Universidad de Santiago de Compostela

Trabajo final de Diseño de Software.

Curso 2023/2024

MARS MADNESS

Sara Castro López

sara.castro.lopez@rai.usc.es

Diego Cristóbal Andaluz

diego.cristobal@rai.usc.es

Jorge González Corbelle

jorge.gonzalez.corbelle@rai.usc.es

Jose Luis Estrada García

joseluis.estrada@rai.usc.es

Iago Feijóo Rey

iago.feijoo.rey@rai.usc.es

Índice general

Introducción

En este proyecto, se aplicaron los conocimientos adquiridos a lo largo del curso de Diseño de Software para desarrollar un videojuego, llamado Mars Madness. Se implementaron modelos de casos de uso, estructurales y de comportamiento, siguiendo las indicaciones del profesor. Se empleó activamente starUML para la creación de estos modelos, herramienta para el modelamiento de software basado en los estándares UML.

El equipo de diseño está conformado por 5 integrantes: Sara Castro López, Diego Cristóbal Andaluz, Iago Feijóo Rey, Jorge González Corbelle y Jose Luis Estrada García. Se ha trabajado en el proyecto aproximadamente 3 horas a la semana, durante las clases prácticas de la asignatura de Diseño de Software, si bien también se ha realizado trabajo fuera del aula.

Fase de inicio

En esta primera fase, nos centramos en realizar una lectura comprensiva del enunciado del proyecto, subrayando los puntos clave del videojuego. Aquí pudimos identificar los elementos principales del juego, como son las potencias, las unidades, las construcciones o las celdas que conforman el tablero.

Sin embargo, nos percatamos de que había ciertas ambigüedades en el guión, por lo que se propusieron en el equipo ciertas reglas para poder resolverlas:

- Se estableció un límite mínimo y máximo de acciones por jugador, es decir, por cada turno que pase, el jugador debe respetar las acciones que se deben cumplir para que el jugador pueda pasar a la siguiente ronda. El mínimo se estableció en 1 acción por turno para cada jugador, mientras que el máximo se configuró en 5.
- Tanto los robots como las potencias tendrán 3 depósitos distintos, para así poder recoger y almacenar los recursos. Estos serán: agua, metal y silicio (son los recursos especificados por el enunciado)
- En una misma celda, no podrá haber dos unidades de dos potencias distintas simultáneamente. En una celda solo podrá haber una construcción en un mismo instante temporal.
- Habrá una dificultad asociada al tablero de juego. Esta modificará los valores de ataque (las unidades y construcciones infringirán más daño), la cantidad de recursos (a mayor dificultad, menos recursos) y la duración de las tormentas (como se describe en el punto siguiente). Sin embargo, debemos mencionar que el equipo de implementación software del proyecto podrá trabajar con libertad en este aspecto.
- Las tormentas de arena generadas en Marte tendrán una duración concreta de turnos. Después, esta se disipará totalmente. Para que la duración de estos fenómenos y el tamaño y dificultad del tablero estén relacionados (queremos que el crecimiento de la tormenta sea proporcional de alguna manera al tamaño del mapa), haremos que los turnos de tormenta se calculen como:

$$\text{turnosTormenta} = \left\lceil \sqrt{\text{areaTablero}} \right\rceil + \text{dificultadTablero}^2$$

- La partida finalizará cuando solo quede una potencia en pie. Si quedan solamente dos potencias activas y estas eran aliadas, a partir de ese momento se romperá su acuerdo y deberán seguir hasta que solo quede una.

Fase de elaboración

Durante la Fase de Elaboración, que abarcó cuatro semanas e incluyó tres sesiones de prácticas, nos centramos principalmente en actividades de análisis. En esta etapa, se trabajó en la creación de un modelo de comportamiento que describiera las acciones que el sistema debía realizar, sin entrar en detalles sobre cómo se llevarían a cabo.

Es por ello que, durante esta etapa, nos hemos enfocado en el desarrollo del modelo de casos de uso. Aunque hemos abordado este aspecto de manera gradual a lo largo del proyecto, ha servido como un sólido punto de partida para establecer la base de nuestro trabajo. Además, hemos elaborado el modelo de vocabulario, capturando de manera precisa la estructura del dominio de aplicación, identificando los conceptos importantes y las relaciones que hay entre ellos.

3.1. Modelo de casos de uso

Hemos modelado el diagrama de casos de uso siguiendo las especificaciones del enunciado del proyecto. Como podemos ver en la Figura 3.1, hay dos actores diferenciados en nuestro diagrama, el Administrador y el Jugador.

El Administrador es responsable de 6 casos de uso. Podemos generalizarlos en acciones necesarias para iniciar una partida (Iniciar partida, Alta jugador, Elegir tipo tablero), manipulación de tormentas aleatorias en el mapa (Generar tormenta, Expandir tormenta) y visualización del terreno de juego (Dibujar tablero).

Por otra parte, el Jugador se encarga de realizar las acciones pertinentes para dar sentido a la dinámica del juego, como pueden ser crear unidades y construcciones, mover una unidad o atacar a una unidad o construcción enemiga. Como se puede ver en la leyenda definida en la Figura 3.1, los diferentes colores están asociadas a diferentes elementos del juego (administrador, construcciones, unidades, información y alianzas).

Podemos ver además en este diagrama completo, que existen relaciones del tipo *extend* e *include* entre los casos de uso. Con el fin de proporcionar claridad al informe, mostramos en las Figuras 3.2 y 3.3 las diferentes relaciones que tienen el caso de uso Dibujar tablero y Ataque automático torre[1].

Como podemos observar, el ataque automático de la torre (Figura 3.2) se produce cada vez que la torre detecta que hay una unidad moviéndose a su alrededor, ya sea con un movimiento explícito o cada vez que ocurre una agrupación o desagrupación de unidades. Veremos en un capítulo posterior la relación de este caso de uso con el patrón de diseño Observer.

[1]Pasar turno también tiene *extends* asociados con los casos de uso Generar tormenta y Expandir tormenta. Esto se debe a que existe la posibilidad de que se genere una tormenta de arena cuando un jugador pasa su turno. No hemos añadido una captura explícita de este caso ya que hemos querido otorgarle más importancia a las relaciones de las otras dos acciones.

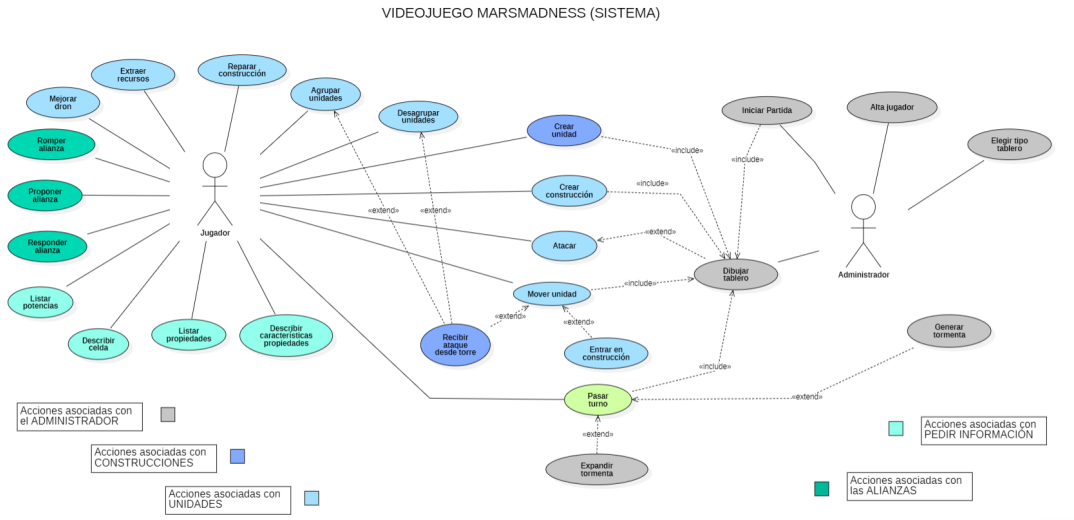

Figura 3.1: Diagrama completo de casos de uso

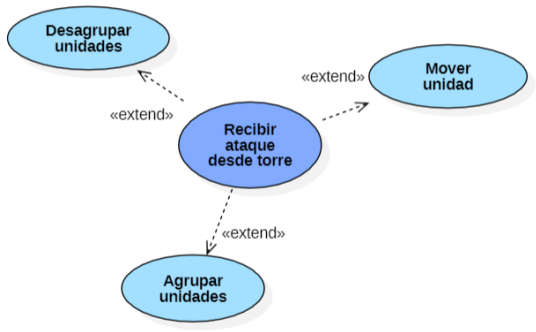

Figura 3.2: Relaciones con el caso de uso Ataque automático torre

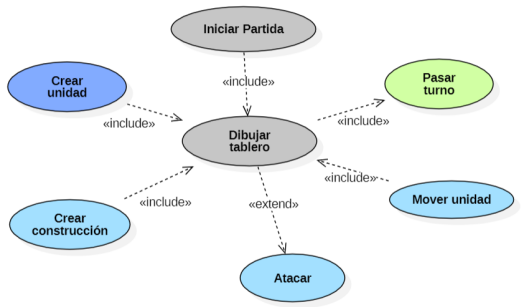

Figura 3.3: Relaciones con el caso de uso Dibujar tablero

Por otra parte, vemos que el caso de uso Dibujar tablero (Figura 3.3) es extendido/incluído por otros casos de uso de nuestro diagrama. Esto quiere decir que las acciones que incluyen este caso de uso, al final de su acción deben dibujar el tablero de juego en ese instante de la partida. Por otra parte, vemos que Atacar y Dibujar tablero se relacionan mediante un *extend*. Solamente cuando haya cambios en el mapa debido a una acción de ataque se debe llamar a Dibujar tablero, y esto es lo que refleja esta relación entre ambos casos.

3.2. Modelo de vocabulario

Se presenta a continuación el diagrama de vocabulario expuesto en nuestro starUML (Figura 3.4). Este diagrama representa los conceptos más importantes del juego, así como las relaciones entre ellos.

Comenzaremos describiendo los elementos principales del juego. Como podemos ver, tenemos una clase Potencia, que tiene diferentes unidades, construcciones y recursos. Observamos una doble jerarquía en la clase Unidad no tripulada, dividida en Rover, Robot y Dron. Este último puede generalizarse de manera parcial en diferentes clases, siendo estas Blindado, Tanque y UAV, ya que los drones se pueden especializar

de diferentes maneras (sin dejar de lado el tipo básico de dron). Por último, en la clase Construccion podemos ver otra generalización, ya que cada una de las instancias puede ser del tipo Estación, Torre o Base.

Debemos mencionar también la clase Celda, que compone la clase Tablero de nuestro juego. Es importante guardar en estas clases la posición y la dimensión respectivamente; utilizaremos este último atributo para delimitar el terreno de juego. Las clases tienen diferentes asociaciones con los elementos del juego: Una celda puede albergar construcciones, unidades o recursos, y además, puede estar afectada por una tormenta de arena o tener la capacidad de ser visualizada por una potencia.

Veremos más adelante, en el modelo estructural, que las clases Recurso y Tormenta de este modelo no persistirán. Sin embargo, a nivel conceptual, sigue teniendo sentido mantenerlas en este punto del proyecto.

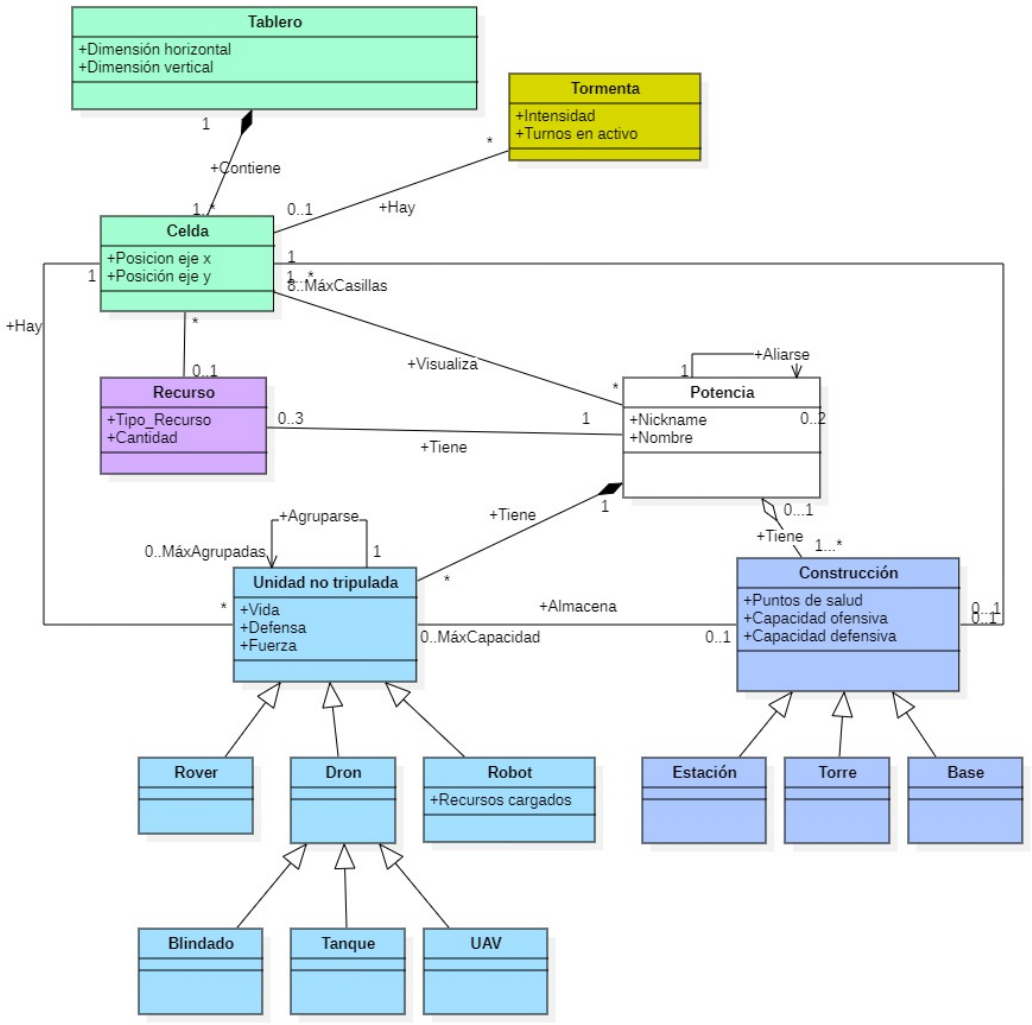

Figura 3.4: Diagrama completo de vocabulario

Fase de construcción

En este capítulo, nos adentramos en la fase de construcción del proyecto, donde comenzaremos a diseñar el comportamiento de nuestro videojuego. Debemos tener claro que, en un proyecto real, se debería seguir una secuencia lineal de análisis - diseño - implementación - pruebas (lo que en el mundo de la Ingeniería del Software se conoce como "ciclo de vida en cascada"). Sin embargo, como bien sabemos, en esta asignatura no disponemos del suficiente tiempo para abarcar todo el proceso que describimos.

Por ello, se ha decidido optar por algo más flexible y adaptable: un enfoque iterativo e incremental. En lugar de intentar abarcar todo el proyecto de una vez, este se dividirá en partes más manejables llamadas iteraciones. Cada una se enfoca en una parte específica del proyecto.

Esta forma de trabajar tiene muchas ventajas. Permite manejar mejor la complejidad, otorga la libertad de ajustar y adaptar nuestro enfoque sobre la marcha y hace que podamos aprender y mejorar continuamente a medida que avanzamos.

Podríamos decir que, al adoptar este enfoque más ágil, estamos apostando por una forma más efectiva y eficiente de trabajar, lo que nos ayudará a obtener un resultado final de mayor calidad.

4.1. Planificacion

La organización es un pilar fundamental en todo proyecto, tanto individual como grupal. Sin una planificación temporal efectiva, incluso la idea más brillante puede desmoronarse. Es por ello que hemos definido detalladamente nuestra planificación propuesta en la Tabla 4.1. Este cronograma nos permitirá avanzar de manera progresiva y sistemática hacia nuestros objetivos a lo largo del desarrollo del proyecto.

Un aspecto crucial de esta planificación es el diagrama de casos de uso, el cual nos brinda una visión clara de las funcionalidades que deben ser implementadas. Definir la prioridad de estas acciones nos ayudará a determinar la secuencia adecuada para abordar cada aspecto del proyecto.

Como primera iteración, se plantea modelar la "columna vertebral" de nuestro software. Esto incluye la funcionalidad básica, con el fin de poder tener una primera versión de prueba para el funcionamiento del juego. Se han tenido en cuenta solamente acciones consideradas esenciales, como son las relacionadas a comenzar una partida, acciones de construcción de unidades y construcciones, movimiento básico de unidades, primeras funcionalidades de descripción y la posibilidad de pasar turno entre potencias.

Para la segunda iteración, consideramos seguir avanzando en este desarrollo del proyecto, e intentamos implementar el funcionamiento de lo que podemos considerar el "cerebro" de nuestra aplicación. Aquí podemos considerar acciones más avanzadas, como pueden ser la extracción de recursos, reparación de construcciones, el tratamiento de las agrupaciones de unidades o el ataque. El ataque básico se planificó para la iteración 2, mientras que otros tipos de ataque estaban previstos para la iteración 3. Sin embargo,

por motivos de comodidad, decidimos agrupar todos los modelados de ataque en la última iteración. A pesar de esto, el inicio del modelado de ataques realmente pertenece a la segunda iteración, como se refleja en nuestra planificación.

Finalmente, en la última iteración, planteamos añadir los detalles finales al trabajo, como pueden ser las acciones restantes de listado, la posibilidad de incorporar otros tipos de tableros, la gestión de alianzas o el tratamiento de tormentas.

Es importante resaltar que, durante el transcurso de las iteraciones 2 y 3, nos sumergimos más en el aprendizaje de patrones de diseño y en la creación de diagramas de secuencia. Aunque nuestro progreso está marcado por las iteraciones, es crucial entender que este proceso no ha sido completamente lineal. En algunos casos, hemos tenido que retroceder a una iteración anterior para ajustar detalles que requerían mayor atención o refinamiento. Es fundamental reconocer que esta ida y vuelta no representa un obstáculo, sino más bien una oportunidad para perfeccionar nuestro trabajo.

