FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — AlgoCraft

[7507/9502] Algoritmos y Programación III
Curso 1
Primer cuatrimestre de 2019

Integrantes del grupo		
DAVEREDE Agustín	98540	agusdavi64@gmail.com
HUENUL Matías	102135	matias.huenul.07@gmail.com
HUZAN Hugo	67910	hhuzan@gmail.com
LAMPROPULOS Santiago	101862	santiagolampropulos@gmail.com

Índice

1. Introducción

En el presente informe se expone la aplicación desarrollada para el segundo trabajo práctico de la materia, en lenguaje Java, y los conceptos teóricos de la programación orientada a objetos utilizados en el diseño de la misma.

2. Supuestos

Debido a detalles no especificados en la consigna del trabajo práctico, se ha decidido adoptar los siguientes supuestos.

- Supuesto 1...
- Supuesto 2...

3. Modelo de dominio

El diseño del trabajo consiste principalmente en las siguientes clases.

Jugador Modela al jugador de la partida, que puede moverse en el mapa del juego. Posee un inventario de herramientas y materiales. Puede recolectar materiales y luego usarlos para fabricar herramientas.

Mapa Modela al mundo en el cual el jugador puede moverse. El mapa es un conjunto de celdas, las cuales pueden estar vacías o ocupadas, ya sea por el jugador o por distintos materiales.

Herramienta Es una clase abstracta que modela una herramienta genérica. Posee una durabilidad y una fuerza determinadas por el tipo específico de herramienta. Puede ser usada en materiales, reduciendo la durabilidad de éstos y también la propia.

Material Es una clase abstracta que modela un material. Posee una durabilidad, que puede ser desgastada por una herramienta. Se encuentran distribuidas en el mapa del juego y al reducirse por completo su durabilidad puede ser obtenida por el jugador.

4. Diagramas de clase

A continuación se encuentran los diagramas que muestran las clases implementadas y cómo se relacionan entre sí.

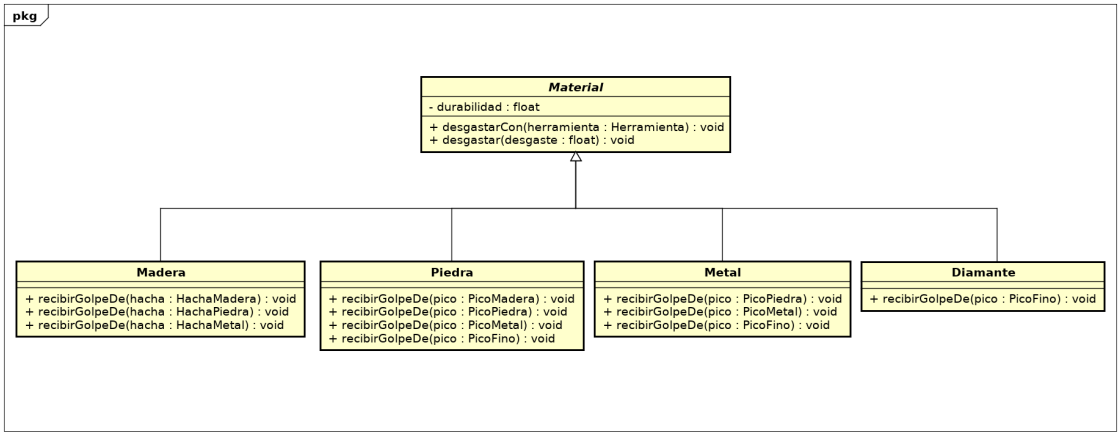

Figura 1: Diagrama de clases de Material.

