Proyecto #3

Algoritmos y Estructura de Datos I

Estudiantes:

Carlos Andrés Jiménez Brenes

Leandro José Ruíz Acuña

2023125267

2023184994

Instituto Tecnológico de Costa Rica

AirWar

Profesor:

Leonardo Andrés Araya Martínez

Tabla de contenido

Introducción 3

Diseño 4

a. 4

b. 5

i. 5

ii. 16

iii. 29

c. 30

d. 33

e. 34

Introducción

AirWar es un juego de guerra aérea. Que trata de derribar aviones con un arma. Cuando se inicia el juego se crea un mapa que muestra aeropuertos, porta aviones y aviones en todo el mapeado, la función del jugador es derribar los aviones con una bomba antiaérea, el objetivo del jugador es eliminar la mayor cantidad de aviones que pueda

Diseño

a.

ID	Descripción
001	El objetivo del juego es destruir la mayor cantidad de aviones en un periodo de tiempo definido por el estudiante.
002	Generación aleatoria de aeropuertos, portaaviones y rutas. Se modela con un grafo utilizando listas de adyacencia.
003	Batería antiaérea que se mueve en velocidad constante entre izquierda y derecha de la pantalla. El jugador presiona click para disparar balas con trayectoria recta y de velocidad variable según el tiempo que se presionó el click antes de lanzar la bala.
004	Entre aeropuertos y portaaviones, se generarán rutas aleatorias con distintos pesos, considerando que: • Una ruta tendrá un peso general dado por la distancia entre los puntos que conecta. • El peso de una ruta considerará también el destino que conecta. Es más caro aterrizar en un portaaviones que en un aeropuerto y es más caro seguir una ruta interoceánica (que atraviesa el océano) que una continental.
005	Cuando un avión va a despegar, decide a qué destino quiere ir aleatoriamente y calcula la mejor ruta a ese destino, considerando los pesos anteriormente dados. Cuando un avión aterriza, dura una cantidad aleatoria de segundos antes de retomar una nueva ruta. Durante el tiempo de espera, el avión recarga una cantidad aleatoria de combustible.
006	No necesariamente todos los aeropuertos tienen suficiente combustible para todos los aviones, por lo que lo racionan, de alguna forma determinada por el estudiante. Un avión puede caerse si se le acaba el combustible antes de aterrizar.
007	Cada cierto tiempo los aeropuertos deben construir nuevos aviones. Por regulaciones internacionales, cada aeropuerto no podrá generar más aviones que la que permita sus hangares; por lo tanto, uno de los atributos de los aeropuertos es la cantidad de aviones que soporta su hangar. Los aviones deben tener un ID generado aleatoriamente utilizando GUIDs.
008	Suponga que los aviones son realmente drones autónomos (se manejan solos), pero que contienen cuatro módulos de AI independientes que trabajan juntos para volar el avión. Los módulos de AI se llaman: 1. Pilot: maneja el avión activamente. 2. Copilot: se encuentra en standby y tomará control activo del avión únicamente si algo sucede con el módulo Pilot. 3. Manteinance: identifica e intenta corregir cualquier problema en las piezas físicas del avión. 4. Space awarness: es el módulo encargado de monitorear los sensores y radares, para darle insumos al módulo Pilot. Y de cada uno de ellos, interesa conocer su: 1. ID: es un identificador que consta de tres letras (de la A a la Z) generadas aleatoriamente a la hora de crear el avión. 2. Rol (piloto, copiloto, asistente) 3. Horas de vuelo El sistema debe permitir obtener la lista de aviones derribados y ordenarlos por su ID usando Merge Sort.
009	Para cada avión derribado, es posible obtener su tripulación (módulos Pilot, Copilot, Manteinance y Space Awarness) y ordenarlo por ID, rol o horas de vuelo, utilizando Selection Sort.
010	En pantalla se debe mostrar todos los datos relevantes del juego. Por ejemplo, se deben mostrar los caminos calculados por los aviones, los pesos de cada ruta, los atributos de los aviones, entre otros.

b.

Los cinco problemas serán los siguientes:

001	El objetivo del juego es destruir la mayor cantidad de aviones en un periodo de tiempo definido por el estudiante.
002	Generación aleatoria de aeropuertos, portaaviones y rutas. Se modela con un grafo utilizando listas de adyacencia.
003	Batería antiaérea que se mueve en velocidad constante entre izquierda y derecha de la pantalla. El jugador presiona click para disparar balas con trayectoria recta y de velocidad variable según el tiempo que se presionó el click antes de lanzar la bala.
004	Entre aeropuertos y portaaviones, se generarán rutas aleatorias con distintos pesos, considerando que: • Una ruta tendrá un peso general dado por la distancia entre los puntos que conecta. • El peso de una ruta considerará también el destino que conecta. Es más caro aterrizar en un portaaviones que en un aeropuerto y es más caro seguir una ruta interoceánica (que atraviesa el océano) que una continental.
006	No necesariamente todos los aeropuertos tienen suficiente combustible para todos los aviones, por lo que lo racionan, de alguna forma determinada por el estudiante. Un avión puede caerse si se le acaba el combustible antes de aterrizar.

i.

Soluciones para cada problema:

001.

Las propuestas para poder crear un juego que cumpla el requerimiento son las siguientes:

Solución 1. Aviones con patrones de vuelo predefinidos

Descripción:

- Los aviones siguen trayectorias predefinidas que varían en dificultad (curvas, velocidad, o cambios de dirección inesperados).
- El jugador controla una nave (o defensa antiaérea) que puede moverse y disparar proyectiles para destruir los aviones.
- El tiempo límite se ajusta desde el menú principal.

Implementación:

- **Generación de aviones**: Se generan aviones en posiciones aleatorias fuera de la pantalla y se les asignan trayectorias preprogramadas (rectas, zigzag, espirales, etc.).

- **Controles**: El jugador usa teclas o el mouse para apuntar y disparar, en nuestro caso se implementó la barra espaciadora para disparar.
- **Colisiones**: Cada proyectil que impacta a un avión lo destruye, incrementando el puntaje.
- **Temporizador**: Un cronómetro en pantalla muestra el tiempo restante.
- **Fin del juego**: Al finalizar el tiempo, el juego muestra un resumen del puntaje total.

Factores adicionales:

- Dificultad progresiva: A medida que avanza el juego, aparecen más aviones, se mueven más rápido o tienen trayectorias más complejas.
- Potenciadores: Munición más rápida, disparos múltiples, o una habilidad de destrucción en área.

Solución 2: Aviones controlados por inteligencia artificial (IA)

Descripción:

- Los aviones tienen un comportamiento controlado por IA, como esquivar proyectiles, agruparse en formación, o atacar al jugador.
- El jugador tiene que usar estrategia y reflejos para maximizar la cantidad de aviones destruidos.

Implementación:

- **Generación de aviones**: Los aviones aparecen en oleadas, cada una con un patrón de movimiento diferente basado en la IA.
- **Controles**: Igual que en la Solución 1, pero con énfasis en ataques más estratégicos, como disparar en un área donde los aviones se agrupan.
- **IA básica**:
    - Los aviones más simples se mueven recto y no reaccionan.
    - Los avanzados esquivan disparos o atacan al jugador si se acercan lo suficiente.
- **Puntaje**: Los aviones más difíciles de destruir otorgan más puntos.
- **Fin del juego**: Mismo sistema de cronómetro y resumen de puntaje.

