[75.07] ALGORITMOS Y PROGRAMACIÓN III

1º CUATRIMESTRE 2019

TURNO TARDE

TP2: ALGOCRAFT

AUTORES

Carbón Posse, Ana Sofía - #101.187
<asofi06@hotmail.com>

Day, Francisco José - #100.513
<franday_7@hotmail.com>

Fanciotti, Tomás - #102.179
<tomasfanciotti@gmail.com>

Mazza Reta, Tizziana - #101.715
<tizzianamazza@gmail.com>

CORRECTOR

Degiovannini, Marcio

28 de junio de 2019

Índice

1. Objetivo del trabajo

En el presente trabajo práctico se desarrolla una aplicación que implementa el juego Minecraft utilizando el lenguaje de programación Java. Se busca cumplir los siguientes objetivos:

- Realizar un análisis de la problemática planteada y su modelado mediante notación UML.
- Debido a su extensión, describir resumidamente los componentes más importantes del programa.
- Aplicar la teoría de programación orientada a objetos estudiada en el curso.
- Utilizar la técnica de desarrollo *Test-Driven Development*.
- Implementar y aplicar patrones de diseño para resolver problemas puntuales en la implementación del juego.
- Realizar la interacción con los jugadores mediante una interfaz gráfica de usuario utilizando la plataforma JavaFX.
- Tomar conciencia del uso del paradigma de orientación a objetos para modelar problemas que presentan una complejidad media.

2. Supuestos

Para la realizacion del juego debimos interpretar la consigna considerando los siguientes supuestos:

- Cuando se crea el jugador, inicialmente tiene en su inventario un hacha de madera.m
- Para que el jugador pueda interactuar con un material este se debe encontrar en una posicion adyacente.
- Si una herramienta luego de ser usada alcanza una durabilidad igual a cero, la misma se borra del inventario y no se puede utilizar mas.
- El jugador es el único objeto del juego que tiene la capacidad de moverse por el tablero.
- El inventario del jugador tiene un tamaño maximo de 54 unidades.
- El tamaño del mapa es estático a lo largo del juego. Se adopoto un tamaño de 820x820.
- La posicion de los materiales a lo largo del mapa permanece constante durante todo el transcurso del juego (a menos que sean recolectados por el jugador).
- Los materiales se generan en posiciones aleatorias del mapa respetando un patrón para que el juego sea jugable, permitiendo que cada partida sea única.
- Una vez que el jugador pudo recolectar todos los materiales gana el juego.
- Cuando el jugador no puede recoger mas materiales, pero en el mapa siguen existiendo, pierde el juego.

3. Modelo de dominio

En primer lugar, para poder llegar a la solucion que se pretende y considerando que el juego requiere de muchos modulos independientes entre sí, conviene clasificar las clases segun el patron de diseño Modelo-Vista-Controlador. De este modo, podemos encapsular el comportamiento del juego independientemente de como se va a mostrar al usuario y de cómo este interactua con el juego mismo.

3.1. Modelo

El modelo base consta de 3 clases principales: Herramienta, Materiales y Jugador. La clase Herramientas será padre de los dos tipos de herramientas que hay en el juego: Pico y Hacha, que se comportarán distinto segun su tipo y su fabricación. Dicho comportamiento está encapsulado en clases que heredan de Desgaste.

Las instancias de Materiales son aquellos objetos que están distribuidos en el mapa, por lo que dicha clase será tambien padre de todos los tipos de materiales. Las herramientas deben interactuar con los estos ya que necesitamos que los recolecten para el usuario, por lo que un objeto modificará al otro. Finalmente Jugador, ademas de tener la informacion del estado actual (posicion, inventario, herramienta en mano, etc.) y de ser el intermediario entre las herramientas y los materiales para la recoleccion, es quien se encargará mediante una clase, MesaDeCrafteo, de usar los materiales para construir las herramietas. Esta ultima clase interpretará el pedido del jugador y delegará la tarea de construir una determinada herramienta con su respectivo comportamiento a la clase Constructor.

3.2. Controlador

El controlador tiene la responsabilidad de implementar las reglas que hacen referencia a los movimientos del juego, ya que es la que permite al jugador realizar diferentes pedidos. .

3.3. Vista

Se creó la vista para poder interactuar con el usuario, mostrando cada instancia del juego mediante una interfaz gráfica utilizando JavaFX como plataforma. La misma se encarga de contener una escena y realizar las actualizaciones de dibujo cuando suceden diferentes eventos.

