Sistemas Distribuidos 2023-2024

Práctica 2 - Art With Drones

Santiago Nogales Chiarenza 74011341D
Alejandro Molines Molina 48800938K

INTRODUCCIÓN

El objetivo principal de esta práctica es crear un sistema distribuido que simule el control de drones para formar imágenes en el espacio en tiempo real. Nos enfrentamos a la tarea de diseñar y desarrollar varios componentes esenciales, incluyendo el motor central (Engine), el registro de drones (Registry), los drones autónomos (Drone) y un servidor de clima (Weather). Cada componente desempeña un papel crucial en el funcionamiento coordinado del sistema, y nuestro desafío fue implementar una solución resiliente, escalable y segura.

A lo largo de esta memoria, detallaremos el diseño, la implementación y los desafíos que enfrentamos durante el desarrollo de "Art With Drones". Exploraremos las decisiones de diseño, los protocolos de comunicación y las soluciones técnicas que aplicamos para superar los obstáculos encontrados.

TECNOLOGÍAS UTILIZADAS

Exploramos una variedad de tecnologías que nos permitieron crear un sistema distribuido robusto y escalable. A continuación, se detallan las tecnologías clave que implementamos y utilizamos para llevar a cabo este proyecto:

- Docker y Docker Compose: Utilizamos Docker para la contenerización de nuestros componentes distribuidos. Esta tecnología nos permitió encapsular cada componente en un contenedor independiente, asegurando la portabilidad y facilitando el despliegue del sistema en múltiples entornos. Docker Compose fue esencial para la orquestación de los contenedores, permitiéndonos definir y gestionar la configuración de múltiples servicios.

- Apache Kafka y Zookeeper: Implementamos Apache Kafka para establecer un sistema de mensajería de alta capacidad y tolerante a fallas. Utilizamos Docker Compose para crear un clúster de Kafka junto con Zookeeper para la gestión de configuraciones distribuidas. Kafka fue fundamental para facilitar el flujo de eventos entre los componentes del sistema, garantizando una comunicación eficiente y confiable.

- Python: Implementamos la lógica de los drones y otros componentes utilizando Python, un lenguaje de programación versátil y fácil de entender. Python nos brindó las herramientas necesarias para gestionar la comunicación mediante sockets, procesar datos en tiempo real y realizar operaciones complejas con facilidad.

COMPONENTES

ADDrone

Esta clase representa un dron en el sistema. El dron se puede registrar, unir a la figura, moverse en el mapa, y realizar diversas operaciones en una base de datos MongoDB. A continuación, se explica brevemente cómo funciona el código:

La clase ADDrone se inicializa con las direcciones del motor (engine_address), el registro (registry_address), y el broker de Kafka (broker_address). Además, se inicializan algunas variables de estado del dron y se establece la conexión con Kafka y la base de datos.

El método input_drone_data permite al usuario ingresar un ID de dron y un alias. Luego, verifica si el ID es válido y único en la base de datos MongoDB. Si es válido y único, el dron se registra en la base de datos y se obtiene un token de acceso.

El método register_drone conecta el dron al módulo de registro (AD_Registry). Si el registro es exitoso, el dron se agrega a la lista de drones registrados.

El método authenticate verifica la autenticidad del dron comparando su ID y token con los datos almacenados en la lista de drones registrados.

El método consume_kafka_messages consume mensajes desde un tema de Kafka llamado "register_movement". Estos mensajes contienen instrucciones para el movimiento del dron en el mapa.

El método join_show permite al dron unirse al espectáculo. Mientras esté en el espectáculo, el dron puede recibir instrucciones para moverse en el mapa o detenerse. Además, puede solicitar mostrar el mapa actualizado.

Los métodos modify_drones, delete_drones, y list_drones permiten modificar, eliminar y listar los drones en la base de datos MongoDB, respectivamente.

El método show_menu muestra un menú interactivo al usuario, permitiéndole seleccionar diferentes opciones para interactuar con el dron.

En el bloque if __name__ == "__main__":, se crean instancias de la clase ADDrone con direcciones específicas para el motor, el registro y el broker de Kafka. Luego, se muestra el menú interactivo para que el usuario interactúe con el dron.

