UNIVERSIDAD DE ANTIOQUIA
FACULTA DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE TELECOMUNICACIONES
SEMESTRE II

Desafío 3
Creación de videojuego en lenguaje C++
Documentación Bart contra Australia

Asignatura:
Informática II Teoría

Autores:
Oscar Miguel López Peñata
Luis Carlos Romero Cardenas

Tutores:
Anibal Guerra
Augusto Salazar

Medellín, Diciembre de 2024

INTRODUCCIÓN

El desarrollo del videojuego Bart vs. Australia surge como un desafío de programación en el marco del aprendizaje de lenguaje C++, con el propósito de implementar conceptos avanzados de programación orientada a objetos (POO), manejo de estructuras de datos dinámicas y patrones de diseño. Este proyecto tiene como base narrativa el episodio "Bart contra Australia" de la reconocida serie animada Los Simpson, adaptando su trama y personajes a un entorno interactivo que pone a prueba tanto las habilidades técnicas del desarrollador como la creatividad en el diseño del juego.

El diseño del juego se ha basado en principios sólidos de POO, empleando clases para abstraer entidades clave como Bart, objetos del inventario, enemigos, y niveles. La implementación se apoya en el uso eficiente de la Plantilla STL de C++, utilizando contenedores como std::vector y std::unique_ptr para gestionar recursos dinámicos y garantizar una administración segura de la memoria.

Este proyecto no solo constituye un ejercicio práctico en programación de videojuegos, sino también un medio para explorar las posibilidades del lenguaje C++ en la creación de sistemas robustos y eficientes. A lo largo del desarrollo, se han enfrentado y resuelto problemas de lógica, optimización, y diseño de software, lo que ha contribuido al fortalecimiento de competencias técnicas en un entorno de programación realista.

El videojuego Bart vs. Australia es un ejemplo destacado del potencial de C++ para desarrollar aplicaciones interactivas que no solo destacan por su lógica compleja, sino también por su capacidad de adaptarse a narrativas dinámicas y sistemas escalables. A través del uso de técnicas avanzadas de programación orientada a objetos, este proyecto demuestra cómo un lenguaje de propósito general como C++ puede ser utilizado para abstraer entidades del mundo real, gestionar interacciones entre ellas y optimizar el rendimiento en tiempo de ejecución.

Este proyecto resalta cómo la precisión técnica del C++ puede fusionarse con un diseño creativo, logrando que cada componente, desde los personajes hasta los sistemas de niveles y enemigos, contribuya de manera cohesionada a una experiencia de usuario inmersiva. Asimismo, el enfoque en la reutilización de código y la modularidad facilita futuras expansiones y modificaciones del juego, mostrando cómo C++ no solo es una herramienta poderosa, sino también versátil para proyectos de software interactivo.

OBJETIVOS

General

Diseñar e implementar un videojuego interactivo, denominado Bart vs. Australia, utilizando el lenguaje C++ y aplicando conceptos avanzados de programación orientada a objetos (POO), con el propósito de desarrollar una experiencia de juego dinámica que integre elementos narrativos, lógicos y técnicos, mientras se refuerzan las competencias en el desarrollo de software eficiente y estructurado.

