Algoritmos y Programación III
Trabajo Práctico 2

Fang Robinson
Frontera Federico
Martin Tomas
Wu Nicolás

6 de Junio del 2019

1 Introducción y supuestos

En el presente informe se detalla una posible solución al segundo trabajo práctico de la materia Algoritmos y Programación III de la Facultad de Ingeniería de la UBA que consiste en desarrollar una aplicacion similar al juego Minecraft utilizando un lenguaje de tipado estático (Java) con un diseño del modelo orientado a objetos, trabajando con técnicas de TDD e Integración Contínua.

A la hora de comenzar a desarrollar la aplicación, nos encontramos con algunas situaciones que no estaban definidas, por esto presentamos la siguiente lista con los supuestos:

- El jugador no puede caminar sobre los materiales.
- Al iniciar la aplicacion, el jugador siempre comienza con un hacha de madera equipada.
- El jugador no puede golpear materiales sin una herramienta equipada.
- El jugador tiene dos inventarios: uno para herramientas y otro para materiales.
- El jugador siempre golpea en la dirección en que está mirando.
- Una herramienta puede usarse siempre que su durabilidad sea positiva.
- Una vez que se rompe una herramienta se pierde para siempre.
- Al golpear un material la herramienta se desgasta independientemente del éxito del golpe. Por ejemplo, un hacha de madera golpeando un diamante nunca podrá romperlo, pero eventualmente sí se romperá el hacha.

- Las herramientas no se desgastan si golpean al "aire".
- Si, al momento de crear una herramienta, la disposición de los materiales es incorrecta no se construye nada y se le devuelven los materiales al jugador.

