

Práctica de Sistemas Distribuidos

Art With Drones Hito 2



Integrantes:

- Alejandro Guillén Merino **DNI:** 48790456G
- Víctor Padrón García **DNI:** 55178804X

Universidad de Alicante
Escuela Politécnica Superior
Curso 2023-24

1. Índice

1. Índice.....	2
2. Componentes software:.....	3
2.1. AD_Drone.....	3
- Class AD_Drone:.....	3
2.2. AD_Registry.....	4
- Class AD_Registry:.....	4
2.3. AD_Engine.....	5
- Class AD_Engine:.....	5
- Class RecibirDrones.....	6
- Class EscucharDrone.....	7
2.4. BD.....	7
- drones:.....	7
- mapa:.....	7
2.5. API_Engine.....	7
- /drones:.....	8
- /mapa:.....	8
2.6. Front.....	8
- index.html:.....	8
- drones.html:.....	8
- mapa.html:.....	8
3. Despliegue de la aplicación:.....	8
4. Funcionamiento de las aplicaciones:.....	9

2. Componentes software:

2.1. AD_Drone

AD_Drone es el encargado de crear los drones que se van a utilizar y en él se implementa la clase de los drones y los métodos para realizar transferencias de datos que se van a detallar a continuación:

- Class AD_Drone:

La clase drone tiene los siguientes atributos:

- + **alias:** Nombre del drone.
- + **id:** Id del drone. Viene a ser como el número de serie por el que se identifica a cada drone.
- + **id_virtual:** Id del drone dentro del sistema. 1, 2, 3, 4,....
- + **token:** Almacena el token de acceso después de haberse registrado, para autenticar el drone con AD_Engine
- + **broker:** Almacena los datos de conexión con kafka.
- + **posicionFin:** Indica la posición donde debe acabar el drone.
- + **posicionActual:** Indica la posición en la que se encuentra el drone.
- + **estado:** Indica si el drone ha llegado a la posición final (Verde) o no (Rojo).
- + **detener:** Si el espectáculo se detiene se pone a True y el drone se va a la base.
- + **auto:** Permite el registro y la inscripción en el espectáculo de manera automática.

Por otro lado, existen diferentes funciones para su funcionamiento. Antes de acceder a ellas, se comprueban los parámetros que deben contener las ip y puerto de engine, registry y de kafka, así como el alias del drone.

Una vez iniciado, se muestra un menú para elegir entre registrarse (conectar con Registry) y entrar al espectáculo (conectar con Engine).

- + **registrarse(ip, puerto):** Solicita registrarse en el sistema por medio de la API implementada en el registry. Manda el id y el alias y recibe si se ha registrado y, en caso afirmativo, recibe un token para identificarse
- + **solicitar_inclusión(ip, puerto):** Recibe la ip y el puerto de escucha del AD_Engine. Establece conexión mediante socket seguro y envía su id y el token. En caso de poderse unir, recibe de vuelta el id_virtual, que representa su posición dentro del espectáculo.
- + **run():** En este método se crean el consumidor del destino y el productor de la posición y de su actividad. A continuación, se crean Threads para paralelizar los procesos y que puedan funcionar simultáneamente de manera concurrente. Por último, se muestra un menú con opciones para mostrar o no el mapa y salir del espectáculo.
- + **escucharPorKafkaDestino(consumidor):** Se utiliza un topic llamado "destino" y, en caso de recibir un mensaje, se lee la posición final y se almacena. En tal caso, el drone comenzará a moverse hacia esa posición.

- + **enviarPosicion(productor)**: Se utiliza un topic “posiciones” y envía al Engine por kafka el id y la posición del drone.
- + **mover(productor)**: Comprueba el estado del drone y si no ha llegado a la última posición se mueve a una adyacente más cercana a su posición final. Por último, se invoca a “enviarPosicion()” para transmitir la posición actualizada al Engine.
- + **requireApiKey(viewFunction)**: Es un decorador que al ponerlo sobre una función que utiliza api fuerza que en el header se ponga la APIKey. Esta se encuentra en un archivo llamado API_KEY_REGISTRY.txt.
- + **borrarBD()**: Limpia la base de datos cada vez que se realiza nuevamente el espectáculo.
- + **controlarToken(id)**: Se elimina el token de los drones después de 20 segundos de proporcionarlo.

