Informe Final
Proyecto - Informática II
Semestre 2025-1

Benjamin Ruiz Guarin
Kevin Jimenez Rincón

14 de julio de 2025

1. Análisis del problema

El Objetivo del proyecto es desarrollar un juego con temática de Dragon Ball donde se pongan a prueba los conocimientos adquiridos a lo largo del semestre.

Consideraciones importantes en el desarrollo:

- El proyecto se desarrollo en 3 etapas o "Momentos".
- Se debían de incluir por lo menos 3 modelos físicos.
- El juego debí de tener al menos dos niveles diferenciables.
- Era obligatorio el uso de POO, memoria dinámica, herencia, contenedores y la interfaz gráfica de Qt.

2. Diseño de la Solución

Para el diseño de la solución fue fundamental establecer desde el inicio una visión clara del videojuego a implementar. Tras analizar el capítulo seleccionado de *Dragon Ball*, se decidió desarrollar una dinámica de combate uno contra uno (1v1) en un entorno bidimensional, inspirado en juegos clásicos del estilo *Street Fighter*.

A partir de esta visión, se estructuró el proyecto utilizando Programación Orientada a Objetos (POO), identificando dos clases base fundamentales: `Personaje` y `Habilidad`. Estas clases proporcionan la arquitectura general desde la cual heredan las distintas instancias del juego.

La clase `Personaje` actúa como superclase tanto para el jugador (Goku) como para el enemigo (Piccolo), encapsulando atributos y comportamientos comunes como el control del *ki*, la salud, los movimientos y la interacción con habilidades. Por otro lado, `Habilidad` sirve como superclase para todas las técnicas ofensivas, permitiendo la creación de habilidades con distintas trayectorias, animaciones y físicas.

Se emplearon modificadores de acceso (`public`, `protected`, `private`) para garantizar un adecuado encapsulamiento, y se utilizaron funciones virtuales para permitir la sobrescritura de métodos clave en las clases derivadas. También se hizo uso del sistema de *signals* y *slots* de Qt, facilitando la comunicación entre objetos y promoviendo una arquitectura desacoplada y extensible.

En general, el diseño modular y orientado a objetos permitió manejar de forma clara las diferencias entre las entidades del juego, manteniendo una estructura coherente y reutilizable a lo largo del desarrollo. Posteriormente se consideró la posibilidad de utilizar la clase `Game` como clase base para representar los distintos niveles, pero debido al estado avanzado del desarrollo, se optó por mantener la estructura ya implementada.

2.1. Modelos Físicos Implementados

Como parte de los requerimientos del curso, se implementaron tres modelos físicos parametrizables. Se hizo uso de modelos simples como el movimiento rectilíneo o parabólico pero en este caso se usaran trayectorias más interesantes y visualmente atractivas. Las físicas finalmente empleadas fueron:

2.1.1. Atractor de Lorenz

Este modelo describe un sistema dinámico caótico tridimensional. Las ecuaciones diferenciales que lo rigen son:

$$\frac{dx}{dt} = \sigma(y - x) \tag{1}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{2}$$

$$\frac{dz}{dt} = xy - \beta z \tag{3}$$

Donde σ, ρ y β son parámetros configurables del sistema pero hay que tener en cuenta que debido a ser 2D no se aprecia el fenomeno completo.

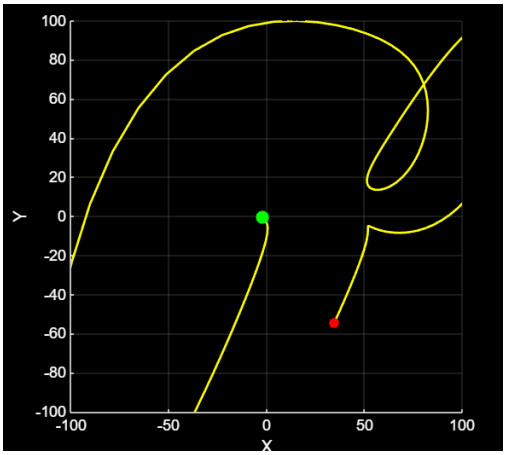

Figura 1: Simulacion Atractor de Lorenz.

2.1.2. Movimiento en Espiral

Este tipo de trayectoria se construye a partir de coordenadas polares, donde el radio r crece con el tiempo, y el ángulo θ también cambia linealmente:

$$r(t) = v_r \cdot t \tag{4}$$

$$\theta(t) = \omega \cdot t \tag{5}$$

$$x(t) = r(t) \cdot \cos(\theta(t)), \quad y(t) = r(t) \cdot \sin(\theta(t)) \tag{6}$$

Siendo v_r la velocidad radial y ω la velocidad angular.

