Proyecto Final

Isabela Aguirre Ceballos
Juan Felipe Higuita Perez

Informática II

Universidad de Antioquia

Análisis del problema y consideraciones para la alternativa de solución propuesta

Inspirados en los episodios 64 y 65 de *Dragon Ball*, nos propusimos desarrollar un videojuego que recreara la narrativa y acción de la saga de la Patrulla Roja. El problema principal consistió en traducir una narrativa animada a una experiencia jugable que fuese intuitiva, entretenida y coherente con la historia original. En el **Momento I** se definió el concepto inicial, estableciendo los tres niveles principales con sus respectivos protagonistas y objetivos, y una sinopsis basada directamente en el contenido de los episodios.

El proyecto busca cumplir los siguientes objetivos:

- Diseñar tres niveles diferentes con mecánicas independientes pero conectadas narrativamente.
- Utilizar conceptos de física y programación orientada a objetos (POO).
- Incluir elementos de dificultad creciente, sin perder jugabilidad ni coherencia.
- Implementar una interfaz gráfica amigable y funcional usando Qt.
- Adaptar personajes clásicos como Goku y Tao Pai Pai a un entorno jugable.
- Incluir efectos de sonido, eventos de victoria y derrota, y controles intuitivos.

Consideraciones clave para la solución (Momento II):

- **Nivel 1: Torre Karin (Tao Pai Pai sube)**

  Se planteó una vista vertical, simulando el ascenso por una torre infinita. Se incorporó una mecánica de gravedad y la aparición de obstáculos como piedras que caen desde lo alto, obligando al jugador a reaccionar rápidamente.

- **Nivel 2: Combate Goku vs Tao Pai Pai**

  Se implementó una simulación de trayectoria parabólica para la granada lanzada por el enemigo. Además, se permite devolver el ataque si el jugador presiona en el momento exacto. Esta mecánica agrega dificultad táctica y variedad al combate.

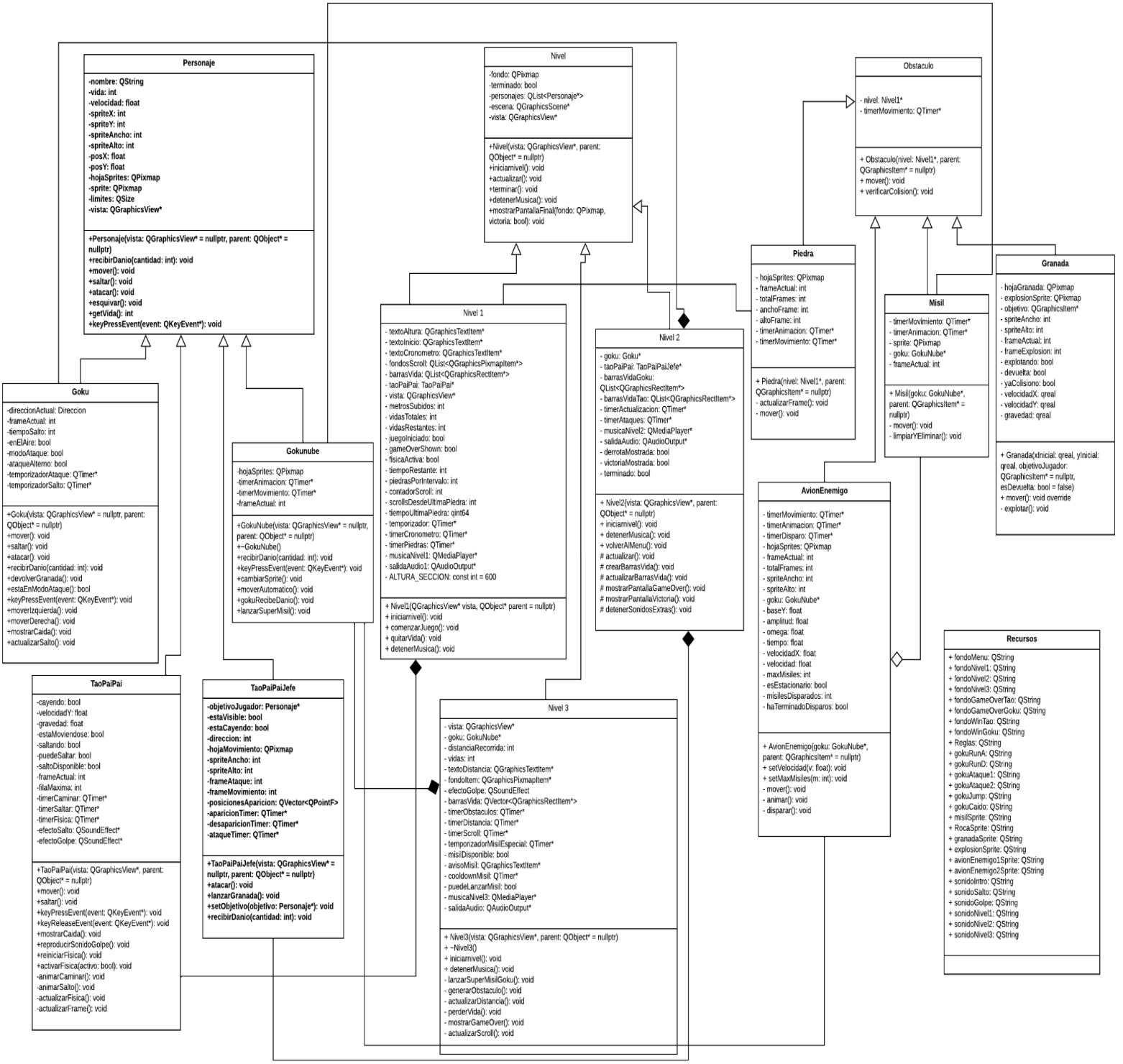
- **Nivel 3: Vuelo a la base de la Patrulla Roja**