De esta forma funciona un drone de manera interna, transmitiendo y recibiendo datos con AD_Registry y AD_Engine. Todo este código se encuentra en el archivo AD_Drone.py.

2.2. AD_Registry

AD_Registry es el encargado de registrar drones en la BD, proporcionar tokens de verificación a estos y controlar su borrado. Funciona de manera independiente a Engine.

- Class AD_Registry:

La clase AD_Registry tiene los siguientes atributos:

- + **id_nueva**: Id que representará al drone en la bd.
- + **token_dron_actual**: Variable con el token para escribirlo en el socket.

Además, se han implementado diferentes métodos para controlar todo lo que ocurre, así como para dirigir el funcionamiento y el envío de los drones.

Primero, se va a explicar el funcionamiento de la API implementada, encargada de modificar los datos de la BD de manera interna.

- + **app.route('/obtenerdatos')**: Permite la obtención de todos los datos de la tabla drones de la BD. (GET)
- + **app.route('/unirme')**: Permite registrar un drone. Se llama desde AD_Drone para registrarse. (POST)

A continuación, los métodos implementados en AD_Registry:

- + **generar_token()**: Crea un token temporal para el drone que se registra, que se almacena en la BD de manera temporal.
- + **controlar_token(token)**: Comprueba si ha pasado el tiempo de expiración del token. Una vez han pasado los 20 segundos, se elimina de la BD.
- + **borrar_bd(id, alias)**: Borra todos los datos de la tabla drones de la base de datos.

De esta manera conseguimos que el registry permita que los drones se registren en la BD por medio de una API.

2.3. AD_Engine

En AD_Engine se almacenan las clases y métodos necesarios para que todo funcione y se coordine de manera correcta. Es el encargado de comunicarse con los drones, obtener la figura e informarse del clima, así como de verificar los drones.

- Class AD_Engine:

La clase AD_Engine tiene los siguientes atributos:

- + **figuras:** Vector con las figuras leídas del archivo "Figuras.json".
- + **dronesFinales:** Almacena las posiciones finales de todos los drones para cada figura.
- + **detener:** En caso de fallo, se pone a true y se detiene el espectáculo.
- + **detener_por_clima:** Si el motivo de detener es por el clima, se pone a True.
- + **en_base_por_clima:** Si los drones se han retirado por el clima, se pone a True, y se finaliza el espectáculo.
- + **logger:** Almacena los logs generados por el programa.
- + **last_position_received:** Es un diccionario que se encarga de almacenar para cada id_virtual el ultimo momento en el que recibió una posición de dicho drone.

A parte de estos atributos son necesarios algunos métodos para que funcione todo correctamente.

Iniciando desde el main, se van a explicar todos los métodos en orden de invocación según el flujo del programa. En el main se comprueban los parámetros por línea de comando y se crea un proceso que crea un objeto de la clase EscucharDrones para escuchar los drones y autenticarlos drones. Por otro lado, se espera hasta que se detecta el archivo Figuras.json. A partir de ahí, se crea una instancia de la clase AD_Engine, el productor de destinos que enviará las posiciones finales de los drones, el productor de mapa, que enviará el mapa, y el consumidor de posiciones, que recibirá las posiciones de los drones. Por último, se inician todos estos procesos.

- + **start(productor_destinos, productor_mapa, consumidor):** Inicia el espectáculo. Crea un thread para controlar el clima y lee las figuras del archivo Figuras.json y crea un thread para enviar las posiciones a los drones. Comprueba constantemente si la figura se ha terminado o no para volver a enviar repetidamente el mapa.
- + **stop():** Detiene la acción que realiza el engine y para el espectáculo.
- + **escribe_socket(p_datos):** Manda datos de verificación al socket.
- + **lee_socket(p_datos):** Recibe los datos de verificación del drone.
- + **figura_completada():** Comprueba si todos los drones han alcanzado sus posiciones finales.
- + **leer_figuras():** Lee las figuras que existen en el archivo Figuras.json y lo elimina, a la espera de que llegue uno nuevo con nuevas figuras para el espectáculo.
- + **escuchar_posición_drones(consumidor):** Recibe por kafka las posiciones de los drones y las actualiza en el vector correspondiente. Si no existe, crea

un nuevo drone en el vector (no un AD_Drone) con ese identificador. Además, actualiza la posición actual de cada drone en la BD.