2 Diagramas de Clases

2.1 Diseño del Modelo

Diagrama de Clases: Modelo

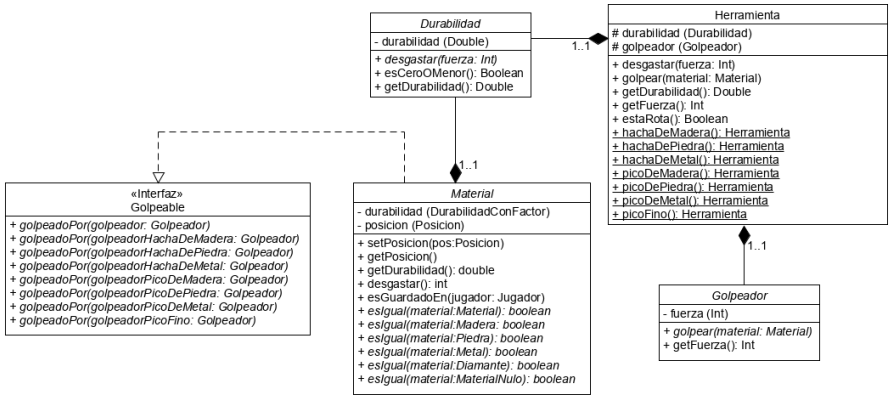

Figure 1: Digrama de Clases: Modelo

2.2 Modelado de Herramientas

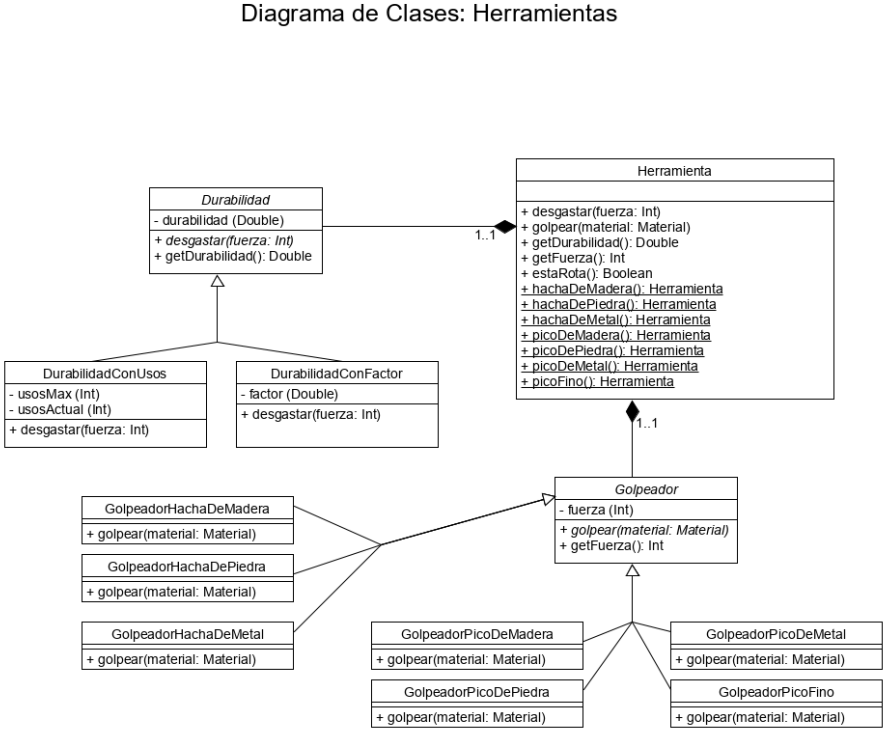

Figure 2: Digrama de Clases: Herramienta

2.3 Modelado de Jugador

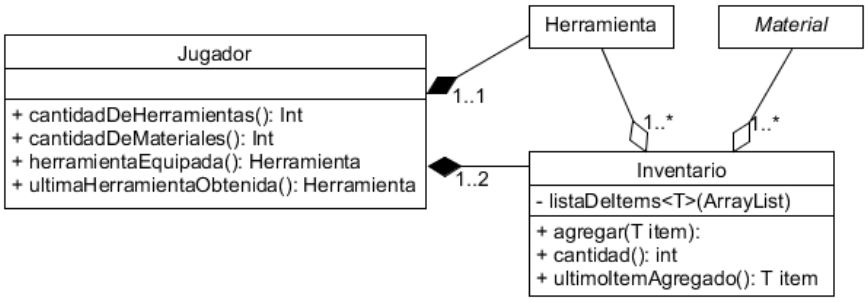