  Se diseñó una vista horizontal de scroll automático. Goku viaja en su nube voladora, esquivando obstáculos aéreos como misiles y aviones enemigos. Se añadió variación en la velocidad y cantidad de obstáculos para representar un aumento progresivo en la dificultad.

Consideraciones técnicas generales:

- Se diseñó un menú principal con tres botones para iniciar cada nivel.
- Se usaron sprites y fondos escalados para adaptarse al tamaño de la vista sin deformaciones.
- Se definieron clases generales para los personajes y obstáculos, utilizando herencia.
- La dificultad progresiva se planteó como una abstracción mediante incremento en la frecuencia de aparición de obstáculos y reducción de tiempos de respuesta.
- Las colisiones se manejan mediante detección de superposición entre elementos gráficos.
- Se incorporaron pantallas de victoria y derrota para cada nivel, con botón funcional.

Diagrama de clases de la solución planteada



Descripción en alto nivel de la lógica de tareas complejas

Lógica del Nivel 1: Torre Karin (Tao Pai Pai)

El personaje principal de este nivel es **Tao Pai Pai**, quien debe ascender verticalmente por la Torre Karin. Para hacerlo, el jugador debe mantener un movimiento constante hacia arriba, ya que se ha implementado un sistema de **gravedad** que simula una fuerza que empuja al personaje hacia abajo, obligándolo a no detenerse.

Durante el ascenso, el jugador debe esquivar piedras que caen desde la cima con aceleración vertical, simulando un movimiento de **caída libre**. Estos objetos representan el principal obstáculo del nivel. Si Tao colisiona con una de ellas, pierde una vida y se activa un breve periodo de inmunidad temporal, evitando que reciba daño inmediatamente después del impacto.

El número de vidas disponibles se muestra al jugador a través de una interfaz ubicada en la parte superior de la pantalla, permitiéndole tener siempre claro su estado actual.

Finalmente, al alcanzar la cima de la torre, se activa una **pantalla de victoria** y el nivel concluye.

Lógica del Nivel 2: Combate Goku vs Tao Pai Pai

En este nivel, el jugador toma el control de **Goku** en un escenario de combate de desplazamiento horizontal, enfrentándose directamente a Tao Pai Pai en una batalla cuerpo a cuerpo.

Tao lanza ataques físicos simples y también utiliza granadas, que se comportan de acuerdo con una **trayectoria parabólica** basada en principios físicos. Estas granadas representan un desafío mayor, ya que requieren precisión para ser esquivadas o incluso devueltas.