Iteración	**Objetivo**	**Fecha inicio**	**Fecha fin**	**Casos de uso**
1	Realizar las acciones básicas que proporcionan el funcionamiento más sencillo para poder comenzar a hacer pruebas con el programa	07/02	06/03	Prioridad alta: Alta jugador, Elegir tipo tablero (DEFAULT), Dibujar tablero, Iniciar partida, Mover unidad, Crear unidad, Crear construcción, Pasar turno, Describir celda
2	Implementar acciones más complejas, pero que aportan el verdadero sentido al juego	06/03	09/04	Prioridad media: Extraer recursos, Entrar en construcción, Reparar construcción, Agrupar unidades, Desagrupar unidades, Mejorar dron, Atacar
3	Añadir detalles finales de implementación, para dar completitud al software	09/04	17/05	Prioridad baja: Listar propiedades, Listar potencias, Describir características propiedades, Crear Tormenta, Expandir tormenta, Otros tableros, Responder alianza, Proponer alianza, Eliminar alianza

Tabla 4.1: Planificación temporal del proyecto (por iteraciones)

4.2. Iteración 1

4.2.1. Diagrama de clases

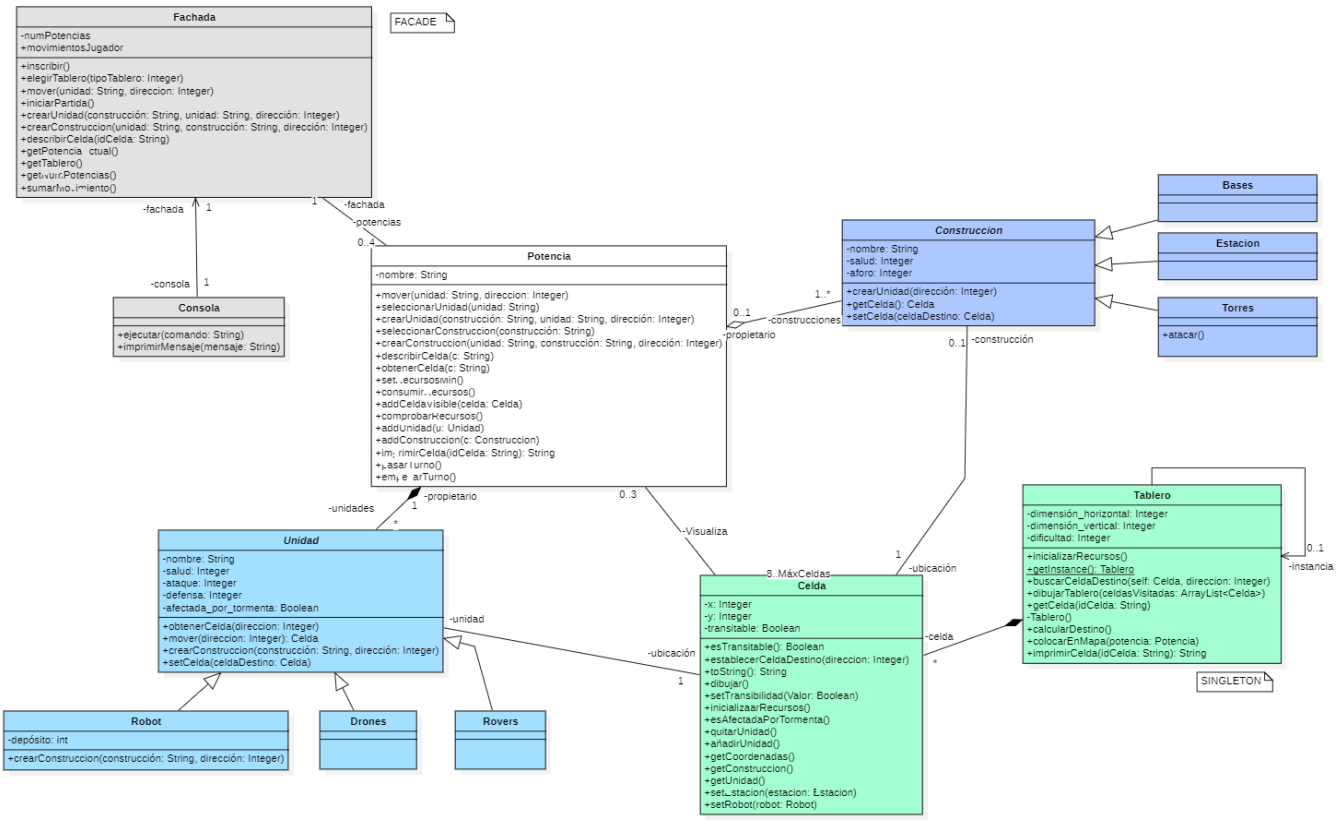

Figura 4.1: Diagrama de clases (Iteración 1)

En la Figura 4.1 podemos ver la primera versión de nuestro diagrama de clases. Aún queda mucho por desarrollar, pero como primer modelo asienta una buena base de nuestro programa. Comentaremos los aspectos más importantes, así como algunos de los diagramas de secuencia.

Nuestra clase **Consola** es la parte más externa de nuestro software. Se encarga de procesar el texto que nuestro Usuario inserte por pantalla, redirigiéndolo a la siguiente clase, la Fachada.

La **Fachada** es el resultado de aplicar el patrón Facade, descrito en el siguiente punto.

La clase **Potencia** representa y alberga todas las propiedades del jugador, siendo estas **Unidades** y **Construcciones**. Para reflejar que la existencia de una unidad depende de la existencia de la potencia que la contiene, usamos composición entre estas dos clases. A su vez, una construcción pertenece a una potencia, pero su existencia no depende de la de la potencia, por lo que usamos agregación entre ellas. Algo parecido ocurre en la relación entre **Celda** y **Tablero**, en la que usamos composición para dejar claro que una celda existe siempre y cuando el tablero que la contiene exista también.

La clase denominada **Unidad** es una clase abstracta que permite a todos sus tipos heredar las funciones y los atributos comunes, en donde se encuentra información a cerca de estos tipos de unidades. Dentro de los tipos encontramos los **Robots**, los **Drones** y los **Rovers**, estas clases que heredan de **Unidad**

gestionan todos los atributos de las unidades de cada tipo respectivamente.

La clase **Construcción** es una clase abstracta que permite a todos los tipos de construcción heredar los atributos y funciones comunes. Dentro de los tipos de construcciones están las **Bases**, las **Bases**, las **Estaciones** y las **Torres**, estas clases que heredan de **Construcción** gestionan todos los atributos de las construcciones de cada tipo respectivamente.

La clase **Celda** es en la que se almacena todos los datos relacionados con las celdas, indicando su posición mediante unas coordenadas, si es transitable, las unidades que estan en ella, etc.

La clase **Tablero** será la clase que contiene el tablero correspondiente a una partida, donde se guarda información de la misma. Se aplica en él el patrón de Singleton, el cuál describiremos en el siguiente apartado de la memoria. Asimismo, veremos en la iteración 3 del proyecto que el modo de visualización de celdas de cada potencia sufrirá una modificación importante y efectiva, implementando el patrón Proxy.

4.2.2. Patrones de diseño

4.2.2.1. Facade

El patrón Facade es un tipo de patrón estructural que permite proporcionar una "interfaz" unificada sencilla que haga o funcione como intermediaria entre un cliente y una interfaz o un grupo de interfaces más complejas, proporcionando una interfaz simple en un subsistema complejo, sirviendo estas de punto de entrada a cada nivel. En nuestro, caso, Fachada es la intermediaria entre la Consola y las Potencias del juego. Mejora la modularidad y el mantenimiento al exponer solo lo esencial para los clientes y ocultar los detalles internos.

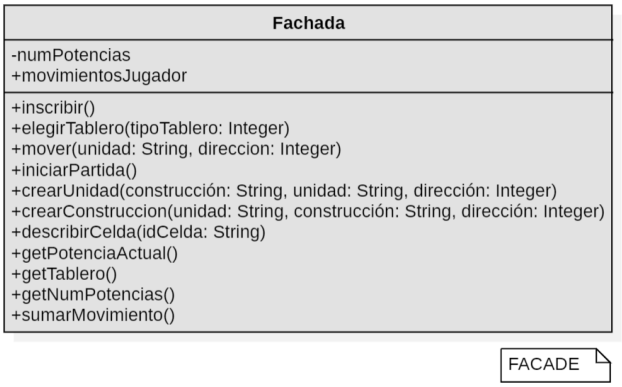

Figura 4.2: Patrón Facade

4.2.2.2. Singleton

El patrón Singleton nos asegura que solo haya una única instancia de una clase en la aplicación, lo que facilita el acceso global a esa instancia y evita la creación repetida de la misma. Sin duda este patrón es idóneo para ser aplicado a la clase Tablero, ya que para una partida solo existe un tablero. Esto es útil para conservar recursos y mantener la consistencia en el sistema.

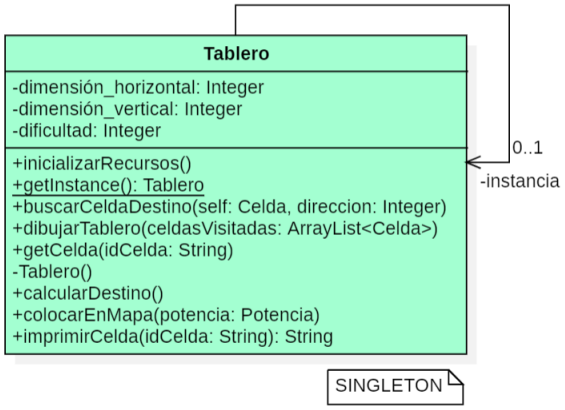

Figura 4.3: Patrón Singleton

4.2.3. Casos de uso

4.2.3.1. Alta Jugador

En el siguiente diagrama de secuencia de la Figura 4.4 se muestra el funcionamiento del caso de uso **AltaJugador**, que crea un nuevo jugador otorgándole los elementos iniciales necesarios, descritos en el enunciado del proyecto.

Esta funcionalidad la ejecuta el administrador, el cual usa la Consola para poder mandar la petición a la Fachada. Una vez en la Fachada, esta nos dirige para llevar a cabo el procedimiento.

Primero comprueba que la potencia puede crearse, es decir, comprueba que el número de potencias creadas hasta el momento es menor a el máximo establecido. Si las condiciones son favorables, se crea una nueva potencia que se liga al jugador. Asimismo, se le adjudica a la nueva potencia una estación y un robot además de los recursos mínimos establecidos.

Este diagrama se corresponde con una de las primeras tomas de contacto con los diagramas de secuencia, por lo que nos ha servido para familiarizarnos con este tipo de modelos.

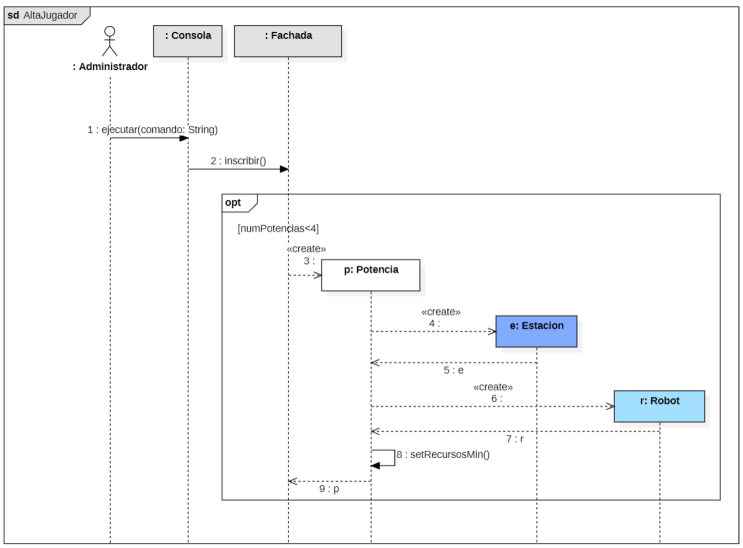

Figura 4.4: Diagrama de secuencia de Alta Jugador

4.2.3.2. Mover Unidad

A continuación, analizaremos uno de los diagramas de secuencia más interesantes de esta iteración: el **movimiento de una unidad**. En este proceso, el jugador inicia la acción enviando un comando a la consola, la cual notifica a la fachada sobre la acción requerida. La fachada entonces identifica la potencia que tiene el turno actual y le envía la solicitud de mover la unidad correspondiente.

Desde la potencia, se selecciona la unidad que se desea mover. La unidad proporciona su posición actual, y el tablero calcula la celda de destino basándose en la dirección especificada por el jugador. Una vez identificada la celda de destino, se verifica si es transitable. Si es así, la unidad se añade a la nueva celda y se elimina de la celda anterior.

Además, se actualiza la posición registrada en la celda afectada.Finalmente, la celda de destino se añade al conjunto de celdas visibles para la potencia y se envía una señal de confirmación a la fachada. La fachada, al recibir esta confirmación, da por completada la acción e incrementa el número de movimientos realizados por la potencia en el turno actual.

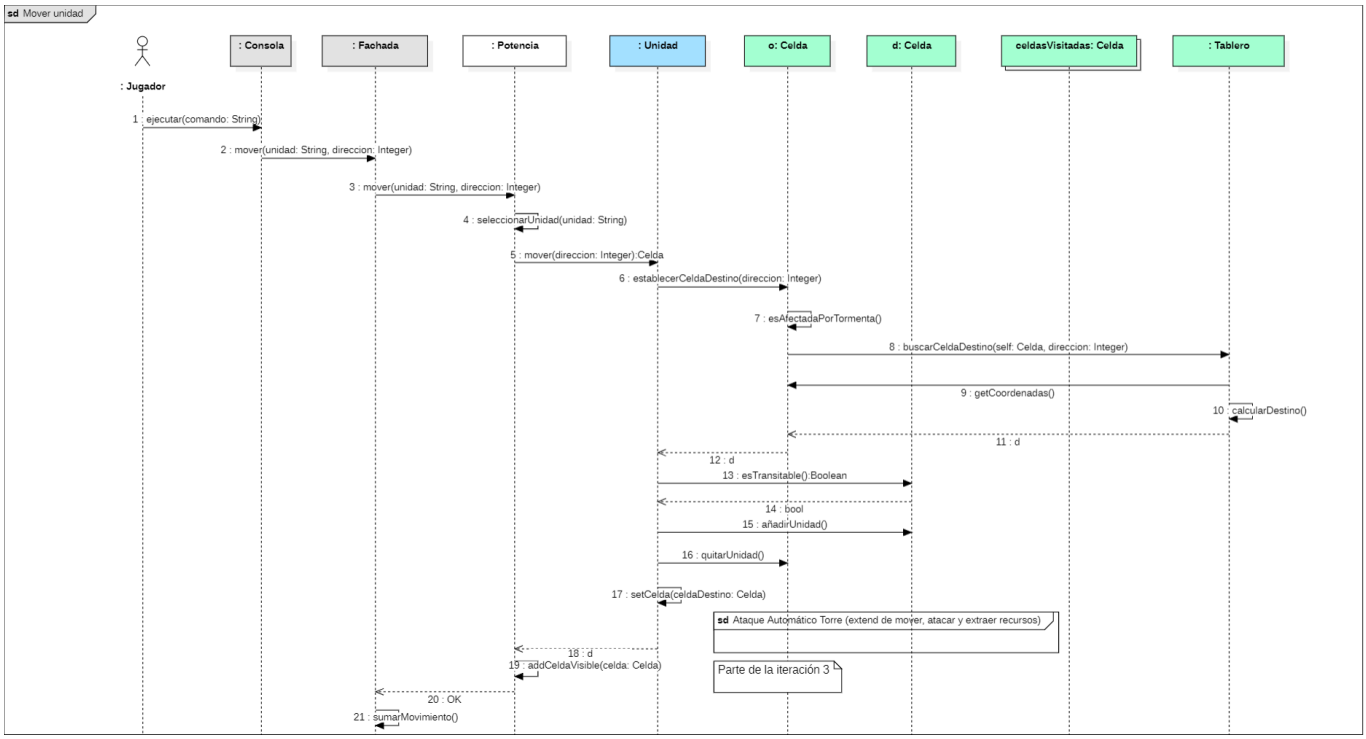

Figura 4.5: Diagrama de secuencia de Mover Unidad (Iteración 1)

4.2.3.3. Crear Construcción

Por último, explicaremos la implementación del diagrama de secuencia correspondiente al caso de uso **CrearConstrucción**, que permite a un jugador crear una nueva construcción siempre y cuando se cumplan las condiciones establecidas. El jugador inicia esta funcionalidad enviando un comando a la consola, la cual notifica a la fachada sobre la acción solicitada.

La fachada primero identifica la potencia que ejecuta el comando para obtener la información necesaria sobre ella.Se verifica si la potencia dispone de los recursos suficientes para llevar a cabo la creación. Si es así, se selecciona la unidad encargada de la construcción.La unidad seleccionada debe ser un robot, ya que es la única capaz de crear construcciones. A partir del robot, se obtiene la celda en la que se encuentra y,

usando la dirección pasada como parámetro, se calcula la celda de destino. El robot verifica si la celda de destino es transitable. Si es transitable, se procede a la creación de la construcción en la celda de destino.

Finalmente, la nueva construcción se registra en la potencia, que actualiza su registro de construcciones y deduce los recursos invertidos en la operación. La fachada recibe una aprobación para confirmar la operación e incrementa el número de movimientos de la potencia en el turno actual.

Es importante señalar que hay una funcionalidad no contemplada en este diagrama: cuando una unidad se mueve al perímetro de una torre, esta ejerce un ataque automático. Este caso se recoge en la iteración 3.

