



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería Informática

**SIMULADOR DE CONTROL DE TRÁFICO AÉREO EN
POSICIÓN DE TORRE COMO ENTRENADOR PARA LA RED DE
VUELO IVAO**

ÁLVARO BRAVO PANADERO

Dirigido por: ALFONSO URQUÍA MORALEDA

Codirigido por CARLA MARTÍN VILLALBA

Curso: 2024/2025, convocatoria SEPTIEMBRE



SIMULADOR DE CONTROL DE TRÁFICO AÉREO EN POSICIÓN DE TORRE COMO ENTRENADOR PARA LA RED DE VUELO IVAO

**Proyecto de Fin de Grado en Ingeniería Informática
de modalidad específica**

ÁLVARO BRAVO PANADERO

Dirigido por: ALFONSO URQUÍA MORALEDA

Codirigido por CARLA MARTÍN VILLALBA

Fecha de lectura y defensa: 6 DE OCTUBRE DE 2025

Resumen

La proliferación de los simuladores de vuelo en la década de 1990 dio lugar a la creación de redes de vuelo a las que los usuarios conectan los simuladores para simular la operativa aérea real, entre las que se encuentra IVAO. Dentro de la operativa aérea, el servicio de control de tráfico aéreo es un elemento fundamental encargado de velar por la seguridad y el flujo eficiente de aeronaves en el espacio aéreo, siendo el Servicio de control de aeródromo el encargado de gestionar el movimiento de las aeronaves en el entorno aeroportuario desde la torre de control.

Los usuarios de IVAO que desean conectarse como controladores en una torre capaz de ofrecer servicios de control de aeródromo deben disponer de los conocimientos necesarios para desempeñar su función de forma segura y eficaz. El problema que encuentran la mayoría de usuarios es la curva de dificultad inicial que supone iniciarse en el control aéreo.

Este Proyecto Fin de Grado se centra en la implementación de una aplicación capaz de simular el movimiento típico de aeronaves que un usuario de la red puede encontrar en el entorno aeroportuario con el fin de que pueda entrenar la operativa básica del control de tráfico aéreo a nivel de torre. La aplicación, desarrollada en Java, posee una interfaz gráfica implementada en JavaFX. El usuario puede añadir sus propios aeropuertos y modelos de avión que desee incluir en el simulador mediante una estructura modular de archivos JSON. Además de mostrar información de la sesión en curso, presenta una pantalla radar interactiva donde el usuario puede visualizar una representación del aeródromo y las aeronaves, con la capacidad de seleccionar la aeronave a la que quiere enviar instrucciones. Los datos más relevantes de la sesión son almacenados para su posterior consulta en una sección dedicada para ello.

Palabras clave

ATC, IVAO, Aeropuerto, Control de tráfico aéreo, Control de aeródromo, Simulador, Aviación, Java, JavaFX, Ingeniería del Software.

Abstract

The proliferation of flight simulators led to the creation of online flight networks where users can connect in order to simulate real-world air operations. Within air operations, the air traffic control service (ATC) is a key element responsible for ensuring the safety and efficient usage of the airspace. The Aerodrome Control Service is responsible for managing aircraft in the airport environment from the airport control tower.

Users who wish to connect as controllers at a tower position capable of providing aerodrome control services must possess the necessary knowledge to perform their role safely and effectively. The most common issue is the initial learning curve involved in getting started with ATC on the network.

This project focuses on implementing an application capable of simulating the typical airport environment that users may encounter, enabling them to practise basic tower air traffic control operations. The application, developed in Java, has a graphical interface implemented in JavaFX. Users can add their own airports and aircraft models to the simulator using a modular JSON file structure. In addition to displaying information about the current session, it presents an interactive radar screen where users can view a representation of the airfield and aircraft, with the ability to select the aircraft to which they want to send instructions. Relevant session data is stored for later reference using a dedicated section.

Keywords

ATC, IVAO, Airport, Air Traffic Control, Aerodrome control, Simulator, Aviation, Java, JavaFX, Software Engineering.

Índice

Índice.....	I
Índice de figuras.....	V
Índice de tablas.....	VII
Capítulo 1. Introducción, objetivos y estructura.....	9
1.1 Introducción.....	9
1.2 Objetivos.....	10
1.3 Estructura.....	12
Capítulo 2. Estado del arte y marco teórico.....	13
2.1 Introducción.....	13
2.2 Marco teórico del problema.....	14
2.2.1. International Virtual Aviation Organization (IVAO).....	14
2.2.2. Tipos de reglas de vuelo.....	15
2.2.3. Control básico IFR.....	16
2.2.4. Control básico VFR.....	19
2.3 Análisis tecnológico.....	22
2.4 Aplicaciones similares existentes.....	23
2.4.1. Endless ATC.....	23
2.4.2. VoiceATC Simulator.....	24
2.4.3. OpenScope.....	24
2.4.4. Tower! Simulator 3.....	24
2.5 Conclusiones.....	25
Capítulo 3. Análisis.....	27
3.1 Introducción.....	27

3.2 Modelo de negocio.....	27
3.2.1 Clases conceptuales.....	27
3.2.2 Diagrama de modelo de dominio	29
3.3 Análisis de requisitos.....	29
3.3.1. Análisis de requisitos de usuario	29
3.3.2. Análisis de requisitos del sistema	30
3.3.3 Casos de uso	33
3.4 Entidades principales.....	37
3.5 Conclusiones.....	41
Capítulo 4. Diseño	43
4.1 Introducción	43
4.2. Diseño Arquitectónico	43
4.3. Patrones de diseño utilizados.....	44
4.3.1. Modelo-Vista-Controlador	44
4.3.2. Observer	44
4.3.3. Strategy/Provider.....	45
4.3.4. Otras consideraciones.....	46
4.4. Diseño del agente	46
4.5. Modelo de datos	57
4.6 Conclusiones.....	62
Capítulo 5. Implementación y pruebas	63
5.1 Introducción	63
5.2 Entorno de desarrollo.....	63
5.2.1. Entorno y librerías.....	63
5.2.2. Gestión de dependencias en Maven	64

Índice

5.3 Implementación	65
5.3.1 Organización del código fuente	65
5.3.2. Implementación de requisitos funcionales	68
5.4. Flujo de desarrollo del proyecto.....	78
5.4 Pruebas.....	81
5.4.1 Pruebas unitarias	81
5.4.2 Pruebas de integración	82
5.4.3 Pruebas de validación	82
5.5 Conclusiones.....	89
Capítulo 6. Planificación	91
6.1 Introducción	91
6.2 Planificación temporal.....	91
6.2.1 Fases del proyecto	91
6.2.2. Metodología de desarrollo	92
6.2.3. Backlog del proyecto.....	93
6.3. Conclusiones.....	94
Capítulo 7. Conclusiones y trabajos futuros	95
7.1 Introducción	95
7.2 Conclusiones.....	95
7.3 Trabajos futuros.....	97
Bibliografía	99
Glosario	103
Anexo A – Manual de usuario	105
A.1 Introducción	105
A.2 Estructura de la aplicación	105

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

A.3 Menú principal	105
A.5 Cómo añadir nuevos modelos de aeronave	112
A.6 Cómo añadir nuevos aeropuertos.....	113
A.7 Consejos	124
Anexo B – Manual del programador	127
B.1 Introducción	127
B.2 Entorno y librerías utilizadas	127
B.3 Código fuente	128
B.4 Documentación	129

Índice de figuras

Figura 2.1. Algunas de las restricciones de conexión a posiciones ATC de aeródromo por rango	13
Figura 2.2. Ejemplo de ruta de rodaje en el aeropuerto de Sevilla. Carta aeronáutica obtenida de https://aip.enaire.es/AIP/	17
Figura 2.3. Barra de parada del punto de espera HP3 en Sevilla	18
Figura 2.4 Ejemplo de ruta de salida que utilizaría una aeronave bajo reglas VFR. Carta aeronáutica obtenida de https://aip.enaire.es/AIP/	20
Figura 2.5. Ejemplo de ruta de llegada que utilizaría una aeronave bajo reglas VFR. Carta aeronáutica obtenida de https://aip.enaire.es/AIP/	20
Figura 2.6. Ejemplo de circuito de tránsito aéreo al sur en el aeropuerto de Sevilla	21
Figura 2.7. Tramos del circuito de tránsito aéreo. Imagen tomada de https://wiki.es.iva0.aero/books/alumno-atc-avanzado-as3/page/control-basico-vfr	22
Figura 2.8. Interfaz gráfica de Aurora	23
Figura 3.1 Diagrama de modelo de dominio	29
Figura 3.2 Diagrama de casos de uso principal	34
Figura 4.1. Representación del Modelo-Vista-Controlador	43
Figura 4.2. Diagrama de transición de estados del agente	56
Figura 4.3. Representación del orden de numeración de los vértices de la pista	61
Figura 5.1. Estructura de paquetes y clases	67
Figura 5.2. Estructura de paquetes y clases (continuación)	67
Figura 5.3. Menú principal de la aplicación	84
Figura 5.4. Ventana de simulación	84
Figura 5.5. Aspecto de la aplicación tras iniciar una sesión de entrenamiento	85
Figura 5.6. Selección de la ruta de rodaje	86
Figura 5.7. Dos aeronaves rodando a la pista	86
Figura 5.8. Dos aeronaves esperando para despegar mientras otra rueda hacia la pista y otra rueda al estacionamiento	87
Figura 5.9. El usuario autorizó el despegue sin dar suficiente distancia de seguridad, considerándose un despegue simultáneo de dos aeronaves a la vez	88

Figura 5.10. Dos aeronaves realizando circuitos de tránsito aéreo mientras una aeronave se dirige a la pista para despegar y otra se aproxima para aterrizar	88
Figura 5.11. Ventana de estadísticas de las sesiones realizadas. En la parte inferior se muestran los fallos cometidos en la sesión seleccionada.....	89
Figura A.1. Menú principal de la aplicación	106
Figura A.2. Vista de sesión de entrenamiento	107
Figura A.3. Mensaje de advertencia al cerrar la aplicación	108
Figura A.4. Presentación de la pantalla radar durante una sesión de entrenamiento	108
Figura A.5. Ventana para seleccionar la ruta de rodaje	110
Figura A.6. Ventana de estadísticas de sesiones realizadas	111
Figura A.7. Representación del orden de numeración de los vértices de la pista	118

Índice de tablas

Tabla 3.1. Caso de uso 1.1: Iniciar entrenamiento ATC.....	34
Tabla 3.2. Caso de uso 1.2: Revisar estadísticas de entrenamientos realizados.....	35
Tabla 3.3. Caso de uso 2.1: Selección de una aeronave en la pantalla radar.....	35
Tabla 3.4. Caso de uso 2.2: Instruir a una aeronave a realizar una acción.....	36
Tabla 3.5. Caso de uso 2.3: Seleccionar y enviar la ruta de rodaje a una aeronave.....	37
Tabla 4.1. Matriz de transición de estados de la máquina de estados finitos del agente.....	54
Tabla 5.1. Pruebas de validación del sistema.....	83

Capítulo 1. Introducción, objetivos y estructura

1.1 Introducción

Los simuladores son una herramienta fundamental de entrenamiento muy utilizada hoy en día, permiten hacer de puente entre los conocimientos teóricos y la entidad real que se está simulando. En su formación un piloto de aerolínea debe pasar por un simulador que replique las condiciones que se va a encontrar en el avión real antes de poder volarlo. En los simuladores se hace énfasis en los sistemas más interesantes e importantes del entorno que se desea simular.

Los simuladores profesionales tienen un elevado precio de adquisición y mantenimiento. Por otro lado, hay un tipo de simuladores que han dado lugar a toda una familia que cualquier persona puede utilizar en su casa, no tienen tanta calidad ni son tan precisos como los simuladores profesionales, pero son baratos y accesibles. Entre ellos están los simuladores de vuelo. Al proliferar estos e Internet, los usuarios comenzaron a crear redes donde conectarse y poder volar juntos siguiendo las mismas normas y operativas que se siguen en la realidad hasta donde sea posible. De esta forma nacieron las redes de vuelo siendo una de ellas IVAO (International Virtual Aviation Organization) que basa su nombre en el organismo real ICAO (International Civil Aviation Organization).

Los aviones en la realidad no vuelan descoordinados entre sí, alrededor de aeropuertos y zonas con gran cantidad de tráfico es necesaria la figura de un coordinador que garantice una operativa segura, ordenada y proporcione servicios a las aeronaves en su espacio aéreo, estos son los controladores de tráfico aéreo o ATC por sus siglas en inglés.

En las redes de vuelo hay usuarios que se conectan con el rol de ATC y para ello disponen de un software de control aéreo. Dentro de los diferentes espacios aéreos que puede gestionar un ATC se encuentra el espacio aéreo del entorno aeroportuario junto con la rodadura (movimiento de las aeronaves en tierra, generalmente desde su estacionamiento hasta la pista y viceversa) [1]. Es por ello que resulta de interés para los usuarios de las redes que existan simuladores que permitan practicar la operativa con aeronaves simuladas fuera de la red de vuelo.

1.2 Objetivos

El principal objetivo del proyecto consiste en la creación de un simulador de control de tráfico aéreo a nivel de torre. Los objetivos individuales del proyecto son los siguientes:

- Desarrollo de un entrenador que pueda ser de utilidad a nuevos usuarios de la red de vuelo para aprender y practicar el control de tráfico aéreo en posición de torre/rodadura.
- Simulación del espacio aéreo controlado típico alrededor de un aeropuerto que admita aeronaves bajo reglas visuales (VFR).
- Implementación de un conjunto de reglas que marcan las acciones que puede realizar un piloto y las instrucciones o servicios que puede dar un controlador.
- Implementación de una GUI que represente los elementos más importantes del aeropuerto: el espacio aéreo, las pistas, la rodadura del aeropuerto, las barras de parada, la plataforma de estacionamientos y los estacionamientos.
- Implementación de una base de datos con información sobre aeronaves, aeródromos y estadísticas anónimas de las sesiones de entrenamiento realizadas.
- Proporcionar una arquitectura capaz de permitir la expansión del software permitiendo ejecutar el entrenamiento en otros aeródromos añadidos con posterioridad sin necesidad de que el usuario disponga del código fuente de la aplicación.

Además, el código fuente debe poseer la mayor genericidad posible para facilitar su ampliación tras la finalización del proyecto sin comprometer el rendimiento, debe ser capaz de funcionar de forma fluida sin una gran necesidad de consumo de recursos de CPU y memoria. Esto es clave en la implementación de los aeropuertos y modelos de avión, el tratamiento de los datos debe ser lo suficientemente genérico para poder funcionar con cualquier aeródromo que el usuario quiera añadir a la base de datos.

Para la consecución de estos objetivos se han propuesto las siguientes tareas:

1. Estudio del contexto del problema para decidir qué es necesario tener en cuenta para la realización del proyecto priorizando los elementos encontrados por importancia.
2. Estudio y valoración de las tecnologías necesarias para la implementación de los diferentes componentes que se necesitarán en el sistema final.

Capítulo 1. Introducción, objetivos y estructura

3. Análisis sobre qué elementos software son necesarios en la aplicación con el fin de representar los diferentes aspectos descubiertos en el estudio del contexto del problema.
4. Diseño de las diferentes partes del sistema apoyándose en el modelado.
5. Comprobación de que el diseño cumple los requisitos y objetivos propuestos.
6. Construcción de la aplicación utilizando el análisis y el diseño realizado.
7. Pruebas y evaluación de la aplicación comprobando que el sistema se ajusta a los requisitos y objetivos propuestos.
8. Elaboración de la memoria y documentación final del proyecto.

La metodología de desarrollo a emplear, debido a las características propias de un Proyecto Fin de Grado de elaboración individual, sigue un ciclo de vida del software en cascada tratando de aplicar metodologías ágiles donde sea práctico. El desarrollo comienza con el análisis encargado de especificar el software, luego se pasa al diseño arquitectónico y al diseño detallado, en estos pasos se crea un backlog que, junto al resto de ítems creados durante el análisis y diseño, sirven de guía durante el desarrollo y las pruebas.

En la fase de codificación se adaptó un enfoque iterativo basado en metodologías ágiles. En cada iteración se propone la implementación de una nueva funcionalidad o cambio sobre el software existente siguiendo los siguientes pasos:

- Paso 1: Se decide de qué cambios se quieren aplicar al proyecto en el ciclo actual.
- Paso 2: Se realiza una sesión para considerar qué es necesario realizar o modificar para efectuar esos cambios (clases implicadas, conocimiento sobre la forma de aplicar los cambios, ...), y cómo se van a realizar y comprobar que los cambios se han realizado correctamente revisando los ítems producidos en la fase de análisis y diseño, actualizándolos si es necesario.
- Paso 3: Implementación de los cambios.
- Paso 4: Realización de test unitarios y de integración.
- Paso 5: Corrección de errores y problemas encontrados en los test.
- Paso 6: Satisfecho con el resultado, se documentan los cambios en un cuaderno de desarrollo explicando los cambios efectuados, decisiones tomadas y otros ítems que se deban tener en cuenta en el futuro.
- Paso 7: Actualización del backlog.

Tras el desarrollo se realizan las pruebas de validación y se elabora la documentación final utilizando los ítems creados durante el desarrollo: análisis, diseño y notas realizadas durante el desarrollo.

1.3 Estructura

La memoria del trabajo se ha organizado en los siguientes apartados:

- **Capítulo 1, Introducción, objetivos y estructura:** Presenta una breve descripción del problema que aborda este trabajo, los objetivos propuestos y la estructura de la memoria.
- **Capítulo 2, Estado del arte y marco teórico:** Introduce una descripción del marco teórico en el que se engloba la aplicación, un análisis a las tecnologías utilizadas en el proyecto y un repaso a aplicaciones disponibles similares a la que se va a realizar.
- **Capítulo 3, Análisis:** Presenta el análisis del software a desarrollar, cubriendo el modelado de negocio y el análisis de requisitos, describiendo qué hay que realizar para resolver el problema, pero no cómo.
- **Capítulo 4, Diseño:** Aborda diseño de la aplicación indicando cómo se va a resolver el problema, algunos de los patrones de diseño utilizados, el diseño del agente y el diseño del modelo de datos.
- **Capítulo 5, Implementación y pruebas:** Describe el entorno de desarrollo, cómo se implementó el sistema y la cobertura de los requisitos funcionales, el flujo de desarrollo del proyecto y las pruebas realizadas.
- **Capítulo 6, Planificación:** Cubre la planificación temporal del proyecto y metodología utilizada para la realización.
- **Capítulo 7, Conclusiones y trabajos futuros:** Contiene las conclusiones alcanzadas al finalizar la realización del proyecto y las posibles líneas de trabajo con las que se podría continuar el desarrollo en el futuro.
- **Anexo A, Manual de usuario:** Explica cómo utilizar la aplicación desde el punto de vista del usuario final.
- **Anexo B, Manual del programador:** Explica los aspectos más importantes para continuar con el desarrollo de la aplicación.

Capítulo 2. Estado del arte y marco teórico

2.1 Introducción

En este capítulo exploraremos los conocimientos teóricos necesarios para entender el problema que se ha abordado en el proyecto.

La curva de aprendizaje en las redes de vuelo es bastante pronunciada al inicio, hay muchos conocimientos que adquirir, entender y es necesario que otros usuarios que tomen el rol de piloto o controlador para poder practicar lo aprendido en la teoría. Como muestra la figura 2.1, los usuarios nuevos que desean comenzar a ofrecer servicios de tráfico aéreo sólo pueden conectarse a la red en un conjunto limitado de aeropuertos pequeños y medianos, sólo controlar tráficos en tierra y en el espacio aéreo del aeropuerto [2].

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	
LEAB_APP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	
LEAB_TWR	AS3	AS3	AS3	AS3	AS3	AS3	AS3	
LEAL_APP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	
LEAL_GND	AS1	AS1	AS1	AS1	AS1	AS1	AS1	
LEAL_TWR	AS1	AS1	AS1	AS1	AS1	AS1	AS1	
LEAM_TWR	AS3	AS3	AS3	AS3	AS3	AS3	AS3	
LEAS_GND	AS3	AS3	AS3	AS3	AS3	AS3	AS3	
LEAS_TWR	AS3	AS3	AS3	AS3	AS3	AS3	AS3	
LEBA_FIS_TWR	AS1	AS1	AS1	AS1	AS1	AS1	AS1	
LEBB_APP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	
LEBB_GND	AS3	AS3	AS3	AS3	AS3	AS3	AS3	
LEBB_TWR	AS3	AS3	AS3	AS3	AS3	AS3	AS3	
LEBG_FIS_TWR	AS1	AS1	AS1	AS1	AS1	AS1	AS1	
LEBL_C_GND	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3
LEBL_DEL	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3
LEBL_FNL_APP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	
LEBL_N_GND	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3
LEBL_N_TWR	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3
LEBL_S_GND	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3
LEBL_S_TWR	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3	AS3	ADC AS3
LEBL_T1_APP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	
LEBL_T2_APP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	
LEBT_MIL_TWR	AS3	AS3	AS3	AS3	AS3	AS3	AS3	
LEBZ_APP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	
LEBZ_TWR	AS3	AS3	AS3	AS3	AS3	AS3	AS3	

Figura 2.1. Algunas de las restricciones de conexión a posiciones ATC de aeródromo por rango

En la división española de la red de vuelo IVAO existe una academia para nuevos usuarios llamada CAVOK [3], en ella usuarios veteranos hacen de tutores de los recién llegados alumnos. En las tutorías se enseña primero la teoría en que se basa el control aéreo en general, con énfasis en el

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

control a nivel de torre [4], y luego se pasa a unas sesiones teórico-prácticas conectándose a la red con el software de control y con otros compañeros de CAVOK haciendo de tráficos.

Sería interesante poder mostrar ya en las tutorías teóricas pequeños ejemplos prácticos de lo que se está enseñando de forma rápida, sencilla, sin depender de la disponibilidad de otros compañeros y sin necesidad de conectarse a la red. Además, el alumno podría tener una toma de contacto inicial con lo que se va a encontrar al conectarse a la red, ejercitando en prácticas y entrenamientos los conocimientos teóricos adquiridos.

Por esto, un simulador de control de tráfico aéreo a nivel de torre englobado en el contexto de la red de vuelo es interesante. Permitirá a los nuevos usuarios que deseen iniciarse en el control aéreo una primera toma de contacto con la que entrenar dirigiendo el movimiento de las aeronaves en el aeropuerto y practicar dando control a aeronaves simuladas.

2.2 Marco teórico del problema

En este apartado se describirán los conceptos necesarios para comprender el desarrollo de la aplicación y el contexto en el que se presenta.

2.2.1. International Virtual Aviation Organization (IVAO)

La Organización Internacional de Aviación Virtual (IVAO por sus siglas en inglés) es una organización fundada en 1998 para proporcionar una red de simulación aérea para entusiastas de forma que puedan disfrutar del hobby de la simulación aérea en un entorno realista en compañía de otras personas, volando o proporcionando servicios de tráfico aéreo. La red proporciona el software necesario para conectar el simulador de vuelo a la red, así como el software de control aéreo construido "in house". Además, proporcionan entrenamiento para los usuarios y todos sus servicios están disponibles de forma gratuita [5].

Todos los servicios que proporciona IVAO se hacen de forma desinteresada sin ánimo de lucro gracias a las aportaciones que hacen los usuarios en su tiempo libre.

Capítulo 2. Estado del arte y marco teórico

Si se organiza un grupo suficiente de gente dentro de la red y cumplen los requisitos, este grupo puede fundar una división. Generalmente las divisiones funcionan por países, este es el caso de IVAO España (IVAO ES) [6]. En otros casos se organizan por regiones formadas por varios países, por ejemplo, la división del este de Asia (IVAO XE) está formada por países como Corea del sur, China, Filipinas [7].

Las divisiones ofrecen servicios que están disponibles en su mayoría para todos los usuarios de la red, pero otros sólo para los miembros de la división. Uno de los servicios de IVAO España es la academia CAVOK [3], un lugar en el que los usuarios pertenecientes a IVAO España pueden aprender las bases del funcionamiento de la red, de la operativa desde el punto de vista del piloto y de la operativa desde el punto de vista del controlador.

Esta academia está formada por usuarios veteranos que ejercen de tutores, usuarios experimentados que se ofrecen como tráficos o controladores para las clases prácticas y usuarios alumnos. CAVOK proporciona la documentación necesaria para el curso en su web y en la Wiki de IVAO España. El curso está formado por una serie de tutorías comunes para pilotos y ATC (software, fraseología, planes de vuelo, meteorología, fundamentos de altimetría e introducción a los espacios aéreos), una serie de tutorías específicas para pilotos (circuito de tránsito y movimientos por la CTR) y una serie de tutorías específicas para ATC (autorizaciones IFR y control básico VFR).

2.2.2. Tipos de reglas de vuelo

Existen 4 tipos de reglas de vuelo:

- VFR: Reglas de vuelo visuales. El piloto opera la aeronave en condiciones meteorológicas suficientemente claras como para poder ver el suelo y hacia dónde se dirige la aeronave, evitando así visualmente los obstáculos y otras aeronaves [8].
- IFR. Reglas de vuelo por instrumentos. El piloto opera la aeronave guiándose únicamente por la información proporcionada por los instrumentos de la cabina de vuelo [9]
- Y. El vuelo comienza bajo reglas instrumentales y en algún punto de la ruta pasa a reglas visuales.
- Z. El vuelo comienza bajo reglas visuales y en algún punto de la ruta pasa a reglas instrumentales.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Para el simulador a desarrollar en este proyecto sólo consideraremos las dos primeras ya que desde el punto de vista operativo del controlador de torre alguien con plan de vuelo “Y” (*Yankee*) se comporta como un IFR si sale de su aeropuerto o como un VFR si llega a su aeropuerto. Ocurre algo análogo con las reglas “Z” (*Zulú*).

Para ilustrar los conceptos emplearé cartas aeronáuticas obtenidas del AIP (Publicación de Información Aeronáutica) del gestor de navegación aérea en España ENAIRE [10].

2.2.3. Control básico IFR

Todas las aeronaves comienzan en el estacionamiento. En primer lugar, deben solicitar la autorización ATC para la salida instrumental, de forma que desde el aeropuerto alcancen el primer punto de la ruta de su plan de vuelo. Con la autorización recibida proceden a la puesta en marcha, retroceso y rodaje.

Para poder realizar cualquiera de estas acciones requieren el permiso del ATC de rodadura (en inglés, “ground”). El ATC de rodadura instruye a la aeronave a rodar por unas calles de rodaje (en inglés, “taxiway”) determinadas que la misma debe seguir obligatoriamente. Esta ruta generalmente siempre tiene la misma estructura:

- El inicio se interpreta siempre respecto a la posición donde se encuentra la aeronave en el momento de dar la ruta.
- La aeronave abandona la plataforma de estacionamientos (en inglés, “parking ramp”) por una calle de rodaje que conecta la plataforma con el resto del aeropuerto. A estas calles de rodaje que conectan la plataforma con el resto se las denomina “puertas” (en inglés, “gate”) y el identificador de la calle de rodaje suele empezar por la letra “G”.
- Continúa por las calles de rodaje en el orden que fueron instruidas por el controlador.
- La ruta acaba siempre en un punto de espera (en inglés, “holding point”) de alguna pista de aterrizaje.

La figura 2.2 muestra mediante flechas rojas un ejemplo de ruta de rodaje que podría seguir una aeronave desde la plataforma de estacionamientos al punto de espera HP3 de la pista 27 en Sevilla por: Puerta G8, A.

Capítulo 2. Estado del arte y marco teórico

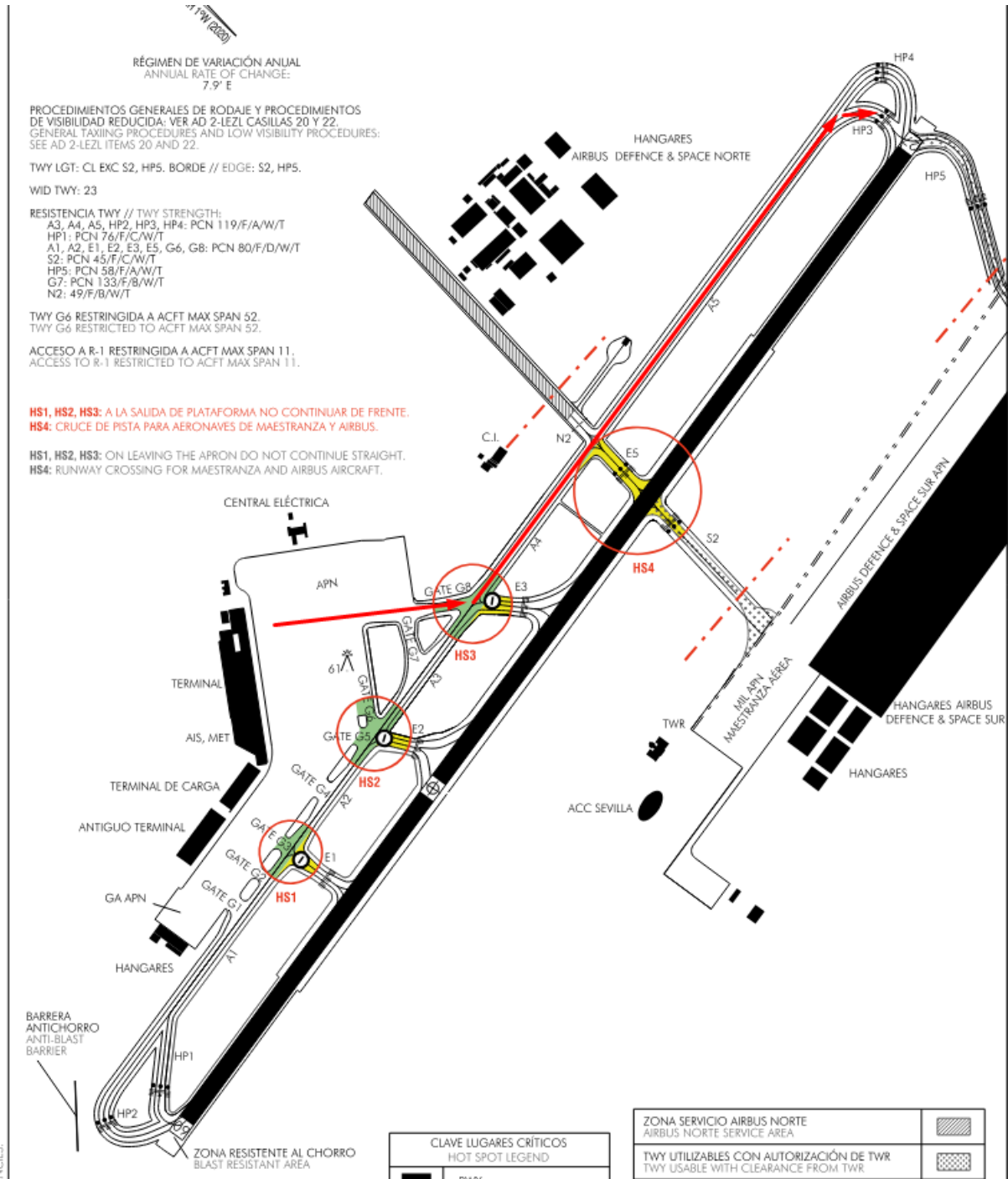


Figura 2.2. Ejemplo de ruta de rodaje en el aeropuerto de Sevilla. Carta aeronáutica obtenida de <https://aip.enaire.es/AIP/>

Todas las calles que conectan con una pista de aterrizaje tienen una señal en el suelo denominada “barra de parada” que indica el límite hasta donde una aeronave puede rodar sin que se considere una invasión de la pista. La figura 2.3 muestra el aspecto de una barra de parada en una calle de rodaje.



