TEC | Tecnológico de Costa Rica

AirWar, 2024, 1–4

https://github.com/dgcgary/AirWar.git

AirWar

David Garcia Cruz[1,*,†] and Rodrigo Arce Bastos[2,*,†]

[1]Tecnologico de Costa Rica

[*]r.arce.1@estudiantec.cr [†]d.garcia.1@estudiantec.cr

Introducción

El presente documento tiene como propósito acreditar el desarrollo de competencias orientadas al diseño de soluciones innovadoras para problemas de ingeniería complejos. Este proceso considera no solo la satisfacción de necesidades específicas, sino también un enfoque integral que incorpora factores críticos como la salud y la seguridad pública, el costo total de vida, la sostenibilidad ambiental, el carbono neto cero, y las implicaciones culturales, sociales y de recursos.
A través de los objetivos planteados, se busca no solo identificar y analizar problemas técnicos, sino también evaluar alternativas viables y validar soluciones que sean coherentes con los principios de responsabilidad social y ambiental. De esta manera, se garantiza que los resultados estén alineados con las demandas actuales y futuras de la ingeniería en un contexto global.

Tabla de Contenidos

Diseño

User Stories

1. Como desarrollador quiero crear el juego utilizando C con WPF, MAUI o Unity para asegurar una experiencia visual y funcional óptima.

2. Como usuario quiero destruir la mayor cantidad de aviones en un período de tiempo definido para obtener la mayor puntuación posible dentro del tiempo asignado.

3. Como desarrollador quiero generar aeropuertos, portaaviones y rutas de forma aleatoria usando un grafo modelado para simular un entorno dinámico y desafiante en el juego.

4. Como usuario quiero controlar una batería antiaérea que se mueva a velocidad constante entre los lados de la pantalla y dispare balas según el tiempo de presión del click para derribar los aviones.

5. Como desarrollador quiero generar rutas aleatorias con pesos basados en distancia, tipos de destino y tipos de ruta para crear un sistema de rutas realista que afecte la dinámica del juego.

6. Como desarrollador quiero que el avion elija un destino aleatorio y calcular la mejor ruta según los pesos asignados para optimizar el uso de combustible y el tiempo de vuelo.

7. Como desarrollador quiero que los aeropuerto racionen el combustible de los aviones según una lógica definida para manejar limitaciones de recursos.

8. Como desarrollador quiero que los aeropuertos construyan nuevos aviones según la capacidad del hangar y asignarles un ID único generado con GUID para mantener un flujo constante de aviones dentro de los límites establecidos.

9. Como desarrollador quiero que los aviones tengan cuatro módulos de IA con identificadores únicos para asegurar el funcionamiento autónomo y el monitoreo del avión en todo momento.

10. Como administrador del sistema quiero obtener la lista de aviones derribados y ordenarlos por su ID utilizando Merge Sort para analizar y visualizar el desempeño del jugador.

Compiled on: November 26, 2024.
Draft manuscript prepared by the author.

11. Como usuario quiero ver en pantalla todos los datos relevantes del juego, como rutas calculadas, pesos de rutas y atributos de aviones para llevar un seguimiento de la partida.

12. Como administrador del sistema quiero obtener y ordenar por ID, rol o horas de vuelo la tripulación de cada avión derribado utilizando Selection Sort para analizar los detalles de cada módulo de avión de manera eficiente.

Problemas

Problema 1: Seleccionar el entorno para desarrollar el juego
Alternativas:

1. Usar Csharp con WPF

Ventajas: Ideal para aplicaciones de escritorio con una interfaz gráfica simple. Menor curva de aprendizaje para quienes ya tienen experiencia con WPF. Compatible con múltiples versiones de Windows.

Desventajas: Limitado en gráficos avanzados y animaciones. No es ideal para desarrollos multiplataforma o con gráficos complejos.

2. Usar Unity

Ventajas: Ideal para gráficos avanzados y animaciones 3D o 2D. Compatible con múltiples plataformas (Windows, Mac, móvil). Amplia comunidad y soporte.

