PROYECTO FINAL: PROMETEO

LUIS SNEYDER RODRIGUEZ MENJURA

INFORMÁTICA II

FACULTAD DE INGENIERÍA

MAYO DE 2023

MEDELLÍN

INTRODUCCIÓN

Actualmente los videojuegos se han convertido en una forma de entretenimiento muy popular en todo el mundo. Desde los juegos clásicos de árcade hasta los títulos más recientes, los videojuegos han evolucionado en términos de gráficos, interacción y narrativa, convirtiéndose en una forma de arte en sí mismos.

En este trabajo, se explorará el proceso de desarrollo del videojuego “Prometeo”, desde la conceptualización hasta la fase de producción y la presentación del producto final. Se abordarán los aspectos clave del diseño, narrativa, modelos físicos implementados, estética visual y especificaciones en los objetos que intervienen en su ejecución.

El juego se desarrolla utilizando el paradigma de programación orientada a objetos, con el lenguaje de programación C++ en la interfaz gráfica de usuario de Qt Creador.

CONCEPTO

Narrativa:

En este videojuego de ciencia ficción basada en la saga "Alíen", el jugador asume el papel de un explorador que ha sido enviado en una misión espacial a Pólux, una de las lunas del planeta Zeta. La misión es encontrar y estudiar a los Ingenieros, una raza alienígena misteriosa y poderosa.

Sin embargo, una vez que el jugador llega a Pólux, descubre que la luna ha sido invadida por los letales Xenomorfos quienes han exterminado a los Ingenieros y se han adueñado del lugar. Con su nave destruida, el jugador debe luchar por sobrevivir mientras intenta conseguir las celdas de energía necesarias para escapar a bordo de una de las naves de los Ingenieros.

Desarrollo:

El juego tendrá dos fases, En la primera, el jugador está a bordo de una nave espacial que se dirige hacia Pólux, y tendrá que esquivar peligrosos asteroides, cometas y agujeros negros que amenazan su viaje.

Una vez que el jugador llega a la luna, comienza la segunda fase del juego: obtener las celdas de energía necesarias para despegar la nave de los Ingenieros. Sin embargo, no será una tarea fácil, ya que estas celdas están ubicadas en zonas con una fuerza gravitacional intensa y es necesario utilizar una capsula especial que solo cuenta con movimiento parabólico, lo que limita el movimiento del jugador

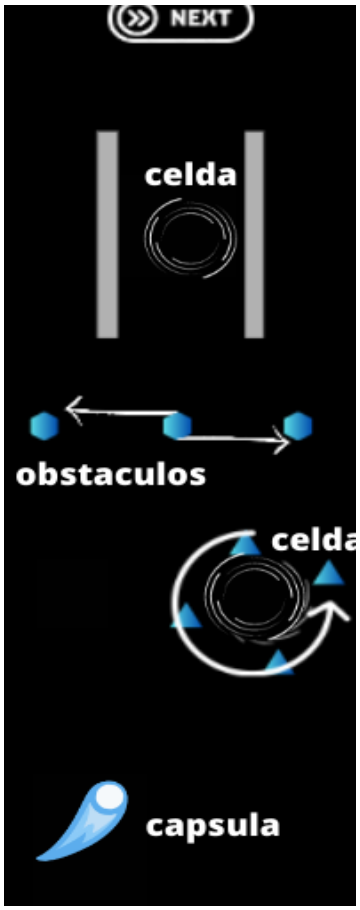
Además, las celdas están protegidas por estructuras altamente avanzadas que el jugador debe evitar ya que lo desintegrarían inmediatamente, las estructuras tienen varios movimientos y patrones de defensa para evitar que el jugador pueda acceder a las celdas.

El jugador gana si logra llegar a Pólux y consigue todas las celdas de energía necesarias para escapar del planeta.

VISUALES

Fase 1:

Fase 2:



MODELOS FÍSICOS

Movimientos del jugador:

1. Movimiento Rectilíneo Uniforme:

   En el primer momento en que el jugador viaja en la nave rumbo a la luna de los Ingenieros, a medida que avanza, tendrá la posibilidad de moverse hacia arriba y abajo para esquivar los asteroides que se cruzan en su camino.

