controla el tiempo restante; si el tiempo se agota, también se pierde. Si se recolectan todos los cohetes, se muestra la pantalla de victoria y se desbloquea el siguiente nivel.

11.2 Lógica del Nivel 2 La NaveEspacial se mueve automáticamente hacia la derecha. El jugador controla el movimiento vertical y puede acelerar horizontalmente. Se generan obstáculos estáticos y meteoritos con cierta aleatoriedad para aumentar la rejugabilidad. Las colisiones con obstáculos o meteoritos terminan el nivel en “GAME OVER”. Si la nave llega al final del escenario o supera el tiempo objetivo sin colisionar, se considera victoria.

11.3 Lógica del Nivel 3 Se dibuja una mira que el jugador mueve en la pantalla con el teclado. Los aliens descienden desde la parte superior en oleadas. El jugador dispara; si el disparo coincide con un alien, este es eliminado. Si un número máximo de aliens atraviesa la línea inferior, el jugador pierde. Al eliminar suficientes oleadas, se muestra la pantalla de victoria.

12. PROCEDIMIENTOS DE PRUEBA 12.1 Pruebas funcionales Verificación de movimientos del jugador y la nave en cada nivel. Comprobación de que las colisiones producen los efectos esperados (derrota, recolección de ítems). Confirmación de que el desbloqueo de niveles se cumple en el orden correcto.

12.2 Pruebas de integración Interacción entre MainWindow y las escenas Nivel y Nivel2. Correcto cambio de escena al iniciar y al finalizar cada nivel. Reutilización de la ventana principal sin reinicios forzados de la aplicación.

12.3 Pruebas de estrés básicas Pruebas de ejecución continua durante varios ciclos de juego para verificar que no se produzcan errores graves ni caídas de rendimiento.

13. GUÍA DE INSTALACIÓN Y USO 13.1 Requisitos técnicos Sistema operativo compatible con Qt (Windows, Linux, etc.). Qt Creator y compilador C++ (MinGW, GCC u otro). Proyecto configurado con archivo .pro y recursos (.qrc).

13.2 Instalación Abrir Qt Creator. Cargar el archivo del proyecto. Verificar que las rutas de los recursos gráficos sean correctas. Compilar el proyecto en modo Debug o Release.

13.3 Ejecución y uso Ejecutar el programa desde Qt Creator o desde el ejecutable generado. En el menú principal, seleccionar el Nivel 1 para comenzar. Controles típicos: movimiento con W, A, S, D o flechas, disparo con barra espaciadora en el Nivel 3. Tras ganar un nivel, el siguiente se desbloquea automáticamente.

14. RESULTADOS Y DISCUSIÓN El juego “Carrera Espacial” cumple con el objetivo de integrar múltiples mecánicas de juego en un solo proyecto, aplicar conceptos de programación orientada a objetos en un contexto práctico y mostrar el potencial de Qt como herramienta para desarrollar videojuegos 2D sencillos. Entre los resultados positivos se encuentran niveles jugables y funcionales, navegación correcta entre menú y niveles y uso efectivo de la herencia y la modularidad. Entre las limitaciones se encuentran la ausencia de sistema de puntuación global o tabla de récords, una IA de enemigos relativamente simple y la falta de opciones avanzadas (configuración de dificultad, sonido, etc.). Como propuestas de mejora se plantea incorporar un sistema de puntuación, vidas y guardado de récords, añadir efectos de sonido y música de fondo, implementar patrones de movimiento más complejos para los enemigos e incluir nuevos niveles o jefes finales.

15. CONCLUSIONES El desarrollo de este proyecto permitió reforzar y aplicar conocimientos de C++ y programación orientada a objetos en un contexto completo, conocer el uso de Qt para el manejo de interfaces gráficas, escenas y eventos, y entender la importancia de una arquitectura modular para facilitar extensiones y mantenimiento. En conclusión, “Carrera Espacial” no solo cumple con los requisitos del curso, sino que también sirve como base para futuros desarrollos de videojuegos más complejos, motivando el aprendizaje autónomo y el interés por el desarrollo de software interactivo.

16. REFERENCIAS Documentación oficial de Qt. Apuntes y material del curso Informática 2 – Universidad de Antioquia. Referencias generales de C++ y orientación a objetos.

17. ANEXOS (SUGERIDOS) Anexo A. Capturas de pantalla de cada nivel y del menú principal. Anexo B. Diagrama de clases simplificado del sistema. Anexo C. Fragmentos de código relevantes (por ejemplo, la clase Personaje y la lógica de generación de meteoritos), con comentarios explicativos.