Alpha Creative

Mikael Rodríguez López

1ºDAM

PROYECTO PROGRAMACIÓN 1ºDAM

INDICE

1. INTRODUCCIÓN

Soy Mikael Rodriguez y este documento contiene la documentación de mi **proyecto final de programación** para 1º de Desarrollo de Aplicaciones Multiplataforma (DAM). El proyecto consiste en el desarrollo de un **motor gráfico sencillo en 2D** utilizando Java, junto con un juego inspirado en "Flappy Bird" que utiliza este motor.

El desarrollo de este motor gráfico ha sido un desafío significativo que ha requerido mucho esfuerzo y dedicación. El motor permite la **gestión y renderización de gráficos en dos dimensiones**, proporcionando una base sólida para la creación de una amplia variedad de juegos y aplicaciones gráficas.

El juego incluido, que reproduce la **mecánica del popular "Flappy Bird"**, es solo un ejemplo de las **muchas posibilidades que ofrece el motor gráfico**. Este juego demuestra las capacidades del motor para manejar gráficos, detectar colisiones y actualizar la lógica del juego en tiempo real. Es una muestra del potencial del motor y de las numerosas aplicaciones que se pueden desarrollar a partir de él.

En esta documentación, se describen los detalles técnicos del motor gráfico y las características del juego. Además, se proporcionan instrucciones para la ejecución y prueba del proyecto, destacando los desafíos superados y las soluciones implementadas.

Mi objetivo con este proyecto es demostrar mis habilidades y dedicación, así como crear una herramienta útil que pueda ser ampliada y mejorada en el futuro, mostrando el potencial y la versatilidad del motor gráfico desarrollado.

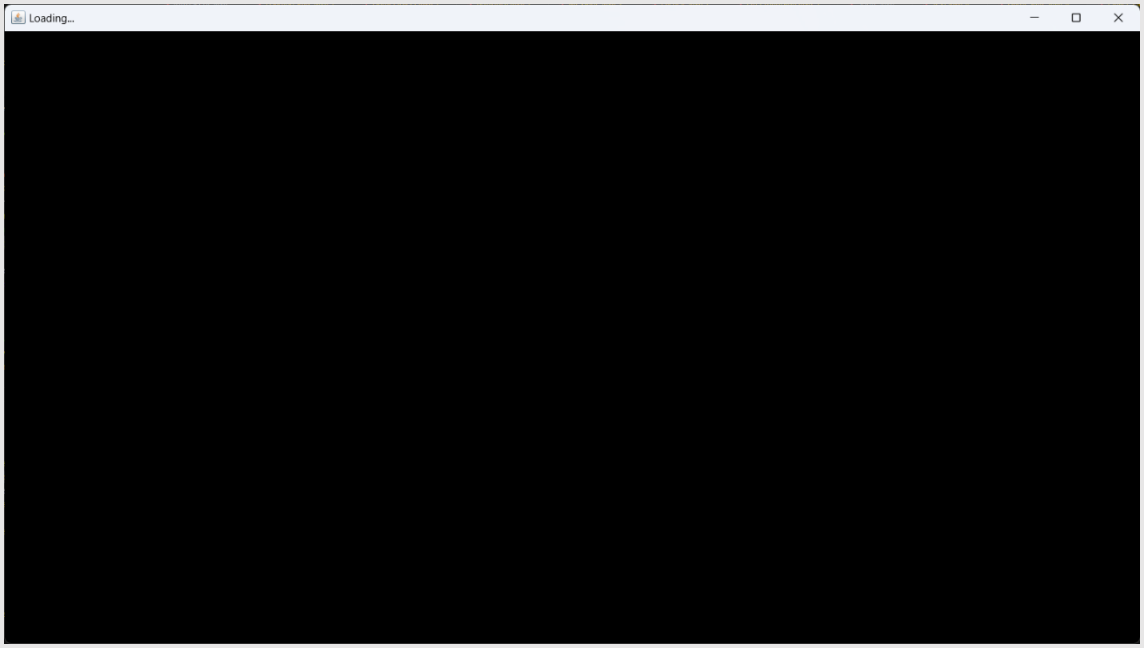
EJECUCIÓN DEL PROYECTO

Información previa:

- Asegúrate de tener instalado el **JDK 21** de Java.
- Recomendado usar **VSCode o IntellijIdea** como entorno de desarrollo.
- La ejecución main se encuentra en **org.alphacreative.main.Window**
- Incluyo un **.jar ejecutable** para probar el juego.

1.1. MOTOR GRÁFICO 2D

Clase Window(Ventana)



La clase ventana es la clase principal del proyecto, donde se ejecuta nuestro primer y único "main". es la responsable de crear y gestionar la ventana del juego, así como de manejar el ciclo principal del mismo. A continuación, se ofrezco una descripción resumida de sus principales funcionalidades y métodos:

Canvas: Se utiliza para la representación gráfica.

KeyBoard y MouseInput: Se añaden como oyentes de eventos de teclado y ratón para recibir y gestionar entradas del usuario.

Jerarquía de Escenas: El juego maneja diferentes escenas (GameState), como la introducción, el menú y el juego principal, cambiando entre ellas según sea necesario. La escena 0 Siempre será la escena de Carga (*Imagen).

MÉTODOS PRINCIPALES

Update(): Actualiza el estado actual del juego, incluyendo entradas del usuario y lógica del juego.

Draw(): Renderiza el contenido gráfico en la ventana, limpiando la pantalla y dibujando la escena actual.

init(): Inicializa los recursos del juego, como los assets gráficos y los datos del juego.

run(): Ejecuta el ciclo principal del juego, gestionando el tiempo y asegurando que el juego se actualice y renderice correctamente.

start() y stop(): Inician y detienen el hilo del juego respectivamente.

