Proyecto Final Planteamiento

Integrantes 1: Daniel Cruz

Integrantes 2: Miguel Cacabelos

Evento escogido: Ataque de la flota británica a Cartagena de Indias.

Profesor de teoría: Aníbal José Guerra.

**Resumen Histórico:**

El juego "Defensa de Cartagena de Indias" está inspirado en el histórico ataque de la flota británica a la ciudad de Cartagena de Indias en 1741. En este evento, el almirante español Blas de Lezo lideró la defensa de la ciudad contra las fuerzas navales británicas.

**Niveles del Juego:**

**Nivel 1: El Asedio Inicial**

En este nivel, el jugador asume el papel de Blas de Lezo, defendiendo la ciudad de Cartagena de los barcos británicos que se aproximan desde el norte. Los barcos enemigos descienden verticalmente hacia la zona inferior del juego, representando el avance de la flota enemiga hacia la ciudad. El jugador debe proteger las murallas de la ciudad utilizando un cañón que dispara verticalmente hacia arriba, emulando la estrategia defensiva utilizada por Blas de Lezo para repeler el primer asalto enemigo.

**Nivel 2: El Contraataque**

En este nivel, la batalla se intensifica, y los barcos británicos comienzan a disparar contra las defensas de la ciudad. Los barcos enemigos se mueven en zigzag descendente, representando el intento de los británicos de sortear las defensas españolas. El jugador debe proteger las murallas de la ciudad de los disparos enemigos mientras sigue utilizando el cañón para repeler los ataques y mantener a raya a los invasores.

**Nivel 3: Enfrentamiento con el Navío jefe**

En el nivel final, el jugador se enfrenta al navío insignia británico, representando el momento culminante del asedio. Este navío es controlado por el jefe enemigo más poderoso y representa la mayor amenaza para la ciudad. El jugador debe utilizar todas las habilidades adquiridas en los niveles anteriores para derrotar al navío jefe y asegurar la victoria para Cartagena de Indias.

**Explicación Mecánica de los Niveles:**

**Nivel 1: Defensa con Cañón**

En este nivel, el jugador controla un cañón que puede disparar proyectiles verticalmente hacia arriba. Debe apuntar con precisión y calcular la trayectoria de los proyectiles para impactar a los barcos enemigos que descienden hacia la ciudad.

**Nivel 2: Estrategia de Protección**

En este nivel, además de controlar el cañón, el jugador también debe manejar la defensa de las murallas de la ciudad contra los disparos enemigos. Se introduce la mecánica de movimientos en zigzag de los barcos enemigos, lo que requiere una mayor habilidad para anticipar y contrarrestar los ataques.

**Nivel 3: Batalla Final**

En este nivel, el jugador se enfrenta a un desafío final contra el navío jefe británico. Debe utilizar todas las habilidades adquiridas en los niveles anteriores para derrotar al enemigo más poderoso y asegurar la victoria para Cartagena de Indias.

**Entornos Físicas y Objetivos:**

**Nivel 1: El Asedio Inicial**

Vista: Vista superior en 2D, mostrando la ciudad de Cartagena y el mar circundante desde arriba.

Entorno: La ciudad de Cartagena se extiende a lo largo de la parte inferior de la pantalla, con las murallas defensivas claramente visibles. El mar ocupa la mayor parte de la pantalla, con barcos británicos descendiendo desde la parte superior hacia la ciudad.

Física de los Personajes: Blas de Lezo controla el cañón defensivo, con movimientos fluidos y precisos para apuntar y disparar proyectiles. Los proyectiles siguen una trayectoria balística realista hacia arriba (MRU).

Enemigos: Barcos británicos que descienden verticalmente desde la parte superior de la pantalla hacia la ciudad. Tienen diferentes tamaños y velocidades, lo que requiere al jugador ajustar su estrategia de defensa (MRU),

**Nivel 2: El Contraataque**

Vista: Vista superior en 2D, similar al nivel anterior, pero con un mayor énfasis en la interacción entre los proyectiles y los disparos enemigos.

Entorno: La ciudad de Cartagena sigue siendo visible en la parte inferior, pero ahora hay más actividad en la superficie del mar. Los barcos británicos se mueven en patrones de zigzag hacia abajo, representando un desafío adicional.