Figura 2.3. Barra de parada del punto de espera HP3 en Sevilla

Alcanzado el punto de espera, la aeronave pasa del controlador de rodadura al controlador de torre (en inglés, “tower”) quien autoriza el despegue cuando es seguro. Tras el despegue, el controlador de torre transfiere a la aeronave IFR a la siguiente frecuencia finalizando el trabajo del controlador de torre [11].

Tanto en despegue como en llegada, la pista escogida es la que lleve a que las aeronaves despeguen y aterricen en contra del viento para aumentar la cantidad de aire que circula sobre las superficies de control. En caso de no tener viento prevalente se emplea la pista de uso preferente.

En la llegada de una aeronave bajo reglas IFR esta contacta al controlador de torre establecido en final de la pista en uso (alineado, descendiendo y siguiendo una trayectoria que concluiría con el aterrizaje). Si las condiciones son seguras el controlador de torre autoriza a aterrizar a la aeronave, si no, puede demorar la autorización o instruir a la aeronave a realizar la maniobra de “motor y al aire” (en inglés, “go around”). Si el controlador no le autoriza a aterrizar y la aeronave se encuentra en corta final (muy cerca de la pista, listo para aterrizar), realizará la maniobra de motor y al aire por sí solo al no estar autorizado a utilizar la pista de aterrizaje [11].

La maniobra de motor y al aire que el controlador de torre ve realizar a una aeronave bajo reglas IFR consiste en cancelar el descenso hacia la pista, ascender y comenzar a aumentar la velocidad manteniendo el rumbo de pista. Tras esta maniobra el controlador debe transferir la aeronave a la dependencia de control superior para que continúe con la ruta que debe seguir al realizar una aproximación fallida (también llamada “frustrada”).

Capítulo 2. Estado del arte y marco teórico

Si la aeronave aterriza, abandona la pista por una de las calles de rodaje y se detiene en el punto de espera de dicha calle de rodaje ya que no tiene autorización para rodar. En este punto, el controlador de torre transfiere a la aeronave con el controlador de rodadura que le da la ruta de rodaje que debe seguir para llegar al estacionamiento, siendo por tanto una ruta inversa a la realizada por las aeronaves que desean despegar.

2.2.4. Control básico VFR

Las aeronaves bajo reglas de vuelo VFR contactan con el controlador de rodadura listas para el rodaje. No necesitan autorización de salida, generalmente tampoco retroceso y, al ser generalmente aeronaves pequeñas, necesitan encender el motor para evitar quedarse sin batería al no disponer de otros mecanismos que les suministren corriente eléctrica durante un largo periodo de tiempo. El rodaje es análogo al que hemos visto en el subapartado de control básico IFR.

La principal diferencia en la metodología de control respecto a las aeronaves bajo reglas IFR es en el control de torre. Las aeronaves bajo reglas VFR no tienen sólo la posibilidad de despegar y marcharse a otro aeródromo, sino que pueden realizar, entre muchas de las posibles, las siguientes acciones [12]:

- Despegar con la finalidad de viajar a otro aeródromo saliendo del espacio aéreo aeroportuario por uno de los puntos de notificación visual. Estas son áreas designadas por el gestor de navegación aérea, en España ENAIRE, por las que una aeronave bajo reglas de vuelo visuales debe salir o entrar al espacio aéreo aeroportuario. La figura 2.4 muestra mediante flechas la ruta que podría seguir una aeronave bajo reglas VFR que desee salir por el punto de notificación “N” despegando desde la pista 27 en Sevilla.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

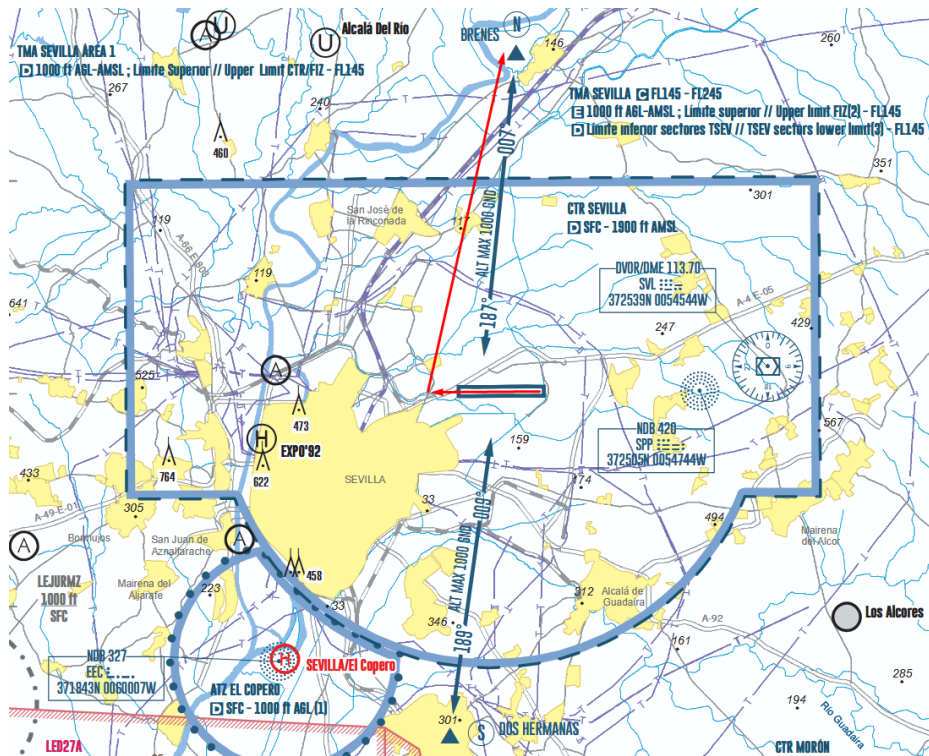


Figura 2.4 Ejemplo de ruta de salida que utilizaría una aeronave bajo reglas VFR. Carta aeronáutica obtenida de <https://aip.enaire.es/AIP/>

- Ingresar al espacio aéreo aeroportuario por uno de los puntos de notificación visual con intención de aterrizar o atravesar el espacio aéreo para salir volver a salir. La figura 2.5 muestra con flechas un ejemplo en el que la aeronave entra por “N” para aterrizar.

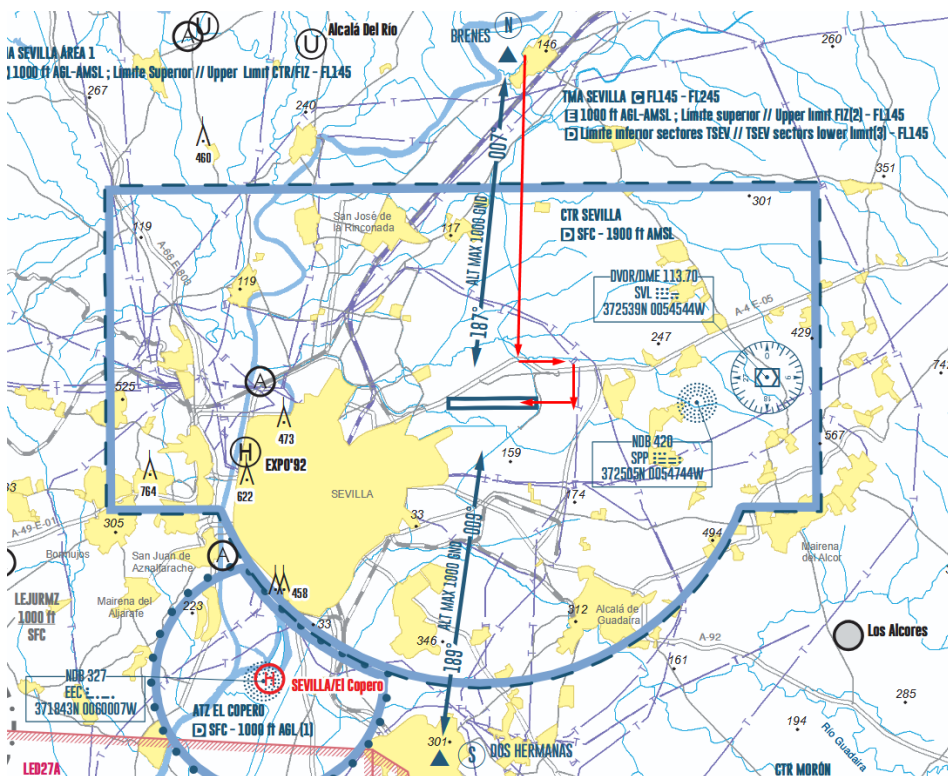


Figura 2.5. Ejemplo de ruta de llegada que utilizaría una aeronave bajo reglas VFR. Carta aeronáutica obtenida de <https://aip.enaire.es/AIP/>

Capítulo 2. Estado del arte y marco teórico

- Realizar circuitos de tránsito aéreo. Consiste en despegar y volar un patrón rectangular a uno de los lados de la pista para volver a aterrizar. La figura 2.6 ejemplifica mediante flechas la ruta aproximada que realizaría una aeronave bajo reglas VFR realizando circuitos al sur en el aeropuerto de Sevilla.



Figura 2.6. Ejemplo de circuito de tránsito aéreo al sur en el aeropuerto de Sevilla

- Cuando se realizan circuitos generalmente no es para realizar un solo despegue y aterrizaje, sino para realizar varios. En lugar de aterrizar se realiza una maniobra llamada “toma y despegue” (“touch and go” en inglés) consistente en aterrizar en la pista y, mientras la aeronave continúa rodando por la pista, configurarla para el despegue y volver a despegar continuando con la realización de circuitos.

Como se muestra en la figura 2.7, el circuito de tránsito aéreo está formado por 5 tramos, nombrados respecto a la dirección del viento respecto a la aeronave. Recordemos que, tanto en despegue como en llegada, la pista escogida es la que lleve a que las aeronaves despeguen y aterricen en contra del viento para aumentar la cantidad de aire que circula sobre las superficies de control. En caso de no tener viento prevalente se emplea la pista de uso preferente.

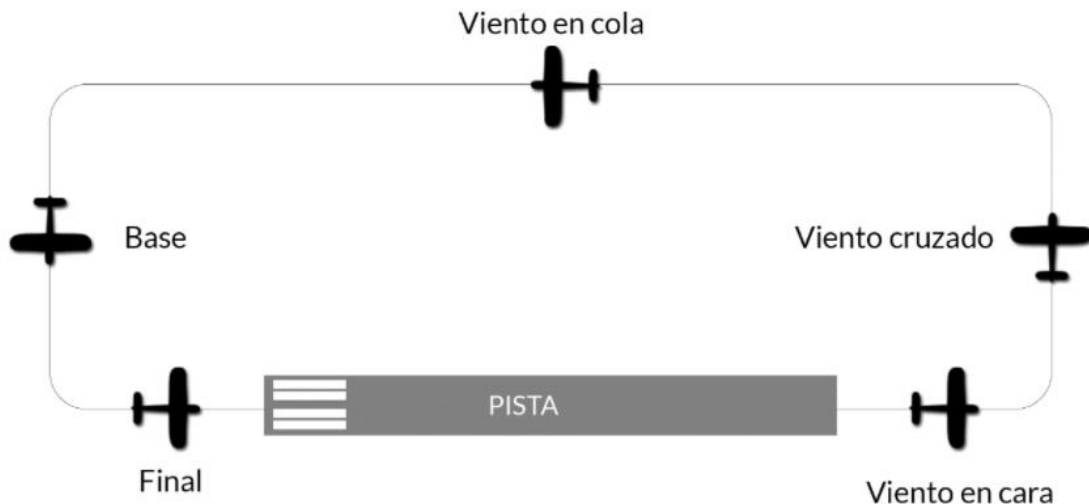


Figura 2.7. Tramos del circuito de tránsito aéreo. Imagen tomada de <https://wiki.es.ivao.aero/books/alumno-atc-avanzado-as3/page/control-basico-vfr>

2.3 Análisis tecnológico

La aplicación desarrollada en este trabajo hace de puente entre el desconocimiento absoluto en el control aéreo y el software de control utilizado en IVAO, desarrollado por la propia organización, llamado Aurora. En este apartado se describirá brevemente este software.

El cliente ATC Aurora es uno de los productos software desarrollados por IVAO. Esto les permite tener control total sobre el funcionamiento, características y desarrollo del mismo, ofreciendo la experiencia de control de tráfico aéreo en la red más realista posible. El software está disponible de forma gratuita en [13].

Para conectarse como controlador aéreo en la red el usuario debe descargar el software y conectarse a los servidores, autenticándose en el proceso e introduciendo el nombre de la dependencia que desea controlar. Si el usuario tiene las habilitaciones necesarias la conexión se realizará exitosamente y, una vez conectado, puede utilizar libremente los distintos paneles con la información que necesite: ATIS (ventana para configurar el mensaje del Servicio de Información Automática Terminal), COM (ventana de comunicaciones), ATC (ventana en la que se muestran otros usuarios cercanos conectados en la red como controladores), etc.

Capítulo 2. Estado del arte y marco teórico

La figura 2.8 ilustra la interfaz gráfica de Aurora:

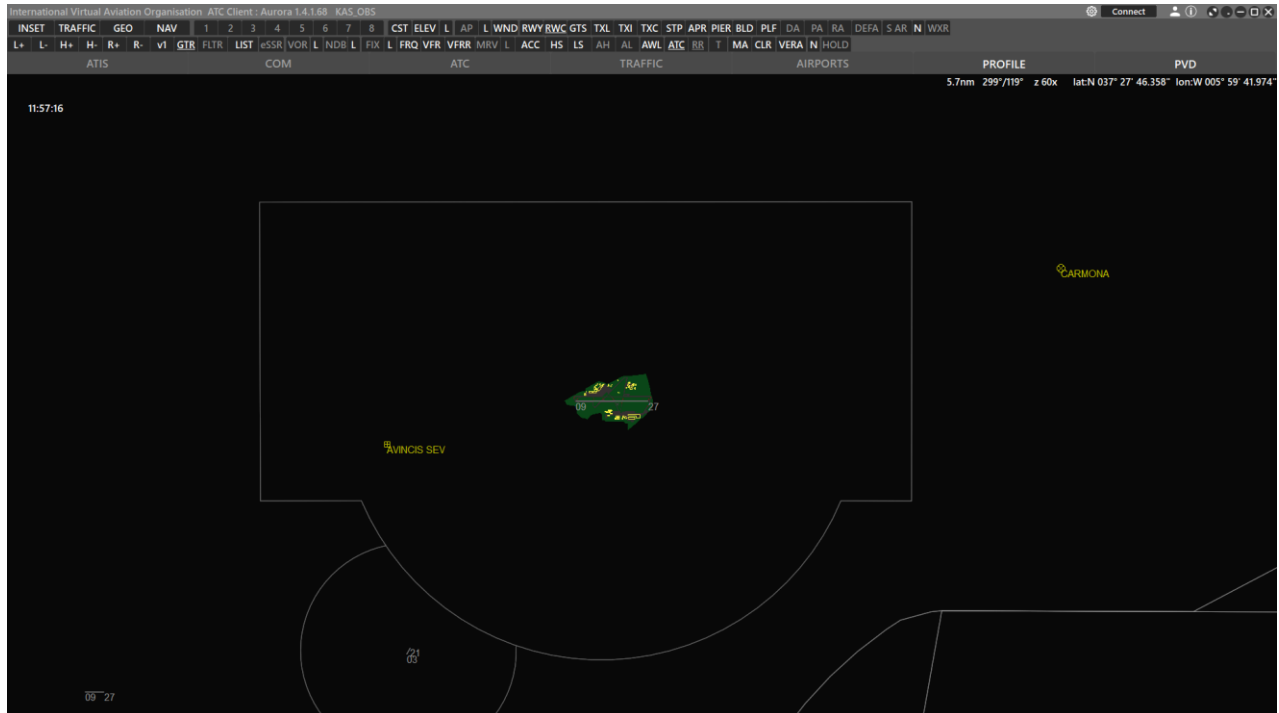


Figura 2.8. Interfaz gráfica de Aurora

2.4 Aplicaciones similares existentes

En esta sección se exploran aplicaciones similares a la que se va a desarrollar en este proyecto. El punto fundamental que deben presentar las aplicaciones es la simulación del movimiento de aeronaves controladas por la máquina, ya que es fundamental para nuestro trabajo que el usuario pueda entrenar con la aplicación en cualquier momento sin depender de la disponibilidad de otros usuarios. Por este motivo Aurora, el software ATC de IVAO, no se considera una aplicación similar a la que vamos a desarrollar al no contar con esta funcionalidad.

2.4.1. Endless ATC

Es un simulador de control de tráfico aéreo realista desde el punto de vista de un controlador de aproximación (aproximador), disponible para PC (Steam) [14] y Android [15] por 7,79€. En la aplicación sólo existe la posibilidad de controlar aeronaves bajo reglas IFR. La aplicación simula bien el trabajo que necesita realizar un aproximador, por ello es una de las aplicaciones que utilizan los usuarios de IVAO para practicar a secuenciar aeronaves en aproximación (organizarlas para que

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

todas puedan aterrizar maximizando el uso del espacio aéreo de forma eficiente y segura). En la realidad el control de aproximación es algo diferente debido a que los espacios aéreos están a su vez subdivididos para que los controladores puedan gestionar mejor y de forma más segura la carga de trabajo.

Dispone la capacidad de cargar aeropuertos personalizados y así poder generar diferentes escenarios de entrenamiento.

Dispone de una versión demo. El desarrollador continúa trabajando en él mediante parches que añaden y corrigen funcionalidades.

2.4.2. VoiceATC Simulator

Es otro simulador de control de tráfico aéreo similar a Endless ATC, también dispone la capacidad de añadir aeropuertos personalizados por el usuario. La principal diferencia con Endless ATC es que está enfocado a controlar mediante comandos por voz.

Actualmente está en acceso anticipado en Steam por 4,99€ sin demo gratuita [16].

2.4.3. OpenScope

Simulador de control de tráfico aéreo a nivel de aproximación, esta vez en formato web programado en JavaScript [17].

Sus principales puntos fuertes respecto a las aplicaciones presentadas en los subapartados anteriores son la capacidad de correr directamente en el navegador web y que ser open source, por lo que se dispone del código fuente y la capacidad de contribuir en el proyecto gracias a su GitHub [18].

2.4.4. Tower! Simulator 3

En este caso, el simulador de control aéreo presenta un entorno 3D aeroportuario y está centrado en la posición de torre. Las funciones que permite desempeñar es dar rodaje a aeronaves,

Capítulo 2. Estado del arte y marco teórico

autorizaciones de despegue y aterrizajes tanto bajo reglas VFR como IFR, pero la implementación del control VFR no es del todo correcta. Tampoco permite añadir nuevos aeropuertos y escenarios por parte de los usuarios.

Está disponible en Steam por 49.99€ [19]

2.5 Conclusiones

La mayoría de las aplicaciones presentadas se centran en el control de aeronaves bajo reglas IFR, la única centrada en el control de aeronaves bajo reglas VFR similar a lo que se realiza en redes de vuelo como IVAO no posee una funcionalidad adecuada que cubra los objetivos del proyecto.

En la mayoría simulan el rol de un controlador de aproximación, no de un controlador de torre. Por lo tanto, hay una carencia de aplicaciones de fácil acceso para los nuevos usuarios de la red para aprender y practicar en las fases iniciales de aprendizaje de control de tráfico aéreo a nivel de torre.

Tampoco se considera Aurora como una aplicación similar ya que no cubre la funcionalidad de simular el movimiento de aeronaves controladas por la máquina, lo que permitiría el entrenamiento del usuario en cualquier momento sin depender de la disponibilidad de otros usuarios de la red.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Capítulo 3. Análisis

3.1 Introducción

En este apartado se cubre el análisis de requisitos y necesidades funcionales que debe cumplir el sistema, documentando qué debe hacer el sistema y no cómo. Utilizando los objetivos del proyecto y en el marco teórico presentado se realiza el modelado y análisis de las diferentes partes que forman la aplicación.

3.2 Modelo de negocio

Mediante el modelo de negocio se obtiene una lista de entidades claves que deben estar presentes en el sistema final. Estas entidades se expresan mediante clases. A partir de las clases se construye el modelo de dominio que define qué bloques deben formar parte del sistema y las relaciones entre ellos.

3.2.1 Clases conceptuales

- **CC1. Aeronave:** Representa cualquier tipo de aeronave que pueda estar presente en el entorno aeroportuario en la red de vuelo IVAO. Posee los siguientes atributos:
 - **Indicativo:** Identificador único para cada aeronave. Generalmente se trata de la matrícula para aeronaves VFR o de la compañía y número de vuelo para aeronaves IFR.
 - **Modelo de aeronave:** Indica el código ICAO del modelo de la aeronave.
 - **Posición:** Coordenadas geográficas de la aeronave.
 - **Rumbo:** Indica el rumbo al que apunta el morro de la aeronave.
 - **Altitud:** Altitud respecto al nivel del mar a la que se encuentra la aeronave en pies.
 - **Velocidad vertical:** Tasa con la que la aeronave está ascendiendo o descendiendo en pies por minuto.
 - **Estacionamiento:** Parking asignado a la aeronave.
 - **Agente:** Agente que está controlando la aeronave.

- **CC2. Aeropuerto:** Representa cualquier aeropuerto que pueda controlar un usuario recién llegado a la red de vuelo. Contiene los siguientes atributos:
 - **ICAO:** Código identificativo único del aeropuerto.
 - **Nombre:** Nombre del aeropuerto.
 - **Elevación:** Altitud sobre el nivel del mar a la que se encuentra el aeropuerto en pies.
 - **Pista activa:** Indica la pista actualmente en uso.
 - **Localización:** Coordenadas geográficas del centro del aeropuerto.
 - **Pistas:** Lista de pistas que contiene el aeropuerto.
 - **Puntos de notificación visual:** Lista de puntos de notificación para aeronaves VFR que entren o salgan del espacio aéreo asociado al aeropuerto.
 - **Espacios aéreos:** Lista de espacios aéreos asociados al aeropuerto.
 - **Estacionamientos:** Lista de estacionamientos que posee el aeropuerto.
 - **Calles de rodaje:** Lista de calles de rodaje que posee el aeropuerto.
 - **Barras de parada:** Lista de barras de parada que posee el aeropuerto.
 - **Plataforma de estacionamientos:** Define la plataforma de estacionamientos del aeropuerto.
- **CC3. Agente:** Representa el sistema capaz de pilotar las aeronaves, realizando las diversas maniobras que pudiera realizar un usuario de la red conectado como piloto. Sus atributos son:
 - **Aeronave:** Determina la aeronave que está controlando el agente.
 - **Aeropuerto:** Determina el aeropuerto donde está volando el agente.
 - **Estado:** Determina el estado interno del agente.
- **CC4. Infracción:** Representa cualquier infracción que el usuario pueda cometer al realizar control de tráfico aéreo. Está caracterizado por:
 - **Tipo de infracción:** Nombre que indica el tipo de infracción cometida.
 - **Conteo:** Número de veces que se ha cometido la infracción.
- **CC5. Sesión:** Representa una sesión de entrenamiento realizada en el simulador. Los atributos son:
 - **Fecha de la sesión:** Contiene la fecha en la que se realizó la sesión de entrenamiento.
 - **Duración de la sesión:** Contiene el tiempo que se estuvo realizando el entrenamiento.
 - **ICAO del aeropuerto:** Indica el código identificador único del aeropuerto

Capítulo 3. Análisis

- **Contadores:** Indica el número de veces que se han producido eventos importantes en la sesión como despegues, aterrizajes o infracciones.

3.2.2 Diagrama de modelo de dominio

La figura 3.1 muestra el diagrama de modelo de dominio resultante del estudio del modelo de negocio.

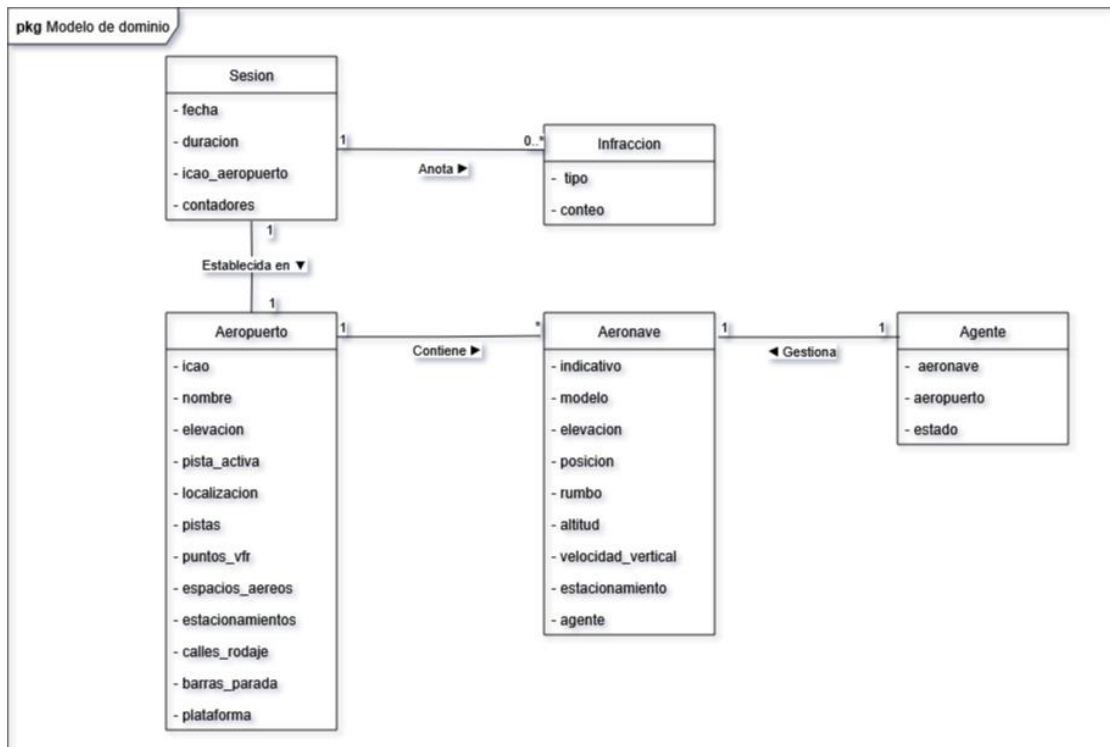


Figura 3.1 Diagrama de modelo de dominio

3.3 Análisis de requisitos

3.3.1. Análisis de requisitos de usuario

Los requisitos de usuario describen lo que debe ser capaz de hacer el usuario con el sistema, centrándose en objetivos y elementos percibidos por el usuario de la aplicación.

El usuario debe ser capaz de realizar una sesión de entrenamiento de control de tráfico aéreo a nivel de torre en el aeropuerto que desee. Además, debe poder revisar las estadísticas de los entrenamientos realizados donde se muestre la duración de la sesión, aeronaves controladas y fallos cometidos.

3.3.2. Análisis de requisitos del sistema

Los requisitos del sistema describen qué debe hacer el sistema según las condiciones. Se emplean para definir qué se debe implementar y qué se debe probar en el sistema.

Requisitos funcionales

- **RF-01: Inicializar una sesión de simulación**
 - El usuario debe poder iniciar la sesión en el aeropuerto deseado de los cargados desde el almacenamiento persistente.
 - El usuario debe poder escoger las condiciones meteorológicas de la sesión (VFR o MVFR). No se contemplarán condiciones no aptas para el vuelo VFR.
- **RF-02: Renderizado de la pantalla radar**
 - El estilo debe estar inspirado en la representación radar del software de control aéreo de IVAO "Aurora".
 - Se deben representar las pistas del aeropuerto controlado.
 - Se deben representar las calles de rodaje del aeropuerto controlado.
 - Se deben representar las barras de parada del aeropuerto controlado.
 - Se debe representar la plataforma de parking del aeropuerto controlado.
 - Se debe representar el espacio aéreo del aeropuerto controlado.
 - Se deben representar los puntos de notificación asociados con el aeropuerto controlado.
 - Se deben mostrar la estela de las aeronaves.
 - Se deben mostrar las etiquetas con información de las aeronaves
- **RF-03: Selección de aeronaves en la pantalla radar**
 - El usuario debe ser capaz de seleccionar a qué aeronave desea enviar la instrucción.
- **RF-04: Representación de las peticiones de los agentes en la interfaz gráfica**
 - La interfaz gráfica debe informar al usuario de que tiene peticiones de las aeronaves pendientes de atender.
- **RF-05: Actualización continua de la simulación**
 - El sistema debe ser capaz de calcular el siguiente estado de la simulación recalculando la posición de las aeronaves en base a sus propiedades como velocidad, altura, velocidad vertical, posición, ...

Capítulo 3. Análisis

- **RF-06: Aeronaves controladas por un agente**
 - El agente debe ser capaz de seguir una ruta de vuelo o una ruta en tierra.
 - El agente debe ser capaz de recibir instrucciones, comprobar si son válidas y cumplirlas.
 - El agente debe exponer sus intenciones al usuario.
 - En caso de no recibir autorización de aterrizaje o toma y despegue por parte del controlador, el agente debe realizar automáticamente la maniobra de “motor y al aire”.
- **RF-07: El usuario debe poder instruir a las aeronaves a realizar alguna acción**
 - Instruir a una aeronave a rodar a pista.
 - Instruir a una aeronave a entrar en pista y mantener posición sin despegar.
 - Instruir a una aeronave a despegar.
 - Instruir a una aeronave bajo reglas VFR a virar a base y final.
 - Instruir a una aeronave en final a aterrizar.
 - Instruir a una aeronave bajo reglas VFR en final a realizar toma y despegue.
 - Instruir a una aeronave a hacer motor y al aire.
 - Instruir a una aeronave a rodar al estacionamiento.
 - Instruir a una aeronave a cambiar de frecuencia.
- **RF-08: El sistema debe permitir al usuario elegir la ruta de rodaje que desea instruir al tráfico**
 - La interfaz de usuario debe proporcionar algún método para que el usuario escoja la ruta de rodaje.
- **RF-09: El sistema debe ser capaz de validar la ruta de rodaje recibida por el usuario**
 - Se debe comprobar que para el rodaje a la pista desde la plataforma de aparcamientos la ruta debe comenzar en una calle conectada a la plataforma, continuar mediante calles conectadas entre sí y finalizar en una calle que contenga una barra de parada.
 - Para el rodaje desde la pista a la plataforma de aparcamientos se debe comprobar que la primera calle corresponde con la calle por la que salió de la pista la aeronave, la última calle de rodaje debe conectar con la plataforma de aparcamientos y la ruta debe estar formada por calles de rodaje conectadas entre sí.