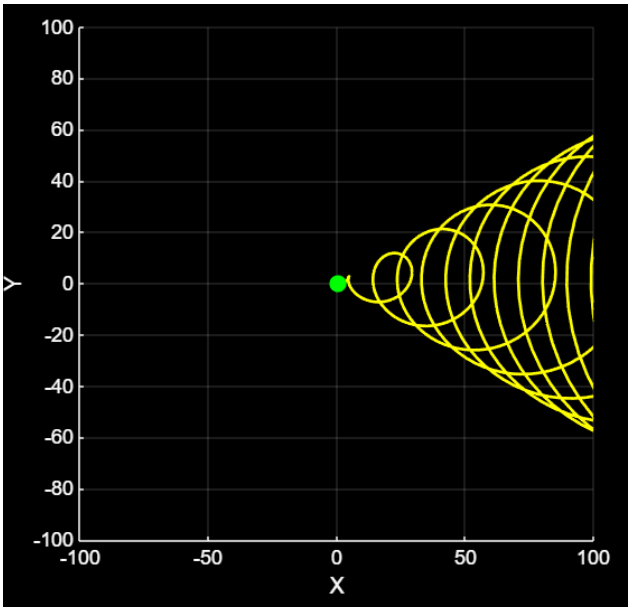

Figura 2: Simulacion Movimiento en Espiral.

2.1.3. Atracción Gravitacional

En esta física, un proyectil se ve atraído constantemente hacia el personaje Goku, con un comportamiento inspirado en la gravitación:

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \vec{a}(t) \cdot \Delta t \tag{7}$$

$$\vec{a}(t) = G \cdot \frac{\vec{r}}{|\vec{r}|} \tag{8}$$

Donde $\vec{r}$ es el vector desde el proyectil hacia Goku, normalizado, y G representa una constante de fuerza gravitacional configurable.

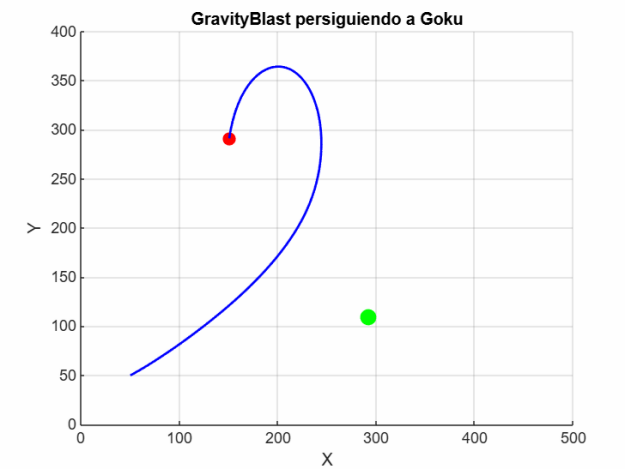

Figura 3: Simulacion Atracción Gravitacional.

3. Diagrama de Clases

Más abajo en este docuemnto se puede encontrar una figura que representa una versión simplificada del diagrama de clases de la solución planteada. Dada la extensión, el diagrama fue simplificado. Los detalles omitidos se pueden ver en el archivo adjunto "Miembros_Atributos_Clases.pdf".

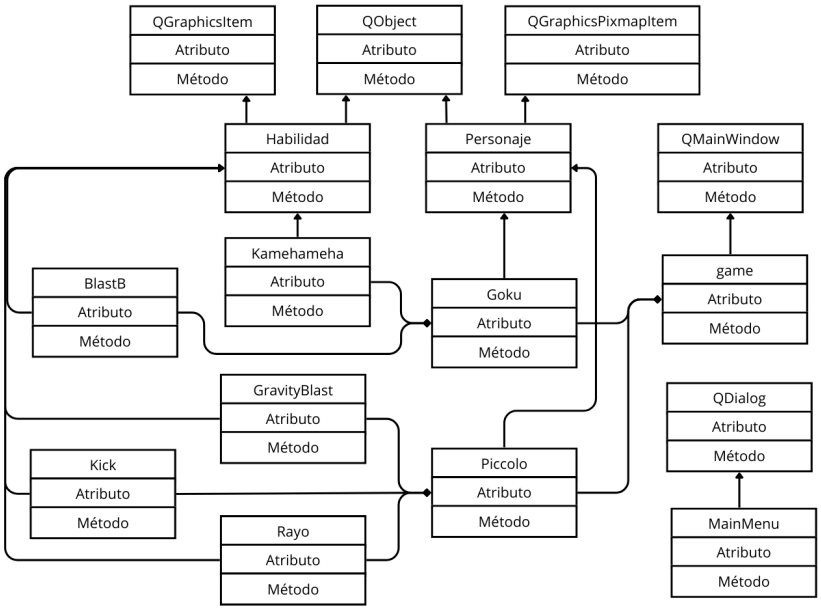

Figura 4: Diagrama de Clases de la solución planteada.