Desventajas: Requiere mayor tiempo de aprendizaje si no se ha usado antes. Más consumo de recursos del sistema durante la ejecución.

Para el desarrollo del AirWar optamos ya por utilizar la alternativa 1, ya que es mas sencilla trabajar en WPF y es necesario investigar menos, a pesar de ser limitado tiene lo necesario para crear un videojuego simple.

Problema 2: Movimiento y disparo de la batería antiaérea
Alternativas:

1. Movimiento continuo en ambos sentidos con velocidad fija

Ventajas: Simplicidad de implementación. Predecible para el jugador, facilitando la estrategia de disparo.

Desventajas: Puede volverse monótono rápidamente. Menor dificultad para el jugador.

2. Movimiento con velocidad variable o patrones aleatorios

Ventajas: Incrementa la dificultad y el interés del juego. Ofrece una experiencia menos predecible.

Desventajas: Mayor complejidad en la programación. Puede ser frustrante si el patrón es demasiado errático.

Para este problema se opto por la alternativa 1 ya que esta se adapta a lo solicitado por la user story 4, ademas de ser mas sencilla su implementacion.

Problema 3: Generación de rutas entre aeropuertos y portaaviones
Alternativas:

1. Asignar pesos fijos dependiendo del tipo de si es aeropuerto o portaaviones por distancia

Ventajas: Implementación simple utilizando un algoritmo que guarde la posicion y modificar los valores fijos a partir de eso por ejemplo una ruta interoceanica un valor de 100 y una ruta a un aeropuerto un valor de 50.

Desventajas: No considera factores como el tipo de destino o el tipo de ruta. Menor realismo en la simulación.

2. Asignar pesos considerando distancia, tipo de destino y tipo de ruta

Ventajas: Simulación más realista y desafiante. Introduce factores estratégicos para los jugadores.

Desventajas: Mayor complejidad de implementación. Más recursos computacionales para el cálculo de rutas

Se opto por la alternativa 1 ya que la implementación es menos compleja y la manipulación de las rutas y sus pesos es mas sencilla.

Problema 4: Gestión de combustible en aeropuertos
Alternativas:

1. Establecer un algoritmo constante el cual rellena siempre lo mismo para todos los que pasen por un aeropuerto.

Ventajas: Simple de implementar y fácil de entender. Simula un sistema justo de asignación.

Desventajas: Podría llevar a un manejo ineficiente del recurso.

2. Priorizar aviones con menos combustible donde se le de una cantidad de gasolina proporcional dependiendo de la cantidad de gasolina en el tanque.

Ventajas: Reduce el riesgo de que aviones se queden sin combustible en el aire. Simula un sistema más estratégico y realista.

Desventajas: Mayor complejidad en el algoritmo de racionamiento. Posibles retrasos para otros aviones con niveles adecuados de combustible.

Se opto por utilizar la alternativa 1, ya que es mas facil de implementar y la manera de recargar el combustible es equitativa para todos los aviones.

Se opto

Problema 5: Ordenamiento de aviones derribados
Alternativas:

1. Usar Merge Sort

Ventajas: Eficiente para grandes listas (O(n log n)). Estable, preserva el orden relativo de elementos con claves iguales.

Desventajas: Mayor uso de memoria por la recursión. Más complejo de implementar manualmente.

2. Usar Quick Sort

Ventajas: Generalmente más rápido para conjuntos de datos pequeños o medianos. Menor uso de memoria que Merge Sort.

Desventajas: No garantiza estabilidad en el ordenamiento. En el peor caso, tiene una complejidad de $O(n^2)$.

Se opto por utilizar merge sort ya que se adapta a lo solicitado en la user story 10, ademas de proporcionar estabilidad y mejor manejo para listas muy grandes.

Diagrama UML

En la siguiente pagina se presenta el diagrama de clases para todo el proyecto cada clase con sus respectivos métodos y atributos. Las flechas con un rombo negro, representan una relación de composición, las flechas con un rombo vació representan una relación de agregación. Las lineas continuas con una flecha vacía representan herencia y la linea discontinua con la punta en V representa una relación de dependencia.