- **RF-10: El sistema debe contar con un agente supervisor que compruebe si se incumplen alguna de las reglas definidas como malas prácticas en el control de tráfico aéreo**
 - El sistema debe penalizar al usuario si autoriza dos despegues a la vez.
 - El sistema debe penalizar al usuario si detecta un conflicto por excesiva cercanía de dos aeronaves en el aire.
 - El sistema debe penalizar al usuario si detecta un conflicto por excesiva cercanía de una aeronave en el aire con una en tierra, especialmente pensado para el caso de un conflicto de aterrizaje y otra aeronave alineada manteniendo en la pista o despegando.
 - El sistema debe penalizar al usuario si instruye una ruta de rodaje inválida.
 - El sistema debe penalizar al usuario si una aeronave realiza una maniobra de motor al aire de forma automática por considerarse un descuido del usuario (No así si el usuario instruye el motor y al aire antes de que el agente deba tomar la decisión por sí sola)
- **RF-11: La interfaz gráfica debe proporcionar al usuario un feedback inmediato cuando cometa un error**
 - Deben representarse los fallos en la interfaz gráfica.
- **RF-12: Carga modular de aeropuertos y modelos de avión**
 - El sistema debe cargar aeropuertos y modelos de avión desde el almacenamiento persistente.
 - Debe abortar la carga si los datos son inválidos.
- **RF-13: Guardado de los resultados de la sesión de entrenamiento en el almacenamiento persistente**
 - El sistema debe guardar los resultados de la sesión de entrenamiento para que el usuario pueda consultarlos más adelante.
- **RF-14: Carga de los resultados de las sesiones de entrenamiento realizadas desde el almacenamiento persistente**
 - El sistema debe cargar los resultados de las sesiones de entrenamiento guardadas en el almacenamiento persistente para su representación mediante una interfaz gráfica.

Capítulo 3. Análisis

- **RF-15: El usuario debe poder revisar las estadísticas de sesiones de control realizadas**
 - Debe disponer de una interfaz gráfica en la que visualizar las estadísticas de las sesiones.

Requisitos no funcionales

- **RNF-01: El sistema debe ser capaz de mantener una representación de la pantalla radar fluida**
 - La tasa de refresco debe garantizar una fluidez visual en la simulación.
- **RNF-02: Soporte multiplataforma**
 - La aplicación debe poder ser ejecutada en múltiples plataformas como Windows, MacOS y Linux.
- **RNF-03: El bucle de simulación debe funcionar de forma fluida para que no se aprecien saltos en el movimiento de las aeronaves**
 - La tasa de actualización debe ser suficientemente elevada para que no se perciban saltos en las actualizaciones de la simulación.
- **RNF-04: El sistema debe ser fácilmente extensible por el usuario**
 - Añadir un nuevo aeropuerto sólo debe requerir añadir su información en el almacenamiento persistente en el lugar y formato adecuado.
 - Añadir un nuevo modelo de avión sólo debe requerir añadir su información en el almacenamiento persistente en el lugar y formato adecuado.
- **RNF-05: El sistema debe ser fácilmente extensible por desarrolladores futuros**
 - Las clases deben tener la mayor genericidad posible y ser intercambiables por otras que realicen la misma función.

3.3.3 Casos de uso

En este apartado se describen los casos de uso que representan la funcionalidad que debe cubrir la aplicación.

Caso de uso principal

La figura 3.2 muestra el diagrama de casos de uso que representa los dos casos de uso principales para los que el usuario puede utilizar la aplicación.

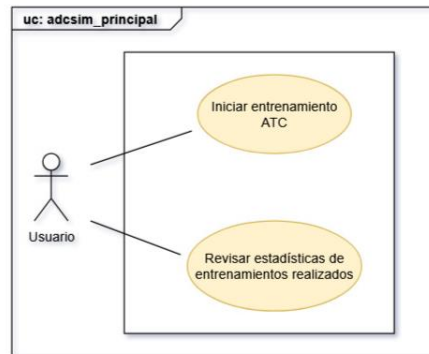


Figura 3.2 Diagrama de casos de uso principal

A continuación, desarrollamos cada caso de uso. En primer lugar, la tabla 3.1 desarrolla el caso de uso principal en el que el usuario desea iniciar una nueva sesión de entrenamiento. La tabla 3.2 desarrolla el caso de uso en el que el usuario desea revisar las estadísticas de los entrenamientos realizados.

Caso de uso 1.1: Iniciar entrenamiento ATC	
Actores:	Usuario
Resumen:	Inicia una nueva sesión de entrenamiento
Precondición:	El usuario tiene una base de datos de aeropuertos y modelos de avión válida
Secuencia normal:	<ol style="list-style-type: none"> 1. El usuario inicia la aplicación y selecciona la opción para iniciar una nueva sesión. 2. Escoge el aeropuerto donde quiere realizar el entrenamiento de la lista. 3. Escoge la meteorología deseada en la sesión. 4. Pulsa el botón para iniciar la sesión.
Postcondición:	Se inicia la simulación para que el usuario realice la sesión de entrenamiento.
Comentarios:	Es el caso de uso raíz de todos los casos de uso relacionados con el simulador

Tabla 3.1. Caso de uso 1.1: Iniciar entrenamiento ATC

Capítulo 3. Análisis

Caso de uso 1.2: Revisar estadísticas de entrenamientos realizados	
Actores:	Usuario
Resumen:	Permite al usuario consultar las estadísticas de sesiones pasadas
Precondición:	El usuario tiene sesiones de entrenamiento en la base de datos.
Secuencia normal:	<ol style="list-style-type: none">1. El usuario inicia la aplicación y selecciona la opción para consultar las estadísticas de entrenamientos realizados.2. Escoge el entrenamiento que desea revisar con más detalle.
Postcondición:	Se muestra al usuario los detalles del entrenamiento seleccionado.
Comentarios:	Ninguno.

Tabla 3.2. Caso de uso 1.2: Revisar estadísticas de entrenamientos realizados

Casos de uso relacionados con la simulación

Todos los siguientes casos de uso dependen de que el usuario haya iniciado una simulación, es decir, haya realizado el caso de uso 1.1.

Caso de uso 2.1: Selección de una aeronave en la pantalla radar	
Actores:	Usuario
Resumen:	Permite marcar una aeronave de la simulación como objetivo.
Precondición:	El usuario tiene una sesión de simulación iniciada y hay aeronaves presentes en el motor de simulación.
Secuencia normal:	<ol style="list-style-type: none">1. El usuario pulsa sobre la aeronave que desea seleccionar
Postcondición:	La aeronave objetivo queda seleccionada y se muestran las posibles instrucciones que se le pueden dar, si hay alguna.
Comentarios:	Es el caso de uso raíz de todos los relacionados con instruir a la aeronave a realizar una acción

Tabla 3.3. Caso de uso 2.1: Selección de una aeronave en la pantalla radar

La tabla 3.3 se centra en el caso de uso en el que el usuario desea seleccionar una aeronave en la pantalla radar. La tabla 3.4 desarrolla el caso de uso en el que el usuario desea instruir a una aeronave a realizar una acción en concreto.

Caso de uso 2.2: Instruir a una aeronave a realizar una acción	
Actores:	Usuario
Resumen:	Envía una instrucción al agente de la aeronave la cual realiza la acción si es posible.
Precondición:	Caso de uso 2.1.
Secuencia normal:	<ol style="list-style-type: none">1. El usuario comprueba las peticiones realizadas por los agentes pendientes de atender.2. Selecciona la aeronave que desea atender.3. Selecciona la instrucción a enviar al agente que controla la aeronave.
Postcondición:	El agente de la aeronave analiza la acción y la acepta y realiza, o rechaza y no hace nada.
Comentarios:	Es el caso general a la hora de enviar cualquier acción a un agente.

Tabla 3.4. Caso de uso 2.2: Instruir a una aeronave a realizar una acción

La tabla 3.5 cubre el caso de uso en el que el usuario desea seleccionar y enviar la ruta de rodaje a una aeronave.

Caso de uso 2.3: Seleccionar y enviar la ruta de rodaje a una aeronave	
Actores:	Usuario
Resumen:	Envía una ruta de rodaje al agente asociado a la aeronave la cual realiza si es una ruta válida.
Precondición:	Hay una aeronave pendiente de instruir rodaje.
Secuencia normal:	<ol style="list-style-type: none"> 1. El usuario comprueba que tiene una petición de rodaje pendiente de atender 2. Selecciona la aeronave a la que desea instruir el rodaje. 3. Selecciona la ruta de rodaje que desea enviar a la aeronave.
Postcondición:	El agente de la aeronave analiza la ruta y la acepta y realiza, o rechaza y no hace nada.
Comentarios:	Es el caso general para cualquier rodaje.

Tabla 3.5. Caso de uso 2.3: Seleccionar y enviar la ruta de rodaje a una aeronave

3.4 Entidades principales

Utilizando la información de los apartados anteriores podemos finalmente determinar las entidades que deberán estar presentes en el sistema:

- **Aircraft:** Representa una aeronave, se identifica mediante su indicativo y cada una es controlada por una instancia de agente. Contiene propiedades estáticas y propiedades dinámicas.
 - **Propiedades estáticas:** Indicativo, modelo de avión, agente que lo controla.
 - **Propiedades dinámicas:** Posición, rumbo, altitud, velocidad, velocidad vertical.
- **AircraftModel:** Representa un modelo de avión, sirve como molde para generar instancias de Aircraft. Se identifica por el código ICAO del modelo de la aeronave.
 - **ICAO:** Nombre identificativo del modelo de avión, no necesariamente único ya que un mismo modelo de avión puede tener variantes menores y todas comparten el mismo ICAO.
 - **Peso:** Peso en vacío del modelo.
 - **Velocidad máxima:** Velocidad máxima por diseño del modelo en nudos.

- **Velocidad de crucero:** Velocidad de crucero del modelo en nudos.
- **Velocidad V2:** Velocidad mínima de seguridad tras el despegue en nudos.
- **Velocidad de aproximación:** Velocidad típica a la que el modelo vuela en la fase de aproximación en nudos.
- **Altitud máxima:** Altitud máxima que puede alcanzar el modelo por diseño en pies.
- **Ratio de ascenso inicial:** Velocidad vertical a la que el modelo asciende típicamente en los momentos iniciales tras despegar en pies por minuto.
- **Rol:** Indica si el modelo es típicamente VFR o IFR para inyectarlo en la simulación como un tipo u otro.
- **Airport:** Representa un aeropuerto, se identifica de forma inequívoca mediante su código ICAO.
 - **ICAO:** Identificador inequívoco del aeropuerto. Es único para cada uno.
 - **Nombre:** Nombre del aeropuerto.
 - **Elevación:** Altitud sobre el nivel del mar en pies.
 - **Pista activa:** Pista utilizada para despegues y aterrizajes actualmente.
 - **Latitud/Longitud:** Coordenadas geográficas del aeropuerto.
 - **Pistas:** Conjunto de pistas que posee el aeropuerto.
 - **Puntos VFR:** Conjunto de puntos de notificación visual que posee el aeropuerto.
 - **Espacios aéreos:** Conjunto de espacios aéreos asociados al aeropuerto.
 - **Parkings:** Estacionamientos para aeronaves que posee el aeropuerto.
 - **Calles de rodaje:** Conjunto de taxiways (calles de rodaje) que posee el aeropuerto.
 - **Barras de parada:** Conjunto de stopbars que posee el aeropuerto.
 - **Rampa de estacionamiento:** Plataforma de estacionamientos del aeropuerto.
 - **Designador de la calle de salida de pista:** Indica al agente por qué calle de rodaje debe abandonar la pista tras aterrizar.
- **AirportWaypoints:** Representa el conjunto de puntos de ruta asociados al aeropuerto necesarios para el vuelo local.
 - **Airport:** Entidad de Airport asociada a esta entidad.
 - **Pista:** Pista de referencia para la que existen estos puntos de ruta.
 - **Puntos de ruta:** Viento en cara, viento cruzado, viento en cola, base y zona de aterrizaje de la pista activa.

Capítulo 3. Análisis

- **Airspace:** Representa un espacio aéreo. Se identifica por el nombre.
 - **Nombre:** Identificador del espacio aéreo.
 - **Coordenadas:** Conjunto de coordenadas que delimitan el espacio aéreo.
- **Parking:** Representa un estacionamiento del aeropuerto. Se identifica por el designador.
 - **Designador:** Código que distingue al parking del resto.
 - **Posición:** Coordenadas de su ubicación geográfica.
 - **Rol:** Indica si es un estacionamiento para aeronaves VFR o aeronaves IFR. Esto permitirá separar aviación general de tráfico comercial de forma simple.
 - **En uso:** Indica si está siendo utilizado por alguna aeronave.
- **ParkingRamp:** Representa la rampa o plataforma donde se encuentran los estacionamientos. Se identifica por su designador.
 - **Designador:** Código que distingue la rampa.
 - **Coordenadas:** Lista de coordenadas geográficas que forman el polígono que delimita la plataforma.
- **Runway:** Representa una pista de aterrizaje. Se distingue por su nombre el cual viene dado por la orientación de la pista.
 - **Nombre:** Código de dos números y, opcionalmente, las letras “L”, “C,” o “R”, que da nombre a la pista de forma inequívoca dentro del mismo aeropuerto.
 - **Longitud:** Longitud de la pista en metros.
 - **Ancho:** Anchura de la pista en metros.
 - **Coordenadas:** Conjunto de coordenadas geográficas que delimitan las cuatro esquinas de la pista.
- **Stopbar:** Representa una barra de parada típicamente encontrada en los puntos de espera de las calles de rodaje antes de entrar en una pista de aterrizaje.
 - **Nombre:** Designador de la barra de parada, debe coincidir con el de la calle de rodaje que la contiene.
 - **Coordenadas:** Localización geográfica de la barra de parada.
- **Taxiway:** Representa una calle de rodaje caracterizada por su nombre y su línea central.
 - **Nombre:** Designador de la calle de rodaje.
 - **Coordenadas:** Conjunto de coordenadas que forman la línea central de la calle de rodaje.

- **VFRpoint:** Representa un punto de notificación visual.
 - **Nombre:** Designador del punto, típicamente una o dos letras para indicar la referencia geográfica y un número si hay varios en la misma zona geográfica. Ejemplo. S, N, S-1, NW.
 - **Coordenadas:** Coordenadas geográficas del punto de notificación visual.
- **Waypoint:** Representa un punto de ruta, en su mínima esencia es una posición geográfica que tiene asociada una altitud. Será utilizado por el agente para saber hacia dónde debe dirigirse y a qué altitud debe estar en ese punto.
 - **Coordenadas:** Localización geográfica del punto de ruta.
 - **Altitud:** Altitud a la que se encuentra el punto de ruta.
- **AircraftAI:** Representa un agente capaz de controlar una entidad de aeronave.
 - **Aircraft:** Aeronave que controla
 - **Airport:** Aeropuerto del que obtiene los puntos de ruta y otros datos necesarios para la navegación.
 - **Estado:** Representa el estado en que se encuentra el agente.
 - **Supervisor:** Agente supervisor al que informar si detecta que ha recibido una instrucción errónea por parte del usuario.
 - **Ruta de rodaje:** Ruta de rodaje validada recibida del usuario que debe seguir.
 - **Flags internos de comportamiento:** Indica al agente qué puede o no hacer en función del estado interno en que se encuentra.
- **SimulationEngine:** Representa el motor de simulación.
 - **Aeronaves actualmente en la simulación:** Conjunto de aeronaves simuladas en el instante de simulación actual.
 - **Contadores de número de aeronaves IFR o VFR:** Mantienen un conteo para saber si se deben inyectar nuevas aeronaves a la simulación y de qué tipo.
 - **Aeropuerto:** Identifica el aeropuerto cargado en la simulación.
 - **Supervisor:** Identifica el supervisor encargado de detectar, notificar y anotar las infracciones cometidas por el usuario.

Capítulo 3. Análisis

- **SimulationHypervisor:** Representa el supervisor encargado de comprobar en cada bucle de simulación si el usuario está incumpliendo alguna regla, o recibir notificaciones de infracciones cometidas por el usuario.
 - **Fallos:** Conjunto de fallos cometidos con sus contadores.
 - **Contadores:** Llevan la cuenta del número de despegues, aterrizajes, aeronaves IFR o VFR gestionadas.

3.5 Conclusiones

Hemos obtenido los componentes fundamentales del sistema al realizar el análisis del software, definiendo el modelo de negocio, analizando los requisitos que deben satisfacerse, los casos de uso principales y las entidades principales que formarán los bloques de la aplicación.

En los siguientes capítulos utilizaremos estos productos de ingeniería del software para resolver el problema planteado en el proyecto.

Capítulo 4. Diseño

4.1 Introducción

En este capítulo se dará una solución al problema planteado por los requisitos funcionales planteados en el análisis, detallando cómo interactuarán las entidades entre sí. El lenguaje escogido para el proyecto es Java, por lo que se realizará un diseño orientado a objetos.

4.2. Diseño Arquitectónico

El estilo arquitectónico predominante a utilizar es el Modelo-Vista-Controlador (MVC), mostrado en la figura 4.1, en el que la aplicación se divide principalmente en 3 capas:

- **Modelo:** Representa los datos con los que va a trabajar la aplicación y la lógica de negocio que se aplica a cada uno.
- **Vista:** Es la representación visual de los datos contenidos en el modelo.
- **Controlador:** Es el intermediario entre la vista y el modelo. Recibe las acciones del usuario desde la vista y actúa sobre el modelo para proporcionar una respuesta desde el mismo que sea presentada al usuario.

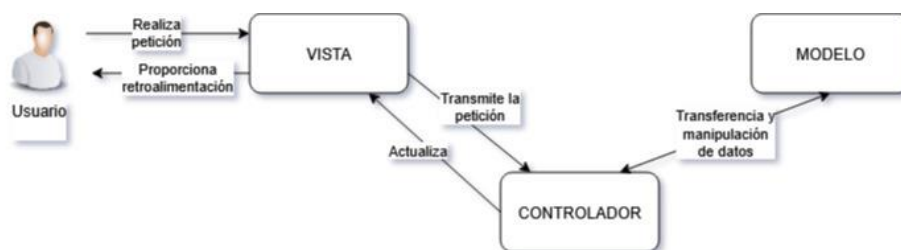


Figura 4.1. Representación del Modelo-Vista-Controlador

Además de estas capas proporcionadas por el MVC, son necesarias las siguientes:

- **Capa de servicios:** Proporciona acceso y guardado de datos almacenados en almacenamiento externo.
- **Capa de utilidades:** Proporciona herramientas comunes que son utilizadas por el resto de la arquitectura.

4.3. Patrones de diseño utilizados

Para resolver diversos problemas encontrados en el diseño de la aplicación se han aplicado patrones de diseño en medida de lo posible debido a ser soluciones comúnmente aceptadas. Aunque un patrón sea una solución aceptada, no siempre es la indicada, por lo que se indicará el motivo por el que se ha escogido este patrón y no otro.

4.3.1. Modelo-Vista-Controlador

Es el patrón que domina la estructura de la aplicación para una organización de grano grueso. En un proyecto formado por múltiples personas ayuda a que cada parte del equipo pueda trabajar en una parte por separado, en este caso en que sólo es una persona, ayuda a organizar las ideas, separar conceptos en el diseño y desarrollo.

Principalmente aborda del problema de separar la aplicación en partes claramente diferenciadas para evitar mezclar código, encapsulando cada parte y facilitando la refactorización y mantenimiento. En caso de JavaFX la separación de vista y controlador altamente recomendada ya que la vista la forman en la mayoría de los casos los archivos FXML y el controlador clases Java que interactúan con los FXML mediante anotaciones en el código “@FXML”.

En nuestro caso, también habrá clases Java que forman la vista como por ejemplo clases encargadas de redibujar el canvas de JavaFX que forma la pantalla radar en cada actualización.

4.3.2. Observer

Este patrón permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que suceda al objeto que están observando [20].

En la aplicación tenemos principalmente dos entidades que deben ser observadas por otras para actuar acorde a los cambios: El motor de simulación (para que la GUI se actualice acorde a los cambios) y el supervisor de la simulación (para que la GUI notifique las infracciones detectadas por el supervisor).

La solución consiste en añadir un mecanismo de suscripción en la clase notificadora de forma que tras un cambio esta clase revise quienes se han suscrito y llame a un método en concreto del

Capítulo 4. Diseño

observador, provocando su reacción al cambio. Para saber qué método se debe llamar, las clases observadoras deben implementar una interfaz que define el método que deben implementar si quieren ser observadores. Además, la clase notificadora utiliza la interfaz para comprobar que el llamante es efectivamente alguien que ha implementado el método al que va a llamar para notificar a los observadores de su cambio.

La principal ventaja de este patrón es la extensibilidad, ya que permite añadir fácilmente más observadores, y el desacoplamiento de la clase notificadora con las observadoras, la clase notificadora sólo sabe que debe notificar a aquellos que implementaron la interfaz, no quienes son en concreto.

4.3.3. Strategy/Provider

Permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables [20].

En la aplicación soluciona el problema de poder obtener los datos de fuentes diferentes. En los primeros ciclos de desarrollo todavía no se tiene el proveedor de datos para la lectura de archivos JSON implementado, una solución rápida para poder avanzar mientras se aborda es codificar directamente los datos en el código, sin embargo, con esto corremos el riesgo de acoplar la aplicación con un elemento que no va a ser el definitivo.

Para solucionar este problema se define una interfaz que indica qué datos o solución va a proporcionar aquellos que implementen esta interfaz. Quien necesite los datos simplemente llama a los métodos definidos en la interfaz. Si se crean varios proveedores distintos: uno que proporcione datos desde el código, otro que proporcione los datos desde JSON, otro que proporcione los datos desde web, etc., las implementaciones son independientes de aquellas clases que utilizan los datos implementados.

Otra importante ventaja que proporciona este patrón en este proyecto es la capacidad de probar cambios en el modelo de datos rápidamente, o cambios en la estructura de los datos que se obtienen desde el exterior de la aplicación.

4.3.4. Otras consideraciones

Hay algún patrón que se han evitado deliberadamente porque, a pesar de solucionar el problema, introducen otros. Uno de ellos es el patrón singleton, este patrón permite asegurarnos que sólo existe una instancia de una clase, a la vez que proporciona un punto de acceso global a dicha instancia.

En su lugar, se ha empleado la inyección de dependencias. Establecemos por norma quién es la clase encargada de crear la instancia de esa clase y la creamos únicamente allí. Posteriormente, cuando otras clases necesitan esa clase única porque dependen de ella, se le proporciona la referencia a la instancia en su constructor al crear la clase.

Esto introduce el problema de que nada impide a alguien crear otra instancia de la clase que debería ser única y romper la lógica de la aplicación, pero se prefiere frente a singleton debido a [20]:

- Aumenta la testeabilidad.
- Se hace explícito el acoplamiento al pasar explícitamente esa instancia, mientras que el singleton produce un acoplamiento implícito al funcionar de forma global.
- Puede enmascarar el acoplamiento de unos componentes de la aplicación con otros.

4.4. Diseño del agente

Es necesario que la aplicación disponga de una entidad que aparente poseer comportamiento inteligente siendo capaz de enviar solicitudes al usuario, procesar las solicitudes recibidas y controlar la aeronave de forma adecuada para que el usuario perciba el movimiento de la aeronave como si fuera un usuario de la red de vuelo.

Para ello, como se indicó en el análisis del modelo de dominio, necesitamos un agente que controle cada aeronave. El diseño del agente se basa en una máquina de estados finitos (FSM) con transiciones definidas, en esencia permite descomponer el comportamiento general de un agente en pedazos más manejables. Los conceptos más importantes de una FSM son los estados y las transiciones. (Ver el punto 1.1.4 “Diseño de agentes basado en estados” del Capítulo 1: “Inteligencia Artificial” de [21])

Capítulo 4. Diseño

Para diseñar el agente definimos, para cada tipo de vuelo, todos los estados en los que se puede encontrar, las acciones que pueden provocar un cambio de estado, si el cambio es causado por un evento automático o que se cumpla una condición, los cambios que se puedan producir en flags de comportamiento, y el estado al que llega. La lista de posibles estados es la siguiente:

- **Waiting_taxi_out:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** El agente solicita instrucciones de rodaje para rodar la pista.
 - **Acción del usuario:** Instruir rodaje al punto de espera.
 - **Condición para cambiar de estado:** No hay.
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Taxi_out.
- **Taxi_out:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Seguir la ruta asignada por el usuario.
 - **Acción del usuario:** No hay
 - **Condición para cambiar de estado:** Alcanzar la barra de parada. Si está autorizado a entrar en pista o a despegar, continuar hasta terminar la ruta de rodaje.
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Waiting_takeoff (no está autorizado a despegar ni a entrar en pista). Waiting_for_takeoff_lined_up (flag AllowedToEnterRunway = true). Takeoff (flag ClearedForTakeoff = true)
- **Waiting_takeoff:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Solicitar autorización de despegue.
 - **Acción del usuario:** Autorizar despegue o autorizar a entrar y mantener.
 - **Condición para cambiar de estado:** No hay.
 - **Cambios en flags:** Si se le instruye a entrar y mantener: AllowedToEnterRunway = true. Si se le instruye a despegar: ClearedForTakeoff = true.
 - **Nuevo estado:** Taxi_out.

- **Waiting_for_takeoff_lined_up:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Solicitar despegue.
 - **Acción del usuario:** Autorizar despegue.
 - **Condición para cambiar de estado:** No hay.
 - **Cambios en flags:** Reseteo de las flags AllowedToEnterRunway y ClearedForTakeoff.
 - **Nuevo estado:** Takeoff.

- **Takeoff:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Despegar.
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** Alcanzar 100 pies sobre la altitud del aeródromo.
 - **Cambios en flags:** isAirborne = true.
 - **Nuevo estado:** After_takeoff

- **After_takeoff:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Volar al punto de ruta de fin de viento en cara.
 - **Acción del usuario:** No hay
 - **Condición para cambiar de estado:** Alcanzar el punto de ruta de fin de viento en cara.
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Si es VFR: Crosswind. Si es IFR: Waiting_transfer.

- **Waiting_transfer:**
 - **Tipo:** IFR
 - **Acción del agente:** Mantener rumbo actual.
 - **Acción del usuario:** Transferir de frecuencia
 - **Condición para cambiar de estado:** No hay.
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Transferred.

Capítulo 4. Diseño

- **Transferred:**
 - **Tipo:** IFR
 - **Acción del agente:** Mantener rumbo actual
 - **Acción del usuario:** No hay
 - **Condición para cambiar de estado:** Pasan 20 segundos desde que entró en este estado
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Despawn.
- **Crosswind:**
 - **Tipo:** VFR
 - **Acción del agente:** Volar al punto de ruta de fin de viento cruzado.
 - **Acción del usuario:** No hay
 - **Condición para cambiar de estado:** Alcanzar el punto de ruta de fin de viento cruzado.
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Downwind.
- **Downwind:**
 - **Tipo:** VFR
 - **Acción del agente:** Volar al punto de ruta de fin de viento en cola.
 - **Acción del usuario:** No hay
 - **Condición para cambiar de estado:** Alcanzar el punto de ruta de final de viento en cola.
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Request_base_and_final. Si tiene la flag okBaseAndFinal = true, pasa a Base.

- **Downwind:**
 - **Tipo:** VFR
 - **Acción del agente:** Volar al punto de ruta de fin de viento en cola.
 - **Acción del usuario:** Aprobar a virar a base y final.
 - **Condición para cambiar de estado:** Alcanzar el punto de ruta de final de viento en cola.
 - **Cambios en flags:** okBaseAndFinal = true.
 - **Nuevo estado:** Request_base_and_final. Si tiene la flag okBaseAndFinal = true, pasa a Base.
- **Request_base_and_final:**
 - **Tipo:** VFR
 - **Acción del agente:** Mantener rumbo.
 - **Acción del usuario:** Aprobar base y final
 - **Condición para cambiar de estado:** No hay
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Base
- **Base:**
 - **Tipo:** VFR
 - **Acción del agente:** Volar al punto de ruta de fin de base.
 - **Acción del usuario:** No hay
 - **Condición para cambiar de estado:** Alcanzar el punto de ruta de fin de base.
 - **Cambios en flags:** No hay.
 - **Nuevo estado:** Final.

Capítulo 4. Diseño

- **Final:**
 - **Tipo:** VFR
 - **Acción del agente:** Solicitar aterrizaje.
 - **Acción del usuario:** Autorizar aterrizaje. Autorizar toma y despegue. Instruir motor y al aire.
 - **Condición para cambiar de estado:** Altitud menor a 50 pies sobre el aeropuerto.
 - **Cambios en flags:** Si se autoriza el aterrizaje, clearedToLand = true. Si se autoriza la toma y despegue, clearedTouchAndGo = true.
 - **Nuevo estado:** Si se le autoriza el aterrizaje, la toma y despegue o desciende por debajo de 50 pies sobre el aeródromo: Pre_land. Si se le instruye motor y al aire: Go_around.
- **Final_IFR:**
 - **Tipo:** IFR
 - **Acción del agente:** Solicitar aterrizaje
 - **Acción del usuario:** Autorizar aterrizaje. Instruir motor y al aire.
 - **Condición para cambiar de estado:** Altitud menor a 50 pies sobre el aeropuerto.
 - **Cambios en flags:** Si se autoriza el aterrizaje, clearedToLand = true.
 - **Nuevo estado:** Si se le autoriza el aterrizaje o desciende por debajo de 50 pies sobre el aeródromo: Pre_land. Si se le instruye motor y al aire: go_around
- **Pre_land:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Comprobar si clearedToLand = true (IFR y VFR) y la flag de clearedTouchAndGo = true (Sólo VFR).
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** Estado de la flag clearedToLand (IFR y VFR) y la flag clearedTouchAndGo (sólo VFR)
 - **Cambios en flags:** No hay
 - **Nuevo estado:** Si clearedToLand = true (IFR y VFR) o clearedTouchAndGo = true (sólo VFR): Land. Si no: Go_around

- **Land:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Aterrizar
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** Altitud igual a la del aeropuerto.
 - **Cambios en flags:** isAirborne = false.
 - **Nuevo estado:** After_land

- **After_land:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Decelerar y salir por calle de rodaje de salida de pista.
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** touchAndGo = true, o alcanzar la barra de parada de la calle de salida de pista.
 - **Cambios en flags:** No hay
 - **Nuevo estado:** Si touchAndGo = true (sólo VFR): Touch_and_go. Si alcanza la barra de parada de la calle de rodaje de salida de pista: Runway_vacated.

