FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — AlgoCraft

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

Alumnos	Numero de Padron
AGUADA, Belen	96851
ARENAS, Alejandro	98199
BUDNECHKY, Jenko	102983
PANETTA, Martina	103713

Índice

1. Introducción

El presente informe reune la documentación de la solución del tercer trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación reducida y similar al reconocido juego MineCraft en el lenguaje java. utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Como algunas especificaciones no fueron provistas por la catedra, se tuvieron en cuenta algunos supuestos para abordar determinadas situaciones que quedaban a cargo del equipo de trabajo, las cuales se enuncian a continuación.

- Un jugador utiliza el mismo mecanismo para moverse que para romper un material, es decir si al moverse en una dirección se topa con un material, lo desgasta .
- No hay turnos o niveles, si no que el juego dura mientras existan materiales que romper y herramientas por construir.
- Una vez çrafteada"(construida) una herramienta, el jugador debe realizar un movimiento para actualizarla en el inventario.
- Tras construir herramientas, el jugador siempre puede elegir herramientas anteriores incluso si están totalmente gastadas sólo que, en ese caso, no podrá obtener ningún material al intentar utilizarla.
- El jugador no puede seguir avanzando una vez que alcanzó el borde del tablero.
- El jugador sólo puede desplazarse en cuatro direcciones; casillero derecho, casillero izquierdo, casillero superior y casillero inferior.
- Al momento de construirse una herramienta, habiendo una coincidencia entre la disposición de materiales realizada por el usuario y algun patrón (receta de construcción") existente, se comunica el match y el usuario puede decidir si quiere construir o no esa herramienta.

3. Modelo de dominio

El dominio del problema se baso en ubicar en un tablero tipo grilla, distintos tipos de materiales. Bajo el patron MVC el usuario controla a un objeto de instancia unica llamado Jugador, el mismo se desplaza por el mapa utilizando controladores de javaFX e interactua con el tablero y los materiales, pudiendo recoger materiales para craftear a futuro distintas herramientas.

Se utilizaron distintos patrones para la resolucion, como Double Dispatch, Observer, Null Object.

4. Diagramas de paquetes

A continuación se muestra la interacción entre paquetes y las clases contiene cada uno.

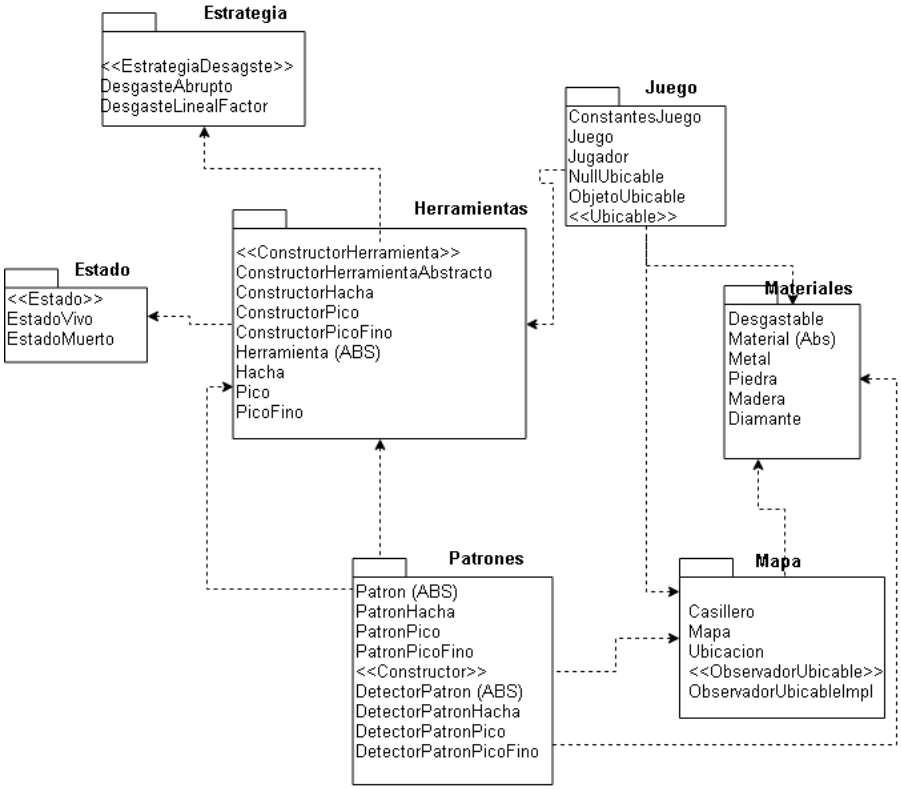

Figura 1: Diagrama de paquetes.