   Posición final = Posición Inicial + Velocidad x Tiempo.

2. Movimiento Parabólico

   Cuando el jugador se encuentra dentro de la cápsula y busca las celdas de energía, su único movimiento posible es el movimiento parabólico. Además, es necesario que esté en constante movimiento, ya que, de lo contrario, la fuerza gravitacional del área lo llevará hacia el fondo donde están los Xenomorfos.

- Movimiento rectilíneo uniforme para el eje x

$$x = x_0 + v_x \cdot t$$

X es la distancia horizontal recorrida por el objeto

Vx es la velocidad horizontal del objeto

T es el tiempo transcurrido desde el lanzamiento del objeto

- Movimiento rectilíneo uniformemente acelerado para el eje y

$$y = y_0 + v_{0y} \cdot t + \frac{1}{2} \cdot a_y \cdot t^2$$

Y es la altura del objeto respecto al punto de lanzamiento

Yo es la altura inicial del objeto en el momento del lanzamiento

V es la velocidad inicial del objeto

t es el tiempo transcurrido desde el lanzamiento del objeto

g es la aceleración debida a la gravedad (9.81 m/s^2)

Movimientos de los obstáculos:

Las estructuras que protegen las celdas de energía para lograr encender la nave tendrán movimientos.

1. Movimiento Circular Uniforme

   $\theta = \Omega * t$

   Θ es el ángulo de rotación del objeto en el tiempo t

   Ω es la velocidad angular del objeto

   t es el tiempo transcurrido

2. Movimiento Rectilíneo Uniforme:

   Posición final = Posición Inicial + Velocidad x Tiempo.

CONTROLES

Las teclas para interactuar con el juego serán las flechas del teclado:

- Arriba y abajo para controlar la nave en el espacio.
- Izquierda y derecha para controlar la capsula con movimiento parabólico en las zonas con gravedad intensa.

REQUERIMIENTOS

1. Contexto e Instrucciones

Al inicio del juego se presenta un breve resumen del juego y se dan las instrucciones para jugar. Esto se hará por mediante una función que lee un archivo texto y mostrara las instrucciones en pantalla

2. Escenas

Las escenas del videojuego serán dinámicas y conforme avance el jugador, el programa debe actualizar la escena que se le presenta al usuario.

Primera escena:

- Es necesario que la escena que representa el espacio en la primera fase del videojuego tenga por lo menos 5000 pixeles de ancho para que con la función foco se pueda ir mostrando cierta parte del mapa.

- Para colocar cada asteroide y cometa en el espacio, se tendrán las coordenadas y el tipo de obstáculo (x, y, n) en un archivo de texto, se implementara una función para que el juego lea las coordenadas y posiciones cada objeto en su posición.

- Si la nave choca un obstáculo, es necesaria una función que reinicie el juego tantas veces como sea necesario.

- Al finalizar la primera fase, se envía una señal para dar inicio a la segunda fase que iniciara con la llegada de la nave a Pólux, la luna destruida por los Xenomorfos y se dan nuevas instrucciones al jugador.

Segunda Escena:

- En la segunda fase el juego tendrá una orientación vertical y las escenas tendrán 3000 pixeles de alto.

- Para la puesta en escena de los obstáculos dinámicos, el programa recibe de un archivo de texto la coordenadas (x, y) de cada estructura y el tipo de movimiento (mcu o mru).

- En esta fase el jugador tiene tres oportunidades de obtener las celdas energía y es necesario una función que cuente los intentos del jugador.

