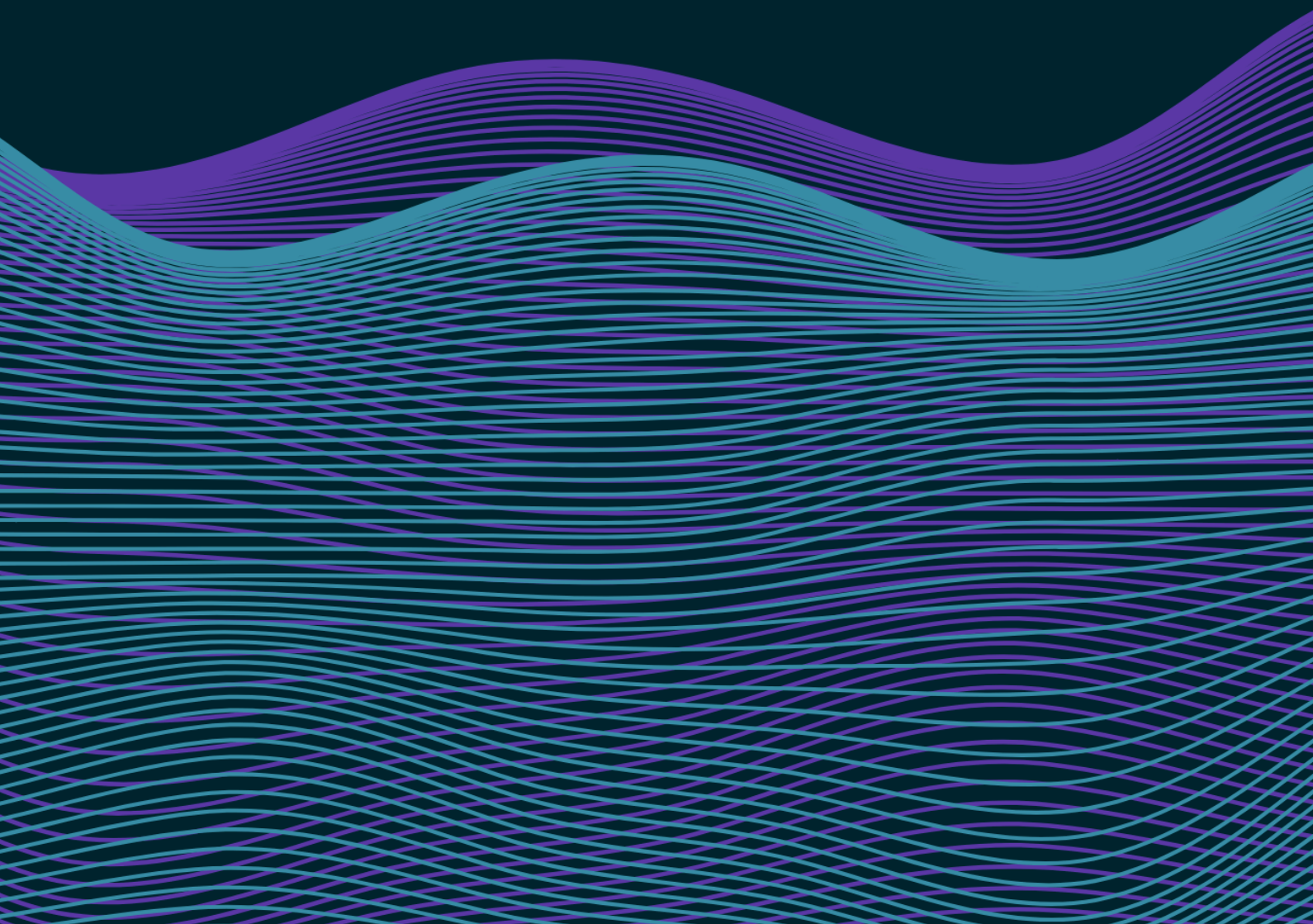
DAM2: Programación de servicios y procesos

