UNIVERSIDAD DE ANTIOQUIA

Facultad de Ingeniería

Etapa 4: Entrega Final

Informática II

Mario Esteban Estrada Gonzalez

CC: 1233191679

Juan José Díaz Zuluaga

CC: 1001456540

Docente

Aníbal José Guerra Soler

2023-2

UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA

SEDE MEDELLÍN

Diciembre 2023

Medellín, Colombia

Introducción

El juego se sitúa en el espacio, en este caso ambos personajes están intentando huir del "Story Master", el enemigo de este capítulo que intenta atraparlos en su tren de historias sin fin. El juego se plantea como un Space - Shooter en el que para lograr huir deben enfrentar a diferentes enemigos, a los cuales deben asesinar desplazándose y disparándoles (con teclado). Para ganar, se escogerá a uno de los personajes que debe ir superando los retos nivel a nivel, en lo que irá aumentando la dificultad y cambiando las mecánicas mientras obtiene ventajas que le ayudarán con su misión. Cada nivel finaliza cuando todos los enemigos hayan muerto

Personajes

Los personajes de esta aventura son Rick y Morty, cada uno de los cuales posee habilidades distintas.

- **Rick:** El abuelo alcohólico de Morty puede infringir un porcentaje de daño más alto (100%) pero a costa de esto será más lento (80%)
- **Morty:** Morty será más ágil por lo que se podrá mover mucho más rápido por la pantalla (100%), mientras que hará menos daño a los enemigos (80%).

Mecánicas Físicas

Las principales físicas del juego se definen a través de movimientos conocidos de la cinemática; en resumen se utilizan 3 movimientos diferentes.

- **MRU:** Tanto el personaje principal, como los enemigos tipo 1 en el primer nivel se mueven con un movimiento rectilíneo uniforme sin aceleraciones. El único cambio es cuando los enemigos colisionan con los límites del juego
- **MUA:** Las balas en los niveles se moverán con MRUA en el caso del primer nivel se hará con MRUA y en el segundo con tiro parabólico
- **Espiral Circular:** Finalmente los enemigos tipo 2 se mueven en una espiral alrededor de toda la pantalla de juego. Este movimiento se mueve utilizando un parámetro de tiempo. Y se descompone en coordenadas cartesianas x,y.

Diseño y Estructura de Datos

Para la realización de este juego con interfaces gráficas se utilizaron dos metodologías principales de programación: Programación Orientada a Objetos (OOP) y la Programación Dirigida a Eventos (EDP); por tal motivo se requiere de una estructuración a través de clases, las cuales son:

Clase **Game**:
La clase Game desempeña un papel central en la lógica del juego, encargándose de la manipulación de la interfaz gráfica y gestionando eventos clave, así como los movimientos del personaje principal, controlado mediante el teclado. Contiene listas (QLists) que almacenan instancias de las clases Ammunition y Enemy. Además, gestiona los cambios de escena a través de botones durante momentos de selección y finalización de nivel.

Clase **Character**:
Character es una clase abstracta que define la vinculación de sprites de personajes a la interfaz gráfica, estableciendo escalas y tamaños, y compartiendo características comunes como velocidad de movimiento y cantidad de vida.

Clase **Main_Character**:
Main_Character hereda públicamente de la clase Character y controla los movimientos de los dos personajes principales (Rick o Morty). Su constructor toma un parámetro para definir el tipo de personaje (1 o 0) y maneja las animaciones de movimiento, incluyendo secuencias de desplazamiento, daño y muerte. Además, gestiona el "bounding rectangle" de los sprites y evalúa las condiciones de colisión.

Clase **Enemy**:
Enemy, similar a Main_Character, hereda de forma pública los atributos de Character. Esta clase se encarga de los movimientos autónomos de los enemigos, incluyendo el uso de timers y señales para controlar secuencias de movimiento y muerte. Gestiona tres enemigos diferentes mediante instancias de la clase.