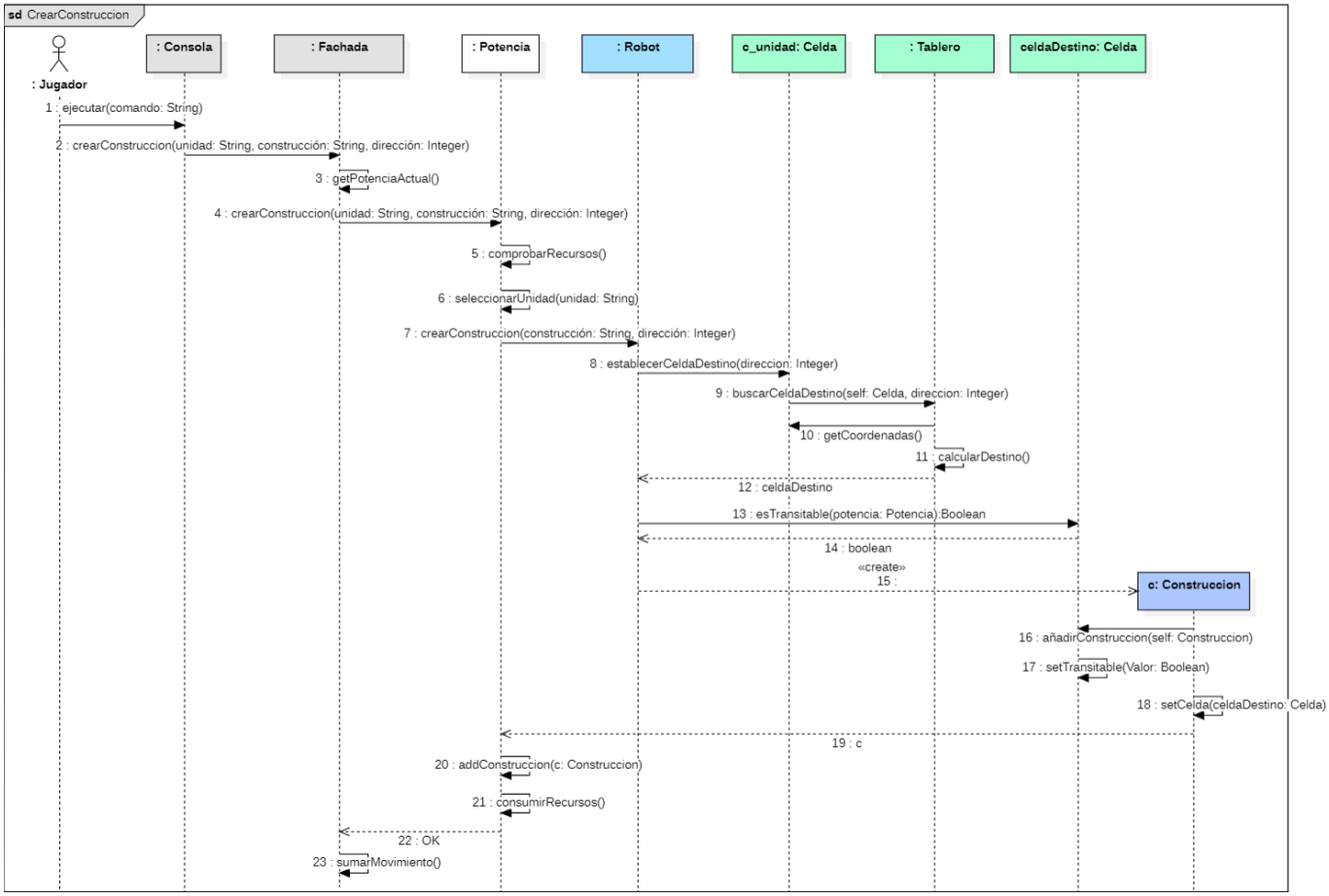

Figura 4.6: Diagrama de secuencia de Crear Construccion (Iteración 1)

4.3. Iteración 2

4.3.1. Diagrama de clases

Presentamos en la Figura 4.7 el diagrama de clases asociado a la iteración 2 del proyecto. En esta iteración se han aplicado varios patrones de diseño tras adquirir un conocimiento más profundo de las clases y sus comportamientos. Los patrones utilizados incluyen el patrón de creación Factory Method, los patrones estructurales Decorator y Composite, y los patrones de comportamiento Observer y Strategy. Algunos de estos patrones han requerido la creación de nuevas clases para definir correctamente su funcionamiento.

Entre las nuevas clases, destacan las relacionadas con el **Dron**, que se pueden observar en la parte inferior derecha de la Figura 4.7. La clase Dron ha sido modificada para aplicar el patrón Decorator. Ahora, existe una clase abstracta derivada de `Unidad`, llamada `Dron`, que engloba todo lo relacionado con esta estructura y sus decoradores. De esta estructura derivan el tipo básico de dron y también la clase decorador **DronDecorator**. Esta clase tiene tres especializaciones: `TanqueDecorator`, `UAVDecorator` y `BlindadoDecorator`, que representan diferentes equipamientos del dron. Veremos en el diagrama de secuencia del caso de uso Mejorar dron la aplicación de este patrón.

En relación con el patrón **Observer**, hemos introducido las clases `Sujeto` e `IObservador`, que son la base de este patrón y se explicarán en detalle en el siguiente apartado. Estas clases están estrechamente relacionadas con `Celda` y `Torre`, como se puede ver en la parte inferior izquierda de la Figura 4.7.

Finalmente, se destacan las nuevas clases creadas para el patrón **Strategy**. Estas clases están coloreadas en verde y se encuentran relacionadas con `Tablero` en la parte izquierda de la Figura 4.7. `IGenerationStrategy` se conecta mediante una asociación con la clase `Tablero`, mientras que sus descendientes, `LoadedStrategy`, `PseudorandomStrategy` y `DefaultStrategy`, derivan de ella mediante una generalización. Esto implica que `Tablero` tiene diferentes métodos de creación que se delegan a estas clases. Se presentarán ejemplos de diagramas de secuencia de este patrón en esta sección.

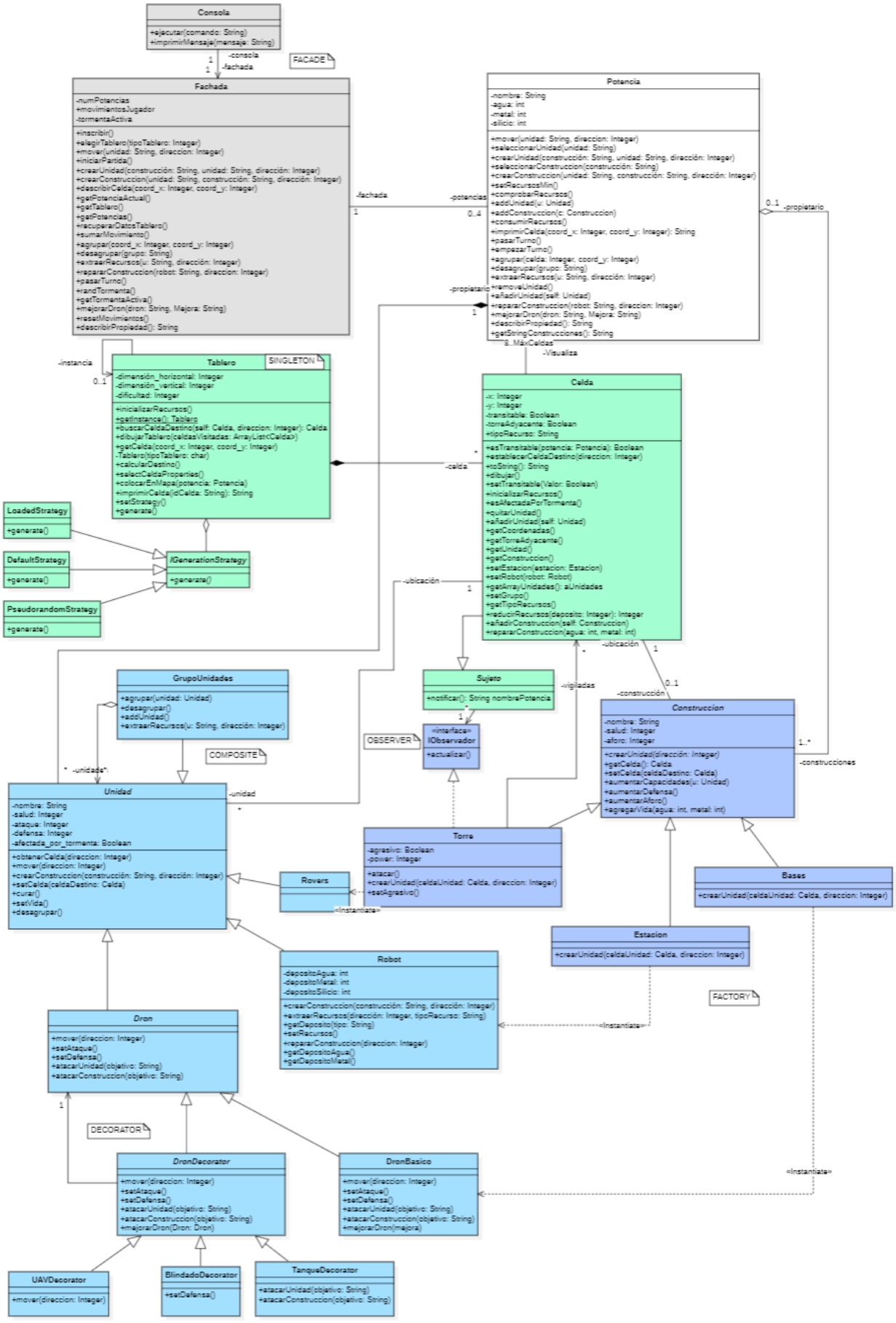

Figura 4.7: Diagrama de clases (Iteración 2)

4.3.2. Patrones de diseño

4.3.2.1. Composite

El patrón **Composite** es un patrón de diseño estructural que permite tratar objetos individuales y compuestos de manera uniforme. Se utiliza para representar jerarquías de objetos, facilitando la manipulación de estructuras complejas de forma recursiva. En nuestro trabajo, hemos decidido aplicar este patrón para las agrupaciones de unidades, tal y como podemos ver en la Figura 4.8. Veremos cómo se maneja la agrupación y desagrupación de unidades, así como la gestión de los grupos a la hora de extraer recursos.

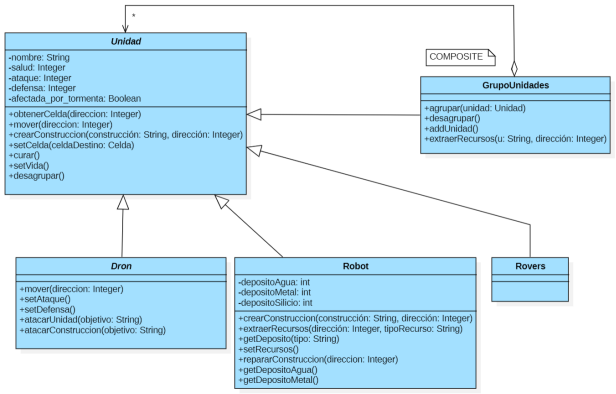

Figura 4.8: Patrón Composite

4.3.2.2. Factory Method

El patrón de diseño Factory Method es un patrón de creación que sirve para delegar la creación de objetos a una clase "fábrica" en lugar de instanciarlos directamente. Hemos pensado que este patrón podría encajar perfectamente entre construcciones y unidades, ya que cada tipo de Construcción instancia un tipo diferente de Unidad.

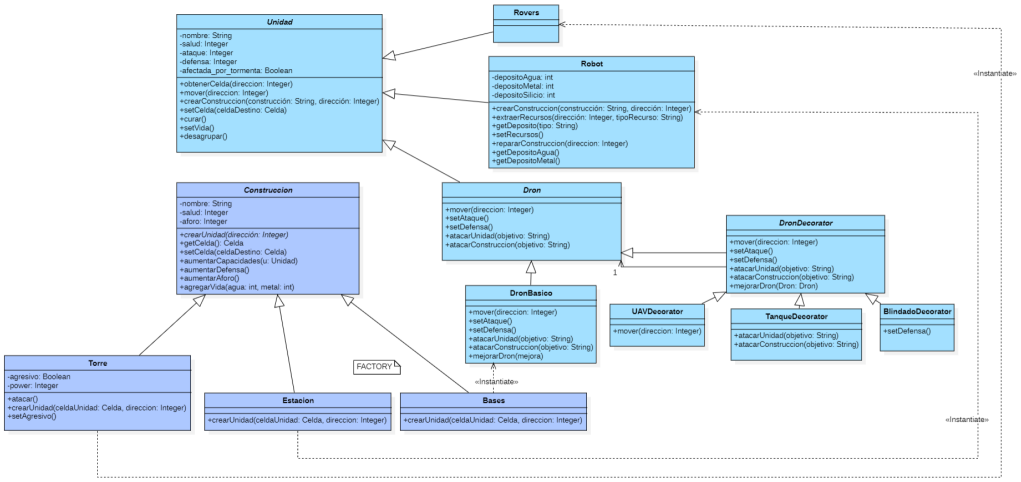

Figura 4.9: Patrón Factory Method

4.3.2.3. Strategy

El patrón **Strategy** es un patrón de diseño de comportamiento que permite definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. Esto permite que el algoritmo varíe independientemente de los clientes que lo usan. En nuestro diagrama, podemos ver en la Figura (4.10) que aplicamos este patrón en el tablero, ya que este tiene diferentes modos de generación. En esta iteración, representaremos dos diagramas de secuencia relacionados con este patrón: DefaultStrategy y PseudorandomStrategy. Dejaremos el diagrama de secuencia del LoadedStrategy para la tercera iteración, en la que explicaremos el patrón DAO + Abstract Factory.

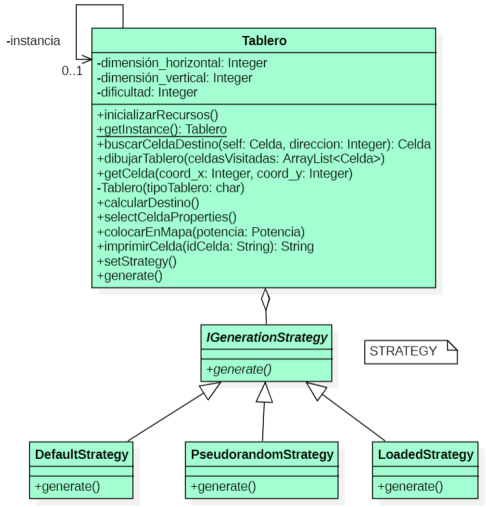

Figura 4.10: Patrón Strategy

4.3.2.4. Decorator

El patrón **Decorator** es un patrón de diseño estructural que permite añadir funcionalidad a objetos de manera dinámica y flexible sin modificar su estructura original. Utiliza una serie de clases envolventes (decoradores) que "envuelven" al objeto original para extender su comportamiento. Hemos tomado la decisión de aplicar este patrón al caso del dron, como vemos en la Figura (4.11). El dron puede especializarse en diferentes clases: UAV, Blindado o Tanque.

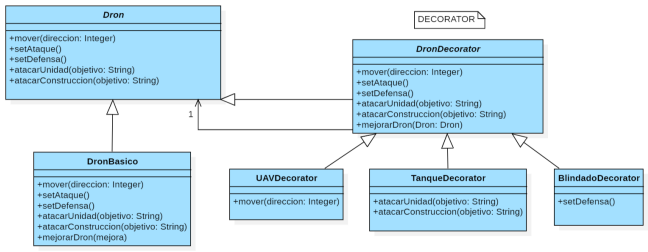

Figura 4.11: Patrón Decorator

4.3.2.5. Observer

El patrón **Observer** (Figura 4.12) es un patrón de diseño de comportamiento que define una relación de dependencia uno-a-muchos entre objetos, de manera que cuando uno de ellos cambia de estado, todos sus dependientes son notificados y actualizados automáticamente. Decidimos que este patrón sería conveniente a la hora de manejar el ataque automático de las torres, siendo estas las observadoras y las celdas los sujetos observados. Para su aplicación se crea la interfaz IObservador, y la clase abstracta *Sujeto*, las cuales sirven como intermediarias entre el tablero Mostraremos este patrón en el diagrama de secuencia Ataque automático torre en la iteración 3.

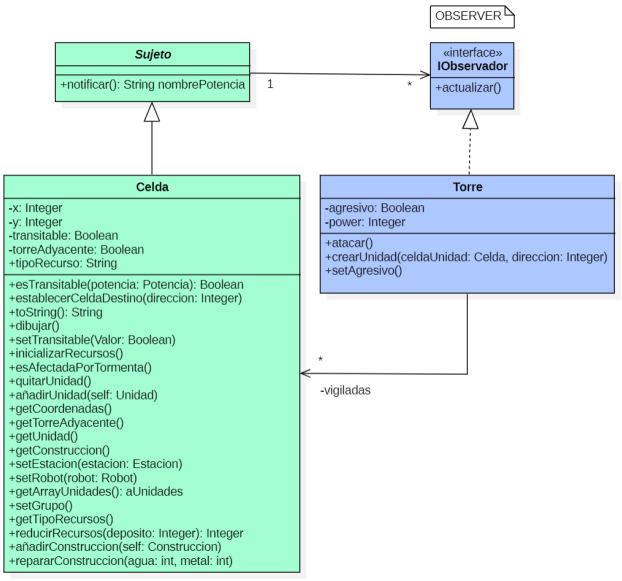

Figura 4.12: Patrón Observer

4.3.3. Casos de uso

4.3.3.1. Agrupar unidades

Describiremos un caso de uso en el que se expone el patrón Composite: **Agrupar unidades**. Para crear un grupo de unidades, debemos incluirle las unidades que lo conforman, como hacemos en el mensaje 8 (Figura 4.13). Además de esto, se deben quitar las unidades individuales de la potencia y de la celda y añadir el grupo, como hacemos en los mensajes 10, 11 y 12 respectivamente. De esta manera, nos aseguramos de que, mientras las unidades estén agrupadas, solo podremos dirigirnos al grupo para que realice acciones, no a las unidades individualmente.