Curso 2025-2026

SIMULACIÓN DE TRÁFICO URBANO

Fecha de entrega: 18/01/2026

Por: Andrés Alejandro Pacheco Castillo y Luis Miguel Agra Álvarez



Índice

Índice **1**
1. Introducción **2**
2. Descripción de tecnologías usadas y diseño de la arquitectura del sistema. **3**
2.1. Tecnologías utilizadas 3
2.2. Diseño de la arquitectura 3
3. Descripción de las funcionalidades con capturas del proceso **5**
3.1. Pantalla de Simulación 6
3.2. Pantalla de Configuración 6
3.3. Estadísticas 7
3.4. Historial 7
4. Breve Manual de usuario **7**
4.1. Cómo importar o instalar el proyecto 7
Opción 1: Instalación mediante APK 7
Opción 2: Importar el proyecto desde el repositorio 8
4.2. Cómo utilizar la aplicación 8
4.3. Botones y eventos 9
4.4. Posibles errores 9
5. Pruebas **9**
6. Conclusiones y dificultades encontradas **12**
7. Referencias **12**
8. Anexos obligatorios **13**
ANEXO: Uso de herramientas de IA en el desarrollo del proyecto 13
Introducción 13
Aclaración de conceptos de programación y del módulo PSP 14
Generación y revisión de código de la simulación 14
Resolución de errores y dudas técnicas puntuales 15
Apoyo en la organización del proyecto y arquitectura 16
Uso de IA en tareas accesorias 16

1. Introducción

Este proyecto consiste en el desarrollo de una aplicación Android que implementa un **simulador de tráfico urbano sencillo**, con un enfoque principalmente didáctico. El objetivo del sistema es recrear de forma simplificada el comportamiento del tráfico en una ciudad, modelando elementos como vehículos, semáforos, eventos y estadísticas, y permitiendo al usuario interactuar con la simulación mientras esta se ejecuta.

El proyecto se ha realizado dentro del módulo **Programación de Servicios y Procesos (PSP)**, aplicando conceptos relacionados con la gestión del estado, la ejecución concurrente y la separación de responsabilidades entre los distintos componentes del sistema. La aplicación sirve como apoyo para entender cómo distintos factores pueden influir en el flujo del tráfico y cómo se puede organizar la lógica de una simulación de este tipo.

La simulación permite configurar distintos escenarios de tráfico y añadir eventos que modifican el comportamiento normal de los vehículos, como accidentes, obras, congestiones o situaciones de emergencia. Además, el sistema recopila y muestra estadísticas en tiempo real que permiten observar la evolución de la simulación y detectar comportamientos anómalos o cuellos de botella.

Para su implementación se ha utilizado el lenguaje **Kotlin** y el framework **Jetpack Compose**, organizando el proyecto siguiendo una arquitectura **MVVM (Model–View–ViewModel)** que facilita la separación entre la lógica de la simulación y la gestión del estado de la aplicación. El desarrollo se ha gestionado mediante **Git y GitHub**, y se han incluido pruebas unitarias básicas para comprobar el correcto funcionamiento de partes concretas del sistema.

En los siguientes apartados se describen las tecnologías empleadas, el diseño de la arquitectura, las funcionalidades principales de la aplicación, el manual de usuario, las pruebas realizadas y las conclusiones obtenidas tras el desarrollo del proyecto.

2. Descripción de tecnologías usadas y diseño de la arquitectura del sistema.

2.1. Tecnologías utilizadas

Aquí está el listado de las tecnologías utilizadas.

- Lenguaje principal: Kotlin
- Entorno de desarrollo: Android Studio Otter 2 Feature Drop | 2025.2.2 (basado en JetBrains IntelliJ IDEA).
- Máquina Virtual de Java (JVM): OpenJDK 21 para compilación y ejecución del proyecto.
- Sistema de construcción: Gradle 8.14.3.
- Interfaz gráfica: Jetpack Compose
- Plataforma objetivo: Android SDK
- Control de versiones: Git + GitHub.

2.2. Diseño de la arquitectura

El proyecto adopta una **arquitectura basada en MVVM (Model-View-ViewModel)** adaptada al uso de Jetpack Compose

2.2.1. Capa de Modelo

La capa de modelo agrupa los datos y la lógica principal de la simulación y se encuentra organizada dentro del paquete model. En esta capa se diferencian dos responsabilidades principales: la lógica de negocio de la simulación y la gestión de datos persistentes.