Física de los Personajes: Blas de Lezo sigue controlando el cañón defensivo, pero ahora debe esquivar los disparos enemigos mientras sigue disparando proyectiles hacia arriba.

Enemigos: Siguen habiendo barcos con físicas respectivas de (MRU) también cuentan con movimientos físicos que son (péndulo y movimiento armónico)

Objetivo: Proteger las murallas de la ciudad, manteniendo la ciudad a salvo del asedio.

**Nivel 3: Enfrentamiento con el Navío jefe**

Vista: Vista superior en 2D, similar a los niveles anteriores, pero con un enfoque más centrado en el enfrentamiento directo con el navío jefe.

Entorno: La ciudad de Cartagena sigue siendo visible en la parte inferior, pero ahora el navío jefe británico ocupa una posición prominente en la parte superior de la pantalla

Física de los Personajes: Blas de Lezo controla el cañón defensivo con movimientos aún más ágiles y precisos, necesarios para enfrentarse al poderoso navío jefe.

Enemigos: El navío jefe británico es el enemigo principal en este nivel, tiene armamento más poderoso que los barcos comunes. También puede desplegar cañones con movimientos físicos como la parábola o el péndulo, adicionales para complicar la batalla.

Objetivo: Soportar al navío jefe británico y asegurar la victoria para Cartagena de Indias, resistiendo su embate final y protegiendo las murallas de la ciudad contra su poderío naval.

**PLANTEAMIENTO POO**

**Clase Disparo**

La clase Disparo define el comportamiento de los disparos en el juego, incluyendo su velocidad y la física que los gobierna.

**Atributos:**

* float speed: Velocidad del disparo.
* float direction: Dirección del disparo.
* sprites \*pixmap\_management: Gestión de los sprites del disparo.
* QVector<enemigo \*> enemy: Vector de punteros a objetos de la clase enemigo.
* QTimer \*timer: Temporizador para controlar el movimiento del disparo.

**Métodos:**

* Disparo(QVector<enemigo \*> enemy, float scale, float speed, float direction): Constructor que inicializa un disparo con los enemigos especificados, escala, velocidad y dirección.
* ~Disparo(): Destructor que libera la memoria utilizada por los recursos del disparo.
* QRect set\_complete\_sprites(): Método para establecer los sprites completos del disparo.
* bool test\_collition(): Método para detectar colisiones entre el disparo y los enemigos.
* bool test\_range(): Método para comprobar si el disparo está fuera de rango.
* void move(): Método para mover el disparo y gestionar su colisión y rango.

**Clase enemigo**

La clase enemigo representa a los enemigos en el juego y define su comportamiento y apariencia.

**Atributos:**

* sprites \*pixmap\_management: Gestión de los sprites del enemigo.

**Métodos:**

* enemigo(int x, int y, int h, float scale, QString ruta): Constructor que inicializa un enemigo con la posición, altura, escala y ruta de la imagen especificadas.
* ~enemigo(): Destructor que libera la memoria utilizada por los recursos del enemigo.
* void muerte(): Método para cambiar la imagen del enemigo cuando muere.

**Clase escenario**

La clase escenario representa el entorno del juego, como la muralla de la ciudad, y gestiona su apariencia y comportamiento.

**Atributos:**

* sprites \*pixmap: Gestión de los sprites del escenario.

**Métodos:**

* escenario(float scale): Constructor que inicializa el escenario con la escala especificada.
* ~escenario(): Destructor que libera la memoria utilizada por los recursos del escenario.
* QRect set\_complete\_sprites(): Método para establecer los sprites completos del escenario.
* QRect set\_non\_destructible\_block(): Método para establecer los bloques no destructibles del escenario.

**Clase finalboss**

La clase finalboss representa al jefe final del juego y gestiona su apariencia y comportamiento.

**Atributos:**

* sprites \*pixmap: Gestión de los sprites del jefe final.

**Métodos:**

* finalboss(float scale): Constructor que inicializa al jefe final con la escala especificada.
* ~finalboss(): Destructor que libera la memoria utilizada por los recursos del jefe final.
* QRect set\_complete\_sprites(): Método para establecer los sprites completos del jefe final.
* QRect objeto(): Método para establecer el objeto del jefe final.