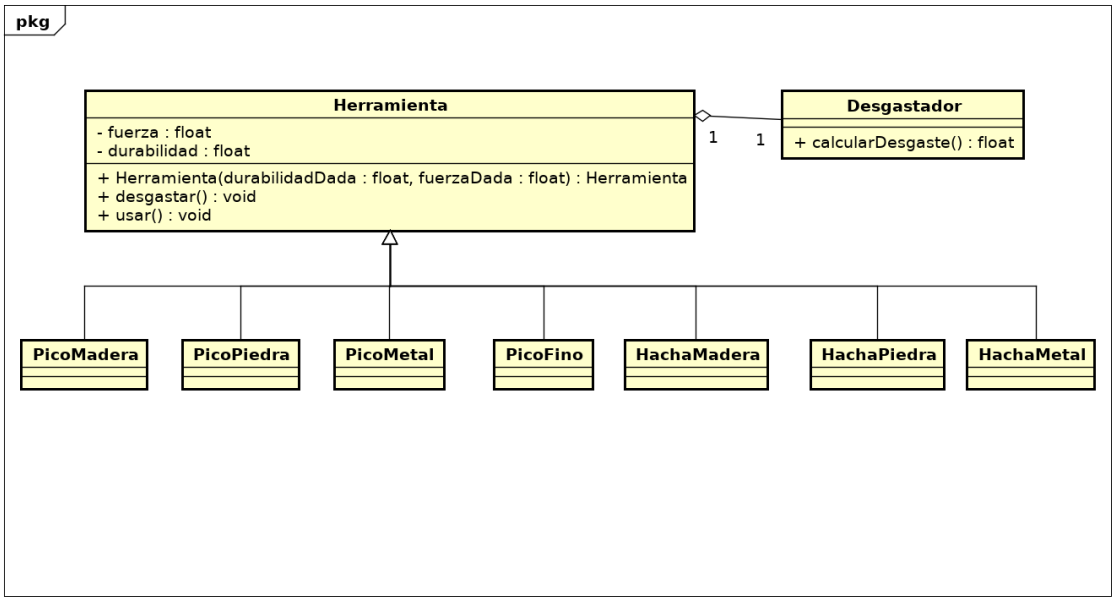

Figura 2: Diagrama de clases de Herramienta.

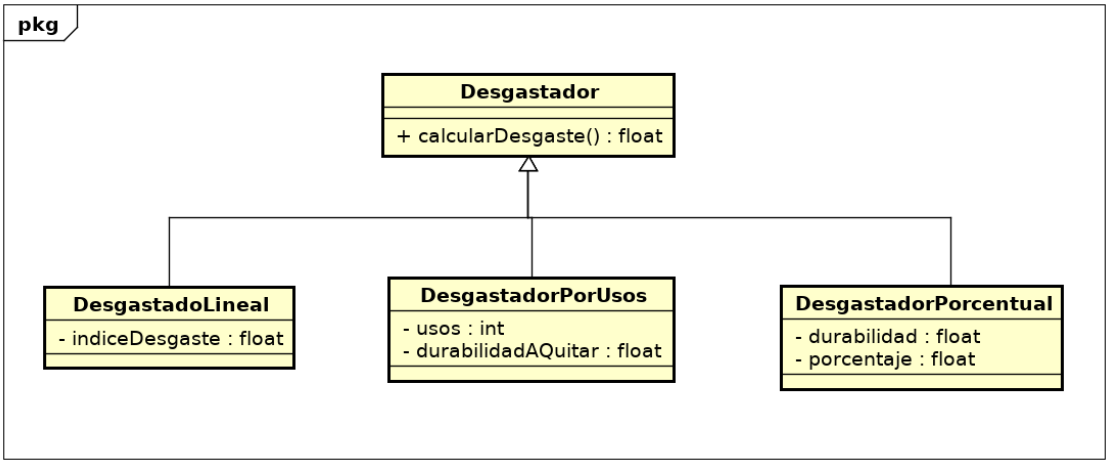

Figura 3: Diagrama de clases de Desgastador.