Dentro del subpaquete model.domain se concentra la lógica central del sistema. El archivo Models.kt define las estructuras de datos utilizadas en la simulación, como los vehículos, los semáforos, los distintos estados y las métricas recogidas durante la ejecución. El archivo SimulationEngine.kt contiene el núcleo de la simulación, donde se implementan las reglas de movimiento, la evolución temporal del sistema y el control general del tráfico. Por su parte, TrafficLightRuntime.kt se encarga de gestionar el comportamiento específico de los semáforos y sus cambios de estado a lo largo del tiempo.

El subpaquete model.data está orientado a la persistencia de información. En él, HistoryManager.kt gestiona el almacenamiento y la recuperación del historial de simulaciones realizadas, mientras que PreferencesManager.kt se ocupa de guardar y cargar las preferencias de configuración definidas por el usuario. Esta separación permite mantener aislada la lógica de la simulación de los aspectos relacionados con el almacenamiento de datos.

2.2.2. Capa de Vista

La capa de vista se encuentra en el paquete view y es la encargada exclusivamente de la presentación visual y de la interacción con el usuario. No contiene lógica de negocio, sino que se limita a mostrar la información y a enviar eventos al ViewModel.

Esta capa se organiza en distintas pantallas:

- **SimulationScreen.kt**, que representa la pantalla principal de la simulación.
- **ConfigScreen.kt**, desde donde se configuran los parámetros y escenarios de la simulación.
- **StatsScreen.kt**, que muestra las estadísticas en tiempo real.

- **HistoryScreen.kt**, donde se visualiza el historial de simulaciones anteriores.

Además, el subpaquete view.theme contiene los archivos relacionados con el tema visual de la aplicación, como colores, tipografías y estilos generales.

2.2.3. Capa de ViewModel

La capa de ViewModel se encuentra en el paquete viewmodel y actúa como intermediaria entre la vista y el modelo.

El archivo **SimulationViewModel.kt** es el encargado de:

- Gestionar el estado global de la simulación.
- Exponer datos observables a la interfaz.
  Recibir las acciones del usuario desde las distintas pantallas.
- Coordinar la comunicación con la lógica de simulación y los gestores de datos.

Gracias a esta capa, se evita que la lógica del sistema dependa directamente de la interfaz.

2.2.4. Punto de entrada

El archivo **MainActivity.kt** constituye el punto de entrada de la aplicación. Se encarga de inicializar la app, configurar el contenido de Jetpack Compose y vincular las distintas pantallas con el SimulationViewModel correspondiente.

3. Descripción de las funcionalidades con capturas del proceso.

Esta aplicación permite recrear una simulación muy básica de una ciudad. Los principales actores de esta simulación son los edificios, los

vehículos y las calles.

Aumentando su complejidad y cumpliendo con los requisitos, se han añadido semáforos funcionales, distintos tipos de vehículos como coches y ambulancias y eventos. En este último caso presentamos 4 distintos: accidentes, obras, congestiones de tráfico y emergencias.

Los eventos son una parte interesante de la simulación, puesto que modifican el comportamiento que tendrían los vehículos en caso de que estos no ocurrieran. Por defecto, están configurados para que ocurran de modo aleatorio y cada pocos segundos.

3.1. Pantalla de Simulación

Esta es la pantalla principal del proyecto. En ellas podemos ver una representación muy simplificada del mapa de una ciudad simulando calles, semáforos y vehículos en un grid de 10x10. También se presentan unos botones de control que nos permiten reiniciar e iniciar la simulación, así como añadir a mano 4 tipos de eventos que son: accidente, obras, congestión y emergencia. Además, tenemos a nuestra disposición las estadísticas en tiempo real de la simulación actual.

3.2. Pantalla de Configuración

Como pantalla de configuración, tenemos a nuestra disposición una serie de herramientas para cambiar el comportamiento de la simulación. Las opciones principales para modificar el comportamiento de nuestra simulación son el bloque de “Escenarios Predefinidos” y el de “Opciones Avanzadas”.

- Los escenarios predefinidos nos dan la oportunidad de hacer una configuración rápida sin tener que profundizar en el resto de opciones. Es ideal para probar la aplicación y es recomendable empezar por la opción de tráfico ligero.