Específicos

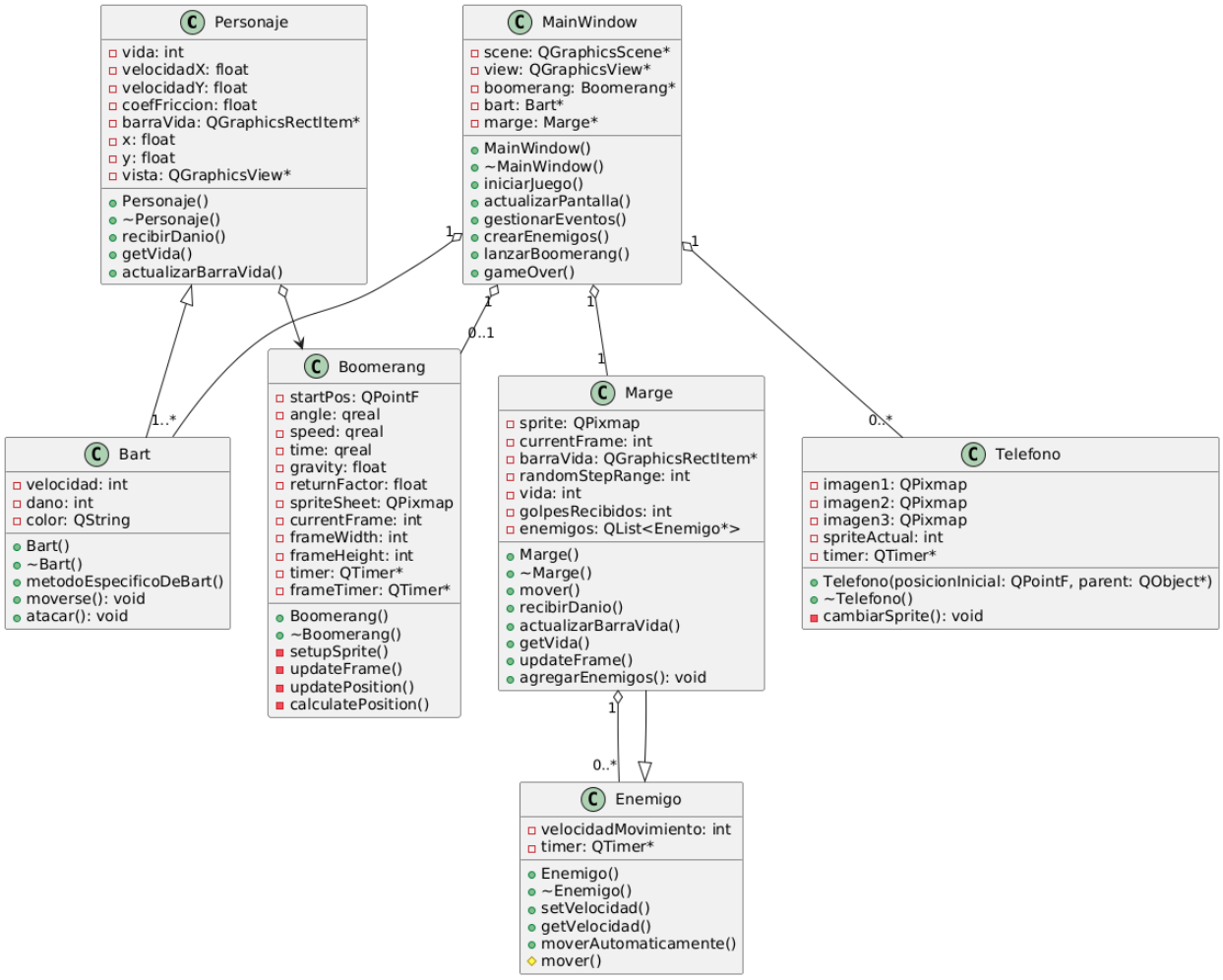
Representar las entidades principales del videojuego, como personajes, objetos y enemigos, utilizando clases e implementando herencia y polimorfismo para fomentar la reutilización del código y garantizar una estructura robusta y modular.

Diseñar y desarrollar un sistema que permita gestionar la progresión del jugador a través de niveles, integrando retos específicos y adaptando las dificultades de acuerdo con la narrativa y las acciones realizadas por el jugador.

Emplear contenedores estándar de la STL (Standard Template Library) de C++ para gestionar dinámicamente inventarios, enemigos y niveles, asegurando un uso eficiente de la memoria y la correcta administración de los recursos durante la ejecución del programa.

Diseñar las clases y sus métodos siguiendo los principios fundamentales de encapsulamiento y abstracción, promoviendo un código que sea fácil de mantener, ampliar y entender, y que permita la interacción controlada entre las diferentes partes del sistema.

DIAGRAMA DE CLASES



Análisis del diagrama de clases

El diagrama de clases muestra cómo se estructuran las clases principales del juego y sus relaciones. A continuación, se explica cada clase y su relación con otras clases:

1. Personaje

Atributos

- vida, velocidadX, velocidadY, coefFriccion: Atributos que definen las características físicas y la salud del personaje.
- barraVida: Un puntero a un QGraphicsRectItem, que representa gráficamente la barra de vida del personaje.
- x, y: Coordenadas que definen la posición del personaje en el escenario.

- vista: Un puntero a un QGraphicsView, que proporciona la vista del juego.