ADRegistry

Esta clase representa un módulo de registro para drones en un sistema. El módulo escucha las solicitudes de registro de los drones, verifica el formato y registra los drones en una base de datos MongoDB. Además, envía un mensaje de registro del dron al servidor Kafka para su posterior procesamiento. A continuación, se explica brevemente cómo funciona el código:
La clase "ADRegistry" se inicializa con el puerto en el que escucha las solicitudes de registro ("listen_port"), la dirección del servidor de base de datos ("db_host"), el puerto de la base de datos ("db_port"), el nombre de la base de datos ("db_name"), y la dirección del broker de Kafka ("broker_address").

El método "start" crea un socket del servidor y escucha las solicitudes de registro de los drones. Cuando recibe una solicitud, verifica el formato del JSON recibido. Si la solicitud es válida, registra el dron en la base de datos MongoDB y envía un mensaje de registro al servidor Kafka.

El método "register_drone" recibe el ID del dron, su alias y la dirección del cliente que envía la solicitud. Genera un token de acceso único para el dron, crea un diccionario con los datos del dron, lo inserta en la base de datos MongoDB y envía un mensaje de registro al servidor Kafka.

En el bloque "if __name__ == "__main__":", se configuran los parámetros del módulo de registro y se crea una instancia de la clase "ADRegistry". Luego, se inicia el servidor para que pueda escuchar las solicitudes de registro de los drones.

ADEngine

Esta clase representa el motor del sistema de drones. Esta clase escucha las solicitudes de registro de los drones y, una vez registrados, maneja sus movimientos y coordina sus acciones para seguir las instrucciones del espectáculo. A continuación, se explica brevemente cómo funciona el código:
La clase "ADEngine" se inicializa con el puerto en el que escucha las solicitudes de registro ("listen_port"), la dirección del broker de Kafka ("broker_address"), la dirección de la base de datos MongoDB ("database_address") y la dirección del servicio de clima ("weather_address").

El método "receive_dron_id" se encarga de recibir el ID del dron cuando se conecta al sistema. Utiliza un consumidor de Kafka para recibir mensajes de registro de drones. Una vez recibido el ID, se almacena y se inicializa el hilo del productor para ese dron.

El método "calculate_move_instructions" calcula las instrucciones de movimiento para un dron específico en función de su posición actual y final. Estas instrucciones se envían al dron a través de hilos del productor. La clase también contiene lógica para determinar si el dron ha alcanzado su posición final y manejar su desconexión.

La clase mantiene un mapa de las posiciones de los drones. El método "update_map_with_dron_position" actualiza el mapa con la nueva posición de un dron.

El método "procesar_datos_json" carga datos desde un archivo JSON que contiene información sobre las figuras y los drones. Estos datos se utilizan para inicializar las posiciones finales de los drones.

En el bloque "if __name__ == "__main__":", se configuran los parámetros del motor y se crea una instancia de la clase "ADEngine". Luego, se procesan los datos JSON y se inicia el motor para que pueda coordinar las acciones de los drones en el espectáculo.

ADWeather

La clase se encarga de proporcionar información sobre el clima a la aplicación de gestión de drones. A continuación, se explica cómo funciona el código:

La clase "ADWeather" se inicializa con el puerto en el que escucha las solicitudes de información sobre el clima ("listen_port"), y carga datos sobre ciudades y temperaturas desde el archivo "ciudades.json" en el método "load_city_data".

El método "get_temperature" recibe el nombre de una ciudad y devuelve la temperatura actual de esa ciudad. Utiliza los datos cargados previamente desde el archivo "ciudades.json".

En el método "start", se crea un socket del servidor y se espera recibir solicitudes de información sobre el clima. Cuando recibe una solicitud, verifica si contiene el nombre de una ciudad. Si es así, obtiene la temperatura de esa ciudad utilizando el método "get_temperature" y almacena la información del clima en un diccionario llamado "clima". Además, inicia un hilo de productor para enviar datos climáticos al broker de Kafka.

El código verifica si la temperatura es inferior a 0. Si es así, podría implementar lógica para notificar a la aplicación "AD_Engine" sobre las condiciones climáticas adversas y finalizar el espectáculo.

En el bloque "if __name__ == "__main__":", se configura el puerto en el que escucha el servidor de clima y se crea una instancia de la clase "ADWeather". Luego, se inicia el servidor para que pueda proporcionar información sobre el clima a la aplicación de gestión de drones.