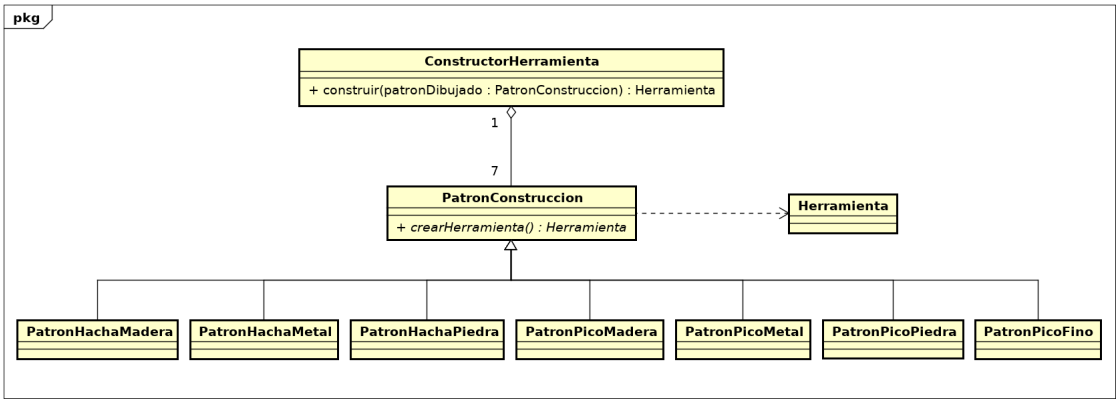

Figura 4: Diagrama de clases de ConstructorHerramienta.

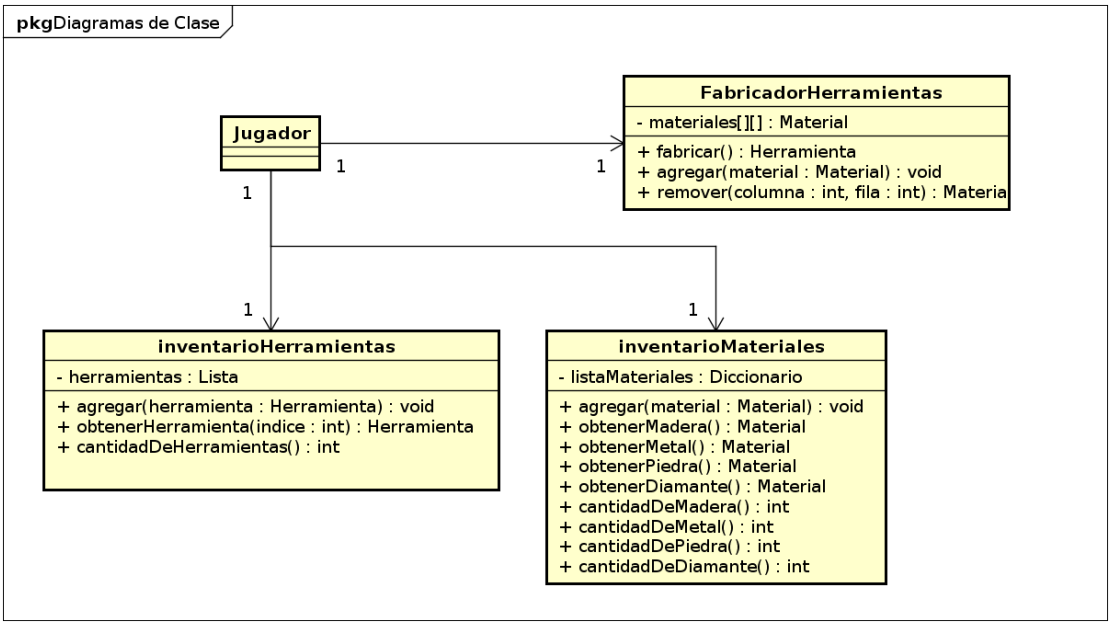

Figura 5: Diagrama de clases de Jugador.

