Estudiante:
Juan José Medina Mejía

Institución:
Universidad de Antioquia

Desafío Final:
Juego Dragon Ball

Curso:
Informática 2

1. Introducción

Este informe técnico detalla el desarrollo del juego "Desafío Final: Dragon Ball", un proyecto de plataformas y aventura. El objetivo principal fue crear una experiencia de juego interactiva y fluida, centrado en una implementación robusta de mecánicas de juego a través de Qt. El informe aborda la arquitectura del código, el uso de las librerías clave de Qt, la implementación de las físicas del juego y los procesos de prueba.

Índice

8. Imágenes del juego

9. Como se implementó el sonido?

10. Como se juega?

2. Arquitectura y Componentes del Código

El diseño del juego sigue una arquitectura modular, permitiendo una clara separación de responsabilidades y facilitando el mantenimiento y la escalabilidad. Los componentes principales se estructuran en varias clases fundamentales:

MainWindow: Actúa como el contenedor principal de la aplicación. Gestiona el QStackedWidget para alternar entre el MainMenu y los diferentes niveles (Nivel1, Nivel2, Nivel3). Es responsable de la configuración inicial de la ventana y de la gestión del ciclo de vida de los niveles.

Librerías Qt clave: QMainWindow, QStackedWidget, QScreen, QGuiApplication.

MainMenu: Es la interfaz de usuario para la selección de niveles. Utiliza un QVBoxLayout para organizar los botones y etiquetas, y un QLabel para el fondo.

Librerías Qt clave: QWidget, QVBoxLayout, QLabel, QPushButton, QPainter, QPixmap. Destaca el uso de QPainter para el escalado y centrado del fondo, lo que demuestra un manejo avanzado de QPixmap.

Clases de Nivel (Nivel1, Nivel2, Nivel3): Cada clase representa un nivel específico del juego, heredando de QGraphicsView. Son las encargadas de:

Configurar la escena (QGraphicsScene).

Cargar los recursos gráficos (QPixmap) para fondos, plataformas y personajes.

Crear y posicionar las plataformas (usando QGraphicsRectItem o QGraphicsPixmapItem).

Instanciar y gestionar la lógica de Jugador y Enemigo.

Implementar timers (QTimer) para la actualización periódica de la lógica del juego.

Gestionar las interacciones específicas del nivel (por ejemplo, trampolines, objetos empujables, sensores de gravedad).

Librerías Qt clave: QGraphicsView, QGraphicsScene, QGraphicsRectItem, QGraphicsPixmapItem, QTimer.

Jugador (Goku): Esta clase, que hereda de QObject y QGraphicsPixmapItem, encapsula toda la lógica y el estado de Goku.

Manejo de entrada del usuario (keyPressEvent).

Lógica de movimiento, salto y aplicación de gravedad.

Detección de colisiones con plataformas y enemigos.

Gestión de sprites para diferentes estados (caminar, saltar, empujar).

Manejo del modo empuje y la orientación (mirandoDerecha).

Librerías Qt clave: QObject, QGraphicsPixmapItem, QKeyEvent, QVector, QRectF.

Enemigo: Hereda de QObject y QGraphicsPixmapItem. Implementa un comportamiento de movimiento horizontal simple, invirtiendo la dirección al alcanzar límites.

Librerías Qt clave: QObject, QGraphicsPixmapItem, QTimer.

La comunicación entre estos componentes se logra a través del mecanismo de señales y slots de Qt, un patrón de diseño fundamental que permite que los objetos se comuniquen de manera desacoplada. Por ejemplo, Jugador::solicitarMenu es una señal que MainWindow conecta para cambiar la vista al menú principal.

3. Implementación de Físicas Clave

Las mecánicas físicas del juego se simulan utilizando una aproximación por pasos discretos, donde los cambios en la velocidad y la posición se calculan en pequeños intervalos de tiempo (cada frame). Este método, aunque simple, emula eficazmente las leyes del movimiento sin necesidad de resolver ecuaciones diferenciales complejas.