ChangeScene(int scene): Cambia la escena actual del juego, permitiendo reiniciar y gestionar diferentes partes del juego de forma eficiente. Empezamos ejecutando desde la escena 1, la escena número 0 siempre será la escena de carga

StartLoadingScene(Thread loadingThread): Inicia la escena de carga para gestionar la transición entre el inicio(Escena de carga) y la carga completa de los recursos.

OBJETOS DE JUEGO

- GameObject (Clase de Objetos de Juego)

  La clase GameObject es una clase abstracta que sirve como base para la creación y manipulación de objetos de juego en el entorno 2D desarrollado en Java. Proporciona funcionalidades para el dibujado avanzado utilizando la biblioteca Graphics2D, así como métodos para facilitar rotaciones, escalas y traslaciones de objetos.

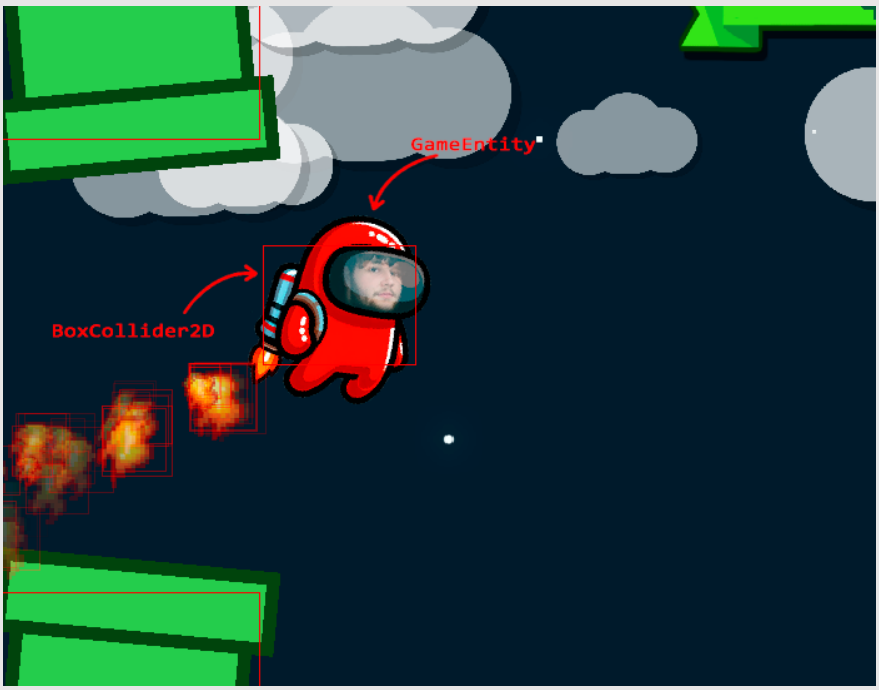
  Draw(Graphics g): Dibuja el objeto en el lienzo especificado utilizando el contexto gráfico Graphics.

  Update(): Método abstracto que debe ser implementado por las clases derivadas para actualizar el estado del objeto en cada fotograma.

-GameEntity (Clase de Entidades de Juego):
La clase GameEntity representa entidades de juego con capacidades de colisión en el entorno 2D desarrollado en Java. Extiende la clase GameObject y agrega funcionalidades para gestionar colisiones utilizando objetos BoxCollider2D.

Métodos Especiales

- isCollidingWithEntity2D(GameEntity other): Verifica si esta entidad de juego está colisionando con otra entidad de juego.
- isCollidingFromAbove2D(GameEntity other, double margin): Verifica si esta entidad de juego está colisionando desde arriba con otra entidad de juego con un margen dado.
- isCollidingFromBelow2D(GameEntity other, double margin): Verifica si esta entidad de juego está colisionando desde abajo con otra entidad de juego con un margen dado.
- isCollidingFromLeft2D(GameEntity other, double margin): Verifica si esta entidad de juego está colisionando desde la izquierda con otra entidad de juego con un margen dado.
- isCollidingFromRight2D(GameEntity other, double margin): Verifica si esta entidad de juego está colisionando desde la derecha con otra entidad de juego con un margen dado.



-Particle (Clase de Particulas de Juego):

La clase Particle representa una partícula en el juego, heredando de GameEntity. Las partículas pueden moverse en el espacio 2D, cambiar su opacidad con el tiempo y, opcionalmente, tener animaciones. La clase también permite personalizar la velocidad de destrucción de las partículas.

Atributos

- float xMove, yMove: Movimientos en los ejes X e Y.
- float startOpacity: Opacidad inicial de la partícula.
- BufferedImage[] textureAnim: Arreglo de texturas para la animación de la partícula.
- int index: Índice actual de la animación.
- float anim_timer: Temporizador para controlar la animación.
- float destroyVel: Velocidad a la que la partícula se desvanece.

Notas

- Movimiento: Las partículas pueden moverse en los ejes X e Y con velocidades específicas.
- Opacidad: Las partículas se desvanecen con el tiempo a una velocidad configurable.
- Animación: Soporte para animaciones de texturas, con control de tiempo para cambiar las imágenes.
- Destrucción: Las partículas se destruyen automáticamente cuando su opacidad es muy baja.

GRÁFICOS GUI (INTERFAZ)

-Sprite (Clase para manejar imágenes en UI):

La clase Sprite representa un elemento de interfaz de usuario (UI) que se visualiza como una imagen estática. Esta clase extiende la clase GameEntity y proporciona funcionalidades específicas para la interacción con el mouse.

Atributos

- targetScale: Escala deseada del sprite.
- realPosition: Posición real del sprite en relación con el centro de la pantalla.