- Las opciones avanzadas tienen dos de las opciones más importantes del simulador, que son la posibilidad de anular las colisiones y la de evitar que los eventos se ejecuten de forma periódica y aleatoria.

Otros de los bloques de configuración son la “Configuración de Vehículos, la “Velocidad de Simulación” y la “Configuración de Semáforos”.

3.3. Estadísticas

En esta pantalla se presenta la versión completa de las estadísticas en tiempo real. Estas tienen un aspecto visual atractivo y es una herramienta para comprobar si la simulación funciona correctamente a lo largo del tiempo.

3.4. Historial

En esta pantalla podemos ver información básica de las últimas simulaciones relevantes que se han ejecutado.
Se han añadido unos botones extra para limpiar dicho historial y otro para compartir la información en formato texto simplificado.

4. Breve Manual de usuario

4.1. Cómo importar o instalar el proyecto

Existen dos formas principales de utilizar la aplicación: **mediante la instalación de la APK** o **importando el proyecto desde el repositorio** para ejecutarlo desde el entorno de desarrollo.

Opción 1: Instalación mediante APK

La forma más sencilla de probar la aplicación es mediante la instalación directa de la APK en un dispositivo Android.

En primer lugar, es necesario descargar el archivo APK proporcionado junto con la entrega del proyecto. Una vez descargado, la APK puede copiarse al

dispositivo Android o descargarse directamente desde el navegador del propio dispositivo.

Antes de instalar la aplicación, el sistema Android puede solicitar permiso para instalar aplicaciones de orígenes externos. En ese caso, se debe acceder a los ajustes de seguridad del dispositivo y permitir temporalmente la instalación desde el navegador o gestor de archivos utilizado.

Tras conceder el permiso, basta con abrir el archivo APK y completar el proceso de instalación. Una vez finalizado, la aplicación aparecerá en el listado de aplicaciones del dispositivo y podrá ejecutarse con normalidad.

Opción 2: Importar el proyecto desde el repositorio

También es posible ejecutar la aplicación importando el proyecto completo desde su repositorio en GitHub.

Para ello, se debe clonar o descargar el repositorio del proyecto y abrirlo con **Android Studio Otter 2 Feature Drop (2025.2.2)**, que es la versión utilizada durante el desarrollo. Una vez abierto el proyecto, Android Studio iniciará automáticamente la sincronización de Gradle.

Cuando el proceso de sincronización haya finalizado, la aplicación podrá ejecutarse en un emulador de Android o en un dispositivo físico conectado al equipo.

4.2. Cómo utilizar la aplicación

Una vez tengamos el proyecto abierto en nuestro Android (sea en emulador o en dispositivo físico) ya se nos presentará la pantalla de Simulación. Recomendamos dos pasos previos antes de Iniciar una emulación.

- **Primer paso:** arrastrar el grid del mapa de tráfico hacia el centro de la pantalla para tener una visión más cómoda del área de simulación.

- **Segundo paso:** acceder a la pantalla de configuración, seleccionar el escenario predefinido de **tráfico ligero** y desactivar las colisiones en las opciones avanzadas.

Estos dos pasos permitirán una primera experiencia de simulación más fluida y permitirá tener una mejor experiencia.

4.3. Botones y eventos

La aplicación dispone de varios botones de control que permiten interactuar con la simulación:

- **Iniciar simulación:** comienza la ejecución del tráfico y los eventos configurados.
- **Reiniciar simulación:** detiene la simulación actual y restablece el estado inicial.
- **Añadir evento:** permite introducir manualmente distintos tipos de eventos, como accidentes, obras, congestiones de tráfico o emergencias.

Además, durante la ejecución se muestran estadísticas en tiempo real que permiten seguir la evolución de la simulación.

4.4. Posibles errores

Uno de los errores más habituales es que el tráfico termine atascándose debido a la acumulación de eventos o a una configuración con demasiado tráfico y pocas calles disponibles. En estas situaciones, es relativamente común que se forme un conjunto de vehículos detenidos sin posibilidad de avanzar hacia su destino, lo que acaba provocando que la simulación quede sin actividad.

Cuando esto ocurre, se recomienda **reiniciar la simulación** o ajustar la configuración reduciendo la densidad de tráfico o el número de eventos activos.