Se emplearon tres formulaciones físicas principales:

Cinemática Vertical con Gravedad:

Aplicación: Movimiento de Goku (salto y caída) y los bloques empujables.

Fórmula discreta:

$$v_y = v_y + g$$
$$y = y + v_y$$

Explicación en código: En los métodos Jugador::mover y Jugador::moverConGravedad (para el Nivel 3), la velocidadY se incrementa con el valor de la gravedad en cada actualización, y luego la posición y del objeto se ajusta con esa nueva velocidad. Esto simula la aceleración constante que causa la caída libre. El valor de gravedad es una variable que puede ser manipulada, como se ve en el Nivel 3.

Impulso Inicial para el Salto:

Aplicación: Goku salta y se impulsa en trampolines.

Fórmula base:

$$v_y = -v_0$$

Explicación en código: Al presionar la tecla de salto (Qt::Key_Space en Jugador::keyPressEvent), la velocidadY de Goku se establece a un valor negativo (-12). Esto lo impulsa hacia arriba. Para los trampolines (Jugador::boostJump), se añade una extraForce a este valor inicial (-12 - extraForce), creando un salto más alto. La condición if (goku->getVelocidadY() > -1) en Nivel2::actualizar asegura que el impulso del trampolín solo se aplique si Goku está cayendo o en reposo, evitando saltos infinitos.

Movimiento Rectilíneo Uniforme (Plataformas y Empuje):

Aplicación: Desplazamiento horizontal de las plataformas móviles y el movimiento de los bloques cuando son empujados por Goku.

Fórmula:

$$x = x + v$$

$$y = y + v$$

Explicación en código:

Plataformas Móviles (Nivel2::moverPlataformas): Las plataformas se mueven a una velocidad (velHor, velVer) constante. Cuando alcanzan límites predefinidos (platHor->x() <= 200 || platHor->x() >= 600 para horizontal, y límites en Y para vertical), su velocidad se invierte (*=-1), creando un movimiento pendular. La plataforma vertical tiene una lógica avanzada: solo sube si Goku o un bloque están sobre ella y regresa a su posición inicial si no hay carga.

Empuje de Objetos (Nivel2::keyPressEvent y Nivel2::actualizar): Cuando Goku está en modoEmpuje (activado por Qt::Key_S) y colisiona con un bloque, el bloque se desplaza horizontalmente una cantidad fija (VELOCIDAD_EMPUJE = 3.0) en la dirección en que Goku mira. item->moveBy(dx, 0);.

3.1. Detección de Colisiones

La colisión de Goku con las plataformas para detectar el "apoyo" se realiza verificando la intersección de un pequeño rectángulo en los "pies" de Goku con la parte superior de la plataforma (Jugador::isOnPlatform). Es crucial que esta colisión solo se registre si Goku está cayendo (velocidadY >= 0), evitando que se "pegue" a la parte inferior de las plataformas al subir.

C++

```
// Lógica clave de colisión en Jugador::isOnPlatform
return feetRect.intersects(platformTop) && velocidadY >= 0;
```

3.2. Cambio de Gravedad Dinámico

En el Nivel 3 (Nivel3::verificarSensores), la gravedadActual se modifica en tiempo real. Al entrar en un "sensor" (QGraphicsEllipseItem), la gravedad puede volverse inversa (-0.3), haciendo que Goku "flote" hacia arriba. Al salir del sensor, la gravedad vuelve a su valor normal (0.5). Esto demuestra la capacidad del motor de físicas para adaptarse a diferentes condiciones ambientales del juego.

4. Pruebas y Resultados

Las pruebas se centraron en validar la funcionalidad de cada componente del código y la coherencia de las físicas implementadas.

4.1. Tipos de Pruebas Realizadas

Pruebas Unitarias (implícitas): Verificación de funciones individuales como Jugador::boostJump o Jugador::isOnPlatform en escenarios controlados.