5. Diagramas de clase

Los diagramas de clases estan divididos en 2 partes, de acuerdo a la interaccion de las mismas, por un lado tenemos los Materiales, Estados, Herramientas y sus Constructores correspondientes.

El segundo diagrama muestra las clases Juego, Jugador, cada DetectorPatron, cada Patron, Mapa y objetos Ubicables.

Se cuenta también con un diagrama de paquetes, el cual muestra la interaccion entre paquetes y que clases contiene cada uno.

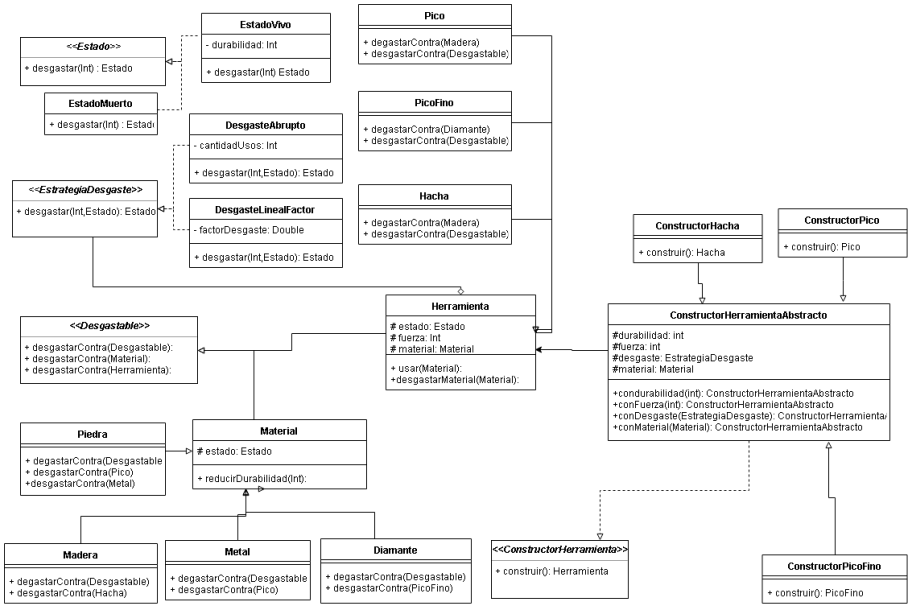

Figura 2: Diagrama de clases 1.

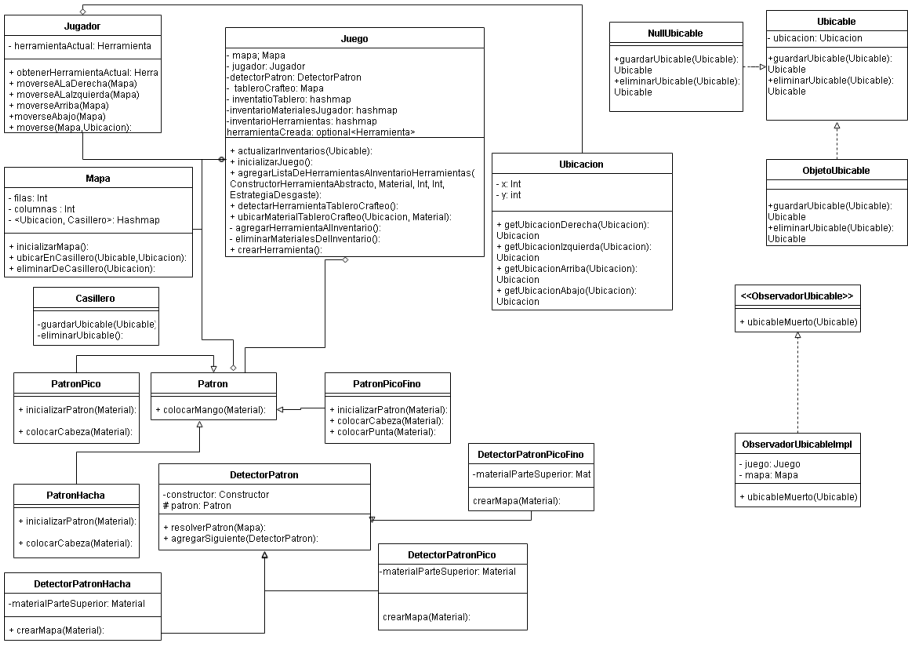

Figura 3: Diagrama de clases 2.