**Clase fisicas**

La clase fisicas maneja la física de los movimientos de los objetos en el juego, como el movimiento parabólico, el movimiento en zigzag, el movimiento armónico y el movimiento de péndulo.

**Atributos:**

* QGraphicsPixmapItem \*item: Objeto gráfico al que se aplica la física.
* QTimer \*p\_time, \*default\_movement, \*z\_time, \*harmonic\_time, \*pendulum\_time: Temporizadores para controlar los diferentes tipos de movimiento.
* float x, y, vx, vy, ax, ay, g, x0, y0, vx0, vy0, n: Variables para controlar la posición, velocidad y aceleración de los objetos, así como los parámetros para los movimientos específicos.
* int h, zigzag\_counter: Altura del objeto y contador para el movimiento en zigzag.
* float amplitude, frequency, pendulum\_length, pendulum\_angle: Parámetros para los movimientos armónico y de péndulo.

**Métodos:**

* fisicas(int x, int y, int h, QGraphicsPixmapItem \*item): Constructor que inicializa los parámetros de física del objeto.
* ~fisicas(): Destructor que libera la memoria utilizada por los recursos de física.
* Métodos para iniciar diferentes tipos de movimientos como start\_parabolic\_movement(), start\_zigzag\_movement(), start\_harmonic\_movement(), y start\_pendulum\_motion().
* Métodos para controlar el movimiento de los objetos como parabolic\_shoot(), MRU(), zigzag(), harmonic\_motion(), y pendulum\_motion().
* Métodos auxiliares como set\_pos\_item() y set\_starting\_parameters() para configurar la posición inicial y otros parámetros.

**Clase muerte**

La clase muerte gestiona la animación de la muerte de los enemigos en el juego.

**Atributos:**

* sprites \*pixmap\_management: Gestor de sprites para administrar la imagen de la muerte de los enemigos.
* QTimer \*animation\_timer: Temporizador para controlar la animación.
* int current\_frame: Índice del fotograma actual en la animación.
* const int total\_frames: Número total de fotogramas en la animación.
* const int animation\_duration: Duración total de la animación en milisegundos.

**Métodos:**

* muerte(float scale): Constructor que inicializa los parámetros de la animación de la muerte.
* ~muerte(): Destructor que libera la memoria utilizada por los recursos de la animación.
* void start\_animation(): Método para iniciar la animación de la muerte.
* void update\_animation\_frame(): Método para actualizar el fotograma de la animación.
* QRect set\_complete\_sprites(): Método para establecer las dimensiones de los sprites de la animación.
* void set\_animations(): Método para configurar diferentes animaciones (no implementado en este ejemplo).

**Señales:**

* void animation\_finished(): Señal emitida cuando la animación ha terminado.

**Clase personaje**

La clase personaje define el comportamiento y las características del personaje controlado por el jugador en el juego.

**Atributos:**

* sprites \*pixmap\_management: Gestor de sprites para administrar las imágenes del personaje.
* unsigned int keys[4]: Arreglo que almacena las teclas de control para el movimiento del personaje.

**Métodos:**

* personaje(unsigned int scale): Constructor que inicializa los parámetros del personaje y carga sus imágenes.
* void set\_keys(unsigned int \*keys): Método para establecer las teclas de control del personaje.
* void move(unsigned int key, bool is\_valid): Método para mover el personaje según la tecla presionada y la validez del movimiento.
* ~personaje(): Destructor que libera la memoria utilizada por los recursos del personaje.
* QRect set\_complete\_sprites(): Método para establecer las dimensiones de los sprites del personaje.
* void set\_animations(): Método para configurar las animaciones del personaje.
* void set\_down\_animation(): Método para configurar la animación de movimiento hacia abajo del personaje.

**Clase reglas\_juego**

La clase reglas\_juego define las reglas y la lógica del juego, incluyendo la configuración de escenas, la creación de personajes y enemigos, el manejo de eventos y la gestión de niveles.