Checklist de user stories

Listado:

1. Completado
2. Incompleto
3. Completado
4. Completado
5. Incompleto
6. Incompleto
7. Completado
8. Incompleto
9. Incompleto
10. Incompleto
11. Incompleto
12. Incompleto

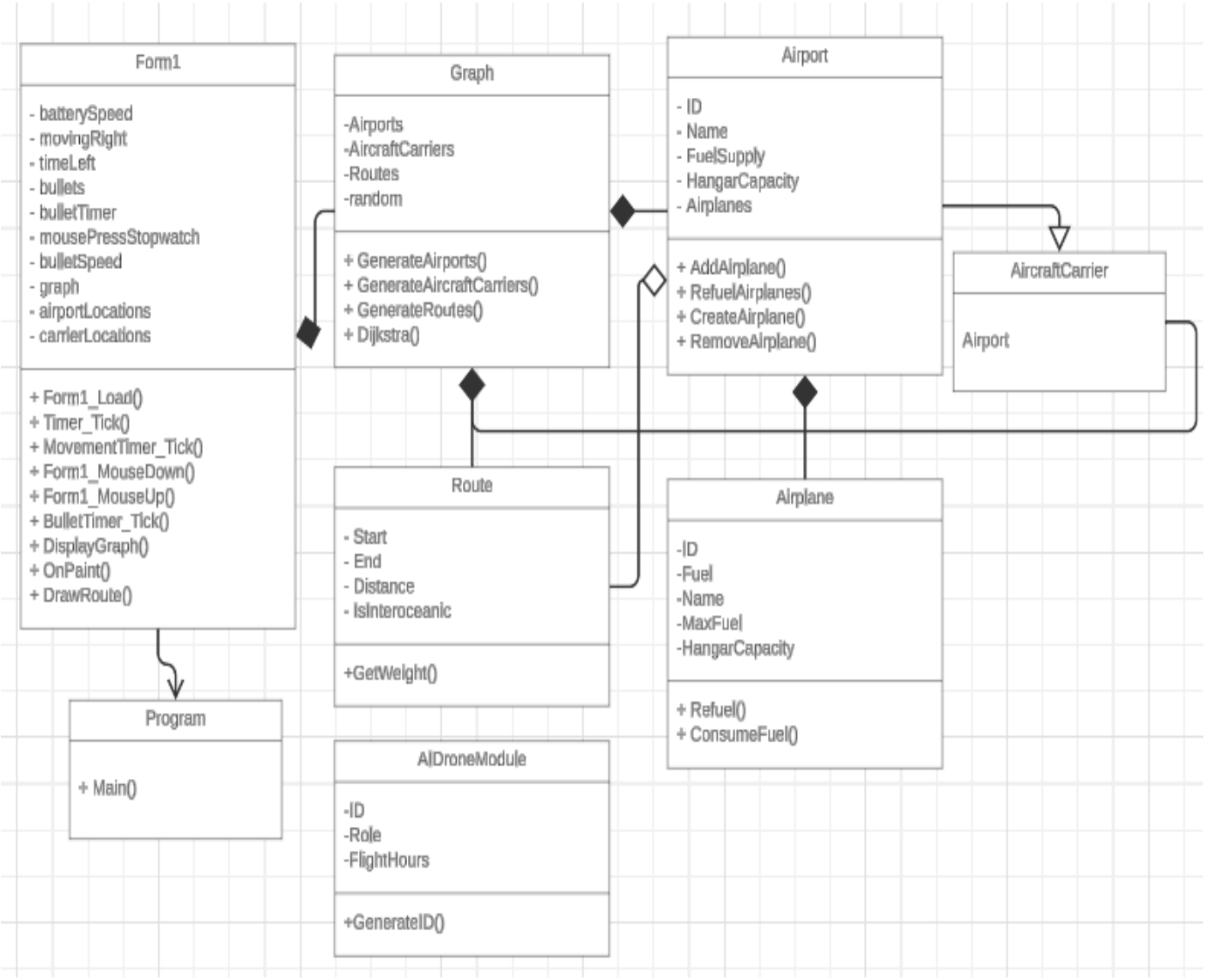

Figure 1. Diagrama UML del proyecto

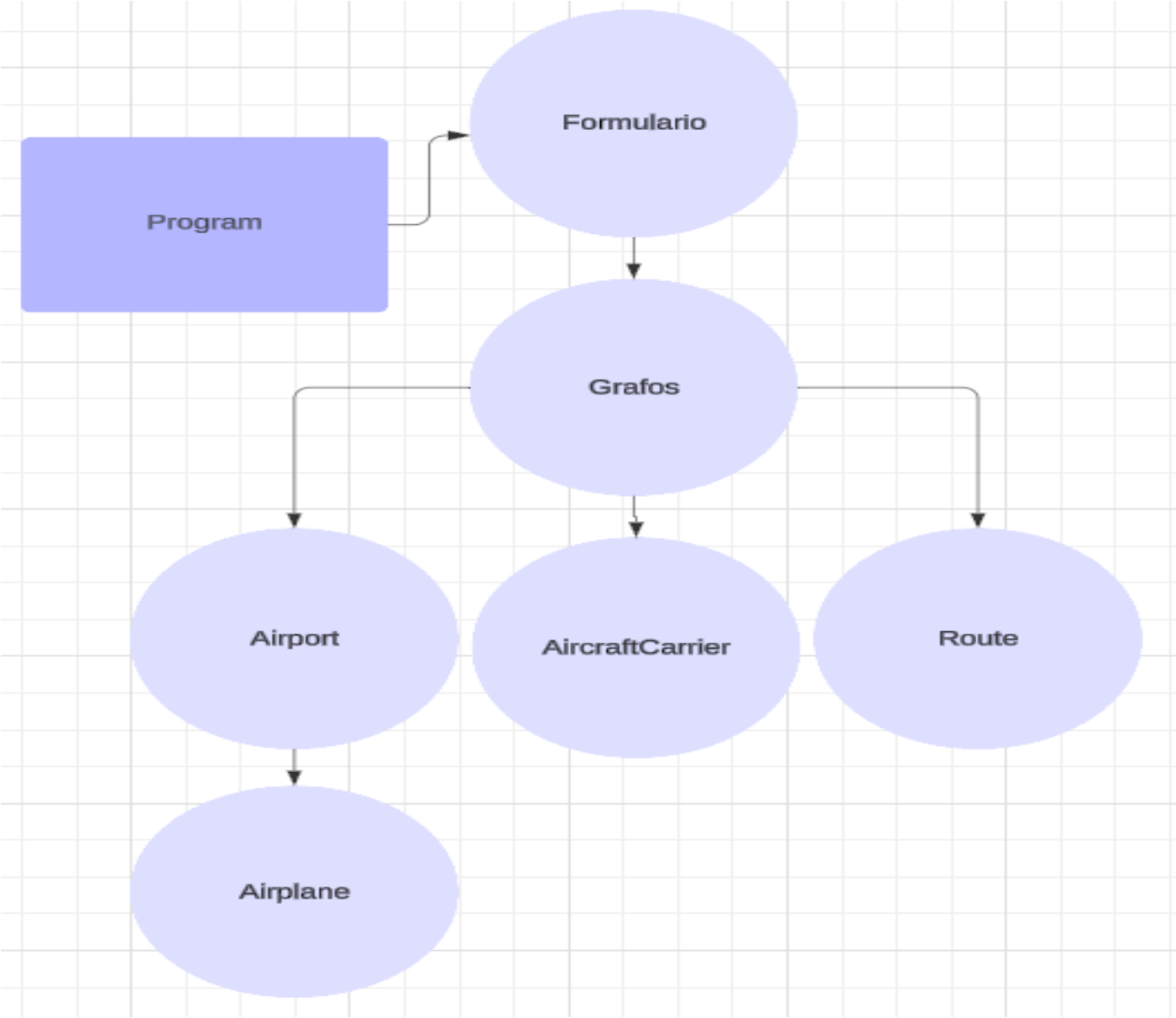

Figure 2. Diagrama de arquitectura del proyecto