Instituto Tecnológico de Costa Rica

Escuela de Computadores

CE 1103: Algoritmos y Estructuras de Datos I

Proyecto Programado AirWar

Estudiantes

Guillermo Sánchez Cedeño

Luis Castro Montenegro

Profesor

Leonardo Araya Martínez

Índice

1 INTRODUCCIÓN

En este proyecto, se desarrollará un videojuego utilizando el lenguaje de programación C# y aplicando los principios del Paradigma Orientado a Objetos (POO). El objetivo principal es implementar una solución que integre estructuras de datos eficientes y algoritmos de búsqueda y ordenamiento, los cuales son fundamentales para resolver los retos planteados en la especificación del problema.

Este videojuego no solo busca ser una aplicación interactiva, sino también una plataforma para demostrar habilidades técnicas, como la modelación de problemas utilizando POO y el diseño de soluciones optimizadas mediante algoritmos clásicos de ordenamiento (como Merge Sort y Selection Sort) y estructuras de datos como grafos. La aleatoriedad en la generación de rutas y decisiones introduce un dinamismo que hará cada partida única, aumentando el desafío tanto en el diseño del juego como en su jugabilidad.

Además, el proyecto incluirá una documentación exhaustiva para garantizar la trazabilidad y claridad del desarrollo. Esto se logrará mediante el uso de estándares de documentación técnica, que detallarán tanto el diseño como la implementación de cada uno de los componentes. Las herramientas de gestión de proyectos también serán empleadas para mantener el control de las tareas y la organización del equipo durante el proceso de desarrollo.

A través de este proyecto, se busca no solo cumplir con los objetivos propuestos, sino también fomentar un aprendizaje integral en la aplicación práctica de técnicas avanzadas de programación, diseño y documentación, consolidando habilidades clave para el desarrollo de software orientado a objetos.

2 Diseño

Esta sección como objetivo presentar los elementos clave del desarrollo del videojuego. En esta etapa, se desglosan los requerimientos del proyecto en historias de usuario, se analizan problemas críticos del diseño proponiendo múltiples alternativas de solución, y se seleccionan las más adecuadas según sus ventajas y desventajas. Además, se incluyen diagramas de clases y de arquitectura. Estos diagramas permitirán visualizar cómo se organizan y comunican los elementos del software. Finalmente, se presenta un *checklist* de las historias de usuario implementadas, destacando aquellas que se completaron con éxito y las que quedaron pendientes.

2.1 Historias de usuario

Con los requerimientos del proyecto como base se formulan las historias de usuario que se necesitan satisfacer mediante la implementación de este juego.

Historia 01	
Historia de usuario:	Como jugador, quiero destruir la mayor cantidad de aviones en un periodo de tiempo, para alcanzar la máxima puntuación posible y ganar el juego.
Criterios de aceptación	
• El juego debe iniciar con un temporizador visible que indique el tiempo restante para destruir aviones. • Cada avión destruido debe sumar al puntaje y reflejarse en la interfaz. • Al finalizar el tiempo, el juego debe detenerse automáticamente. • El jugador debe tener una forma de disparar a los aviones.	

Historia 02	
Historia de usuario:	Como jugador, quiero que los aeropuertos, portaaviones y rutas se generen de forma aleatoria, para garantizar una experiencia dinámica en cada partida.
Criterios de aceptación	
• Los aeropuertos, portaaviones deben generarse en ubicaciones diferentes al inicio de cada partida, dentro de zonas específicas. • Se deben generar rutas para cada aeropuerto y portaaviones para que estén interconectados.	

Historia 03	
Historia de usuario:	Como jugador, quiero que la batería antiaérea se mueva de forma constante entre izquierda y derecha de la pantalla, para poder calcular mejor mis disparos.
Criterios de aceptación	
• La batería antiaérea debe moverse continuamente de izquierda a derecha y viceversa dentro de los límites de la pantalla.	

Historia 04	
Historia de usuario:	Como jugador, quiero disparar con trayectorias rectas y velocidades variables dependiendo del tiempo que presione el clic, para controlar la efectividad de los disparos.
Criterios de aceptación	
• Los disparos deben seguir una trayectoria recta desde el punto de origen hasta salir de la pantalla o impactar un objetivo. • La velocidad del disparo debe aumentar proporcionalmente al tiempo que el jugador mantenga presionado el botón del clic. • Debe haber un límite superior en la velocidad para evitar disparos excesivamente rápidos. • Si el clic se libera inmediatamente, el disparo debe tener una velocidad mínima predeterminada.	