Métodos

- recibirDanio(): Método que reduce la vida del personaje cuando recibe un ataque.
- getVida(): Método para obtener la cantidad actual de vida del personaje.
- actualizarBarraVida(): Actualiza visualmente la barra de vida del personaje.

Relación con otras clases

La clase Personaje es la superclase de Bart y Enemigo. Esto significa que ambas clases heredan sus atributos y métodos.

2. Bart (subclase de Personaje)

Atributos y métodos específicos

- La clase Bart hereda de Personaje, pero agrega un comportamiento específico para el personaje de Bart.
- Tiene un método adicional: metodoEspecificoDeBart(), que podría ser una acción o movimiento único que solo Bart puede ejecutar.

Relación con otras clases

Bart es una subclase de Personaje, lo que significa que hereda las propiedades y métodos de Personaje.

3. Enemigo (subclase de Personaje)

Atributos

- velocidadMovimiento: Define la velocidad a la que se mueve el enemigo.
- timer: Un puntero a un QTimer, que podría ser usado para mover al enemigo automáticamente o para otras acciones relacionadas con el tiempo.

Métodos

- setVelocidad(): Define la velocidad del enemigo.
- moverAutomaticamente(): Controla el movimiento del enemigo de forma automática en el escenario.

Relación con otras clases

- Enemigo es una subclase de Personaje, heredando las propiedades y comportamientos básicos.
- Enemigo está relacionado con Marge, ya que Marge gestiona y controla a los enemigos en el juego.

4. Marge (subclase de Enemigo)

Atributos

- sprite, currentFrame, barraVida, randomStepRange, vida, golpesRecibidos: Atributos específicos de Marge, que definen su apariencia, su salud y su comportamiento.
- enemigos: Lista de punteros a Enemigo, lo que indica que Marge puede tener varios enemigos asociados.

Métodos

- mover(), recibirDanio(), actualizarBarraVida(), getVida(), updateFrame(): Métodos que definen el comportamiento de Marge en cuanto a movimiento, daño y actualización de su barra de vida.
- agregarEnemigos(): Método que permite agregar enemigos a la lista de enemigos de Marge.

Relación con otras clases

Marge es una subclase de Enemigo, y tiene una relación de composición con los enemigos, lo que significa que Marge controla la existencia de estos enemigos.

5. Boomerang

Atributos

- startPos, angle, speed, time, gravity, returnFactor: Atributos que controlan la física y el movimiento del boomerang.
- spriteSheet, currentFrame, frameWidth, frameHeight: Atributos relacionados con la animación del boomerang.
- timer, frameTimer: Temporizadores que controlan la actualización del boomerang en pantalla.

Métodos

- setupSprite(), updateFrame(), updatePosition(), calculatePosition(): Métodos encargados de gestionar el sprite, el movimiento y la física del boomerang.

Relación con otras clases

Boomerang está asociado a Personaje a través de una relación de agregación. Un personaje puede tener un boomerang, pero no es necesario que todos los personajes lo tengan.

6. MainWindow

Atributos

- scene, view, boomerang, bart, marge: Punteros a los componentes principales del juego, como la escena gráfica, la vista, y las instancias de los personajes y objetos.

Métodos

- iniciarJuego(), actualizarPantalla(), gestionarEventos(), crearEnemigos(), lanzarBoomerang(), gameOver(): Métodos que gestionan el flujo principal del juego, incluyendo la inicialización, actualización de la pantalla, y la lógica de eventos y acciones del juego.

Relación con otras clases

- MainWindow tiene relaciones de composición con Bart, Boomerang y Telefono, lo que indica que estos objetos están gestionados y controlados por la clase MainWindow.
- MainWindow tiene una relación con Marge para gestionar los enemigos en el juego.

7. Telefono

Atributos

- imagen1, imagen2, imagen3, spriteActual, timer: Atributos que gestionan la apariencia y el comportamiento del teléfono en el juego.

Métodos

- cambiarSprite(): Método que cambia la imagen del teléfono de manera cíclica.

Relación con otras clases

Telefono tiene una relación con MainWindow, lo que significa que el teléfono está presente dentro de la escena del juego gestionada por la clase MainWindow.