Figure 3: Digrama de Clases: Jugador e Inventario

2.4 Modelado de Durabilidad

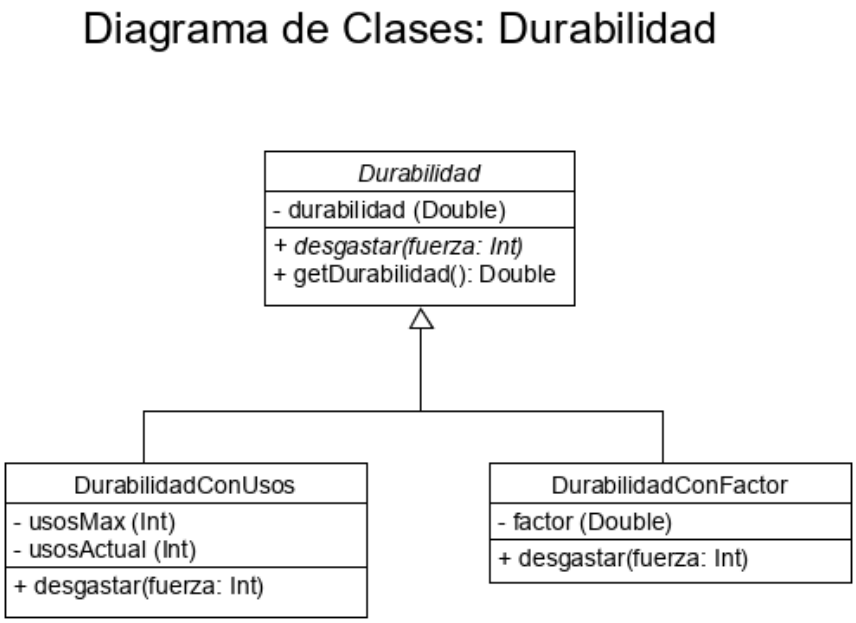

Figure 4: Digrama de Clases: Durabilidad

2.5 Modelado de Esquema

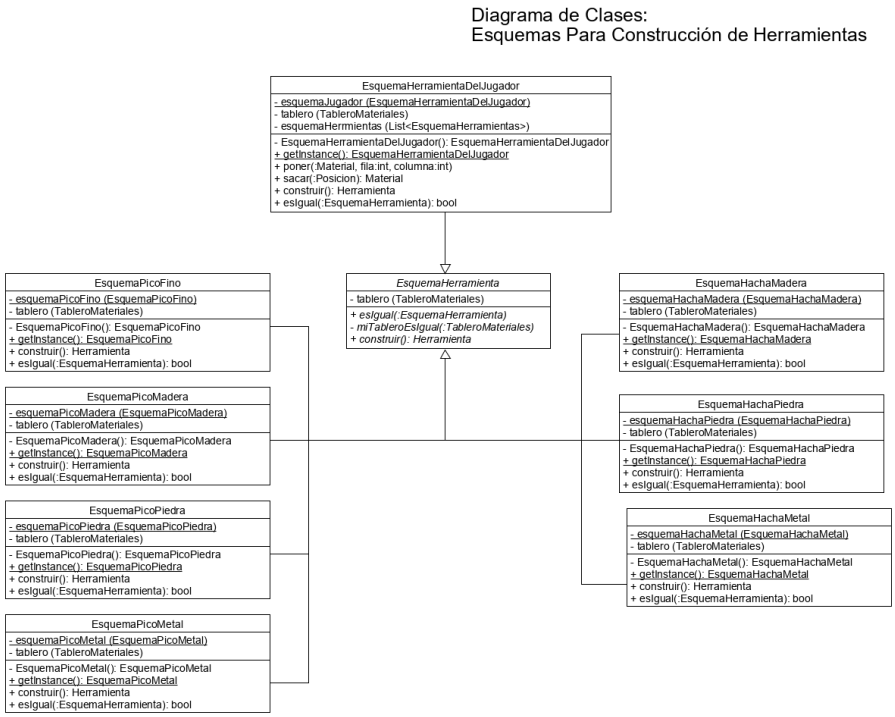

Figure 5: Una clase Esquema para cada herramienta que se pueda crear y una que representa lo que el jugador quiere construir