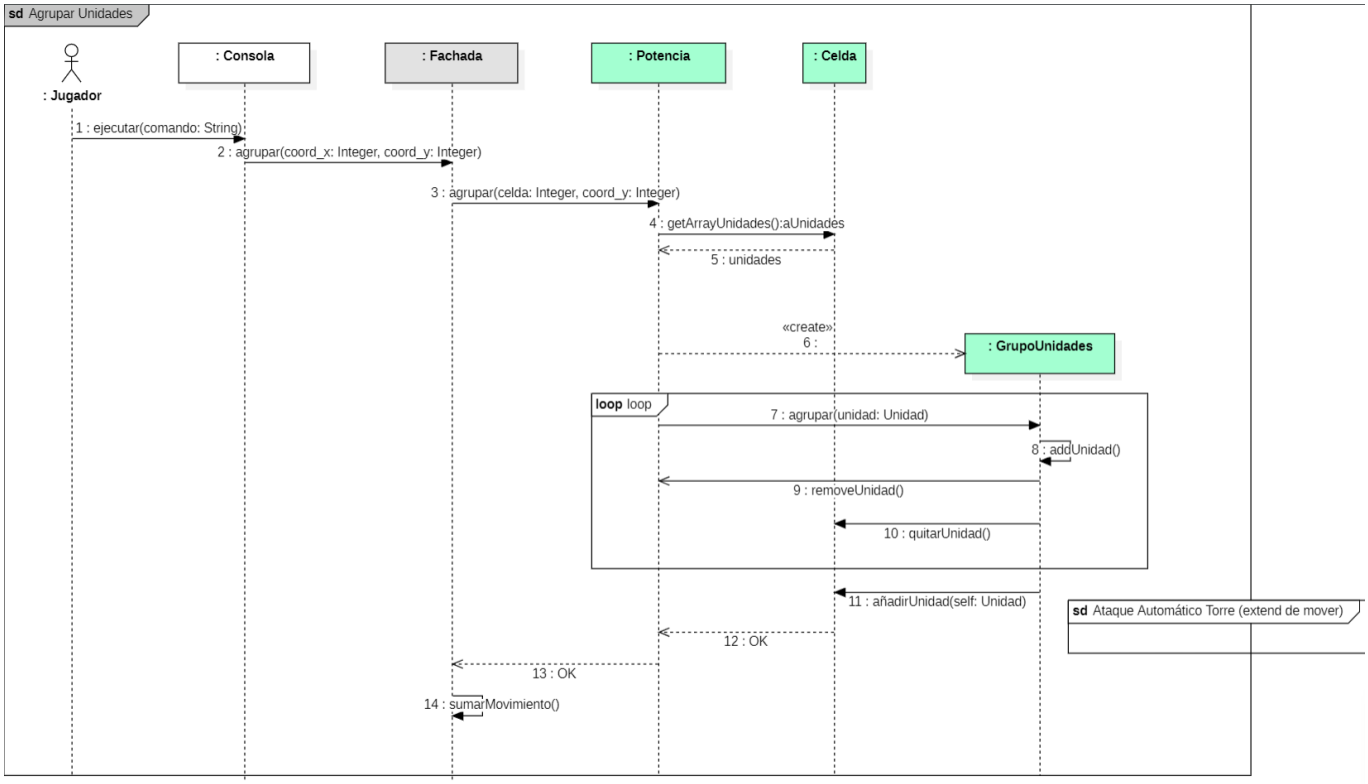

Figura 4.13: Diagrama de secuencia de Agrupar unidades (Iteración 2)

4.3.3.2. Desagrupar unidades

Cuando desagrupamos un conjunto de unidades, debemos asegurarnos de que cada unidad recupere su independencia. Para ello, es necesario desacoplar cada unidad del grupo de manera recursiva, volviéndolas a añadir a la potencia y a la celda de forma individual. **Aquí es donde entra en juego la filosofía del patrón Composite**. Este patrón nos permite acceder a cada elemento de una estructura compuesta de manera individual, de modo que cada elemento pueda realizar las acciones necesarias.

Para demostrar un buen manejo del patrón Composite, consideramos un escenario donde necesitamos desagrupar un grupo de unidades (Figura 4.14). Este grupo incluye a su vez otro grupo con dos elementos y un elemento adicional. Este es un ejemplo típico que ilustra cómo el patrón Composite permite tratar a los objetos individuales y compuestos de manera uniforme, facilitando la gestión de estructuras jerárquicas complejas de manera clara y eficiente.

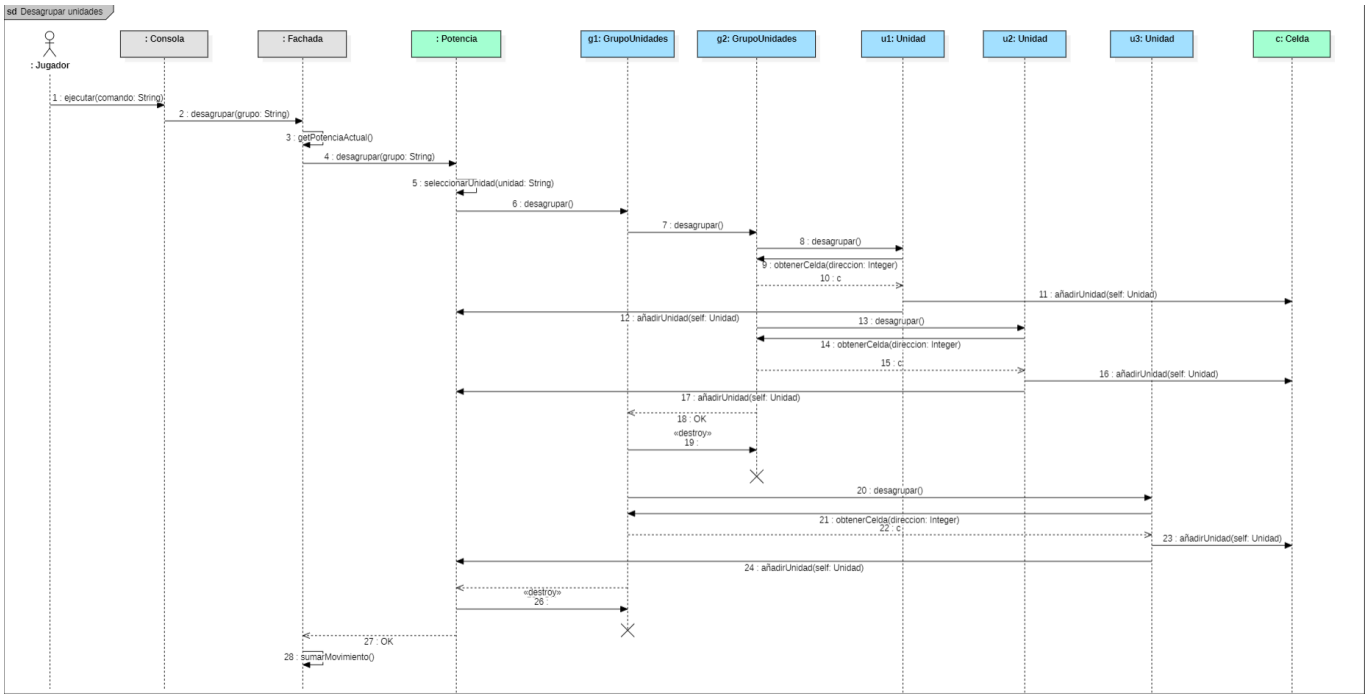

Figura 4.14: Diagrama de secuencia de Agrupar unidades (Iteración 2)

4.3.3.3. Elegir tipo tablero (default)

En este diagrama de secuencia (Figura 4.15), demostraremos el manejo del patrón *Strategy* con el caso de uso *Elegir tipo tablero (DEFAULT)*. En este diagrama podemos ver que, después de crear el tablero de juego, se establece la estrategia del mismo, de manera que cuando llamamos a la función `generate()`, se crea una instancia de la clase `DefaultStrategy`, en la que se delega el trabajo de crear nuestro tablero. La función `generate()` crea todas las celdas del tablero en un bucle, e inicializa (o no) de manera predeterminada cada una. Una vez que están todas las celdas creadas, podemos concluir que el tablero está creado, por lo que ya se puede devolver hasta la *Fachada*.

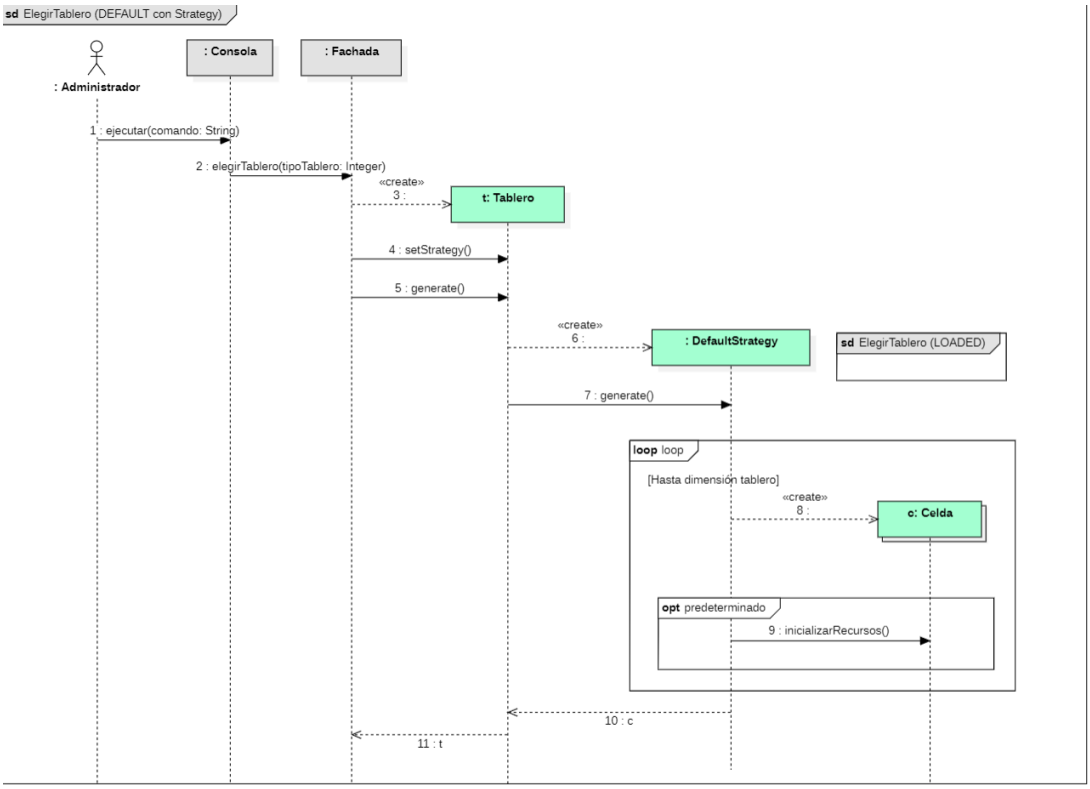

Figura 4.15: Diagrama de secuencia de Elegir tipo tablero (DEFAULT) (Iteración 2)

4.3.3.4. Extraer recursos

Aquí podemos ver el diagrama de secuencia de uno de las acciones más importantes de nuestro juego: Extraer recursos (Figura 4.16). Como podemos ver, el jugador insertará por terminal el comando, introduciendo el nombre de la unidad que extraerá y la dirección en la que lo hará. Cabe destacar que solamente los robots pueden realizar esta acción, por lo que hemos instanciado una clase Robot en vez de una clase Unidad. El Robot buscará la celda de destino, es decir, en la que realizará la extracción, y procederá a hacerla. Restará los recursos de la celda y se lo sumará a su depósito, siempre y cuando tenga espacio (si tiene espacio, pero no el suficiente como para coger todos los recursos de la celda, almacenará los que pueda). Finalmente, se sumará un movimiento al jugador.

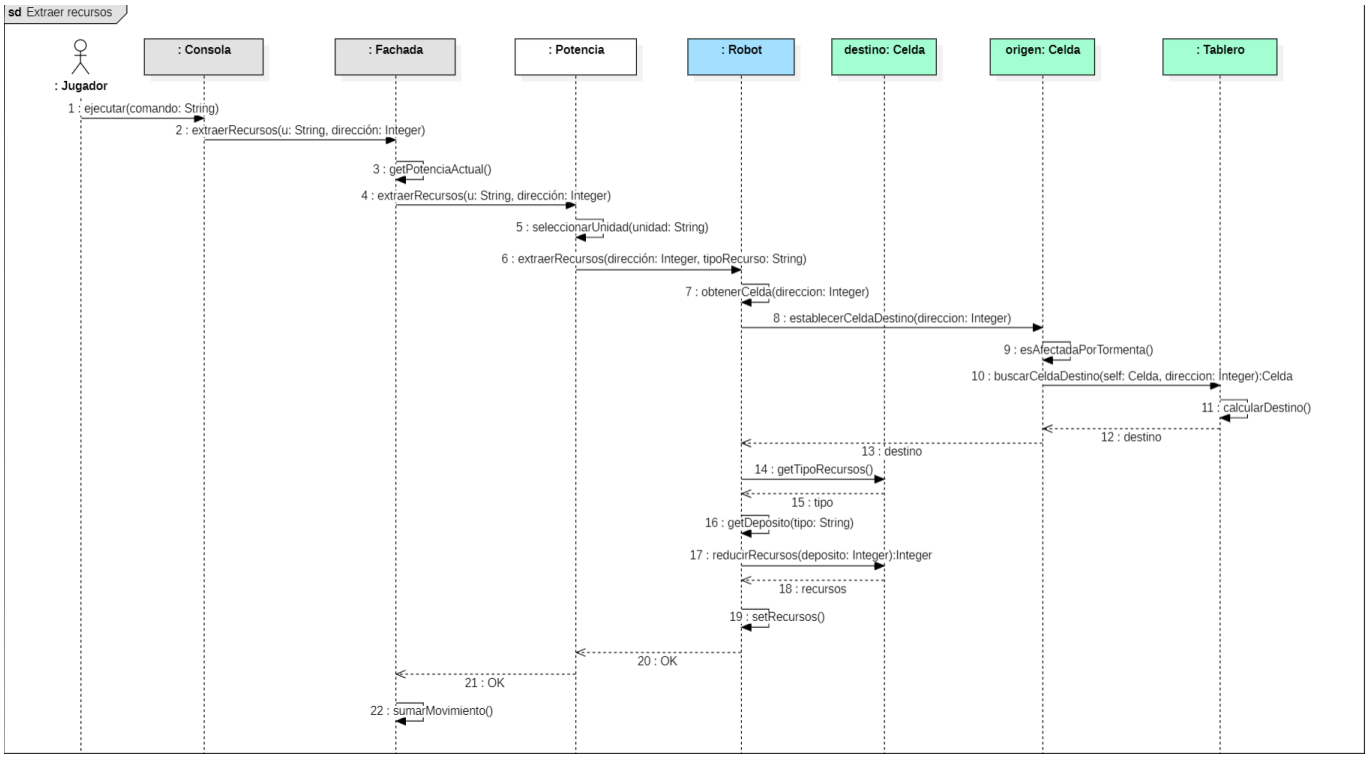

Figura 4.16: Diagrama de secuencia de Extraer recursos (Iteración 2)

4.3.3.5. Extraer recursos en grupo

Esta variante de extraer recursos tiene en cuenta a un grupo como unidad extractora. Cabe destacar que todas las unidades que la conformen deben ser del tipo Robot, como podemos ver en el diagrama. Tal y como aparece representado, al mandar la acción `extraerRecursos` al grupo de robots, esta la delegará a sus componentes y realizarán la acción por separado.

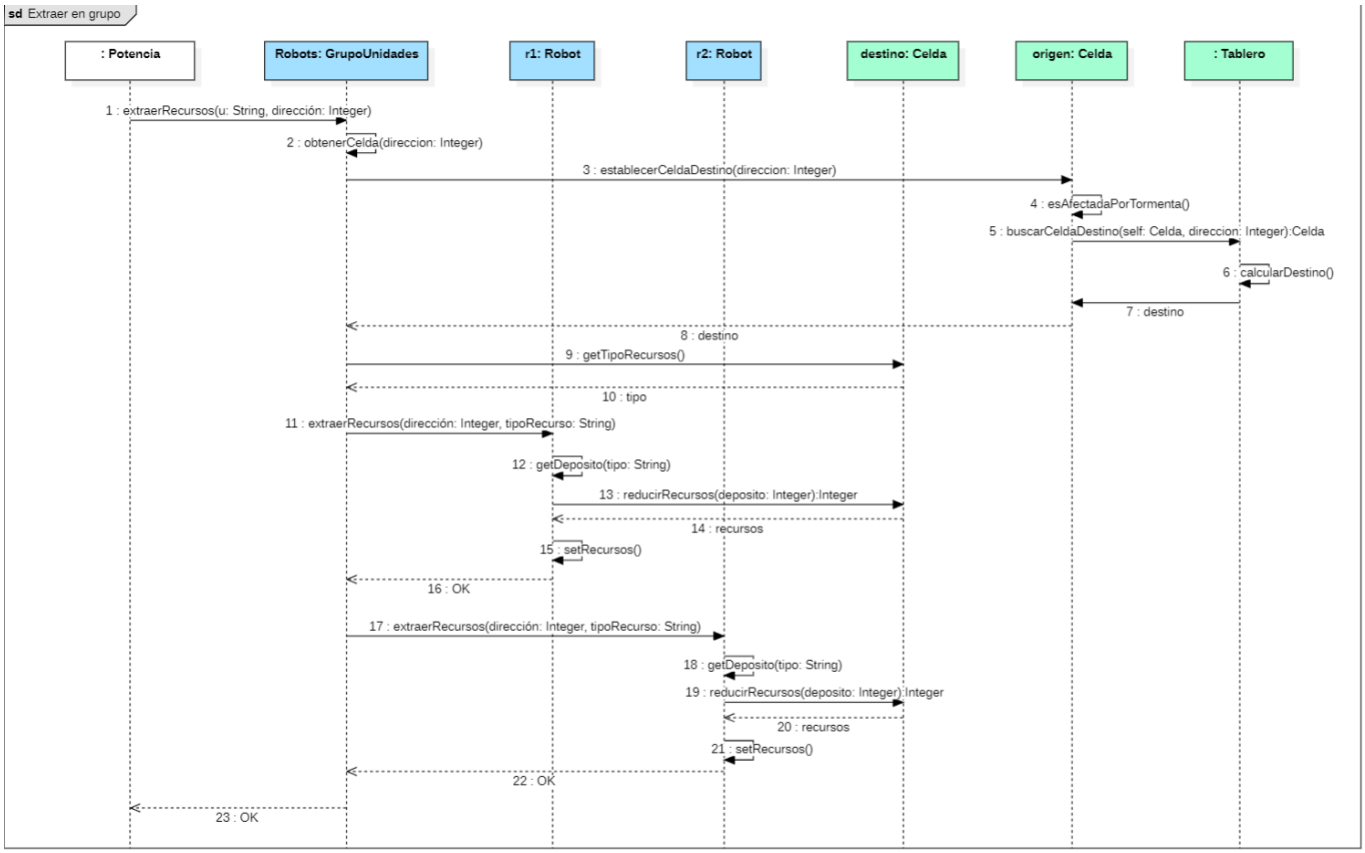

Figura 4.17: Diagrama de secuencia de Extraer recursos grupo (Iteración 2)

4.3.3.6. Entrar en construcción

Por último, mostraremos el diagrama de secuencia de **Entrar en construcción**, `extend` del caso de uso *Mover unidad*. En este diagrama, asumimos que la unidad entra en una celda en la que hay una construcción amiga. Por lo tanto, al añadir la unidad a esta celda, debemos también aumentar las capacidades de la construcción (teniendo en cuenta las estadísticas de la unidad, por ello la pasamos como parámetro hasta la construcción). Finalmente, si la unidad y la construcción son compatibles en tipo, la construcción reparará a la unidad, curándole toda la vida.