Resumen de las Multiplicidades en el diagrama de clases

Relación	Clases involucradas	Multiplicidad
Herencia	Personaje	<-- Bart
Herencia	Personaje	<-- Enemigo
Herencia	Enemigo	<-- Marge
Composición	MainWindow o-- Bart	1 a 1
Composición	MainWindow o-- Boomerang	1 a 0..1
Composición	MainWindow o-- Marge	1 a 0..*
Agregación	Personaje o--> Boomerang	0..1 a 0..1
Composición	MainWindow o-- Telefono	1 a 0..1

Explicación de las relaciones

Herencia

La herencia entre Personaje, Bart, y Enemigo establece una jerarquía donde Bart y Enemigo heredan las características básicas de un Personaje, pero pueden comportarse de manera diferente debido a su especialización (por ejemplo, Bart tiene un método único, y Enemigo tiene métodos de movimiento automatizado).

Composición

MainWindow actúa como el controlador principal del juego y contiene los elementos claves como el Bart, Boomerang, Marge y Telefono. Cada uno de estos elementos está contenido en MainWindow, lo que significa que el ciclo de vida de estos objetos está gestionado por MainWindow.

Agregación

El Boomerang está agregado al Personaje (y no necesariamente al MainWindow), lo que implica que un personaje puede lanzar un boomerang, pero este boomerang puede existir independientemente del personaje.

DOCUMENTACIÓN DEL JUEGO

Objetivo del juego

El objetivo del juego es controlar a los personajes (como Bart y Marge) y enfrentar a los enemigos, gestionando la interacción con diversos elementos como boomerangs y teléfonos. Los jugadores deberán moverse, atacar a los enemigos y evitar daños mientras avanzan a través de las fases del juego.

Componentes principales del juego

Personaje

- Cada jugador controla un personaje en el juego. Los personajes pueden recibir daño, moverse y actualizar su barra de vida.
- Los personajes tienen atributos como vida, velocidadX, velocidadY, y coefFriccion, que afectan su movimiento y su resistencia a los ataques.

Bart

- Una subclase de Personaje, que tiene características y comportamientos específicos, como un ataque especial (metodoEspecificoDeBart()).
- Bart puede moverse y atacar a los enemigos.

Enemigo

- Los enemigos se mueven automáticamente en el juego, con una velocidad controlada por el atributo velocidadMovimiento.
- Los enemigos son gestionados por la clase Marge, que también puede recibir daño y tiene su propia barra de vida.

Marge

- Marge es una subclase de Enemigo. Se encarga de controlar el comportamiento de los enemigos dentro del juego y su interacción con los personajes.
- Marge tiene un conjunto de enemigos asociados que se gestionan dinámicamente en el juego.

Boomerang

- Un objeto lanzable por el jugador que tiene una trayectoria determinada. Se mueve en función de la gravedad y el ángulo, y puede regresar al personaje que lo lanzó.
- La clase Boomerang controla la animación del boomerang y su movimiento a través del escenario.

MainWindow

- La clase principal que gestiona la escena del juego. Aquí se agregan todos los elementos gráficos, incluidos los personajes, los enemigos y los objetos interactivos.
- MainWindow también gestiona los eventos de juego como iniciar el juego, actualizar la pantalla, crear enemigos, lanzar boomerangs y manejar el fin del juego.
- Tiene métodos como iniciarJuego(), actualizarPantalla(), y gestionarEventos().

Telefono

- Un objeto interactivo dentro del juego que cambia de apariencia cada vez que el temporizador se dispara.
- Se utiliza un QTimer para cambiar el sprite de la imagen del teléfono cada 500 ms.

Movimiento de Bart

En el diseño del juego, cuando el usuario presiona las teclas W, A, S o D, el personaje (en este caso Bart) se mueve en las direcciones correspondientes: arriba, izquierda, abajo y derecha. El control de estas teclas se puede implementar utilizando el método keyPressEvent() en la clase MainWindow, y este control afectará la posición de Bart actualizando sus coordenadas x y y según la tecla presionada.