1. MapaViewer es la clase encargada de hacer que los materiales, como los arboles, las piedras, los metales y los diamantes aparezcan en el mapa. Se encarga, también, de como tiene que aparece el jugador, si este se desplaza hacia la derecha, izquiera, arriba o abajo.

2. JuegoViewer es el encargado de inicializar todas las clases que se nombraran a continuacion, una vez que apretamos *"Jugar"* al inicio del juego .

Figura 3.1: Imagen del comienzo del juego.

3. HerramientasInvViewer nos muestra el inventario de las herramientas que el jugador posee.

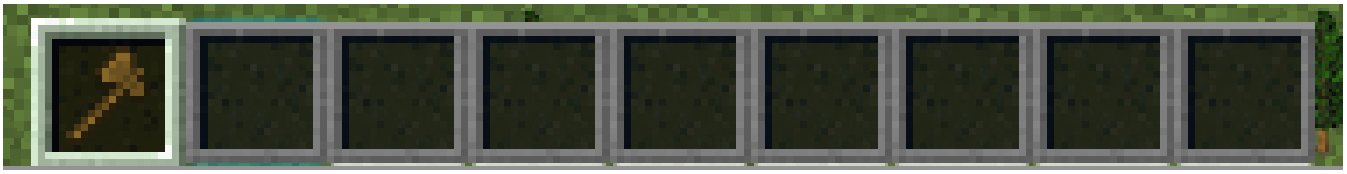

Figura 3.2: Imagen del inventario de herramientas que posee el jugador.

4. MesaDeCrafteoViewer visualiza la mesa de crafteo en donde se ponen los materiales en forma de recetas"para crear las diferentes herramientas.

5. InventarioViewer nos expone el inventario de los materiales que tiene el jugador. Se va actualizando cada vez que este recolecta materiales del mapa.

   Las dos clases nombradas anteriormente se abren sobre el mapa, al apretar la tecla *i"*, de la siguiente forma (figura 3.3)

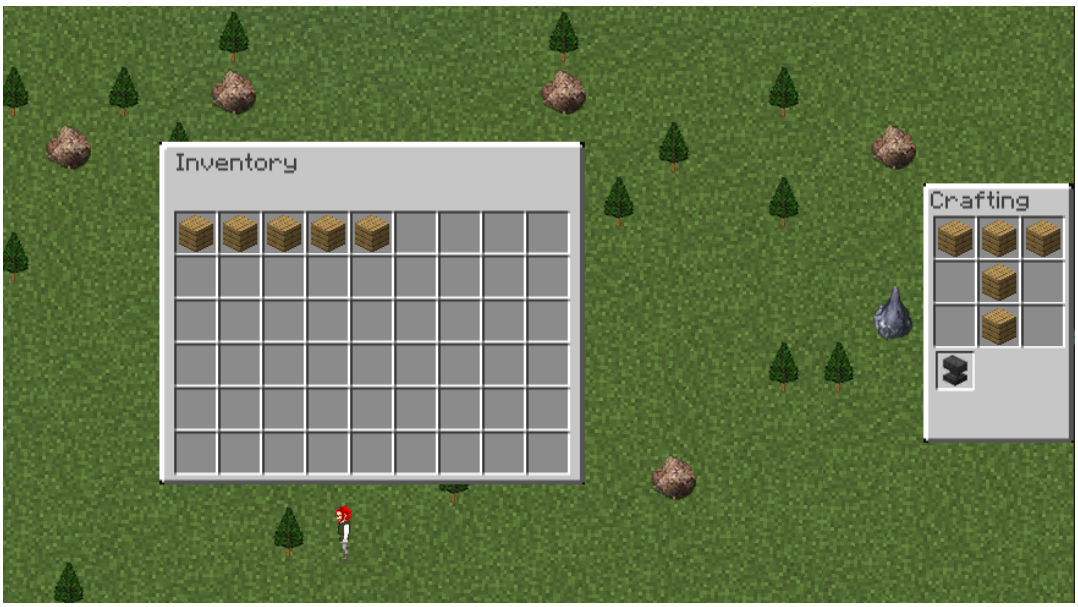

Figura 3.3: Imagen de la mesa de crafteo y del inventario de materiales.