2.6 Modelado de Tablero

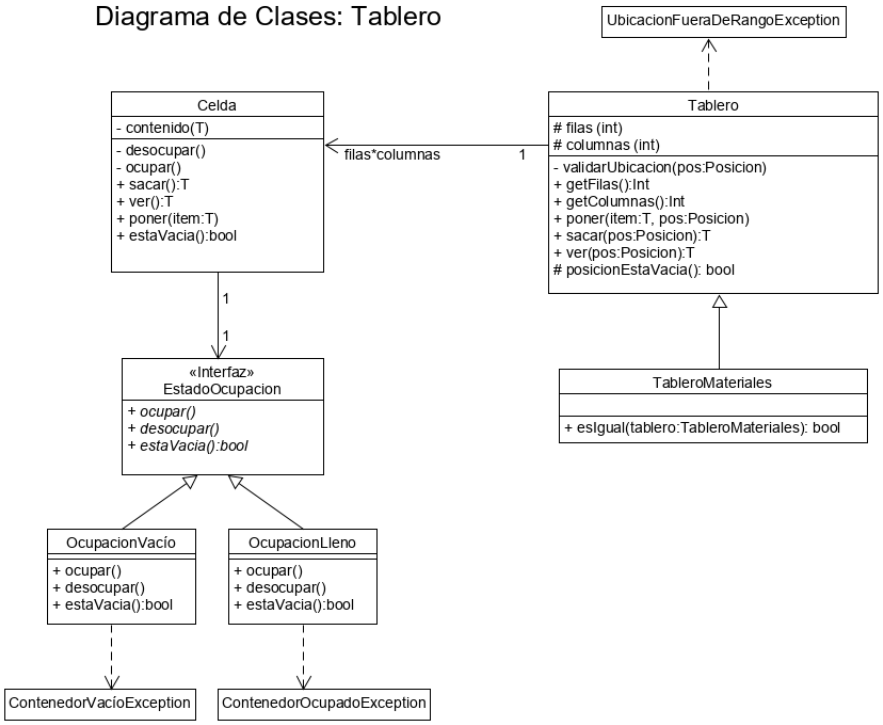

Figure 6: Digrama de Clases: Tablero

2.7 Modelado de AlgoCraft

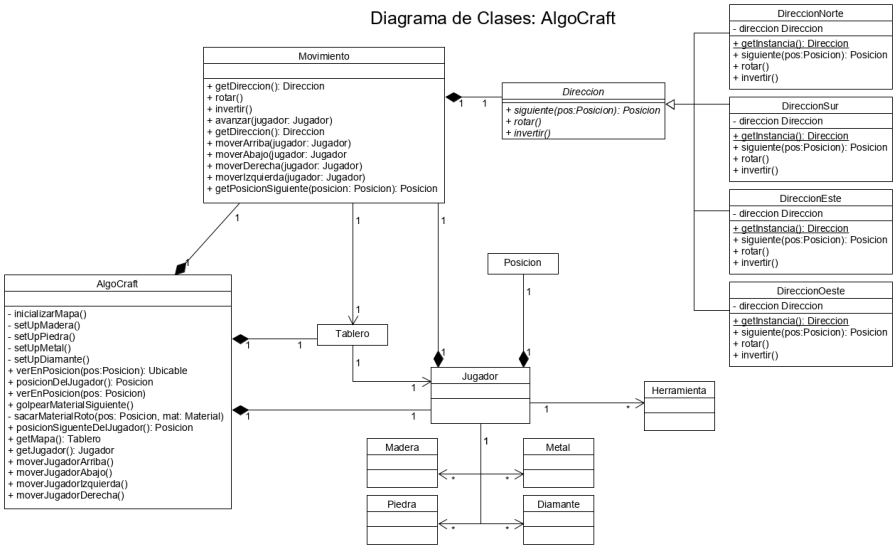