Métodos

- isMouseOn(): Verifica si el mouse está sobre el sprite.
- isClickingOn(): Verifica si se está haciendo clic en el sprite.
- getRealPosition(): Obtiene la posición real del sprite.
- setRealPosition(Vector2D pos): Establece la posición real del sprite.

-ButtonUI (Clase para manejar botones):

La clase ButtonUI representa un botón en la interfaz de usuario (UI). Extiende la clase Sprite y añade funcionalidades adicionales para manejar interacciones visuales cuando el botón es clicado o el mouse pasa sobre él.

Atributos:

- textureArray: Un array de imágenes (BufferedImage) que contiene las texturas del botón para diferentes estados (normal y al hacer clic).
- increaseFactor: Un factor de aumento utilizado para escalar el botón cuando el ratón está sobre él.

-TextUI (Clase para manejar textos en pantalla con fuentes, escalas, bordes)

La clase TextUI representa un texto en la interfaz de usuario (UI). Extiende la clase GameEntity y añade funcionalidades específicas para la renderización de texto en pantalla, con opciones de personalización como el color, la fuente, el centrado y el grosor del contorno.

Atributos

- text: Texto que se muestra.
- color: Color del texto.
- center: Indica si el texto debe centrarse en su posición.
- font: Fuente utilizada para renderizar el texto.
- fontscale: Escala de la fuente.
- outlineGrosor: Grosor del contorno del texto.
- outlineColor: Color del contorno del texto.

Métodos

- SetText(Object text): Establece el texto a mostrar, convirtiendo el objeto a una cadena.

- GetText(): Obtiene el texto actual.

COORDENADAS

-Vector2D (Clase para manejar coordenadas bidimensionales)

La clase Vector2D representa un vector bidimensional con componentes x e y. Proporciona métodos para realizar operaciones comunes con vectores, como obtener la magnitud, establecer la dirección, calcular la distancia entre vectores y normalizar el vector.

getX(), getY(): Devuelve las componentes x e y del vector.

setX(double x), setY(double y): Métodos para establecer las componentes x e y del vector.

increaseX(double x_increment), decreaseX(double x_decrement): Métodos para incrementar o decrementar la componente x del vector.

increaseY(double y_increment), decreaseY(double y_decrement): Métodos para incrementar o decrementar la componente y del vector.

-BoxCollider2D (Clase para manejar colisiones bidimensionales)

La clase BoxCollider2D representa un colisionador rectangular en un espacio bidimensional (2D). Se utiliza para detectar colisiones entre objetos en un juego

UTILIDADES

-Mathf (Clase de funciones matemáticas):

La clase Mathf proporciona funciones matemáticas comunes similares a las que se encuentran en Unity. Estas funciones son útiles para cálculos de interpolación, limitación de valores y generación de números aleatorios dentro de rangos específicos.

Utilidades:

- **Interpolación (Lerp):** Útil para suavizar transiciones y animaciones.
- **Limitación (Clamp):** Nos asegura que los valores se mantengan dentro de un rango específico.
- **Generación Aleatoria (RandomRange y RandomBool):** Facilita la creación de booleanas aleatorias y números.
- **Normalización (NormalizeValue):** Permite mapear valores de un rango a otro, lo cual es útil en muchas aplicaciones de gráficos y juegos.

GESTIÓN DE DATOS

-DataManager (Clase de funciones estáticas de cargado y guardado)

La clase DataManager es un controlador para gestionar datos de guardado utilizando archivos de propiedades. Permite leer y escribir datos persistentes en un archivo, facilitando la gestión de configuraciones y estadísticas del juego.

Atributos

- **Properties properties:** Objeto Properties para almacenar y manejar pares clave-valor.
- **File file:** Archivo donde se guardan los datos.

Notas

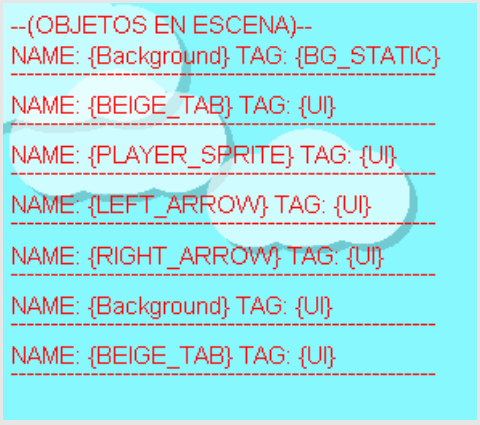
- **Gestión de Archivos:** Asegura que el archivo de propiedades exista antes de intentar cargar datos.
- **Valores por Defecto:** Permite la inicialización con valores por defecto si el archivo no existe.
- **Versatilidad:** Facilita la gestión de diferentes tipos de datos a través de métodos genéricos para guardar y obtener valores.

ESCENA

- GameState (Clase abstracta de escena de juego):
  La clase GameState es una clase abstracta que sirve como plantilla para definir diferentes escenas de juego dentro del motor gráfico. Define métodos abstractos para actualizar y dibujar la escena, así como métodos concretos para gestionar la jerarquía de objetos en la escena y dibujarla en modo de depuración.

- SceneObjects (Clase que maneja todos los objetos en escena)
  La clase SceneObjects es una colección de objetos presentes en una escena de juego. Permite agregar, eliminar, actualizar y dibujar objetos en la escena, así como buscar objetos por nombre o etiqueta.

- AddObject(GameObject object): Agrega un objeto a la escena.
- FindObjectWithName(String name): Busca un objeto por su nombre.
- RemoveObject(GameObject object): Elimina un objeto de la escena.
- FindObjectsWithTag(String tag): Devuelve una lista de objetos con una etiqueta específica.
- FindObjectsWithNoTag(String tag): Devuelve una lista de objetos sin una etiqueta específica.