Se puede observar que cuando intentamos poner un material en donde ya hay otro nos salta una ventana de error.

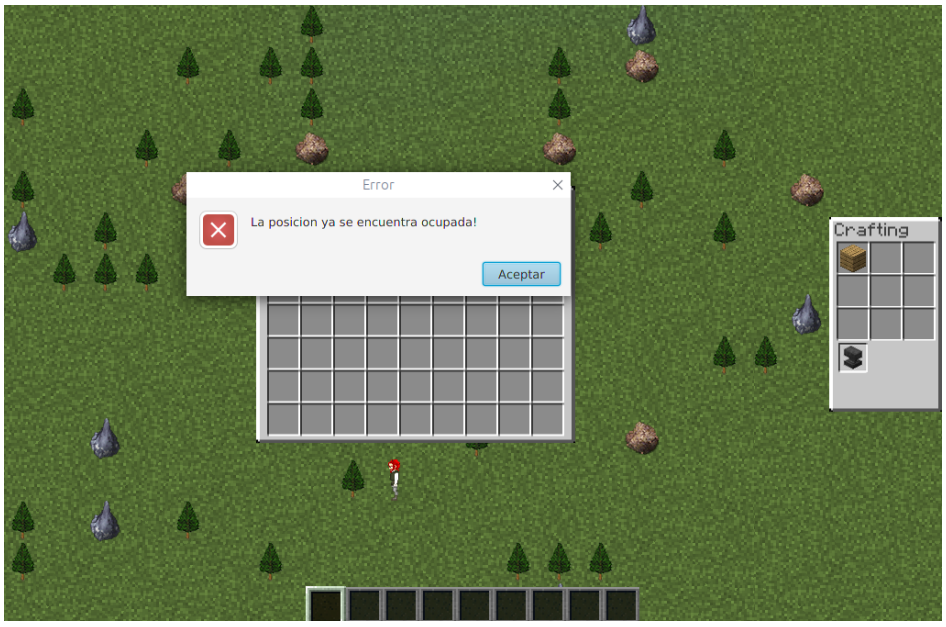

Figura 3.4: Imagen del error que aparece cuando intentamos ubicar un material en la mesa de crafteo cuando ya hay otro alli..

4. Diagramas de clase

En el diagrama mostrado debajo, figura 4.1, se muestra el arbol de herencia de las Herramientas y como todas ellas se relacionan con su comportamiento, dado por Desgaste, y con sus componentes, que son los materiales con los que estan hechas, dado por Material.

De la clase Herramienta salen dos clases hijas, Hacha y Pico, que a su vez heredan más clases. HachaMadera, HachaPiedra, HachaMetal por un lado, y PicoMadera, PicoPiedra, PicoMetal y PicoFino por el otro. Todas las herramientas poseen los mismos atributos y métodos, pero, estos ultimos cambian en cada una de ellas. Se puede observar como todas las hachas poseen la misma implementacion a la hora de recoger los materiales, en cambio, cada pico tiene su propia forma de hacerlo.