Pruebas de Integración: Comprobación de la interacción entre Jugador, Plataformas, Enemigos y Objetos Empujables dentro de cada nivel.

Pruebas Funcionales: Evaluación de la experiencia de juego general, asegurando que las mecánicas se comportan como se espera y son intuitivas para el jugador.

4.2. Casos de Prueba

Se ejecutaron casos de prueba exhaustivos que incluyeron: saltos normales y potenciados, caídas desde diferentes alturas, aterrizajes en plataformas estáticas y móviles, empuje de bloques en diversas ubicaciones, y la interacción con los sensores de gravedad variable.

4.3. Resultados de las Pruebas

La mayoría de las funcionalidades se comportaron según lo previsto. Las animaciones y movimientos de Goku, las interacciones con los enemigos y las respuestas de las plataformas móviles fueron fluidas y predecibles. El sistema de gravedad variable del Nivel 3 funcionó correctamente, alterando la jugabilidad de manera significativa.

4.4. Errores Encontrados

Goku se "pegaba" a las plataformas al subir: Este error, donde Goku se detenía al colisionar con la parte inferior de una plataforma durante un salto ascendente, fue crítico.

Solución: Se añadió la condición velocidadY >= 0 en la lógica de colisión de los pies de Goku con las plataformas. Esto asegura que la detección de apoyo solo ocurre cuando Goku está descendiendo o ya está en la plataforma.

Objetos empujables no siempre aplicaban gravedad: En ciertos escenarios, los bloques empujables no reaccionaban a la gravedad o no detenían su caída correctamente al salir de una plataforma.

Solución: Se revisó y mejoró la lógica de Nivel2::aplicarGravedadObjetos, asegurando que la gravedad se aplicara consistentemente y que la colisión con superficies detuviera la caída.

5. Conclusiones y Recomendaciones

5.1. Resumen de los Resultados

El desarrollo de "Desafío Final: Dragon Ball" ha demostrado la capacidad de Qt para la creación de juegos 2D, gestionando de forma efectiva la lógica del juego, las físicas y la interacción con el usuario. Las aproximaciones de físicas discretas fueron suficientes para simular un comportamiento realista, y la arquitectura modular facilitó la implementación de mecánicas complejas como las plataformas interactivas y la gravedad variable. La integración de los timers de Qt para las actualizaciones por frame resultó ser un método eficiente para el bucle de juego.

5.2. Recomendaciones para Futuras Mejoras

Sistema de animaciones más robusto: Implementar un sistema de animaciones basado en estados o finite state machines (FSM) para Goku y los enemigos, permitiendo transiciones más suaves y complejas entre sprites.

Agregar más efectos visuales y de sonido: Añadir efectos de partículas (explosiones, polvo al aterrizar) y sonidos para mejorar la inmersión del jugador.

Sistema de niveles basado en datos: Externalizar la configuración de plataformas y enemigos en archivos de datos (JSON, XML) para facilitar el diseño y la creación de nuevos niveles sin recompilar el código.

Fricción y fuerzas de reacción: Para una simulación más avanzada, considerar la implementación de fricción en las superficies y fuerzas de reacción en colisiones.

5.3. Lecciones Aprendidas

El proyecto subraya la importancia de una arquitectura de código limpia y el uso efectivo de los mecanismos de Qt, como señales y slots, para un desarrollo desacoplado. La implementación de físicas, aunque simplificada, requiere una comprensión clara de los principios cinemáticos y una depuración meticulosa de las condiciones de colisión para evitar comportamientos inesperados. La fase de pruebas es indispensable para garantizar una experiencia de juego fluida y libre de errores.