Modelos físicos implementados en el juego

Los modelos físicos utilizados en este juego toma en cuenta factores básicos como la posición, la velocidad y la aceleración para modelar cómo se mueve un personaje en el espacio del juego. El movimiento de los personajes se controla principalmente a través de la manipulación de las coordenadas x y y de cada objeto, las cuales corresponden a su ubicación en el entorno gráfico. A partir de estas coordenadas, se calcula el cambio en la posición en función

del tiempo y la velocidad, a continuación se describe teórica y físicamente cada uno de los modelos físicos:

1. **Modelo de Fricción en Bart**

La fricción es un fenómeno físico que se opone al movimiento de un objeto. En el caso del modelo implementado para `Bart`, la fricción se aplica de forma simplificada utilizando un coeficiente de fricción, que reduce la velocidad en cada actualización.

Fórmulas utilizadas:

- La fricción actúa sobre la velocidad en las direcciones X y Y. En el modelo que implementamos, la fricción se aplica como una reducción constante en la velocidad, que es proporcional al coeficiente de fricción.

Para **velocidad horizontal (X)**:

$$\text{velocidadX}(t) = \text{velocidadX}(t-1) - \text{coefFriccion} \quad \text{si} \quad \text{velocidadX}(t-1) > 0$$

$$\text{velocidadX}(t) = \text{velocidadX}(t-1) + \text{coefFriccion} \quad \text{si} \quad \text{velocidadX}(t-1) < 0$$

Para **velocidad vertical (Y)**:

$$\text{velocidadY}(t) = \text{velocidadY}(t-1) - \text{coefFriccion} \quad \text{si} \quad \text{velocidadY}(t-1) > 0$$

$$\text{velocidadY}(t) = \text{velocidadY}(t-1) + \text{coefFriccion} \quad \text{si} \quad \text{velocidadY}(t-1) < 0$$

Luego, la **posición** se actualiza con la siguiente fórmula:

$$x(t) = x(t-1) + \text{velocidadX}(t)$$

$$y(t) = y(t-1) + \text{velocidadY}(t)$$

2. **Modelo de Caminatas Aleatorias para Marge**

La caminata aleatoria es un modelo en el cual el movimiento del objeto es aleatorio, y cada paso se da en una dirección aleatoria (en este caso, en el eje Y para `Marge`).

Fórmula utilizada:

La posición de `Marge` en cada iteración se actualiza con un valor aleatorio, que representa un paso hacia arriba o hacia abajo. La fórmula es:

$$y(t) = y(t-1) + \Delta y(t)$$

Donde $\Delta y(t)$ es un valor aleatorio entre $-\text{randomStepRange}$ y $+\text{randomStepRange}$, que representa el paso aleatorio en el eje Y en cada actualización.

$$\Delta y(t) = \text{QRandomGenerator::global()->bounded}(-\text{randomStepRange}, \text{randomStepRange})$$

El valor de `randomStepRange` controla cuánta variabilidad puede tener el movimiento de `Marge` en cada paso. Este valor es ajustable para hacer que el movimiento sea más errático o más suave.

3. **Modelo Parabólico para el Boomerang**

El movimiento parabólico describe cómo un objeto se mueve bajo la influencia de la gravedad, es decir, sigue una trayectoria curva. En el caso del boomerang, su movimiento depende de su velocidad inicial y el ángulo de lanzamiento, y está influenciado por la aceleración debida a la gravedad.

Fórmulas utilizadas:

La trayectoria del boomerang se puede modelar con las ecuaciones clásicas del movimiento rectilíneo uniformemente acelerado en dos dimensiones (en el plano XY):

Componente horizontal (X):

$$x(t) = x_0 + v_{x0} \cdot t$$

donde:

- x_0 es la posición inicial en X,
- v_{x0} es la velocidad inicial en X,
- t es el tiempo.

Componente vertical (Y):

$$y(t) = y_0 + v_{y0} \cdot t + \frac{1}{2} \cdot g \cdot t^2$$

donde:

- y_0 es la posición inicial en Y,
- v_{y0} es la velocidad inicial en Y,
- g es la aceleración debida a la gravedad (aproximadamente $9.81\, m/s^2$),
- t es el tiempo.

Desglosado en el código:

1. **Velocidades iniciales en X y Y**: Cuando el boomerang es lanzado, su velocidad inicial en X y Y se calcula con las siguientes fórmulas usando el ángulo de lanzamiento:

   $$v_{x0} = v_0 \cdot \cos(\theta)$$

   $$v_{y0} = v_0 \cdot \sin(\theta)$$

   donde v_0 es la velocidad inicial del boomerang y θ es el ángulo de lanzamiento.