- **Touch_and_go:**
 - **Tipo:** VFR
 - **Acción del agente:** Mantener velocidad.
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** Han pasado 2 segundos desde que entró a este estado.
 - **Cambios en flags:** No hay
 - **Nuevo estado:** Takeoff

- **Go_around:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Volar al punto de ruta de fin de viento en cara.
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** Alcanzar el punto de ruta de final de viento en cara.
 - **Cambios en flags:** No hay
 - **Nuevo estado:** After_takeoff

Capítulo 4. Diseño

- **Runway_vacated:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Solicitar rodaje al estacionamiento
 - **Acción del usuario:** Instruir rodaje al estacionamiento
 - **Condición para cambiar de estado:** No hay.
 - **Cambios en flags:** No hay
 - **Nuevo estado:** Taxi_in
- **Taxi_in:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** Seguir la ruta de rodaje asignada.
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** Pasar 60 segundos desde alcanzar el estacionamiento.
 - **Cambios en flags:** No hay
 - **Nuevo estado:** Despawn
- **Despawn:**
 - **Tipo:** IFR y VFR
 - **Acción del agente:** No hay
 - **Acción del usuario:** No hay.
 - **Condición para cambiar de estado:** No hay.
 - **Cambios en flags:** No hay
 - **Nuevo estado:** No hay

En las siguientes páginas se muestra la representación de la máquina de estados finitos mediante una matriz de transiciones representada en la tabla 4.1, y mediante la figura 4.2 que contiene el diagrama de estados del agente.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

<u>Estado</u>	<u>Tipo</u>	<u>Acción de la IA</u>	<u>Acción del usuario</u>	<u>Evento automático o condición</u>	<u>Cambios en estados internos</u>	<u>Nuevo estado</u>
WAITING_TAXI_OUT	VFR/IFR	Solicitar rodaje a la pista	Instruir rodaje al punto de espera de la pista activa	-	-	TAXI_OUT
TAXI_OUT	VFR/IFR	Seguir la ruta de taxi asignada	-	Alcanzar la stopbar, o continuar hasta terminar la ruta	-	WAITING_TAKEOFF o WAITING_FOR_TAKEOFF_LINED_UP o TAKEOFF
WAITING_TAKEOFF	VFR/IFR	Solicitar despegue	Instruir despegue o instruir entrar y mantener	-	AllowedToEnterRunway = true Si se permite el despegue: clearedForTakeoff = true	TAXI_OUT
WAITING_FOR_TAKEOFF_LINED_UP	VFR/IFR	Solicitar despegue	Instruir despegue	-	Reinicio de las flags allowedToEnterRunway y clearedForTakeoff	TAKEOFF
TAKEOFF	VFR/IFR	Despegar	-	Alcanzar 100 ft sobre el aeródromo	isAirborne = true	AFTER_TAKEOFF
AFTER_TAKEOFF	VFR	Volar al punto de ruta de upwind_end	-	Alcanzar el punto de ruta de upwind_end	-	CROSSWIND
AFTER_TAKEOFF	IFR	Volar al punto de ruta de upwind_end	-	Alcanzar el punto de ruta de upwind_end	-	WAITING_TRANSFER
WAITING_TRANSFER	IFR	Mantener el rumbo actual	Transferir de frecuencia	-	-	TRANSFERRED
TRANSFERRED	IFR/VFR	Mantener el rumbo actual	-	Pasan 20 segundos en este estado	-	DESPAWN
CROSSWIND	VFR	Volar al punto de ruta de crosswind_end	-	Alcanzar el punto de ruta de crosswind_end	-	DOWNWIND
DOWNWIND	VFR	Volar al punto de ruta de downwind_end	-	Alcanzar el punto de ruta de downwind_end	-	BASE o REQUEST_BASE_AND_FINAL
REQUEST_BASE_AND_FINAL	VFR	Solicitar base y final	Aprobar base y final	-	-	BASE
BASE	VFR	Volar al punto de ruta de base_end	-	Alcanzar el punto de ruta de base_end	-	FINAL
FINAL	VFR	Solicitar aterrizaje	Autorizar aterrizaje (setea la flag cleared_to_land = true)	Altitud sobre el aeródromo menor a 50 pies	-	PRE_LAND
FINAL_IFR	IFR	Solicitar aterrizaje	Autorizar aterrizaje (setea la flag_cleared_to_land = true)	Altitud sobre el aeródromo menor a 50 pies	-	PRE_LAND
PRE_LAND	IFR/VFR	Comprueba la flag ClearedToLand y clearedTouchAndGo (Sólo VFR)	-	Cleared_to_land = true	-	LAND
PRE_LAND	IFR/VFR	Comprueba la flag ClearedToLand y clearedTouchAndGo (Sólo VFR)	-	Cleared_to_land = false	-	GO_AROUND

Capítulo 4. Diseño

LAND	IFR/VFR	Aterrizar	-	Altitud = altitud aeródromo	isAirborne = false	AFTER_LAND
TOUCH_AND_GO	VFR	Mantener velocidad	-	Haber pasado 2 segundos desde entrar a este estado	-	TAKEOFF
GO_AROUND	IFR/VFR	Volar al punto de ruta de upwind_end	-	Alcanzar el punto de ruta upwind_end	-	AFTER_TAKEOFF
AFTER_LAND	IFR/VFR	Decelerar y rodar a la taxiway de salida de pista	-	Si touchAndGo, pasar a TOUCH_AND_GO_ Si alcanza el WP de salida de pista pasar a RUNWAY VACATED	-	Si touchAndGo: TOUCH_AND_GO, Si alcanza la barra de parada de la calle de salida de pista: RUNWAY_VACATED
RUNWAY_VACATED	IFR/VFR	Solicitar rodaje a parking	Instruir rodaje a parking	-	-	TAXI_IN
TAXI_IN	IFR/VFR	Seguir la ruta de taxi asignada	-	Pasar 60 segundos desde alcanzar el estacionamiento	-	DESPAWN
DESPAWN	IFR/VFR	-	-	-	-	-

Tabla 4.1. Matriz de transición de estados de la máquina de estados finitos del agente

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

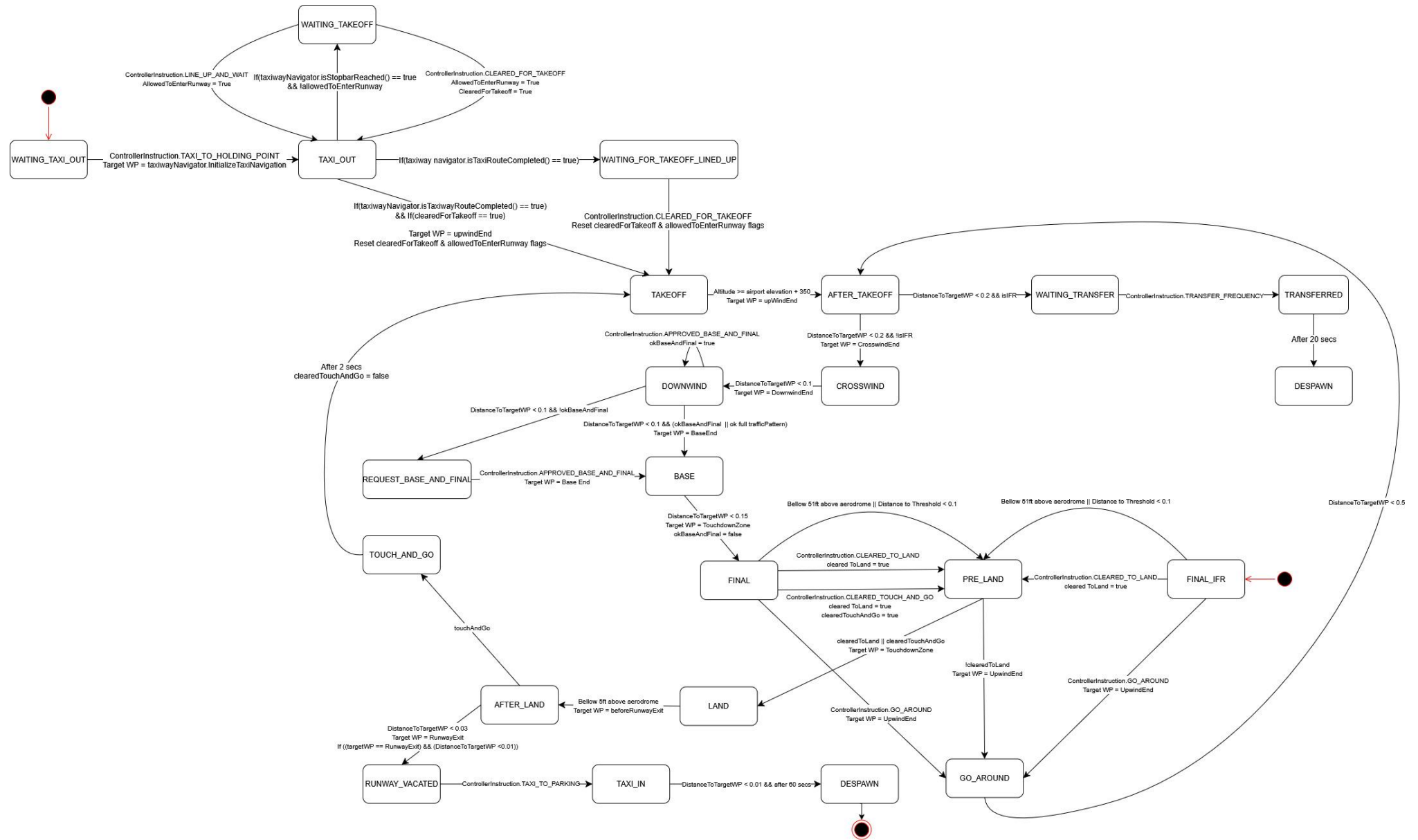


Figura 4.2. Diagrama de transición de estados del agente

4.5. Modelo de datos

Para el almacenamiento de datos externo a la aplicación se ha optado por el uso de JSON como soporte. Su elección se debe a:

- Facilidad de uso tanto para el usuario como para el programador.
- Capacidad de crear rápidamente datos externos de prueba.
- No requiere que el usuario disponga de ningún programa adicional a los disponibles con el sistema operativo.
- Existen bibliotecas en Java que resuelven el problema de tener que serializar y deserializar datos en formato JSON de forma sencilla.

La estructura es la siguiente:

- Partiendo del directorio en que se encuentra el jar ejecutable de la aplicación, debe existir una carpeta llamada “**bbdd**” junto al mismo. Dentro de dicha carpeta nos encontramos con:
 - **Carpeta “aircraftModels”**: Contiene los JSON con los distintos modelos de avión.
 - La información necesaria se puede obtener de la base de datos de Eurocontrol [22]. La estructura de los JSON de aircraftModels es un objeto con los siguientes atributos:
 - **icao**: String con el código ICAO del modelo.
 - **weight**: Int con el peso en kilogramos.
 - **maxSpeed**: Int con la velocidad máxima que puede alcanzar en nudos.
 - **cruiseSpeed**: Int con la velocidad de crucero en nudos.
 - **v2Speed**: Int con la velocidad segura de ascenso tras el despegue.
 - **appSpeed**: Int con la velocidad de aproximación.
 - **initialROC**: Int con la velocidad vertical inicial tras el despegue en pies por minuto.
 - **maxAltitude**: Int con la altitud máxima que puede alcanzar el modelo en pies.
 - **role**: String con el rol que se desea que tome el modelo. Debe ser “IFR” o “VFR”.

- **Carpeta “stats”:** Contiene los JSON con los resultados de las sesiones realizadas por el usuario.
 - La estructura de los JSON de estadísticas es un objeto con los siguientes atributos:
 - **sessionDate:** String con la fecha de la sesión en formato yyyy-MM-dd_HH-mm-ss.
 - **sessionDuration:** String con la duración de la sesión en formato “hh:mm:ss”
 - **airportName:** String con el nombre del aeropuerto. Se añade aquí para evitar tener que consultar la base de datos expresamente para este único dato.
 - **airportICAO:** String ICAO del aeropuerto.
 - **takeoffs:** Int con el número de despegues registrados en la sesión.
 - **landings:** Int con el número de aterrizajes registrados en la sesión.
 - **touchAndGos:** Int con el número de tomas y despegues registradas en la sesión.
 - **dispatchedIFR:** Int número de aeronaves IFR gestionadas por completo.
 - **dispatchedVFR:** Int con el número de aeronaves VFR gestionadas por completo
 - **failureCounters:** Objeto con un atributo por cada regla implementada en el supervisor. Actualmente:
 - **TAXI_ROUTE_INVALID:** Int con el número de veces que se ha enviado una ruta de rodaje inválida a una aeronave.
 - **CONFLICT_AIRBORNE_GROUND:** Int con el número de veces que ha habido un conflicto de una aeronave en el aire con una en tierra.
 - **CONFLICT_AIRBORNE:** Int con el número de veces que ha habido un conflicto entre dos aeronaves en el aire por cercanía.

- **AUTO_GO_AROUND:** Int con el número de veces que un agente ha tenido que ejecutar una maniobra de motor y al aire de forma automática.
- **Carpeta “airports”:** Contiene una carpeta por cada aeropuerto añadido a la base de datos. La información geográfica puede obtenerse mediante la herramienta geojson [23]. El nombre de la carpeta del aeropuerto debe ser el código ICAO ya que es el identificador inequívoco. Dentro de la carpeta de cada aeropuerto tenemos los siguientes JSON:
 - **airportInfo.json:** Describe la información general del aeropuerto. Contiene un objeto con los siguientes atributos:
 - **icao:** String con el ICAO del aeropuerto
 - **name:** String con el nombre del aeropuerto
 - **elevation:** Int con la elevación en pies sobre el nivel del mar.
 - **activeRunway:** Int con el designador de la pista activa preferente.
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos.
 - **designatorOfRunwayExitTaxiway:** String con el identificador de la calle de rodaje utilizada por las aeronaves para abandonar la pista tras aterrizar.
 - **airspaces.json:** Describe los espacios aéreos asociados al aeropuerto. Contiene un objeto con los siguientes atributos:
 - **name:** String con el nombre del espacio aéreo.
 - **airspaceCoords:** Array de objetos, donde cada objeto denota un punto del polígono que forma el espacio aéreo y está formado por:
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos
 - **parkingRamp.json:** Describe la rampa de estacionamientos del aeropuerto. Contiene un objeto con los siguientes atributos:

- **designator:** String con el nombre de la plataforma.
- **airspaceCoords:** Array de objetos, donde cada objeto denota un punto del polígono que forma la rampa de estacionamientos y está formado por:
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos
- **parkings.json:** Array de objetos, cada objeto describe un estacionamiento y sus atributos son:
 - **designator:** String con el designador del estacionamiento.
 - **position:** objeto con los siguientes atributos:
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos
 - **parkingType:** String, sólo puede valer “VFR” o “IFR”. Indica qué tipo de aeronave puede utilizar el estacionamiento, de esta forma separamos aviación comercial de aviación general.
- **runways.json:** Array de objetos, cada uno representando una pista física (cada pista física tiene a su vez 2 designadores de pista, uno en cada dirección). Por convención, las pistas siempre deben llevar el designador más pequeño posible. Además, las coordenadas de la pista se listarán siempre comenzando por la pista con el designador de menor número, por lo que la primera coordenada estará a al lado del designador de menor número y la segunda coordenada al lado del designador de mayor número. La figura 4.3 ilustra esta explicación.
 - **name:** String con el designador de la pista, este debe de ser el de menor número posible de los 2 designadores que tiene cada pista.
 - **heading:** Int con el rumbo de la pista.
 - **longitud:** Int con la longitud en metros.
 - **width:** Int con el ancho en metros.

- **lat1:** Double con la latitud geográfica de la primera esquina de la pista. Norte son valores positivos, sur negativos.
- **lon1:** Double con la longitud geográfica de la primera esquina de la pista. Este son valores positivos, oeste negativos.
- **lat2:** Double con la latitud geográfica de la segunda esquina de la pista. Norte son valores positivos, sur negativos.
- **lon2:** Double con la longitud geográfica de la segunda esquina de la pista. Este son valores positivos, oeste negativos.
- **lat3:** Double con la latitud geográfica de la tercera esquina de la pista. Norte son valores positivos, sur negativos.
- **lon3:** Double con la longitud geográfica de la tercera esquina de la pista. Este son valores positivos, oeste negativos.
- **lat4:** Double con la latitud geográfica de la cuarta esquina de la pista. Norte son valores positivos, sur negativos.
- **lon4:** Double con la longitud geográfica de la cuarta esquina de la pista. Este son valores positivos, oeste negativos.

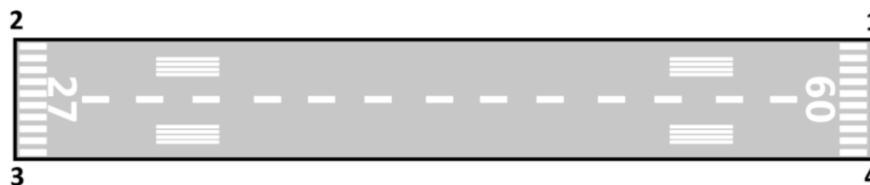


Figura 4.3. Representación del orden de numeración de los vértices de la pista

- **stopbars.json:** Array de objetos. Cada objeto describe una barra de parada mediante los siguientes atributos:
 - **designator:** String con el designador de la barra de parada.
 - **start:** Objeto con las coordenadas de inicio de la barra de parada.
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos
 - **end:** Objeto con las coordenadas de fin de la barra de parada.
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.

- **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos
- **taxiways.json:** Array de objetos, cada uno describe una calle de rodaje mediante los siguientes atributos:
 - **designator:** String con el designador de la calle de rodaje.
 - **centerlineCoords:** Array de posiciones que describen la línea central de la calle de rodaje, cada una con:
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos
- **vfrPoints.json:** Array de objetos, cada uno describe un punto de notificación visual mediante los siguientes atributos:
 - **name:** String con el designador del punto de notificación visual.
 - **latitude:** Double con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Double con la longitud geográfica. Este son valores positivos, oeste negativos

4.6 Conclusiones

En este capítulo se ha explorado el diseño del software de la aplicación. Se ha comenzado por una visión general del sistema mediante el diseño arquitectónico, para luego pasar a elementos más concretos como el diseño del agente que controla las aeronaves y la estructura del sistema de persistencia mediante el modelo de datos. Además, se ha hecho énfasis en los patrones de diseño más utilizados en la aplicación.

Este capítulo se apoya en el capítulo 3 en el que se describe el análisis y es un pilar fundamental para el capítulo 5 en el que se describe la implementación y pruebas.

Capítulo 5. Implementación y pruebas

5.1 Introducción

En este capítulo trataremos los detalles de la implementación de la aplicación, documentando las librerías utilizadas y explicando detalles importantes del desarrollo.

5.2 Entorno de desarrollo

En este apartado se describe el contexto de desarrollo en el que se implementó la aplicación.

5.2.1. Entorno y librerías

- **Windows 10 Home (Versión 22H2).**
- **Eclipse:** El proyecto fue realizado utilizando el IDE Eclipse en su versión 2025-03 (4.35.0).
- **Java:** La versión del JDK utilizado es la 22. El JRE es Java 22.0.2 (OpenJDK 22)
- **JavaFX:** La versión de JavaFX que se referencia en los FXML es la versión 21 [24]
- **Apache Maven:** Se utilizó la versión 3.9.9 de Maven en su variante embebida en el proyecto de Eclipse [25].
- **GSON:** Para el trabajo con archivos en formato JSON se utilizó la versión 2.10.1 de la librería [26]. No recomiendo utilizar versiones más nuevas ya que se producen errores de compilación, presuntamente por incompatibilidad entre la librería y el sistema modular empleado en el proyecto (module-info.java)
- **JUnit:** Para la ejecución de pruebas unitarias, se utilizó la versión 5.13.4 [27].
- **JUnit Jupiter Engine:** Esta dependencia es utilizada por Maven para lanzar las pruebas de JUnit.
- **Maven Surefire:** Este plugin es utilizado por Maven para ejecutar las pruebas mediante el ciclo de vida de Maven [28].
- **GitHub:** Plataforma para el desarrollo de software en equipo que emplea el control de versiones Git. Permite de forma rápida y sencilla el control de versiones del proyecto y

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

acceder a él desde cualquier lugar al estar alojado en la nube. Durante su desarrollo el proyecto se alojó en un repositorio privado.

5.2.2. Gestión de dependencias en Maven

Para la gestión de dependencias del proyecto se ha empleado Maven. Con esta herramienta se indican las dependencias necesarias en el archivo “pom.xml”, las cuales han sido las siguientes:

```
<dependencies>
  <!-- JAVAFX (Para la UI) -->
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>21</version>
  </dependency>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>21</version>
  </dependency>
  <!-- Gson (Para la BBDD usando archivos GSON) -->
  <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
  <!-- Hay un bug en Eclipse/VS Code con vers. superiores a 2.10.1 Gson, son incapaces de resolver el paquete
  ver: https://github.com/google/gson/issues/2839 y https://github.com/google/gson/issues/2679 -->
  <dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.10.1</version>
  </dependency>
  <!-- JUnit para la realización de pruebas -->
  <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.13.4</version>
    <scope>test</scope>
  </dependency>
  <!-- JUnit Jupiter Engine para poder ejecutar las pruebas -->
  <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.13.4</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Capítulo 5. Implementación y pruebas

Para que Maven pueda ejecutar las pruebas es necesario añadir también el plugin Maven Surefire:

```
<!-- Maven Surefire para lanzar las pruebas mediante el ciclo de vida de Maven (tests) -->
<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-plugin -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>3.5.3</version>
</plugin>
```

5.3 Implementación

5.3.1 Organización del código fuente

Siguiendo el diseño de la aplicación, el código fuente se dividió en paquetes organizados según la familia a la que pertenecen. A rasgos generales, los paquetes de la familia “es.uned.pfg.adcsim.data” se encargan de la importación y exportación de datos externos a la aplicación. Los paquetes de la familia “es.uned.pfg.adcsim.model” representan los distintos modelos de dominio de la aplicación. El listado completo de paquetes en los que está dividido el código fuente es el siguiente:

- **es.uned.pfg.adcsim.app:** Aquí se encuentra la clase principal que hace de punto de entrada a la aplicación.
- **es.uned.pfg.adcsim.controller:** Contiene las clases controladoras de las diferentes vistas que forman la interfaz gráfica.
- **es.uned.pfg.adcsim.data.service:** Siguiendo el patrón estrategia, aquí están las interfaces que deben implementar las clases que se quieran dedicar a la entrada/salida de datos de la aplicación.
- **es.uned.pfg.adcsim.data.service.impl:** Aquí se encuentran las diferentes implementaciones de los servicios de entrada/salida de datos
- **es.uned.pfg.adcsim.model.ai:** Contiene los enum que forman el modelo del agente. Dictan los posibles estados en los que puede estar el agente, posibles solicitudes que puede realizar, y posibles instrucciones que puede recibir del controlador.
- **es.uned.pfg.adcsim.model.aircraft:** Contiene las clases relacionadas con el modelo de dominio de las aeronaves.
- **es.uned.pfg.adcsim.model.airport:** Contiene las clases relacionadas con el modelo de dominio de los aeropuertos. Cada una de ellas describe una parte del mismo.

- **es.uned.pfg.adcsim.model.session:** Aquí se encuentran las clases que describen los datos que se obtienen de una sesión de entrenamiento. Se utilizan especialmente por gson para la serialización/deserialización.
- **es.uned.pfg.adcsim.service.ai:** Aquí están las clases relacionadas con la lógica de negocio del agente que controla las aeronaves.
- **es.uned.pfg.adcsim.simulation:** Contiene las clases relacionadas con el motor de simulación, tanto el propio motor de simulación como el supervisor de infracciones. También contiene las interfaces que deben implementar las clases que deseen suscribirse a actualizaciones del motor de simulación o al supervisor de infracciones siguiendo el patrón observador.
- **es.uned.pfg.adcsim.simulation.traffic:** Contiene lo relacionado con la lógica de inyección de aeronaves en la simulación.
- **es.uned.pfg.adcsim.util:** Aquí se encuentran las clases utilitarias empleadas por el resto de clases de la aplicación.
- **es.uned.pfg.adcsim.view:** Contiene las clases relacionadas con elementos de la GUI que no son implementados mediante FXML, el renderizado del radar en el canvas de JavaFX y la implementación de la ventana de diálogo para la selección de la ruta de rodaje.

Capítulo 5. Implementación y pruebas

Las figuras 5.1 y 5.2 muestran la estructura de paquetes y clases del código:

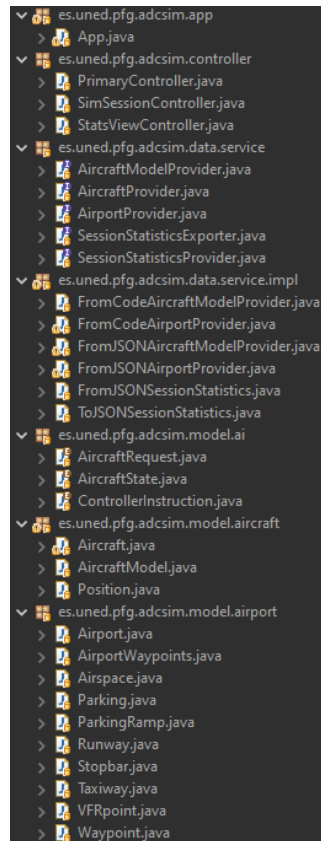


Figura 5.1. Estructura de paquetes y clases

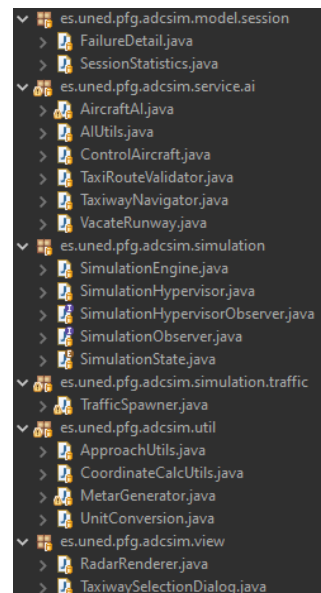


Figura 5.2. Estructura de paquetes y clases (continuación)

5.3.2. Implementación de requisitos funcionales

RF-01: Inicializar una sesión de simulación

Para la implementación de este requisito funcional se implementó el punto de entrada a la aplicación ubicado en la clase `es.uned.pfg.adcsim.app.App`. El punto de entrada consiste en el método `main` que se limita a invocar al método “`launch`” de JavaFX. Para ello la clase `App` debe heredar de la clase `Application` del paquete `javafx.application`.

Al ejecutar el método `launch` JavaFX realiza internamente varias cosas, entre las que destaca:

- Se llama al método `init` en el hilo principal de la aplicación, el cual puede ser sobrescrito por el programador para inicializar datos y configuraciones.
- Se crea el hilo principal de JavaFX, el cual es el único que puede manipular la GUI.
- Se crea automáticamente una instancia de `Stage` que representa la ventana principal de la aplicación y se llama al método `start` pasando dicha `Stage`.
- En el método `start` se implementa el código relacionado con la ventana principal de la aplicación (Título, tamaño, acción al cerrar, ...).

Una vez creada la ventana y establecidas las propiedades en el método `start`, la clase controladora `es.uned.pfg.adcsim.controller.PrimaryController` se encarga de registrar las acciones del usuario. Para ello utiliza referencias a los elementos FXML apuntados por su `id` definido en el FXML con la etiqueta `@FXML`. Así, al pulsar en el botón “Nueva Sesión” se llama al método `handleNewSession`, el cual pide al método estático de la clase `App` que cambie el FXML raíz al correspondiente a la vista de sesión de entrenamiento.

JavaFX al cambiar de vista (de FXML) carga la vista, carga el controlador y llama automáticamente al método `initialize`. Por lo tanto, al terminar de cargar la vista de sesión de entrenamiento, tras cargar la clase controladora inicializando los campos de clase, lo primero que hace es llamar al método `initialize` de `es.uned.pfg.adcsim.controller.SimSessionController`. Este método se encarga de inicializar los elementos de la GUI que deben ser cargados en tiempo de ejecución, así como los listeners para registrar las acciones del usuario en los distintos elementos de la GUI. La instancia encargada de renderizar el radar se crea aquí pasándole una referencia del `graphicsContext` de JavaFX y el número de puntos que forman la estela en el radar.

Capítulo 5. Implementación y pruebas

Una vez inicializado todo, el usuario puede seleccionar el aeródromo de la lista, la meteorología y pulsar en el botón “Iniciar sesión”, el cual llama al método `handleStart` del controlador. Este método se encarga de:

- Comprobar si existe una sesión activa, si no, la inicia.
- Inicia el temporizador de tiempo de sesión transcurrido.
- Deshabilita los `comboBox` de aeródromos y meteorología.
- Recupera el aeródromo seleccionado del proveedor y obtiene las coordenadas geográficas en las que se encuentra para luego centrarlo en la pantalla radar.
- Crea una instancia del motor de simulación (`es.uned.pfg.adcsim.simulation.SimulationEngine`), se suscribe a sus cambios y a los de la instancia del supervisor que crea el motor de simulación al ser inicializado. Recordemos que para que pueda suscribirse esta clase controladora debe implementar las interfaces `es.uned.pfg.simulation.SimulationObserver` y `es.uned.pfg.simulation.SimulationHypervisorObserver`.
- Llama al método `start` del motor de simulación para iniciar la simulación.
- Llama al método privado `redrawRadar` encargado de centralizar las llamadas al método encargado de redibujar el canvas de JavaFX que actúa de pantalla radar.

Con esto, la simulación ha quedado iniciada.

RF-02: Renderizado de la pantalla radar

Este requisito funcional lo cubre la clase en `es.uned.pfg.adcsim.view.RadarRenderer`. Como vimos en RF-01 la instancia encargada del renderizado se crea en el método `initialize` de `SimSessionController`, llamado por JavaFX al completar la carga de la vista y el controlador. Su constructor se limita a almacenar las referencias al `GraphicsContext` de JavaFX, la longitud de la traza radar y cargar la fuente personalizada.

Esta clase contiene además dos métodos: El método `findAircraftAtPosition` que devuelve la aeronave, si la hay, en la posición del canvas pasada como parámetro, y el método `render` que implementa el requisito funcional. Este método se limita a dibujar en el canvas de JavaFX de la vista

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

de sesión de simulación la pantalla radar mediante los métodos de dibujado de JavaFX, de este modo, el método se encarga del dibujado de:

- Pistas de aterrizaje. Incluyendo sus designadores
- Espacios aéreos.
- Plataforma de estacionamientos.
- Calles de rodaje.
- Barras de parada.
- Estacionamientos.
- Puntos de notificación visual.
- Aeronaves. Incluyendo el icono, línea de rumbo con longitud proporcional a la velocidad, etiquetas y trazas.

Con esto, todos los elementos de la pantalla radar están representados y el requisito funcional queda cubierto.

RF-03: Selección de aeronaves en la pantalla radar

Este requisito funcional lo cubre el otro método de la clase RadarRenderer que vimos en RF-02, findAircraftAtPosition y la clase controladora SimSessionController mediante un listener que está a la escucha de si se hace clic en el canvas, reportando la posición x/y donde se hizo clic. El método findAircraftAtPosition comprueba si existe un avión en esa posición realizando transformaciones del espacio de coordenadas de la pantalla al espacio de coordenadas geográficas y, si existe, devuelve la instancia Aircraft encontrada en esa posición al listener y llama al método selectAircraft de misma clase SimSessionController, actualizando el texto informativo sobre qué aeronave ha sido seleccionada y la aeronave marcada en el controlador como actualmente seleccionada.

RF-04: Representación de las peticiones del agente en la interfaz gráfica

Una vez seleccionada la aeronave con mediante el proceso descrito en RF-03, el método selectAircraft de SimSessionController llama al método updatePossibleActions de la misma clase.

Capítulo 5. Implementación y pruebas

Este método se encarga de añadir al comboBox de acciones las instrucciones que el controlador puede enviar al agente en función del estado en que se encuentra.

De esta forma, el usuario dispone de una lista actualizada de las acciones que puede enviar al agente.

RF-05: Actualización continua de la simulación

Al iniciar el motor de simulación tal y como se describe en RF-01, una de las instrucciones que realiza el constructor de SimulationEngine es crear un AnimationTimer del paquete javafx.animation, para ello implementamos el método handle para que en cada llamada actualice el instante de tiempo en que es llamado. Luego, SimSessionController puso en marcha el AnimationTimer llamando al método start de SimulationEngine. Una vez puesto en marcha el motor de simulación, en cada fotograma se llama al método handle de AnimationTimer que devuelve el tiempo actual, luego se calcula el tiempo transcurrido desde la última actualización y se pasa al método update de la propia clase SimulationEngine.

El método update llama con cada aeronave al método updateAircraftPosition de la propia clase SimulationEngine para que se recalcule la posición de la aeronave, además, llama al método update del agente para que controle la aeronave y compruebe si se han cumplido las condiciones para cambiar de estado.

Una vez actualizadas las aeronaves, el método update de SimulationEngine itera en la lista de aeronaves en la simulación buscando si algún agente está en el estado “despawn” eliminando las aeronaves asociadas a los agentes con dicho estado y actualiza los contadores de IFR y VFR en la simulación.

Eliminadas las aeronaves marcadas para borrado, el método update de SimulationEngine inyecta nuevas aeronaves. Primero genera un número aleatorio del 0 al 10, si este número es menor a 1 inyecta una nueva aeronave VFR mediante el método injectNewVFR Aircraft de la propia clase SimulationEngine. Tras esto, vuelve a generar otro número aleatorio del 0 al 10, si el número es menor a 1 inyecta una nueva aeronave IFR en final mediante el método injectNewIFR Aircraft On Final. Finalmente, vuelve a generar otro número aleatorio de 0 a 10 y si el número es menor a 1 inyecta una nueva aeronave IFR en un estacionamiento aleatorio mediante el método injectNewIFR Aircraft On Parking.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

En cada llamada a los métodos para inyectar nuevas aeronaves se comprueba primero que no se haya superado el número de aeronaves máximas permitidas a la vez en la sesión.

Lo siguiente que realiza el método update de SimulationEngine llama al supervisor para que compruebe si algún avión de la lista de aviones actualmente en la simulación incumple las reglas.

Por último, notifica a las clases observadoras de SimulationEngine que el estado de la simulación ha cambiado.

RF-06: Aeronaves controladas por un agente

Al inyectar nuevas aeronaves en la simulación primero se instancia la aeronave, y luego se instancia un agente que se asocia a la aeronave que debe controlar.

En cada actualización del motor de simulación se llama al método update de la clase AircraftAI, el cual se encarga de llamar a los métodos encargados de controlar la aeronave. Para el rodaje llama al método updateTaxiNavigation encargado de indicar al agente cuál es el siguiente punto de la ruta de rodaje al que debe dirigirse. Para el control de la aeronave en vuelo se utiliza el método controlAircraft el cual, dependiendo del estado en que se encuentra el agente, llama a un método al método encargado de gestionar esa fase del vuelo.

RF-07: El usuario debe poder instruir a las aeronaves a realizar alguna acción

En RF-04 se describió cómo el usuario puede visualizar las acciones que puede enviar al agente que controla la aeronave seleccionada. Si el usuario hace clic en cualquiera de esas acciones se dispara el listener asociado al comboBox de acciones, el cual envía la instrucción seleccionada al método handleControllerInstruction.

Este método se encarga, de forma general, de enviar la instrucción al método processInstruction del agente.

De forma particular, en caso de que la instrucción sea de rodaje al estacionamiento o al punto de espera de la pista, invoca la ventana para que el usuario escoja la ruta de rodaje. Una vez escogida,

Capítulo 5. Implementación y pruebas

es enviada al agente para que la compruebe utilizando el método `setTaxiwayRoute`. Si la ruta no es correcta el agente la rechaza y el usuario puede volver a intentar enviar la ruta de rodaje, si es correcta la almacena. Una vez el agente tiene la ruta de rodaje, se llama al método `processInstruction` del agente.

Tras esto, la clase controladora `SimSessionController` actualiza las posibles acciones del `comboBox` y las solicitudes pendientes de los agentes.

El método `processInstruction` de la clase `AircraftAI` encargada de gestionar los agentes comprueba primero en qué estado está el agente y, en función del estado, procesa la instrucción, realiza los cambios necesarios en las variables internas del agente y cambia de estado al agente si es necesario.

RF-08: El sistema debe permitir al usuario elegir la ruta de rodaje que desea instruir al tráfico

Como se comentó en RF-07, al escoger una acción relacionada con el rodaje la clase controladora de la vista de sesión de entrenamiento `SimSessionController` crea y abre una ventana modal para que el usuario pueda escoger la ruta de rodaje. Esta ventana está implementada mediante la clase `TaxiwaySelectionDialog` del paquete `es.uned.pfg.adcsim.view`.

Cuando el usuario pulsa el botón `Ok` de la ventana, la clase controladora de la vista de sesión de entrenamiento envía la ruta escogida por el usuario al agente utilizando el método `setTaxiwayRoute`. Este procesa la ruta recibida y, si es correcta, la almacena, si no, la rechaza y el usuario debe volver a introducir la ruta de rodaje.

RF-09: El sistema debe ser capaz de validar la ruta de rodaje recibida por el usuario

Al final de RF-08 comentamos que la ruta de rodaje es procesada para comprobar si es correcta. Para esto el agente dispone del método `isValidTaxiwayRoute` de la clase `TaxiwayRouteValidator`.

Este método comprueba si la ruta cumple los siguientes requisitos:

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

- Si el agente está en el estado “taxi_in”, la primera calle de rodaje de la ruta debe coincidir con la calle de rodaje marcada en la instancia de aeropuerto como la calle de rodaje de salida de pista.
- Si el agente está en el estado “taxi_out”, la primera calle de rodaje de la ruta debe ser una que esté conectada con la plataforma de estacionamientos. Es decir, se comprueba si alguno de los puntos que forman la primera calle de rodaje de la ruta está dentro de la rampa de estacionamiento mediante el algoritmo de ray casting.
- No hay calles de rodaje repetidas. Con esto se busca eliminar bucles en la ruta.
- Para cada par de calles de rodaje de la ruta se comprueba que estén conectadas. Para ello se comprueba si el último o el primer punto de la calle de rodaje anterior está suficientemente cerca de primer o último punto de la calle de rodaje siguiente.
- Si el agente está en el estado “taxi_in”, la última calle de rodaje debe acabar en la plataforma de estacionamientos.
- Si el agente está en el estado “taxi_out”, la última calle de rodaje debe ser una cuyo designador coincida con el de alguna de las barras de parada del aeropuerto.

Si cumple los requisitos, la ruta es válida y se almacena en el agente que controla la aeronave.

RF-10: El sistema debe contar con un agente supervisor que compruebe si se incumplen alguna de las reglas definidas como malas prácticas en el control de tráfico aéreo

La clase encargada de implementar este requisito funcional es SimulationHypervisor. Esta clase es instanciada por la clase SimulationEngine y es la única clase autorizada a instanciarla.

En SimulationHypervisor se define la lista de reglas que el usuario no debe incumplir, junto con la lógica encargada de comprobar si se incumple alguna de ellas. En cada ciclo de simulación SimulationEngine llama al método checkRules de SimulationHypervisor, que llama sucesivamente a los métodos que implementan la comprobación de cada regla.

Si se incumple alguna regla, se actualiza el mapa que lleva el conteo de fallos.

Capítulo 5. Implementación y pruebas

RF-11: La interfaz gráfica debe proporcionar al usuario un feedback inmediato cuando cometa un error

Cuando alguno de los métodos de la clase `SimulationHypervisor` encargados de comprobar si se incumple alguna regla detecta una infracción se llama al método `notifyHypervisorObservers`. Este método se encarga de notificar a las clases suscritas a `SimulationHypervisor` llamando a su método `onFailure`.

La clase controladora de la vista de sesión de simulación `SimSessionController` está suscrita implementando la interfaz `SimulationHypervisorObserver`, por lo que implementa el método `onFailure`. Este método recibe el texto formateado de la regla incumplida y el avión implicado en la infracción de la regla, mostrándolo al usuario en el área de texto de fallos cometidos.

RF-12: Carga modular de aeropuertos y modelos de avión

Al instanciarse el controlador de la vista de sesión de entrenamiento, `SimSessionController`, se instancia el proveedor de aeropuertos a través de la interfaz `AirportProvider`. En el estado final del proyecto la implementación encargada de proveer los aeropuertos es `FromJSONAirportProvider`.

Esta clase se encarga de comprobar que la estructura de directorios del almacenamiento persistente es correcta y existen los archivos JSON necesarios para la carga completa del aeropuerto: `airportInfo.json`, `runways.json`, `vfrPoints.json`, `airspace.json`, `parkings.json`, `taxiways.json`, `stopbars.json` y `parkingRamp.json`. Para cada uno de estos archivos se realizan los siguientes pasos:

- Se comprueba que existe el archivo.
- Se recupera el contenido del JSON mediante la librería `gson`.
- Se comprueba que el contenido es correcto y no hay valores ilegales o no permitidos en el JSON.
- Si el JSON contiene una lista de objetos, se comprueba que la lista es correcta y no está mal formada.

Una vez se han recuperado todos los datos se llama al constructor de la clase `Airport` para instanciar el aeropuerto, se añade el aeropuerto al mapa de aeropuertos cargados y se añade el ICAO del aeropuerto a la lista de aeropuertos cargados en la propia clase `FromJSONAirportProvider`.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Si se detecta alguna excepción durante la carga, se captura, se informa y se interrumpe la carga.

Una vez están los aeropuertos cargados, la clase `SimSessionController` puede llamar al método `getAirport` o `getAirports` de la implementación concreta a través de la interfaz `AirportProvider` para obtener los aeropuertos cargados en memoria.

Para la carga de modelos de avión se instancia el proveedor de modelos de avión en la clase `SimulationEngine` de forma similar a como se hizo para el proveedor de aeropuertos y se accede a los métodos de la implementación concreta a través de la interfaz `AircraftModelProvider`.

La implementación de la clase `FromJSONAircraftModelProvider` es similar a la vista anteriormente con `FromJSONAirportProvider`. Se comprueba que exista el directorio donde deben estar los JSON, se comprueba que existan archivos JSON en el directorio y se los deserializa utilizando `gson`, informando si en algún caso el formato del JSON es incorrecto.

Realizadas todas las comprobaciones y deserializados todos los JSON de la carpeta “`bbdd/aircraftModels`”, se devuelve la lista de modelos cargados.

RF-13: Guardado de los resultados de la sesión de entrenamiento en el almacenamiento persistente

El comportamiento del botón “Salir” en la vista de sesión de entrenamiento está definido por el método `handleExit` de la clase controladora de la vista de sesión de entrenamiento, `SimSessionController`. Cuando se activa este botón y se llama al método, la clase controladora invoca el proceso de guardado de resultados de la sesión de entrenamiento.

En este proceso, calcula el tiempo transcurrido desde que se inició la sesión, obtiene el ICAO y nombre del aeropuerto actualmente seleccionado y recupera las estadísticas almacenadas en el supervisor. Por último, obtiene la fecha y hora actual.

Una vez que tiene todos los datos necesarios crea una instancia de la clase `SessionStatistics`. Finalmente, llama al método `export` de la clase que implementa la exportación de estadísticas `ToJSONSessionStatistics` a través de la interfaz `SessionStatisticsExporter`.

Capítulo 5. Implementación y pruebas

La clase ToJSONSessionStatistics se encarga de comprobar si existe el directorio “bbdd/stats”, si no existe lo crea. Posteriormente genera el nombre del archivo utilizando la fecha y hora actual, serializa el objeto SessionStatistics a un String utilizando la librería gson y almacena el contenido del String en un nuevo archivo que tiene de nombre la fecha y hora obtenida antes y extensión json. Si durante este proceso de guardado se produce alguna excepción, esta es capturada y se informa del error.

RF-14: Carga de los resultados de las sesiones de entrenamiento realizadas desde el almacenamiento persistente

La clase encargada de implementar este requisito funcional es FromJSONSessionStatistics. Su funcionamiento es análogo al que vimos para las clases FromJSONAirportProvider y FromJSONAircraftModelProvider en RF-12.

La implementación comprueba que existe el directorio “bbdd/stats” y comprueba si hay archivos con extensión json en el directorio, si no los hay no devuelve nada. Si los hay, recupera uno a uno los archivos, deserializa los JSON utilizando la librería gson creando una instancia de la clase SessionStatistics por cada archivo JSON, los añade a la lista de sesiones recuperadas y devuelve la lista al completar la lectura de todos los archivos JSON encontrados.

Si se produce una excepción durante la lectura, se captura y se informa de lo ocurrido.

RF-15: El usuario debe poder revisar las estadísticas de sesiones de control realizadas

Para satisfacer este requisito funcional se implementó un botón en el menú principal de la aplicación que cambie la raíz de la escena a la vista de estadísticas. Esto abre la ventana de estadísticas de sesión y carga el controlador de la vista StatsViewController.

Al crearse StatsViewController se instancia la clase FromJSONSessionStatistics que implementa la interfaz SessionStatisticsProvider, descrita en RF-14, obteniendo así la lista de sesiones realizadas guardadas en el almacenamiento externo. Además, JavaFX llama automáticamente al método initialize que se encarga de configurar las tablas de la vista de estadísticas de sesiones realizadas y rellenar la tabla de sesiones.

Para implementar la tabla donde se muestran en detalle los fallos cometidos en la sesión se hace uso de un listener implementado en la tabla que lista las sesiones de forma que, al seleccionar una sesión de dicha tabla, recupera los pares clave-valor del mapa de fallos cometidos y los muestra en la tabla de detalles de fallos cometidos.

5.4. Flujo de desarrollo del proyecto

En la práctica los sprint de desarrollo descritos en el capítulo de planificación del proyecto, y llevados a cabo entre abril y agosto, han sido aproximadamente los siguientes:

- Construcción de la estructura base de la aplicación de JavaFX sobre Maven en Eclipse.
- Implementación de la vista principal.
- Terminar la implementación del requisito funcional RF-01: Inicializar una nueva sesión de simulación
- Primera iteración de la implementación del requisito funcional RF-02: Renderizado de la pantalla radar, orientado a tener una funcionalidad básica.
- Segunda iteración de la implementación del renderizado de la pantalla de radar orientado a añadir más funcionalidad: zoom, panning, ...
- Primera refactorización de código. La clase `SimSessionController` tenía demasiada responsabilidad, se separaron funcionalidades a nuevas clases: El redibujado de la pantalla radar se llevó a una nueva clase `"RadarRenderer"`, el método auxiliar para transformar coordenadas geográficas a coordenadas del canvas de la pantalla radar fue movido a una clase utilitaria propia `"CoordinateCalcUtils"` en el paquete `es.uned.pfg.adcsim.util`, el método utilitario para generar METAR fue movido también a una clase propia `"MetarGenerator"` en el paquete de utilidades.
- Primera iteración del sistema de inyección de aeronaves. Esta primera aproximación sirve de toma de contacto y sólo se implementaron 2 posibles puntos de inyección: Larga final o en la cabecera de la pista activa.
- Segunda refactorización del proyecto. Fue motivada al detectar que la clase proveedora temporal para pruebas durante el desarrollo `"FromCodeAircraftProvider"` tenía mezclados los datos de los aviones simulando que provienen de una base de datos, con la lógica de inyección de aeronaves en la simulación. Esto provocó una reorganización de código con la

Capítulo 5. Implementación y pruebas

introducción del patrón strategy/provider, se separaron la lógica relacionada con los modelos de avión de los aviones en sí, ...

- Implementación de los parkings.
- Segunda iteración del sistema de inyección de aeronaves. Esta está centrada a evitar algunos conflictos que se pueden producir al inyectar una aeronave, especialmente sistemas que detectan si ya hay una aeronave en final para no inyectar otra. Esta implementación volverá a ser revisada en ciclos de desarrollo posteriores cuando sea implementado el agente y pasará a estar gobernada por el estado en que se encuentren las inteligencias artificiales de los aviones actualmente en la simulación.
- Primera iteración de la implementación del agente. Se definieron los posibles estados en los que se pueden encontrar el agente, las posibles acciones o peticiones que puede hacer un agente en cada estado, las posibles peticiones que puede hacer el usuario al agente, la matriz de transiciones para la máquina de estados finitos que forma el agente, la clase "AirportWaypoints" que representa los puntos de ruta que debe seguir el agente al controlar la aeronave en función del estado en que se encuentra y la clase "AircraftAI" que implementa la máquina de estados finitos diseñada antes y otros métodos utilizados para controlar la aeronave asociada al agente, procesar instrucciones del controlador, actualizar el estado del agente si se han cumplido los objetivos y obtener solicitudes pendientes.
- Revisión del flujo de simulación. Se empleó este ciclo de desarrollo para optimizar el proceso de inicialización, funcionamiento del ciclo de simulación y realizar un análisis retrospectivo para documentar correctamente el funcionamiento del núcleo del simulador de cara al mantenimiento en el futuro.
- Tercera iteración de la implementación de la pantalla radar. En este caso, se enfocó el esfuerzo en implementar nuevos elementos como la selección de aeronaves en la pantalla radar o poder hacer zoom mediante scroll con la rueda del ratón. Este ciclo fue mucho más costoso de lo que se previó al inicio del mismo debido a problemas relacionados con JavaFX: IndexOutOfBoundsException al seleccionar una acción en el comboBox de acciones ya que JavaFX intenta modificar el comboBox mientras está activo, y el no restablecimiento del promptText del comboBox de acciones tras seleccionar una acción debido a que JavaFX no es capaz de restablecer el promptText si el comboBox tiene elementos seleccionados y tiene el foco.
- Segunda iteración de la implementación del agente. Asignación de agentes a aeronaves. Refinamiento del funcionamiento y comportamiento del agente. Implementación de

métodos auxiliares del agente. Documentación explicativa de cara al futuro sobre cómo funciona el agente en la aplicación, quién llama al método update del agente, rutina de control de la aeronave según el estado actual, etc. de cara al mantenimiento en el futuro.

- Implementación de la eliminación de aeronaves ya atendidas. El motor de simulación ahora sabe qué aeronaves puede eliminar de la lista de aeronaves actualmente en la simulación porque su agente se encuentra en el estado “DESPAWN”.
- Implementación del sistema de inyección aleatoria de aeronaves basado en probabilidades.
- Implementación de calles de rodaje y barras de parada.
- Cuarta iteración de la implementación de la pantalla radar. Dibujado de las calles de rodaje, estacionamientos, barras de parada y uso de fuentes personalizadas.
- Implementación del cuadro de diálogo para la selección de la ruta de rodaje.
- Implementación de la plataforma de estacionamientos. También se implementó el algoritmo de ray casting para saber si un punto está dentro o fuera de un polígono, útil para saber si un punto está dentro o fuera de la plataforma de estacionamientos.
- Implementación de la validación de la ruta de rodaje enviada al agente.
- Implementación de la navegación del agente siguiendo la ruta de rodaje. Centrado en la implementación para cuando el agente está en el estado “TAXI_IN”.
- Refactorización de la clase AircraftAI. Fue motivada debido a que la clase había crecido excesivamente para trabajar cómodamente con ella (1100 líneas de código aproximadamente), por lo que se decidió dividir la responsabilidad de la clase en las siguientes: “ControlAircraft” encapsula el movimiento de la aeronave por parte del agente, “TaxiwayNavigator” gestiona la navegación durante el taxi, “TaxiRouteValidator” valida la ruta de rodaje enviada por el usuario, “VacateRunway” gestiona la salida de pista tras el aterrizaje por la calle de rodaje indicada en la base de datos hasta detenerse en la barra de parada a la espera de recibir la ruta de rodaje, “AIUtils” contiene los métodos utilitarios utilizados por las otras clases del paquete del agente es.uned.pfg.adcsim.service.ai.
- Implementación de la navegación del agente siguiendo la ruta de rodaje para el estado “TAXI_OUT”.
- Implementación de la base de datos de aeronaves en formato JSON. También se implementaron los proveedores y se cambió fácilmente del proveedor de datos desde código al nuevo gracias al patrón strategy/provider.

Capítulo 5. Implementación y pruebas

- Implementación de la base de datos de aeropuertos en formato JSON. Análogo al ciclo anterior, pero más extenso y complejo debido al mayor volumen de datos que hay que manejar.
- Implementación del sistema de evaluación del usuario. Se centró en la implementación del supervisor mediante la clase “SimulationHypervisor”, definiendo las reglas que el usuario no debe incumplir, conteo de distintas estadísticas como número de despegues, número de aterrizajes, etc.
- Implementación del guardado de estadísticas de la sesión en JSON.
- Implementación de la vista de estadísticas de sesiones realizadas.

5.4 Pruebas

En este apartado se describen los tipos de pruebas que se han llevado a cabo en el proyecto, desde las utilizadas para probar partes de la aplicación de forma aislada, hasta las encargadas de probar la aplicación completa.

5.4.1 Pruebas unitarias

Este tipo de pruebas permite probar elementos individuales de la aplicación como puede ser un paquete o una clase completa. Su principal finalidad es detectar errores en la implementación.

Las pruebas son implementadas siguiendo la estructura del proyecto Maven, por lo que se encuentran en la carpeta “adcsim/src/test/java” del proyecto. El ciclo de vida de Maven facilita la ejecución automática de las pruebas, aunque también se puede solicitar a Maven ejecutar únicamente las pruebas. Para la ejecución de las pruebas Maven utiliza JUnit [27] y el plugin Surefire [28].

5.4.2 Pruebas de integración

Además de probar cada parte del sistema por separado, es necesario probar el funcionamiento en conjunto de todas ellas. Estas pruebas, realizadas tras las pruebas unitarias, se realizan con la finalidad de buscar errores producidos cuando cada parte se encuentra integrada en el sistema.

El principal inconveniente de estas pruebas es que normalmente requiere realizar las pruebas a mano, ejecutando la aplicación, por lo que consumen más tiempo que las pruebas unitarias. Para realizarlas generalmente se lanza la aplicación buscando ejercitar todas las partes del sistema buscando la mayor cobertura posible.

Al terminar de implementar una nueva parte de la aplicación, tras una refactorización, o después de cualquier otra modificación marcada como objetivo de desarrollo a corto plazo, siempre se realizaron estas pruebas de integración para comprobar si se había cumplido el objetivo buscado.

5.4.3 Pruebas de validación

Estas pruebas tienen como finalidad comprobar que el programa tiene toda la funcionalidad que se ha indicado en la especificación de requisitos y pueden descubrir carencias en la aplicación que hayan sido pasado inadvertidos durante el desarrollo incluso después de realizar pruebas unitarias o pruebas de integración.

Las pruebas de validación escogidas han sido la comprobación directa de los requisitos funcionales.

La tabla 5.1 contiene las pruebas de validación realizadas a la aplicación y el resultado. Posteriormente se muestran un conjunto de capturas de pantalla para ilustrar el aspecto visual de la aplicación en diferentes momentos de ejecución.