002.

Las propuestas para poder crear un juego que cumpla el requerimiento son las siguientes:

Solución 1: Uso de un grafo no dirigido simple

Descripción:

- Cada nodo en el grafo representa un aeropuerto o un portaaviones.
- Las aristas conectan nodos y representan rutas entre ellos, donde el peso de cada arista es la distancia euclidiana entre los nodos.
- Las conexiones entre nodos se generan aleatoriamente con ciertas restricciones para evitar un grafo completamente conectado (por ejemplo, probabilidad de conexión).
- Se utiliza una lista de adyacencia para modelar el grafo, almacenando para cada nodo una lista de nodos conectados con sus respectivas distancias.

Implementación conceptual:

1. **Generación de nodos:** Generar una cantidad fija de nodos (aeropuertos y portaaviones) con coordenadas aleatorias en un plano 2D.
2. **Cálculo de distancias:** Usar la fórmula de distancia euclidiana para calcular las distancias entre pares de nodos.
3. **Generación de aristas:** Conectar nodos aleatoriamente basándose en una probabilidad predefinida o en un umbral de distancia máxima.
4. **Lista de adyacencia:** Representar el grafo como un diccionario o mapa, donde la clave es un nodo y el valor es una lista de pares (nodo_destino, distancia).

Solución 2: Grafo con características adicionales (interoceánico y aterrizaje)

Descripción:

- Similar al grafo básico, pero los pesos de las aristas consideran no solo la distancia, sino también factores como:
    - **Bonificaciones:** Si la ruta incluye un aterrizaje en un portaaviones.
    - **Penalizaciones:** Si la ruta es interoceánica (por ejemplo, cruza un umbral definido en el plano 2D).
- Esto se logra multiplicando la distancia por un factor adicional según las condiciones específicas.
- Se utiliza una lista de adyacencia extendida que incluye tanto la distancia como el peso final de cada arista.

Implementación conceptual:

1. **Generación de nodos:** Igual que en la primera solución.
2. **Cálculo de distancias:** Igual que en la primera solución.
3. **Factores adicionales:**
    - Si el nodo de destino representa un portaaviones, aplicar una bonificación (reducir el peso).
    - Si la ruta es interoceánica, aplicar una penalización (incrementar el peso).
4. **Lista de adyacencia extendida:** Representar cada arista con un par (nodo_destino, (distancia, peso_ajustado)).

003.

Las propuestas para poder crear un juego que cumpla el requerimiento son las siguientes:

Solución 1: Movimiento básico con entrada de usuario y físicas

Descripción

- **Movimiento de la batería**:
  La batería se desplaza automáticamente de un lado a otro de la pantalla a velocidad constante, rebotando en los bordes definidos. Este comportamiento es simple y no utiliza sistemas de físicas.
- **Disparo de balas**:
  Al hacer clic, se mide el tiempo durante el cual el botón del ratón está presionado. Cuando se suelta, se genera una bala cuya velocidad depende de este tiempo de carga. Las balas se mueven hacia arriba en una trayectoria recta y se destruyen al salir de la pantalla.

Implementación

1. **Movimiento de la batería**:
   - Define los límites de movimiento en el eje X, como -10 y 10.
   - La batería cambia su dirección multiplicando su velocidad por -1 al alcanzar los límites.
   - Usa la posición del objeto y el tiempo transcurrido (Time.deltaTime) para calcular su desplazamiento.
2. **Disparo de balas**:
   - Escucha el evento de clic (MouseButtonDown) para iniciar un temporizador y mide cuánto tiempo está presionado.
   - Cuando el clic se libera (MouseButtonUp), calcula la velocidad de la bala en función del tiempo acumulado.
   - Genera una nueva bala y establece su velocidad hacia arriba.

3. **Balas y destrucción**:
    - Crea un prefab para las balas. Las balas se mueven en línea recta hacia arriba usando una fórmula básica:

$$nueva\ posición = posición\ actual + (velocidad\ \times \Delta t)$$

    - Las balas se destruyen cuando superan un límite superior en el eje Y.

Solución 2: Movimiento con Rigidbody y entrada basada en eventos

Descripción

- **Movimiento de la batería**:
  La batería utiliza un componente de físicas (Rigidbody) para moverse, lo que permite un manejo más realista y escalable si se quiere agregar más complejidad. El movimiento sigue siendo constante, rebotando en los bordes definidos.
- **Disparo de balas**:
  El sistema detecta cuándo el jugador hace clic y usa un temporizador para medir el tiempo de carga. Al soltar el botón, la bala se dispara con una velocidad basada en una curva matemática, proporcionando un control más refinado sobre la velocidad.
- **Trayectoria de las balas**:
  Las balas se disparan hacia arriba, con una trayectoria controlada por físicas, lo que permite futuras interacciones con otros objetos mediante colisiones. Se destruyen automáticamente al salir de la pantalla.

Implementación

1. **Movimiento de la batería**:
    - Usa un Rigidbody para manejar el movimiento horizontal aplicando una velocidad constante en el eje X.
    - Configura límites de movimiento y cambia la dirección al llegar a un borde ajustando la velocidad del cuerpo rígido.
    - Esto permite que la batería sea afectada por otros componentes de físicas en el futuro, si es necesario.

2. **Cálculo de la velocidad de disparo**:
    - Usa una fórmula no lineal (como una curva cuadrática) para que la velocidad de las balas sea más controlable:

$$velocidad = base + (tiempo\ de\ carga^2 \times factor\ de\ escala)$$

    - Establece límites para la velocidad mínima y máxima.
3. **Gestión de las balas**:
    - Aplica una fuerza inicial a las balas mediante el Rigidbody, usando la velocidad calculada.
    - Usa detección de colisiones para permitir interacciones con enemigos u obstáculos.
    - Configura un sistema que destruye las balas cuando están fuera de los límites de la pantalla (OnBecameInvisible).
4. **Optimización**:
    - Implementa un sistema de pooling de objetos para las balas en lugar de crear y destruir instancias repetidamente. Esto mejora el rendimiento en juegos con muchas balas activas.

004.

Las propuestas para poder crear un juego que cumpla el requerimiento son las siguientes:

Solución 1: Función de Ponderación Dinámica

Descripción

Esta solución asigna un peso a cada ruta según varios factores:

- **Distancia:** La base del peso es la distancia entre los dos puntos.
- **Destino:** Aterrizar en un portaaviones es más caro que en un aeropuerto.
- **Ruta Interoceánica:** Las rutas que cruzan océanos tienen un costo adicional.

El cálculo del peso se realiza mediante una fórmula que suma el impacto de estos factores. Los coeficientes del costo por destino y tipo de ruta pueden ajustarse para equilibrar el juego según las necesidades del diseño.

Implementación

1. **Definir las rutas:** Cada ruta se representa como una entidad con los atributos:
    - Distancia: calculada entre los puntos en el mapa.
    - Indicadores: si la ruta es interoceánica y si el destino es un portaaviones.
2. **Calcular el peso:** Una función toma estos atributos y retorna el peso total.
3. **Ajustar coeficientes:** Los valores asociados a aterrizar en un portaaviones y cruzar océanos pueden configurarse dinámicamente para modificar la dificultad o el costo en el juego.

Ventajas:

- Es simple y directa.
- Permite calcular el peso de cada ruta en tiempo real.
- Muy útil si las rutas se generan dinámicamente y no forman parte de una red más amplia.