5. Pruebas

El objetivo de este apartado es comprobar el correcto funcionamiento de la aplicación y verificar que los distintos componentes del sistema se comportan de forma estable ante diferentes situaciones de uso. Para ello, se ha seguido un enfoque combinado de **pruebas de caja blanca** (*SimulationEngineTest.kt*) y **pruebas de caja negra** (*SimulationUiTest.kt*), intentando aplicar ambos tipos de test de forma coherente a lo largo del desarrollo del proyecto.

- **Pruebas de caja blanca:** se han centrado en la verificación del comportamiento interno de la lógica de la simulación, revisando el flujo de ejecución, la gestión del estado y las condiciones que afectan al movimiento de los vehículos, la activación de eventos y la actualización de estadísticas. Este tipo de pruebas ha permitido detectar comportamientos no deseados en determinadas configuraciones y comprobar que los cambios realizados en la lógica no introducían errores adicionales.
- **Pruebas de caja negra:** se han orientado a validar el comportamiento observable de la aplicación desde el punto de vista del usuario, comprobando que las acciones realizadas desde la interfaz producen los resultados esperados, que la aplicación responde correctamente a las interacciones básicas y que la información mostrada en pantalla es coherente con el estado de la simulación.

Al tratarse de un **proyecto educativo de alcance limitado**, las pruebas realizadas no buscan cubrir todos los posibles casos ni alcanzar un nivel de exhaustividad propio de un entorno profesional. Aun así, se ha procurado comprobar el funcionamiento del sistema en situaciones habituales y en algunos casos problemáticos detectados durante el desarrollo, con el objetivo de mejorar la estabilidad general de la aplicación y asegurar que cumple correctamente los requisitos planteados.

Otras pruebas realizadas:

- Comprobar si el mapa puede salirse de los bordes de la pantalla sin crashear:
    - Funcionamiento correcto: este es un problema que nos encontramos en nuestras primeras versiones del proyecto. Arrastrar el mapa fuera de los bordes del simulador de Android Studio
- Comprobación con números altos:
    - Funcionamiento problemático: En situaciones de alto tráfico y pocas calles (grid reducido) hay mucha tendencia a que se creen agrupaciones de vehículos parados que acaban creando inactividad en la simulación. Esto sucede porque al no llegar los vehículos a sus destinos, estos no desaparecen y no permiten la aparición de nuevos vehículos.
- Comprobación de salida en formato UTF-8:
    - Funcionamiento correcto: el formato se imprime correctamente.

Conclusión de las pruebas

Tras la realización de las pruebas, se puede concluir que la aplicación funciona de forma correcta en los escenarios habituales de uso y cumple con los requisitos principales establecidos en el enunciado. Durante las pruebas no se han detectado errores críticos que impidan el uso normal de la aplicación.

En el proceso de verificación se han tenido en cuenta tanto **pruebas de caja blanca**, centradas en la lógica interna de la simulación, como **pruebas de caja negra**, orientadas al comportamiento observable desde la interfaz. Este enfoque ha permitido comprobar el funcionamiento general del sistema desde distintos puntos de vista.

Aun así, se es consciente de que el conjunto de pruebas no cubre la totalidad de los posibles casos. Debido a las limitaciones de tiempo propias del desarrollo del proyecto, se estima que la cobertura de pruebas ronda aproximadamente el **65 %** del comportamiento total de la aplicación. Esta cobertura se considera suficiente para un proyecto de carácter educativo, quedando la ampliación de pruebas como una posible mejora futura.

6. Conclusiones y dificultades encontradas

La aplicación consigue ejecutar lo exigido en los requisitos del proyecto. Funciona principalmente como MVP ya que necesita pulir varios aspectos. Igualmente, presentamos una interfaz con estética genérica pero con estructura y funcionalidad de buena calidad, resultando en una aplicación fácil de entender y de navegar para cualquier usuario.

El proyecto, al estar impulsado por IA, ha ayudado principalmente en la adquisición de experiencia de arquitectura, documentación y presentación del mismo.