Figure 7: Conoce los distintos elementos del juego y los manipula

3 Diagramas de Secuencia

3.1 Hacha de madera golpea Madera

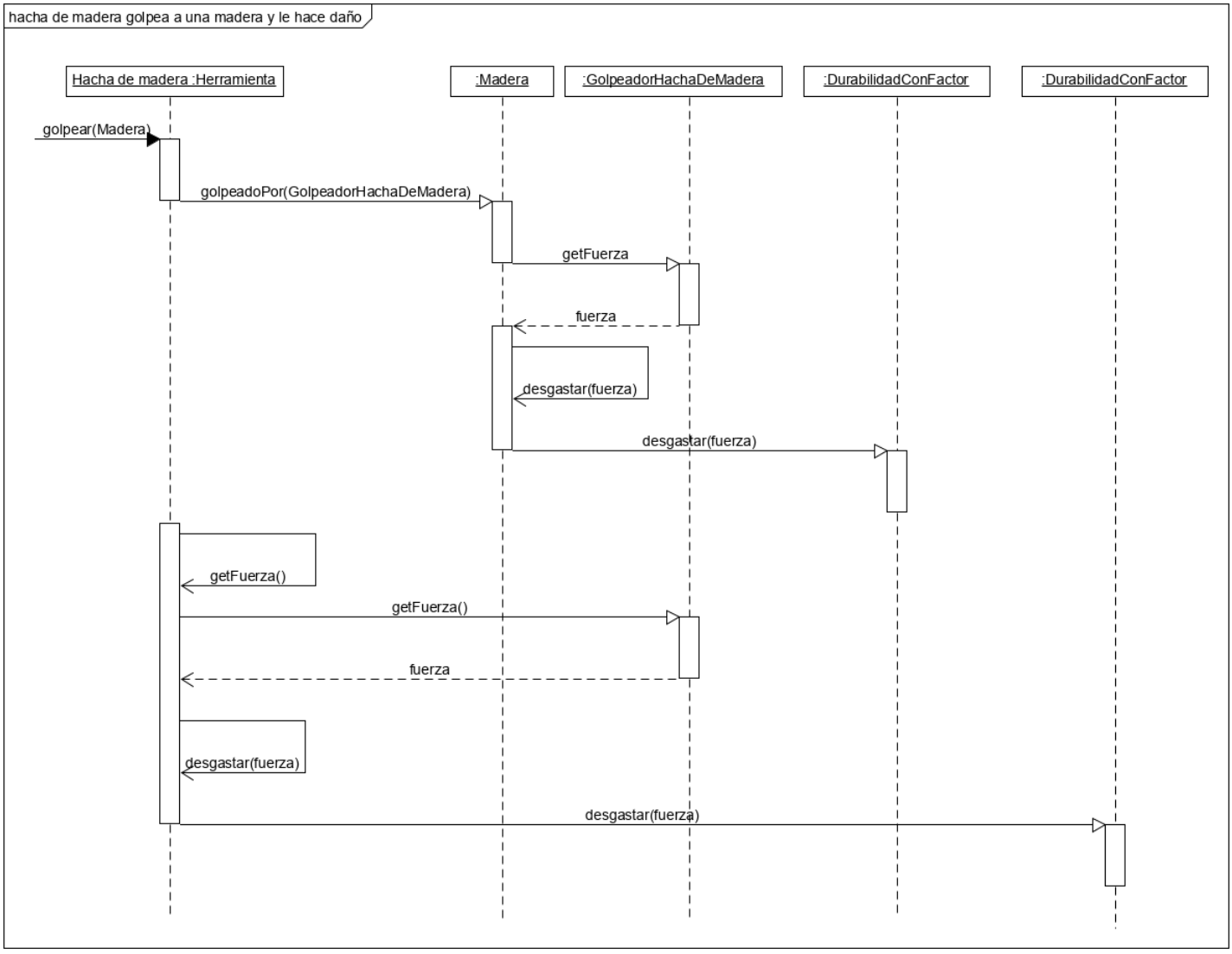