4. Principales Métodos utilizados

Los siguientes son los principales métodos, signals y slots utilizados para el funcionamiento del proyecto.

4.1. Goku::moverDerecha()

Este método se encarga de mover al personaje de Goku hacia la derecha. Restringe el movimiento si el personaje no está visible o si está realizando animaciones específicas (entrada o recarga de Ki). Maneja el movimiento horizontal tanto en el suelo como durante el salto, actualizando la posición del personaje y cambiando su sprite correspondiente. Finalmente, verifica los límites de la pantalla y actualiza la visualización de la hitbox. Los demás métodos relacionados con el movimiento son similares a este.

4.2. Goku::recibirDanio(int danio)

Este método aplica daño al personaje de Goku. Si Goku está recargando Ki, detiene esta acción. Actualiza la vida del personaje, asegurándose de que no sea negativa, y emite una señal vidaCambiada para actualizar la interfaz de usuario. Cambia el sprite de Goku a "herido"temporalmente y, si la vida llega a cero, llama al método morir(). Para ambos personajes se maneja un control del daño similar.

4.3. Goku::iniciarRecargaKi()

Este método inicia el proceso de recarga de Ki del personaje de Goku. Activa una animación visual de carga de Ki y detiene cualquier otra animación activa (como el idle o movimiento). Inicia temporizadores para la animación y la recarga gradual del valor de Ki.

4.4. GravityBlast::iniciar(QPointF posicionInicial, QPointF direccion)

Este método inicializa y activa la habilidad "Gravity Blast". Establece la posición inicial y la dirección de movimiento de la habilidad en la escena del juego. Una vez configurada, la habilidad se activa y sus temporizadores de actualización de física y animación se inician, permitiendo que la habilidad se mueva y se anime en el juego según una fórmula física. Las demás habilidades, especialmente las que llevan físicas complejas funcionan de forma similar a esta.

4.5. Habilidad::actualizarFisica()

Este método es responsable de actualizar la física de cualquier habilidad en el juego. Si la habilidad está activa, calcula su nuevo desplazamiento basándose en la dirección y velocidad de movimiento. Actualiza la distancia recorrida y verifica si la habilidad ha alcanzado su alcance máximo, en cuyo caso se detiene. También se asegura de que la habilidad permanezca dentro de los límites de la pantalla y llama a una función de actualización específica de la habilidad. En caso de ser necesario es sobreescrita como método de las funciones hijo.

4.6. Piccolo::atacar()

Este método permite a Piccolo realizar un ataque de patada. Verifica la existencia de una escena de juego y, si es válida, crea una nueva instancia de la clase Kick. Configura la patada con la escena actual y determina su posición inicial basándose en si es una patada alta o baja. Finalmente, inicia la patada y la añade a la escena del juego para su visualización y colisión. Esto solo se usa en la fase 2 de piccolo (segundo nivel).

4.7. Piccolo::lanzarGravityBlast()

Este método permite a Piccolo lanzar su habilidad "Gravity Blast". Instancia un nuevo objeto GravityBlast, establece la escena del juego para la habilidad y el objetivo (Goku). Calcula la posición y dirección de lanzamiento de la habilidad basándose en la posición actual de Piccolo y su dirección horizontal. Después de configurar la habilidad, la inicia y la añade a la escena. Esta habilidad siempre va directa hacia goku, manejando un modelo de gravitación.

4.8. game::keyPressEvent(QKeyEvent *e)

Este método sobrescrito de QMainWindow maneja los eventos de presión de teclas en la ventana principal del juego. Controla el movimiento de Goku (teclas A, S, D, W), sus ataques cuerpo a cuerpo (X para golpear, C para patear), y el salto direccional (Espacio). También gestiona las habilidades de Ki de Goku (K para recargar, J para Kamehameha, L para ráfaga, T para teletransporte) y opciones de depuración como alternar la visualización de hitboxes (H) o cambiar el fondo (B).

