Momento 3 Proyecto Final

Juan Angel Omaña Montañez

Universidad de Antioquia

Informática 2

Aníbal José Guerra Soler – Augusto Salazar

Medellín, Colombia

Julio, 2025

Introducción

El videojuego Dragon Ball Tournament es un juego de lucha uno contra uno basado en Dragon Ball. Fue desarrollado en C++ utilizando librerías de Qt para la interfaz grafica, incluye manejo de física básica, animaciones con sprites y lógica de combate. El videojuego, permite enfrentar a ocho personajes seleccionables en combates que culminan con un torneo para definir al campeón. Cada personaje posee atributos de vida, energía (Ki), velocidad y daño base, los cuales determinan su rendimiento en combate. En cada combate, el jugador puede mover a su luchador horizontalmente, saltar, atacar, defenderse o usar un ataque especial (según la clase de personaje), interactuando con otro oponente controlado por la IA.

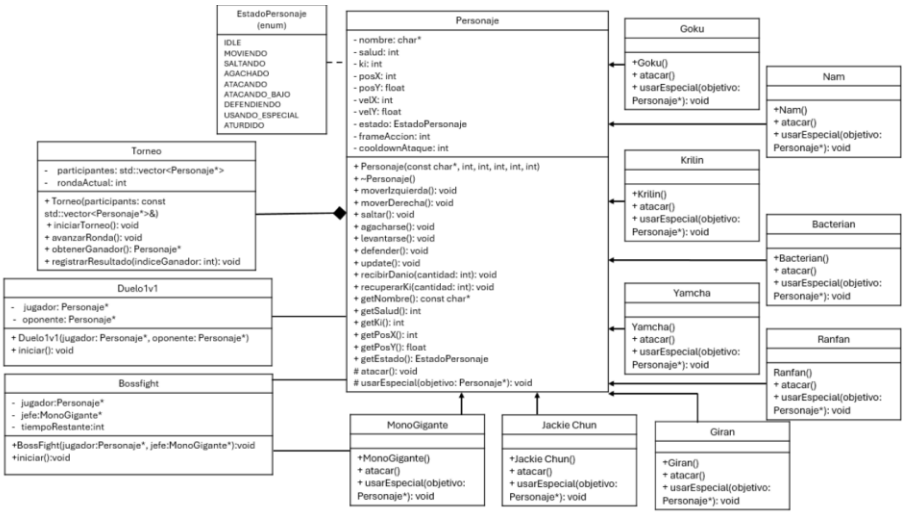
Desarrollo

Momento I: Diseño inicial

En el diseño inicial se establecieron los requisitos y la estructura básica del juego. Se definió la existencia de una clase base Personaje con atributos esenciales (vida máxima, vida actual, Ki, posición y velocidad) y un estado interno (idle, moviendo, atacando, defendiendo, saltando, aturdido, usando especial). Cada personaje también tenía valores iniciales de vida, velocidad horizontal y daño base. Sobre esta clase base se planearon subclases para los ocho luchadores jugables (Goku, Krilin, Yamcha, Jackie Chun, Bacterian, Nam, Ranfan y Giran), cada una con su ataque especial implementado mediante el método virtual usarEspecial(Personaje*). Inicialmente se consideraron modos adicionales (como un BossFight contra un jefe final) y estados como la postura de agachado, pero estos no se incluyeron en la versión final. En esta etapa se identificaron las clases principales (Personaje, Duelo1v1, Torneo, etc.), sus responsabilidades y la lógica de interacción. Cada personaje podía moverse horizontalmente, saltar, atacar o defenderse, lo que incluye recibir daño según su estado actual en el combate, y los enfrentamientos transcurrirían de forma secuencial hasta que quedara un campeón.

Momento II: Análisis lógico y visual

En esta fase se detalló la arquitectura interna, representada en un diagrama de clases (ver figura). Se diseñó cómo se relacionan las clases principales y cómo se gestiona el flujo de datos y estados del juego.



El diagrama muestra que la clase base Personaje contiene los atributos esenciales (vida, Ki, posición y velocidad) y métodos para acciones básicas (moverse, saltar, atacar, defender). Cada personaje específico hereda de esta clase y sobreescribe el comportamiento de atacar() y usarEspecial(), ajustando los efectos según el luchador. Se incluyó un enum EstadoPersonaje con estados como IDLE, MOVIENDO, SALTANDO, ATACANDO, DEFENDIENDO, USANDO_ESPECIAL y ATURDIDO para representar las diferentes acciones de los luchadores.
La clase Torneo gestiona la lógica del campeonato: recibe el jugador inicial, una lista de contrincantes y organiza pares de combates. En los primeros diagramas de diseño aparecían métodos conceptuales como iniciarTorneo(), avanzarRonda() y registrarResultado(). En la versión final, estas funciones se concretan en métodos que preparan y registran duelos en un vector de pares std::vector<std::pair<Personaje*,Personaje*>>. La clase Duelo1v1 representa un enfrentamiento 1 contra 1: en su constructor se inicializan los dos personajes (_jugador y _oponente), reiniciando sus estadísticas y fijando sus posiciones iniciales en pantalla.

Momento III: Implementación final
En la implementación final se integraron todos los componentes siguiendo la arquitectura planificada, con adaptaciones prácticas. El código final coincide con las especificaciones de diseño y cumple los requisitos definidos, gestionando recursos y eventos correctamente.