Jerarquía de objetos en escena visible en modo Debug.

GESTOR DE JUEGO

-GameManager (Clase contenedora y gestora de componentes globales)

La clase GameManager centraliza la gestión y el control de varios aspectos del juego, como la ventana, la escena actual, los objetos en la escena y configuraciones de depuración. Proporciona métodos estáticos para acceder y manipular estos componentes de manera global.

Atributos

- **Window window:** Referencia a la ventana principal del juego.
- **GameState currentScene:** La escena actual que se está ejecutando.
- **SceneObjects currentSceneObjects:** Los objetos presentes en la escena actual.
- **double worldX, worldY:** Coordenadas globales del mundo del juego.
- **boolean isDebug:** Indicador de modo depuración.
- **boolean showColliders:** Indicador para mostrar colisionadores.

Métodos

Inicialización de la Ventana

setWindow(Window window) Establece la referencia a la ventana principal del juego.

Acceso a la Ventana

getWindow() Devuelve la referencia a la ventana principal del juego.

Control del Juego

Pause() Pausa el juego deteniendo los hilos de actualización y dibujo.

Resume() Reanuda el juego reiniciando los hilos de actualización y dibujo.

Tamaño de la Pantalla

getScreenSize() Devuelve las dimensiones de la ventana como un Vector2D.

Cambio de Escena

ChangeScene(int scene) Cambia a una nueva escena del juego identificada por un índice.

Título de la Ventana

setTitle(String title) Establece el título de la ventana del juego.

Datos del Juego

DataManager() Devuelve el gestor de datos para acceder y manipular datos guardados.

Gestión de la Escena

setCurrentScene(GameState scene) Establece la escena actual.

getCurrentScene() Devuelve la escena actual.

setCurrentSceneObjects(SceneObjects sceneObjects) Establece los objetos actuales en la escena.

SceneObjects() Devuelve los objetos actuales en la escena.

Centro de la Pantalla

getScreenCenter() Devuelve el centro de la pantalla como un Vector2D.

Coordenadas del Mundo

WorldX() Devuelve la coordenada X del mundo del juego.

WorldY() Devuelve la coordenada Y del mundo del juego.

Modo Depuración

IsDebug() Devuelve true si el modo depuración está activado.

SetDebug(boolean debug) Activa o desactiva el modo depuración.

Mostrar Colisionadores

ShowColliders() Devuelve true si los colisionadores están siendo mostrados.

SetShowColliders(boolean showColliders) Activa o desactiva la visualización de los colisionadores.

GESTOR DE RECURSOS

-Assets (Clase encargada de cargar los recursos utilizados en juego)

La clase Assets se encarga de gestionar la carga de todos los recursos multimedia necesarios para el juego, como imágenes, fuentes y sonidos. Estos recursos se utilizan en distintas partes del juego, incluyendo animaciones, elementos de la interfaz de usuario y efectos sonoros.

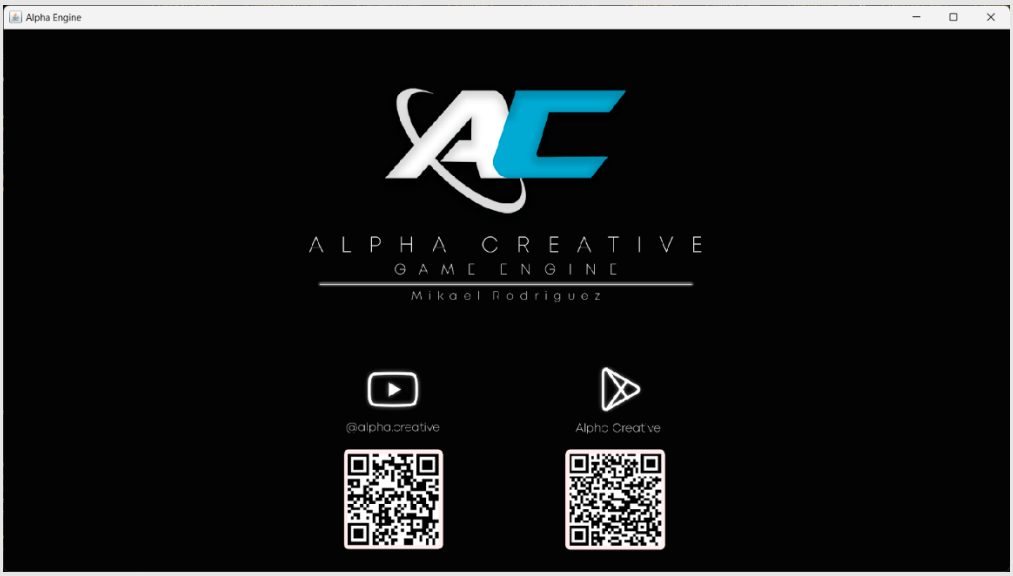
- Cargar Imágenes:
  La clase define numerosos atributos estáticos que representan imágenes y animaciones utilizadas en el juego.
  Utiliza el método LoadImage para cargar imágenes desde rutas específicas y actualizar el contador de recursos cargados (count).
- Cargar Fuentes:
  Define atributos estáticos para diferentes tipos de fuentes utilizadas en la interfaz del juego.
  Usa el método LoadFont para cargar las fuentes desde rutas específicas y actualiza el contador de recursos cargados.
- Cargar Sonidos:
  Incluye atributos para clips de sonido que se utilizan en el juego.
  Los sonidos se cargan y se gestionan de manera similar a las imágenes y fuentes.
- Inicialización de Recursos:
  El método init se encarga de cargar todos los recursos necesarios para el juego al inicio. Los cuales cargaremos de forma asincrónica en otro hilo de la clase Window(). Para así evitar problemas de rendimiento en el caso de usar un gran número de recursos
  Este método carga imágenes para animaciones, texturas, fondos, elementos de la interfaz de usuario, botones del menú y fuentes.