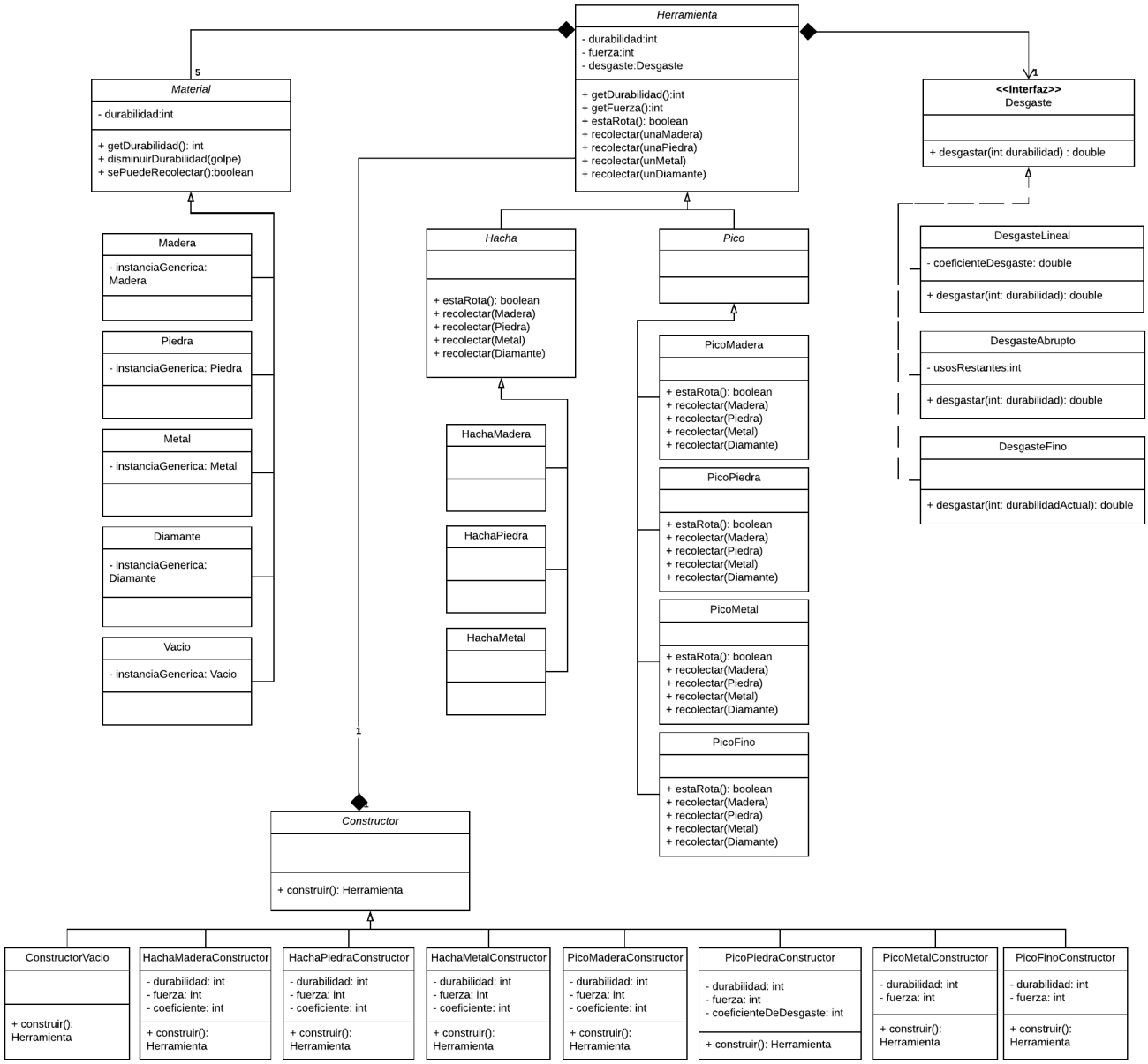

Figura 4.1: Diagrama de Herramientas.

En la figura 4.2 se pueden ver los distintos desgastes que existen, con el método definido en la interfaz Desgaste, pero implementado en cada clase hija de forma diferente, y sus distintos atributos.

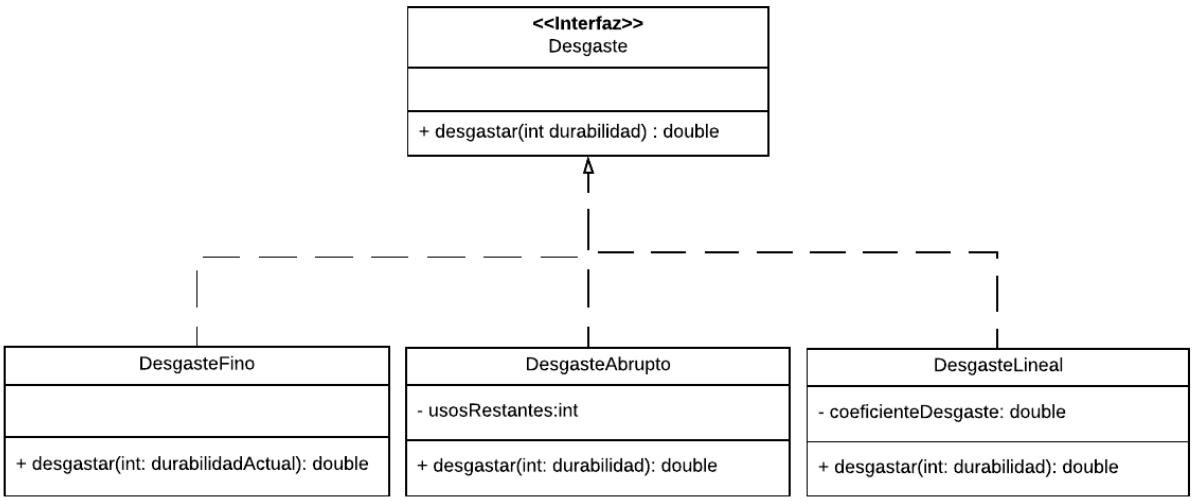

Figura 4.2: Diagrama Desgaste.