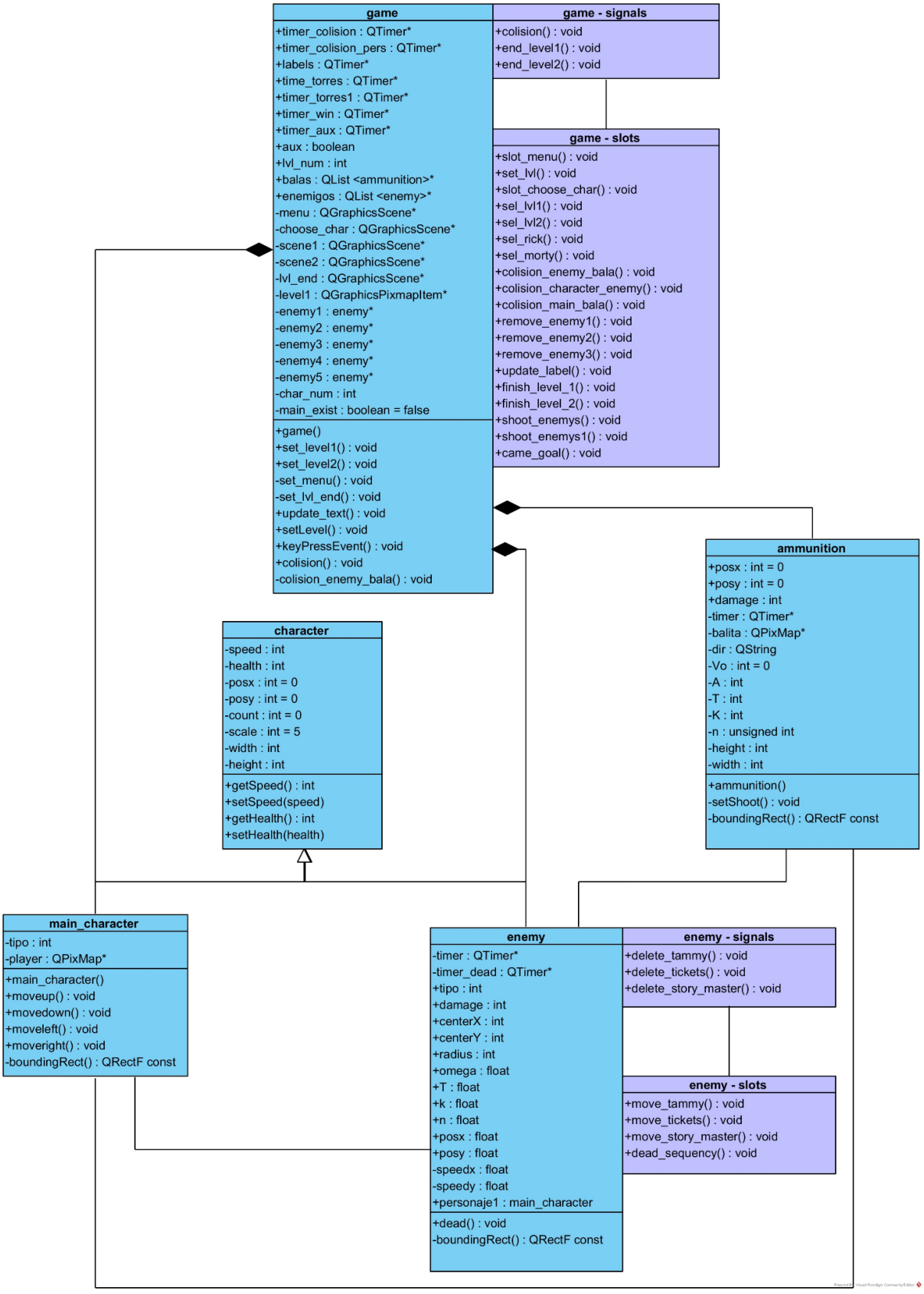
Clase **Ammunition**:
Ammunition se encarga de controlar los disparos del personaje principal, determinando la dirección y velocidad de la bala. Utiliza un timer para manipular la velocidad de la bala, que sigue un movimiento rectilíneo uniformemente acelerado (MRUA). Además, define el "bounding rectangle" de la bala para detectar colisiones con los enemigos.

Contenedores
Para almacenar las instancias de las clases se requerían contenedores y al tener clases que heredan propiedades de QObject, se necesitaba un tipo que permitiese manipular este tipo de objetos; por tal motivo se emplean dos contenedores, uno para los objetos ammunition y otro para los objetos enemy. De esta manera era posible establecer como eran agregados a las escenas y como cambiaban atributos como su posición, vida.

Timers, Señales y Slots
Una parte fundamental de la programación dirigida por eventos implica el empleo de señales y slots, los cuales posibilitan el control de la activación de funciones específicas en la interfaz gráfica cuando se cumple una determinada condición. En este contexto, los timers se utilizan para actualizar continuamente una condición durante la ejecución del programa. Las señales se activan cuando se cumple una condición específica, y el slot es la función que se ejecuta al recibir una señal del programa. Esta dinámica permite establecer conexiones entre los distintos elementos de la escena, tales como los movimientos de los enemigos, los disparos y las muertes de los personajes.

Widgets y Elementos de Escena
En el programa, es esencial considerar todos los elementos que forman parte de la escena, los cuales suelen modelarse como "QObjects". Para manipular eficientemente estos elementos, se utilizan punteros, que ofrecen acceso directo en memoria en lugar de introducirlos cada vez que se requieran. En consecuencia, los contenedores, objetos y escenas son punteros de la clase QObject que se asignan sólo cuando se necesitan y se liberan posteriormente.