-Loader (Clase contenedora de métodos de carga)

La clase Loader proporciona métodos estáticos para cargar diferentes tipos de recursos multimedia, incluyendo imágenes, fuentes y sonidos. Estos métodos son ampliamente utilizados en la clase Assets para cargar los recursos necesarios para el juego.

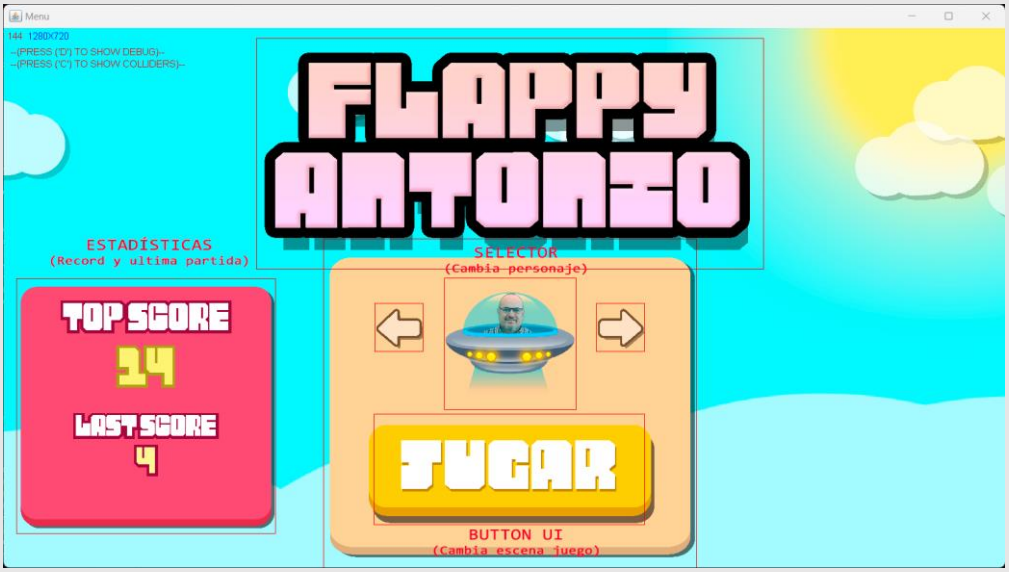
1.2. JUEGO FLAPPY ANTONIO

1.2.1. PANTALLA INICIAL



La pantalla inicial contiene una presentación animada de mi nombre y el nombre de mi marca personal usada tanto en la Play Store como en YouTube.

1.2.2. MENU

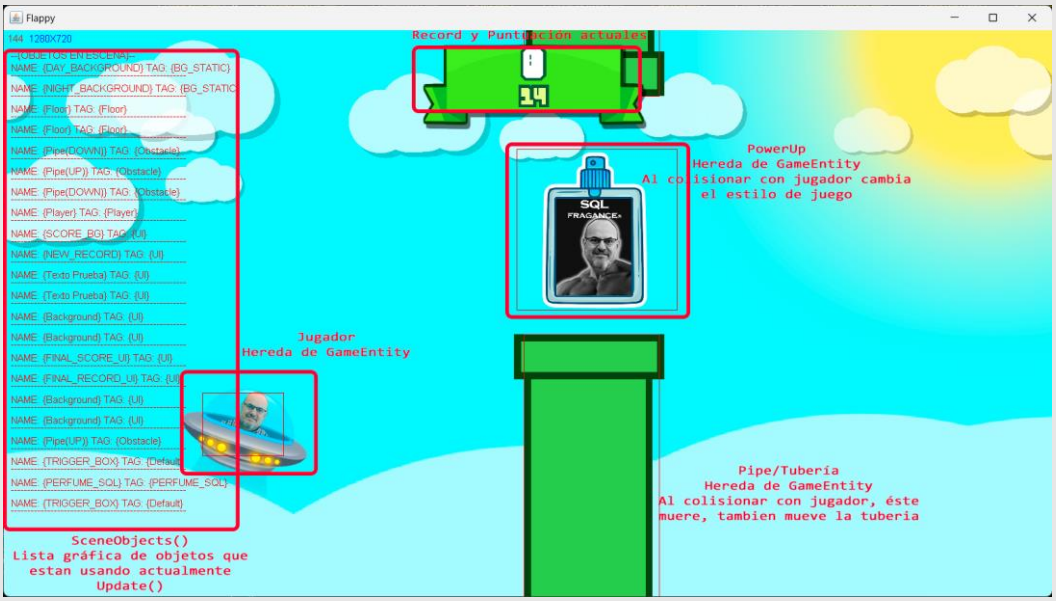


Después de la pantalla de presentación, tenemos un menú con interfaz donde podemos ver las estadísticas de partida, escoger personaje y por supuesto, jugar.