En el diagrama del constructor, figura 4.3, se puede apreciar como se redefinen el metodo que tiene la clase madre, Constructor, para cada herramienta. Ademas se observa que los atributos de estas clase hijas son diferentes según sus cualidades.

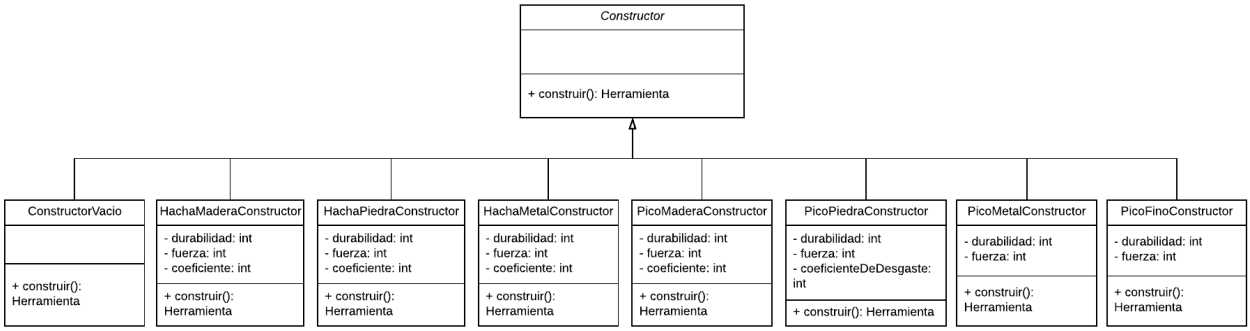

Figura 4.3: Diagrama Constructor.

En el caso de la figura 4.4, la clase Jugador esta en asociacion con las herramientas y con los materiales, ya que este puede contener ninguno o varios de ellos en su inventario. Por otro lado está la clase MesaDeCrafteo, que se relaciona con el jugador, ya que este realiza el pedido de la creacion de herramientas. La MesaDeCrafteo tambien depende del los materiales y del constructor debido a que, una vez ingresado los materiales necesarios e interpretado el pedido, invocará al constructor para que finalmente pueda construir la herramienta que necesita y devolversela al jugador.

Podemos notar que, para interpretar el pedido realizado por el jugador necesitamos tener las recetas de cada herramienta decretada por el juego. En consiguiente, la clase Receta tiene como hijas cada una de ellas. Cuando se quiera crear una nueva herramienta se tiene que comparar la receta actual, que está en la mesa de crafteo, con las de las herramientas ya establecidas, y, según corresponda, se creará una nueva herramienta o se lanzará un error, explicando que la receta planteada por el jugador no corresponde con ninguna ya instaurada.