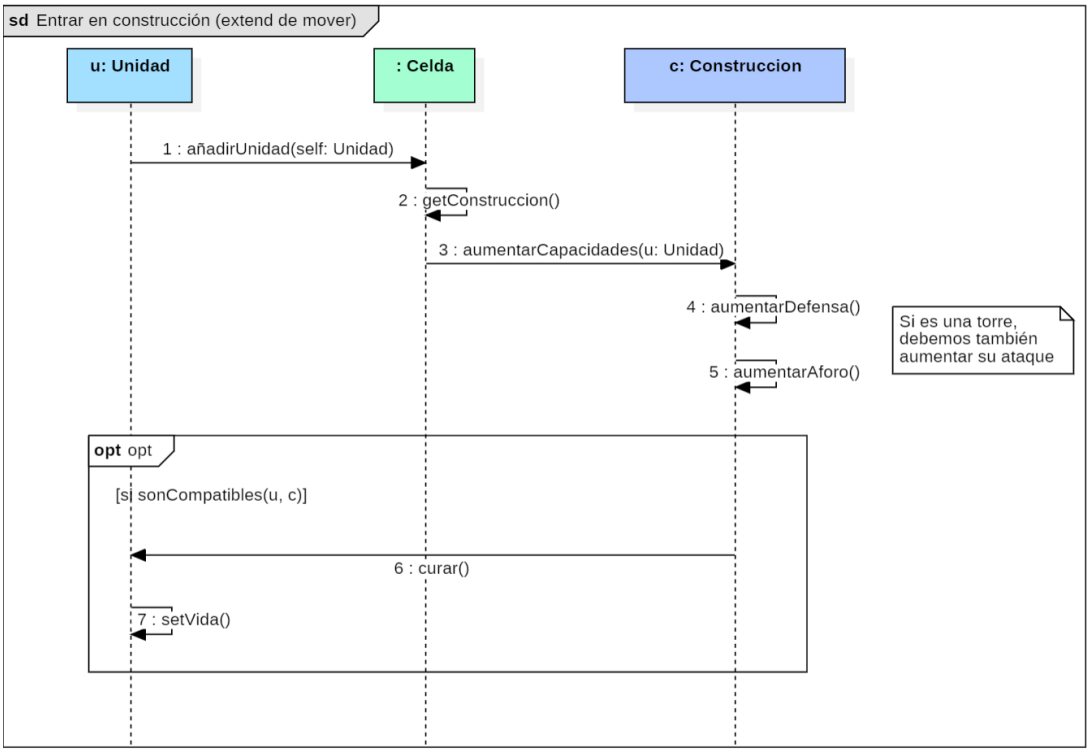

Figura 4.18: Diagrama de secuencia de Entrar en construcción (Iteración 2)

4.4. Iteración 3

4.4.1. Diagrama de clases

Finalmente, llegaremos al diagrama de clases definitivo: El diagrama de la iteración 3 (Figura **??**). Este es el diagrama estructural final que queda en nuestro trabajo en relación con el videojuego Mars Madness. Aparecen ya todos los patrones aplicados; junto con los anteriores, tenemos ahora el patrón DAO combinado con el Abstract Factory, el patrón Proxy y el patrón State. Comentaremos ahora brevemente dónde han sido colocados en el diagrama de clases estos patrones y, por lo tanto, qué clases nuevas han sido introducidas.

El patrón **DAO y Abstract Factory** se identifican con las clases de color rosa que podemos ver en la Figura 4.9. Debemos destacar que esta imagen no tiene una alta resolución, al igual que el diagrama de la iteración 2, por lo que se recomienda verlo mejor en el starUML entregado junto con esta memoria. Vemos que la clase DAOTablero está conectada 1 a 1 con la clase LoadedStrategy.

Por una parte, tenemos la interfaz DAOFactory y las clases que la implementan, las cuales representan el patrón Abstract Factory. Estas crean los DAOTablero, que son los encargados de instanciar las Celdas. Finalmente, el Tablero interactúa con el DAO correspondiente para cargar o guardar los mapas en el formato seleccionado.

La relación entre DAOTablero y LoadedStrategy se puede entender mejor en el contexto de cómo se maneja el tablero de juego. DAOTablero es responsable de la interacción con la base de datos para cargar y guardar datos del tablero. Una vez que los datos son cargados por DAOTablero, se utiliza LoadedStrategy para aplicar configuraciones específicas al tablero basado en los datos obtenidos.

En relación con el patrón **State**, observamos su integración con la clase Robot, ubicada justo debajo de las construcciones (azul oscuro). Se creó la interfaz IRobotState para gestionar los distintos estados del robot. A partir de esta interfaz, se derivan las clases NotDamagedState, DamagedState, y CriticalDamagedState. Cada una de estas clases maneja un estado específico, restringiendo las capacidades del robot a medida que su salud disminuye.

Finalmente, en relación con el patrón **Proxy**, hemos introducido las clases representadas en amarillo en la Figura 4.19. La interfaz IMapa se relaciona con Potencia y es implementada por la clase Tablero. Este patrón será útil para gestionar la visibilidad de las celdas para cada potencia, como explicaremos a continuación.

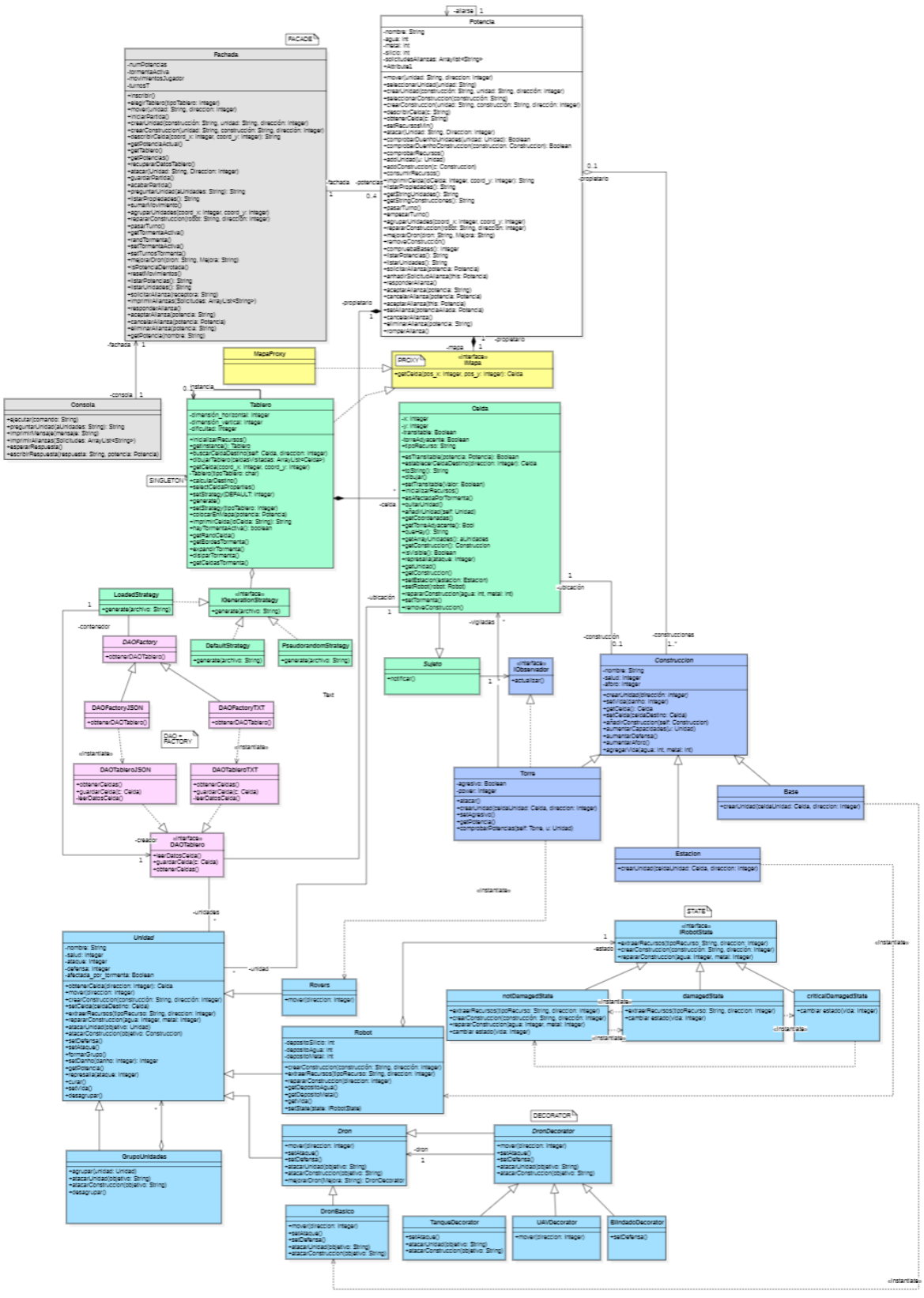

Figura 4.19: Diagrama de clases (Iteración 3)

4.4.2. Patrones de diseño

4.4.2.1. Proxy

En patrones de diseño, el Proxy es un patrón estructural que proporciona un sustituto o punto de acceso controlado a otro objeto. Este patrón permite controlar el acceso a un objeto, actuando como intermediario. Dentro de este contexto, hemos aplicado el patrón Proxy para optimizar el manejo de las celdas del tablero. En lugar de mantener un arreglo de tipo Celda que contenga todas las celdas del tablero, cada potencia ahora accede a un mapa específico que muestra únicamente las celdas relevantes para ella. Esta estructura mejora la eficiencia al reducir la cantidad de datos manipulados y garantiza que cada potencia solo tenga acceso a las celdas que le corresponden, lo que contribuye a un control más preciso y seguro del acceso.

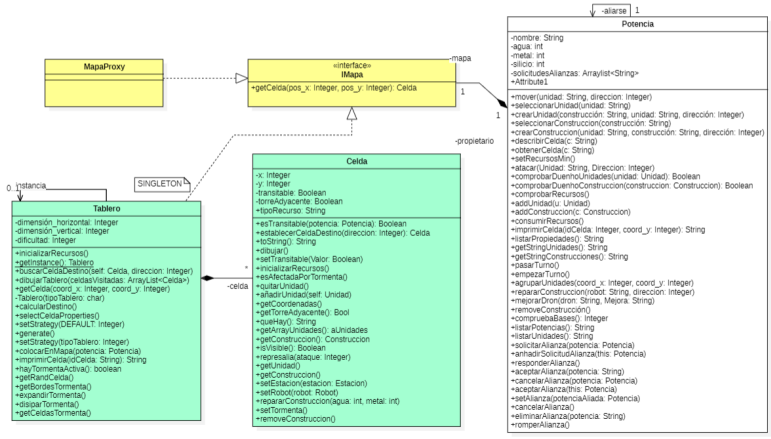

Figura 4.20: Patrón proxy para el tablero

4.4.2.2. State

El patrón State permite a un objeto cambiar su comportamiento según su estado interno, encapsulando los estados en objetos separados y delegando el comportamiento al objeto que representa su estado actual. En nuestra clase Robot, este patrón se aplica a estados como "no dañado" y "daño crítico", cada uno con acciones diferentes. Un robot dañado puede tener restricciones en sus acciones o no poder realizar ciertas tareas que un robot en buen estado sí puede. Así, el patrón State modela la variación de comportamiento en función del estado interno del objeto.

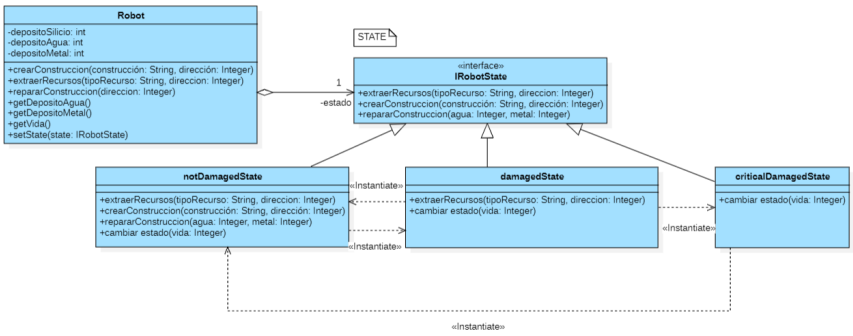

Figura 4.21: Patrón State

4.4.2.3. DAO + Abstract Factory

Combinar los patrones DAO (Data Access Object) y Abstract Factory en un diseño de software proporciona una solución robusta para gestionar el acceso a datos y la creación de objetos relacionados de una manera desacoplada y flexible. En nuestro caso, hemos decidido implementar este patrón a la hora de generar tableros cargados. Estos se pueden obtener desde ficheros txt y ficheros json, por lo que hay dos fábricas DAO que se encargan de gestionarlos. Podemos ver el uso de este patrón en la Figura 4.22.

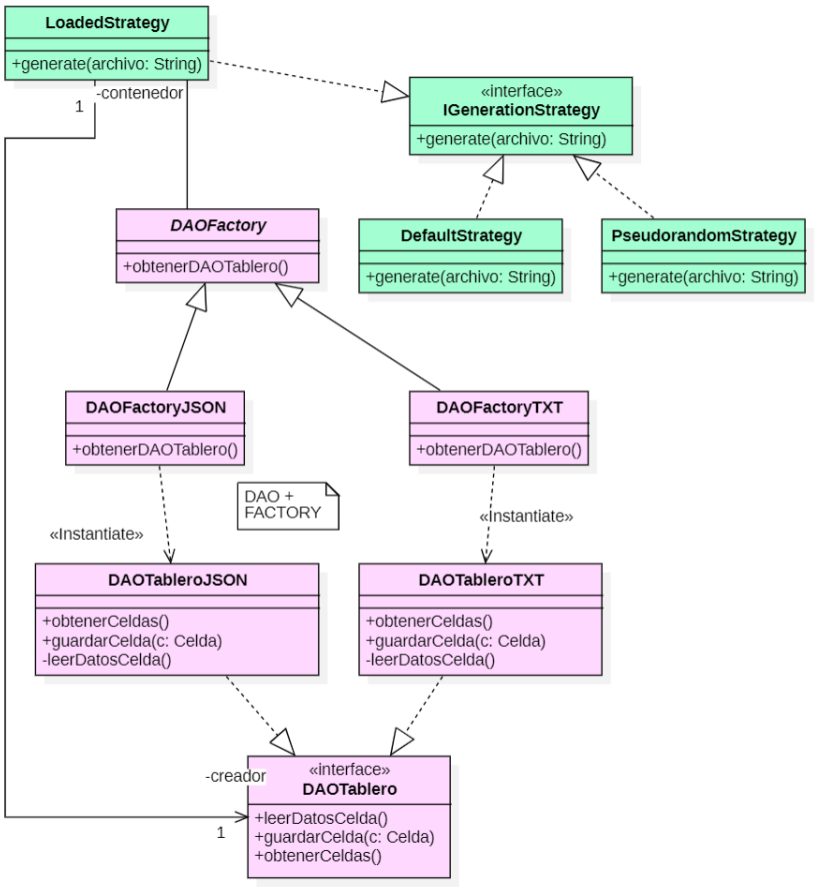

Figura 4.22: Patrón DAO + Abstract Factory

4.4.3. Casos de uso

4.4.3.1. Atacar

Atacar es un caso de uso con muchas variaciones, ya que puede tener muchas consecuencias. El diagrama del ataque básico, no merece demasiado la pena comentarlo mucho, por lo que se ha colocado en el Anexo 2 8.1. Primero, se busca la celda a la que se va a realizar el ataque, a partir de ahí hay 2 opciones, que haya unidades o una construcción. Para comprobar esto se usa el método `queHay()`. Si hay una construcción, se realiza el ataque sin más complicación, pero si hay unidades, es necesario preguntar al jugador cual de ellas quiere atacar.

Ahora sí, hay dos casos de atacar que son interesantes de comentar. El primero es cuando se vence a una potencia (Figura 4.23). Este sucede cuando se destruye la última de las bases de una potencia y sus consecuencias son la destrucción de todas las unidades de la potencia y que todas las construcciones de la potencia derrotada pasan a ser de la potencia atacante. Las unidades deben desaparecer el mapa, por lo que es necesario retirarlas de las casillas en las que se encuentren.