**Atributos:**

* QTimer \*timer: Temporizador para controlar eventos en el juego.
* QTimer \*duracion\_nivel: Temporizador para controlar la duración del primer nivel.
* QTimer \*duracion\_nivel2: Temporizador para controlar la duración del segundo nivel.
* QGraphicsView \*graph: Vista gráfica para mostrar la escena del juego.
* int muralla\_vida: Vida de la muralla que protege al jugador.
* QVector<QLabel \*> labels: Vector para almacenar etiquetas en la interfaz del juego.
* QGraphicsScene \*scene: Escena donde ocurre la acción del juego.
* personaje \*blas: Objeto que representa al personaje controlado por el jugador.
* QVector<enemigo \*> enemys: Vector que contiene los enemigos del juego.
* escenario \*muralla: Objeto que representa la muralla que protege al jugador.
* finalboss \*barcofinal: Objeto que representa al jefe final del juego.
* muerte \*animacion\_muerte: Objeto que representa la animación de muerte de los enemigos.
* unsigned int blas\_keys[4]: Arreglo que almacena las teclas de control del personaje.
* float difficult: Dificultad del juego.

**Métodos:**

* reglas\_juego(QGraphicsView \*graph): Constructor que inicializa los parámetros del juego y carga el primer nivel.
* ~reglas\_juego(): Destructor que libera la memoria utilizada por los recursos del juego.
* void key\_event(QKeyEvent \*event): Método para manejar eventos de teclado en el juego.
* bool pared\_valida(bool izq): Método para verificar si el movimiento del personaje es válido en la pared.
* void load\_level\_1(): Método para cargar el primer nivel del juego.
* void clear\_scene(): Método para limpiar la escena actual del juego.
* void remove\_shoot(QGraphicsItem \*shoot): Método para eliminar los disparos de la escena.
* void enemigos\_Collisi(): Método para verificar las colisiones entre los enemigos y la muralla.
* void nivel2(): Método para cargar el segundo nivel del juego.
* void nivel3(): Método para cargar el tercer nivel del juego.
* void disparar(): Método para que el personaje realice disparos.
* void enemy\_is\_reached(QGraphicsItem \*item, int enemy): Método para manejar la colisión entre los disparos y los enemigos.
* void set\_blas\_keys(): Método para establecer las teclas de control del personaje.
* void generate\_map(): Método para generar la muralla del juego.
* void generate\_finalboss(): Método para generar al jefe final del juego.
* void setup\_scene(QString fondo): Método para configurar la escena del juego.
* void setup\_blas(): Método para configurar al personaje del juego.
* void setup\_enemigo(QString sprite,QString sprite2, float scale): Método para configurar los enemigos del juego.
* void start\_parabolic(): Método para iniciar el movimiento parabólico de los enemigos.
* void start\_zigzag(): Método para iniciar el movimiento en zigzag de los enemigos.
* void start\_harmonic(): Método para iniciar el movimiento armónico de los enemigos.
* void start\_pendulum(): Método para iniciar el movimiento pendular de los enemigos.
* void star\_music(): Método para reproducir la música de fondo del juego.
* void star\_musicboss(): Método para reproducir la música del jefe final del juego.

**Clase sprites**

La clase sprites se utiliza para gestionar los sprites de imágenes en un juego.

**Atributos:**

* QPixmap \*main\_pixmap: Pixmap principal que contiene los sprites.
* QPixmap \*character\_pixmap: Pixmap del personaje.
* QVector<QRect> animations: Vector que contiene los rectángulos de recorte para las animaciones.
* QVector<unsigned int> animations\_size: Vector que contiene el tamaño de cada animación.
* unsigned int width, height: Ancho y alto de los sprites.
* unsigned int animation\_counter: Contador de animación.
* float scale: Escala de los sprites.

**Métodos:**

* sprites(QString main\_pixmap, float scale): Constructor que inicializa la clase con la ruta del pixmap principal y la escala.
* void set\_design\_size(unsigned int x, unsigned int y): Método para establecer el tamaño de diseño de los sprites.
* void cut\_character\_pixmap(QRect size): Método para recortar el pixmap del personaje.
* void add\_new\_animation(QRect size, unsigned int number): Método para agregar una nueva animación con su tamaño.
* QPixmap get\_current\_pixmap(unsigned int animation): Método para obtener el pixmap actual de una animación.
* QPixmap get\_fixed\_image(QRect size): Método para obtener una imagen fija de los sprites.

**DIAGRAMA UML**

Diagrama

Descripción generada automáticamente