Proyecto Final - Informática II

Santiago Londoño Giraldo  
Julián Andrés Sánchez Ballesteros

Universidad de Antioquia.

10 de Julio de 2025

# Resumen

El presente documento expone el desarrollo de un videojuego 2D implementado en C++ utilizando el framework Qt, como parte del proyecto final del curso Informática II. El objetivo principal fue aplicar conceptos fundamentales de la programación orientada a objetos (POO), como herencia, polimorfismo, encapsulamiento y modularidad, integrados con los recursos gráficos que proporciona Qt. El juego cuenta con 2 niveles, cada uno con mecánicas diferentes, el nivel1 presenta una mecánica tipo plataforma, en el que el jugador tiene que ir esquivando diversos obstaculos, e intentar recoger los objetos que salen en pantalla, el nivel 2 tiene una dependecia del 1 en el sentido de que en el primero se recoge la “leche” que son las curas para subir la vida, ademas de los lanzables “esferas”, este nivel presenta una mecánica más de un juego tipo arcade en el que hay un jefe que tiene mecánicas y el objetivo es derrotarlo.

# 1. Introducción

Este videojuego fue concebido en base al capítulo 23 de Dragon Ball y se desarrolla entorno al lore(historia) del anime,A lo largo del documento se describe la arquitectura del sistema, las clases involucradas y las decisiones técnicas que guiaron su implementación.

# 2. Metodología y Arquitectura del Sistema

El sistema está organizado en torno a la clase base personaje, de la cual heredan jugador y jefe. Se implementan niveles (nivel1, nivel2), obstáculos (obstáculos), ítems (recolectables) y una clase central (MainWindow) que gestiona las escenas y la interacción con el usuario. La clase menu gestiona la pantalla inicial.  
  
Todas las entidades gráficas usan QGraphicsPixmapItem para aprovechar la integración con la escena de Qt.

# 3. Clases principales y justificación de diseño

## 3.1 Clase personaje

La clase personaje es una clase base que encapsula propiedades comunes como posición, velocidad, salud y tamaño. Se usaron tipos short int y float para balancear precisión y eficiencia. El uso de QTimer\* permite temporizar acciones como movimiento o animaciones. La herencia de QGraphicsPixmapItem facilita su visualización directa en la escena.

## 3.2 Clase jugador

jugador hereda de personaje y agrega funciones específicas como ataqueBasico, ataqueEspecial y gestión de teclas presionadas a través de un QSet<int> (eficiente para operaciones de búsqueda y eliminación sin duplicados). Se utilizan QVector<QPixmap> para animaciones, ya que QVector ofrece compatibilidad nativa con Qt y eficiencia en acceso aleatorio.

## 3.3 Clase jefe

La clase jefe también hereda de personaje y posee tres temporizadores para animaciones y ataques, uno de ellos y el que representa un movimiento físico es el de “explosión” que es un movimiento relativo es algo similar a doppler, dependiendo la distancia a la que el jugador perciba el impacto se le hace daño. Se eligió QVector<QPixmap> para manejar sprites debido a su rendimiento en acceso secuencial y compatibilidad con las funciones gráficas de Qt.

## 3.4 Clase obstaculos

Los obstaculos representan elementos del entorno que afectan al jugador o al jefe. Se utilizan múltiples funciones para representar diferentes trayectorias (recta, parabólica, senoidal). La elección de QVector<QPixmap> permite reutilizar animaciones fácilmente, y las banderas booleanas como disparando o disponible permiten manejar su ciclo de vida sin costos adicionales de memoria.

## 3.5 Clase recolectables

recolectables representa objetos interactivos del juego. Incluye referencias al `jugador` para detectar colisiones y aplicar efectos. Se usan tipos como `unsigned short int` para representar valores positivos como aportes sin desperdiciar memoria. El uso de timers y sprites permite animar objetos como esferas y leche.

## 3.6 Clases nivel1 y nivel2

Cada nivel gestiona sus propios obstáculos, recolectables y enemigos. Se usan QVector<obstaculos\*> para almacenar múltiples entidades con acceso rápido, ademas para evitar la creacion innecesario de objetos se rehusan constantemente. Se incluyen barras de salud mediante QProgressBar, incrustadas en la escena con QGraphicsProxyWidget. Esto facilita una visualización intuitiva del estado del jugador y jefe.

## 3.7 Clase menu

La clase menu actúa como pantalla inicial. Los botones se manejan como QPushButton y se colocan en la escena mediante QGraphicsProxyWidget. El personaje animado roshi usa un QTimer para recorrer QVector<QPixmap> con fines decorativos.

## 3.8 Clase MainWindow

MainWindow gestiona el cambio entre escenas (nivel1, nivel2, menu) y el ciclo principal del juego. El uso de punteros a cada escena y objeto permite mantener una arquitectura modular y clara. El sistema de entrada usa eventos keyPressEvent y keyReleaseEvent, que modifican los estados del jugador según la tecla.

# 4. Cambios en medio del desarrollo

En los primeros momentos, no había claridad de cómo manejar el juego aquí tomamos la decisión de manejarlo en el mainWindow, además agregamos una clase del menú que permite que el juego se repita las veces que se desee,agregamos una clase para cada nivel para que dicha clase gestione la interacciones dentro su nivel,así como los objetos que le perteneces a cada uno.

Se profundizó en la interacción entre clases, declarando que el jugador pertenece al juego(mainwindow), y las demás instancias a cada nivel, así se clarifica que existe en qué momento y que no.

El mayor problemas que presentamos en desarrollo fue la elección de cuándo eliminar los objetos, y pues de manera más pequeña el manejo de los sprites.

# 5. Abstracción de la dificultad

La dificultad se maneja desde la concepción del tipo de juego, al ser lineal(consecutivo entre niveles) damos una dificultad de que tiene que tener la habilidad para superar el primer nivel para poder acceder al segundo nivel, para hacer mas dinámico este cambio el primer nivel maneja unas mecánicas bastante predecibles como movimiento recto o senoidal, ademas de una velocidad lenta para poder pasarlo sin recibir una gran cantidad de daño, el segundo nivel maneja una dificultad mayor en dos tiempos, primera fase: aquí el jefe se mueve de izquierda a derecha y en los bordes lanza proyectiles y golpea al jugador, cuando percibe muchos impactos genera una “explosión” (esto para evitar que el jugador se pegue a matarlo), si el jugador se encuentra muy cerca lo puede matar el propio impacto, cuando el jefe llega a la mitad de su vida se activa la segunda fase:en esta fase tiene un movimiento mas “errático”, salta cada cierto tiempo, lanza proyectiles. Explota al caer del salto, además también golpea.

# 7. Conclusiones

El proyecto permitió poner en práctica los principios de la programación orientada a objetos y el uso del framework Qt para aplicaciones gráficas. El uso de contenedores eficientes como QVector y QSet, así como el aprovechamiento de QGraphicsScene y sus derivados, permitió desarrollar un juego fluido, modular y visualmente interactivo.