Solución 2: Grafo Ponderado con Algoritmo de Caminos Óptimos

Descripción

En esta solución, los aeropuertos y portaaviones se modelan como nodos en un grafo, mientras que las rutas entre ellos son las aristas. Cada arista tiene un peso basado en:

- Distancia: La longitud de la ruta en el mapa.
- Destino: Costos adicionales si el destino es un portaaviones.
- Ruta Interoceánica: Penalización adicional si la ruta cruza océanos.

El grafo permite analizar la red completa y encontrar rutas óptimas (las menos costosas) entre nodos.

Implementación

1. **Construir el grafo:**
    - Crea nodos para todos los aeropuertos y portaaviones.
    - Conecta los nodos con aristas, asignando pesos calculados según la distancia y los costos adicionales.
2. **Calcular pesos: Los pesos de las aristas se determinan sumando:**
    - La distancia entre nodos.
    - Un costo adicional para rutas interoceánicas.
    - Un costo adicional si el destino es un portaaviones.
3. **Encontrar rutas óptimas:**
    - Usa un algoritmo como A* o Dijkstra para identificar el camino con menor peso entre dos nodos.
    - Dijkstra es ideal para un grafo denso con rutas predefinidas, mientras que A* es más eficiente si buscas optimizar recorridos en áreas grandes.

006.

Las propuestas para poder crear un juego que cumpla el requerimiento son las siguientes:

Solución 1: Sistema de Planificación de Rutas con Reabastecimiento Inteligente

Idea principal:
Antes de despegar, cada avión planifica una ruta optimizada basada en los niveles de combustible disponibles en los aeropuertos y la distancia necesaria para llegar a su destino.

Pasos:

1. **Evaluación inicial del combustible del avión:**
   Cada avión comienza con una cierta cantidad de combustible y un rango máximo.
   - Si el destino está fuera del rango inicial, el avión debe realizar paradas en aeropuertos intermedios.
2. **Asignación de racionamiento en los aeropuertos:**
   Los aeropuertos priorizan distribuir el combustible según la criticidad:
   - Aviones con destinos más lejanos tienen prioridad.
   - Si dos aviones tienen destinos similares, el combustible se reparte equitativamente.
3. **Cálculo dinámico de rutas:**
   Antes de cada despegue, el sistema verifica si el aeropuerto de destino puede proveer suficiente combustible para el próximo tramo. Si no, busca el aeropuerto más cercano con capacidad de reabastecimiento.
4. **Prevención de accidentes:**
   Si un avión está en riesgo de quedarse sin combustible, recibe instrucciones para aterrizar de emergencia en el aeropuerto más cercano dentro de su rango actual.

Solución 2: Simulación de Economía de Combustible y Decisión Local

Idea principal:
Cada aeropuerto y avión toma decisiones autónomas basadas en su estado local (disponibilidad de combustible y distancia al siguiente destino).

Pasos:

1. **Economía de combustible en los aeropuertos:**
   Los aeropuertos imponen restricciones en la cantidad de combustible que pueden ofrecer a cada avión dependiendo de:
   - La demanda de combustible en el día.
   - Su nivel de reservas.
2. **Reglas para los aviones:**
   - Los aviones deben conservar combustible al máximo (volando más lento o eligiendo rutas menos directas pero más cortas en distancia).
   - Si un aeropuerto de destino tiene un nivel bajo de combustible, el avión debe redirigirse a un aeropuerto alternativo antes de continuar.
3. **Gestión de emergencias:**
   Si un avión está cerca de quedarse sin combustible, automáticamente entra en modo de emergencia y selecciona el aeropuerto más cercano donde aterrizar.
4. **Penalización por racionamiento:**
   Si un avión no puede despegar debido a falta de combustible, la operación se reprograma o se comparten tanques móviles de combustible desde aeropuertos cercanos.

ii.

Para el requerimiento 001:

Solución 1: Aviones con Patrones de Vuelo Predefinidos

Ventajas:

1. **Simplicidad en la implementación:**
   Los patrones de vuelo predefinidos son fáciles de implementar y no requieren de algoritmos complejos, lo que facilita el desarrollo del juego.
2. **Previsibilidad en la dificultad:**
   Como los patrones de los aviones son fijos, la dificultad puede ser ajustada fácilmente al variar la velocidad, la complejidad de las trayectorias o el número de aviones.
3. **Control del jugador:**
   El jugador tiene más control sobre cómo interactuar con los aviones, ya que el comportamiento de los aviones es predecible. Esto permite una jugabilidad más controlada y no tan impredecible.
4. **Optimización de recursos:**
   Al no tener que calcular IA compleja, el juego puede ser más eficiente en términos de rendimiento, especialmente en dispositivos con menos recursos.
5. **Escalabilidad fácil:**
   Puedes fácilmente agregar más niveles o más aviones simplemente ajustando los patrones predefinidos, lo que permite un diseño más simple y expansible.

Desventajas:

1. **Falta de dinamismo:**
   Los aviones siguen un patrón fijo, lo que puede hacer que el juego se vuelva repetitivo y predecible después de un tiempo, afectando la emoción y desafío.
2. **Menos realismo:**
   La falta de comportamiento adaptativo de los aviones puede hacer que la experiencia de juego sea menos realista o inmersiva, ya que los enemigos no reaccionan de manera lógica o estratégica.

3. **Jugabilidad limitada:**
   La jugabilidad puede volverse monótona porque no hay sorpresas ni cambios de comportamiento por parte de los aviones, lo que puede disminuir el interés del jugador a largo plazo.

4. **Dificultad limitada:**
   La dificultad no puede escalar de manera tan dinámica como con una IA avanzada, ya que está limitada a las variaciones de los patrones de vuelo predefinidos.

Solución 2: Aviones Controlados por Inteligencia Artificial (IA)

Ventajas:

1. **Comportamiento más dinámico:**
   Los aviones controlados por IA ofrecen una jugabilidad más impredecible y dinámica, ya que los aviones pueden adaptarse a las acciones del jugador, hacer movimientos evasivos o atacar en grupo.

2. **Mayor desafío:**
   La IA puede crear situaciones más complejas y desafiantes para el jugador, aumentando la dificultad a medida que el juego avanza. Esto puede mantener al jugador interesado durante más tiempo.

3. **Mayor realismo:**
   Los aviones que reaccionan a los movimientos del jugador y siguen estrategias lógicas (esquivar disparos, atacar cuando están cerca, etc.) ofrecen una experiencia de juego más realista e inmersiva.

4. **Variedad en la jugabilidad:**
   Con la IA, los aviones pueden actuar de diversas maneras, lo que hace que el jugador se enfrente a situaciones siempre cambiantes. Esto agrega variedad y permite más profundidad estratégica en las decisiones del jugador.

5. **Potencial de mejora y personalización:**
   La IA se puede ajustar para mejorar la jugabilidad a través de actualizaciones o modificando su comportamiento para hacerla más desafiante según el progreso del jugador.