Historia 05	
Historia de usuario:	Como jugador, quiero que mi avión elija aleatoriamente un destino y tenga que calcular la mejor ruta para llegar a él, considerando factores como la distancia y los costos, para que mi experiencia de juego sea más estratégica y desafiante.
Criterios de aceptación	
• El avión debe elegir aleatoriamente un destino dentro de las rutas disponibles al inicio de cada viaje. • El juego debe calcular la ruta óptima hacia el destino seleccionado, tomando en cuenta los costos asociados a cada ruta.	

Historia 06	
Historia de usuario:	Como jugador, quiero que los aeropuertos racionen el combustible de alguna manera lógica, para aumentar el desafío del juego y evitar caídas de aviones debido a falta de combustible.
Criterios de aceptación	
• Los aeropuertos deben tener una cantidad limitada de combustible disponible para reabastecer a los aviones • El reabastecimiento de combustible debe ser proporcional a la capacidad de cada aeropuerto y la demanda de los aviones que llegan. • Los aviones no deben ser capaces de reabastecerse completamente si el aeropuerto no tiene suficiente combustible disponible. • Si un avión se queda sin combustible debe destruirse.	

Historia 07	
Historia de usuario:	Como jugador, quiero que los aeropuertos generen nuevos aviones de forma periódica, respetando la capacidad de sus hangares, para que siempre haya aviones disponibles para disparar dentro del tiempo.
Criterios de aceptación	
• Los aeropuertos deben generar aviones a intervalos regulares, asegurando que siempre haya nuevos aviones disponibles para que el jugador pueda disparar durante el juego. • Los hangares de los aeropuertos deben tener una capacidad máxima de aviones. • Si el hangar está lleno, no se generarán nuevos aviones hasta que se liberen espacios.	

Historia 08	
Historia de usuario:	Como jugador, quiero que los aviones tengan un identificador único, para poder identificarlos al final de la partida.
Criterios de aceptación	
• Cada avión en el juego debe tener un identificador único. • Al final de la partida, el jugador debe poder ver un resumen de los aviones destruidos o involucrados en su desempeño, junto con sus identificadores.	

Historia 08	
Historia de usuario:	Como jugador, quiero que cada avión autónomo cuente con cuatro módulos de inteligencia artificial (Pilot, Copilot, Maintenance y Space Awareness) que colaboren entre sí, para garantizar vuelos seguros y un desafío dinámico durante la partida.
Criterios de aceptación	
• Cada avión autónomo debe tener los cuatro módulos (Pilot, Copilot, Maintenance, Space Awareness) configurados y funcionando en conjunto para asegurar el vuelo. • Cada módulo debe tener un identificador único compuesto por tres letras generadas aleatoriamente al momento de crear el avión. • El jugador debe poder visualizar los identificadores de los módulos de los aviones derribados al final de la partida. • Cada módulo debe acumular y registrar las horas de vuelo durante la partida.	

Historia 09	
Historia de usuario:	Como jugador, quiero obtener una lista de los aviones derribados ordenados por su identificador, para analizar mi desempeño y revisar detalles de su tripulación.
Criterios de aceptación	
• El jugador debe poder ver una lista de los aviones derribados al final de la partida. • Los aviones derribados deben estar ordenados alfabéticamente por su identificador único en la lista presentada al jugador. • El jugador debe poder elegir otras opciones de ordenación si lo desea: Por rol, por horas de vuelo acumuladas.	

Historia 10	
Historia de usuario:	Como jugador, quiero ver un resumen al final de la partida con toda la información relevante del juego, incluyendo rutas, pesos de rutas, atributos de los aviones y sus datos de vuelo, para consultar fácilmente los detalles.
Criterios de aceptación	
• El resumen debe incluir las rutas recorridas por los aviones durante la partida, mostrando: El peso asociado a cada ruta, el destino y origen de cada ruta. • Cada avión debe mostrar: Identificador único, estado final (derribado, en vuelo, estacionado), atributos acumulados, como distancia recorrida y tiempo total de vuelo. • Para cada avión, deben listarse los módulos de AI indicando: Identificador único del módulo, rol asignado, horas de vuelo acumuladas por cada módulo.	