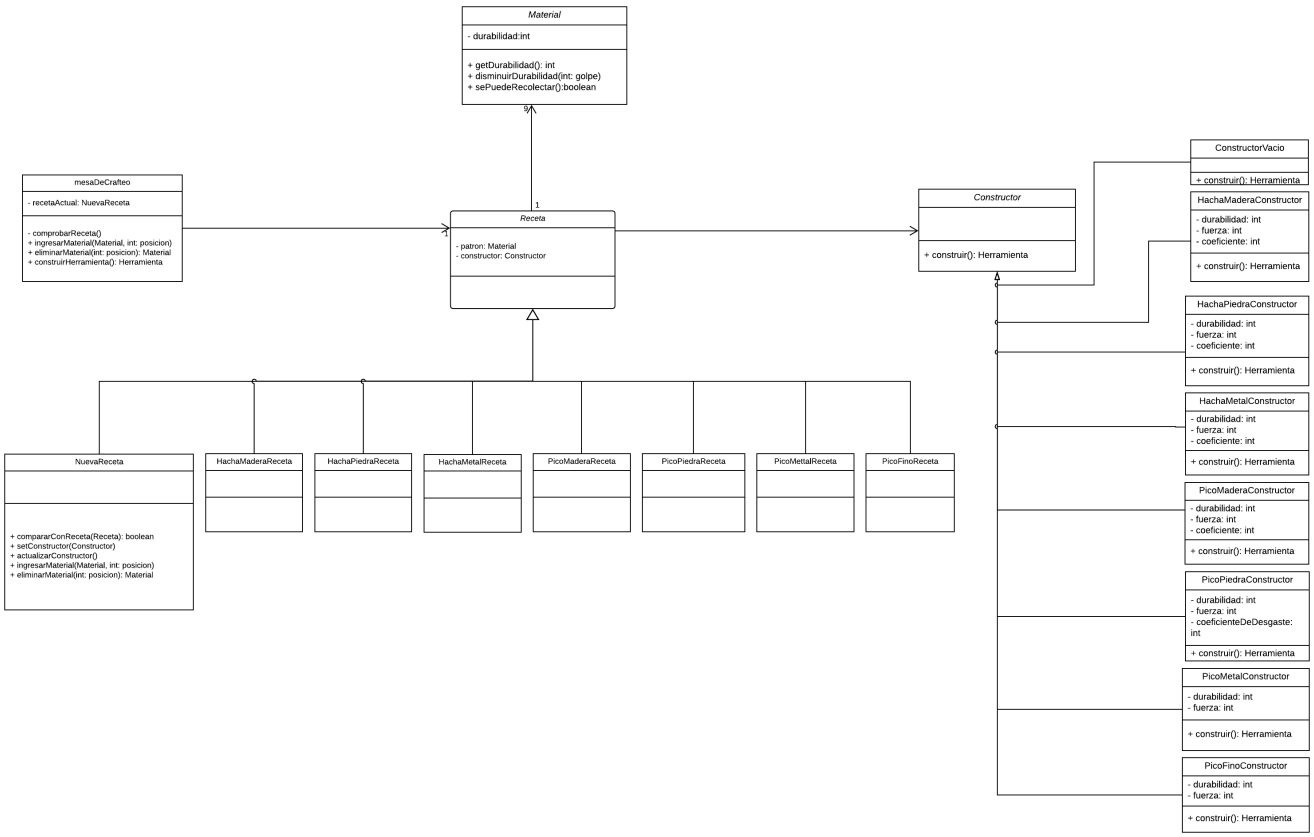

Figura 4.4: Diagrama Receta.

En la figura 4.5 se ve como las clases Jugador y Material tienen una interfaz Contenible. Jugador, a su vez, se relaciona con Inventario, ya que posee uno que se divide en dos, invetario de materiales e inventario de herramientas, en donde se guardan los materiales y las herramientas, respectivamente. A si mismo, se relaciona con Posición y con Mapa, puesto que este se encuetra en una posicion especifica del mapa.

Mapa se relaciona con Posicion y Casillero, pues tiene en cada posicion un casillero. Ademas, el diagrama nos indica que el mapa tiene materiales, que estan distribuidos por la parcela.