- + **enviar_por_kafka_destinos(productor)**: Envía a través del topic “destinos” el destino de cada drone para finalizar la figura.
- + **consumidor_posiciones(broker)**: Crea el consumidor de las posiciones de los drones.
- + **productor_destinos(broker)**: Crea el productor de los destinos que se enviarán a los drones mediante kafka.
- + **clear_terminal()**: Método para borrar la terminal y conseguir sobrescribir el mapa cada vez que se actualiza.
- + **clima(engine,apiKey)**: Lee del archivo “ciudades.txt” la ciudad que hay y realiza consultas periódicas a la API de OpenWeather para obtener la temperatura en °C. Si es menor que 0 °C, se finaliza el espectáculo.
- + **todosRojo(self)**: Al finalizar una figura, se establece el estado de todos los drones a rojo para comenzar a moverse al nuevo destino.
- + **comprobarFinBD(self)**: Comprueba si la posición del drone es la posición final, y que en la BD esté la posición guardada correctamente.
- + **pintarVerde(self, id_virtual)**: En caso de que esté en la posición final un drone, se establece a Verde el id del drone en la BD
- + **actualizarPosicionesBD(self, drone)**: Se encarga de actualizar la posición de los drones en la BD cuando se reciben.
- + **actualizar_momento(self)**: Actualiza el valor de la variable `las_position_receibed`.
- + **check_drones(self)**: Comprueba que drones han estado sin enviar valores durante 10 segundos o más. Si no está activo actualiza el valor de la columna activos a 0 y si está activo lo actualiza a 1.
- + **leerApiKeyOpenWeather(archivoApiKey)**: Lee el valor de la apiKey del archivo proporcionado.

Además, como se ha comentado anteriormente, es necesaria una clase que se llama EscucharDrones que se explicará más adelante.

- Class RecibirDrones

Esta clase se encarga de la comunicación con los drones mediante el socket; mantiene abierta la escucha para que si hay nuevos drones se puedan autenticar a pesar de que el espectáculo haya comenzado. Tiene los siguientes atributos:

- + **puerto**: Puerto del AD_Engine dónde se va a escuchar para recibir nuevos drones.

Además, tiene el siguientes métodos:

- + **run()**: Obtiene la ip de la red actual y comienza el funcionamiento de todo. Abre el socket y se mantiene a la escucha de que algún drone se conecte. Si se conecta, se crea una instancia de la clase EscucharDrone para recibir los datos y se mantiene a la espera de recibir un nuevo drone.

El código de estas dos clases se puede encontrar en el archivo AD_Engine.py.

- Class EscucharDrone

Esta clase sirve para recibir los datos de los drones que desean incorporarse al espectáculo y verificar su id y token. Tiene los siguientes atributos:

- + **drone:** Información del drone que se ha conectado.

Además se tienen los siguientes métodos para el funcionamiento.

Los métodos son los siguientes:

- + **run():** Recibe el drone por socket y lee la información de este con `lee_socket()`. La separa en diferentes posiciones de un vector separándolos por los espacios. A continuación, se llama al método `autenticar()` para verificar la información que ha llegado. Por último se escribe el socket con la información según si ha sido aceptado o denegado.
- + **lee_socket(p_datos):** Recibe el drone, decodifica la información y la devuelve en un string.
- + **autenticar(token, id):** Recibe el token y el id de un drone y consulta en la BD si corresponden. Si el token no ha caducado y es igual al almacenado, se permite que el drone entre al espectáculo y se devuelve el `id_virtual` al drone.
- + **escribe_socket(p_datos):** En función de lo recibido de `autenticar` devuelve "aceptado" con el id con el orden de inclusión en el espectáculo o "denegado" para indicar que no se ha podido verificar el drone.

El código de esta clase se puede encontrar en el archivo `EscucharDrone.py`.

2.4. BD

Se ha creado una BD con `sqlite` para almacenar los drones, el mapa, y el estado de cada componente del sistema. Para ello se han utilizado 3 tablas dentro de la BD llamada `registry`:

- drones:

La tabla `drones` almacena los drones registrados, identificado cada uno de ellos por un id, que además tienen el `id_virtual`, `alias`, `token` (de manera temporal), posición actual y fin (ok o no, según si la posición actual es la misma que la final).