Desventajas:

1. **Complejidad en la implementación:**
   Crear una IA que se mueva de manera realista, esquive proyectiles o actúe en grupo puede ser mucho más difícil de implementar y requerir algoritmos más complejos, lo que incrementa el tiempo de desarrollo.
2. **Rendimiento del juego:**
   Los cálculos para manejar IA avanzada pueden afectar el rendimiento del juego, especialmente en dispositivos con menor capacidad de procesamiento o cuando hay muchos aviones en pantalla.
3. **Dificultad de equilibrado:**
   El comportamiento impredecible de la IA puede ser difícil de equilibrar. Si los aviones se vuelven demasiado inteligentes, el juego puede volverse frustrante para el jugador; si son demasiado fáciles, perdería desafío.
4. **Mayor demanda de recursos:**
   La IA generalmente consume más recursos de CPU y memoria en comparación con patrones de vuelo predefinidos, lo que puede limitar el número de aviones o la cantidad de otras mecánicas en el juego.

Para el requerimiento 002:

Solución 1: Uso de un Grafo No Dirigido Simple

Ventajas:

1. **Simplicidad de implementación:**
   - La estructura básica del grafo es sencilla de entender y fácil de implementar.
2. **Eficiencia:**
   - La lista de adyacencia es una representación eficiente en términos de espacio y acceso rápido a los nodos vecinos.
3. **Escalabilidad:**
   - La solución es fácilmente escalable al aumentar el número de nodos (aeropuertos, portaaviones) o rutas sin complejizar demasiado la estructura.

4. **Generación aleatoria controlada:**
    - Al generar conexiones aleatorias, se pueden evitar grafos completamente conectados, lo que agrega algo de variabilidad al problema.

Desventajas:

1. **Falta de realismo:**
    - No considera factores específicos como bonificaciones por aterrizajes en portaaviones o penalizaciones por rutas interoceánicas, lo que hace que las rutas sean puramente basadas en distancia.
2. **Falta de personalización:**
    - La estructura no permite modificar fácilmente las rutas o los nodos según reglas específicas, limitando la flexibilidad.
3. **Escasa flexibilidad en el modelado:**
    - La ruta es solo una conexión entre dos nodos sin tener en cuenta ninguna característica especial de los caminos (como el tipo de terreno o peligros).

Solución 2: Grafo con Características Adicionales (Interoceánico y Aterrizaje)

Ventajas:

1. **Mayor realismo:**
    - Al incorporar factores adicionales (bonificaciones por aterrizajes y penalizaciones por rutas interoceánicas), el grafo refleja un comportamiento más realista y complejo de las rutas.
2. **Personalización:**
    - Permite personalizar las reglas del grafo según las necesidades del modelo (por ejemplo, ajustar las rutas según condiciones específicas).
3. **Flexibilidad para adaptar reglas:**
    - Puedes modificar fácilmente las condiciones del grafo, como agregar más penalizaciones o bonificaciones según el comportamiento del sistema.

4. **Modelado más preciso:**
    - El grafo puede ser utilizado para modelar escenarios más complejos, como rutas de aviación que dependen no solo de la distancia sino también de otras condiciones que afectan las decisiones.

Desventajas:

1. **Mayor complejidad:**
    - La implementación es más compleja debido a los factores adicionales que deben ser considerados al calcular el peso de las aristas.
2. **Mayor consumo de recursos:**
    - El cálculo adicional de bonificaciones y penalizaciones puede aumentar la carga computacional, especialmente si el número de nodos y conexiones es grande.
3. **Mantenimiento más complejo:**
    - Las modificaciones en las reglas del grafo pueden requerir ajustes más frecuentes y detallados en el código, lo que hace que el mantenimiento sea más costoso en términos de tiempo y esfuerzo.
4. **Posible sobrecarga en la representación:**
    - Al extender la lista de adyacencia para incluir pesos ajustados, puede volverse más difícil de gestionar y mantener, especialmente si se agregan muchos factores adicionales.

Para el requerimiento 003:

Solución 1: Movimiento Básico con Entrada de Usuario y Físicas

Ventajas:

1. **Simplicidad y facilidad de implementación:**
    - Esta solución es fácil de implementar y entender, ya que no involucra el uso de componentes avanzados de físicas ni cálculos complejos.
2. **Buen rendimiento:**
    - Al no usar Rigidbody ni físicas complejas, el rendimiento será en general más rápido, especialmente en dispositivos de bajo rendimiento.
3. **Control directo del movimiento:**
    - El movimiento de la batería es directo y predecible, lo que puede ser ventajoso en juegos donde el control preciso es importante.
4. **Eficiencia en uso de recursos:**
    - El enfoque sencillo reduce el consumo de recursos, ya que se evita la sobrecarga de cálculos físicos o de movimiento complejo.

Desventajas:

1. **Falta de realismo:**
    - El movimiento de la batería es lineal y predecible, lo que puede resultar en una experiencia menos dinámica e inmersiva.
2. **Limitaciones en la física de colisiones:**
    - No se aprovechan las físicas, lo que limita las interacciones más complejas entre objetos, como rebotes o efectos de colisión realistas.
3. **Dificultad en la expansión futura:**
    - Si en el futuro se desean agregar interacciones físicas más complejas (como la gravedad o el deslizamiento), el sistema actual podría necesitar una reestructuración significativa.

4. **Falta de control fino sobre el disparo:**
    - Aunque el tiempo de carga ajusta la velocidad de la bala, el comportamiento de la bala es más básico y no permite una gran variedad en el control de su trayectoria.

Solución 2: Movimiento con Rigidbody y Entrada Basada en Eventos

Ventajas:

1. **Realismo y flexibilidad:**
    - El uso de Rigidbody permite un control más realista del movimiento y facilita la integración de interacciones físicas más complejas en el futuro (por ejemplo, gravedad, fuerzas externas).
2. **Mejor control sobre la trayectoria de las balas:**
    - Al usar una fórmula no lineal para calcular la velocidad de la bala, se puede obtener un control más preciso y flexible sobre su comportamiento, lo que mejora la jugabilidad.
3. **Escalabilidad:**
    - Este enfoque es más escalable para añadir nuevas características, como efectos de colisiones entre balas y otros objetos, o la inclusión de más efectos de físicas en el juego.
4. **Optimización con pooling de objetos:**
    - Implementar un sistema de pooling para las balas mejora el rendimiento en juegos con muchas instancias de balas, evitando la sobrecarga de crear y destruir objetos constantemente.

Desventajas:

1. **Mayor complejidad:**
    - El uso de Rigidbody y el cálculo de trayectorias no lineales agrega complejidad tanto en la implementación como en el mantenimiento del código.

2. **Mayor consumo de recursos:**
    - El uso de físicas y la gestión de objetos a través de Rigidbody pueden aumentar el consumo de recursos, lo que puede afectar el rendimiento en dispositivos con recursos limitados.
3. **Curva de aprendizaje más empinada:**
    - La integración de físicas y el cálculo de trayectorias más complejas puede requerir más tiempo para entender y ajustar, especialmente para quienes no están familiarizados con estos conceptos.
4. **Posibles problemas de rendimiento con muchas balas:**
    - Aunque el pooling ayuda, el cálculo de físicas y la gestión de colisiones de muchas balas simultáneas pueden seguir afectando el rendimiento si no se gestionan adecuadamente.