2. **Actualización de la posición en cada frame**: En cada iteración, el boomerang se mueve de

donde v_0 es la velocidad inicial del boomerang y θ es el ángulo de lanzamiento.

2. **Actualización de la posición en cada frame**: En cada iteración, el boomerang se mueve de acuerdo con la fórmula de la posición, ajustando la velocidad en Y debido a la gravedad:

$$v_y(t) = v_y(t-1) + g \cdot \Delta t$$

donde Δt es el intervalo de tiempo entre actualizaciones (por ejemplo, 16 ms o 1/60 segundos), y luego:

y(t) = y(t-1) + v_y(t) \cdot \Delta t] Esto simula el ascenso y posterior descenso del boomerang a medida que se ve afectado por la gravedad.

Resumen de las Fórmulas:

1. **Fricción en Bart**:
   - $\text{velocidadX}(t) = \text{velocidadX}(t-1) - \text{coefFriccion}$ (si $\text{velocidadX}(t-1) > 0$)
   - $x(t) = x(t-1) + \text{velocidadX}(t)$
   - $y(t) = y(t-1) + \text{velocidadY}(t)$
2. **Caminata Aleatoria de Marge**:
   - $y(t) = y(t-1) + \Delta y(t)$
   - $\Delta y(t) = \text{QRandomGenerator::global()->bounded}(-\text{randomStepRange}, \text{randomStepRange})$
3. **Movimiento Parabólico del Boomerang**:
   - $x(t) = x_0 + v_{x0} \cdot t$
   - $y(t) = y_0 + v_{y0} \cdot t + \frac{1}{2} \cdot g \cdot t^2$
   - $v_{x0} = v_0 \cdot \cos(\theta)$
   - $v_{y0} = v_0 \cdot \sin(\theta)$

DIFICULTADES Y DESAFÍOS ENCONTRADOS EN EL DESARROLLO

El desarrollo del juego presentó varios desafíos y dificultades que pusieron a prueba la capacidad del equipo para resolver problemas técnicos y mejorar la experiencia general de juego. A continuación, se detallan los principales obstáculos encontrados durante el proceso de desarrollo y las soluciones implementadas para superarlos.

Implementación del movimiento de los personajes

Uno de los primeros desafíos fue crear un sistema de movimiento fluido y responsivo para los personajes, en especial para Bart. La dificultad radicaba en garantizar que los movimientos fueran precisos y que el control respondiera de manera inmediata a las entradas del jugador (es decir, las teclas presionadas).

Problema. El movimiento no se sentía fluido debido a la implementación inicial de la lógica, donde la actualización de las posiciones no estaba sincronizada adecuadamente con los intervalos de tiempo.

Solución. Para resolver esto, se implementó un sistema de actualización de la posición basado en el manejo adecuado de las teclas y un modelo de movimiento discreto, que permite una actualización constante de la posición del personaje. Además, se ajustaron las velocidades en los ejes X y Y de manera independiente, lo que permitió un control más preciso sobre el movimiento.

Interacción con el entorno (Colisiones y límites)

A medida que se implementaba el movimiento de los personajes, también surgieron problemas relacionados con las colisiones y los límites del entorno del juego. Los personajes debían interactuar con los objetos del escenario sin salirse de los bordes del mundo virtual o atravesar paredes, lo que representaba un desafío técnico importante.

Problema. Los personajes podían atravesar objetos del entorno o moverse fuera de los límites visibles, lo que afectaba la jugabilidad y la inmersión del jugador.

Solución. Para corregir esto, se implementaron métodos de detección de colisionesque verifican la proximidad de los personajes con los objetos del escenario. Además, se ajustaron las restricciones de la pantalla para asegurarse de que los personajes se mantuvieran dentro de los

límites visibles. Esto implicó el uso de condiciones que verifican las posiciones X e Y de los personajes y aseguran que no puedan exceder los márgenes del área de juego.

Implementación de la barra de vida

El sistema de barra de vida fue otro desafío importante. Cada personaje, como Bart y los enemigos, tiene una barra de vida visual que debe actualizarse cada vez que el personaje recibe daño. Esta barra debía reflejar con precisión el estado de salud del personaje.

Problema. Inicialmente, la barra de vida no se actualizaba correctamente al recibir daño, lo que generaba inconsistencias visuales. Además, la animación de la barra de vida no era suficientemente fluida, lo que afectaba la experiencia de juego.