- mapa:

En principio sirve para guardar cualquier información relacionada con el mapa, aunque por simplificar se ha utilizado finalmente la tabla `drones` ya que incluye la posición de cada drone en tiempo real.

2.5. API_Engine

La API se ha escrito en JavaScript. Dado que es una API únicamente de consulta y no se pueden modificar datos de la BD a través de ella, se han limitado sus funciones a operaciones GET. Tiene los siguientes `endPoints`:

- /drones:
Recibe de la BD todos los datos de la tabla drones.
- /mapa:
Recibe de la BD todos los datos de la tabla mapa.

2.6. Front

El front se ha implementado en html con css y JavaScript. Cuenta con cuatro páginas que se explican a continuación:

- index.html:
Página principal, dónde se muestra el nombre de la práctica y tres botones, uno para drones.html, otro para mapa.html y otro para estados.html.
- drones.html:
Realiza una consulta cada segundo a la API_Engine para recibir los drones que hay y mostrarlos en una tabla.
- mapa.html:
Realiza una consulta a la API_Engine cada segundo para conocer el id_virtual, posición y estado de cada dron. Crea un tablero con una tabla y muestra cada id_virtual en su posición, con el color que le corresponde según dónde esté. También comprueba si alguno de ellos está desactivado y no lo muestra en dicho caso.

3. Despliegue de la aplicación:

Para desplegar de manera correcta la aplicación se deben ejecutar varios comandos de una manera específica, ya que si se inicia un servicio en otro orden pueden haber fallos por que no se pueden conectar los drones con registry o engine. Se van a tratar 3 ordenadores, aunque podrían ser más. El "PC-1" alojará el front, el gestor de colas y drones, en el "PC-2" se iniciarán la API_Engine, AD_Engine y AD_Registry y en el "PC-3" hasta el "PC-n" se alojarán mas drones.

1. **Iniciar Zookeeper:** Lo primero es iniciar zookeeper en el PC-1. Para ello se debe abrir una terminal en la carpeta /kafka/bin y ejecutar el siguiente comando: "\$./zookeeper-server-start.sh ../config/zookeeper.properties".
2. **Iniciar Kafka-Server:** A continuación, también en el PC-1, se debe arrancar kafka-server. Para ello, desde la carpeta/kafka/bin se debe ejecutar el siguiente comando: "\$./kafka-server-start.sh ../config/server.properties"
3. **Crear los topics:** Para terminar de iniciar todo lo relacionado con kafka se deben crear los topics para el envío de mensajes. De nuevo en el PC-1 se debe ejecutar el

siguiente comando desde la carpeta/kafka/bin: "\$./kafka-topics.sh --create --bootstrap-server (ip_ordenador):9092 --topic (topic) --partitions 100 --replication-factor 1". Se debe hacer esto para cada topic que necesitemos, en nuestro caso los topics "destino" y "posiciones".

4. **Iniciar el AD_Registry:** En el PC-2 se debe abrir una nueva terminal e iniciar el AD_Registry con el siguiente comando: "\$ python3 AD_Registry.py 8081". El 8081 es el puerto que escuchará para recibir peticiones de registro de los drones. Al ejecutar el comando se queda a la espera de recibir un drone para registrarse y se muestran unos logs de la API.
5. **Iniciar al AD_Engine:** En el mismo dispositivo que el anterior, PC-2, se inicia AD_Engine, que será el motor principal del espectáculo y con quien se comunicarán los drones a través de kafka. Se inicia con el siguiente comando: "\$ python3 AD_Engine.py 9090 (max_drones) (ip_brpoker):9092 API_KEY". Los parámetros son los siguientes: "max_drones" es el número máximo de drones permitidos conectados simultáneamente, "ip_broker" es la dirección ip de kafka. Al ejecutar el comando AD_Engine entra en espera hasta leer alguna figura y tener drones para el espectáculo.
6. **Iniciar API_Engine y servidor http:** En el PC-1, se accede al directorio /front y se ejecuta el siguiente comando: "\$ python3 -m http.server". Esto inicia un servidor http con origen en un archivo llamado index.html. A continuación, se inicia la API_Engine con el siguiente comando en el directorio /Nucleo/REST_SD en el PC-2: "\$ npm start".
7. **Crear los AD_Drones:** Por último, desde el PC-1 o PC-3, queda crear los drones. Cada drone se crea con el siguiente comando: "\$ ". Los parámetros son los siguientes: "nombre" es el alias del drone, "ip_engine" es la dirección ip del dispositivo dónde se encuentra AD_Engine, "broker_kafka" es la dirección ip del dispositivo dónde se encuentra kafka y el parámetro "ip_registry" es la dirección ip del dispositivo dónde se encuentra el AD_Registry. Al ejecutar el comando se muestra un menú con las opciones "Registrarse", "Entrar al Espectáculo" y "Salir". Se debe registrar primero y después entrar al espectáculo.