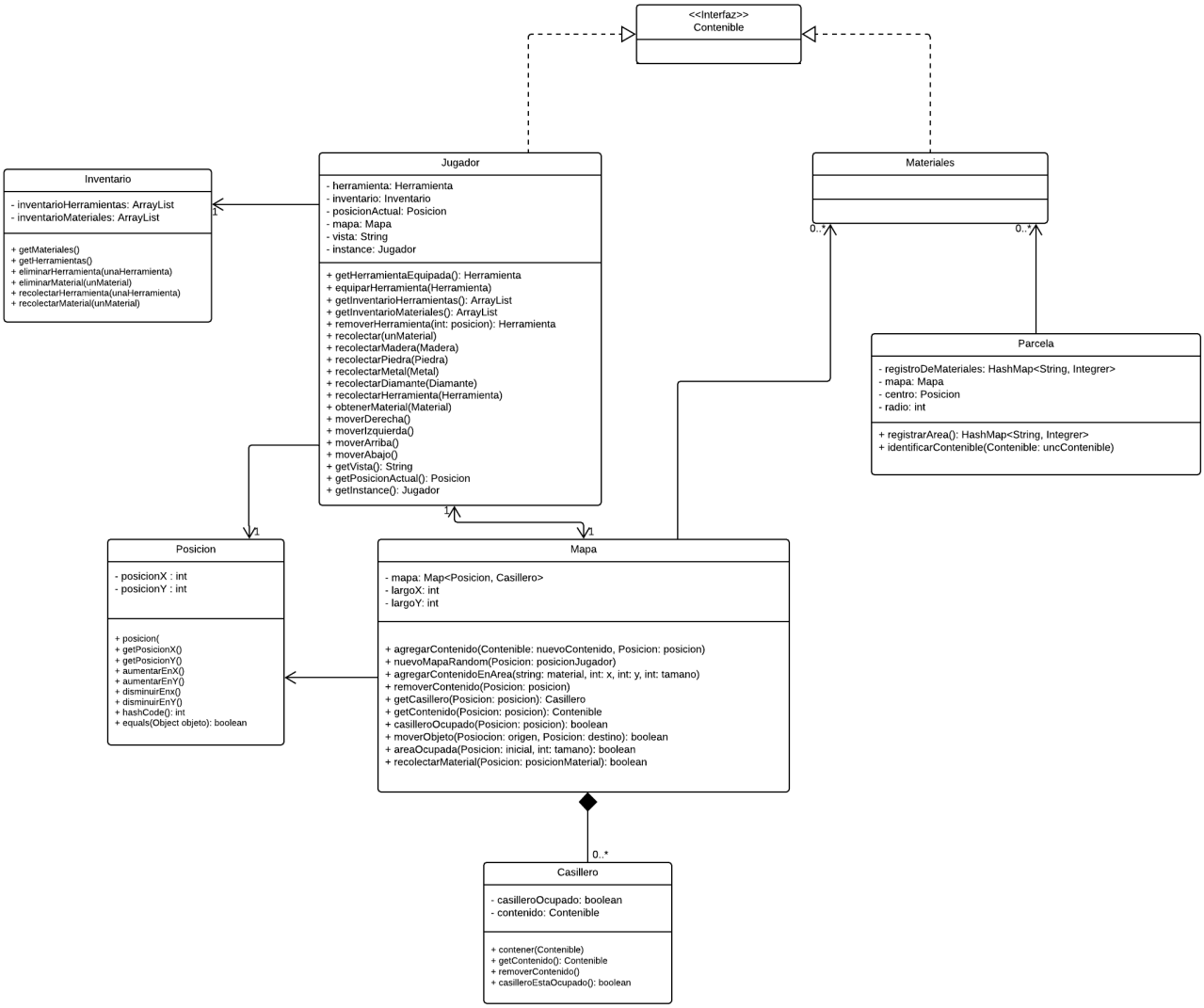

Figura 4.5: Diagrama Jugador - Mapa - Materiales.

El diagrama representado por la figura 4.6 se observa la clase AbstracCreator y la interfaz Creator. Podemos examinar que esta altamente relacionado con Material ya que los generan en el mapa.

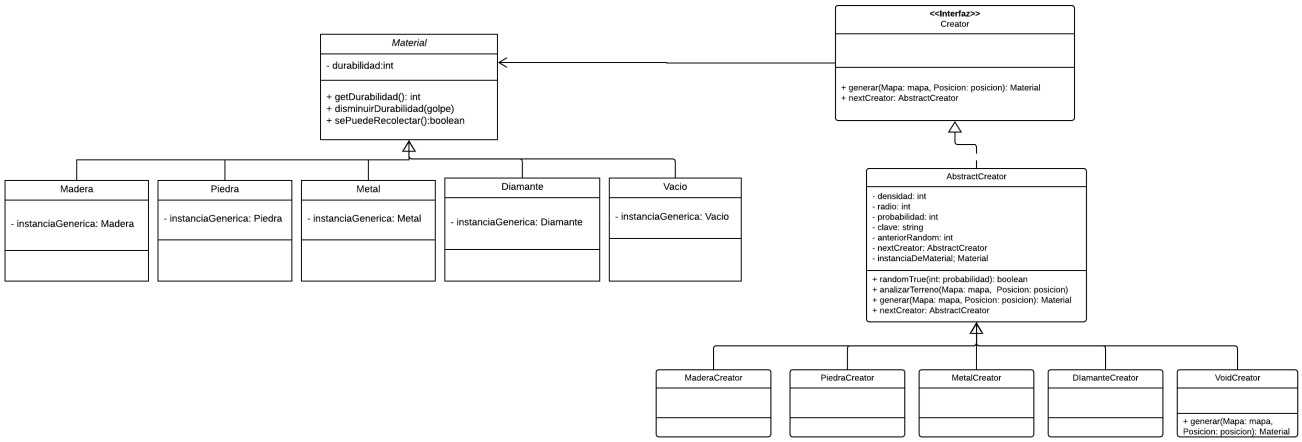

Figura 4.6: Diagrama Creator.