Solución. Se reestructuró la lógica que controla el cálculo y la representación de la barra de vida, utilizando una combinación de QGraphicsRectItem para representar visualmente la barra y actualizaciones periódicas en la interfaz gráfica. También se implementaron métodos para controlar la velocidad de actualización de la barra de vida, asegurando que se viera un cambio gradual en función de la cantidad de daño recibido.

Manejo de la animación del personaje

Otro reto fue el manejo de la animación de los personajes, especialmente en el caso de Bart. Cada acción del personaje (caminar, recibir daño, etc.) debe estar acompañada de una animación que la represente de manera visual.

Problema. La sincronización entre la animación y las acciones del personaje no estaba bien implementada, lo que causaba que las animaciones se reprodujeran de forma errática o sin coincidir con las acciones que el jugador realizaba.

Solución. Para resolver este problema, se implementaron frames de animación controlados por temporizadores que se actualizan solo cuando el personaje realiza una acción específica, como caminar o recibir daño. Esto aseguró que las animaciones se reprodujeran de manera fluida y sincronizada con las acciones del personaje.

RESULTADOS FINALES

El diseño original del juego contemplaba la implementación de varios niveles con distintas mecánicas y desafíos, lo que hubiera proporcionado una experiencia de juego dinámica y progresiva. Sin embargo, debido a diversas dificultades técnicas y limitaciones en el tiempo de desarrollo, solo se logró completar el primer nivel, Bart y la broma internacional. A continuación, se detallan los principales inconvenientes y causas que impidieron la realización del resto de los niveles planeados.

El principal obstáculo fue la falta de tiempo para implementar la totalidad del proyecto. La creación de un juego interactivo con varios niveles requiere una gran cantidad de tiempo para diseñar y desarrollar tanto los gráficos como la programación de las mecánicas de cada nivel. A medida que se avanzaba en el desarrollo del primer nivel, se hizo evidente que el plazo disponible no permitía profundizar en la implementación de los niveles posteriores. La complejidad creciente de cada nivel, especialmente con las interacciones con los nuevos personajes y objetos, requería más tiempo del inicialmente estimado.

Cada nivel planteaba nuevos desafíos que implicaban la integración de mecánicas más complejas, como la evasión de enemigos (guardias en el aeropuerto y en el consulado) y el manejo de objetos interactivos (como el boomerang). Aunque la programación de las interacciones básicas de Bart con el teléfono en el primer nivel se logró con éxito, la implementación de nuevas mecánicas, como la evasión de los enemigos, la recolección de objetos y las interacciones más complejas, habría requerido de un sistema más avanzado de inteligencia artificial (IA) y control de la física de movimiento. La falta de experiencia con algunos de estos sistemas y el tiempo limitado contribuyeron a que no se pudieran desarrollar correctamente.

Los gráficos del juego son esenciales para crear una experiencia inmersiva. El primer nivel se completó con gráficos adecuados para la escena de Bart en su habitación y la interacción con el teléfono. Sin embargo, la creación de nuevos entornos como el aeropuerto australiano o el consulado, con sus respectivos personajes y obstáculos, habría requerido más recursos gráficos, como modelos 3D, texturas y animaciones adicionales. La creación de estos elementos visuales fue una tarea ardua que no se pudo completar dentro del tiempo disponible, lo que contribuyó al retraso en el desarrollo de los niveles siguientes.

Cada nivel adicional introducía interacciones avanzadas, como el uso de pociones de velocidad, el movimiento autónomo de la bota gigante y los guardias, y la recolección de objetos dentro del entorno. La implementación de estas interacciones no solo requería programar las acciones del personaje principal (Bart), sino también asegurarse de que los enemigos y objetos reaccionaran de manera coherente dentro de las reglas del juego. Además, la programación de la evasión de los guardias y la necesidad de evitar la captura sin perder la coherencia del juego resultaron ser tareas complicadas que no pudieron completarse por completo.

El desarrollo de un videojuego implica no solo la programación, sino también la realización de pruebas exhaustivas para garantizar que las mecánicas de juego sean divertidas y jugables. Aunque se completó el primer nivel y las mecánicas básicas funcionaban correctamente, no se tuvo suficiente tiempo para realizar pruebas de jugabilidad en los niveles posteriores. Las pruebas son esenciales para identificar posibles errores o áreas de mejora en la experiencia del jugador, lo que, a su vez, permite hacer ajustes finos en las mecánicas y la dificultad del juego.