6. Diagramas de secuencia

Se realizaron 5 diagramas de secuencia, buscando tener ejemplos lo menos parecido posibles y abarcar diferentes procesos a lo largo del programa.

En este diagrama de secuencia se muestra al Jugador moviendose hacia un casillero vacio por el mapa.

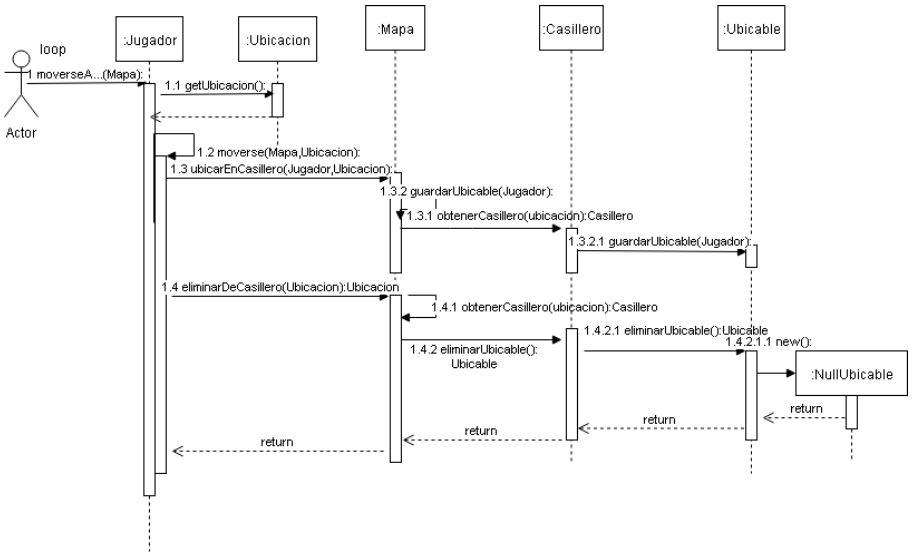

Figura 4: Jugador Moviendose A Ubicacion Vacia.

El siguiente diagrama de secuencia representa el ataque del jugador hacia un material cuando el camina en su direccion. El jugador no puede atravesarlo hasta no haberlo desgastado del todo y recogido el material (se recoge de manera automatica).

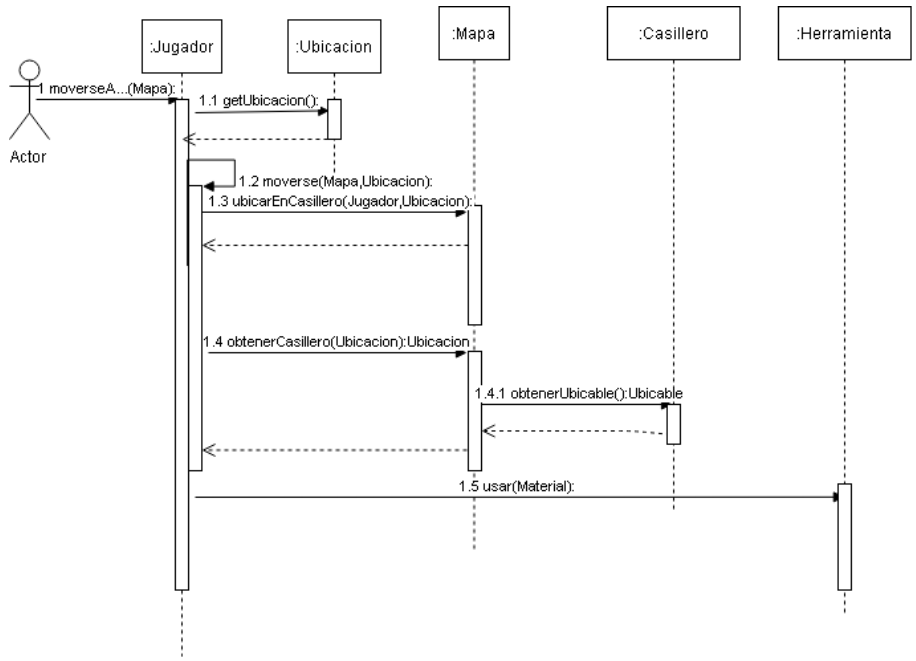

Figura 5: Secuencia Jugador ataca a un material que es contiguo a el.

La siguiente secuencia muestra el uso de un Hacha sobre un Material Madera.