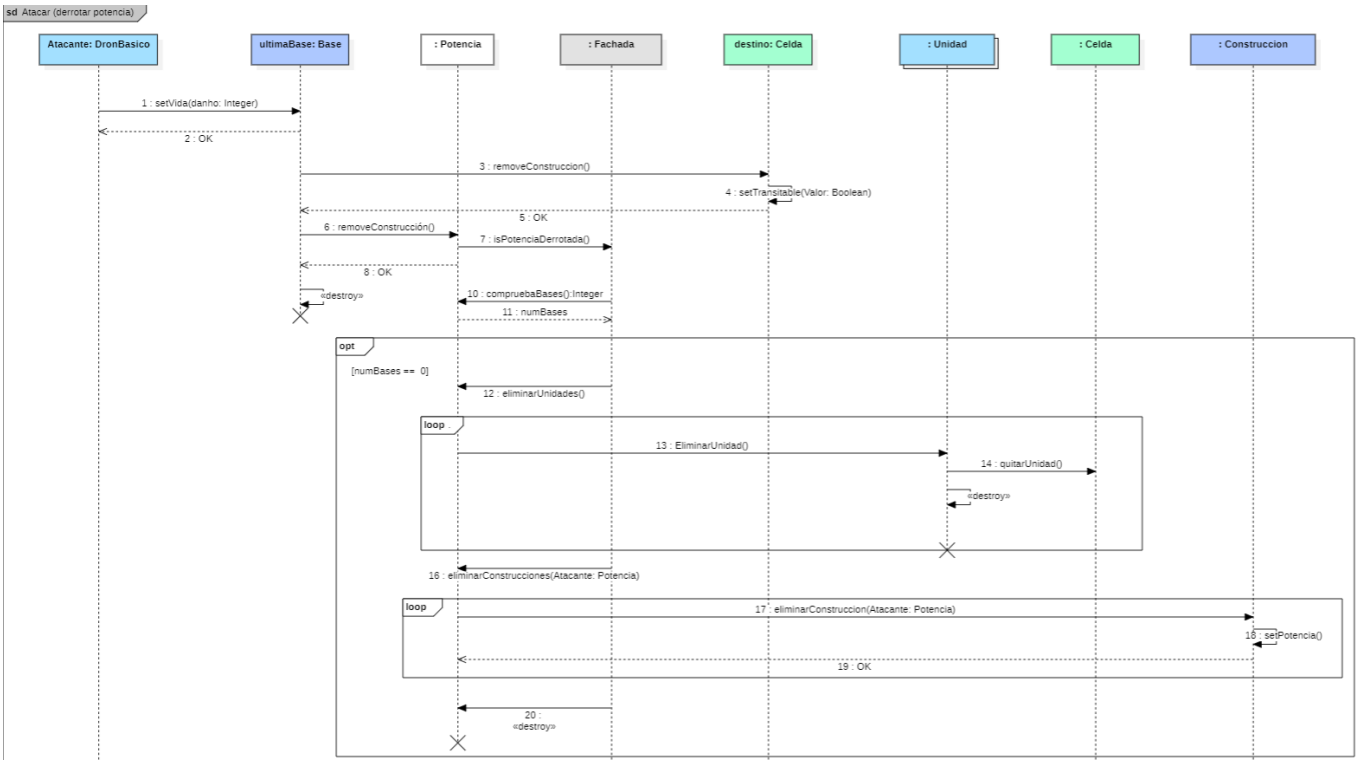

Figura 4.23: Diagrama de secuencia de Derrotar a una potencia (Iteración 3)

Otro caso interesante de atacar es cuando se ataca a un robot, dado que se puede apreciar la aplicación del patrón state. Un robot tiene 3 estados, los cuales se pueden instanciar entre ellos (aunque no son posibles todas las combinaciones, ver 4.21). Este cambio entre estados sucede cuando el robot pierde o gana vida y las consecuencias son la pérdida de las acciones que puede realizar. Con toda la vida puede hacer todo, es decir, extraer recursos, crear construcciones y repararlas. Sin embargo cuando baja de un 66 % de la vida, pierde la capacidad de crear y reparar. Este mismo es el caso que se muestra en el diagrama de secuencia 4.24. Se puede apreciar como la responsabilidad de cambiar de estado no la tiene el robot, sino que se delega al estado actual. Este crea el nuevo estado, lo setea y se autodestruye, provocando así un cambio de estado del robot. Ahora este solo podrá extraer recursos.

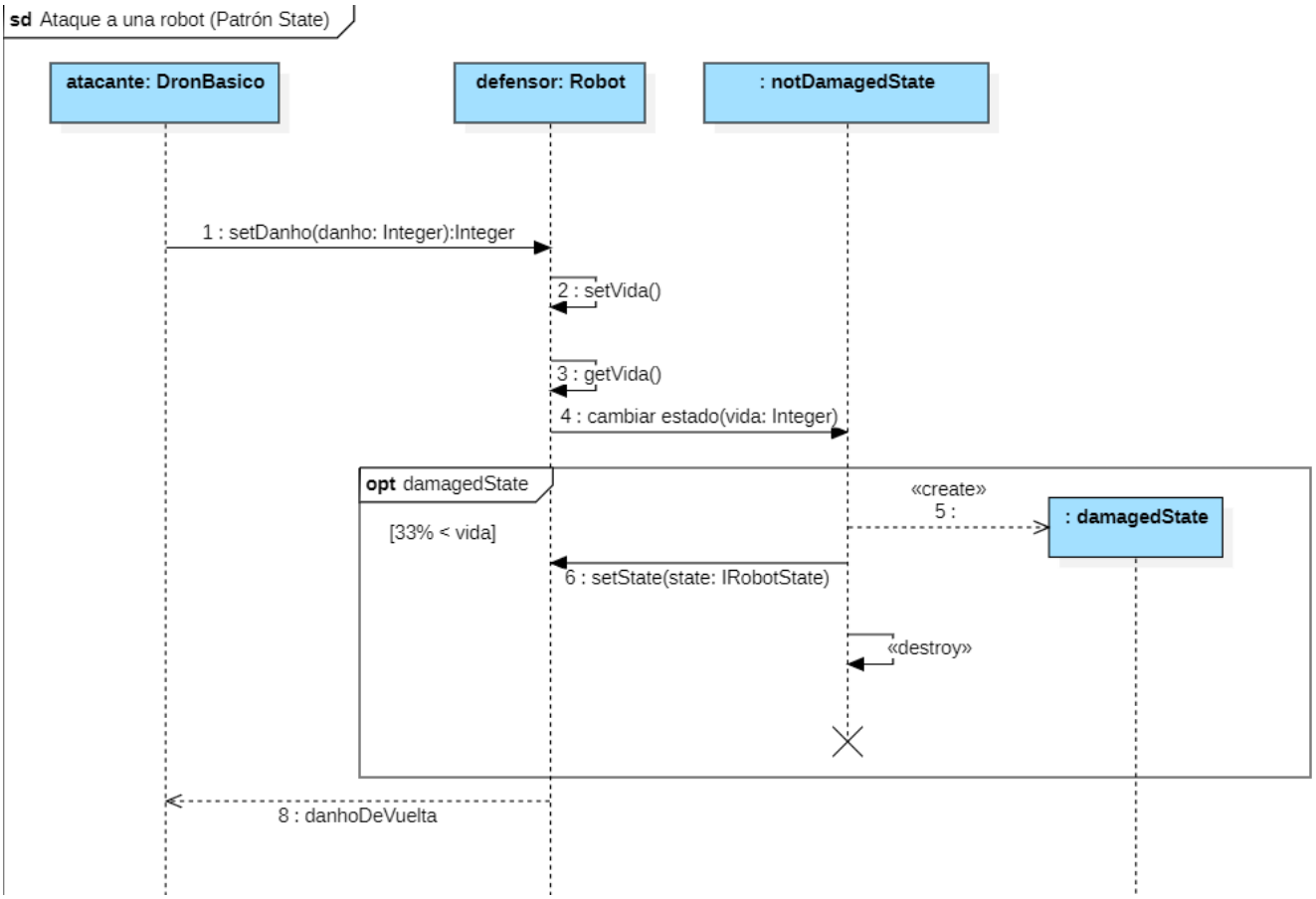

Figura 4.24: Diagrama de secuencia de Atacar a un robot (Iteración 3)

4.4.3.2. Ataque automático

Procedemos a describir un diagrama con bastante relevancia en el trabajo: el ataque automático que realiza una torre al detectar a un enemigo moviéndose a su alrededor. Aquí veremos la aplicación del patrón **Observer** que mencionamos en la iteración 2 (Figura 4.12). En este caso, cuando añadimos una unidad a una celda limítrofe a una torre, debemos hacer que la celda notifique y actualice el estado de la torre.

Hemos decidido emular el caso en el que al lado de una misma celda colindan dos torres: una enemiga (su potencia no coincide con la de la unidad) y una amiga (sus potencias coinciden). Así, podemos ver en el diagrama de la Figura 4.25 cómo cada torre comprueba las potencias en los mensajes 6 y 13. La celda enemiga, al detectar la presencia de un intruso, lanza una represalia a la celda, y esta se encarga de repartir el daño entre sus integrantes. Por otra parte, la torre amiga, al ver que no hay peligro, simplemente devuelve un OK en el mensaje 16.

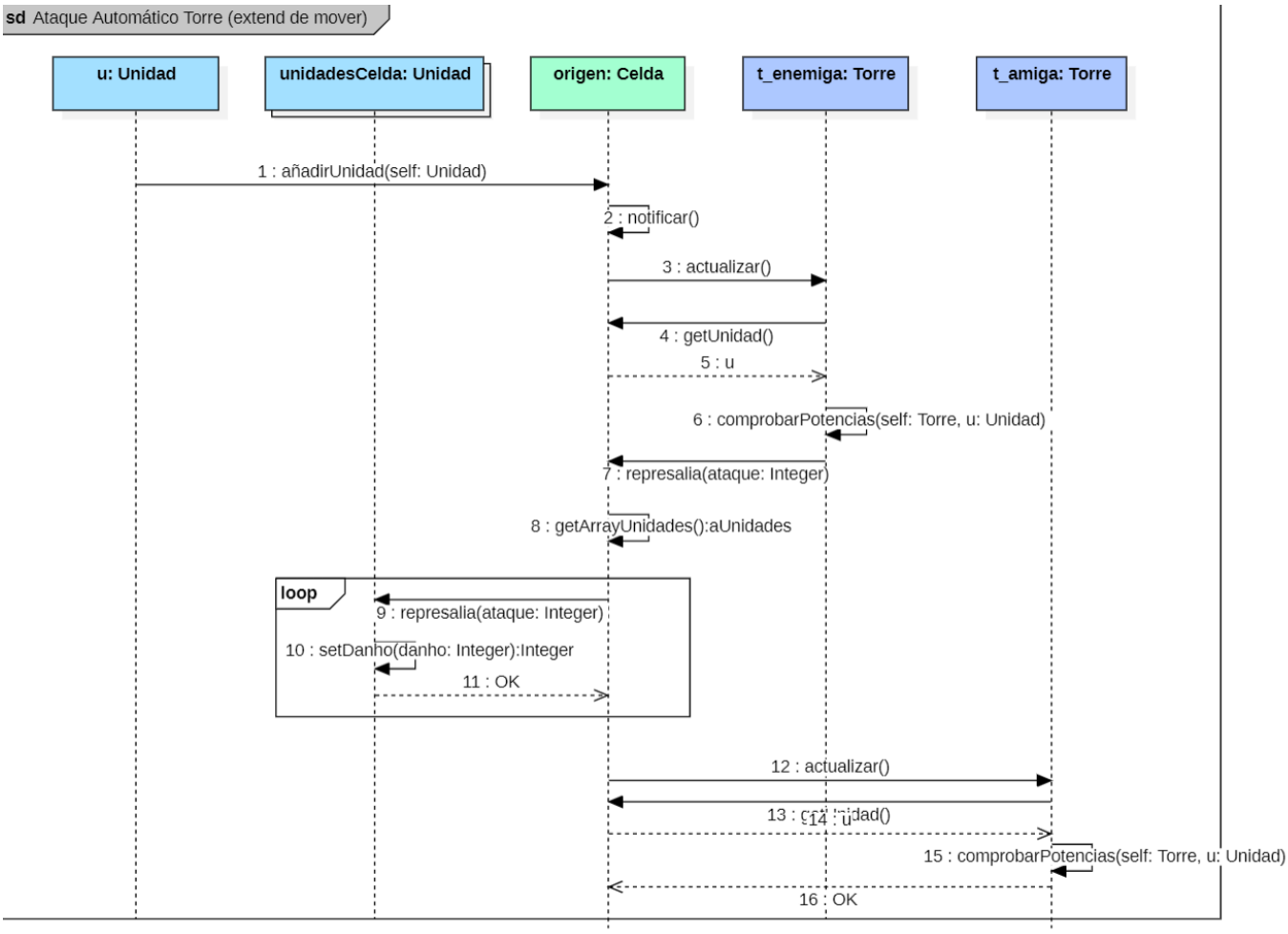

Figura 4.25: Diagrama de secuencia de ataque automático (Iteración 3)

4.4.3.3. Elegir tipo tablero (Loaded)

En este diagrama, podemos ver la aplicación del patrón **DAO combinado con el Abstract Factory**. Al igual que el anterior, este diagrama es bastante destacable al mostrar la aplicación de estos patrones junto con el Strategy, que ya explicamos en la iteración 2 con el caso del tablero por defecto (Figura 4.15). Como podemos observar en la Figura 4.26, cuando se llama con generate(archivo: String) a la clase LoadedStrategy con el fin de delegarle la creación de un tablero de juego, esta debe obtener el DAO para obtener los datos del tablero a través de un fichero (mensaje 5). El Dao se encarga de leer el archivo, procesando cada una de las celdas y creándolas en función de la información leída. Posteriormente se devuelven hasta el tablero, como podemos ver en los mensajes 9 y 10.

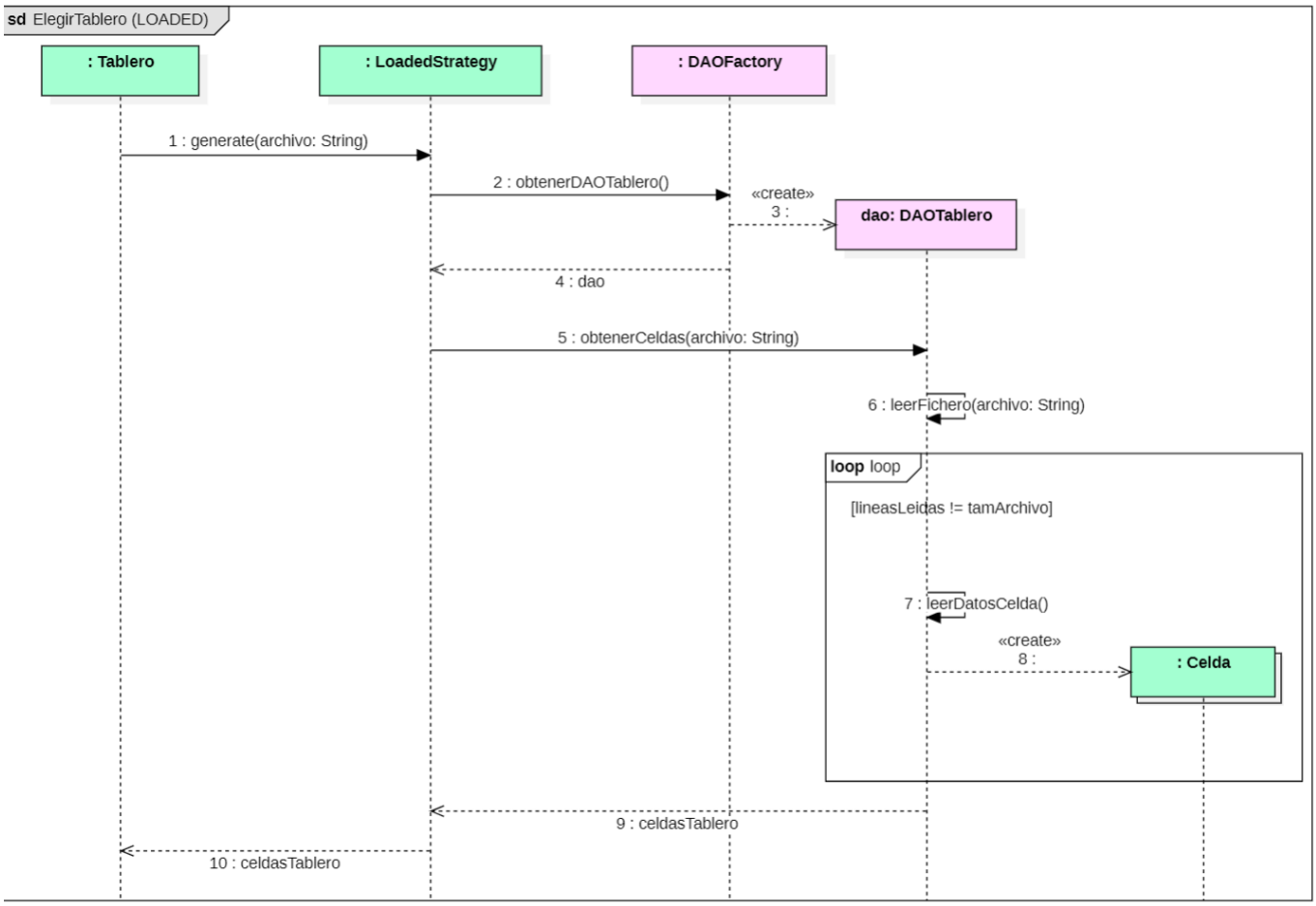

Figura 4.26: Diagrama de secuencia de Elegir tipo Tablero (LOADED) (Iteración 3)

4.4.3.4. Describir características propiedades

Por último, describiremos el último diagrama de secuencia en el que podemos ver que se utiliza el patrón **Proxy** en cierta manera: Describir características propiedades. El Usuario pide por terminal imprimir todas sus propiedades, y para ello, debemos hacer un toString de todas las construcciones, unidades y celdas que puede visualizar. A la hora de imprimir las celdas, vemos en esta iteración que se delega el trabajo a IMapa. Esta interfaz tiene conocimiento de todas las celdas que puede visualizar la potencia. Como dijimos al explicar el patrón Proxy (Figura 4.20), esto resulta en una manera más eficiente de manejar la información.