Físicas
El movimiento se basa en una física simplificada. Los métodos moverIzquierda() y moverDerecha() ajustan la posición posicionX dentro de límites prefijados (entre MIN_X y MAX_X), y cambian el estado del personaje a MOVIENDO siempre que no esté atacando. El salto se inicia fijando velocidadY = 20 cuando el personaje está en el suelo en estado IDLE o MOVIENDO. Luego, en cada frame el método Personaje::update() actualiza la posición vertical y aplica la gravedad restando a la velocidad vertical (constante GRAVEDAD), de modo que el personaje vuelve al suelo cuando posicionY alcanza cero. Durante el combate, en cada iteración de frame se procesan colisiones con la función procesarColision: si un personaje golpea al otro, se aplica daño y se usan banderas (_jugadorYaGolpeó, _oponenteYaGolpeó) para impedir golpes repetidos continuos.

POO y manejo de memoria
El proyecto está organizado con clases específicas para cada entidad. Por ejemplo, CombateWidget recibe punteros a objetos Personaje y un booleano ownsCharacters que determina si debe eliminar esos personajes en su destructor. Si ownsCharacters es true, el destructor de CombateWidget libera los objetos Personaje para evitar fugas de memoria. De forma similar, el destructor de TorneoWidget elimina todos los participantes del torneo al cerrar la ventana. La ventana principal (MainWindow) utiliza qDeleteAll para borrar el vector roster de personajes en su destructor. Adicionalmente, se emplean mecanismos de Qt para comunicación: por ejemplo, CombateWidget::on_btnContinuar_clicked() emite la señal combateTerminado(bool) al cerrar un combate, que activa el método TorneoWidget::onCombateTerminado(bool) para registrar el resultado y continuar el torneo. Asimismo, al destruir un widget (combate o torneo) se ejecuta un slot que regresa automáticamente al menú principal. En el menú de selección de personaje, los objetos seleccionados se clonan (clone()) para no modificar el roster original.

Contenedores utilizados

En el proyecto se emplearon diversos contenedores de STL y de Qt para organizar datos:

- std::vector: utilizado para listar personajes y duelos. Por ejemplo, la clase Torneo almacena sus participantes en std::vector<Personaje*> y genera los duelos en std::vector<std::pair<Personaje*,Personaje*>>.
- *std::pair<Personaje,Personaje>***: cada duelo 1v1 se representa como un par de punteros a Personaje.
- QSet<int>: en Duelo1v1 se usa QSet para mantener el conjunto de teclas presionadas simultáneamente (movimiento y ataques del jugador), facilitando la detección de múltiples entradas en cada frame.
- QMap<QString,QVector<QPixmap>> y QVector<QPixmap>: en CombateWidget las animaciones se almacenan en un mapa (QMap) que asocia cada acción ("idle", "walk", "attack", etc.) a un vector de imágenes (QVector<QPixmap>), permitiendo iterar fácilmente las secuencias de cuadros.
- Layouts de Qt (QGridLayout, QVBoxLayout, etc.) y QLabel, QPixmap: la interfaz gráfica emplea estos contenedores y widgets para mostrar retratos de personajes, barras de vida/ki y textos, organizándolos en la ventana.

Abstracción de dificultad e IA

Se implementó una IA sencilla para los oponentes controlados por computadora. La función ejecutarIA() en IA.cpp llama a decidirAccion(), que selecciona la acción del CPU según su distancia al jugador y el porcentaje de vida restante. Si el oponente está lejos, la IA se acerca (movimiento horizontal). Si están cerca, calcula probabilidades basadas en su vida (por ejemplo, con vida alta tiende a atacar, con vida baja prioriza defender) y genera un número aleatorio para decidir entre atacar, defender, usar especial o saltar. Además, en TorneoWidget::onDuelosObtenidos() los combates entre dos CPUs se resuelven aleatoriamente (50% de probabilidad de ganar cada uno) usando un generador std::mt19937.

Excepciones y manejo de errores

El código evita errores mediante comprobaciones preventivas. No se emplean excepciones explícitas. Por ejemplo, antes de procesar un duelo se verifica que el índice esté dentro de rango (if (indiceDuelo >= duelosActuales.size()) return;). La función Torneo::obtenerDueloJugador() devuelve {nullptr,nullptr} si el jugador no está en ningún duelo actual, evitando referencias inválidas. En el menú de selección se verifica que se haya elegido un personaje antes de habilitar el botón de inicio (checando que selectedCharacter no sea nulo), previniendo accesos incorrectos.

Cambios respecto a los momentos anteriores

En la versión final se hicieron ajustes y se eliminaron características planeadas inicialmente. Entre los cambios más notables:

- Se omitió el modo BossFight y la postura agachado que aparecían en los primeros bocetos, concentrando el desarrollo en los modos básico de torneo y duelo.
- Se optimizó la generación de duelos: en lugar de métodos separados por ronda, el torneo mezcla a los participantes en cada fase con std::shuffle y crea directamente los pares con prepararDuelos().
- En TorneoWidget::onDuelosObtenidos() se añadió la resolución automática de combates CPU vs CPU usando un generador aleatorio (std::mt19937 con distribución uniforme), eliminando la necesidad de simular esos combates.

- Se reforzó la gestión de memoria: al iniciar un modo se clonan los personajes seleccionados en lugar de usar los originales, evitando interferir con el roster principal.
- Se ajustó la IA para que sus probabilidades dependan del estado de salud actual. Para el orden aleatorio del torneo se utiliza un generador Mersenne Twister.
- Se agregaron conexiones de señales entre widgets para controlar el flujo: por ejemplo, al terminar un combate en torneo se emite combateTerminado(bool) activando onCombateTerminado; al destruir el widget de torneo se retorna al menú principal automáticamente.
- Se mejoraron las animaciones y el audio: se reproducen efectos de sonido al atacar, saltar, defender o moverse, y los ciclos de animación solo avanzan cuando cambia la acción del personaje.