Para el requerimiento 004:

Solución 1: Función de Ponderación Dinámica

Ventajas:

1. **Simplicidad y flexibilidad:**
    - La implementación es directa y fácil de entender. No se requieren estructuras de datos complejas como grafos, lo que facilita el desarrollo.
2. **Calculo en tiempo real:**
    - El peso de cada ruta se calcula dinámicamente en función de varios factores, lo que permite adaptarse a cambios en tiempo real si los coeficientes se ajustan, como la dificultad o las condiciones de juego.
3. **Personalización:**
    - Los coeficientes para los diferentes factores (distancia, destino, ruta interoceánica) son ajustables, lo que ofrece flexibilidad para equilibrar el juego y modificar las rutas sin reestructurar completamente la solución.

4. **Ideal para rutas generadas dinámicamente:**
    - Si las rutas son generadas aleatoriamente o de manera procedural, esta solución es adecuada porque permite evaluar rápidamente el peso de cada nueva ruta sin necesidad de una red predefinida.

Desventajas:

1. **No optimiza globalmente las rutas:**
    - Solo calcula el peso de una ruta individual, pero no tiene en cuenta la optimización global entre múltiples rutas o nodos. Es posible que no siempre se elijan las rutas óptimas de manera general.
2. **Dependencia de la fórmula:**
    - El cálculo del peso depende de una fórmula, lo que puede hacer que el ajuste de los coeficientes sea complicado si no se encuentra la combinación correcta de factores.

3. **Limitada a la evaluación de rutas individuales:**
    - Si necesitas evaluar un conjunto de rutas y encontrar el camino más eficiente, esta solución no es ideal. El cálculo del peso no está integrado en un sistema que busque el camino óptimo global.

Solución 2: Grafo Ponderado con Algoritmo de Caminos Óptimos

Ventajas:

1. **Optimización global de rutas:**
    - Usar un grafo ponderado junto con un algoritmo de caminos óptimos (como Dijkstra o A*) permite encontrar el camino más eficiente entre dos nodos, considerando todas las rutas posibles.
2. **Escalabilidad y flexibilidad:**
    - Esta solución es fácilmente escalable para redes de rutas más complejas y extensas. La estructura de grafo permite agregar más nodos o aristas sin complicar demasiado el proceso de cálculo.

3. **Soporte para rutas interdependientes:**
    - Como el grafo modela todas las rutas entre los nodos, puede optimizar la búsqueda de rutas teniendo en cuenta la interdependencia de todas las conexiones, lo cual es útil en redes grandes y complejas.
4. **Optimización de recursos:**
    - Los algoritmos de caminos óptimos permiten calcular rutas eficientes en términos de tiempo, distancia o coste, lo que mejora la experiencia de usuario al hacer que las rutas sean más rápidas o menos costosas.

Desventajas:

1. **Mayor complejidad:**
    - La implementación de grafos y algoritmos de caminos óptimos como Dijkstra o A* es más compleja en comparación con el cálculo dinámico de pesos. Requiere más conocimiento de estructuras de datos y algoritmos de optimización.
2. **Requiere red predefinida de rutas:**
    - Necesita una red de rutas completamente definida antes de usar el algoritmo, lo que puede no ser adecuado para entornos donde las rutas cambian frecuentemente o se generan dinámicamente.
3. **Mayor consumo de recursos:**
    - El cálculo de rutas óptimas en grafos grandes puede ser intensivo en recursos y lento, especialmente si se usan algoritmos como Dijkstra en redes densas. Aunque A* es más eficiente, sigue siendo más costoso que una función de ponderación dinámica.
4. **Puede ser innecesario en redes simples:**
    - Si la red de rutas es pequeña o si no se requiere una optimización precisa, usar un grafo completo con algoritmos de optimización puede ser excesivo y no ofrecer beneficios significativos.

Para el requerimiento 006:

Solución 1: Sistema de Planificación de Rutas con Reabastecimiento Inteligente

Ventajas:

1. **Optimización de rutas:**
    - Los aviones optimizan sus rutas de acuerdo con la disponibilidad de combustible en los aeropuertos de destino y en los puntos intermedios, lo que mejora la eficiencia general del sistema de transporte.
2. **Prevención de accidentes:**
    - La planificación anticipada y el monitoreo constante del combustible aseguran que los aviones no se queden sin combustible, lo que minimiza el riesgo de accidentes y mejora la seguridad.
3. **Distribución equitativa de combustible:**
    - Los aeropuertos priorizan el reabastecimiento para aviones con destinos más lejanos, lo que garantiza que los vuelos de mayor prioridad reciban el combustible necesario, mientras que se distribuye equitativamente el combustible entre otros aviones cuando es posible.
4. **Flexibilidad:**
    - La capacidad de ajustar las rutas y las paradas en aeropuertos intermedios según la disponibilidad de combustible permite un sistema flexible que puede adaptarse a cambios en tiempo real.

Desventajas:

1. **Complejidad en la implementación:**
    - Requiere un sistema complejo que gestione dinámicamente las rutas, el combustible y las prioridades de los aviones, lo que puede ser difícil de implementar y mantener.
2. **Dependencia de la información en tiempo real:**
    - El sistema depende de datos en tiempo real sobre la disponibilidad de combustible, lo que puede ser desafiante si los datos no se actualizan de manera precisa o si los sistemas de monitoreo son ineficientes.

3. **Posible ineficiencia en aeropuertos pequeños:**
    - Los aeropuertos con recursos limitados pueden verse sobrecargados si hay una gran cantidad de aviones necesitando reabastecerse, lo que puede afectar la eficiencia general del sistema.
4. **Riesgo de congestión:**
    - Si muchos aviones dependen de un mismo aeropuerto para reabastecerse, podría haber retrasos o congestión en las operaciones de reabastecimiento, especialmente en momentos de alta demanda.

Solución 2: Simulación de Economía de Combustible y Decisión Local

Ventajas:

1. **Autonomía y descentralización:**
    - Los aeropuertos y aviones toman decisiones autónomas basadas en su estado local, lo que permite una mayor flexibilidad y reduce la dependencia de un sistema centralizado, permitiendo respuestas rápidas y adaptables.
2. **Ajuste dinámico a la disponibilidad de combustible:**
    - Los aviones adaptan su velocidad y trayectoria en función de la disponibilidad de combustible y las condiciones locales, lo que puede mejorar la eficiencia al reducir el consumo de combustible cuando sea necesario.
3. **Mejora de la eficiencia operativa:**
    - Al permitir que cada avión elija rutas más cortas o más lentas en función de la disponibilidad de combustible, el sistema ayuda a minimizar el desperdicio de recursos y a optimizar las operaciones, evitando la escasez de combustible.
4. **Gestión flexible de emergencias:**
    - El sistema tiene un mecanismo automático que permite a los aviones redirigir su destino a un aeropuerto cercano en caso de emergencia, lo que aumenta la seguridad y reduce los riesgos de accidentes por falta de combustible.

Desventajas:

1. **Falta de coordinación entre aeropuertos y aviones:**
    - Aunque el sistema descentralizado permite decisiones rápidas, puede haber falta de coordinación entre los aeropuertos y los aviones en cuanto a la asignación de combustible, lo que podría llevar a ineficiencias en la distribución de recursos.
2. **Posible aumento de costos operativos:**
    - La necesidad de tener reglas estrictas sobre la conservación de combustible y las decisiones autónomas puede generar costos adicionales, tanto en el monitoreo como en la implementación de medidas de emergencia (como compartir combustible o redirigir vuelos).
3. **Desviaciones no deseadas:**
    - Los aviones que toman decisiones locales para evitar un aeropuerto con poco combustible pueden terminar en rutas no óptimas, lo que podría aumentar la duración de los vuelos y el consumo de combustible en el proceso.
4. **Riesgo de decisiones subóptimas:**
    - Dado que las decisiones se toman localmente, es posible que no se estén considerando todas las opciones globales de forma óptima, lo que podría resultar en un sistema menos eficiente si se compara con un enfoque centralizado de planificación.

iii.