6. Apéndices

6.1. Código Fuente

- main.cpp: Punto de entrada de la aplicación.
- mainwindow.h / mainwindow.cpp: Gestión de la ventana principal y cambio de niveles.
- mainmenu.h / mainmenu.cpp: Lógica del menú principal.
- jugador.h / jugador.cpp: Implementación del jugador (Goku).
- enemigo.h / enemigo.cpp: Implementación de los enemigos.
- nivel1.h / nivel1.cpp: Lógica del Nivel 1.
- nivel2.h / nivel2.cpp: Lógica del Nivel 2.
- nivel3.h / nivel3.cpp: Lógica del Nivel 3.

6.2. Diagramas y Esquemas

- Diagrama de clases UML.
- Esquemas de la escena para cada nivel, mostrando la disposición de plataformas, obstáculos y sensores.

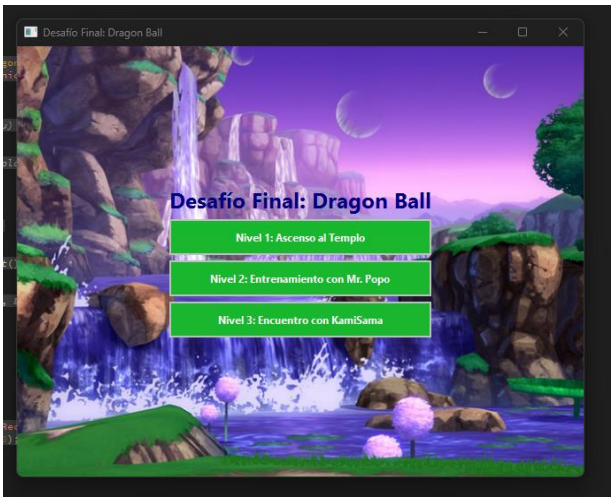
6.3. Recursos Utilizados

- Framework: Qt 5.x (QGraphicsView, QGraphicsScene, QTimer, QKeyEvent, QPixmap, etc.).
- Lenguaje de Programación: C++.
- Entorno de Desarrollo: Qt Creator.
- Activos Gráficos: Archivos PNG y JPG para sprites y fondos.

7. Sprite Usados

8. Imágenes del juego

Menu:



Nivel1:

Nivel2:

Nivel3:

9. Implementación de Audio

El juego incorpora efecto de sonido, para llamar la atención del jugador y confirmar su finalización correcta del juego; La implementación de audio se realizó directamente a través de funciones del API de Windows, específicamente PlaySound, para una integración sencilla y sin necesidad de librerías multimedia complejas de Qt (como Qt Multimedia, que requeriría módulos adicionales).

El sonido se reproduce al completar el Nivel 3, en la función Nivel3::mostrarVictoria().

C++

```
// Fragmento de Nivel3::mostrarVictoria()
#include <windows.h>
#include <mmsystem.h>
#pragma comment(lib, "winmm.lib") // Necesario para PlaySound

// ...

BOOL reproducido =
```

```
PlaySound(TEXT("C:/Users/juanm/Downloads/DragonBall/recursos/DragonBall.wav"),
                        NULL, SND_FILENAME | SND_ASYNC);
if (!reproducido) {
   qDebug() << "No se pudo reproducir el sonido final.";
}
```

Librerías de sistema: Se incluyen las cabeceras <windows.h> y <mmsystem.h>. Es crucial la línea #pragma comment(lib, "winmm.lib") para asegurar que la biblioteca winmm.lib (que contiene PlaySound) se enlace correctamente en Windows.

Función PlaySound: Esta función es una parte del API de Windows que permite reproducir archivos de audio WAV.

El primer argumento (TEXT("C:/Users/juanm/Downloads/DragonBall/recursos/DragonBall.wav")) especifica la ruta del archivo de audio a reproducir. Se utiliza TEXT() para asegurar la compatibilidad con diferentes codificaciones de caracteres.

El segundo argumento (NULL) se utiliza para especificar el módulo desde donde se carga el recurso de sonido (en este caso, es un archivo, no un recurso incrustado).