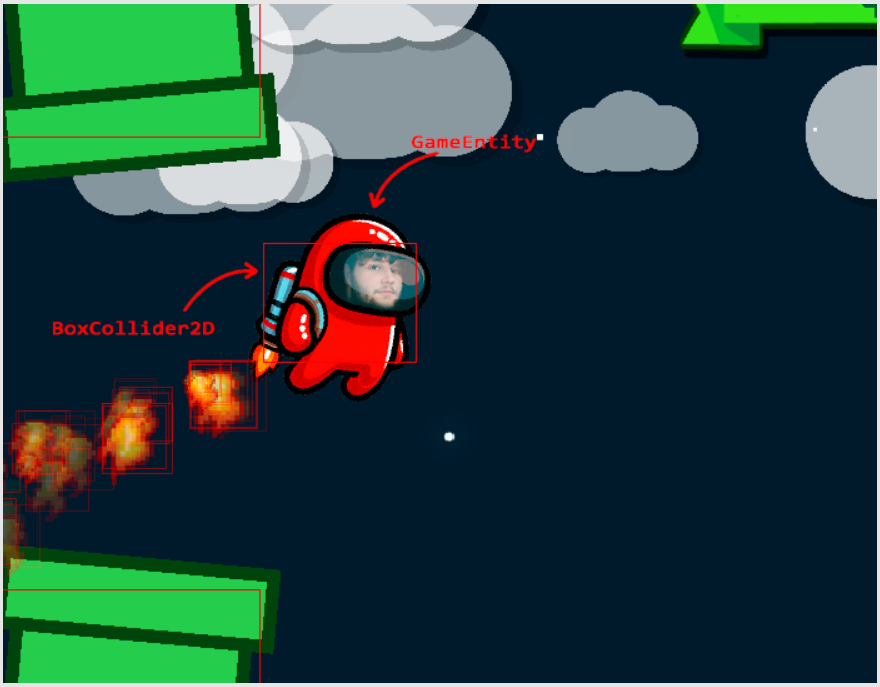
Podemos pulsar D (Para ver los objetos en escena)

Podemos pulsar C (Para ver los límites de los BoxCollider2D)

1.2.3. JUEGO



Dentro de partida, el juego funciona con un jugador que dispone de sus propias físicas programadas como herencia de GameEntity, al igual que la mayoría de objetos en escena, como las tuberías, los PowerUps etc...



Si escogemos este personaje, podemos comprobar la generación de partículas incluida en el motor

Cuando chocamos con la "Fragancia SQL", el juego da un giro interesante. El cielo cambia a una noche estrellada, las tuberías comienzan a moverse de forma diferente, rotarse, cerrarse y el juego se vuelve más rápido y complicado. Además, si superamos nuestro mejor puntaje, veremos una pancarta que dice "new record".

2.0. DIAGRAMA UML

Diagrama (Gestión de escenas y ventana)

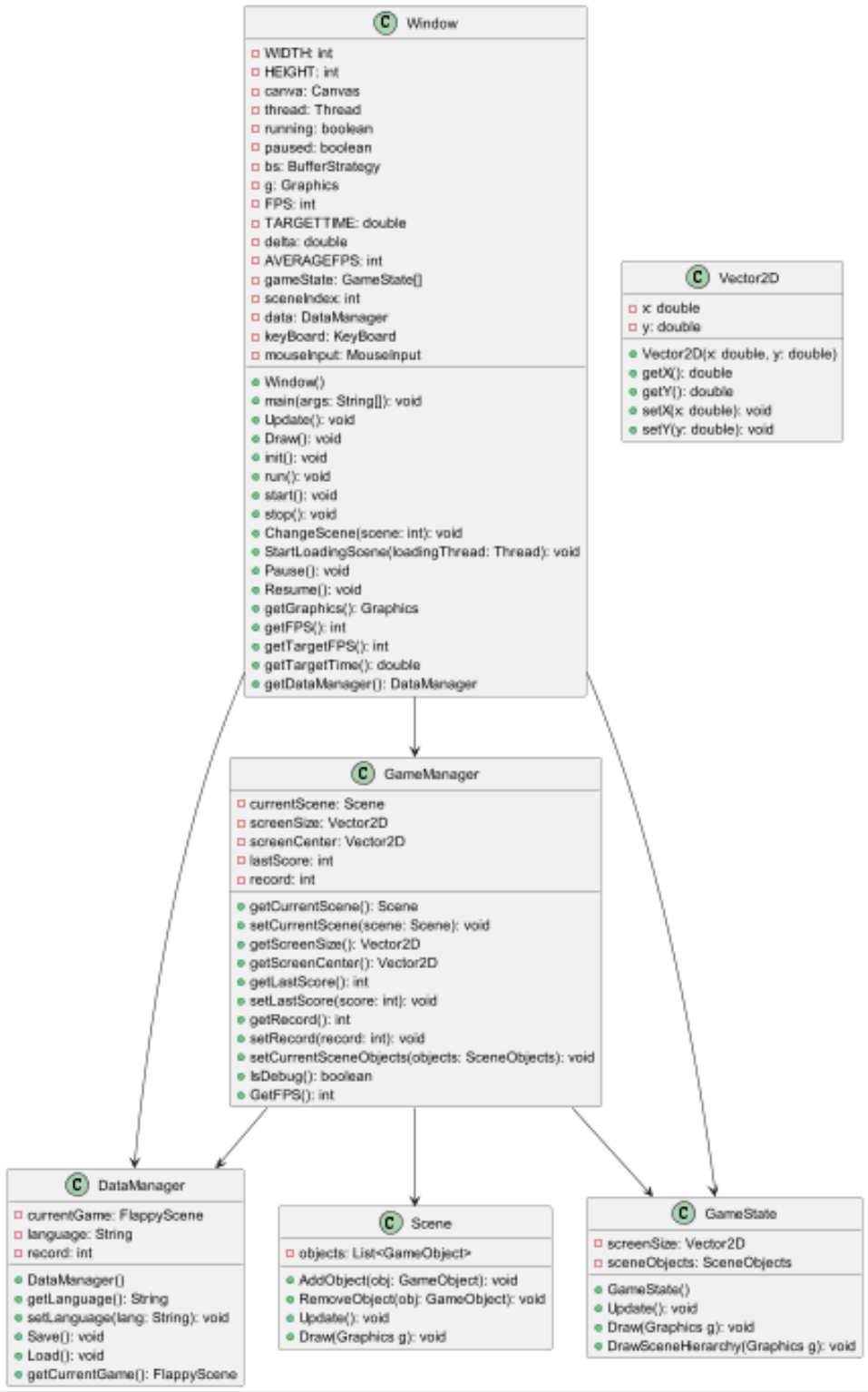


Diagrama (GameObjects)