Para el requerimiento 001:

La solución 1 es la elegida , la razón de porque es porque implementar una IA es mucho más complejo y puede llegar a ser más pesado para el juego, en cambio predefinir el movimiento del avión de manera aleatoria es mucho mejor

Para el requerimiento 002:

La solución 2 fue la elegida, es más sencillo realizar un grafo con características adicionales para que se generen aleatoriamente y correctamente los aeropuertos y porta aviones en el mapa

Para el requerimiento 003:

La solución 2 fue la que se eligió, debido a que la función RigidBody es una que esta implementada en Unity lo cual hace más sencillo la implementación de esta

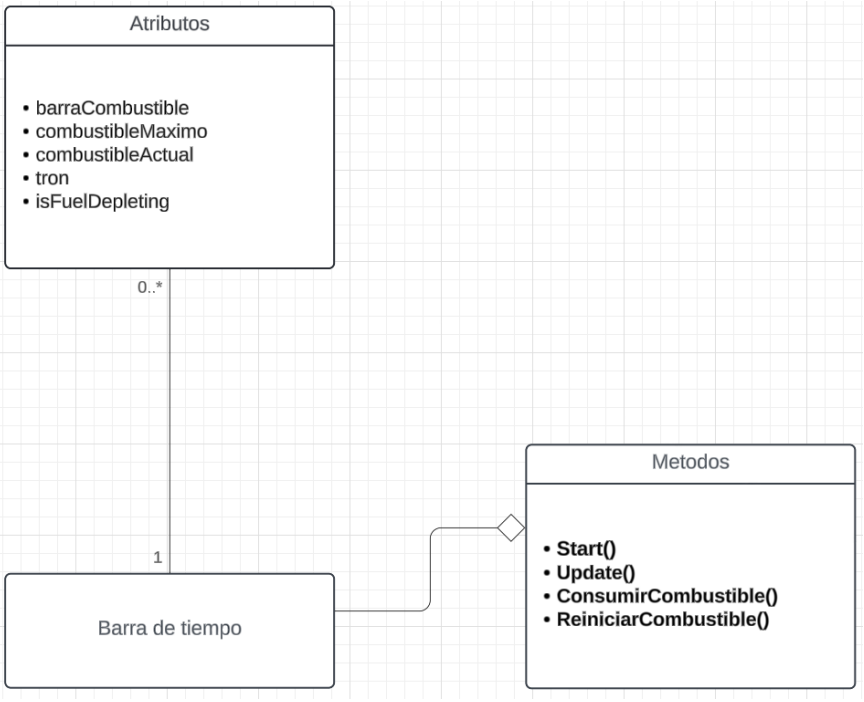
Para el requerimiento 004:

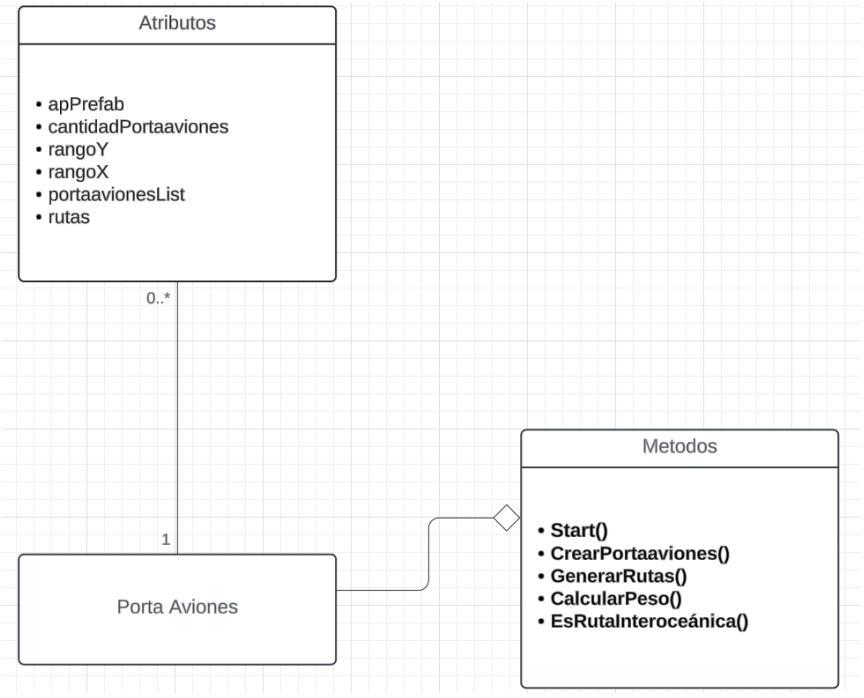
La solución 1 es la que se eligió, debido a que tomando los factores de los aeropuertos y porta aviones, la distancia entre ellos permite calcular un peso aleatorio entre ellos para calcular las rutas debido a su peso

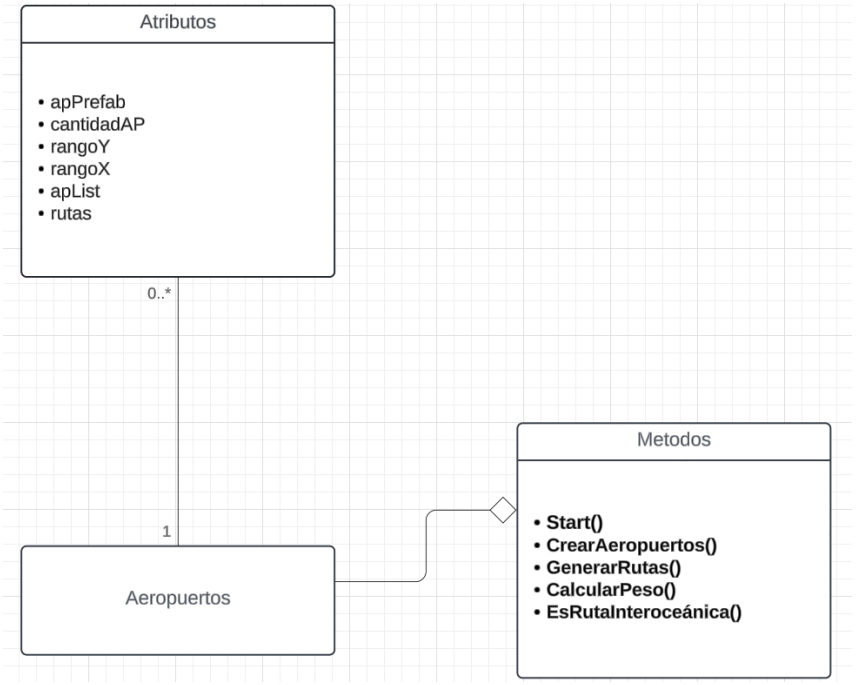
Para el requerimiento 006:

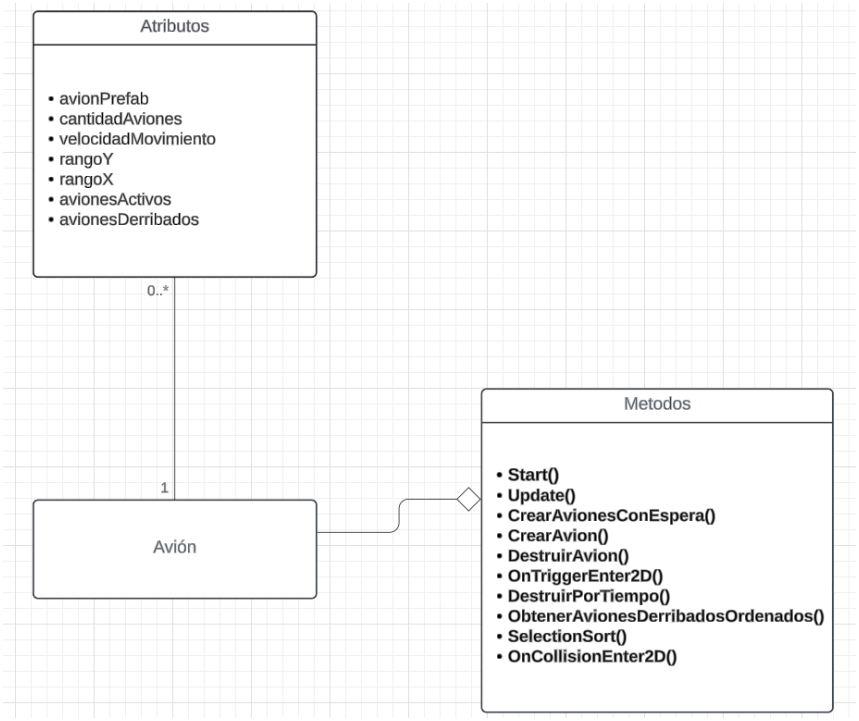
La solución 2 es la que se implementó, dándoles un valor de combustible aleatorio al pasar sobre los aeropuertos y porta aviones, esto para mantenerlos en el aire y que no desaparezcan

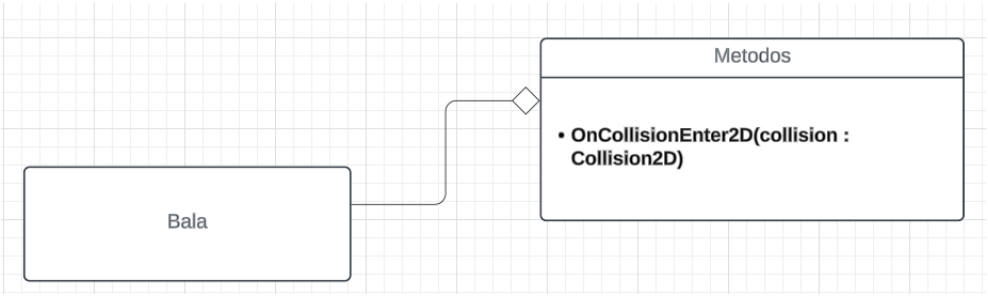
C.

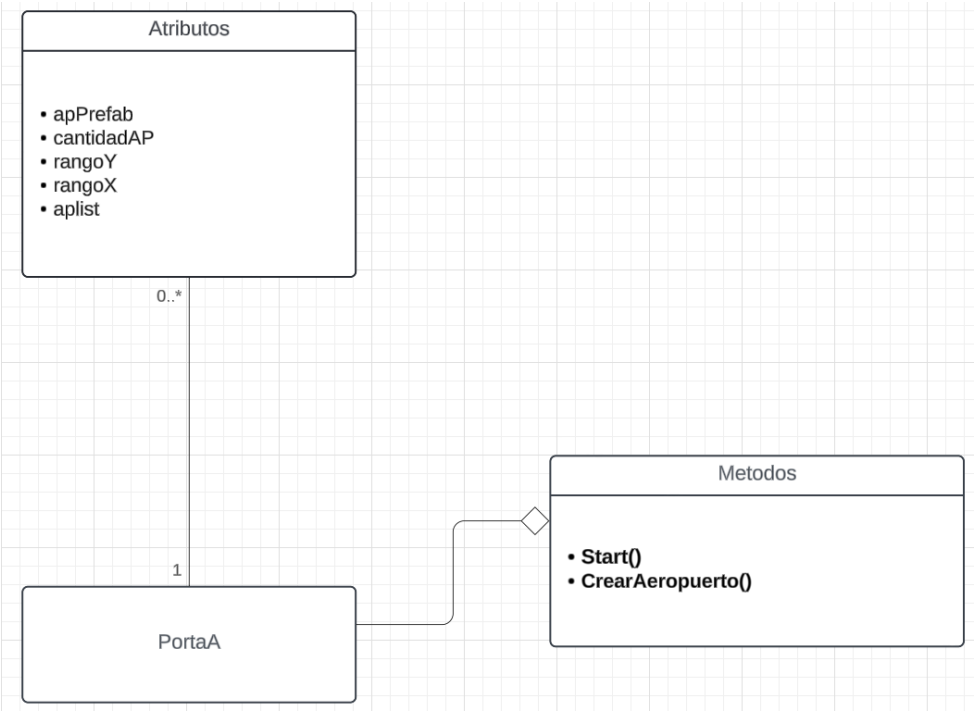

Atributos
• barraCombustible
• combustibleMaximo
• combustibleActual
• tron
• isFuelDepleting
0..*
1
Barra de tiempo
Metodos
• Start()
• Update()
• ConsumirCombustible()
• ReiniciarCombustible()


Atributos
• apPrefab
• cantidadPortaaviones
• rangoY
• rangoX
• portaavionesList
• rutas
0..*
1
Porta Aviones
Metodos
• Start()
• CrearPortaaviones()
• GenerarRutas()
• CalcularPeso()
• EsRutaInteroceánica()