Goku puede:

- **Esquivar** ataques moviéndose lateralmente o saltando.
- **Contraatacar** cuando Tao queda momentáneamente vulnerable.
- **Devolver la granada** si el jugador logra presionar la tecla correspondiente en el momento exacto, lo que introduce una mecánica de riesgo y recompensa.

Tanto Goku como Tao Pai Pai cuentan con cinco vidas. Cada vez que uno recibe daño, pierde una de ellas. Cuando Tao Pai Pai pierde todas sus vidas, se activa una pantalla de victoria que indica el fin del combate.

Lógica del Nivel 3: Vuelo hacia la base de la Patrulla Roja

En este nivel, el jugador controla a Goku viajando en su nube voladora, desplazándose automáticamente hacia la derecha en un entorno de scroll horizontal continuo.Durante el vuelo, el jugador puede mover a Goku hacia arriba y hacia abajo para esquivar distintos tipos de obstáculos que aparecen a lo largo del trayecto.

Los obstáculos incluyen:

- **Aviones enemigos** que se mueven siguiendo un **movimiento armónico simple** (oscilan suavemente en el eje Y mientras avanzan en X).
- **Misiles** que caen o se desplazan con **movimiento rectilíneo uniforme**, apuntando directamente hacia Goku.

Se implementó un sistema preciso de detección de colisiones. Si Goku choca contra alguno de estos objetos pierde una vida.

El nivel concluye exitosamente cuando Goku logra sobrevivir a cierta distancia alcanzando así la base de la Patrulla Roja.

Algoritmos implementado intra-documentados

```
// Método que actualiza la posición del misil enemigo y detecta colisión con Goku
void Misil::mover() {
    try {
        // Mueve el misil 4 píxeles hacia la izquierda
        setX(x() - 4);

        // Si ya no existe la escena o Goku, eliminar el misil
        if (!scene() || !goku || !goku->scene()) {
            limpiarYEliminar();  // Detiene timers y elimina el misil
            return;
        }

        // Verifica colisiones con otros objetos
        QList<QGraphicsItem*> colisiones = collidingItems();
        for (QGraphicsItem* item : colisiones) {
            // Si colisiona con Goku
            GokuNube* gokuColisionado = dynamic_cast<GokuNube*>(item);
            if (gokuColisionado) {
                gokuColisionado->recibirDanio(1);  // Goku recibe daño
                if (scene()) scene()->removeItem(this);  // Se quita el misil de la escena
                limpiarYEliminar();  // Se elimina el misil correctamente
                return;
            }
        }

        // Si el misil ya salió de la pantalla por la izquierda
        if (x() + boundingRect().width() < 0) {
            if (scene()) scene()->removeItem(this);
            limpiarYEliminar();
        }
    }
    catch (const std::exception& e) {
        qDebug() << "Excepción en Misil::mover:" << e.what();
        limpiarYEliminar();  // Si hay un error, eliminar el misil de forma segura
    }
}
```

```cpp
// Método que mueve al avión enemigo con trayectoria oscilante y verifica colisión con Goku
void AvionEnemigo::mover() {
    // Si el objeto no está en una escena válida, se elimina
    if (!scene() || !goku || !goku->scene()) {
        deleteLater();
        return;
    }

    // Si el avión no es estacionario o ya terminó de disparar
    if (!esEstacionario || haTerminadoDisparos) {
        tiempo += 0.1;  // Incrementa el tiempo para el movimiento oscilante

        // Calcula nueva posición en Y con movimiento senoidal
        float nuevaY = baseY + amplitud * std::sin(omega * tiempo);
        setPos(x() - velocidadX, nuevaY);  // Se mueve a la izquierda y cambia su altura
    }

    // Si colisiona con Goku
    if (collidesWithItem(goku)) {
        emit colisionaConGoku();  // Emite señal de colisión
        if (scene()) scene()->removeItem(this);
        deleteLater();  // Elimina el avión
        return;
    }

    // Si el avión ya salió de la pantalla
    if (x() + boundingRect().width() < 0) {
        if (scene()) scene()->removeItem(this);
        deleteLater();
    }
}
```

```cpp
// Metodo que mueve la granada, aplica gravedad, y genera la explosion al colisionar
void Granada::mover() {
    // Si ya está explotando, mostrar los frames de la animación de explosión
    if (explotando) {
        int explosionAncho = 32;
        int explosionAlto = 45;
        if (frameExplosion < 12) {
            setPixmap(explosionSprite.copy(frameExplosion * explosionAncho, 0, explosionAncho, explosionAlto));
            frameExplosion++;
        } else {
            if (scene()) scene()->removeItem(this);  // Elimina el objeto de la escena
            deleteLater();
        }
        return;
    }
    // Aplicar gravedad al movimiento vertical
    velocidadY += gravedad;
    moveBy(velocidadX, velocidadY);  // Movimiento parabólico
    // Cambiar sprite de la granada para animación
    if (!hojaGranada.isNull() && (frameActual * spriteAncho < hojaGranada.width())) {
        setPixmap(hojaGranada.copy(frameActual * spriteAncho, 0, spriteAncho, spriteAlto));
        frameActual = (frameActual + 1) % 17;
    }
    // Si colisiona con el objetivo
    if (!yaColisiono && objetivo && scene() && objetivo->scene() == scene() && collidesWithItem(objetivo)) {
        yaColisiono = true;
        if (!devuelta) {
            Personaje* personajeObjetivo = dynamic_cast<Personaje*>(objetivo);
            if (personajeObjetivo) {
                personajeObjetivo->recibirDanio(1);  // Aplica daño
                qDebug() << "La granada dañó al jugador.";
            }
        }
        explotar();  // Inicia explosión
        return;
    }
    // Si cae al suelo sin chocar
    if (y() > 550) {
        explotar();
    }
```

