Momento III

Marx Eusdav Lopez Montero

Facultad De Ingeniería

Universidad De Antioquia

Informatica II: 2598521

11 de junio del 2025

1. Descripción General

Durante el desarrollo del proyecto final de programación en C++ con Qt, se creó un videojuego inspirado en la estética de Dragon Ball, protagonizado por el personaje Goku. El juego se estructuró en dos niveles de jugabilidad diferenciada, cada uno diseñado para representar distintos desafíos técnicos y mecánicos. El propósito fue aplicar conceptos fundamentales de programación orientada a objetos, manejo de escenas gráficas, implementación de física en videojuegos, control de entrada mediante teclado, animaciones por sprites y lógica de combate entre personajes.

2. Estructura y Dinámica General del Juego

Nivel 1: " Ingreso a la Torre del Ejército Rojo " – Estilo Endless Runner

El primer nivel simula una dinámica de desplazamiento lateral automático, tomando como referencia el clásico juego del dinosaurio de Google. En esta etapa, Goku avanza constantemente hacia la derecha en un entorno industrial, enfrentando obstáculos y enemigos.

Dinámica y estilo de juego:

- Desplazamiento automático del personaje, con fondo en movimiento simulando avance.
- Implementación de salto controlado por física realista.
- Obstáculos como troncos rotatorios y elementos estáticos colocados en el trayecto.
- Enemigos que disparan desde posiciones fijas.

Objetivo:

- Alcanzar una zona segura sin perder todas las vidas.
- Esquivar obstáculos mediante salto o agachamiento.

Interacciones clave:

- Activación del salto mediante la tecla W.
- Agacharse con la tecla S, con lógica para recuperación.
- Activación de disparos especiales tras completar el puntaje en niveles posteriores.

Nivel 2: " Combate contra el Mecha de Black " – Estilo Arcade Lateral

El segundo nivel presenta un enfrentamiento directo entre Goku y un robot enemigo en un escenario cerrado con desplazamiento libre. El diseño emula clásicos títulos arcade de combate uno a uno.

Dinámica del enfrentamiento:

- Movimiento controlado por el jugador hacia la izquierda y derecha.
- Activación de salto, agachamiento y disparo mediante teclas mapeadas.
- El robot enemigo presenta IA personalizada con patrullaje, detección de Goku y ejecución de ataques en función de distancia.

Interacciones clave:

- Goku utiliza teclas A/D para desplazamiento horizontal, W para saltar, S para agacharse y K para disparar.
- El enemigo puede atacar mediante rayos o golpe directo, regulado por temporizadores internos.

3. Sistemas Físicos Implementados

A lo largo del juego se integraron diversos sistemas de física para enriquecer la experiencia de jugabilidad y dotar de realismo a las acciones:

Gravedad aplicada al salto:

- Se utilizó un sistema de aceleración vertical para simular el salto de Goku.
- Se controló la velocidad inicial y el efecto de la gravedad mediante actualizaciones por frame.
- La lógica impide saltos múltiples y respeta condiciones de colisión con el suelo.

Detección de colisión por bounding *boxes:*

Se implementaron condiciones de contacto entre personajes, proyectiles y escenarios usando boundingRect() y collidingItems().

- Se añadieron efectos visuales como cambio de opacidad y animación de daño al recibir impacto.
- Sistema de disparo con trayectoria rectilínea:
- El Kamehameha se desplaza en línea recta con velocidad constante.
- Se elimina automáticamente cuando sale de los límites de la escena o colisiona con otro objeto.
- La lógica de colisión distingue entre dueño del proyectil y enemigos, evitando daño auto-infligido.

Control de movimientos restringidos por mapa:

- Se definieron límites máximos y mínimos para desplazamiento horizontal, impidiendo que Goku se teletransporte fuera del escenario.
- Sistema de vida con representación gráfica:

- Se implementó un sistema de tres vidas visibles mediante sprites.
- Cada impacto reduce la vida, actualiza la interfaz y eventualmente desencadena la condición de derrota.

4. Control de Entrada y Gestión de Eventos

- Se utilizó keyPressEvent y keyReleaseEvent para gestionar acciones como saltar, moverse, agacharse y disparar.
- Se resolvieron conflictos de enfoque en la vista (QGraphicsView) y en los personajes, garantizando recepción de eventos de teclado.
- Se integró el control del enfoque tras transiciones entre niveles y reinicios, usando setFocusPolicy() y llamadas secuenciales de setFocus().

5. Arquitectura de Clases y Modularidad

Se estableció una jerarquía de clases con Personaje como clase base, además, Goku y Robot heredan sus métodos generales como movimiento y vida. Por otro lado, “AnimacionSprite” se encarga de reproducir secuencias visuales desde spritesheet y Kamehameha representa el proyectil con trayectoria, velocidad y detección de colisión.

La clase RobotAI implementa comportamiento inteligente para el enemigo:

- Evaluación de distancia con respecto a Goku.
- Determinación de acciones: atacar, moverse o saltar.
- Enfriamiento de habilidades mediante QTimer para evitar spam de ataques.
- NivelBase sirve como plantilla común para las escenas, permitiendo reutilización de métodos de configuración, posicionamiento y reinicio.

6. Sistema de Mensaje Final y Reinicio

Se diseñó la clase MensajeFinal, responsable de mostrar resultados al jugador:

- Mensaje de “YOU WIN” al derrotar al enemigo.
- Mensaje de “YOU LOSE” al perder todas las vidas.
- Botón de “Reintentar” con conexión a reiniciarNivel(), que restablece estado y posición.
- Se utilizó QGraphicsTextItem, QGraphicsProxyWidget y QPushButton para componer la interfaz visual del mensaje final.

7. Transición entre Escenas

MainWindow administra el cambio entre menú y niveles. Este se conecta mediante señales como nivelCompletado y nivelFallido.Tambien, se configura la vista, foco y reinicio de lógica tras cada transición, además, se agregan retardos controlados con QTimer para estabilizar la interacción.

8. Consideraciones de Optimización y Estabilidad

- Se identificaron y corrigieron errores relacionados con foco, duplicación de eventos, acceso a objetos eliminados y fugas de memoria.
- Se estructuró la eliminación segura de proyectiles y enemigos tras colisiones, evitando referencias colgantes.
- Se mejoró la eficiencia mediante uso de estructuras como QVector en lugar de QList donde fue necesario.
- Se encapsularon atributos y métodos con visibilidad adecuada para preservar la integridad de las clases.

9. Conclusión Técnica

El desarrollo del videojuego permitió implementar satisfactoriamente conceptos de programación orientada a objetos, control gráfico, físicas aplicadas, y lógica de interacción dinámica. Los dos niveles del juego presentan desafíos diferenciados, equilibrando mecánicas de esquiva y combate. La integración de animaciones, físicas, IA y manejo de interfaz gráfica resulta en una experiencia funcional, pulida y coherente. Se cumplieron todos los objetivos técnicos planteados, y se sentaron las bases para futuras extensiones del proyecto.