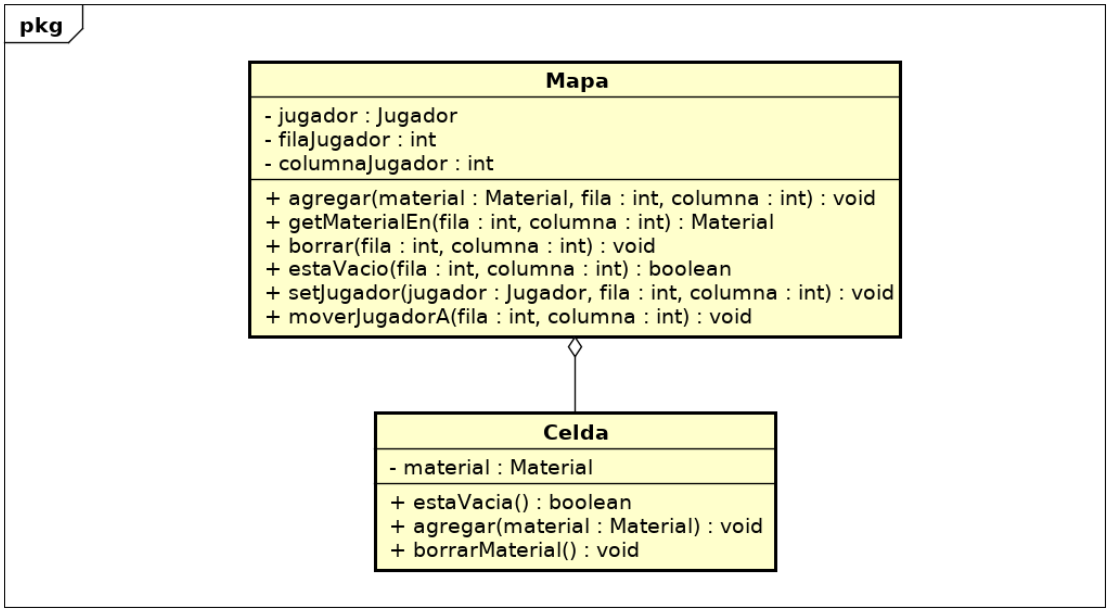

Figura 6: Diagrama de clases de Mapa.

5. Detalles de implementación

5.1. Interacción jugador-material

Uno de los primeros problemas que surgieron al encarar este trabajo práctico fue definir cómo lograr que cada material pueda ser desgastado sólo por los correspondientes tipos de herramientas. Una opción era que cada material pregunte primero que clase de herramienta lo está golpeando y en base a eso decidir si se desgasta o no. Sin embargo, esta solución suponía utilizar muchos *if* y además no resultaba muy apropiada en el paradigma de objetos. Entonces, se decidió utilizar la técnica de *double dispatch*. De esta forma, cada material posee un método por defecto para recibir un golpe de una herramienta genérica que no modifica su durabilidad y luego se redefine este método con cada tipo de herramienta que sí puede dañar al material.

5.2. Inventarios

Los inventarios de materiales y herramientas tienen comportamiento distinto.
Por un lado, se tiene el **inventario de materiales**: al jugador no le interesa qué material en particular está eligiendo, si no sólo su tipo (todos los materiales de un mismo tipo son equivalentes), y ya que el número de materiales recolectados puede ser en principio muy grande como para disponerlos en una lista, se decidió implementarlo como un diccionario donde las claves son el tipo y los valores, *ArrayLists* con las unidades de materiales. De esta forma, lo que se ve en pantalla es un casillero por tipo de material con la cantidad recolectada de cada tipo.
Por otro lado, el **inventario de herramientas**: cada herramienta es única en cuanto a su estado en un determinado momento (todas pueden tener distintas durabilidades) y debido a que el jugador debe ser capaz de elegir exactamente qué herramienta desea utilizar, se decidió implementarlo como un *ArrayList* de herramientas. También justifica esta elección el hecho de que el inventario de herramientas tiene una capacidad menor y pre-fijada, por lo cual tiene sentido poder mostrar todas las herramientas disponibles en pantalla.

5.3. Fabricación de herramientas

En el caso de la fabricación de herramientas, el jugador debe poder crearlas sólo si se agregan los materiales en un patrón específico. Para esto, se creó una clase *FabricadorHerramientas* que contiene distintos patrones pre-establecidos, de clase *PatronConstruccion*: cada clase derivada de esta última posee un método *fabricar()* que devuelve una instancia del tipo de herramienta correspondiente. Así, cuando se ingresa un patrón, el fabricador puede compararlo con los patrones existentes y en caso de coincidir con alguno de ellos, ejecuta su método.

6. Excepciones

NoHayMaterialException Se lanza cuando se intenta quitar un tipo de material del inventario del que no quedan unidades.

EspacioOcupadoException Se lanza cuando se intenta agregar un material en un espacio ocupado en el fabricador de herramientas.

FabricacionNoValidaException Se lanza cuando se intenta construir una herramienta con un patrón no existente.

7. Diagramas de secuencia

A continuación se presentan diagramas de algunas secuencias importantes del juego.