4. Funcionamiento de las aplicaciones:

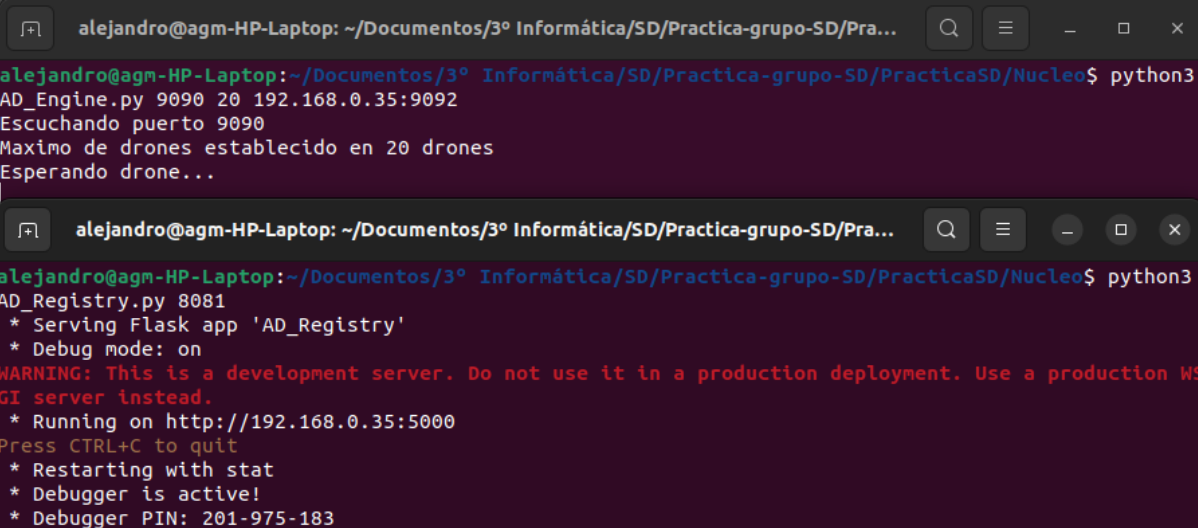
A continuación se va a explicar de manera más detallada cómo funcionan las aplicaciones en el conjunto del sistema. Partiremos desde el punto en el que se ha arrancado cada una de las aplicaciones y se va a explicar el flujo del sistema pasando por cada una de las aplicaciones.

1. **Registro de un Drone:** En el menú que se muestra en la terminal del drone se debe seleccionar la primera opción para registrar un drone. Éste establece conexión mediante API con AD_Registry enviándole su alias y su id. Registry los almacena en la BD de drones (registry) y le devuelve al drone un token, que también almacena en la BD de manera temporal, para autenticarse más adelante. El AD_Registry ya no tiene más interacción en el sistema.
2. **Verificación del Drone:** Ahora en el menú del drone se selecciona la segunda opción para autenticar el drone en el AD_Engine y tomarlo en cuenta para el espectáculo. El drone establece conexión con el engine y le envía su token y su id, que será verificado con el que hay almacenado en la BD. El AD_Engine le devuelve un id para referirse a él cuando lo muestre en el mapa (1, 2, 3, ...).