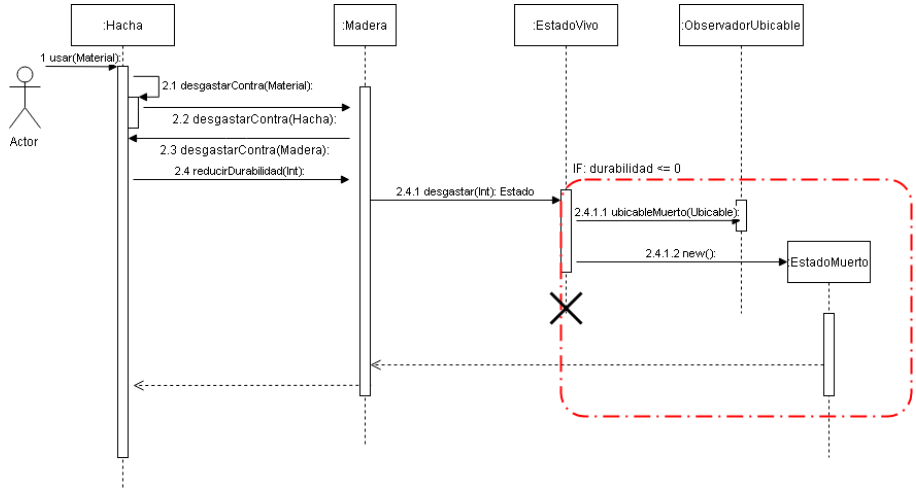

Figura 6: Secuencia Usar Hacha De Madera.

La siguiente secuencia muestra el desgaste completo de un materia, en este ejemplo en concreto es madera, y por medio de un observador se actualiza el mapa.

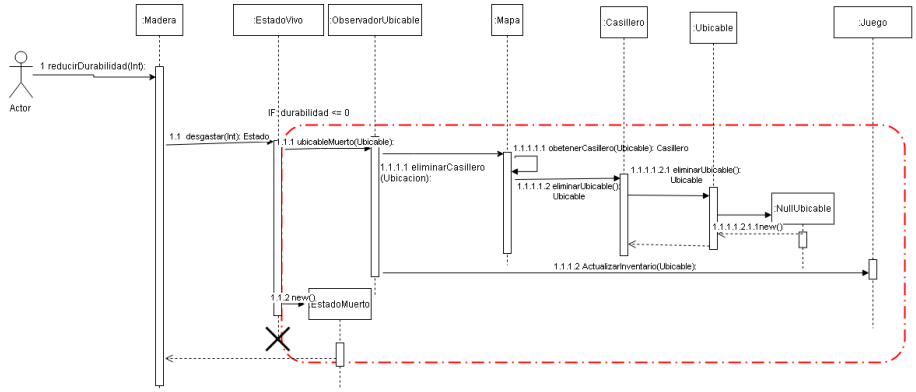

Figura 7: Secuencia Madera es gastada completamente.

La siguiente secuencia muestra la creacion de una herramienta. En ella se ve a deteccion del patron y a creacion de la misma mediante su creador particular.