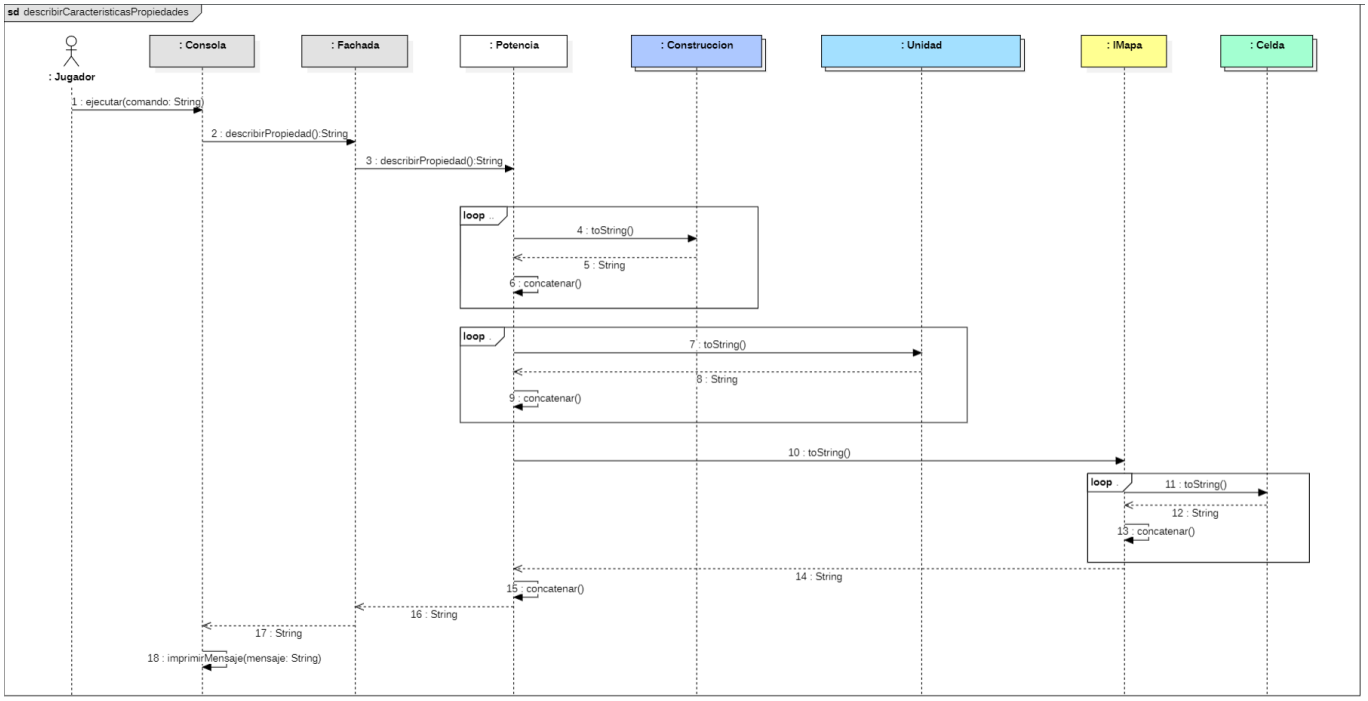

Figura 4.27: Diagrama de secuencia de describir características propiedades (Iteración 3)

Tarjetas CRC

En este capítulo, exploraremos en detalle las tarjetas CRC (Clase-Responsabilidad-Colaboración). Estas tarjetas son esenciales para entender y organizar las clases en el diseño orientado a objetos. En cada tarjeta, se detallará el nombre de la clase, sus responsabilidades y los colaboradores con los que interactúa, basándonos en el diagrama de clases de la iteración 3 del proyecto.

Cabe destacar que este ejercicio se realiza después de completar el trabajo. Al reflexionar sobre lo que ya hemos hecho, podemos identificar con mayor claridad las relaciones y responsabilidades de cada clase. Este enfoque nos da una visión más completa y precisa, ayudándonos a mejorar la creación de las tarjetas CRC y asegurando un diseño más robusto y coherente. Esta retroalimentación es muy útil para afinar y perfeccionar el diseño del sistema, facilitando futuras iteraciones y mejoras.

Consola	
Responsabilidades	**Colaboraciones**
Procesa el texto que se introduce por la terminal para poder ejecutar los comandos.	Fachada

Fachada	
Responsabilidades	**Colaboraciones**
Gestiona todas las acciones que desea realizar el jugador y las redirige.	Potencia, Consola

Tablero	
Responsabilidades	**Colaboraciones**
Almacena todas las celdas disponibles en el juego con toda su información	Celda

IGenerationStrategy	
Responsabilidades	**Colaboraciones**
Gestiona el modo de tablero que se genera	Tablero, LoadedStrategy, DefaultStrategy, PseudorandomStrategy

LoadedStrategy	
Responsabilidades	**Colaboraciones**
Carga un tipo de tablero guardado en un archivo	DAOTablero, DAOFactory
Es un tipo de generación de tablero	IGenerationStrategy

DefaultStrategy	
Responsabilidades	**Colaboraciones**
Carga un tipo de tablero por defecto	
Es un tipo de generación de tablero	IGenerationStrategy

PseudorandomStrategy	
Responsabilidades	**Colaboraciones**
Carga un tipo de tablero pseudoaleatorio	
Es un tipo de generación de tablero	IGenerationStrategy

IMapa	
Responsabilidades	**Colaboraciones**
Almacena las referencias a las celdas del tablero que puede visualizar una potencia	Tablero, MapaProxy, Potencia

MapaProxy	
Responsabilidades	**Colaboraciones**

DAOFactory	
Responsabilidades	**Colaboraciones**
Engloba las distintas fábricas concretas de tableros	DAOFactoryJSON, DAOFactoryTXT

DAOFactoryJSON	
Responsabilidades	**Colaboraciones**
Crea instancias de DAOTableroJSON	
Pertenece a la fábrica de tableros	DAOFactory

DAOFactoryTXT	
Responsabilidades	**Colaboraciones**
Crea instancias de DAOTableroTXT	
Pertenece a la fábrica de tableros	DAOFactory

DAOTableroJSON	
Responsabilidades	**Colaboraciones**
Carga celdas en el tablero desde un archivo JSON	DAOTablero

DAOTableroTXT	
Responsabilidades	**Colaboraciones**
Carga celdas en el tablero desde un archivo TXT	DAOTablero

DAOTablero	
Responsabilidades	**Colaboraciones**
Agrupa los diferentes tipos de DAOTablero que se permiten	DAOTableroJSON, DAOTableroTXT

Celda	
Responsabilidades	**Colaboraciones**
Lleva la cuenta de construcciones y unidades posicionadas	Construccion, Unidad
Permite la creación de construcciones y/o unidades	Construccion, Unidad
Recoge el tipo de recurso y la cantidad que contiene	
Permite a una unidad entrar a una construcción allí posicionada	Construccion
Forma parte del tablero de juego	Tablero

Potencia	
Responsabilidades	**Colaboraciones**
Almacena todos los datos y pertenencias del jugador	Construccion, Unidad
Visualizar las celdas a las que puede acceder	IMapa
Almacena las alianzas que puede hacer con otras potencias	

Unidad	
Responsabilidades	**Colaboraciones**
Entra en construcciones	Construcción
Se mueve por las celdas del tablero	Celda
Almacena las características generales de las unidades	

GrupoUnidades	
Responsabilidades	**Colaboraciones**
Recolectar recursos con robots, sumando las capacidades	Celda
Puede mover el grupo por las celdas del tablero	Celda
Puede atacar como grupo en una dirección	Celda

Rover	
Responsabilidades	**Colaboraciones**
Explora moviéndose por las celdas en diagonal, y puede ver todas las celdas aledañas	Celda
Puede curarse en una torre	Torre

Robot	
Responsabilidades	**Colaboraciones**
Extrae recursos de las celdas	Celda
Crea construcciones	Celda, Construccion
Repara construcciones	Celda, Construccion
Puede curarse en una estación	Estacion

IRobotState	
Responsabilidades	**Colaboraciones**
Indica en qué estado vital está el robot	Robot, criticalDamagedState, damagedState, notDamagedState

criticalDamagedState	
Responsabilidades	**Colaboraciones**
Gestiona el estado crítico del robot (<33 % vida), impidiéndole hacer sus acciones características	IRobotState

damagedState	
Responsabilidades	**Colaboraciones**
Gestiona el estado dañado del robot (entre el 66 y el 33 % de vida), impidiéndole crear y reparar construcciones	IRobotState

notDamagedState	
Responsabilidades	**Colaboraciones**
Gestiona el estado no dañado del robot (+66 % de vida), permitiéndole crear, reparar construcciones y extraer recursos	IRobotState

Dron	
Responsabilidades	**Colaboraciones**
Atacar a unidades enemigas	Celda, Unidad
Atacar a construcciones enemigas	Celda, Construccion
Mejorar sus capacidades	DronDecorator

DronBasico	
Responsabilidades	**Colaboraciones**
Instanciar tipos de drones básicos	Dron

DronDecorator	
Responsabilidades	**Colaboraciones**
Gestionar el acceso a los decoradores de los drones	Dron

TanqueDecorator	
Responsabilidades	**Colaboraciones**
Decorador que aumenta el alcance y el ataque del dron	DronDecorator

UAVDecorator	
Responsabilidades	**Colaboraciones**
Decorador que mejora la capacidad de movimiento	DronDecorator

BlindadoDecorator	
Responsabilidades	**Colaboraciones**
Decorador que aumenta la defensa del dron	DronDecorator

Construccion	
Responsabilidades	**Colaboraciones**
Cobija a las unidades de una potencia	Unidad, Potencia
Cura a las unidades que contiene	Unidad, Potencia
Crea nuevas unidades para la potencia	Unidad, Potencia

Torre	
Responsabilidades	**Colaboraciones**
Crear rovers	Rover
Curar rovers	Rover
Atacar automáticamente a las celdas vecinas ante la presencia de un movimiento o agrupación	Celda, IObservador

IObservador	
Responsabilidades	**Colaboraciones**
Manda a la torre actualizarse tras un cambio en la celda	Torre, Sujeto

Sujeto	
Responsabilidades	**Colaboraciones**
Notificar tras un cambio producido en las celdas	Celda, IObservador

Estacion	
Responsabilidades	**Colaboraciones**
Crear robots	Robot
Curar robots	Robot

Base	
Responsabilidades	**Colaboraciones**
Crear drones	Dron
Curar drones	Dron

CAPÍTULO FINAL. CONCLUSIÓN

Este proyecto de diseño de software ha sido una experiencia enriquecedora que nos ha permitido aplicar los conocimientos adquiridos en la materia durante los últimos tres meses. Trabajar con diagramas de casos de uso, vocabulario, clases y secuencia nos ha dado una visión completa del proceso de diseño, desde la conceptualización hasta la implementación.

A lo largo de este proyecto, hemos aprendido a trabajar en equipo de manera constante, aunque hubiese que dedicarle un poco más de tiempo al final para poder perfeccionar y afinar detalles. Debemos de reconocer que la falta de colaboración de starUML nos ha dificultado un poco el modo de trabajar, ya que no podíamos estar todos a la vez modificando un mismo proyecto. Esto hizo que invirtiésemos más tiempo del esperado en algunos puntos del proyecto.

A pesar de los obstáculos, consideramos que este trabajo ha sido una experiencia didáctica interesante y entretenida. Nos ha dejado con la satisfacción de haber completado un trabajo desafiante y nos ha abierto la puerta para continuar desarrollando y mejorando el proyecto en el futuro. Si bien nos quedamos cortos en tiempo para la implementación del software en Java, un lenguaje que dominamos en cierta medida, esto no descarta la posibilidad de llevar a cabo esta fase en un futuro próximo.

Podríamos decir que este proyecto ha sido un paso significativo en nuestro camino de aprendizaje, y nos ha preparado para enfrentar nuevos desafíos en el mundo del diseño de software y, en general, en el punto de la informática y de la gestión de proyectos.

ANEXO 1. Descripciones de los casos de uso

Hemos incorporado las descripciones de los casos de uso al final de este trabajo, con el fin de que no alterara demasiado la presentación incorporándolo al principio o en el medio de la memoria. En estas tablas se mostrará el nombre del caso, así como un breve resumen del mismo. Aparecen también las precondiciones y las postcondiciones, en las que se explican, respectivamente, las condiciones que deben cumplirse antes de iniciar el caso de uso y las condiciones que se cumplen una vez finalizado.

Además, se detalla el escenario principal, que describe los pasos secuenciales que componen el flujo normal de ejecución del caso de uso. Por último, se incluyen posibles escenarios alternativos que podrían ocurrir debido a variaciones o excepciones en el flujo principal. Estos escenarios alternativos proporcionan una visión completa y robusta del comportamiento del sistema en diferentes situaciones, asegurando así que todas las posibles eventualidades sean consideradas y adecuadamente manejadas. La prioridad de cada caso de uso también está indicada, lo que ayuda a entender la importancia relativa de cada funcionalidad en el contexto del sistema global.

Esta estructura permite una comprensión clara y detallada de cómo se espera que el sistema funcione, facilitando tanto su implementación como su mantenimiento y evolución futura. Cada caso de uso es esencial para definir el comportamiento del sistema desde la perspectiva del usuario, garantizando que todas las interacciones posibles sean tenidas en cuenta y documentadas de manera coherente y precisa.

Tabla 7.1: Caso de uso: Iniciar partida

SECCIÓN	COMENTARIO	
NOMBRE	Iniciar Partida	
RESUMEN	Recoge a los jugadores y da comienzo a la partida	
PRECONDICIONES	El número de jugadores llegue al mínimo (2) y que se haya elegido un tipo de tablero.	
POSTCONDICIONES	Se crea la partida	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Recoge los datos de los jugadores
	2	Recoge los datos del tablero
	3	Da comienzo a la partida
ESCENARIOS	*Paso*	*Acción*
ALTERNATIVOS	–	—
PRIORIDAD	Alta	

SECCIÓN	COMENTARIO	
NOMBRE	Elegir tipo tablero	
RESUMEN	Se escoge el tipo de tablero en el que se quiere jugar.	
PRECONDICIONES	Que no haya una partida empezada	
POSTCONDICIONES	Hay un tablero establecido para la futura partida	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Se abre menú de opciones de tablero
	2	Se escoge uno de los 3 tableros posibles: Tablero por defecto, tablero pseudoaleatorio o cargar tablero existente
	3	Se configura el tablero.
	4	Se genera el tablero.
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	Si se selecciona la opción de "Cargar tablero existente" y no hay un tablero almacenado en memoria, al jugador le aparecerá un mensaje: "No hay tableros cargados" y deberá escoger entre los otros 2 tableros posibles. Volver al paso 2.
	3	Si se selecciona la opción de pseudoaleatorio, el jugador podrá seleccionar entre varios tipos de mundos con diferentes características, con lo que el jugador podrá configurar su mundo a su gusto.
	3	Si se selecciona la opción de cargar tablero existente, hay que elegir el tablero a abrir.
	3	Si se selecciona la opción de tablero por defecto solo se configura la dificultad.
PRIORIDAD	Alta	

Tabla 7.2: Caso de uso: Dibujar tablero

SECCIÓN	COMENTARIO	
NOMBRE	Dibujar tablero	
RESUMEN	Se le muestra al jugador el tablero que sea visible para él en ese instante determinado	
PRECONDICIONES	Haber iniciado una partida	
POSTCONDICIONES	Se dibuja el tablero por terminal con la información correspondiente	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Ver tamaño del tablero
	2	Comprobar jugador que tiene el turno
	3	Dibujar las casillas a las que tiene acceso dicho jugador
	4	Dibujar los obstáculos
	5	Dibujar las construcciones y unidades no tripuladas presentes en la zona visible de dicho jugador
	6	Representar las celdas afectadas por la tormenta de arena
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	-	—
PRIORIDAD	Alta	

Tabla 7.3: Caso de uso: Mover unidad

SECCIÓN	COMENTARIO	
NOMBRE	Mover unidad	
RESUMEN	La unidad seleccionada por la potencia que tiene el turno se desplaza una casilla en la dirección establecida	
PRECONDICIONES	Poseer una unidad	
POSTCONDICIONES	La unidad se encuentra en una nueva posición	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Seleccionar unidad
	2	Seleccionar dirección/casilla
	3	La unidad se desplaza a la posición seleccionada
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	1	Si la unidad está afectada por una tormenta de arena en curso, se cancela la acción.
	2	Si en la posición seleccionada hay una construcción aliada, se deriva al caso de uso "Entrar en construcción"
	2	Si en la posición seleccionada está fuera del mapa, no se podrá desplazar a esa casilla.
	2	Si en la posición seleccionada hay una celda de recursos, no se podrá desplazar a esa casilla.
	2	Si en la posición seleccionada hay un obstáculo (río, pozo, montaña), no se podrá desplazar a esa casilla.
	2	Si la posición seleccionada está ocupada por una unidad o una construcción de otra potencia, no podrá desplazarse a esa casilla.
PRIORIDAD	Alta	

Tabla 7.4: Caso de uso: Crear unidad

SECCIÓN	**COMENTARIO**	
NOMBRE	Crear Unidad.	
RESUMEN	Una construcción del jugador crea una unidad no tripulada sobre el tablero de juego en una celda adyacente.	
PRECONDICIONES	Tener los recursos mínimos para crear la unidad menos costosa	
POSTCONDICIONES	Se crea una unidad, que será una nueva pertenencia de la potencia ejecutante	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Seleccionar construcción.
	2	Crea la unidad en una celda contigua.
ESCENARIOS ALTERNATIVOS	Paso	Acción
	2	Si no se disponen de los recursos necesarios, se cancela la acción.
	2	Si no se dispone de espacio en celdas vecinas, se cancela la acción.
PRIORIDAD	Alta	

Tabla 7.5: Caso de uso: Crear construcción

SECCIÓN	COMENTARIO	
NOMBRE	Crear construcción	
RESUMEN	Una unidad no tripulada crea una construcción en una celda del mapa	
PRECONDICIONES	Tener los recursos mínimos para crear la construcción menos costosa	
POSTCONDICIONES	Hay una nueva construcción creada en el mapa	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Seleccionar unidad
	2	Seleccionar dirección
	3	Se crea la unidad en la casilla adyacente seleccionada
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	3	Si la unidad seleccionada no es un robot, se cancela la acción.
	3	Si la casilla seleccionada está ocupada por otra construcción u otra unidad, no se puede crear.
	3	Si no se disponen de los recursos necesarios, se cancela la acción
PRIORIDAD	Alta	