Diagrama de Clases



Funciones y Atributos Principales

Clase game:

- *Timers:* Existen múltiples timers en la clase game, ya que con estos se actualizan las animaciones y movimientos en tiempo real. Pero principalmente son importantes los de colisión, ya que cada vez que se cumple un ciclo se verifica si dos de los objetos de la escena están interactuando, por ejemplo las balas y los enemigos.
- *set_level:* Existen dos funciones en las que se centra todo el juego y estas son las de set_level; el trabajo de estas funciones consta en crear las escenas en donde se desarrollan los dos niveles de juego. Aquí se crean los personajes, enemigos y todos los elementos que pertenecen a cada nivel. Es básicamente el "set up" donde todo se inicializa.
- *Escenas:* Para que el nivel se pueda desarrollar, se requiere de un montaje en donde se encuentren los personajes, balas y elementos al tiempo, por ello se crean las escenas, que es el espacio donde interactúan; pero además para seleccionar las condiciones de juego estas también se requieren para cambiar de espacio en donde ocurre todo; es decir realizar la selección del nivel o de los personajes.
- *QLists:* Estos son los contenedores que se emplean para almacenar los objetos de la escena, son vitales para la detección de colisiones, ya que a través de ellos es que se puede identificar qué elementos están chocando y de esta manera es posible modificar sus propiedades y eliminarlos
- *Finalizadores de Nivel:* Estos son slots de la clase game que reciben una señal de cuando alguno de los niveles terminó, ya sea porque se completó el objetivo del nivel, o porque el personaje murió. Aquí se limpian las Qlists utilizadas y se detienen todos los timers que estaban corriente para volver al menú principal.

Clase Character:

- *Atributos de tamaño:* Como se mencionó anteriormente esta es una clase abstracta que se centra más que nada en el manejo de sprites, por lo tanto tiene únicamente atributos como tamaño y escala que permiten controlar qué tan grandes son en la escena.
- *Vida y Velocidad:* Ya que tanto personajes como enemigos requieren de estas características, es posible definirlas también dentro de la clase abstracta; estos atributos también tienen sus métodos getters para usarse dentro del programa principal.

Clase main_character:

- *Funciones de movimiento:* Este es el núcleo de la clase y controla todos las secuencias de movimiento de los personajes principales, esto se hace a través de ciclos de imágenes que cambian cuando se da un evento de tecla presionada (KeyPressEvent)
- *Atributo Tipo:* Este es un valor de tipo entero que se utiliza en el constructor de los objetos instanciados en la clase. Sirve para diferenciar entre rick o morty, y es importante porque esto define las características del personaje principal, como velocidad o vida.

Clase enemy:

- *Timers:* Así como en el programa principal se requieren timers para el movimiento, es posible modelar los movimientos de los enemigos utilizando timers, esto se hace para que sea de forma automática y actualice su posición y sprite de acuerdo a sus ciclos.

- *Variables de Posición:* Debido a que en el juego se utilizaron diferentes físicas de movimiento, las ecuaciones que modelan esto cambian y requieren de más o menos variables, y de acuerdo a esto se tiene en cuenta para modificar esos parámetros
- *Tipo:* Similar a la clase main_character existen tipos de personajes diferentes, con atributos modificables. Por ellos existen enemigos tipo 1,2 o 3 que se modifican y crean en el constructor a partir de esta variable.
- *Dead_sequency:* Este slot es el encargado de hacer la animación de muerte cuando la vida de un enemigo llega a cero y funciona recibiendo una señal que activa la animación

Clase ammunition:

- *Set_shoot: Esta función controla cómo se da el movimiento de la bala de acuerdo a la dirección en la que se encuentra el personaje y recibe la información de una variable en el programa principal. Se encarga fundamentalmente de establecer la posición inicial de la bala.*
- *move_up: Es la función que actualiza la posición de la bala utilizando timers, toma el parámetro dirección y hace que se mueva con un MRUA.*

Reflexiones Finales

Durante el desarrollo de este programa, nos enfrentamos a múltiples desafíos de diversos niveles que nos obligaron a replantear las soluciones de manera innovadora, haciendo uso de la creatividad y las herramientas disponibles, conforme a los temas abordados en el curso.

En particular, tres momentos destacaron por presentar desafíos más significativos que los demás. En primer lugar, comprender cómo se modela con programación orientada a objetos generó numerosas preguntas sobre la interacción entre objetos, la anidación y la integración de todas las partes del sistema. Superar esta dificultad implicó alcanzar una estructura coherente que reflejara fielmente el funcionamiento de las cosas en la vida real. Para lograrlo, empleamos pensamiento estructurado y nos apoyamos en herramientas como los diagramas de clases.

El segundo reto se centró en la interacción entre los diversos elementos que conforman una escena. Descubrir las relaciones adecuadas entre ellos fue crucial. Por ejemplo, al eliminar un enemigo con disparos, era necesario establecer interacciones entre la bala, el personaje principal y el enemigo. Coordinar estos tres elementos en la escena y garantizar su correcto funcionamiento no resultó fácil al principio. Sin embargo, mediante la búsqueda activa y el trabajo en equipo, logramos desarrollar soluciones efectivas.

Finalmente, encontramos dificultades en la implementación de cambios de escena, ya que esto requería eliminar a los personajes involucrados en un nivel para avanzar a otros. Este problema demandó un tiempo considerable de trabajo y el uso del depurador, ya que identificamos un error al realizar un "delete" de elementos de memoria dinámica, lo que podía llevar al colapso del programa. Solucionamos este problema creando funciones específicas para la configuración de cada escena, reiniciando los atributos de los elementos y diseñando escenas distintas para cada contexto de interacción.