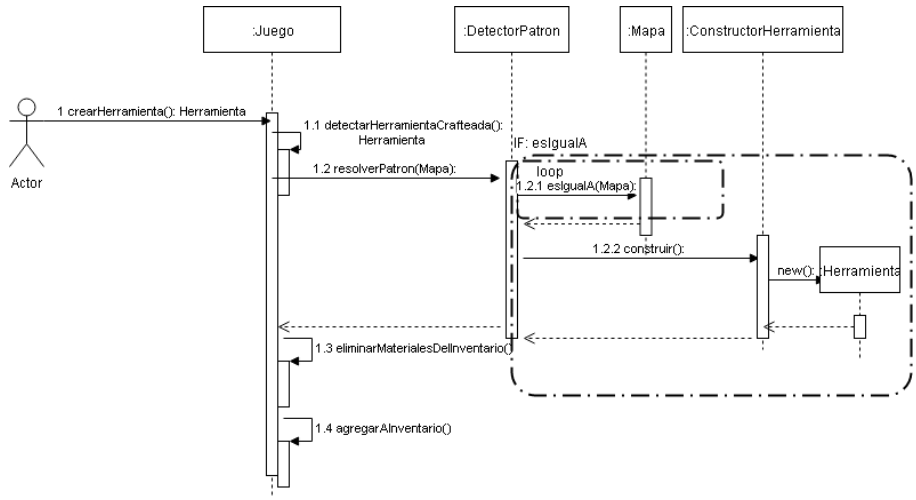

Figura 8: Secuencia de Creacion de una Herramienta generica

7. Detalles de implementación

7.1. Implementación de Clase Patron

La clase Patron es aquella que tiene guardada la cantidad, tipo y disposicion de materiales sobre un tablero de 3x3. El mismo tablero esta implementado con la clase Mapa que contiene un hashmap con objetos Ubicables. Los patrones que estan predefinidos por el juego, se inicializan al arrancar el juego de manera automatica, asi, si eventualmente se añadiera otra herramienta esta tarea se simplificaria ya que solamente se deberia agregar el Patron de la misma. La clase Juego es la que se encarga de inicializarlo cuando el mismo comienza.

7.2. Implementación de Clase DetectorPatron

Esta clase se encarga de buscar coincidencia respecto a un Patron preexistente en un tablero 3x3 que es instancia de la clase Mapa pero con cantidad de filas y columnas 3. Cuando se quiere crear una herramienta con materiales que hay dentro del inventario, y se los va colocando en el tablero en distintas disposiciones, el DetectorPatron va comparandolas con cada uno de los Patrones, y si no hay similitud continua con el siguiente. Cuando sucede que un DetectorPatron detecta un Patron (es decir, cuando hay coincidencia entre el tablero y el patron guardado de dicha herramienta), se permite la construccion de la herramienta correspondiente a dicho "match".

7.3. Implementación de la Clase Ubicable

El objetivo de esta clase fue unificar el comportamiento de cada objeto que podía ubicarse en el tablero. Se trata de una clase abstracta de la cual descienden las clases ObjetoUbicable (clase madre tanto de cada uno de los materiales como de jugador) y clase NullUbicable (implementada con NullPattern, para evitar el uso de nulls en los casos en que no hay un objeto material o jugador en cierta ubicación del mapa.

7.4. Implementación de la Clase ObservadorUbicable

Esta clase sirve para mantener actualizados los estados de los materiales que se encuentran en el mapa notificándole al juego a medida que éstos cambian. La clase se implementó mediante el patrón Observer, acorde al modelo de dominio utilizado (MVC). Asegurándose la actualización de estados de los ubicables, a su vez se permite y facilita la implementación de una interfaz gráfica actualizada.

7.5. Implementación de la clase Estado

Esta clase fue creada para que los materiales y las herramientas definan su comportamiento de acuerdo a si su estado es vivo o muerto. Si su estado está vivo entonces puedo usar una herramienta o desgastar un material, de lo contrario no.

7.6. Implementación de la clase Juego

Esta clase es la encargada de inicializar los componentes del juego necesarios para su funcionamiento como han de ser el jugador, el tablero de juegp trablero de crafteo, y los inventarios, controlando también flujo del mismo.

7.7. Implementación de la interfaz Desgastable

Esta interfaz fue creada para realizar un double dispatch entre las herramientras y los materiales, al querer una herramienta usarse contra un material. Ambas clases implementan desgatable, que solo contiene los métodos DesgastarContra(material/herramienta), devolviendo un optional empty, siendo estos métodos redefinidos según su implementación en cada herramienta y material.

8. Excepciones

HerramientaRotaNoPuedeDesgastarseException : Creada para evitar el uso de una herramienta luego de que sea su durabilidad <= 0.

MaterialSeHaGastadoException : Creada con el fin de no volver a usar una herramienta sobre un material con durabilidad <= 0.

NoExisteNingunCasilleroParaLaUbicacionDadaException : Creada con el fin de no poder ir a una ubicacion invalida fuera del mapa.

NoHayHerramientaParaCrearException : Luego de llamar al metodo crearHerramienta, si la combinacion y disposicion no es la de un Patron, no sera posible que el DetectorDePatron se resuelva y cree la Herramienta. Asi esta Exception fue creada con el fin de atrapar la falta de creacion de la herramienta.

NoSePuedeDesgastarUnElementoConEstadoMuertoException : Cuando la durabilidad del Material pasa a ser <= 0, su estado pasa a ser EstadoMuerto. Esta exception fue creada para capturar el uso de una Herramienta o Material con durabilidad <= 0.

NoSePuedeEliminarPorqueEstaVacioException : Exception creada para capturar el momento donde en eliminacion de materiales se intente eliminar un casillero vacio. Esto ocurre luego de crear una Herramienta.

NoSePuedeUbicarPorqueEstaOcupadoException : Se lanza en el momento que quiera ubicarse sea un Material o el Jugador en un casillero previamente ocupado.