Problemas encontrados a lo largo del proyecto:

- El más importante: conseguir una buena base de simulación. Encontrar la estructura de cómo se comporta la simulación ha sido difícil. Hemos conseguido una versión prometedora con movimientos fluidos, pero fuimos incapaces de conseguir un pathfinding lógico y no nos sirvió para continuar con el proyecto. El presentado, a pesar de su pobre atractivo visual (refiriéndonos solo al mapa), sí nos ha permitido construir el resto de la aplicación con los requisitos que se pedían.
- El pathfinding ha sido uno de los mayores problemas.
- La ausencia de tiempo, las versiones gratuitas de IA y la sensación de que el proyecto era demasiado para nosotros, nos ha generado una cantidad desagradable de frustración que ha mermado el resultado final de la aplicación.

7. Referencias

Android Developers. (2025). *Android Developers Documentation*. https://developer.android.com

JetBrains. (2025). *Kotlin Documentation*. https://kotlinlang.org/docs/home.html

JetBrains. (2025). *Android Studio Documentation*. https://developer.android.com/studio

Gradle Inc. (2025). *Gradle User Manual*. https://docs.gradle.org/current/userguide/userguide.html

OpenAI. (2025). *ChatGPT (modelo GPT-5.2)*. https://chat.openai.com

Anthropic. (2025). *Claude AI Assistant*. https://www.anthropic.com/claude

Consellería de Educación. IES Cotarelo Valledor. (2025). *Aula Virtual – Curso Programación de Servicios y Procesos (PSP). Apuntes del profesor*. https://centros.edu.xunta.gal/iescotarelovilagarcia/aulavirtual/

8. Anexos obligatorios

El siguiente anexo, ya que tiene que ver con las consultas realizadas a la IA, he decidido que la propia IA haga un resumen de los prompts utilizados.

ANEXO: Uso de herramientas de IA en el desarrollo del proyecto

Introducción

Durante el desarrollo de este proyecto se ha hecho uso de herramientas de Inteligencia Artificial como **ChatGPT** y **Claude** con fines de apoyo educativo. Estas herramientas se utilizaron como complemento al trabajo realizado por los integrantes del grupo, principalmente para aclarar conceptos, generar ejemplos de código, resolver dudas puntuales y ayudar en la redacción de la documentación.

Aunque se realizaron algunos intentos iniciales de implementación con **Claude**, el desarrollo final del proyecto, así como la mayor parte de las decisiones técnicas y del código definitivo, se llevaron a cabo utilizando **ChatGPT** como herramienta principal de apoyo. En ningún caso la IA sustituyó el trabajo personal, sino que se empleó como ayuda para mejorar la comprensión y acelerar ciertos procesos.

A continuación se describen distintos tipos de usos realizados, acompañados de ejemplos representativos de prompts y respuestas.

Aclaración de conceptos de programación y del módulo PSP

En varias fases del proyecto se utilizó la IA para comprender mejor conceptos relacionados con la programación concurrente, la organización del código y el enfoque general del módulo PSP, especialmente al inicio del desarrollo.

Ejemplo de prompt utilizado:
“Explícame de forma sencilla qué es el patrón MVVM y cómo se puede aplicar en una aplicación Android con Jetpack Compose.”

Ejemplo de respuesta resumida de la IA:
La IA explicó el patrón MVVM separando claramente el modelo, la vista y el ViewModel, indicando que la vista no debe contener lógica de negocio y que el ViewModel actúa como intermediario gestionando el estado. También se

describió cómo Jetpack Compose encaja bien con este patrón al trabajar de forma reactiva.

Generación y revisión de código de la simulación

La IA se utilizó para generar ejemplos de código relacionados con la simulación de tráfico, especialmente en fases tempranas o cuando se exploraban distintas soluciones posibles. En algunos casos se probaron implementaciones sugeridas por la IA que posteriormente fueron adaptadas o descartadas.

Ejemplo de prompt utilizado:
"Dame un ejemplo de código en Kotlin para simular el movimiento de vehículos en una cuadrícula con semáforos."

Ejemplo de respuesta resumida de la IA:
La IA propuso una estructura básica con clases para vehículos y semáforos, junto con un bucle de actualización temporal. Este código sirvió como referencia inicial, aunque la implementación final fue modificada para adaptarse a los requisitos reales del proyecto.