Problemas de desarrollo que afrontamos

Sincronización de eventos visuales y efectos de sonido:

Tuvimos dificultades para lograr que los efectos de sonido coincidieran con precisión con los eventos visuales, lo que requirió realizar ajustes finos y calibraciones en el código.

Manejo de múltiples ventanas QGraphicsView:

Al trabajar con varias ventanas, detectamos que si no se cerraban correctamente, quedaban ventanas blancas abiertas, lo cual afectaba negativamente la experiencia del usuario

Ajuste del fondo del menú:

Fue un reto configurar el fondo del menú para que se mostrará completo y sin deformaciones, sobre todo al cambiar el tamaño de la ventana o la resolución.

Detección precisa de colisiones entre sprites animados:

Queríamos que la detección de colisiones fuera lo más precisa posible, lo que se complicó por la animación y diferentes tamaños de los sprites.

Manejo de memoria dinámica y cierre de escenas:

Se hizo necesario un manejo cuidadoso de la memoria dinámica y el cierre adecuado de las escenas para evitar errores, fugas de memoria y asegurar la estabilidad del juego.

Evolución de la solución y consideraciones finales

Cambios realizados desde el Momento I:

A partir de una idea general inspirada en los capítulos 64 y 65 de *Dragon Ball*, fuimos concretando una propuesta sólida que dio lugar a un videojuego funcional con tres niveles, cada uno con mecánicas distintas pero conectadas por una narrativa coherente.

Se aplicaron principios de programación orientada a objetos, con herencias propias y manejo de excepciones para controlar errores en la carga de recursos. Se mejoró también la gestión de ventanas, reemplazando la apertura de múltiples vistas por una única ventana principal que actualiza su contenido según el nivel, haciendo más limpia y eficiente la experiencia del usuario.

Además, se introdujo una dificultad progresiva visible en:

- Aumento de la velocidad y frecuencia de los obstáculos.
- Incremento en la cantidad de enemigos y elementos en pantalla.

Consideraciones finales:

El videojuego incluye los **tres modelos físicos** requeridos :

- **Gravedad**: aplicada al movimiento de Tao Pai Pai en su ascenso vertical.
- **Trayectoria parabólica**: usada en el lanzamiento de granadas por parte de Tao Pai Pai en el combate contra Goku.
- **Movimiento armónico simple**: presente en el patrón de movimiento de algunos aviones enemigos durante el vuelo hacia la base.

(El movimiento rectilíneo simple no se considera como modelo físico válido para efectos del reto, por lo que no se cuenta).

El videojuego fue desarrollado completamente en Qt, empleando memoria dinámica, estructura modular con POO, y elementos visuales y sonoros como fondos musicales, efectos de evento y pantallas de Game Over/Victoria con opción para volver al menú principal.