3. **Conseguir una Figura:** Simultáneamente a lo que ocurre en el punto 2, AD_Engine se mantiene a la espera de que se inserte un fichero llamado "Figuras.json". Cuando se recibe, se lee en orden, almacenando cada figura (nombre) con sus posiciones en un vector dentro del engine para poder compartirlo con los drones más adelante.
4. **Consultar el Clima:** Una vez hay drones y se tiene una figura, se empieza a comprobar el clima para saber si el espectáculo puede continuar. AD_Engine consulta cada segundo a la API de OpenWeather el clima de la ciudad que se encuentre en el archivo "ciudades.txt". Si la temperatura cae por debajo de los 0 °C, se manda una señal para finalizar el espectáculo.
5. **Comunicar Posiciones:** Con la temperatura por encima de 0 °C se da comienzo al espectáculo. Engine envía por kafka a todos los drones que se han mantenido a la espera después de autenticarse las coordenadas a las que deben moverse como posición final para completar la primera figura. Éstos empiezan a moverse, respondiendo con la posición a la que se mueven constantemente al engine, que almacena estas posiciones en un vector según el id del drone que la envía y también en la BD.
6. **Imprimir Mapa:** Simultáneamente a todo lo anterior, API_Engine y el servidor http se mantienen en funcionamiento, mostrando cualquier cambio (nuevos drones y mapa) que ocurre en la BD.
7. **Final de Figura:** Una vez se finaliza una figura se espera a recibir una nueva. En caso de que no se reciba ninguna en un espacio de tiempo determinado, se termina el espectáculo.

A continuación, se va a mostrar unas capturas de lo que se debe ver en el front durante el espectáculo

- Ejemplo de ejecución normal:



```
alejandro@agm-HP-Laptop: ~/Documentos/3º Informática/SD/Practica-grupo-SD/Pra...  
alejandro@agm-HP-Laptop:~/Documentos/3º Informática/SD/Practica-grupo-SD/PracticaSD/Nucleo$ python3  
AD_Engine.py 9090 20 192.168.0.35:9092  
Escuchando puerto 9090  
Maximo de drones establecido en 20 drones  
Esperando drone...  
  
alejandro@agm-HP-Laptop: ~/Documentos/3º Informática/SD/Practica-grupo-SD/Pra...  
alejandro@agm-HP-Laptop:~/Documentos/3º Informática/SD/Practica-grupo-SD/PracticaSD/Nucleo$ python3  
AD_Registry.py 8081  
* Serving Flask app 'AD_Registry'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://192.168.0.35:5000  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 201-975-183
```

Imagen 1:Núcleo en ejecución esperando conexiones.

```
alejandro@agm-HP-Laptop: ~/Documentos/3º Informática/SD/Practica-grupo-SD/Pra...
alejandro@agm-HP-Laptop:~/Documentos/3º Informática/SD/Practica-grupo-SD/PracticaSD/Drone$ python3 A
D_Drone.py drone 192.168.0.35 9090 192.168.0.35 9092 192.168.0.35 8081
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
1
{
  "data": [
    {
      "alias": "drone",
      "id": 3705,
      "id_virtual": 1,
      "token": "6579ea6c-ffc9-4d61-a461-3e3bcf985c9b"
    }
  ],
  "error": false,
  "message": "Item Added Successfully"
}
[1] Registrar drone en el sistema
[2] Entrar al espectáculo
[3] Salir
2
---Drone 1 unido de manera satisfactoria---
```

Imagen 2: Drone unido de manera satisfactoria

[MEGA 3º](#) [YouTube](#) [Cómo utilizar SQL Ser...](#)

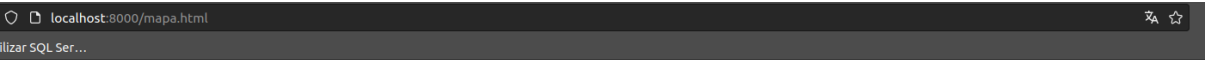
Volver al menú

Drones

Lista de Drones

ID	ID Virtual	Alias	Token	Posicion	Estado
3705	1	drone	null	[0, 0]	no
5947	3	drone	1c87a1e9-d0e5-4cfc-8d38-ff2b8c3f4d58	[0, 0]	no
7404	2	drone	b8cc5081-c3f9-404e-866b-687e91a21f5e	[0, 0]	no

Imagen3: Drones unidos en el font



Mapa

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																				
2																				
3																				
4						1	2	3												
5						4		5												
6						6	7	8												
7						9	12	10												
8						11														
9																				
10																				
11																				
12																				
13																				
14																				
15																				
16																				
17																				
18																				
19																				
20																				

Imagen 4: Drones moviéndose al destino



Mapa

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9										3		2	4	6						
10									1	5		7		9						
11										8		11		12						
12										10		13	14	15						
13																				
14																				
15																				
16																				
17																				
18																				
19																				
20																				

Imágen 5: Drones con figura completada