GUÍA DE DESPLIEGUE

Ejecutar servicios necesarios:

MongoDB: Iniciar el servicio de MongoDB. Usualmente se puede hacer ejecutando mongod en la terminal.
Kafka (si es necesario): Iniciar el servidor de Kafka y crear los temas (topics) necesarios.

Iniciar AD_Registry:

Desde la terminal, navegar al directorio donde se encuentra el archivo AD_Registry.py.
Ejecutar el servidor AD_Registry proporcionando la información necesaria, como el puerto en el que escuchará y la dirección del servidor de base de datos MongoDB, por ejemplo:

python AD_Registry.py 8081 localhost:27017

Iniciar AD_Engine:

Desde la terminal, navegar al directorio donde se encuentra el archivo AD_Engine.py.
Ejecutar el servidor AD_Engine, proporcionando detalles como el puerto en el que escuchará y la dirección del servidor de mensajes (Kafka o similar), por ejemplo:

python AD_Engine.py 8080 10 localhost:9092 localhost:8081 localhost:27017

Iniciar AD_Drone:

Desde la terminal, navega al directorio donde se encuentra el archivo AD_Drone.py.
Ejecuta AD_Drone proporcionando la dirección del servidor AD_Engine y del servidor AD_Registry, por ejemplo:

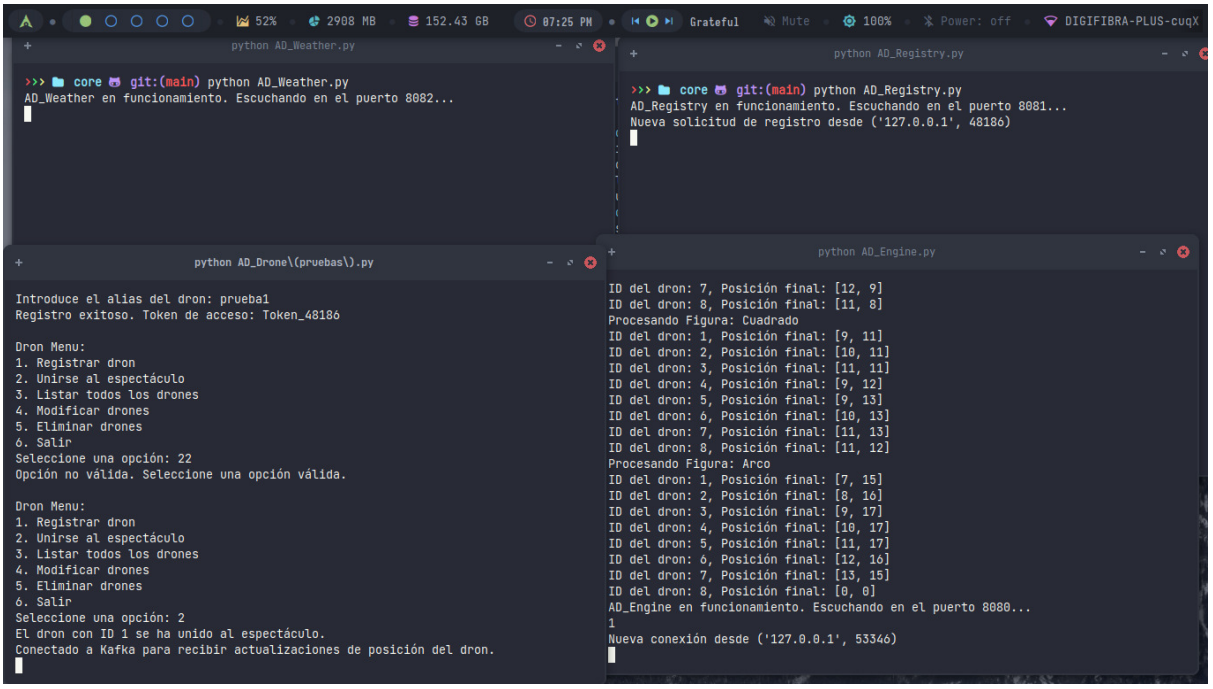
python AD_Drone.py localhost:8080 localhost:9092 localhost:8081