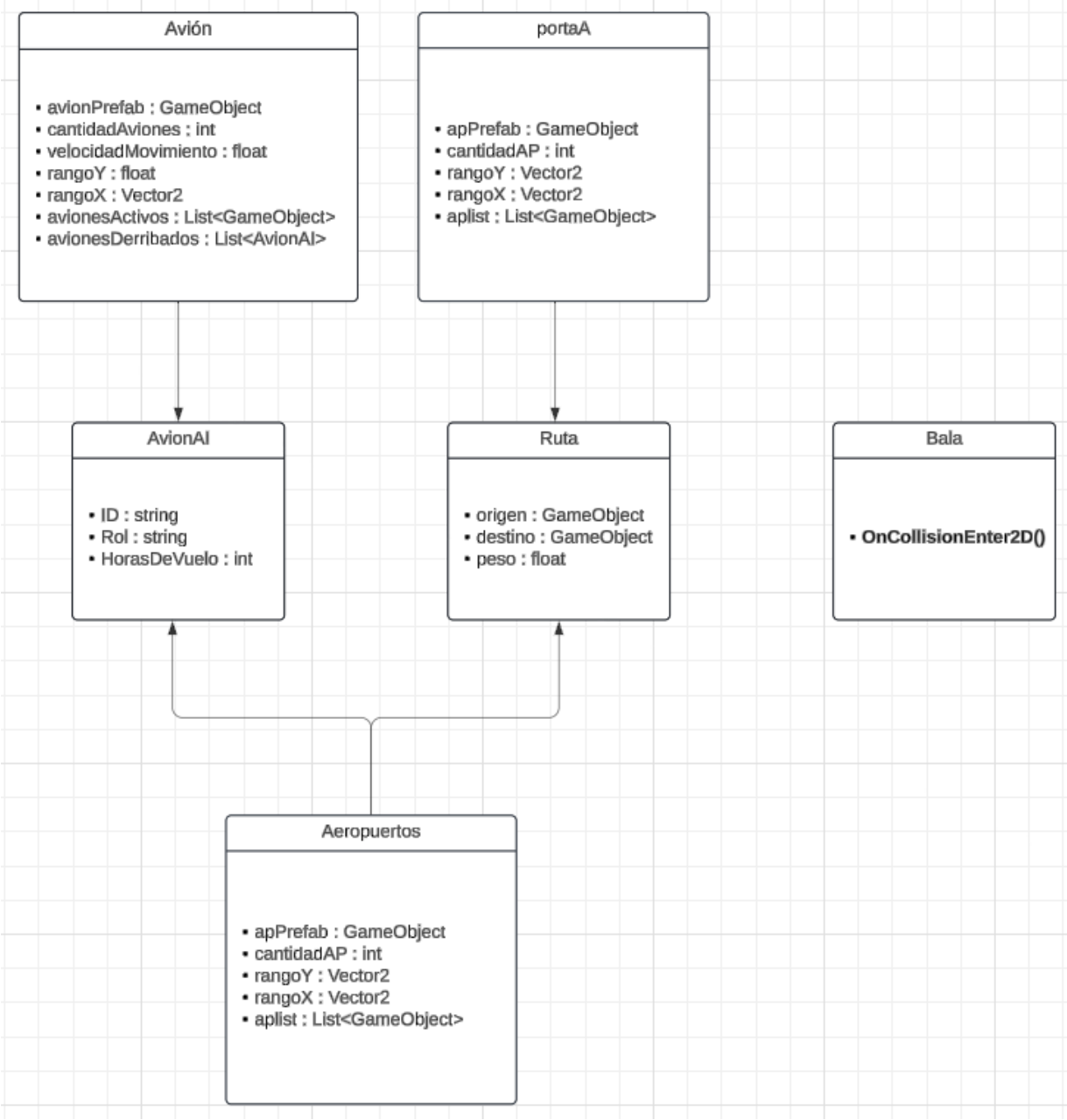

Atributos
• apPrefab
• cantidadAP
• rangoY
• rangoX
• apList
• rutas
0..*
1
Aeropuertos
Metodos
• Start()
• CrearAeropuertos()
• GenerarRutas()
• CalcularPeso()
• EsRutaInteroceánica()


Atributos
• avionPrefab
• cantidadAviones
• velocidadMovimiento
• rangoY
• rangoX
• avionesActivos
• avionesDerribados
0..*
1
Avión
Metodos
• Start()
• Update()
• CrearAvionesConEspera()
• CrearAvion()
• DestruirAvion()
• OnTriggerEnter2D()
• DestruirPorTiempo()
• ObtenerAvionesDerribadosOrdenados()
• SelectionSort()
• OnCollisionEnter2D()


Metodos
• OnCollisionEnter2D(collision : Collision2D)
Bala


Atributos
• apPrefab
• cantidadAP
• rangoY
• rangoX
• aplist
0..*
1
Metodos
• Start()
• CrearAeropuerto()
PortaA

d.


Avión
• avionPrefab : GameObject
• cantidadAviones : int
• velocidadMovimiento : float
• rangoY : float
• rangoX : Vector2
• avionesActivos : List<GameObject>
• avionesDerribados : List<AvionAI>
portaA
• apPrefab : GameObject
• cantidadAP : int
• rangoY : Vector2
• rangoX : Vector2
• aplist : List<GameObject>
AvionAI
• ID : string
• Rol : string
• HorasDeVuelo : int
Ruta
• origen : GameObject
• destino : GameObject
• peso : float
Bala
• OnCollisionEnter2D()
Aeropuertos
• apPrefab : GameObject
• cantidadAP : int
• rangoY : Vector2
• rangoX : Vector2
• aplist : List<GameObject>

e.

ID	Descripción	Verificación de implementación
001	El objetivo del juego es destruir la mayor cantidad de aviones en un periodo de tiempo definido por el estudiante.	VERDADERO
002	Generación aleatoria de aeropuertos, portaaviones y rutas. Se modela con un grafo utilizando listas de adyacencia.	VERDADERO
003	Batería antiaérea que se mueve en velocidad constante entre izquierda y derecha de la pantalla. El jugador presiona click para disparar balas con trayectoria recta y de velocidad variable según el tiempo que se presionó el click antes de lanzar la bala.	VERDADERO
004	Entre aeropuertos y portaaviones, se generarán rutas aleatorias con distintos pesos, considerando que: • Una ruta tendrá un peso general dado por la distancia entre los puntos que conecta. • El peso de una ruta considerará también el destino que conecta. Es más caro aterrizar en un portaaviones que en un aeropuerto y es más caro seguir una ruta interoceánica (que atraviesa el océano) que una continental.	VERDADERO
005	Cuando un avión va a despegar, decide a qué destino quiere ir aleatoriamente y calcula la mejor ruta a ese destino, considerando los pesos anteriormente dados. Cuando un avión aterriza, dura una cantidad aleatoria de segundos antes de retomar una nueva ruta. Durante el tiempo de espera, el avión recarga una cantidad aleatoria de combustible.	FALSO
006	No necesariamente todos los aeropuertos tienen suficiente combustible para todos los aviones, por lo que lo racionan, de alguna forma determinada por el estudiante. Un avión puede caerse si se le acaba el combustible antes de aterrizar.	FALSO
007	Cada cierto tiempo los aeropuertos deben construir nuevos aviones. Por regulaciones internacionales, cada aeropuerto no podrá generar más aviones que la que permita sus hangares; por lo tanto, uno de los atributos de los aeropuertos es la cantidad de aviones que soporta su hangar. Los aviones deben tener un ID generado aleatoriamente utilizando GUIDs.	VERDADERO
008	Suponga que los aviones son realmente drones autónomos (se manejan solos), pero que contienen cuatro módulos de AI independientes que trabajan juntos para volar el avión. Los módulos de AI se llaman: 1. Pilot: maneja el avión activamente. 2. Copilot: se encuentra en standby y tomará control activo del avión únicamente si algo sucede con el módulo Pilot. 3. Manteinance: identifica e intenta corregir cualquier problema en las piezas físicas del avión. 4. Space awarness: es el módulo encargado de monitorear los sensores y radares, para darle insumos al módulo Pilot. Y de cada uno de ellos, interesa conocer su: 1. ID: es un identificador que consta de tres letras (de la A a la Z) generadas aleatoriamente a la hora de crear el avión. 2. Rol (piloto, copiloto, asistente) 3. Horas de vuelo El sistema debe permitir obtener la lista de aviones derribados y ordenarlos por su ID usando Merge Sort.	VERDADERO
009	Para cada avión derribado, es posible obtener su tripulación (módulos Pilot, Copilot, Manteinance y Space Awarness) y ordenarlo por ID, rol o horas de vuelo, utilizando Selection Sort.	VERDADERO
010	En pantalla se debe mostrar todos los datos relevantes del juego. Por ejemplo, se deben mostrar los caminos calculados por los aviones, los pesos de cada ruta, los atributos de los aviones, entre otros.	VERDADERO