2.2 Solución a problemas

En este apartado se seleccionan cinco problemas abarcados en el desarrollo del juego, los cuales pueden tener más de una solución. Se darán dos alternativas para cada uno, indicando sus ventajas y desventajas, se selecciona una y se explica el porqué se tomó esa decisión.

Problema:	Calcular la mejor ruta de vuelo.
Alternativa 1	Alternativa 2
Dijkstra es un algoritmo que permite encontrar la mejor ruta, en un grafo, de un nodo hacia otro. Su complejidad es O(V log V + E) lo que lo hace eficiente para solo encontrar la mejor ruta, en especial para grafos dispersos. Solo aplica para calcular la mejor ruta de un nodo a otro, es decir no calcula la mejor ruta para cada nodo.	Floyd-Warshall calcula las distancias más cortas entre todos los nodos del grafo. Su complejidad es $O(V^3)$ la cual se justifica más con grafos densos. Requiere más recursos pues tiene que calcular la ruta para todos los pesos y se debería almacenar esas rutas para accederlas y no tener que volver a ejecutar el algoritmo.
Selección:	Dado que el grafo no es muy grande y solo se necesita la mejor ruta se decidió utilizar Dijkstra (alternativa 1), además cada avión decide a que nodo va aleatoriamente por lo que es necesario ejecutar el algoritmo múltiples veces, lo cual con Floyd-Warshall se usarían más recursos.

Problema:	Determinación de pesos de las rutas
Alternativa 1	Alternativa 2
La primera alternativa sería implementar una función propia que calcule el peso de las rutas en función de parámetros como la distancia, tipo de origen, tipo de destino y tipo de ruta. Esta solución sería totalmente controlada y ajustable.	Utilizar una librería como GraphX o QuickGraph que se especializa en estructuras de datos de grafos. Estas librerías permiten representar rutas como vértices y aristas, y calcular pesos con facilidad. Puede requerir tiempo para comprender como funcionan y genera una dependencia.
Selección:	Debido a la simplicidad de la lógica de los pesos se opto por la alternativa 1 pues se creó una función bastante simple que permite obtener los pesos sin necesidad de agregar dependencias que podría afectar el rendimiento del juego.