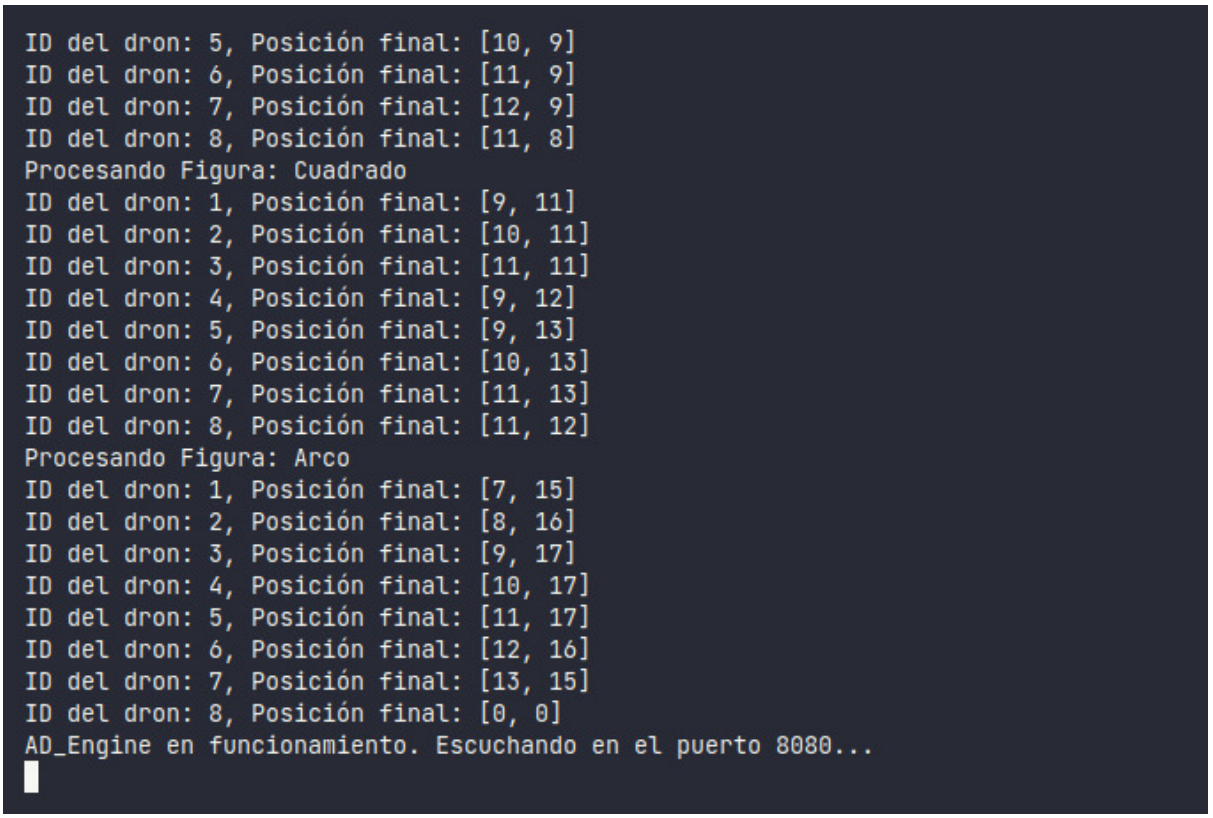
Iniciar AD_Weather

Desde la terminal, navega al directorio donde se encuentra el archivo AD_Weather.py.
Ejecuta el servidor ADWeather, proporcionando el puerto en el que escuchará. Por ejemplo:

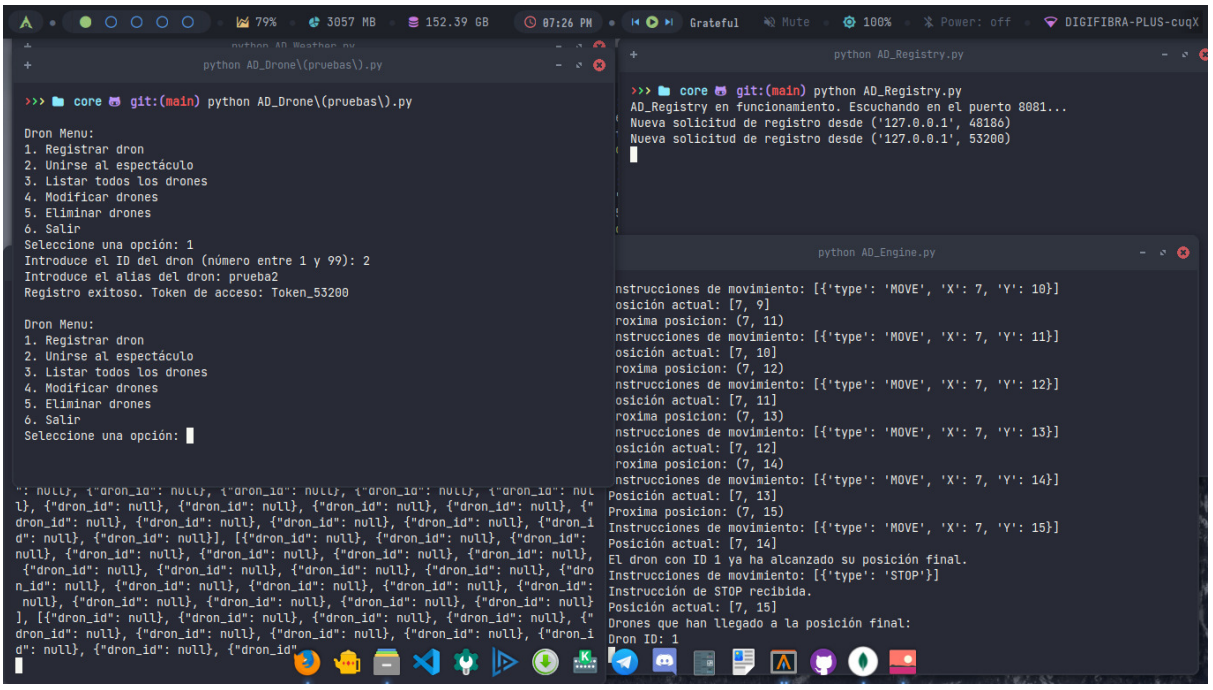
python AD_Weather.py 8082

CAPTURAS


python AD_Weather.py
>>> core git:(main) python AD_Weather.py
AD_Weather en funcionamiento. Escuchando en el puerto 8082...
python AD_Registry.py
>>> core git:(main) python AD_Registry.py
AD_Registry en funcionamiento. Escuchando en el puerto 8081...
Nueva solicitud de registro desde ('127.0.0.1', 48186)
python AD_Drone\(pruebas\).py
Introduce el alias del dron: prueba1
Registro exitoso. Token de acceso: Token_48186
Dron Menu:
1. Registrar dron
2. Unirse al espectáculo
3. Listar todos los drones
4. Modificar drones
5. Eliminar drones
6. Salir
Seleccione una opción: 22
Opción no válida. Seleccione una opción válida.
Dron Menu:
1. Registrar dron
2. Unirse al espectáculo
3. Listar todos los drones
4. Modificar drones
5. Eliminar drones
6. Salir
Seleccione una opción: 2
El dron con ID 1 se ha unido al espectáculo.
Conectado a Kafka para recibir actualizaciones de posición del dron.
python AD_Engine.py
ID del dron: 7, Posición final: [12, 9]
ID del dron: 8, Posición final: [11, 8]
Procesando Figura: Cuadrado
ID del dron: 1, Posición final: [9, 11]
ID del dron: 2, Posición final: [10, 11]
ID del dron: 3, Posición final: [11, 11]
ID del dron: 4, Posición final: [9, 12]
ID del dron: 5, Posición final: [9, 13]
ID del dron: 6, Posición final: [10, 13]
ID del dron: 7, Posición final: [11, 13]
ID del dron: 8, Posición final: [11, 12]
Procesando Figura: Arco
ID del dron: 1, Posición final: [7, 15]
ID del dron: 2, Posición final: [8, 16]
ID del dron: 3, Posición final: [9, 17]
ID del dron: 4, Posición final: [10, 17]
ID del dron: 5, Posición final: [11, 17]
ID del dron: 6, Posición final: [12, 16]
ID del dron: 7, Posición final: [13, 15]
ID del dron: 8, Posición final: [0, 0]
AD_Engine en funcionamiento. Escuchando en el puerto 8080...
1
Nueva conexión desde ('127.0.0.1', 53346)