Figure 8: Tanto el hacha como la madera se desgastan

3.2 Hacha de madera golpea Piedra

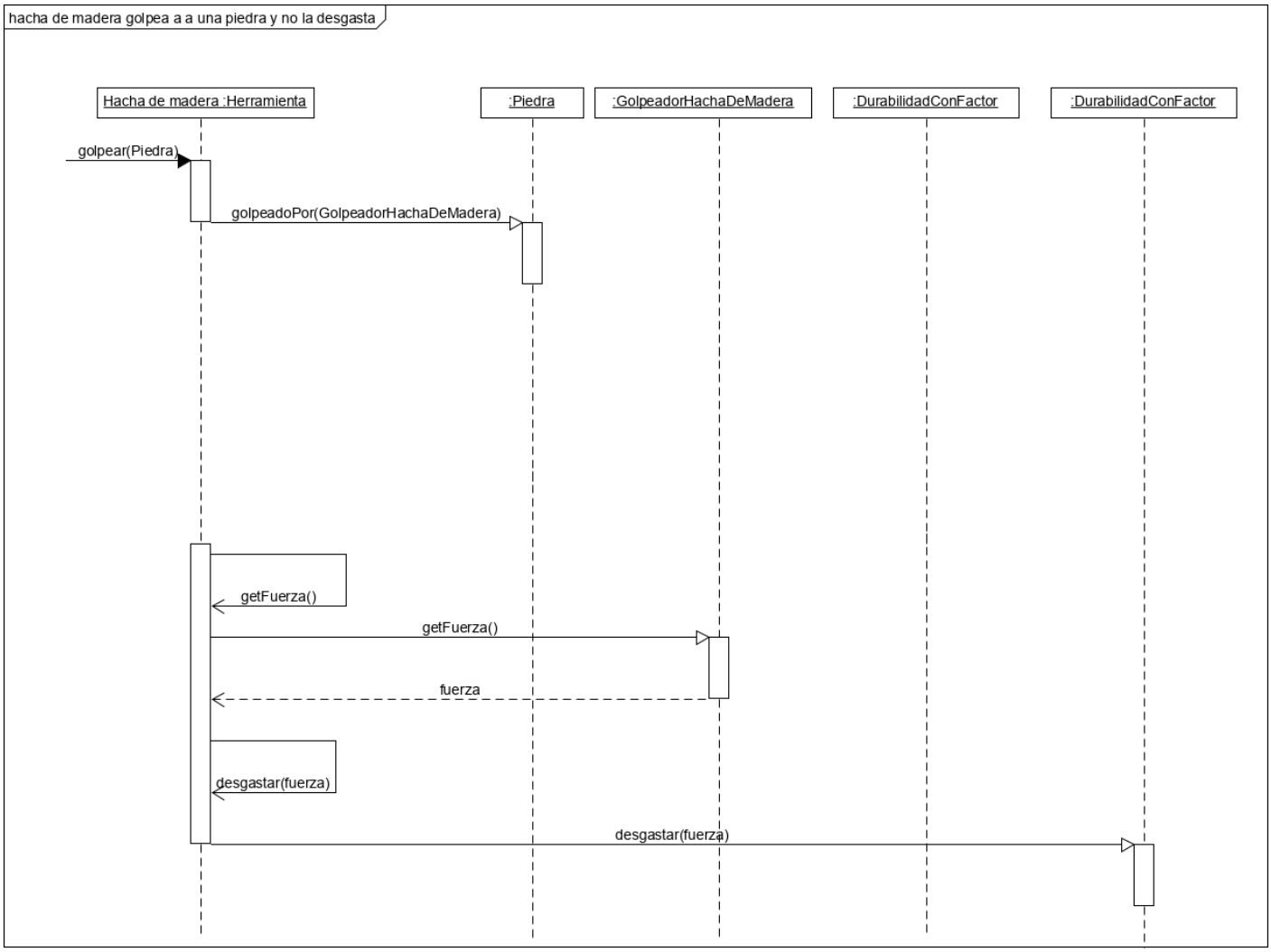

Figure 9: El hacha se desgasta y la madera no