Tabla 7.6: Caso de uso: Describir celda

SECCIÓN	COMENTARIO	
NOMBRE	Describir celda	
RESUMEN	Muestra la descripción de una celda visible y la descripción de lo que contenga (unidades, construcciones o materiales).	
PRECONDICIONES	Debe haber una partida iniciada	
POSTCONDICIONES	Se muestran los datos de la celda. Si hay construcciones (y sus características como daño o defensa), si hay unidades (y los nombres de cada una de las unidades que haya dentro), asi como los recursos que puedes encontrar si es que es una celda de recursos.	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Selecciona la celda a describir
	2	Muestra la información de esa celda
ESCENARIOS ALTERNATIVOS	2	Si la celda no es visible solo dice que no puede dar información.
PRIORIDAD	Alta	

Tabla 7.7: Caso de uso: Pasar turno

SECCIÓN	COMENTARIO	
NOMBRE	Pasar turno	
RESUMEN	Se pasa al siguiente jugador que tiene el turno en la partida.	
PRECONDICIONES	Que haya una partida activa	
POSTCONDICIONES	Se prosigue en la partida con el siguiente jugador.	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	El jugador selecciona la opción "Pasar turno" por terminal
	2	Se pasa el turno al siguiente jugador
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	1	Si se decide de manera aleatoria que debe comenzar una tormenta, se deriva al caso de uso "Generar tormenta"
	1	Si hay una tormenta activa, se deriva al caso de uso "Expandir tormenta"
PRIORIDAD	Alta	

SECCIÓN	COMENTARIO	
NOMBRE	Extraer recursos	
RESUMEN	Un robot extrae recursos que se encuentran en una celda.	
PRECONDICIONES	Disponer mínimo de un robot y una celda de recursos visible.	
POSTCONDICIONES	Se obtienen los recursos de la celda.	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Seleccionar unidad
	2	Selecciona una celda junto a un robot.
	3	Se indica que se desea extraer recursos de esa celda
	4	El robot comienza a extraer los recursos de esa celda.
	5	El robot completa la extracción de recursos de la celda, haciendo que aquellos recursos que haya obtenido sean eliminados de la celda de manera permanente.
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	3	Si la unidad seleccionada no es un robot. Se cancela la acción
	3	La celda adyacente al robot no contiene recursos. Se cancela la acción
	4	La celda no dispone de más recursos. Se cancela la acción.
	4	El robot con el que se pretende extraer recursos se encuentra en su límite. Se cancela la acción
	4	Si, además, el robot extrae la totalidad de recursos disponibles en esa celda la celda se convierte en una celda transitable ahora.
PRIORIDAD	Media	

SECCIÓN	COMENTARIO	
NOMBRE	Entrar en construcción	
RESUMEN	La unidad ingresa dentro una construcción aliada	
PRECONDICIONES	Poseer de una unidad	
POSTCONDICIONES	Se aumenta la capacidad ofensiva y defensiva de la construcción. Además, las unidades en el interior de la construcción (si son drones) podrán atacar a cualquier enemigo que se encuentre en una celda vecina.	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Seleccionar unidad
	2	Moverse dentro de la construcción.
	3	Se aumenta la capacidad ofensiva y defensiva de la construcción.
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	3	Si la construcción es una estación y la unidad es un robot con recursos almacenados, se transfieren automáticamente los recursos de la unidad a la potencia propietaria.
	3	Si el tipo de la unidad y el de la construcción son compatibles (es decir, al entrar un robot en una estación, un dron en una base o un rover en una torre), la unidad recupera toda su salud.
PRIORIDAD	Alta	

SECCIÓN	COMENTARIO	
NOMBRE	Reparar construcción	
RESUMEN	Una construcción es reparada por al menos un robot recuperando puntos de vida a cambio del consumo de materiales.	
PRECONDICIONES	Disponer de un robot y unos recursos mínimos de agua y metal.	
POSTCONDICIONES	La construcción recupera puntos de vida	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Seleccionar unidad
	2	Selecciona una construcción que se encuentre en una celda adyacente a la unidad.
	3	Los recursos necesarios para arreglar la construcción son eliminados de la potencia.
	4	Aumentan los puntos de salud de la construcción. Se vuelve al paso 3 y se siguen recuperando puntos de vida hasta que se llegue al máximo.
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	Si la unidad no es un robot, se cancela la acción.
	2	Si la construcción está intacta, se cancela la acción.
	3	La potencia no cuenta con los recursos necesarios para arreglar esa construcción. Se cancela la acción.
	4	Los recursos disponibles no llegan para que se recupere toda la salud de la construcción.Se interrumpe la acción.
PRIORIDAD	Media	

SECCIÓN	COMENTARIO	
NOMBRE	Describir características propiedades	
RESUMEN	Un jugador pide ver las características de una de sus propiedades (unidad o construcción)	
PRECONDICIONES	Que haya una partida en curso	
POSTCONDICIONES	Se muestran las características de las construcciones, unidades y nombre de las celdas únicamente visibles para la potencia solicitante. En el caso de la unidad, se mostrará concretamente su ataque y defensa. En el caso de la construcción, se mostrará su aforo y una array de las unidades que estén en ella en el momento. Para ambas pertenencias, se mostrará también su nombre, tipo, salud, y celda en la que está ubicada.	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Imprimir la información por pantalla
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	-	—
PRIORIDAD	Media	

SECCIÓN	COMENTARIO	
NOMBRE	Agrupar	
RESUMEN	Se juntan varias unidades en un grupo	
PRECONDICIONES	Disponer mínimo de dos unidades no tripulada	
POSTCONDICIONES	Las unidades que se han agrupado ahora forman una sola unidad agrupada	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Se selecciona una celda donde se desea agrupar unidades.
	2	Se agrupan las unidades que están en esa celda.
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	1	Esa celda no es visible. Se cancela la acción. Se vuelve al paso 1.
	1	Esa celda no contiene 2 o más unidades que te pertenezcan. Se cancela la acción. se vuelve al paso 1.
	1	En la celda hay más de dos unidades que se pueden agrupar. Da opción a seleccionar las unidades que se desean agrupar..
	2	Si todas las unidades que se han agrupado son de tipo robot la unidad agrupada adquiere las características de un robot, es decir, puede reparar y extraer recursos. Su capacidad de extracción vendrá dada por la suma de las capacidades de sus integrantes
PRIORIDAD	Media	

SECCIÓN	COMENTARIO	
NOMBRE	Desagrupar	
RESUMEN	Las unidades agrupadas dejan de formar un único grupo	
PRECONDICIONES	Disponer mínimo de alguna agrupación de unidades no tripuladas.	
POSTCONDICIONES	Las unidades que estaban agrupadas ahora están separadas	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Se selecciona una celda donde se desea desagrupar las unidades.
	2	Las unidades que se encontraban en el grupo ahora se mueven de manera individual.
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	1	Esa celda no es visible Se cancela la acción. Vuelve al paso 1.
	1	Esa celda no contiene alguna unidad agrupada que pertenezca a la potencia que dispone del turno actual. Se cancela la acción. Se vuelve al paso 1.
	1	La celda que has seleccionado tiene dos grupos de unidades. Selecciona uno.
PRIORIDAD	Media	

SECCIÓN	COMENTARIO	
NOMBRE	Mejorar Dron	
RESUMEN	Los drones pueden mejorar sus capacidades defensivas, ofensivas y de movilidad.	
PRECONDICIONES	Tener un dron	
POSTCONDICIONES	Queda un dron mejorado con el	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	La potencia selecciona el dron a mejorar
	2	Se mejoran sus capacidades. Hay 4 niveles, el básico, el UAV (mejora la movilidad), el tanque (mejora el ataque) y el blindado (mejora la defensa)
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	Ya está a nivel máximo, así que no se mejora
PRIORIDAD	Media	

SECCIÓN	COMENTARIO	
NOMBRE	Atacar	
RESUMEN	Un dron ataca a la construcción/unidad de una potencia rival	
PRECONDICIONES	Disponer mínimo de un dron	
POSTCONDICIONES	Se reduce la vida del enemigo en una cantidad determinada	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Seleccionar una unidad y se indica la dirección en la debe atacar
	2	El dron comprueba que unidades o construcción hay en la dirección en la que desea atacar y, en el caso de haber unidades, selecciona un único objetivo para realizar el ataque.
	3	El dron ejecuta un ataque contra el objetivo seleccionado.
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	La celda apuntada está vacía. Se cancela el ataque.
	2	La dirección en la se desea atacar contiene tropas o construcciones aliadas. Se cancela el ataque.
	3	Si la unidad es un grupo, todos los drones de ese grupo atacan
	2	Si la casilla desde la que se desea atacar está dentro de una tormenta no se puede fijar un objetivo y no se puede atacar.
	2	Si la casilla a la que se desea atacar está dentro de una tormenta no se puede fijar un objetivo y no se puede atacar.
	3	Si el ataque del dron destruye la última base de una potencia esta pierde y todas las construcciones restantes de la potencia pasarán a estar en control de la potencia que controlaba el dron.
	3	Si se ataca a una unidad agrupada y se eliminan a todas las unidades que la conformaban menos una, la unidad se desagrupa.
PRIORIDAD	Media	

SECCIÓN	COMENTARIO	
NOMBRE	Listar propiedades	
RESUMEN	Se mostrarán los nombres de todas las construcciones que pertenecen a la potencia junto con el tipo de construcción que son.	
PRECONDICIONES		
POSTCONDICIONES		
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Se selecciona de qué potencia se desea listar las propiedades
	2	Se escribe por pantalla las propiedades de la potencia seleccionada
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	Si la potencia que se desea listar ya ha perdido en vez de mostrarse sus estadísticas se indica que ya no forma parte de la partida
PRIORIDAD	Alta	

SECCIÓN	COMENTARIO	
NOMBRE	Listar Potencias	
RESUMEN	Se mostrarán las cantidades de todos los recursos, la cantidad de construcciones de cada tipo que tiene, la cantidad de unidades de cada tipo que tiene	
PRECONDICIONES	Que haya una partida iniciada	
POSTCONDICIONES	Se muestran las datos de la potencia	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Selecciona la potencia que desea listar
	2	Se escribe por pantalla los datos de la potencia seleccionada
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	Si la potencia que se desea listar ya ha perdido en vez de mostrarse sus estadísticas se indica que ya no forma parte de la partida
PRIORIDAD	Baja	

SECCIÓN	COMENTARIO	
NOMBRE	Generar tormenta	
RESUMEN	Se crea una tormenta de arena en el terreno de juego	
PRECONDICIONES	No hay una tormenta activa	
POSTCONDICIONES	Generación de una tormenta de arena en una celda aleatoria que puede afectar negativamente a las potencias	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Selección aleatoria de una celda
	2	Se genera la tormenta
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	1	Si en la celda seleccionada arbitrariamente hay unidades o construcciones, se cancela la acción. Volver al paso 1.
PRIORIDAD	Baja	

SECCIÓN	COMENTARIO	
NOMBRE	Expandir tormenta	
RESUMEN	Se decide la celda contigua en la que proseguirá la tormenta	
PRECONDICIONES	Que haya una tormenta activa	
POSTCONDICIONES	Aumenta en uno el camino de la tormenta	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Selección aleatoria de una celda contigua a alguna en la que esté activa la tormenta
	2	Se genera la tormenta en esa celda
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	1	Si en la celda seleccionada arbitrariamente hay unidades o construcciones, se cancela la acción. Volver al paso 1
PRIORIDAD	Baja	

SECCIÓN	COMENTARIO	
NOMBRE	Responder alianza	
RESUMEN	Una potencia decide responder ante una petición de alianza hecha por otra potencia. Si decide aceptarla, establece una alianza con esa potencia, lo que significa que ambos jugadores se comprometen a no atacarse entre sí y pueden colaborar estratégicamente para alcanzar la victoria en el juego. Por otro lado, si la potencia decide rechazar la solicitud de alianza, esta solicitud se eliminará de su lista de peticiones pendientes y no se establecerá ninguna alianza con la potencia solicitante.	
PRECONDICIONES	–	
POSTCONDICIONES	–	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	Al pasar de turno aparecen las solicitudes de alianza que otras potencias han propuesto a la potencia activa
	2	El usuario puede acceder al menú de alianzas y responder a ellas
	3	Las posibles respuestas son aceptar o rechazar
	4	Cuando vuelva el turno de la potencia solicitante, se le mostrará la respuesta de la potencia solicitada
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	La potencia activa ya tiene alianzas con todas las potencias menos con una, por lo que no puede aceptarla, y se considerará como rechazada en su turno
	2	La potencia que había propuesto la alianza ha perdido, por lo que se rechaza la alianza
PRIORIDAD	Baja	

SECCIÓN	COMENTARIO	
NOMBRE	Proponer alianza	
RESUMEN	Una potencia solicita una alianza a otra de las potencias activas en el juego, quien podrá aceptarla o rechazarla cuando llegue su turno.	
PRECONDICIONES	Debe haber más de 2 potencias activas en el juego	
POSTCONDICIONES	Se ha preparado un mensaje para la potencia a la que se solicita la alianza	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	La potencia con el turno indica la opción de proponer una alianza
	2	Elige con qué potencia quiere realizar la alianza
	3	Se realiza la solicitud
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	2	Ya tiene una alianza con esa potencia
	1	La potencia activa ya tiene alianza con todas las potencias del juego menos con una, por lo que no puede proponer más alianza
PRIORIDAD	Baja	

SECCIÓN	COMENTARIO	
NOMBRE	Eliminar alianza	
RESUMEN	Una potencia decide acabar con una de las alianzas que ha establecido anteriormente con otra de las potencias activas en el juego.	
PRECONDICIONES	–	
POSTCONDICIONES	–	
ESCENARIO PRINCIPAL	*Paso*	*Acción*
	1	La potencia activa entra al menú de sus alianzas y ver sus alianzas activas
	2	Selecciona una de las potencias con las que tiene una alianza
	3	Rompe esa alianza
	4	Cuando sea el turno de la potencia aparecerá un mensaje para mostrar que se ha roto al alianza
ESCENARIOS ALTERNATIVOS	*Paso*	*Acción*
	3	–
	3	–
PRIORIDAD	Baja	

ANEXO 2. Caso atacar

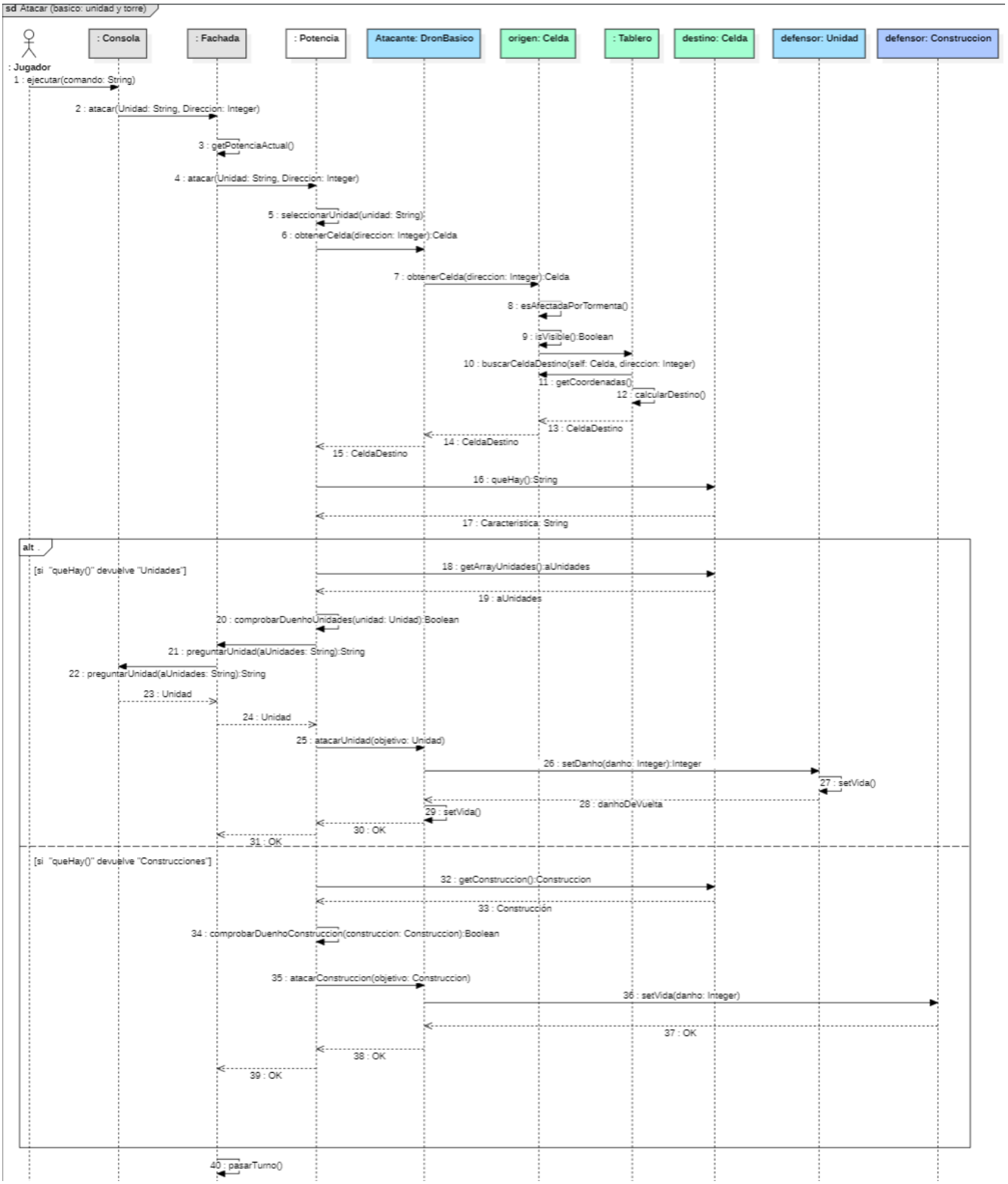

Figura 8.1: Diagrama de secuencia de Agrupar unidades (Iteración 2)