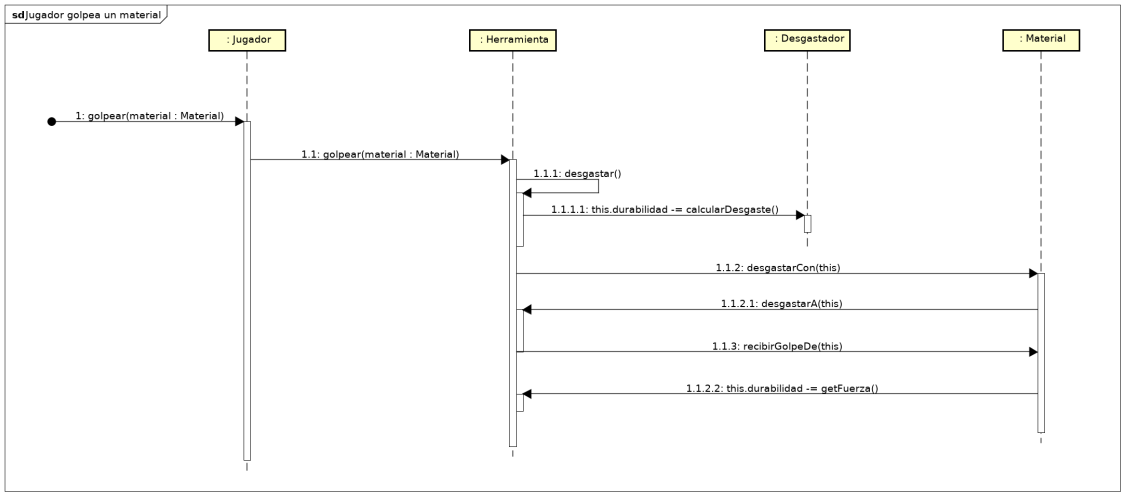

Figura 7: Jugador golpea un material.

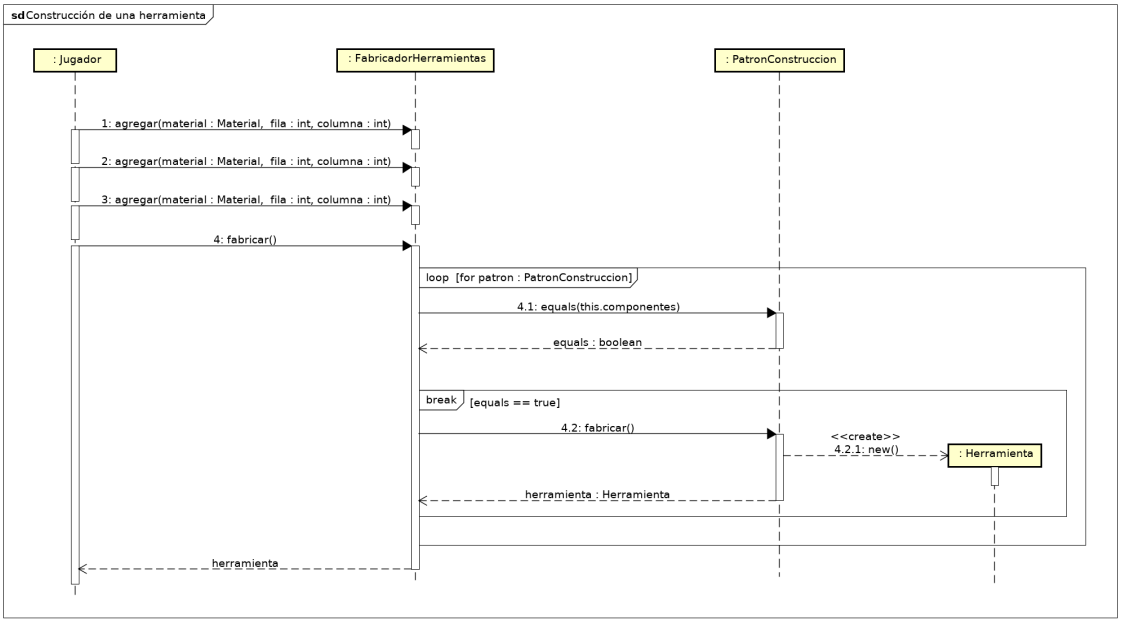

Figura 8: Construcción de una herramienta.