3.3 Jugador golpea madera con hacha de madera

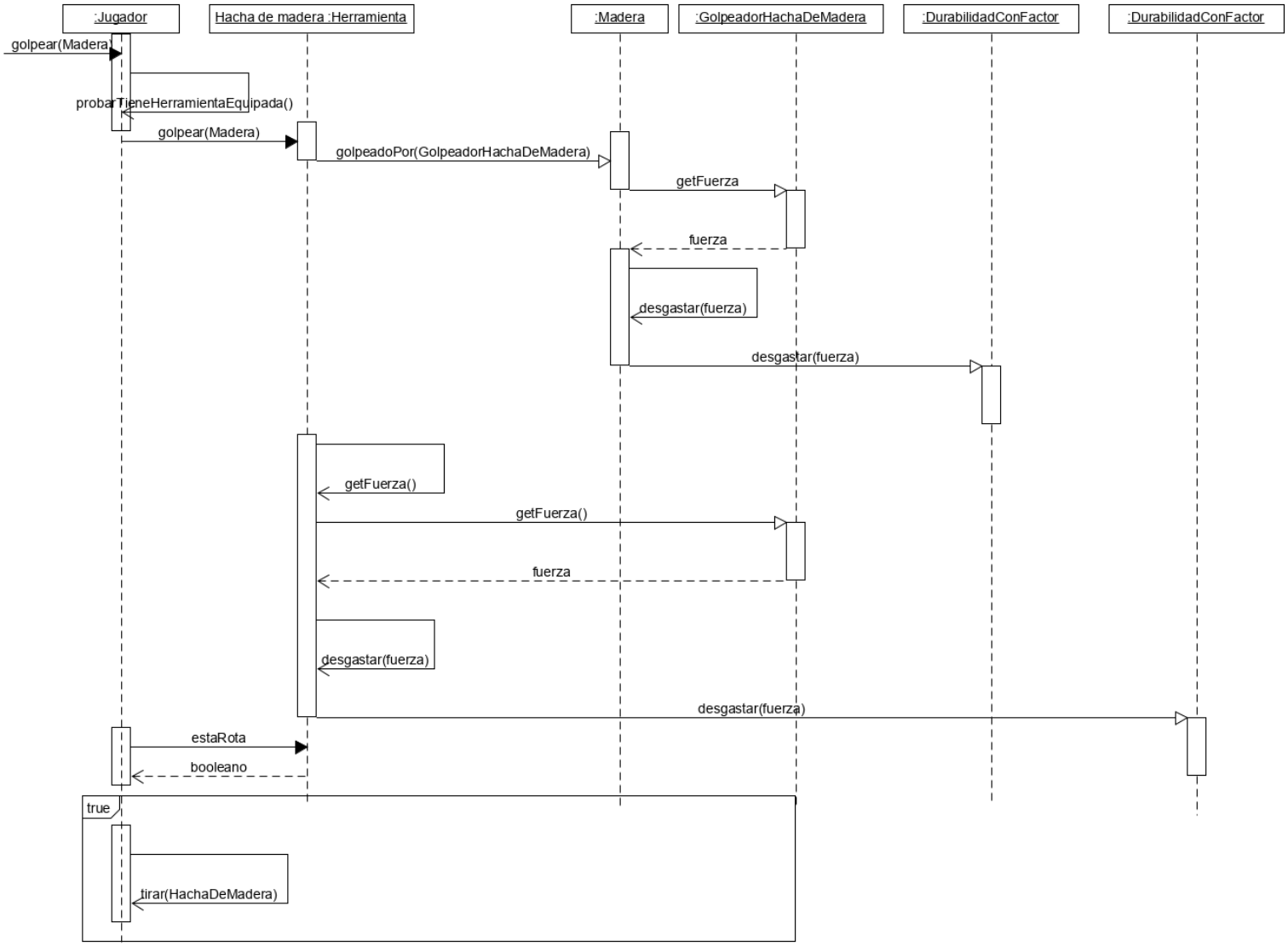

Figure 10: Similar al primer diagrama, se agrega la idea de tirar una herramienta una vez que se rompe