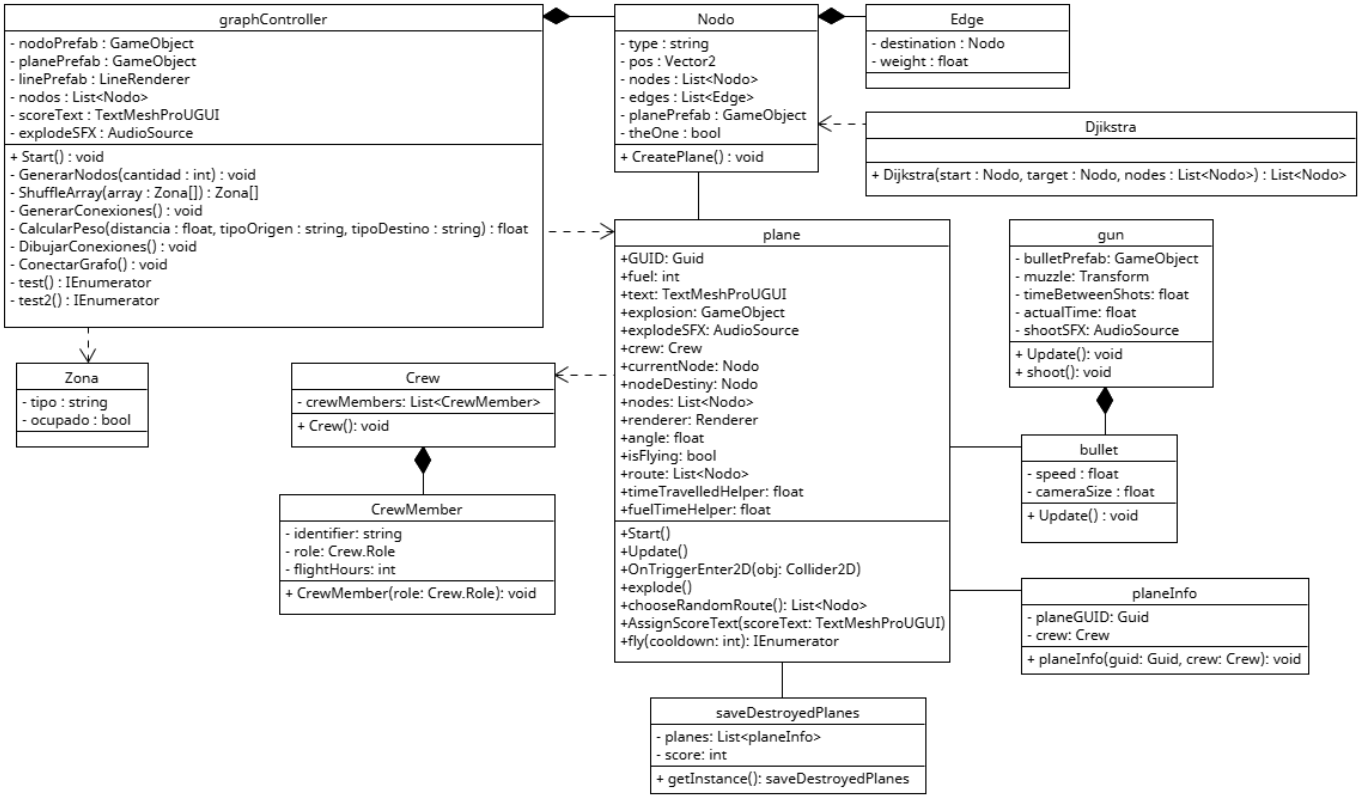
Problema:	Elección de ambiente de desarrollo	
Alternativa 1		Alternativa 2
Unity es una de las opciones para desarrollar el juego. Este cuenta con las herramientas necesarias para desarrollar el juego, aunque al ser un juego pequeño puede añadir complejidad ya que cuenta con una gran cantidad de elementos los cuales no son aplicables o no se necesitan. Tiene una gran comunidad entonces es fácil encontrar soluciones a problemas.		Si bien no esta dentro de los requerimientos, teníamos la libertad de elegir otra opción, MonoGame es un framework para crear videojuegos que permite desacoplar la lógica del juego con los elementos de la interfaz.
Selección:	Se seleccionó la alternativa 1 por experiencias pasadas de miembros del equipo trabajando con ella y por su facilidad de encontrar soluciones a problemas e información que ayuda al desarrollo.	

Problema:	Generación de GUID	
Alternativa 1		Alternativa 2
Utilizar la clase System.Guid de C# para generar identificadores únicos. Guid.NewGuid() genera un nuevo identificador único cada vez que se llama, lo que garantiza que cada avión tendrá un ID único en el juego. Esta clase ya se encuentra por defecto disponible.		Usar un contador numérico que se incremente cada vez que se crea un nuevo avión. Este enfoque genera un identificador único en el sentido de que cada nuevo avión tiene un número único, pero no es tan robusto como los GUIDs en cuanto a unicidad global. Debe ser programado por el estudiante.
Selección:	En este caso, la alternativa 1 es la opción más adecuada para la generación de identificadores únicos para los aviones. Esto se debe a su alta probabilidad de unicidad global y su facilidad de implementación, ya que System.Guid es parte de la biblioteca estándar de C# y está optimizado para generar identificadores únicos.	

