Se abre el archivo, se recorre cada carácter y en una matriz de bits, que empieza con todo en 0 y tiene el mismo tamaño que la imagen satelital, se va poniendo donde hay maleza o barrera según lo que dice el txt. Si hay maleza (1) se pone un 01 y si hay barrera (x o X) se pone 10.

Ahora toca identificar parches de maleza y barreras, definiendo su posición y tamaño con respecto al .txt, esto se hace con filtrar(). Por ahora, se agrega al inicio como una mancha para que funcione como punto de partida del dron y después se recorre toda la matriz de bits verificando donde hay maleza o barrera.

* Cuando encuentra una maleza, verifica si se forma una mancha de 3x3 (la maleza encontrada estaría en la punta superior izquierda del parche). Si se encuentra, toma al centro de la mancha como su posición en el txt y borra la maleza (en la matriz de bits) en la punta superior derecha y en la inferior izquierda para que, si se encuentra alguna de las otras malezas que forman la mancha, no se tome de nuevo a la mancha que se acaba de registrar. Cada mancha se guarda en una cola (ahora esta como vector)
* Cuando encuentra una barrera, guarda la coordenada donde esta esa barrera inicial y se va fijando hasta donde llega esa barrera. Independientemente del largo de la barrera, la termina guardando en una cola (que por ahora es un vector). Cada vez que encuentra otro carácter barrera, lo borra de la matriz de bits para no tenerlo en cuenta después

Una vez que se identificaron las manchas y barreras y se sabe dónde están y el tamaño que tienen, queda ver entre que manchas el dron se puede mover en línea recta (es decir, que no haya barrera que intercepte el recorrido del dron entre las dos manchas). Para esto, primero se construyen 2 matrices: una de adyacencia (que indica si el dron puede ir de una mancha a otra) y otra del peso (que sería la distancia que el dron hace para ir de una mancha a otra, que si no puede ir se setea a la distancia como INFI). Así, se recorre todo el array de manchas de a pares, ósea, si tengo 6 manchas, por ejemplo, me fijo si la 0 puede ir a la 1, la 2, la 3, la 4 y la 5, después si la 1 puede ir a la 2, la 3, la 4 y la 5, y así. En cada iteración, guardo la posición de la mancha x y la mancha y, copio el vectorBarreras y me fijo barrera por barrera si hay una intersección entre el camino entre las 2 manchas y la barrera. Si no hay intersección, se configura la distancia entre las dos manchas y se pone un 1 en la matriz de adyacencia de los vértices correspondientes (los vértices son las manchas), si hay intersección, se setea la distancia como INFI, se pone un 0 en la matriz de adyacencia y se deja de comparar con el resto de las barreras si es que quedaba alguna. De esta forma, **se arma la matriz de adyacencia**.

El siguiente paso es identificar todos los caminos posibles que recorran a todos los vértices una sola vez y haciendo una línea recta (es decir, se identifican ciclos hamiltoneanos) y, además, identificar cual de ellos es el más corto.

Se crea un puntero a un objeto Hamilton(), que se encarga de encontrar a los ciclos hamiltoneanos del grafo. Primero se carga al vértice 0 al camino y se lo marca como visitado. Luego se ejecuta un método recursivo hasta encontrar todos los posibles ciclos hamiltoneanos y que empieza por la posición 1 del camino, que sería la siguiente a la inicial. De esta forma, el método va a ir avanzando en posiciones hasta llegar a la cantidad de vértices que tiene el txt (porque el camino tiene que ir por todas las manchas y por el inicio 2 veces). En el medio, solamente utiliza vértices que todavía no forman parte del camino y que sean adyacentes entre si (ósea, que son “seguros”). Una vez que se recorrieron todos los vértices al armar el camino, comprueba que del último vértice que se agregó se pueda volver al inicio, vuelve a agregar al inicio al camino y agrega el camino a la matriz de caminos. **Al final, se identifican todos los posibles ciclos hamiltoneanos**

Una vez identificados todos los posibles caminos, se copian en una matriz y empiezo a buscar el camino mas corto. Voy fila por fila de la matriz (camino por camino) para luego ver la distancia entre cada vertice que se recorre en el camino e ir guardando el camino por si resulta ser el mas corto. Luego, compara si el camino actual es mas corto que el camino más corto guardado, y si lo es lo guarda como el mas corto. Finalmente, una vez que recorre toda la matriz de caminos, **se obtiene e imprime al camino mas corto**