Resolución de errores y dudas técnicas puntuales

Durante el desarrollo surgieron errores relacionados con el estado de la simulación, la recomposición en Compose y el comportamiento inesperado de ciertos elementos. La IA fue utilizada para diagnosticar estos problemas y proponer posibles soluciones.

Ejemplo de prompt utilizado:
"Mi simulación se queda bloqueada cuando hay muchos vehículos, ¿qué puede estar pasando a nivel lógico?"

Ejemplo de respuesta resumida de la IA:
La IA sugirió que el problema podía estar relacionado con vehículos que no alcanzan su destino y no se eliminan, lo que provoca acumulaciones y bloqueos. Esta explicación ayudó a entender el comportamiento observado y a documentarlo como una limitación conocida del sistema.

Apoyo en la organización del proyecto y arquitectura

La IA también se empleó para revisar la organización de paquetes y mejorar la separación de responsabilidades dentro del proyecto, especialmente al aplicar la arquitectura MVVM.

Ejemplo de prompt utilizado:
"¿Tiene sentido separar la persistencia en un paquete distinto del modelo de dominio en un proyecto Android?"

Ejemplo de respuesta resumida de la IA:
La IA explicó que separar la lógica de negocio de la persistencia mejora la claridad del proyecto y facilita el mantenimiento, recomendando una estructura diferenciada como `domain` y `data`, enfoque que finalmente se adoptó en el proyecto.

Uso de IA en tareas accesorias

De forma puntual, la IA se utilizó para resolver dudas no directamente relacionadas con el código, pero sí con la entrega del proyecto y su presentación.

Ejemplo de prompt utilizado:

"¿Cómo evito que las listas numeradas rompan la numeración de los títulos en Google Docs?"

Ejemplo de respuesta resumida de la IA:

La IA explicó cómo separar estilos de título y listas normales, recomendando el uso de viñetas o texto corrido para evitar conflictos de numeración.

ANEXO: Distribución de tareas del trabajo en grupo

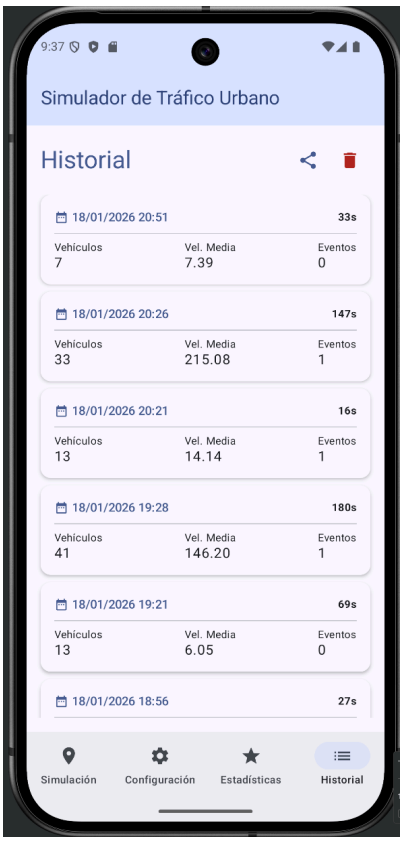
En este anexo se detalla la distribución de tareas realizada entre los integrantes del grupo durante el desarrollo del proyecto. Ambos miembros participaron activamente en distintas fases del trabajo, aunque con responsabilidades principales diferenciadas.

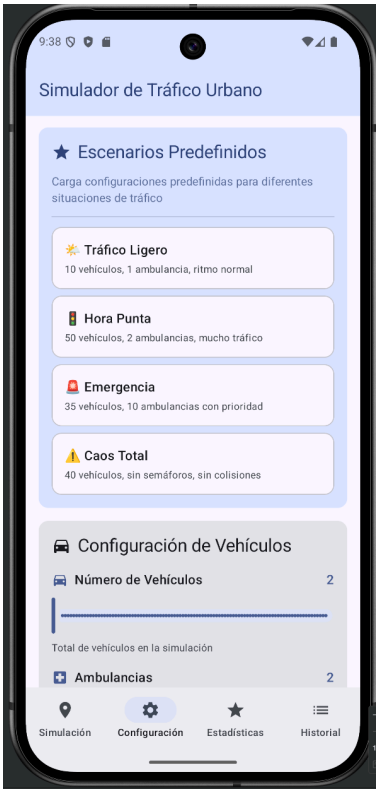
Apartado del trabajo	Integrante(s) responsable(s)
Búsqueda de una base inicial para el proyecto	Andrés Alejandro Pacheco Castillo, Luis Miguel Agra Álvarez
Desarrollo principal del código de la aplicación	Andrés Alejandro Pacheco Castillo
Implementación de funcionalidades secundarias	Luis Miguel Agra Álvarez