- Si el jugador obtiene todas las celdas se presenta mediante video el escape del planeta en la nave de los ingenieros

- Se utilizara el algoritmo CollidesWithItem()

Funciones a implementar

- Evaluar Colisión
- Focus
- Reiniciar Fase
- Reiniciar Juego
- Movimiento(Conectado a su respectivo Timer)
- Movimiento Enemigo (Conectado a su respectivo Timer)
- Lectura de instrucciones
- Lectura de coordenadas
- Ganar
- Perder
- Destruir Elemento

Objetos necesarios

- Ventana Principal
- Elemento (Clase padre)
- Nave
- Capsula
- Asteroide
- Proyectil
- Xenomorfos
- Obstáculo dinámico
- Celda de energía
- Nave Ingeniero

MODELADO DE OBJETOS

Para lograr la implementación del videojuego de manera efectiva, es necesario utilizar objetos debido a la complejidad del proyecto. Al adoptar el paradigma de programación orientada a objetos, se logrará abordar este problema de manera más eficiente.

A continuación, se presenta una aproximación a las clases que se implementaran.

1. Clase Padre Elemento.

Esta clase hereda de forma pública las clases **QObject** y **QGraphicsItem** y es la clase base para crear todos los demás objetos que se agregaran a las escenas.

Atributos:

Enteros: Posición en eje X, Posición en el eje Y.

Flotantes: Ancho, Alto, Columna, Fila. (Para el Sprite)

QTimer (Para iterar sobre el Sprite)

QPixmap (Para el Sprite)

Métodos:

Void Destruir (Útil cuando el elemento colisione y deba ser eliminado de la escena)

QRectF boundingRect ();

Void Paint ();

Slot:

Void Actualizar Sprite (Itera sobre el Sprite según la señal del QTimer)

2. Clase Nave Espacial (Hereda los atributos y métodos de la clase Elemento**)**

Atributos:

Enteros: Velocidad

Métodos

Void MoveUp();

Void MoveDown();

Void Disparar ();

3. Clase Capsula (Hereda los atributos y métodos de la clase Elemento**)**

Atributos:

Enteros: Velocidad, Numero de celdas obtenidas.

QTimer (Para el movimiento Parabólico)

Métodos

Void Movimiento Parabólico ();

4. Clase Proyectil (Hereda los atributos y métodos de la clase Elemento**)**

Atributos:

Enteros: Velocidad.

Métodos

Void Movimiento Rectilíneo Uniforme ();

5. Clase Obstáculo Dinámico (Hereda los atributos y métodos de la clase Elemento)

Constructor: (Posición X, Posición Y, Tipo de Obstáculo)

Atributos:

Enteros: Velocidad

Float: Angulo Rotación

Métodos

Void Movimiento Rectilíneo Uniforme (Velocidad);

Void Movimiento Circula Uniforme (Angulo Rotación, Velocidad);

6. Clase Obstáculo Estático (Hereda de los atributos y métodos de la clase Elemento)

Esta clase reúne a todos los obstáculos como rocas en el espacio y estructuras que no tienen movimiento y al ser un elemento estático, únicamente se utilizara para evaluar las colisiones con el objeto Nave y el objeto Capsula.

Entero: Tipo de Obstáculo

7. Clase Xenomorfos (Hereda los atributos y métodos de la clase Elemento)

Atributos:

Enteros: Velocidad

Métodos

Void Movimiento Rectilíneo Uniforme ();

Void Buscar Humano ();

8. Clase Celda Energía

Hereda de los atributos y métodos de la clase Elemento, no cuenta con métodos ni atributos propios ya que están estáticos esperando a ser obtenidos.

8. Clase Ventana Principal

Esta clase se crea por defecto en Qt con el proyecto y hereda de forma pública la clase **QMainWindow**, incluirá a todas las clases anteriores ya que es donde se integran todos los objetos que componen el videojuego.

Atributos:

QGraphicsScene escena 1, escena 2, escena 3

QTimers (Varios para la animación de todos los elementos en escena)

QList < Obstáculo Estático >

QList < Obstáculo Dinámico >

QList < Xenomorfo>

Slots:

Void Movimiento () (Conectado a su respectivo Timer)

Void Movimiento Enemigo (); (Conectado a su respectivo Timer)

Void Ganar ();

Void Perder ();

Signals:

Void aviso (Numero de aviso)

Métodos:

Void KeyPressEvent(QKeyEvent *evento);

Bool Evaluar Colisión ();

Void Focus ();

Void Reiniciar Fase ();

Void Reiniciar Juego ();