4.9. game::actualizarBarraVida(int vidaActual, int vidaMaxima)

Este método (slot) se conecta a la señal vidaCambiada del personaje Goku. Su función es actualizar la representación visual de la barra de vida de Goku en la interfaz de usuario. Recibe la vida actual y máxima del personaje, calcula el sprite de la barra de vida correspondiente y lo establece en el objeto QGraphicsPixmapItem que representa la barra de vida. Existe una función similar que hace lo mismo con Piccolo.

4.10. game::piccoloActualizarMovimiento()

Este método (slot) es responsable de la lógica de movimiento de la inteligencia artificial de Piccolo. Utiliza un contador (cntPiccolo) para determinar las acciones de Piccolo. Dependiendo del valor del contador, Piccolo puede moverse hacia Goku, atacar, o lanzar habilidades como Rayo y Gravity Blast. También maneja la activación de la transformación de Piccolo basada en su vida.

5. Problemas enfrentados

Aunque se dedicó una cantidad considerable de tiempo al desarrollo del proyecto, uno de los principales desafíos fue la ambición del alcance propuesto en relación con el tiempo disponible. Si bien se está satisfecho con el resultado final, quedaron ideas sin implementar por limitaciones de tiempo.

Otro problema recurrente fue la inconsistencia en los tamaños de los *sprites*. Dado que se utilizaron múltiples animaciones por personaje, y cada habilidad contaba con varios frames, fue común que se presentaran errores gráficos o desajustes visuales al cambiar de sprite durante la ejecución.

El manejo de memoria dinámica también representó una fuente de errores. En particular, el uso inadecuado de temporizadores (`QTimer`) provocaba comportamientos inesperados o cierres inesperados del programa cuando no se liberaban correctamente o se duplicaban conexiones.

Finalmente, uno de los aspectos más complejos fue la incorporación del sistema de `signals` y `slots` de Qt. Al tratarse de la primera vez trabajando con este mecanismo, fue necesario un periodo de aprendizaje para comprender su lógica y aplicarlo correctamente dentro de una arquitectura orientada a objetos. Por ultimo, se quiso agregar el sistema de sonidos independientes por objeto pero el tiempo no dio lo suficiente. Pese a estos inconvenientes, todos los problemas se abordaron y resolvieron dentro del tiempo disponible, permitiendo la entrega de una solución funcional, modular y que cumple con los requisitos establecidos.

6. Evolución de la Solución

El desarrollo del videojuego se dividió en tres momentos fundamentales, cada uno con sus propios objetivos y entregables. A continuación, se presenta un resumen de la evolución del proyecto y los cambios realizados respecto a lo planteado inicialmente.

Momento I: Contextualización

En este primer momento, se definió la idea base del videojuego: un juego de combate 1v1 inspirado en el capítulo final del 23º Torneo Mundial de Artes Marciales de *Dragon Ball*, donde Goku se enfrenta a Piccolo. Se propusieron tres niveles progresivos con dinámicas diferenciadas y se establecieron las físicas principales a implementar, tales como el movimiento parabólico, rebotes, y ataques curvos. Se identificaron desde el principio los personajes principales, las habilidades, y se creó el repositorio para dar inicio al desarrollo.

Momento II: Diseño y Análisis

En este momento se elaboró el diagrama de clases con detalle medio, definiendo las relaciones de herencia, los contenedores utilizados, y los métodos fundamentales. Se propusieron los sprites a utilizar y se describieron con más profundidad las dinámicas de cada nivel. No se registraron cambios sustanciales con respecto a la planificación del Momento I.

Cambios en la Versión Final

Durante la implementación surgieron ajustes que llevaron a modificar parcialmente algunos aspectos de la propuesta original:

- El nivel final inicialmente contemplado (una batalla renovada con nuevas habilidades) fue fusionado con el primer nivel debido a limitaciones de tiempo. Se enfocó el esfuerzo en hacer

más robusto y jugable el primer nivel y desarrollar un segundo con un cambio radical en la jugabilidad.

- La física de salto parabólico fue implementada, pero no resultó significativa en la jugabilidad, por lo que su uso fue limitado y no tuvo un protagonismo en las estrategias del jugador.
- Se incluyeron nuevas físicas no contempladas inicialmente: el atractor de Lorenz y la atracción gravitacional, que aportaron variedad visual y mecánica a los ataques del enemigo.
- La clase `Game` se mantuvo como controlador principal del flujo del juego, aunque se consideró convertirla en una clase base para los diferentes niveles. Esta idea se descartó por falta de tiempo para reestructurar.
- No se logró implementar completamente el sistema de sonido independiente por objeto, aunque sí se dejaron sentadas las bases para hacerlo.
- Los ataques usados por cada personaje cambiaron radicalmente al igual que las físicas que se usaron.