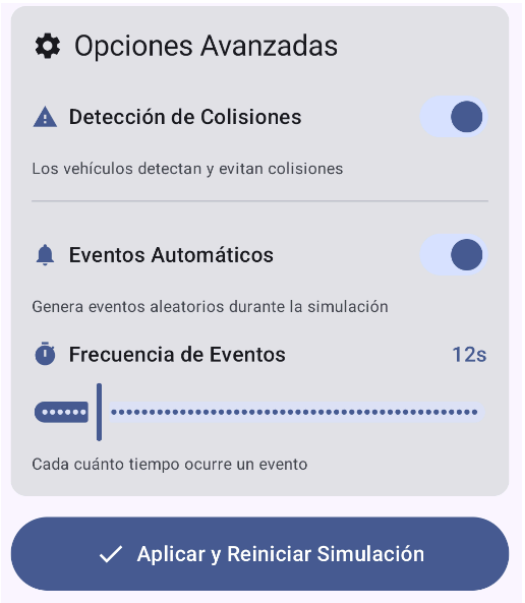
Documentación del proyecto y README	Luis Miguel Agra Álvarez
Grabación del vídeo de demostración	Luis Miguel Agra Álvarez
Revisión y pulido de UI	Luis Miguel Agra Álvarez
Revisión final del proyecto	Ambos

ANEXO: Imágenes


9:37
Simulador de Tráfico Urbano
Estadísticas en Tiempo Real
Zoom: 100%
Vehículos activos 5
En movimiento 1 (20%)
Detenidos 4
Velocidad media 16.00
Tiempo total 5s
Controles de Simulación
Iniciar
Reiniciar
Eventos de Simulación
Accidente
Obras
Simulación
Configuración
Estadísticas
Historial


9:37
Simulador de Tráfico Urbano
Historial
18/01/2026 20:51 33s
Vehículos 7
Vel. Media 7.39
Eventos 0
18/01/2026 20:26 147s
Vehículos 33
Vel. Media 215.08
Eventos 1
18/01/2026 20:21 16s
Vehículos 13
Vel. Media 14.14
Eventos 1
18/01/2026 19:28 180s
Vehículos 41
Vel. Media 146.20
Eventos 1
18/01/2026 19:21 69s
Vehículos 13
Vel. Media 6.05
Eventos 0
18/01/2026 18:56 27s
Simulación
Configuración
Estadísticas
Historial


9:38
Simulador de Tráfico Urbano
Escenarios Predefinidos
Carga configuraciones predefinidas para diferentes situaciones de tráfico
Tráfico Ligero
10 vehículos, 1 ambulancia, ritmo normal
Hora Punta
50 vehículos, 2 ambulancias, mucho tráfico
Emergencia
35 vehículos, 10 ambulancias con prioridad
Caos Total
40 vehículos, sin semáforos, sin colisiones
Configuración de Vehículos
Número de Vehículos 2
Total de vehículos en la simulación
Ambulancias 2
Simulación
Configuración
Estadísticas
Historial


Opciones Avanzadas
Detección de Colisiones
Los vehículos detectan y evitan colisiones
Eventos Automáticos
Genera eventos aleatorios durante la simulación
Frecuencia de Eventos 12s
Cada cuánto tiempo ocurre un evento
Aplicar y Reiniciar Simulación


9:38
Simulador de Tráfico Urbano
5
Vehículos
3
En Movimiento
2
Detenidos
251
Colisiones Evitadas
Velocidad Media en el Tiempo
30.50 c/s
Evolución de la velocidad promedio (celdas por segundo)
Estado de Vehículos
Simulación
Configuración
Estadísticas
Historial

ANEXO: Enlace al repositorio.

https://github.com/castillodeveloper/Simulacion-Trafico-Urbano.git