CONCLUSIONES

El desarrollo del videojuego ha sido un proceso enriquecedor y desafiante que ha implicado la aplicación de conceptos clave en programación, interacción gráfica y física de movimiento. A lo largo de este proyecto, se ha logrado crear un entorno de juego interactivo y dinámico, que no solo pone a prueba las habilidades técnicas del jugador, sino que también ofrece una experiencia divertida y desafiante. Este trabajo ha permitido aplicar y perfeccionar diversas competencias técnicas, a la vez que se han superado obstáculos propios del desarrollo de videojuegos. A continuación, se detallan las principales conclusiones basadas en las decisiones de diseño, los desafíos superados y las lecciones aprendidas durante el proceso.

En primer lugar, uno de los aspectos más fundamentales en el diseño de un videojuego es el movimiento de los personajes, que debe ser fluido y responder de manera precisa a las acciones del jugador. En este caso, la implementación del movimiento de Bart y los demás personajes se basó en el uso de teclas de dirección, lo que permitió un control directo y eficiente. Sin embargo, uno de los principales retos fue asegurarse de que la respuesta del movimiento fuera inmediata y sin lag, lo que se consiguió mediante la sincronización del sistema de actualización de la posición con los tiempos de entrada del jugador. Gracias a esto, se logró obtener un movimiento preciso y controlable, lo que permite al jugador interactuar de manera efectiva con el personaje en el entorno del juego.

Además, la gestión de la interacción entre los personajes y el entorno fue esencial para proporcionar una experiencia coherente. El reto principal fue implementar un sistema de detección de colisiones que asegurara que los personajes no atravesaran objetos del escenario ni se desplazaran fuera de los límites del área de juego. Para ello, se resolvió el problema mediante la incorporación de condiciones que verifican las posiciones de los personajes y la implementación de restricciones de pantalla que limitan su movimiento dentro del área visible. Esta estrategia permitió garantizar que los personajes pudieran interactuar correctamente con su entorno, creando una experiencia más inmersiva y realista.

La correcta animación de los personajes también fue un aspecto clave para mantener el interés visual del jugador. Se implementaron animaciones para las acciones de Bart, los enemigos y los objetos, lo que requería una sincronización precisa con las acciones realizadas. Esto se logró mediante el uso de frames de animación controlados por temporizadores, lo que permitió que las animaciones se actualizaran de acuerdo a las acciones de los personajes, como

caminar, recibir daño y lanzar el boomerang. De esta forma, se consiguió un movimiento fluido y realista, mejorando la inmersión del jugador en el juego.

No obstante, a medida que se añadían más personajes, enemigos y objetos a la escena, el rendimiento del juego comenzó a verse afectado. Por lo tanto, la optimización de recursos fue un aspecto crucial para garantizar un rendimiento adecuado. Se implementaron varias técnicas de optimización de recursos, como la carga diferida de imágenes y la actualización por lotes de elementos en la escena. Estas optimizaciones permitieron mejorar la tasa de fotogramas y reducir el consumo de recursos, asegurando que el juego se mantuviera fluido incluso en escenas con muchos elementos en movimiento.

En cuanto a la gestión de la barra de vida, uno de los elementos más importantes del juego, su implementación fue clave para representar visualmente la salud de los personajes y la interacción con los enemigos. Al recibir daño, la barra de vida se actualiza de manera visual, lo que permite al jugador observar de manera inmediata el impacto de las acciones en la salud del personaje. Para lograrlo, se utilizó QGraphicsRectItem y se realizaron actualizaciones periódicas basadas en las acciones del personaje, asegurando que la barra de vida se mantuviera coherente con el estado del personaje.

Por último, el boomerang, uno de los objetos clave del juego, agregó una capa de complejidad y estrategia. Implementar un sistema que simule su trayectoria de manera realista, detecte colisiones con los enemigos y permita su retorno, fue una tarea desafiante. No obstante, se logró gracias a la implementación de fórmulas físicas que simulan su movimiento, lo que dio como resultado un objeto con un comportamiento natural y efectivo dentro del juego.