El tercer argumento (SND_FILENAME | SND_ASYNC) son flags que indican:

SND_FILENAME: Que el primer argumento es una ruta de archivo.

SND_ASYNC: Que el sonido debe reproducirse de forma asíncrona, permitiendo que el programa continúe su ejecución sin esperar a que el sonido termine.

Manejo de errores: Se verifica el valor de retorno de PlaySound. Si es FALSE, indica que la reproducción falló (por ejemplo, si el archivo no se encontró), y se imprime un mensaje de depuración.

Importante para la compatibilidad del audio, el formato debe ser WAV (PCM 16-bit, 44100 Hz, mono o estéreo), se puede hacer usos de plataformas en la Web, para su conversión, como: cloudconvert.com(usada en este trabajo).

Esta implementación, aunque simple y específica de Windows, es efectiva para la reproducción de un efecto de sonido final, proporcionando una realimentación auditiva al jugador cuando completa el nivel.

10. Cómo se Juega

"Desafío Final: Dragon Ball" es un juego de plataformas 2D que reta al jugador a superar obstáculos y resolver pequeños acertijos usando las habilidades de Goku.

Objetivo General: Guiar a Goku a través de diferentes niveles, cada uno con desafíos únicos, hasta alcanzar la meta final.

Controles Básicos:

Movimiento Horizontal: Usa las teclas A (izquierda) y D (derecha) para mover a Goku horizontalmente por el escenario.

Salto: Presiona la barra espaciadora para que Goku salte. La altura del salto está influenciada por la gravedad del nivel y, en ocasiones, por elementos del entorno como los trampolines.

Modo Empuje: En el Nivel 2, mantén presionada la tecla S para activar el "modo empuje". Mientras estés en este modo, si estás en contacto con un bloque empujable y presionas A o D, Goku empujará el bloque en la dirección correspondiente.

Mecánicas por Nivel:

Nivel 1 (Entrenamiento Básico): Es un nivel introductorio con plataformas estáticas y algunos enemigos básicos. El objetivo es familiarizarse con el movimiento y el salto de Goku.

Nivel 2 (El Desafío de los Bloques): Introduce plataformas móviles y bloques empujables. El jugador debe usar el modo empuje para mover los bloques y activarlos en zonas específicas para resolver un puzle y avanzar. Hay un trampolín que proporciona un impulso extra al saltar sobre él.

Nivel 3 (La Prueba de Kami Sama): Este nivel introduce un cambio dinámico en la gravedad. Goku encontrará zonas especiales (sensores) que alterarán la fuerza de la gravedad, permitiéndole "flotar" hacia arriba o descender más lentamente, abriendo nuevas rutas. El objetivo es usar estos cambios de gravedad para alcanzar la plataforma final y completar la prueba de Kami Sama.

Interacción con el Entorno:

Plataformas: Sirven como superficies para caminar y saltar. Algunas plataformas se mueven automáticamente o interactúan con Goku/objetos.

Enemigos: Evita el contacto con los enemigos. El juego no implementa combate directo, por lo que el enfoque es la evasión.

Objetos Empujables: Utiliza el modo empuje para desplazar estos objetos y, en el Nivel 2, para activarlos en zonas clave del puzle.

Trampolines: En el Nivel 2, los trampolines proporcionan un impulso de salto significativamente mayor cuando Goku aterriza sobre ellos.

Sensores de Gravedad: En el Nivel 3, al entrar en un área con un sensor, la gravedad puede invertirse o disminuir, cambiando drásticamente el control de Goku.

Condiciones de Derrota/Reinicio: Si Goku cae fuera de los límites inferiores de la escena, el nivel se reinicia a su estado inicial.

Victoria: Cada nivel tiene una "meta" (representada visualmente, por ejemplo, por una plataforma verde o un sensor específico). Al alcanzar esta meta, el nivel se completa y el jugador avanza o regresa al menú principal, dependiendo de la lógica del juego.