Capítulo 5. Implementación y pruebas

Identificador	Descripción	Resultado
PV-01	Inicializar una sesión de simulación	OK
PV-02	Renderizado de la pantalla radar	OK
PV-03	Selección de aeronaves en la pantalla radar	OK
PV-04	Representación de las peticiones de los agentes en la interfaz gráfica	OK
PV-05	Actualización continua de la simulación	OK
PV-06	Aeronaves controladas por un agente	OK
PV-07	El usuario debe poder instruir a las aeronaves a realizar alguna acción	OK
PV-08	El sistema debe permitir al usuario elegir la ruta de rodaje que desea instruir al tráfico	OK
PV-09	El sistema debe ser capaz de validar la ruta de rodaje recibida por el usuario	OK
PV-10	El sistema debe contar con un agente supervisor que compruebe si se incumplen alguna de las reglas definidas como malas prácticas en el control de tráfico aéreo	OK
PV-11	La interfaz gráfica debe proporcionar al usuario un feedback inmediato cuando cometa un error	OK
PV-12	Carga modular de aeropuertos y modelos de avión	OK
PV-13	Guardado de los resultados de la sesión de entrenamiento en el almacenamiento persistente	OK
PV-14	Carga de los resultados de las sesiones de entrenamiento realizadas desde el almacenamiento persistente	OK
PV-15	El usuario debe poder revisar las estadísticas de sesiones de control realizadas	OK

Tabla 5.1. Pruebas de validación del sistema

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

La figura 5.3 muestra el aspecto del menú principal de la aplicación.

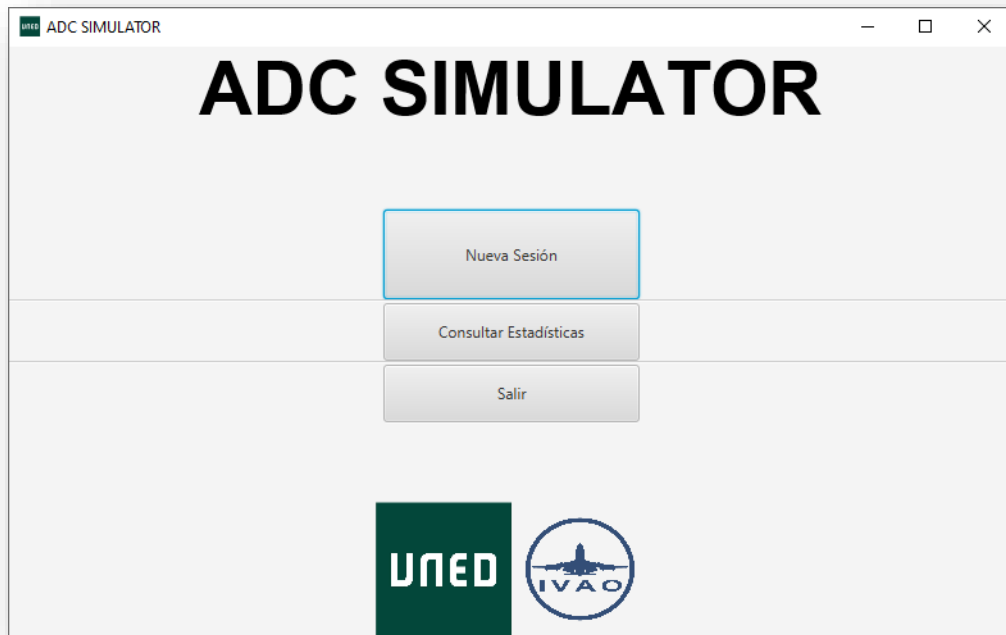


Figura 5.3. Menú principal de la aplicación

La figura 5.4 muestra la ventana de simulación en la que el usuario puede escoger el aeródromo y meteorología donde desea iniciar la simulación.

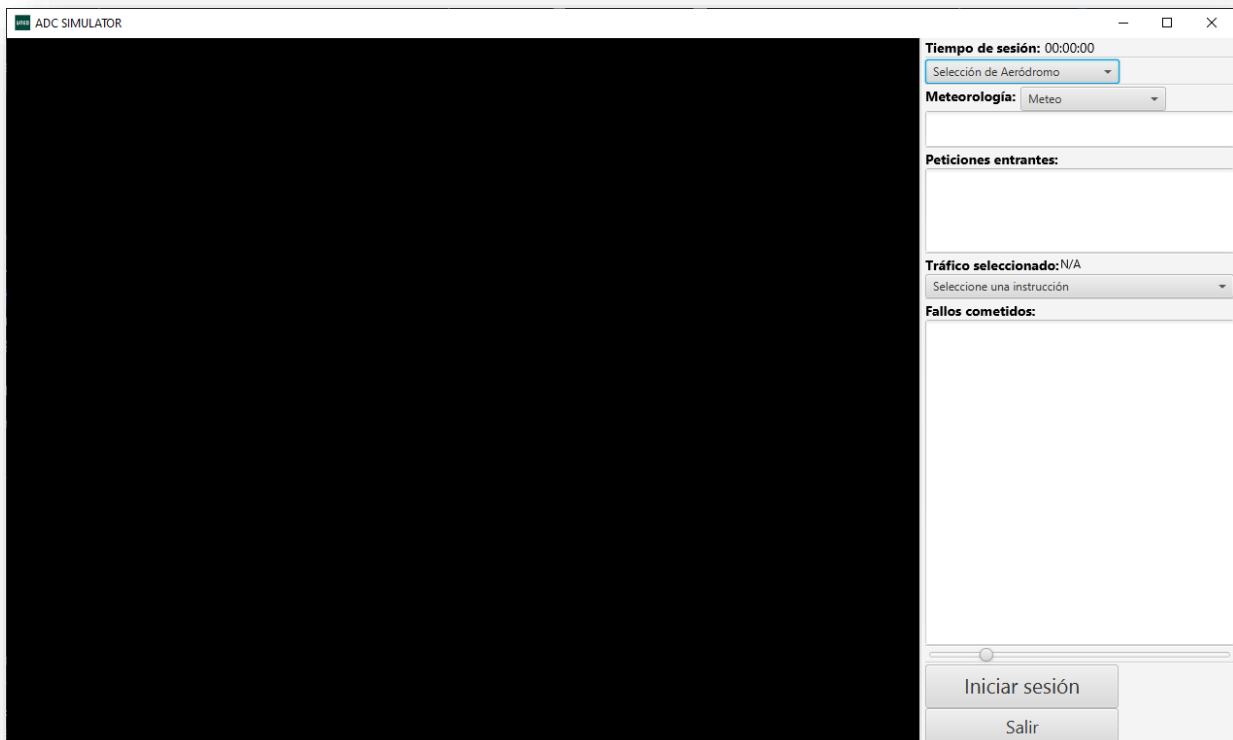


Figura 5.4. Ventana de simulación

Capítulo 5. Implementación y pruebas

La figura 5.5 muestra un ejemplo del aspecto que tiene la aplicación tras iniciar una sesión de entrenamiento. Se ha desplazado el radar a la derecha para mostrar la aeronave en llegada.

La figura 5.6 muestra un ejemplo de selección de ruta de rodaje para la aeronave de indicativo “EC-XWU”. En la lista de la izquierda se encuentran todas las calles de rodaje disponibles, para añadirla a la lista el usuario selecciona la calle que desea y pulsa el botón “→”.

Si el usuario desea eliminar alguna calle de rodaje de la lista se utiliza el botón “←”. La ruta de rodaje es enviada a la aeronave en el orden en que se encuentra listada, para reordenar las calles de rodaje se utilizan los botones “↑” y “↓”.

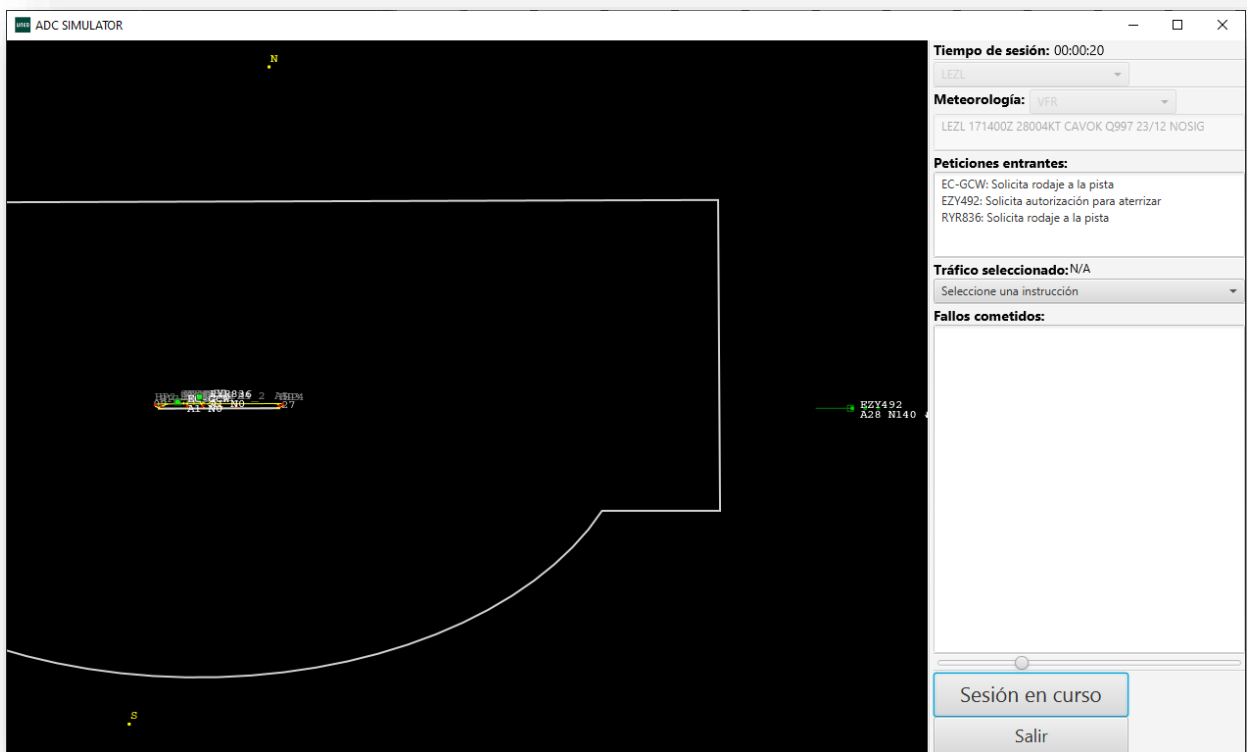


Figura 5.5. Aspecto de la aplicación tras iniciar una sesión de entrenamiento

La figura 5.7 muestra dos aeronaves rodando a la pista mientras una aterriza y otra se encuentra en el estacionamiento esperando rodaje a la pista.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

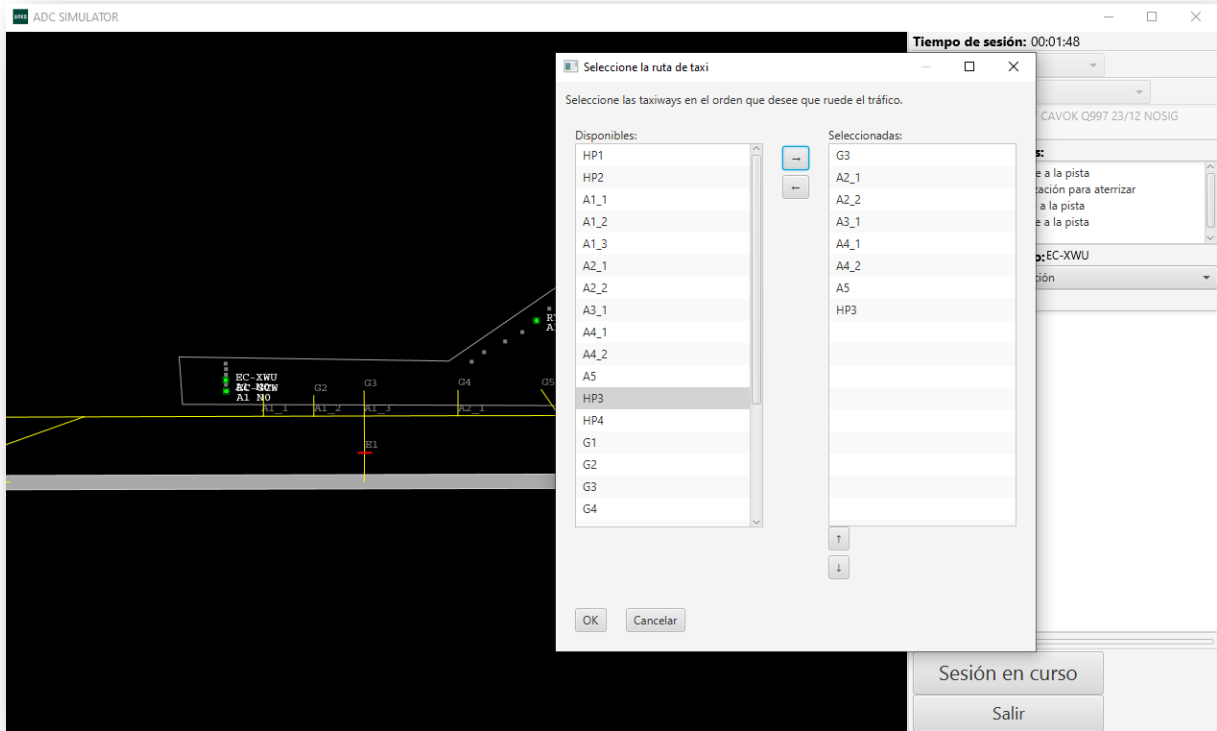


Figura 5.6. Selección de la ruta de rodaje

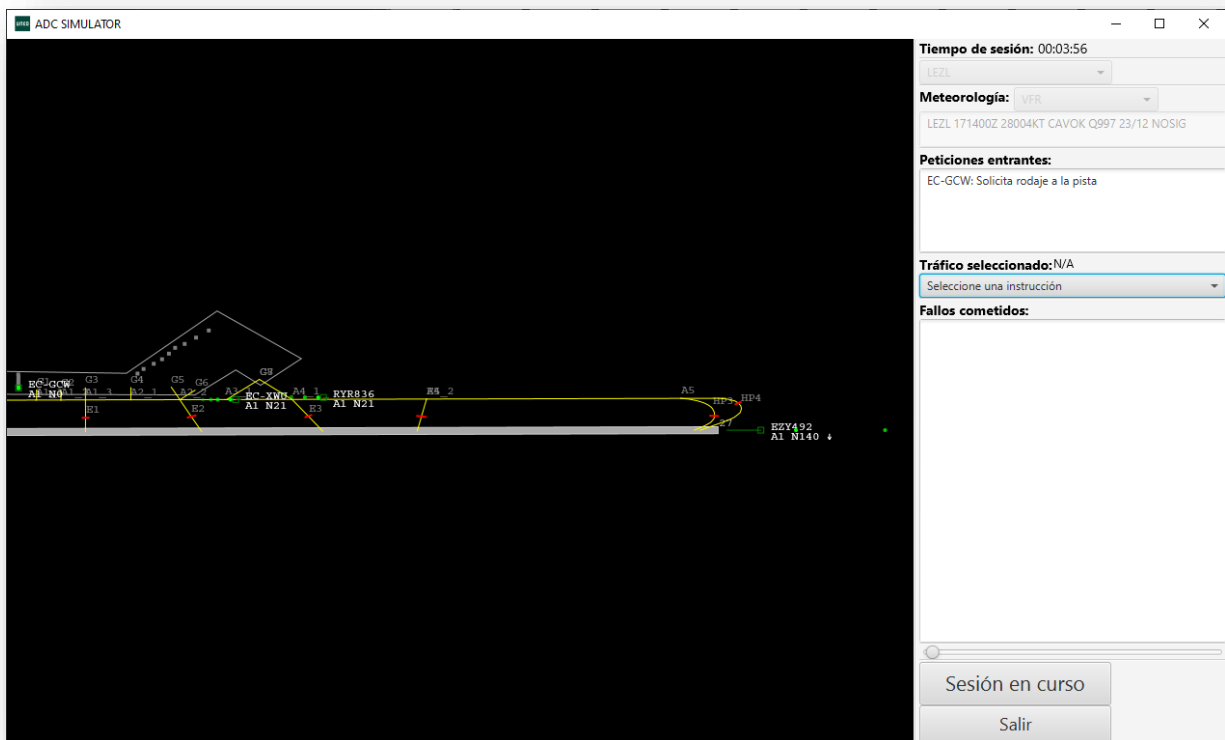


Figura 5.7. Dos aeronaves rodando a la pista

Capítulo 5. Implementación y pruebas

En la figura 5.8 se puede ver cómo las aeronaves que en la figura anterior rodaban a la pista han alcanzado el punto de espera, la aeronave aterrizaba ya ha recibido el rodaje al estacionamiento y la aeronave que solicitaba rodaje a la pista ha recibido la ruta de rodaje.

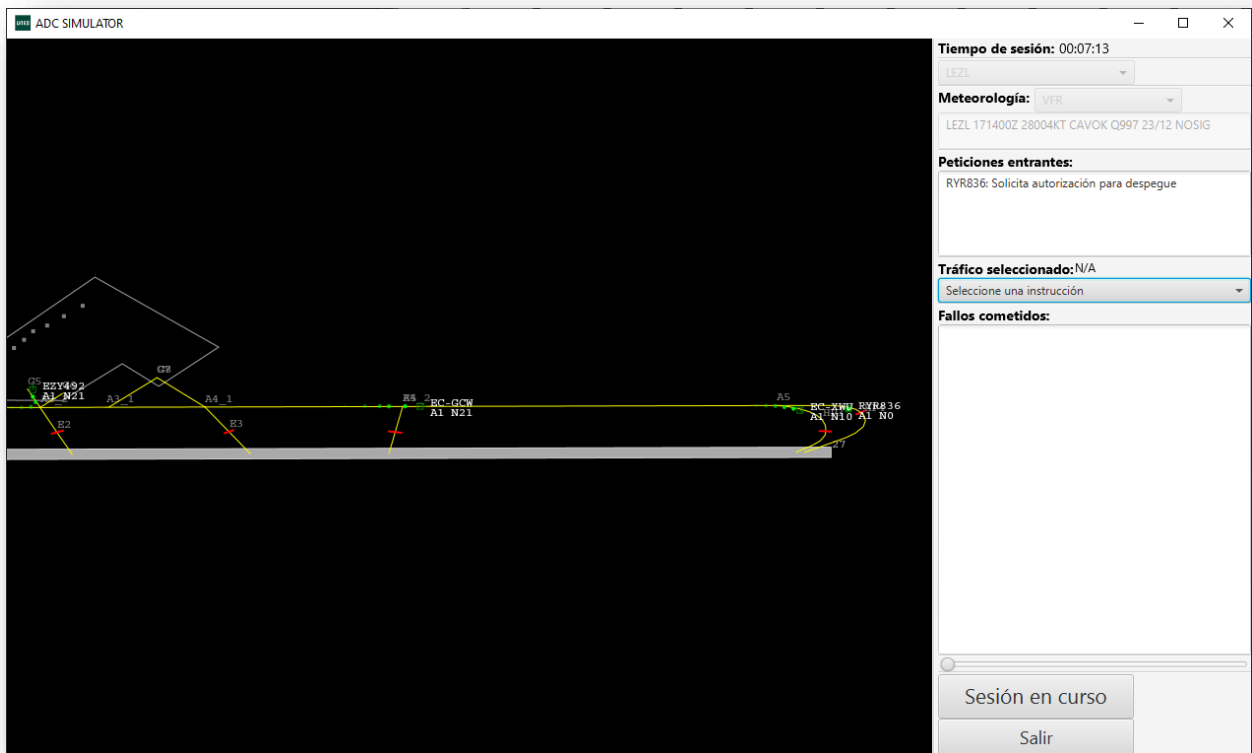


Figura 5.8. Dos aeronaves esperando para despegar mientras otra rueda hacia la pista y otra rueda al estacionamiento

En la figura 5.9 se muestra lo que ocurre cuando el usuario comete una infracción. La aplicación muestra al usuario la infracción en el cuadro de errores cometidos. En este caso, el usuario autorizó el despegue del “EC-XWU” demasiado pronto, la aeronave precedente “RYR836” que puede ver en la figura 5.8 todavía se encontraba despegando cuando el usuario le autorizó el despegue (“RYR836” no aparece en esta captura al haber concluido el despegue hace unos segundos, quedando fuera del encuadre del radar en esta captura).

En la figura 5.10 se muestra a dos aeronaves bajo reglas VFR realizando circuitos (“EC-GCW” en viento en cola, “EC-XWU” en corta final), una aeronave rodando a la pista (“RYR782”) y una aeronave instrumental aproximándose para aterrizar (“EZY37”)

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

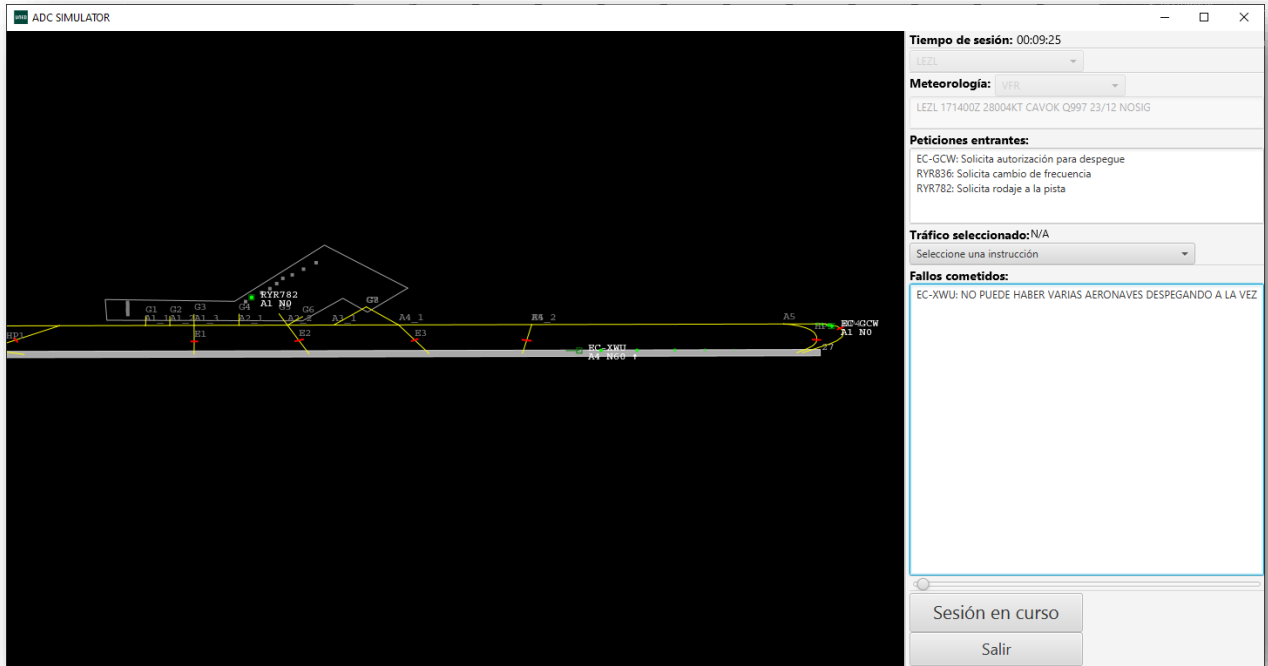


Figura 5.9. El usuario autorizó el despegue sin dar suficiente distancia de seguridad, considerándose un despegue simultáneo de dos aeronaves a la vez

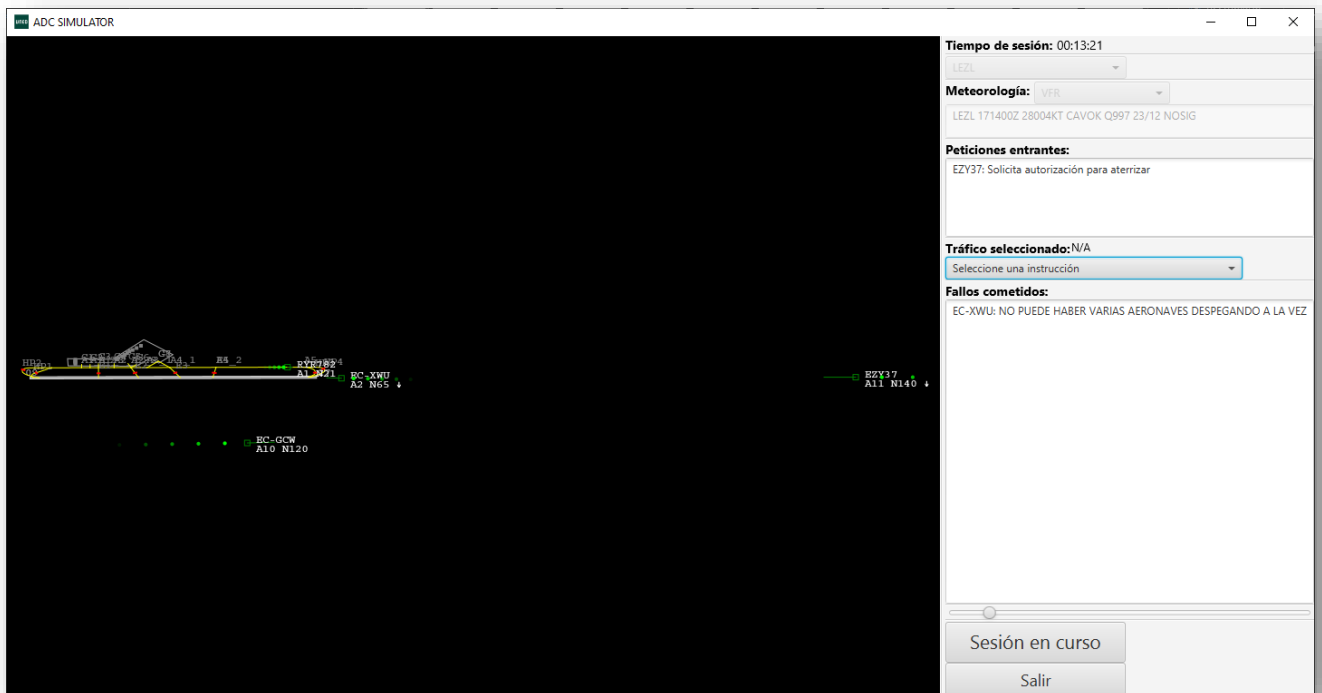


Figura 5.10. Dos aeronaves realizando circuitos de tránsito aéreo mientras una aeronave se dirige a la pista para despegar y otra se aproxima para aterrizar

En la figura 5.11 se muestra la ventana de estadísticas de las sesiones realizadas. Seleccionando una sesión se muestra la información sobre los fallos cometidos.

Capítulo 5. Implementación y pruebas

Panel de Estadísticas

Sesiones Realizadas

Fecha	Aeródromo	Duración	Despegues	Aterrizajes	T/G	IFR gestionados	VFR gestionados	Fallos
2025-09-17_16-24-47	Aeropuerto de Sevilla	00:22:33	5	5	1	4	2	3
2025-09-17_16-02-08	Aeropuerto de Sevilla	00:00:15	0	0	0	0	0	0
2025-09-14_14-16-00	Aeropuerto de Sevilla	00:04:42	1	1	0	0	0	0
2025-09-14_14-10-50	Aeropuerto de Sevilla	00:00:13	0	0	0	0	0	0
2025-09-14_14-10-17	Aeropuerto de Sevilla	00:00:12	0	0	0	0	0	0
2025-09-14_14-09-20	Aeropuerto de Sevilla	00:00:04	0	0	0	0	0	0
2025-09-14_14-06-13	Aeropuerto de Sevilla	00:00:06	0	0	0	0	0	0

Detalles sobre los Fallos Cometidos

Tipo de Fallo Cometido	Número de veces
TAXI_ROUTE_INVALID	1
CONFLICT_AIRBORNE_GROUND	1
CONFLICT_AIRBORNE	0
MULTIPLE_TAKEOFF	1
AUTO_GO_AROUND	0

Volver al Menú

Figura 5.11. Ventana de estadísticas de las sesiones realizadas. En la parte inferior se muestran los fallos cometidos en la sesión seleccionada

5.5 Conclusiones

En este capítulo se ha descrito la implementación de la aplicación, explorando el entorno de desarrollo y herramientas utilizadas, las dependencias necesarias y su integración en Maven, cómo se encuentra organizado el código fuente, cómo se ha garantizado la implementación de los requisitos funcionales mediante una descripción del proceso, los diferentes hitos que se han cumplido al implementar el proyecto y las pruebas realizadas.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Capítulo 6. Planificación

6.1 Introducción

En este capítulo se expone el estudio del plan para la realización del proyecto software, describiendo la planificación temporal con las fases del proyecto y la metodología de desarrollo utilizada. Finalmente, se incluye el backlog del proyecto que proporciona una lista de objetivos a cubrir en el desarrollo del proyecto.

6.2 Planificación temporal

6.2.1 Fases del proyecto

Las fases del trabajo al plantear el anteproyecto fueron las siguientes:

1. Afianzar los conocimientos necesarios para el desarrollo y simulación de este tipo de sistemas.
2. Adquirir los conocimientos necesarios sobre la programación de interfaces gráficas en Java con Java Swing.
3. Diseño de la aplicación.
4. Desarrollo del software.
5. Pruebas y validación.
6. Análisis del resultado del proyecto.
7. Elaboración de la memoria.
8. Defensa del PFG.

Para ello, teniendo en cuenta que el proyecto se realizará en el segundo cuatrimestre, la planificación inicial fue la siguiente:

- Septiembre 2024 a enero 2025 (Mientras se trabaja con las asignaturas del primer cuatrimestre): Planificación de la aplicación, evaluación de los recursos necesarios, estudio y adquisición de conocimientos sobre Java Swing y modelado de Sistemas.
- Febrero 2025: Finalización de la planificación y realización del diseño de la aplicación.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

- Marzo 2025: Codificación.
- Abril 2025: Codificación.
- Mayo 2025: Codificación, evaluación de la aplicación y comienzo de elaboración de la memoria.
- Junio 2025: Elaboración de la memoria.
- Julio 2025: Defensa del proyecto.

Debido a complicaciones en el segundo cuatrimestre y a la mayor dificultad de la esperada para poder realizar este proyecto, fue necesario replantear la planificación manteniendo las fases como estaban. El calendario final ha resultado ser:

- Septiembre 2024 a enero 2025 (Mientras se trabaja con las asignaturas del primer cuatrimestre): Evaluación de los recursos necesarios.
- Febrero 2025: Repaso y adquisición de conocimientos necesarios expuestos en el apartado anterior.
- Marzo 2025: Pruebas de la viabilidad del proyecto con las tecnologías a utilizar. Estudio de nuevas tecnologías que puedan suplir las carencias encontradas en las pruebas. Planificación de la aplicación y realización del diseño.
- Abril 2025: Fin del diseño de la aplicación. Codificación.
- Mayo 2025: Codificación.
- Junio 2025: Codificación.
- Julio 2025: Codificación y elaboración de la memoria.
- Agosto 2025: Codificación, pruebas y elaboración de la memoria.
- Septiembre: Fin de la elaboración de la memoria y defensa del proyecto.

6.2.2. Metodología de desarrollo

Como puede apreciarse en el calendario final, el proyecto pasó por 4 etapas:

- Septiembre 2024 a febrero 2025: Esta etapa se caracteriza por maximizar los recursos y el tiempo en repasar y adquirir los conocimientos necesarios para abordar el proyecto.
- Marzo 2025: En esta etapa se alterna el uso del tiempo para aprender nuevas tecnologías y realizar pruebas antes de comprometer el diseño de la aplicación a algo erróneo.

Capítulo 6. Planificación

- Abril 2025 a agosto 2025: Una vez fijado el Plan de Proyecto de Software y el diseño, la fase de codificación se gestionó con un enfoque iterativo basado en las metodologías ágiles. Se adaptaron reuniones e ítems de equipo por revisiones personales y ajustes continuos al backlog:
 - Fase 1: Se decide de qué cambios se quieren aplicar al proyecto en el ciclo actual.
 - Fase 2: Se realiza una sesión para considerar qué es necesario realizar o modificar para efectuar esos cambios (clases implicadas, conocimiento sobre la forma de aplicar los cambios, ...), y cómo se van a realizar y comprobar que los cambios se han realizado correctamente revisando los ítems producidos en la fase de análisis y diseño, actualizándolos si es necesario.
 - Fase 3: Implementación de los cambios.
 - Fase 4: Realización de test unitarios y de integración.
 - Fase 5: Corrección de errores y problemas encontrados en los test.
 - Fase 6: Satisfecho con el resultado, se documentan los cambios en un cuaderno de desarrollo explicando los cambios efectuados, decisiones tomadas y otros ítems que se deban tener en cuenta en el futuro. También se actualiza el backlog del proyecto.
- Septiembre 2025: Se caracteriza por la realización de ajustes menores a la aplicación y realización de la memoria.

6.2.3. Backlog del proyecto

Para tener una meta clara de la aplicación final se marcaron una lista de objetivos iniciales en la fase de planificación del proyecto. Al conocer cuán difícil sería cada tarea en la práctica la lista de objetivos (backlog) se fue revisando a medida que se avanzaba en el proyecto en la fase de desarrollo tal y como se ha indicado en el apartado anterior. La lista de objetivos iniciales del backlog es la siguiente:

- Implementación de una GUI para la representación del espacio aéreo y rodadura del aeropuerto mediante una pantalla radar similar a Aurora (el software de control de IVAO).
- Simulación del movimiento de aeronaves en el espacio aéreo y rodadura.
- Implementación del conjunto de reglas que dictan las acciones que puede realizar un piloto y las instrucciones o servicios que puede dar un controlador.

- Las aeronaves simuladas solicitarán lo que desean hacer y el usuario tendrá la capacidad de instruir a la aeronave a realizar alguna acción.
- Implementación de un sistema de evaluación del desempeño del usuario en la sesión de entrenamiento.
- Las estadísticas del usuario que deben almacenarse para el sistema de evaluación del usuario deben ser al menos: Número de fallos, duración de la sesión y número de aeronaves controladas.
- Implementación de una base de datos.
- La base de datos debe contener un apartado con aeronaves que se puedan inyectar a la simulación.
- La base de datos debe contener un apartado con aeródromos que el usuario puede utilizar para los entrenamientos.
- La base de datos debe contener un apartado para el almacenamiento persistente de métricas del usuario.
- Proporcionar al usuario la capacidad de expandir el software añadiendo nuevos aeropuertos y aeronaves simuladas sin necesidad de tener acceso al código fuente.

6.3. Conclusiones

En este capítulo se ha descrito cómo se realizó la planificación del proyecto y en qué se tradujo en la práctica. Luego se ha expuesto la metodología de desarrollo utilizada en el proyecto. Finalmente, se ha presentado el backlog del proyecto. Todos estos elementos proporcionan un apoyo para comprobar los ítems que se van cubriendo en todas las fases del proyecto.

Capítulo 7. Conclusiones y trabajos futuros

7.1 Introducción

Examinamos en este capítulo el proyecto considerando el trabajo realizado, valorando el proceso ejecutado, los resultados obtenidos y las posibles líneas de desarrollo futuro.

7.2 Conclusiones

El principal objetivo del proyecto consiste en la creación de un simulador de control de tráfico aéreo a nivel de torre. Este objetivo se puede desgranar en los siguientes:

- Desarrollo de un entrenador que pueda ser de utilidad a nuevos usuarios de la red de vuelo para aprender y practicar el control de tráfico aéreo en posición de torre/rodadura.
- Simulación del espacio aéreo controlado típico alrededor de un aeropuerto que admita aeronaves bajo reglas visuales (VFR).
- Implementación de un conjunto de reglas que marcan las acciones que puede realizar un piloto y las instrucciones o servicios que puede dar un controlador.
- Implementación de una GUI que represente los elementos más importantes del aeropuerto: el espacio aéreo, las pistas, la rodadura del aeropuerto, las barras de parada, la plataforma de estacionamientos y los estacionamientos.
- Implementación de una base de datos con información sobre aeronaves, aeródromos y estadísticas anónimas de las sesiones de entrenamiento realizadas.
- Proporcionar una arquitectura capaz de permitir la expansión del software permitiendo ejecutar el entrenamiento en otros aeródromos añadidos con posterioridad sin necesidad de que el usuario disponga del código fuente de la aplicación.

Para poder cumplir con estos requisitos hemos empleado la ingeniería del software para resolver el problema, analizando el modelo de dominio, diseñando la aplicación, implementándola y probándola.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Este software está orientado principalmente a ayudar a los compañeros de la red que hacen de tutores, y a los alumnos que desean iniciarse en el control de tráfico aéreo. Mediante una interfaz gráfica que muestra la pantalla radar, el usuario puede visualizar los elementos más importantes del aeropuerto: Pistas de aterrizaje, plataforma de estacionamiento, estacionamientos, calles de rodadura, barras de parada, espacios aéreos y puntos de notificación visual.

La aplicación cuenta con agentes que controlan las aeronaves inyectadas en la simulación, lo que permite que el usuario pueda practicar el control de tráfico aéreo a aeronaves bajo reglas de vuelo visual en cualquier momento y en cualquier aeródromo que desee y que permita el vuelo visual.

El supervisor se encarga de comprobar que se cumplen las reglas implementadas y anota tanto las infracciones cometidas por el usuario como otros hitos importantes en la sesión.

El sistema de persistencia se implementó mediante una estructura de directorios y archivos JSON con un esquema de datos definido. Esto permite a la aplicación almacenar los resultados de las sesiones realizadas por el usuario y poder cargar cualquier aeropuerto que el usuario desee introducir en la aplicación sin necesidad de que disponga de acceso al código fuente o tenga conocimientos de programación en Java.

La organización del proyecto en paquetes, el apoyo en patrones de diseño y el enfoque en la genericidad de las clases favorece la expansión y mantenibilidad del software de cara al futuro.

Para la consecución de los objetivos del proyecto se han realizado las siguientes tareas:

1. Estudio del contexto del problema para decidir qué es necesario tener en cuenta para la realización del proyecto priorizando los elementos encontrados por importancia.
2. Estudio y valoración de las tecnologías necesarias para la implementación de los diferentes componentes que se necesitarán en el sistema final.
3. Análisis sobre qué elementos software son necesarios en la aplicación con el fin de representar los diferentes aspectos descubiertos en el estudio del contexto del problema.
4. Diseño de las diferentes partes del sistema apoyándose en el modelado.
5. Comprobación de que el diseño cumple los requisitos y objetivos propuestos.
6. Construcción de la aplicación utilizando el análisis y el diseño realizado.
7. Pruebas y evaluación de la aplicación comprobando que el sistema se ajusta a los requisitos y objetivos propuestos.

Capítulo 7. Conclusiones y trabajos futuros

8. Elaboración de la memoria y documentación final del proyecto.

7.3 Trabajos futuros

Las posibles ampliaciones al software que se proponen son las siguientes:

- Implementación de otras maniobras que puedan realizar o solicitar las aeronaves bajo reglas de vuelo visual y que por limitaciones de tiempo no se encuentran implementadas, por ejemplo:
 - Volar a puntos de notificación visual para abandonar el espacio aéreo.
 - Ingresar al espacio aéreo desde puntos de notificación visual.
 - Realizar “cruces de campo” (sobrevuelos del aeropuerto) para cruzar el espacio aéreo o para incorporarse en el circuito de tránsito aéreo
 - Realizar “órbitas” en la presente posición (realizar giros de 360 grados de forma que tarde 2 minutos en realizar el giro completo)
- Mejora de la interfaz gráfica.
- Aprovechamiento de la API de IVAO para la obtención de los datos necesarios para aeropuertos y modelos de aeronave.
- Implementación de un creador de escenarios.
- Añadir accesibilidad para daltónicos.

La implementación de las maniobras es más compleja de lo que puede parecer debido a que hay varios factores que condicionan la forma en la que se realizan algunas de ellas, por lo que esto debe tenerse en cuenta al escoger esta línea de desarrollo.

La calidad de la interfaz gráfica ha estado condicionada a la experiencia del programador realizando diseño de interfaces gráficas, usabilidad y diseños con JavaFX. Con mayor experiencia en el futuro se puede conseguir una interfaz más atractiva.

IVAO dispone de una API [29], se podría aprovechar la adaptabilidad con la que se ha dotado al proyecto para obtener los datos necesarios para el entrenador de ella añadiendo otro proveedor de datos a la aplicación.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

El creador de escenarios permitiría al usuario definir cuál es la situación inicial en la que desea que se encuentre la simulación. De esta forma, el usuario podría practicar cuantas veces quisiera un escenario que necesita practicar.

Al distribuir la aplicación, uno de los comentarios que me encontré es que no podían distinguir los elementos de color rojo como las barras de parada. Por tanto, una posible mejora de la aplicación sería implementar un modo adaptado para daltónicos en la interfaz gráfica, mejorando así la accesibilidad para las personas con esta condición.

Bibliografía

[1] Control de tráfico aéreo (ATC). Página web de ENAIRE. Último acceso: 10-9-2025.

URL:

https://www.enaire.es/servicios/atm/servicios_de_transito_aereo_atc/control_de_trafico_aereo_atc

[2] IVAO FRA Policy (Facility Rating Assignments) en la Wiki de IVAO. Último acceso: 10-9-2025.

URL: https://wiki.ivao.aero/en/home/atcoperations/fra_policy

[3] Academia CAVOK – IVAO España. Último acceso: 10-9-2025.

URL: <https://cavok.es.ivao.aero/>

[4] Documentación para el alumno ATC avanzado (AS3) - Wiki de IVAO España.

Último acceso: 10-9-2025.

URL: <https://wiki.es.ivao.aero/books/alumno-atc-avanzado-as3>

[5] Web de IVAO HQ. Último acceso: 10-9-2025.

URL: <https://ivao.aero/>

[6] Web de la división española de IVAO. Último acceso: 10-9-2025.

URL: <https://es.ivao.aero/>

[7] Web de la división del este asiático en IVAO. Último acceso: 10-9-2025.

URL: <https://xe.ivao.aero/>

[8] Wiki de IVAO España. Reglas de vuelo VFR. Último acceso: 10-9-2025. URL:

<https://wiki.es.ivao.aero/books/piloto-privado-pp/page/reglas-de-vuelo-vfr>

[9] Wiki de IVAO España. Reglas de vuelo IFR. Último acceso: 10-9-2025.

URL: <https://wiki.es.ivao.aero/books/piloto-privado-senior-spp/page/reglas-de-vuelo-por-instrumentos>

[10] AIP España. Servicio de Información Aeronáutica. Último acceso: 10-9-2025.

URL: <https://aip.enaire.es/AIP/>

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

[11] Wiki de IVAO España. Autorizaciones IFR. Puntos 2.2 “Puesta en marcha, retroceso y rodaje”, 2.3 “Autorización a despegar” y 3 “Autorizaciones IFR en llegadas”. Último acceso: 10-9-2025.

URL: <https://wiki.es.iviao.aero/books/alumno-atc-avanzado-as3/page/autorizaciones-ifr>

[12] Wiki de IVAO España. Control básico VFR. Último acceso: 10-9-2025.

URL: <https://wiki.es.iviao.aero/books/alumno-atc-avanzado-as3/page/control-basico-vfr>

[13] Página de descarga de Aurora. Último acceso: 10-9-2025.

URL: <https://www.iviao.aero/softdev/software/aurora.asp>

[14] Página de Steam de Endless ATC. Último acceso: 10-9-2025.

URL: https://store.steampowered.com/app/666610/Endless_ATC/

[15] Página de Google Play de Endless ATC. Último acceso: 10-9-2025.

URL: <https://play.google.com/store/apps/details?id=com.dirgrats.endlessatc&hl=es>

[16] Página de Steam de VoiceATC Simulator. Último acceso: 10-9-2025.

URL: https://store.steampowered.com/app/3529560/VoiceATC_Simulator/

[17] Página del simulador online OpenScope Air Traffic Control Simulator. Último acceso: 10-9-2025.

URL: <https://www.openscope.co/>

[18] GitHub de OpenScope. Último acceso: 10-9-2025.

URL: <https://github.com/openscope/openscope>

[19] FeelThere, Tower! Simulator 3 en Steam. Último acceso: 10-9-2025.

URL: https://store.steampowered.com/app/2176130/Tower_Simulator_3/

[20] Alexander Shvets. Sumérgete en los Patrones de Diseño. 2021

[21] Francisco Jurado Monroy. Desarrollo de Videojuegos: Un Enfoque Práctico. Volumen 4. Desarrollo de Componentes. Septiembre 2015.

[22] Base de datos de rendimiento de aeronaves de Eurocontrol. Último acceso: 10-9-2025.

URL: <https://learningzone.eurocontrol.int/ilp/customs/ATCPFDB/default.aspx>

Bibliografía

[23] Geojson.io. Último acceso: 10-9-2025.

URL: <https://geojson.io>

[24] API de JavaFX. Último acceso: 10-9-2025.

URL: <https://openjfx.io/javadoc/21/>

[25] Página del Apache Maven Project. Último acceso: 10-9-2025.

URL: <https://maven.apache.org/index.html>

[26] GitHub de Gson by Google. Último acceso: 10-9-2025.

URL: <https://github.com/google/gson>

[27] Documentación de JUnit 5. Último acceso: 10-9-2025.

URL: <https://docs.junit.org/current/api/index.html>

[28] Documentación del plugin Surefire de Maven. Último acceso: 10-9-2025.

URL: <https://maven.apache.org/surefire/maven-surefire-plugin/>

[29] Documentación de la API de IVAO. Último acceso: 10-9-2025.

URL: <https://api.iva0.aero/docs>

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Glosario

API Application Programming Interface, 95, 97

ATC Air Traffic Control, 9, 15, 16, 22, 23, 24, 34

FSM Finite State Machine, 44

FXML Formatting for XML, 42, 63, 66, 68, 126

ICAO International Civil Aviation Organization, 9, 27, 28, 37, 57, 58, 59, 75, 76, 104, 111, 112

IFR Instrumental Flight Rules, 15, 16, 18, 19, 37, 40, 47, 48, 51, 52, 53, 54, 55, 58, 71

IVAO International Virtual Aviation Organization, 9, 13, 14, 15, 22, 23, 25, 27, 30, 92, 95, 97

JDK Java Development Kit, 63, 126

JRE Java Runtime Environment, 63, 126

JSON JavaScript Object Notation, 45, 57, 58, 59, 63, 75, 76, 77, 80, 81, 85, 95

MVC Modelo-Vista-Controlador, 43

VFR Visual Flight Rules, 10, 15, 16, 19, 21, 25, 27, 28, 30, 31, 37, 38, 40, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 60, 71, 88, 94, 105, 112, 115, 124

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Anexo A – Manual de usuario

A.1 Introducción

Este anexo recoge la información necesaria para que el usuario tenga el conocimiento necesario para utilizar la aplicación correctamente.

Se asume que el usuario tiene los conocimientos teóricos básicos sobre aviación impartidos en la academia CAVOK de la división española de IVAO [3].

A.2 Estructura de la aplicación

A continuación, se describe la estructura con la que se distribuye la aplicación.

El ejecutable de la aplicación es el archivo “adcsim.jar”, el cual debe tener en su mismo directorio una carpeta llamada “bbdd” donde se colocan los archivos relacionados con los datos de aeropuertos y que el simulador carga al arrancar. Dentro de la carpeta “bbdd” encontrará la carpeta “aircraftModels” donde se ubican los archivos JSON que describen cada modelo de avión, y “airports” donde se ubica los archivos JSON que describen cada aeropuerto en carpetas separadas diferenciadas por el código ICAO del aeropuerto. Es posible que también encuentre una carpeta llamada “stats”, en esta carpeta.

En los apartados “Cómo añadir nuevos modelos de aeronave” y “Cómo añadir nuevos aeropuertos” se describirá el proceso para añadir nuevos modelos de aeronave y aeropuertos a la aplicación.

A.3 Menú principal

La figura A.1 muestra el menú principal que se muestra tras ejecutar la aplicación.



Figura A.1. Menú principal de la aplicación

Si desea realizar una sesión de práctica pulse en el botón “Nueva Sesión”. Si lo que desea es revisar las estadísticas de sesiones ya realizadas pulse el botón “Consultar Estadísticas”. Tenga en cuenta que si no ha realizado ninguna sesión de entrenamiento la sección de estadísticas estará vacía.

Tras pulsar el botón “Nueva Sesión” nos encontramos en el apartado de donde se encuentra el simulador representado en la figura A.2. A la izquierda tiene la pantalla radar, actualmente no muestra nada hasta que no inicie el simulador, y a la derecha tiene la información de la sesión. Los elementos de arriba hacia abajo son los siguientes:

- Tiempo de sesión: Muestra el tiempo transcurrido desde que se inició la sesión.
- Selector de aeródromo: Aquí aparecerán los aeropuertos que han sido cargados desde la carpeta “bbdd/airports”.
- Meteorología: Aquí podemos escoger la meteorología de la sesión. Puede escoger entre condiciones VFR y condiciones MVFR. No se contempla la posibilidad de escoger condiciones que no permitan el vuelo visual ya que uno de los principales objetivos del simulador es que entrene con aeronaves bajo reglas de vuelo visual.
- METAR: El METAR aparecerá aquí cuando haya seleccionado el aeropuerto y la meteorología.

ANEXO A – Manual de usuario

- **Peticiones entrantes:** Aquí aparecerán las peticiones que realicen las aeronaves y tenga pendientes de atender.
- **Tráfico seleccionado:** Muestra la aeronave actualmente seleccionada al realizar clic sobre ella en la pantalla radar.
- **Lista de instrucciones:** Una vez seleccione una aeronave, aquí aparecerá la lista de instrucciones que puede enviarle.
- **Fallos cometidos:** Aquí se listan las infracciones que ha cometido en la sesión.
- **Selector de zoom:** Este control deslizante le permite ajustar el zoom de la pantalla radar. Adicionalmente también puede cambiarlo más cómodamente con la rueda del ratón mientras tiene el ratón sobre la pantalla radar.
- **Botón de “Iniciar Sesión”:** Inicia la sesión.
- **Botón de “Salir”:** Permite volver al menú principal. Si hay alguna sesión iniciada la finaliza y guarda sus estadísticas automáticamente.

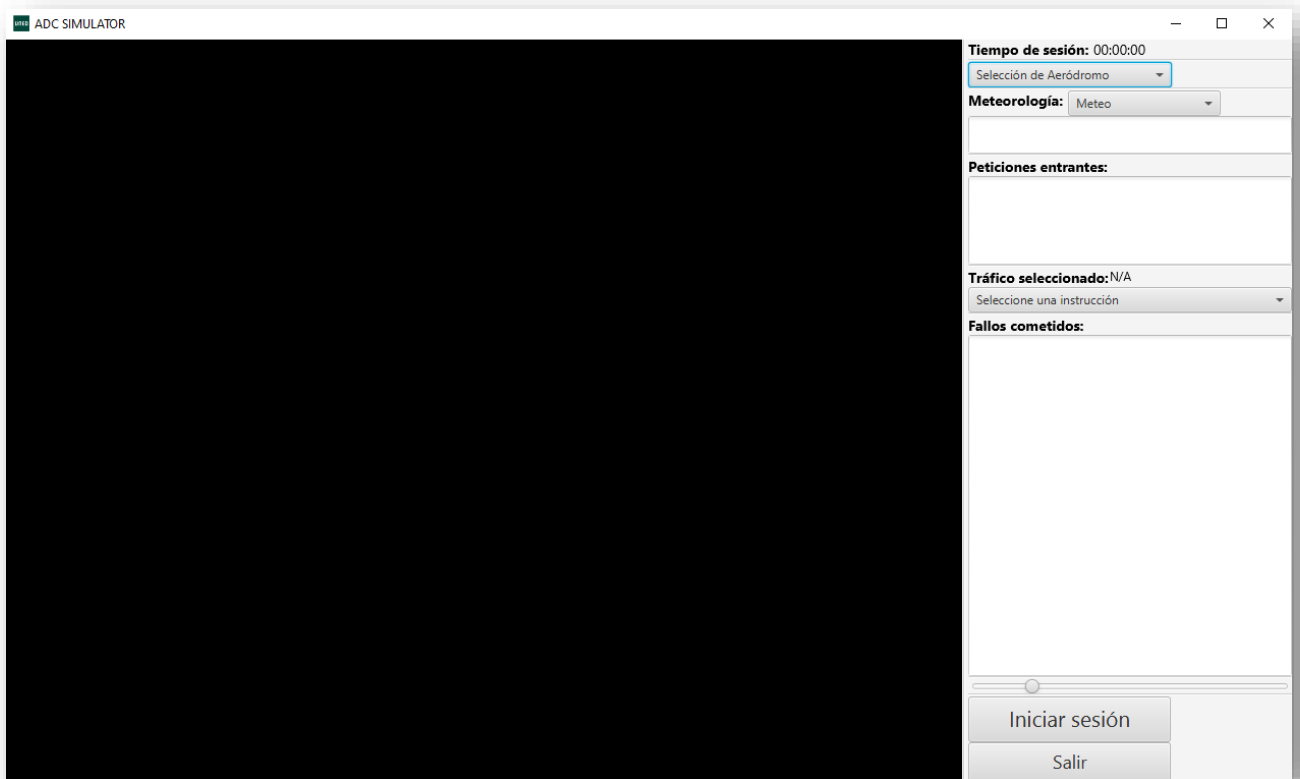


Figura A.2. Vista de sesión de entrenamiento

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Nota: Para salir sin guardar los resultados de la sesión actual, cierre la aplicación mediante la “X” de cierre de ventana que proporciona por defecto el sistema operativo. Se le mostrará el mensaje de advertencia de la figura A.3 recordándole que, si cierra la aplicación de esta forma, perderá los resultados de la sesión.

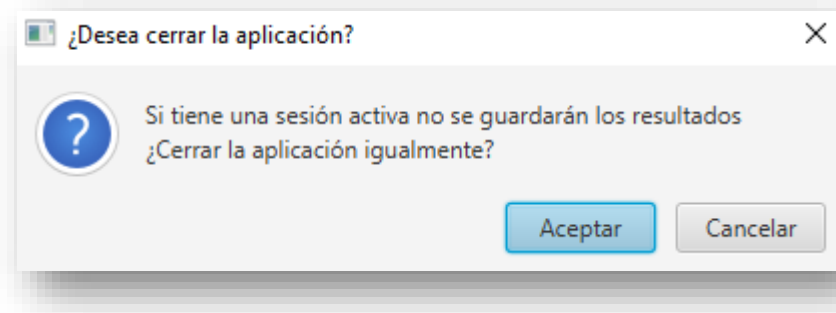


Figura A.3. Mensaje de advertencia al cerrar la aplicación

Una vez seleccionado el aeródromo y la meteorología, si pulsa el botón “Iniciar Sesión” dará comienzo la sesión de entrenamiento. A la izquierda podrá ver la pantalla radar, puede hacer clic y arrastrar para mover la presentación y hacer zoom con el control deslizante del panel derecho o mediante la rueda del ratón.

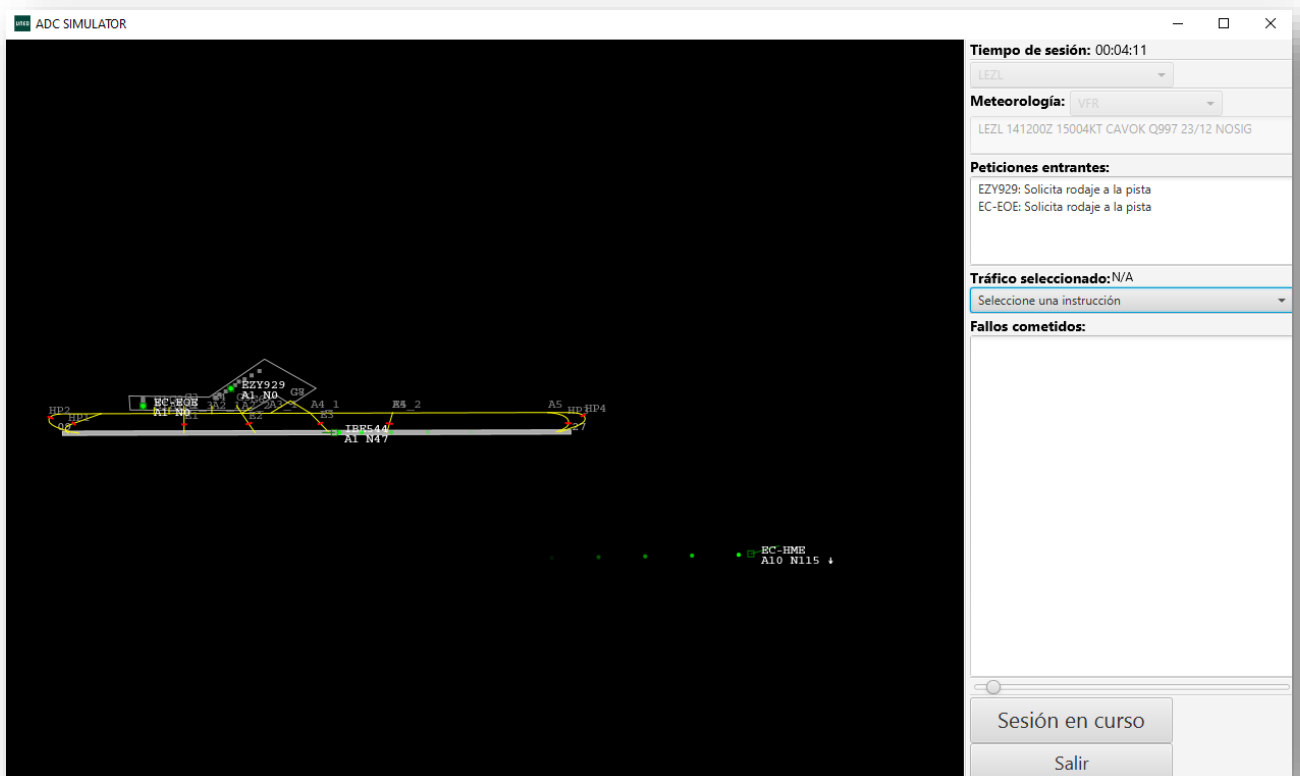


Figura A.4. Presentación de la pantalla radar durante una sesión de entrenamiento

ANEXO A – Manual de usuario

Como puede ver en la figura A.4, en la pantalla radar encontrará:

- Las aeronaves representadas por un cuadrado hueco de color verde claro. Cada aeronave posee:
 - Una etiqueta identificativa con el indicativo en la primera línea y la altitud en cientos de pies, velocidad en nudos y tendencia de velocidad vertical en la segunda línea.
- Las pistas de aterrizaje representadas en color gris claro, con los designadores de las pistas junto a la cabecera de la misma.
- La plataforma de estacionamiento se encuentra en el aeropuerto y está mostrado por un polígono de color gris oscuro, los estacionamientos se muestran como cuadrados de color gris oscuro.
- Las calles de rodaje están representadas mediante líneas de color amarillo. El nombre de la calle de rodaje se encuentra al final de la misma, en el lado derecho.
- Las barras de parada se indican mediante líneas de color rojo. El nombre se encuentra junto a la barra de parada.
- Los espacios aéreos se representan mediante polígonos de color gris claro.
- Los puntos de notificación visual se representan mediante cuadrados rellenos amarillos junto con su designador.

En el cuadro de “Peticiones entrantes” tendrá listadas las peticiones que debe atender y el indicativo de la aeronave que la realizó. Para atender una petición haga clic en la pantalla radar sobre la aeronave que desee, en el panel informativo de la derecha aparecerá la aeronave seleccionada y la lista de instrucciones que puede enviarle. Para enviar una instrucción a la aeronave haga clic en la instrucción que desea enviar.

Si la instrucción es de rodaje se le abrirá la ventana modal de la figura A.5 (no podrá hacer clic en otras partes de la aplicación hasta que la cierre).

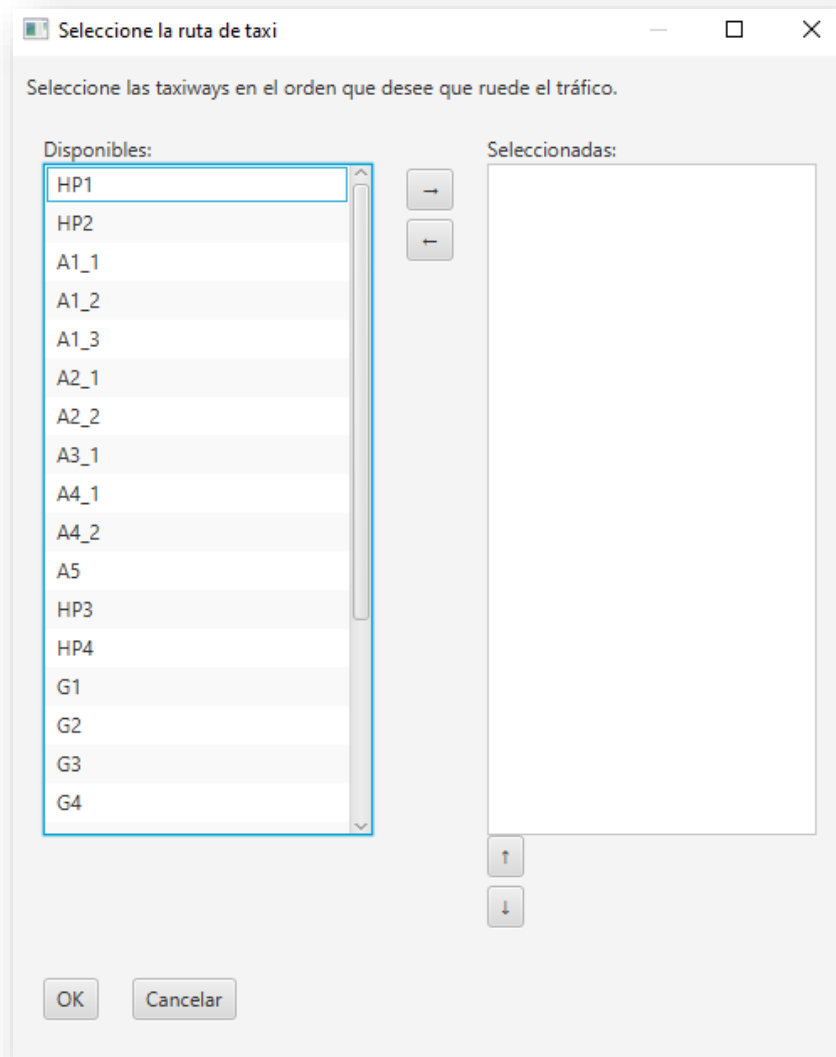


Figura A.5. Ventana para seleccionar la ruta de rodaje

Aquí puede escoger la ruta de rodaje seleccionando la calle de rodaje en la lista izquierda y pulsando el botón “→”. Si desea eliminar alguna calle de rodaje de la lista de calles seleccionadas, selecciónela y pulse el botón “←”. La ruta de rodaje es enviada a la aeronave en el orden en que se encuentra listada, para reordenar las calles de rodaje utilice los botones “↑” y “↓”. Para saber qué calles de rodaje puede escoger puede mover la ventana para visualizar los nombres de las calles de rodaje en la pantalla radar.

Una vez tenga lista la ruta de rodaje, puede enviarla con el botón “Ok”. Para que una ruta sea válida y aceptada por la aeronave se deben cumplir las siguientes condiciones:

- Si la aeronave solicita rodaje a la pista, la primera calle de rodaje de la ruta debe ser la calle de rodaje por la que la aeronave abandonó la pista.

ANEXO A – Manual de usuario

- Si la aeronave solicita rodaje a la pista, la primera calle de rodaje de la ruta debe ser una que esté conectada con la plataforma de estacionamientos.
- No debe haber calles de rodaje repetidas.
- Las calles de rodaje deben estar conectadas entre sí consecutivamente.
- Si la aeronave solicita rodaje a la plataforma, la última calle de rodaje debe acabar en la plataforma de estacionamientos.
- Si la aeronave solicita rodaje a la pista, la última calle de rodaje debe ser una cuyo designador coincida con el de alguna de las barras de parada del aeropuerto.

Cuando desee finalizar la sesión de entrenamiento pulse el botón “Salir”, la aplicación almacenará automáticamente los resultados de la sesión.

Para consultar las estadísticas de las sesiones pasadas pulse en el botón “Consultar Estadísticas” del menú principal. Esto le llevará a la ventana de estadísticas mostrada en la figura A.6 en la que podrá ver los resultados de las sesiones realizadas.

Panel de Estadísticas

Sesiones Realizadas

Fecha	Aeródromo	Duración	Despegues	Aterrizajes	T/G	IFR gestionados	VFR gestionados	Fallos
2025-09-12_13-32-57	Aeropuerto de Sevilla	00:00:16	0	0	0	0	0	2
2025-09-09_12-44-16	Aeropuerto de Sevilla	00:20:16	4	3	1	4	1	2
2025-09-09_00-26-37	Aeropuerto de Sevilla	00:00:03	0	0	0	0	0	0
2025-09-09_00-22-07	Aeropuerto de Sevilla	00:00:03	0	0	0	0	0	0
2025-09-08_23-44-18	Aeropuerto de Sevilla	00:06:30	1	1	0	0	0	0
2025-09-08_22-24-56	Aeropuerto de Sevilla	00:00:05	0	0	0	0	0	0
2025-09-08_13-49-13	Aeropuerto de Sevilla	00:00:23	0	0	0	0	0	0

Detalles sobre los Fallos Cometidos

Tipo de Fallo Cometido	Número de veces
Tabla sin contenido	

Volver al Menú

Figura A.6. Ventana de estadísticas de sesiones realizadas

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Haciendo clic en cualquier sesión podrá comprobar los detalles de los fallos cometidos.

A.5 Cómo añadir nuevos modelos de aeronave

Los archivos que describen los modelos de aeronave cargados se deben colocar en la carpeta “bddd/aircraftModels”. Para añadir un nuevo modelo de avión cree un archivo con extensión “.json” cuyo nombre sea el código ICAO de la aeronave que va a añadir, esto le ayudará a mantener ordenados los modelos y saber cuáles tiene ya y cuáles no. Puede abrir el archivo utilizando un bloc de notas y la estructura del contenido debe ser la siguiente:

- **icao:** Texto con el código ICAO del modelo.
- **weight:** Entero con el peso en kilogramos.
- **maxSpeed:** Entero con la velocidad máxima que puede alcanzar en nudos.
- **cruiseSpeed:** Entero con la velocidad de crucero en nudos.
- **v2Speed:** Entero con la velocidad segura de ascenso tras el despegue.
- **appSpeed:** Entero con la velocidad de aproximación.
- **initialROC:** Entero con la velocidad vertical inicial tras el despegue en pies por minuto.
- **maxAltitude:** Entero con la altitud máxima que puede alcanzar el modelo en pies.
- **role:** Texto con el rol que se desea que tome el modelo. El texto debe ser “IFR” o “VFR”.

Puede obtener estos datos de la base de datos de Eurocontrol [22].

A continuación, mostraré un ejemplo con el modelo de la Cessna 172 cuyo ICAO es C172:

```
{
  "icao": "C172",
  "weight": 1111,
  "maxSpeed": 160,
  "cruiseSpeed": 120,
  "v2Speed": 60,
  "appSpeed": 65,
  "initialROC": 400,
  "maxAltitude": 13000,
  "role": "VFR"
}
```

A.6 Cómo añadir nuevos aeropuertos

Para añadir nuevos aeropuertos debe crear una nueva carpeta en “bbdd/airports” con el código ICAO del aeropuerto que desea añadir. La información geográfica puede obtenerla mediante la herramienta geojson.io [23] marcando puntos y dibujando líneas directamente en el mapa empleando la vista de satélite.

Dentro de la carpeta con el código ICAO del aeropuerto debe crear los siguientes 8 archivos que contienen por separado la información necesaria, para ejemplificar utilizaré el aeropuerto de Sevilla con código ICAO LEZL:

- **airportInfo.json:** Describe la información general del aeropuerto. Contiene un objeto JSON con los siguientes atributos:
 - **icao:** Texto con el ICAO del aeropuerto
 - **name:** Texto con el nombre del aeropuerto
 - **elevation:** Entero con la elevación en pies sobre el nivel del mar.
 - **activeRunway:** Entero con el designador de la pista activa preferente.
 - **latitude:** Número con decimales con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Número con decimales con la longitud geográfica. Este son valores positivos, oeste negativos.
 - **designatorOfRunwayExitTaxiway:** Texto con el identificador de la calle de rodaje utilizada por las aeronaves para abandonar la pista tras aterrizar.