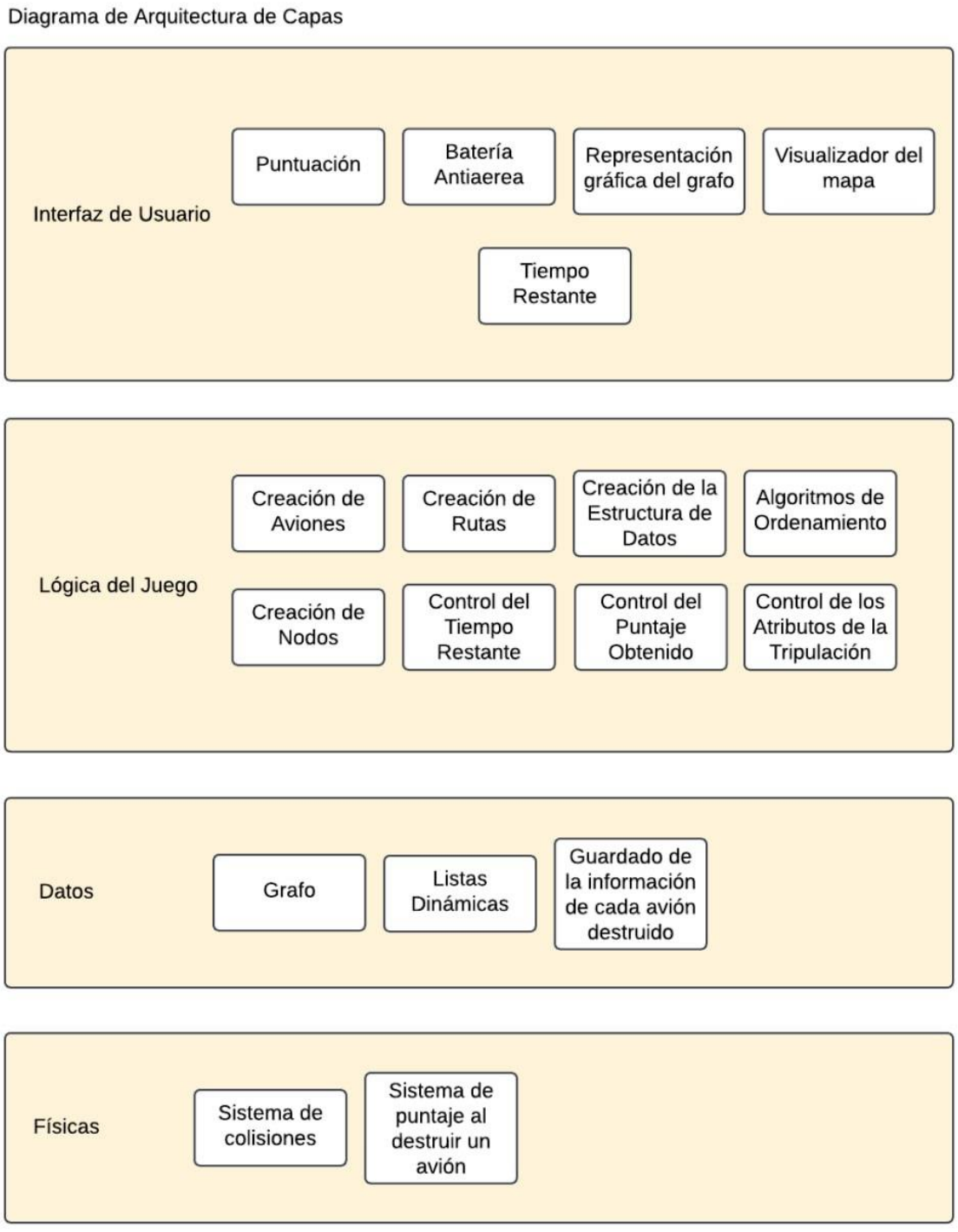
Problema:	Guardar la información del juego	
Alternativa 1		Alternativa 2
Guardar los datos del avión y su tripulación en un archivo para luego poder consultarla. Es implica tener que crear una interfaz para interactuar con el archivo y poder guardar y obtener los datos. Permite conservar la información de cada partida.		Utilizar una clase que permita guardar en tiempo de ejecución los datos del avión y su tripulación en una instancia de la clase. Esto permitirá acceder a los datos al finalizar la partida, pero no se contará con un registro de las distintas partidas una vez que se cierre la aplicación o se inicie otra partida.
Selección:	Por simplicidad, y siguiendo los requerimientos del proyecto se seleccionó la alternativa 2, la cual permitirá mostrar los datos del avión y su tripulación. Si bien no se podrá consultar los datos de cada partida posteriormente el requerimiento es solo mostrarlo al terminar la partida entonces se decidió no guardar los datos para su futura consulta.	

2.3 Diagrama de clases

A continuación se presenta el diagrama de clases que representa la implementación del juego Airwar para este proyecto.



2.4 Diagrama de arquitectura



2.5 Checklist

A continuación, se presenta un listado de las historias de usuario identificadas para este proyecto y su estado actual.

Historia de Usuario	Estado
Historia 01	Completado
Historia 02	Completado
Historia 03	Completado
Historia 04	Incompleto
Historia 05	Completado
Historia 06	Completado
Historia 07	Completado
Historia 08	Completado
Historia 09	Completado
Historia 10	Completado