3.4 Jugador se mueve hacia arriba

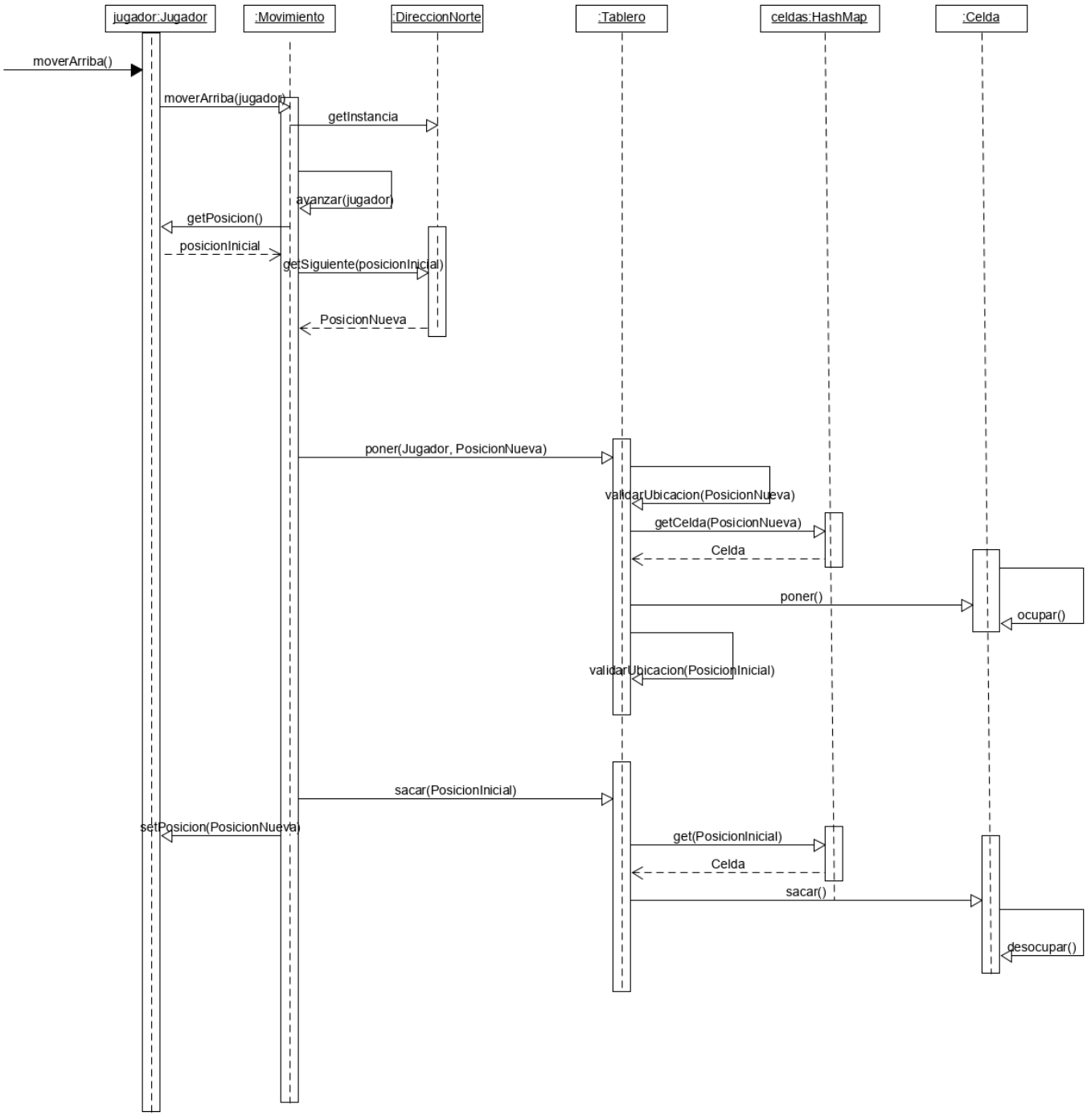

Figure 11: El movimiento en las otras direcciones es análogo

4 Detalles de Implementación

Al comenzar a modelar nuestra solución nos encontramos con el primer problema a la hora de manipular distintos tipos desgastes en distintas herramientas. Decidimos modelar el comportamiento general en la clase abstracta Durabilidad y retrasar la implementación en las clases que heredan: DurabilidadConFactor, DurabilidadConUsos. De esta forma tanto materiales como herramientas están compuestas con un atributo durabilidad en el que delegan la responsabilidad de su propio desgaste.

Luego, tuvimos que modelar cómo interactuaban las herramientas con los materiales, aquí el problema más importante que encontramos es que todas las herramientas pueden interactuar con todos los materiales potencialmente de forma distinta. En particular, todas las herramientas se desgastan al interactuar con cualquier material, pero no todos los materiales se desgastan con todas las herramientas. Para solucionar esto, se creó para cada una de las herramientas una clase golpeador, con esto los materiales mediante el patrón Double Dispatch resuelven contra qué golpeadores se desgastan y contra cuáles no. Estos últimos, además, no tienen la responsabilidad de conocer a qué desgastan.

Por otro lado, tuvimos que pensar una solucion para modelar la creacion de herramientas en el tablero de construccion del usuario. El problema más grande que encontramos aquí fue el hecho de que el usuario puede colocar una gran cantidad de combinaciones posibles en el tablero de construccion (algunas correctas y otras no). Para esto, lo que hicimos fue crear una estructura de clases que modelen un Tablero (que tambien nos iba a servir para modelar el mapa) la cual contiene todo el comportamiento necesario para este problema, luego creamos otra estrucura de clases que modelen los esquemas de las herramientas (una clase de esquema por herramienta), cada uno de estos esquemas se encarga de decidir si otro esquema es igual o no a el, con estos esquemas creados solo teníamos que comparar el esquema ingresado por el usuario con nuestros esquemas para ver si la creacion de una herramienta era correcta o no.

Otro detalle de implementación que consideramos interesante es el del movimiento del jugador. La solución que modelamos e implementamos se basa en delegar lo relacionado al movimiento en una clase que se ocupe de ello, a la que llamamos Movimiento. El jugador puede moverese en cuatro direcciones (arriba, abajo, izquierda y derecha) y para modelar cada dirección también creamos una clase para cada caso. De esa manera, para moverse (cambiar de posición) basta con que el Movimiento le pida a la Dirección cual es la Posición siguiente, se la asigne al jugador y luego lo reubique en el Tablero.

5 Excepciones

NoExisteEsquemaException: Excepcion que se lanza en el caso de que el esquema de herramienta ingresado por el usuario sea invalido.

SinHerramientaEquipadaException: Excepcion que se lanza cuando el jugador intenta golpear sin ninguna herramienta equipada.

ContenedorOcupadoException: Excepcion que se lanza cuando se quiere poner un objeto en una ubicacion ocupada.

ContenedorVacioException: Excepcion que se lanza cuando se quiere sacar un objeto de una ubicacion vacio.

UbicacionFueraDeRangoException: Excepcion que se lanza cuando se quiere acceder a una posicion fuera de los rangos del tablero.

TableroCreacionException: Excepcion que se lanza cuando se quiere crear un tablero con dimensiones invalidas.