```
{
  "icao": "LEZL",
  "name": "Aeropuerto de Sevilla",
  "elevation": 111,
  "activeRunway": 27,
  "latitude": 37.419722,
  "longitude": -5.893333,
  "designatorOfRunwayExitTaxiway": "E1"
}
```

- **airspace.json:** Describe los espacios aéreos asociados al aeropuerto. Contiene un objeto JSON con los siguientes atributos:

- **name:** Texto con el nombre del espacio aéreo.
- **airspaceCoords:** Lista de objetos JSON, donde cada objeto denota un punto del polígono que forma el espacio aéreo y está formado por:
 - **latitude:** Número con decimales con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Número con decimales con la longitud geográfica. Este son valores positivos, oeste negativos

```
[
  {
    "name": "LEZL CTR",
    "airspaceCoords": [
      {"latitude": 37.500556, "longitude": -6.078056},
      {"latitude": 37.501944, "longitude": -5.737000},
      {"latitude": 37.376667, "longitude": -5.736389},
      {"latitude": 37.376667, "longitude": -5.773333},
      {"latitude": 37.369171, "longitude": -5.777565},
      {"latitude": 37.361978, "longitude": -5.782560},
      {"latitude": 37.355108, "longitude": -5.788223},
      {"latitude": 37.348600, "longitude": -5.794521},
      {"latitude": 37.342492, "longitude": -5.801417},
      {"latitude": 37.336818, "longitude": -5.808873},
      {"latitude": 37.331610, "longitude": -5.816844},
      {"latitude": 37.326899, "longitude": -5.825285},
      {"latitude": 37.322713, "longitude": -5.834148},
      {"latitude": 37.319073, "longitude": -5.843382},
      {"latitude": 37.316003, "longitude": -5.852934},
      {"latitude": 37.313518, "longitude": -5.862750},
      {"latitude": 37.311627, "longitude": -5.872772},
      {"latitude": 37.310337, "longitude": -5.882943},
      {"latitude": 37.309652, "longitude": -5.893202},
      {"latitude": 37.309576, "longitude": -5.903492},
      {"latitude": 37.310106, "longitude": -5.913753},
      {"latitude": 37.311237, "longitude": -5.923929},
      {"latitude": 37.312961, "longitude": -5.933962},
      {"latitude": 37.315264, "longitude": -5.943796},
      {"latitude": 37.318130, "longitude": -5.953375},
      {"latitude": 37.321537, "longitude": -5.962646},
      {"latitude": 37.325456, "longitude": -5.971557},
      {"latitude": 37.329857, "longitude": -5.980056},
      {"latitude": 37.334707, "longitude": -5.988092},
      {"latitude": 37.339966, "longitude": -5.995616},
      {"latitude": 37.345596, "longitude": -6.002580},
      {"latitude": 37.351556, "longitude": -6.008938},
      {"latitude": 37.357801, "longitude": -6.014648},
      {"latitude": 37.364286, "longitude": -6.019669},
      {"latitude": 37.370964, "longitude": -6.023964},
      {"latitude": 37.375833, "longitude": -6.026259},
      {"latitude": 37.375556, "longitude": -6.076944}
    ]
  }
]
```

- **parkingRamp.json:** Describe la rampa de estacionamientos del aeropuerto. Contiene un objeto JSON con los siguientes atributos:
 - **designator:** Texto con el nombre de la plataforma.
 - **airspaceCoords:** Lista de objetos, donde cada objeto denota un punto del polígono que forma la rampa de estacionamientos y está formado por:
 - **latitude:** Número con decimales con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Número con decimales con la longitud geográfica. Este son valores positivos, oeste negativos

```
{
  "designator": "LEZL Parking",
  "polygonCoords": [
    { "latitude": 37.42147902660179, "longitude": -5.9071405059424364 },
    { "latitude": 37.42012544296888, "longitude": -5.907066403823222 },
    { "latitude": 37.42008620829564, "longitude": -5.898025945252385 },
    { "latitude": 37.42155749446823, "longitude": -5.896161041914155 },
    { "latitude": 37.42065510904379, "longitude": -5.895024809415645 },
    { "latitude": 37.42222446801068, "longitude": -5.893159906077443 },
    { "latitude": 37.425029615222115, "longitude": -5.89701321628732 },
    { "latitude": 37.42136132464873, "longitude": -5.901162934975019 },
    { "latitude": 37.42147902660179, "longitude": -5.9071405059424364 }
  ]
}
```

- **parkings.json:** Array de objetos JSON, cada objeto describe un estacionamiento y sus atributos son:
 - **designator:** Texto con el designador del estacionamiento.
 - **position:** Objeto JSON con los siguientes atributos:
 - **latitude:** Número con decimales con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Número con decimales con la longitud geográfica. Este son valores positivos, oeste negativos
 - **parkingType:** Texto, sólo puede valer “VFR” o “IFR”. Indica qué tipo de aeronave puede utilizar el estacionamiento, de esta forma separamos aviación comercial de aviación general.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

```
[
  {
    "designator": "GA_R1_01",
    "position": { "latitude": 37.421287900767226, "longitude": -5.906110132196972 },
    "parkingType": "VFR"
  },
  {
    "designator": "GA_R1_02",
    "position": { "latitude": 37.42112961575444, "longitude": -5.906102250126594 },
    "parkingType": "VFR"
  },
  {
    "designator": "GA_R1_03",
    "position": { "latitude": 37.42097281548092, "longitude": -5.906103571586613 },
    "parkingType": "VFR"
  },
  {
    "designator": "GA_R1_04",
    "position": { "latitude": 37.4208172126131, "longitude": -5.90610694961677 },
    "parkingType": "VFR"
  },
  {
    "designator": "GA_R1_05",
    "position": { "latitude": 37.42068702129184, "longitude": -5.9061035644522235 },
    "parkingType": "VFR"
  },
  {
    "designator": "GA_R1_06",
    "position": { "latitude": 37.42049992133232, "longitude": -5.906096808464696 },
    "parkingType": "VFR"
  },
  {
    "designator": "IFR_R3_01",
    "position": { "latitude": 37.421343039228205, "longitude": -5.900655115494061 },
    "parkingType": "IFR"
  },
  {
    "designator": "IFR_R3_02",
    "position": { "latitude": 37.42161602667848, "longitude": -5.900378284067699 },
    "parkingType": "IFR"
  },
  {
    "designator": "IFR_R3_03",
    "position": { "latitude": 37.42192151591546, "longitude": -5.89991305872158 },
    "parkingType": "IFR"
  },
  {
    "designator": "IFR_R3_04",
    "position": { "latitude": 37.422189041909576, "longitude": -5.899532198953835 },
    "parkingType": "IFR"
  },
  {
    "designator": "IFR_R3_05",
    "position": { "latitude": 37.422517704428984, "longitude": -5.899217422464261 },
    "parkingType": "IFR"
  },
  {
    "designator": "IFR_R3_06",
```

```
"position": { "latitude": 37.422860825624866, "longitude": -5.898934576311234 },
"parkingType": "IFR"
},
{
  "designator": "IFR_R3_07",
  "position": { "latitude": 37.42308959950773, "longitude": -5.898498581976071 },
  "parkingType": "IFR"
},
{
  "designator": "IFR_R3_08",
  "position": { "latitude": 37.4234628688704, "longitude": -5.897977435900117 },
  "parkingType": "IFR"
},
{
  "designator": "IFR_R3_09",
  "position": { "latitude": 37.423882324981875, "longitude": -5.897394462774571 },
  "parkingType": "IFR"
}
]
```

- **runways.json:** Array de objetos JSON, cada uno representando una pista física (cada pista física tiene a su vez 2 designadores de pista, uno en cada dirección). Por convención, las pistas siempre deben llevar el designador más pequeño posible. Además, las coordenadas de la pista se listarán siempre comenzando por la pista con el designador de menor número, por lo que la primera coordenada estará al lado del designador de menor número y la segunda coordenada al lado del designador de mayor número. La figura A.7 ilustra la explicación.
 - **name:** Texto con el designador de la pista, este debe de ser el de menor número posible de los 2 designadores que tiene cada pista.
 - **heading:** Entero con el rumbo de la pista.
 - **longitud:** Entero con la longitud en metros.
 - **width:** Entero con el ancho en metros.
 - **lat1:** Número con decimales con la latitud geográfica de la primera esquina de la pista. Norte son valores positivos, sur negativos.
 - **lon1:** Número con decimales con la longitud geográfica de la primera esquina de la pista. Este son valores positivos, oeste negativos.
 - **lat2:** Número con decimales con la latitud geográfica de la segunda esquina de la pista. Norte son valores positivos, sur negativos.

- **lon2:** Número con decimales con la longitud geográfica de la segunda esquina de la pista. Este son valores positivos, oeste negativos.
- **lat3:** Número con decimales con la latitud geográfica de la tercera esquina de la pista. Norte son valores positivos, sur negativos.
- **lon3:** Número con decimales con la longitud geográfica de la tercera esquina de la pista. Este son valores positivos, oeste negativos.
- **lat4:** Número con decimales con la latitud geográfica de la cuarta esquina de la pista. Norte son valores positivos, sur negativos.
- **lon4:** Número con decimales con la longitud geográfica de la cuarta esquina de la pista. Este son valores positivos, oeste negativos.

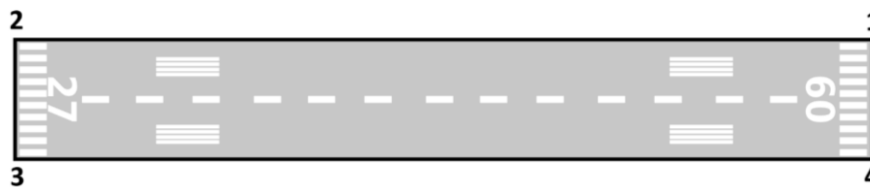


Figura A.7. Representación del orden de numeración de los vértices de la pista

```
[
  {
    "name": "09",
    "heading": 90,
    "length": 3364,
    "width": 45,
    "lat1": 37.418087810214296,
    "lon1": -5.912132123349252,
    "lat2": 37.41821366628157,
    "lon2": -5.874098426641054,
    "lat3": 37.417805389505084,
    "lon3": -5.8740971399757,
    "lat4": 37.417681644824455,
    "lon4": -5.912125859194904
  }
]
```

- **stopbars.json:** Lista de objetos JSON. Cada objeto describe una barra de parada mediante los siguientes atributos:
 - **designator:** Texto con el designador de la barra de parada.
 - **start:** Objeto con las coordenadas de inicio de la barra de parada.
 - **latitude:** Número con decimales con la latitud geográfica. Norte son valores positivos, sur negativos.

ANEXO A – Manual de usuario

- **longitude:** Número con decimales con la longitud geográfica.
Este son valores positivos, oeste negativos
- **end:** Objeto con las coordenadas de fin de la barra de parada.
 - **latitude:** Número con decimales con la latitud geográfica.
Norte son valores positivos, sur negativos.
 - **longitude:** Número con decimales con la longitud geográfica.
Este son valores positivos, oeste negativos

```
[
  {
    "designator": "HP2",
    "start": { "latitude": 37.4194601148416, "longitude": -5.913187839577461 },
    "end": { "latitude": 37.41931763172613, "longitude": -5.912863411039439 }
  },
  {
    "designator": "HP1",
    "start": { "latitude": 37.4189208755852, "longitude": -5.911350891249981 },
    "end": { "latitude": 37.41872267807008, "longitude": -5.911204707927652 }
  },
  {
    "designator": "E1",
    "start": { "latitude": 37.418740467727474, "longitude": -5.90316141968259 },
    "end": { "latitude": 37.418742100130885, "longitude": -5.902888056175186 }
  },
  {
    "designator": "E2",
    "start": { "latitude": 37.41875696843201, "longitude": -5.898354185370835 },
    "end": { "latitude": 37.41887701417561, "longitude": -5.898004488815332 }
  },
  {
    "designator": "E3",
    "start": { "latitude": 37.418774047265586, "longitude": -5.892970704163076 },
    "end": { "latitude": 37.41889331760349, "longitude": -5.892708621861345 }
  },
  {
    "designator": "E5",
    "start": { "latitude": 37.41884797796594, "longitude": -5.887861653936426 },
    "end": { "latitude": 37.41879048756101, "longitude": -5.887475592904565 }
  },
  {
    "designator": "HP3",
    "start": { "latitude": 37.418860670684154, "longitude": -5.874453296051968 },
    "end": { "latitude": 37.41884327122857, "longitude": -5.87410058131934 }
  },
  {
    "designator": "HP4",
    "start": { "latitude": 37.419535016111936, "longitude": -5.873299640531712 },
    "end": { "latitude": 37.4196654003689, "longitude": -5.873064797663602 }
  }
]
```

- **taxiways.json:** Lista de objetos JSON, cada uno describe una calle de rodaje mediante los siguientes atributos:
 - **designator:** Texto con el designador de la calle de rodaje.
 - **centerlineCoords:** Lista de posiciones que describen la línea central de la calle de rodaje, cada una con:
 - **latitude:** Número con decimales con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Número con decimales con la longitud geográfica. Este son valores positivos, oeste negativos

```
[
  {
    "designator": "HP1",
    "centerLineCoords": [
      { "latitude": 37.41790450316219, "longitude": -5.911688708530761 },
      { "latitude": 37.418084544024225, "longitude": -5.911865937183677 },
      { "latitude": 37.41836606159667, "longitude": -5.911919517939452 },
      { "latitude": 37.418624664015, "longitude": -5.911742289286565 },
      { "latitude": 37.41883089061977, "longitude": -5.911293035256563 },
      { "latitude": 37.41976930539995, "longitude": -5.909254575784843 }
    ]
  },
  {
    "designator": "HP2",
    "centerLineCoords": [
      { "latitude": 37.41789359012108, "longitude": -5.910891086462897 },
      { "latitude": 37.41802907525651, "longitude": -5.911589185157368 },
      { "latitude": 37.41847096353354, "longitude": -5.91258384457592 },
      { "latitude": 37.41865827495455, "longitude": -5.912860965301803 },
      { "latitude": 37.41892715785367, "longitude": -5.913110286264242 },
      { "latitude": 37.41923564247158, "longitude": -5.913154901594595 },
      { "latitude": 37.419413098263504, "longitude": -5.913065433821856 },
      { "latitude": 37.41960277353658, "longitude": -5.912868601482643 },
      { "latitude": 37.41969031580743, "longitude": -5.912627153813929 },
      { "latitude": 37.4197546803548, "longitude": -5.9121424062845165 },
      { "latitude": 37.419767343914884, "longitude": -5.909257638914568 }
    ]
  },
  {
    "designator": "A1_1",
    "centerLineCoords": [
      { "latitude": 37.41976727277809, "longitude": -5.909255182109575 },
      { "latitude": 37.41978327890648, "longitude": -5.905313344095447 }
    ]
  },
  {
    "designator": "A1_2",
    "centerLineCoords": [
      { "latitude": 37.41978343668234, "longitude": -5.905312424158552 },
      { "latitude": 37.4197881573964, "longitude": -5.904153840794095 }
    ]
  }
]
```

ANEXO A – Manual de usuario

```
},
{
  "designator": "A1_3",
  "centerLineCoords": [
    { "latitude": 37.41978793704885, "longitude": -5.904153653029226 },
    { "latitude": 37.41979236293976, "longitude": -5.9030383245042515 }
  ]
},
{
  "designator": "A2_1",
  "centerLineCoords": [
    { "latitude": 37.419792096281626, "longitude": -5.903036645729884 },
    { "latitude": 37.41980062544876, "longitude": -5.900959050102358 }
  ]
},
{
  "designator": "A2_2",
  "centerLineCoords": [
    { "latitude": 37.41980040818815, "longitude": -5.9009582847248225 },
    { "latitude": 37.41980412759925, "longitude": -5.898721548630647 }
  ]
},
{
  "designator": "A3_1",
  "centerLineCoords": [
    { "latitude": 37.41980438150665, "longitude": -5.898719625975247 },
    { "latitude": 37.419812644163656, "longitude": -5.896664941041934 }
  ]
},
{
  "designator": "A4_1",
  "centerLineCoords": [
    { "latitude": 37.41981297442227, "longitude": -5.896663840795611 },
    { "latitude": 37.419822820199016, "longitude": -5.893588035103704 }
  ]
},
{
  "designator": "A4_2",
  "centerLineCoords": [
    { "latitude": 37.4198227626522, "longitude": -5.893587347787701 },
    { "latitude": 37.419843102341275, "longitude": -5.887434127618661 }
  ]
},
{
  "designator": "A5",
  "centerLineCoords": [
    { "latitude": 37.4198431015359, "longitude": -5.887433438956265 },
    { "latitude": 37.41988305665235, "longitude": -5.8757971304133605 }
  ]
},
{
  "designator": "HP3",
  "centerLineCoords": [
    { "latitude": 37.41988301310768, "longitude": -5.875796179367995 },
    { "latitude": 37.41984521246437, "longitude": -5.87545849866774 },
    { "latitude": 37.419743939278874, "longitude": -5.875060150279779 },
    { "latitude": 37.419628754023705, "longitude": -5.874763287938862 },
    { "latitude": 37.419507344581874, "longitude": -5.874562770752533 },
  ]
}
```

```

    { "latitude": 37.4193638609714, "longitude": -5.874410716627153 },
    { "latitude": 37.419270272490536, "longitude": -5.8743506643215255 },
    { "latitude": 37.41906959679564, "longitude": -5.874270216893478 },
    { "latitude": 37.41885253627592, "longitude": -5.874258817185506 },
    { "latitude": 37.41868409412757, "longitude": -5.874287977326247 },
    { "latitude": 37.41846275175584, "longitude": -5.874439707935977 },
    { "latitude": 37.41828062006313, "longitude": -5.874691511997639 },
    { "latitude": 37.418010602519814, "longitude": -5.8751806510570646 }
  ]
},
{
  "designator": "HP4",
  "centerLineCoords": [
    { "latitude": 37.419883179268666, "longitude": -5.875795491591788 },
    { "latitude": 37.41988567406355, "longitude": -5.874138579421867 },
    { "latitude": 37.4198585413298, "longitude": -5.87381186735098 },
    { "latitude": 37.41983053746871, "longitude": -5.87362701801527 },
    { "latitude": 37.419727856555724, "longitude": -5.873320360446797 },
    { "latitude": 37.419605657599206, "longitude": -5.873171839883071 },
    { "latitude": 37.419458114535374, "longitude": -5.8730632607655195 },
    { "latitude": 37.41928079119569, "longitude": -5.873025531334605 },
    { "latitude": 37.41903868460392, "longitude": -5.873089640808075 },
    { "latitude": 37.41877472781243, "longitude": -5.873345418959303 },
    { "latitude": 37.418584607388354, "longitude": -5.873655246055961 },
    { "latitude": 37.41800696997029, "longitude": -5.874915994798698 }
  ]
},
{
  "designator": "G1",
  "centerLineCoords": [
    { "latitude": 37.41978324539231, "longitude": -5.905270762598462 },
    { "latitude": 37.4203774278546, "longitude": -5.905272068271358 }
  ]
},
{
  "designator": "G2",
  "centerLineCoords": [
    { "latitude": 37.4197868351426, "longitude": -5.904154951043665 },
    { "latitude": 37.42037150233304, "longitude": -5.904155839306355 }
  ]
},
{
  "designator": "G3",
  "centerLineCoords": [
    { "latitude": 37.41979273653578, "longitude": -5.903035053831871 },
    { "latitude": 37.420505082624345, "longitude": -5.903040659112548 }
  ]
},
{
  "designator": "G4",
  "centerLineCoords": [
    { "latitude": 37.41979832905915, "longitude": -5.900959478195119 },
    { "latitude": 37.420520838396584, "longitude": -5.90095798927328 }
  ]
},
{
  "designator": "G5",
  "centerLineCoords": [

```

ANEXO A – Manual de usuario

```
[
  { "latitude": 37.419805973482866, "longitude": -5.898716866507499 },
  { "latitude": 37.420544317865904, "longitude": -5.899110832277785 }
],
{
  "designator": "G6",
  "centerLineCoords": [
    { "latitude": 37.419805650571035, "longitude": -5.89871693127796 },
    { "latitude": 37.42036176515178, "longitude": -5.898023429845381 }
  ]
},
{
  "designator": "G7",
  "centerLineCoords": [
    { "latitude": 37.41981261989922, "longitude": -5.89658272377639 },
    { "latitude": 37.42099821691771, "longitude": -5.895082167278275 }
  ]
},
{
  "designator": "G8",
  "centerLineCoords": [
    { "latitude": 37.41982252832665, "longitude": -5.8935871592688045 },
    { "latitude": 37.420999069009966, "longitude": -5.8950757726355505 }
  ]
},
{
  "designator": "E1",
  "centerLineCoords": [
    { "latitude": 37.41791184059477, "longitude": -5.903036675211666 },
    { "latitude": 37.418741921355746, "longitude": -5.903030641964506 },
    { "latitude": 37.419790676557184, "longitude": -5.903036422966238 }
  ]
},
{
  "designator": "E2",
  "centerLineCoords": [
    { "latitude": 37.417932412559324, "longitude": -5.8977205409516955 },
    { "latitude": 37.418802985895525, "longitude": -5.898192325584574 },
    { "latitude": 37.4198039325766, "longitude": -5.898720313114978 }
  ]
},
{
  "designator": "E3",
  "centerLineCoords": [
    { "latitude": 37.417950516518545, "longitude": -5.892176375367711 },
    { "latitude": 37.4188329223215, "longitude": -5.892838406770352 },
    { "latitude": 37.4198233413471, "longitude": -5.893587900683684 }
  ]
},
{
  "designator": "E5",
  "centerLineCoords": [
    { "latitude": 37.41796988246901, "longitude": -5.887858061706652 },
    { "latitude": 37.4198422543238, "longitude": -5.8874324058935485 }
  ]
}
]
```

- **vfrPoints.json:** Lista de objetos JSON, cada uno describe un punto de notificación visual mediante los siguientes atributos:
 - **name:** Texto con el designador del punto de notificación visual.
 - **latitude:** Número con decimales con la latitud geográfica. Norte son valores positivos, sur negativos.
 - **longitude:** Número con decimales con la longitud geográfica. Este son valores positivos, oeste negativos

```
[
  {
    "name": "N",
    "latitude": 37.5561750011148,
    "longitude": -5.877915209800789
  },
  {
    "name": "S",
    "latitude": 37.29132004909086,
    "longitude": -5.921677289281501
  }
]
```

A.7 Consejos

Dé prioridad a las aeronaves en el aire, las que están en tierra pueden esperar, pero las que están en el aire generalmente no y están consumiendo más combustible.

Suele ser más conveniente dar prioridad a las aeronaves bajo reglas IFR debido a que, en caso de tener que realizar motor y al aire, su consumo de combustible y tiempo en el aire es mucho más alto que si tiene que realizar motor y al aire una aeronave bajo reglas VFR.

Recuerde revisar periódicamente la zona de aproximación final del aeropuerto. Es habitual al principio enfocarse en las aeronaves visuales en circuito y olvidarse de comprobar si tenemos alguna aeronave IFR establecida en final.

Para agilizar el movimiento de las aeronaves visuales puede aprobarles virar a base y final estando todavía en viento en cola para que, cuando alcancen el punto seguro de giro, giren solas. Para ello seleccione la aeronave a la que desea aprobar virar a base y final y seleccione la acción.

ANEXO A – Manual de usuario

Recuerde que, generalmente, las aeronaves que vuelan bajo reglas VFR son más lentas que las que vuelan bajo reglas IFR debido a que las primeras suelen ser aviones pequeños de hélice y las segundas jets. Con esto en mente, evite que se produzcan posibles alcances.

En aviación ante la duda siempre es mejor ser precavido a arriesgar la seguridad de la operativa.

Simulador de ATC en posición de torre como entrenador para la red de vuelo IVAO

Anexo B – Manual del programador

B.1 Introducción

Este anexo describe información necesaria y útil para el programador que desee revisar o ampliar la aplicación. Se realiza una descripción del sistema y de la documentación de las librerías utilizadas.

B.2 Entorno y librerías utilizadas

En este apartado se describe el entorno y librerías utilizadas en el desarrollo del proyecto. La base del proyecto es un proyecto Maven con el arquetipo simple de JavaFX construido en el entorno de desarrollo integrado Eclipse. El futuro programador puede utilizar otro entorno o actualizar las librerías, pero debe tener en cuenta el contexto en que fue desarrollada la aplicación inicialmente:

- **Eclipse:** El proyecto fue realizado utilizando el IDE Eclipse en su versión 2025-03 (4.35.0).
- **Java:** La versión del JDK utilizado es la 22. El JRE es Java 22.0.2 (OpenJDK 22)
- **JavaFX:** La versión de JavaFX que se referencia en los FXML es la versión 21 [24]
- **Apache Maven:** Se utilizó la versión 3.9.9 de Maven en su variante embebida en el proyecto de Eclipse [25].
- **GSON:** Para el trabajo con archivos en formato JSON se utilizó la versión 2.10.1 de la librería [26]. No recomiendo utilizar versiones más nuevas ya que se producen errores de compilación, presuntamente por incompatibilidad entre la librería y el sistema modular empleado en el proyecto (module-info.java)
- **JUnit:** Para la ejecución de pruebas unitarias, se utilizó la versión 5.13.4 [27].
- **JUnit Jupiter Engine:** Esta dependencia es utilizada por Maven para lanzar las pruebas de JUnit.
- **Maven Surefire:** Este plugin es utilizado por Maven para ejecutar las pruebas mediante el ciclo de vida de Maven [28].

B.3 Código fuente

El código fuente está estructurado en paquetes para facilitar la claridad, encapsulación y separación de responsabilidades. De esta forma, en el apartado de código en la carpeta “src/main/java” encontrará los siguientes paquetes:

- **es.uned.pfg.adcsim.app:** Aquí se encuentra la clase principal que hace de punto de entrada a la aplicación.
- **es.uned.pfg.adcsim.controller:** Contiene las clases controladoras de las diferentes vistas que forman la interfaz gráfica.
- **es.uned.pfg.adcsim.data.service:** Siguiendo el patrón estrategia, aquí están las interfaces que deben implementar las clases que se quieran dedicar a la entrada/salida de datos de la aplicación.
- **es.uned.pfg.adcsim.data.service.impl:** Aquí se encuentran las diferentes implementaciones de los servicios de entrada/salida de datos
- **es.uned.pfg.adcsim.model.ai:** Contiene los enum que forman el modelo del agente. Dictan los posibles estados en los que puede estar el agente, posibles solicitudes que puede realizar, y posibles instrucciones que puede recibir del controlador.
- **es.uned.pfg.adcsim.model.aircraft:** Contiene las clases relacionadas con el modelo de dominio de las aeronaves.
- **es.uned.pfg.adcsim.model.airport:** Contiene las clases relacionadas con el modelo de dominio de los aeropuertos. Cada una de ellas describe una parte del mismo.
- **es.uned.pfg.adcsim.model.session:** Aquí se encuentran las clases que describen los datos que se obtienen de una sesión de entrenamiento. Se utilizan especialmente por gson para la serialización/deserialización.
- **es.uned.pfg.adcsim.service.ai:** Aquí están las clases relacionadas con la lógica de negocio del agente que controla las aeronaves.
- **es.uned.pfg.adcsim.simulation:** Contiene las clases relacionadas con el motor de simulación, tanto el propio motor de simulación como el supervisor de infracciones. También contiene las interfaces que deben implementar las clases que deseen suscribirse a actualizaciones del motor de simulación o al supervisor de infracciones siguiendo el patrón observador.
- **es.uned.pfg.adcsim.simulation.traffic:** Contiene lo relacionado con la lógica de inyección de aeronaves en la simulación.

Anexo B – Manual del programador

- **es.uned.pfg.adcsim.util:** Aquí se encuentran las clases utilitarias empleadas por el resto de clases de la aplicación.
- **es.uned.pfg.adcsim.view:** Contiene las clases relacionadas con elementos de la GUI que no son implementados mediante FXML, el renderizado del radar en el canvas de JavaFX y la implementación de la ventana de diálogo para la selección de la ruta de rodaje.

Siguiendo la estructura de un proyecto Maven, los recursos que no son código Java, pero son empaquetados con la aplicación se encuentran en “src/main/resources/es/uned/pfg/adcsim”. Aquí tenemos 3 directorios:

- **font:** Contiene la fuente personalizada utilizada para la pantalla radar.
- **fxml:** Contiene los archivos FXML que JavaFX utiliza para renderizar las vistas.
- **img:** Contiene las imágenes utilizadas en la aplicación.

B.4 Documentación

Durante el desarrollo se ha documentado todo el código empleando JavaDoc, por lo que puede encontrar numerosas explicaciones a algoritmos y funcionalidad del código en cada clase.