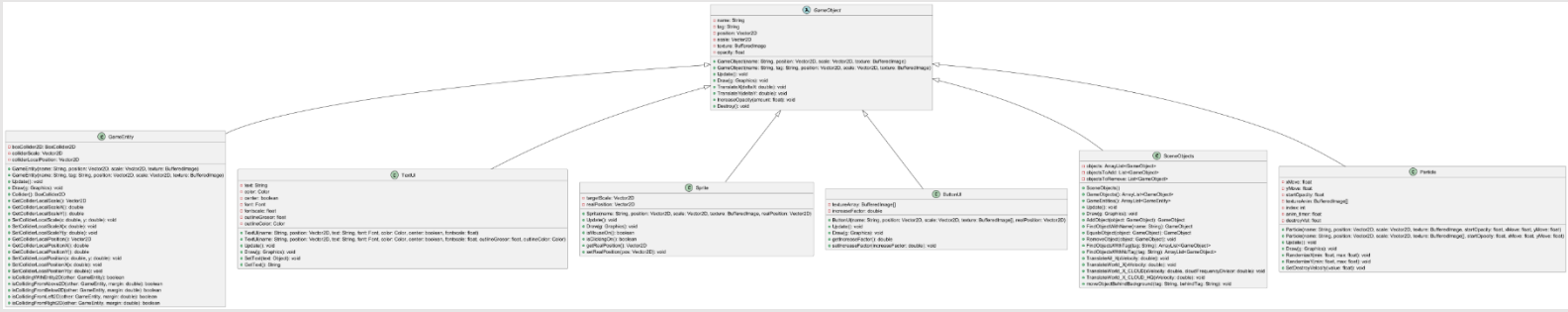


Diagrama (Menu de Juego)

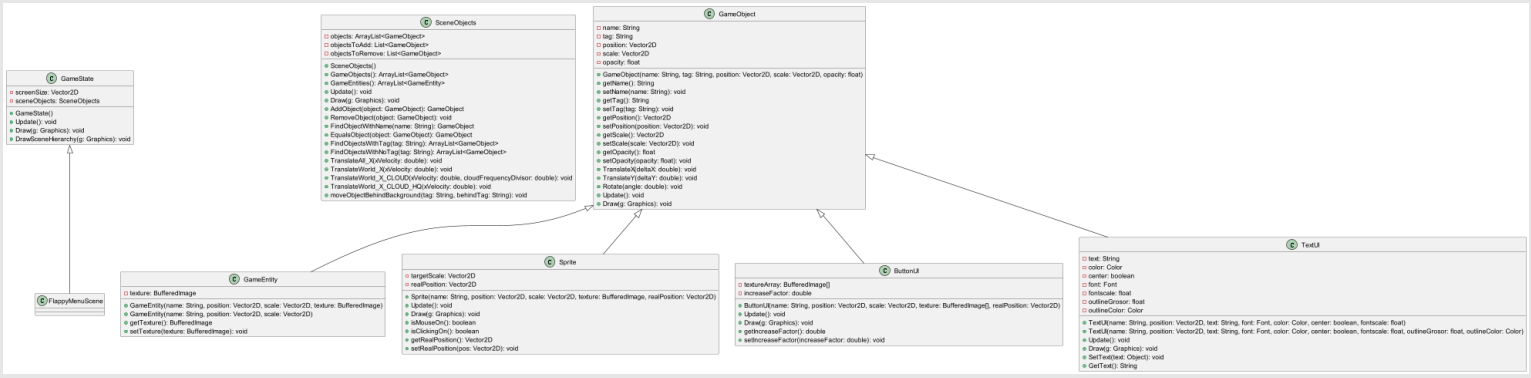


Diagrama (Escena de juego)