ID del dron: 5, Posición final: [10, 9]
ID del dron: 6, Posición final: [11, 9]
ID del dron: 7, Posición final: [12, 9]
ID del dron: 8, Posición final: [11, 8]
Procesando Figura: Cuadrado
ID del dron: 1, Posición final: [9, 11]
ID del dron: 2, Posición final: [10, 11]
ID del dron: 3, Posición final: [11, 11]
ID del dron: 4, Posición final: [9, 12]
ID del dron: 5, Posición final: [9, 13]
ID del dron: 6, Posición final: [10, 13]
ID del dron: 7, Posición final: [11, 13]
ID del dron: 8, Posición final: [11, 12]
Procesando Figura: Arco
ID del dron: 1, Posición final: [7, 15]
ID del dron: 2, Posición final: [8, 16]
ID del dron: 3, Posición final: [9, 17]
ID del dron: 4, Posición final: [10, 17]
ID del dron: 5, Posición final: [11, 17]
ID del dron: 6, Posición final: [12, 16]
ID del dron: 7, Posición final: [13, 15]
ID del dron: 8, Posición final: [0, 0]
AD_Engine en funcionamiento. Escuchando en el puerto 8080...


docker-compose up
compose-mongo-1 | {"t":{"$date":"2023-11-05T18:22:47.378+00:00"},"s":"I", "c":"REPL", "id":7360102, "ctx":"LogicalSessionCacheRefresh","msg":"Added oplog entry for create to transaction","attr":{"namespace":"config.$cmd","uuid":{"uuid":{"$uuid":"3a4d359d-33df-4144-a11d-637f01ad9fe1"}},"object":{"create":"system.sessions","idIndex":{"v":2,"key":{"_id":1},"name":"_id_"}}}}
compose-mongo-1 | {"t":{"$date":"2023-11-05T18:22:47.378+00:00"},"s":"I", "c":"REPL", "id":7360100, "ctx":"LogicalSessionCacheRefresh","msg":"Added oplog entry for createIndexes to transaction","attr":{"namespace":"config.$cmd","uuid":{"uuid":{"$uuid":"3a4d359d-33df-4144-a11d-637f01ad9fe1"}},"object":{"createIndexes":"system.sessions","v":2,"key":{"lastUse":1},"name":"lsidTTLIndex","expireAfterSeconds":1800}}}
compose-mongo-1 | {"t":{"$date":"2023-11-05T18:22:47.401+00:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"LogicalSessionCacheRefresh","msg":"Index build: done building","attr":{"buildUUID":null,"collectionUUID":{"uuid":{"$uuid":"3a4d359d-33df-4144-a11d-637f01ad9fe1"}},"namespace":"config.system.sessions","index":"_id_","ident":"index-5-3281225853819425661","collectionIdent":"collection-4-3281225853819425661","commitTimestamp":null}}
compose-mongo-1 | {"t":{"$date":"2023-11-05T18:22:47.401+00:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"LogicalSessionCacheRefresh","msg":"Index build: done building","attr":{"buildUUID":null,"collectionUUID":{"uuid":{"$uuid":"3a4d359d-33df-4144-a11d-637f01ad9fe1"}},"namespace":"config.system.sessions","index":"lsidTTLIndex","ident":"index-6-3281225853819425661","collectionIdent":"collection-4-3281225853819425661","commitTimestamp":null}}


python AD_Drone\(pruebas\).py
>>> core git:(main) python AD_Drone\(pruebas\).py
Dron Menu:
1. Registrar dron
2. Unirse al espectáculo
3. Listar todos los drones
4. Modificar drones
5. Eliminar drones
6. Salir
Seleccione una opción: 1
Introduce el ID del dron (número entre 1 y 99): 2
Introduce el alias del dron: prueba2
Registro exitoso. Token de acceso: Token_53200
Dron Menu:
1. Registrar dron
2. Unirse al espectáculo
3. Listar todos los drones
4. Modificar drones
5. Eliminar drones
6. Salir
Seleccione una opción:
python AD_Registry.py
>>> core git:(main) python AD_Registry.py
AD_Registry en funcionamiento. Escuchando en el puerto 8081...
Nueva solicitud de registro desde ('127.0.0.1', 48186)
Nueva solicitud de registro desde ('127.0.0.1', 53200)
python AD_Engine.py
Posición actual: [7, 13]
Proxima posicion: (7, 15)
Instrucciones de movimiento: [{'type': 'MOVE', 'X': 7, 'Y': 15}]
Posición actual: [7, 14]
El dron con ID 1 ya ha alcanzado su posición final.
Instrucciones de movimiento: [{'type': 'STOP'}]
Instrucción de STOP recibida.
Posición actual: [7, 15]
Drones que han llegado a la posición final:
Dron ID: 1