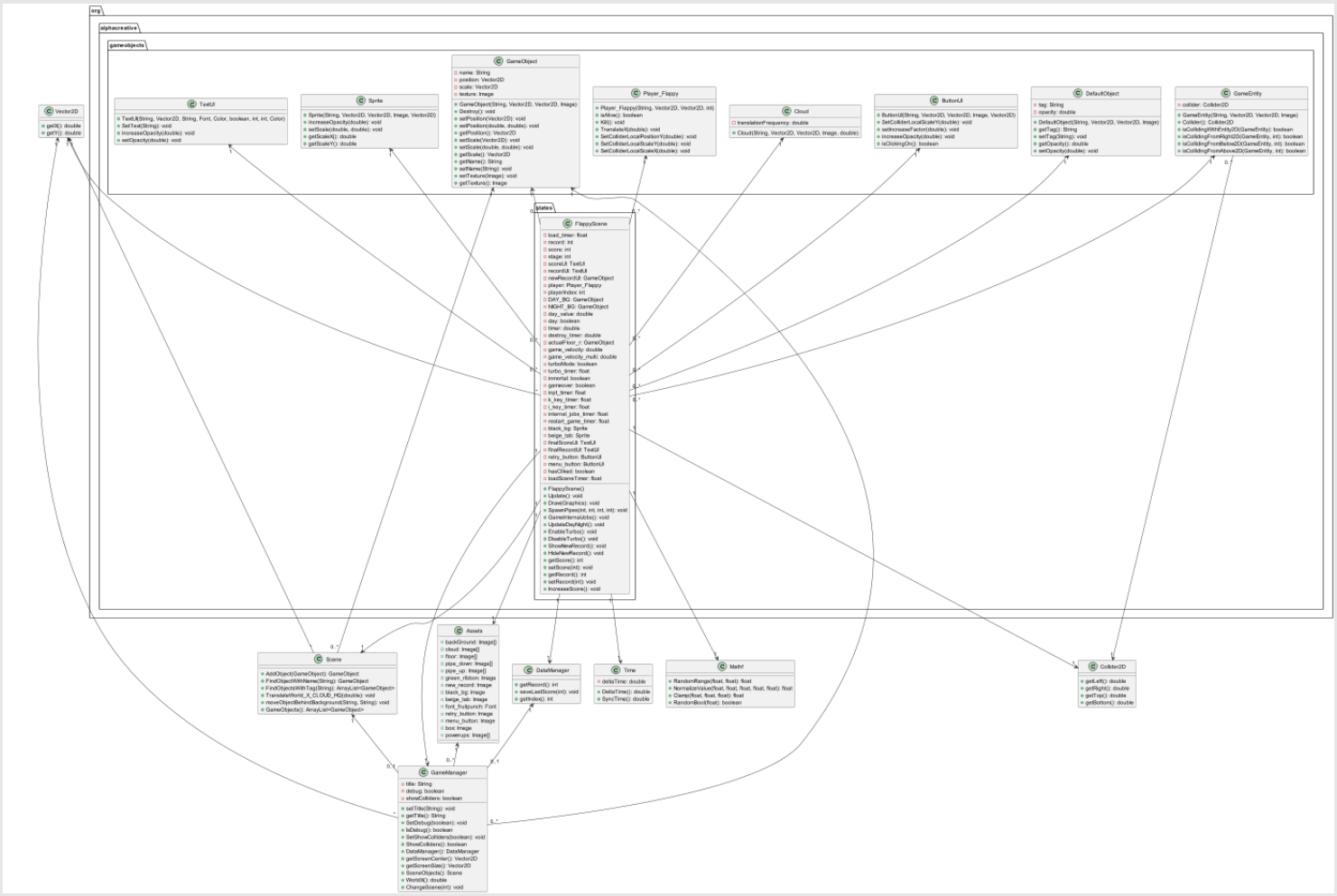
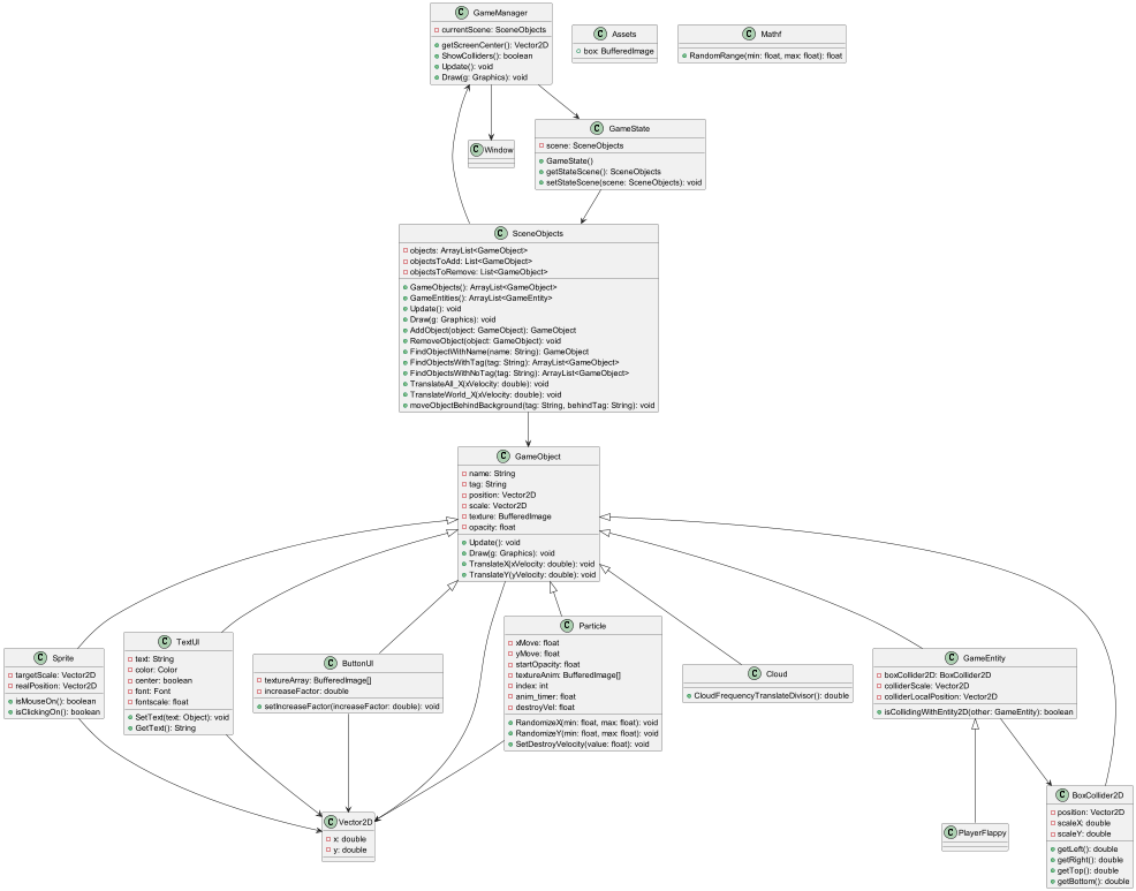


Diagrama genérico



Mikael Rodríguez
1ºDAM
Programación