5. Diagramas de secuencia

A continuación mostraremos algunas de las secuencias presentes en el programa.

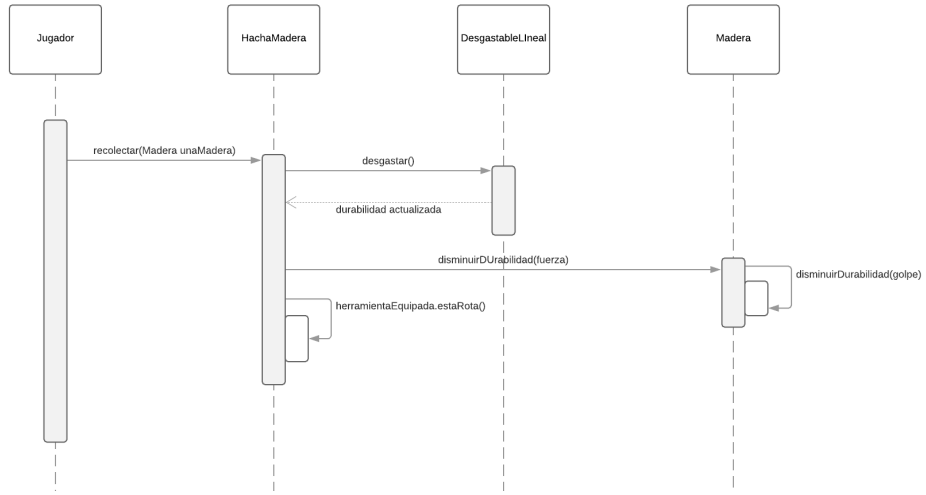

Figura 5.1: Diagrama de secuencia cuando el jugador quiere recolectar una madera con un hacha de madera.

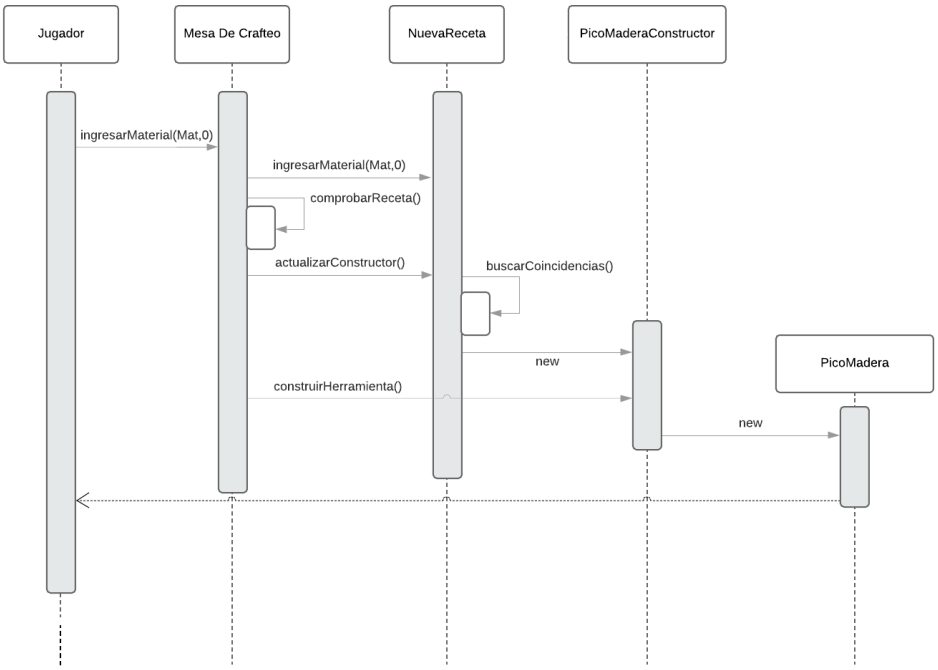

Figura 5.2: Diagrama de secuencia cuando el jugador quiere construir un pico de madera.

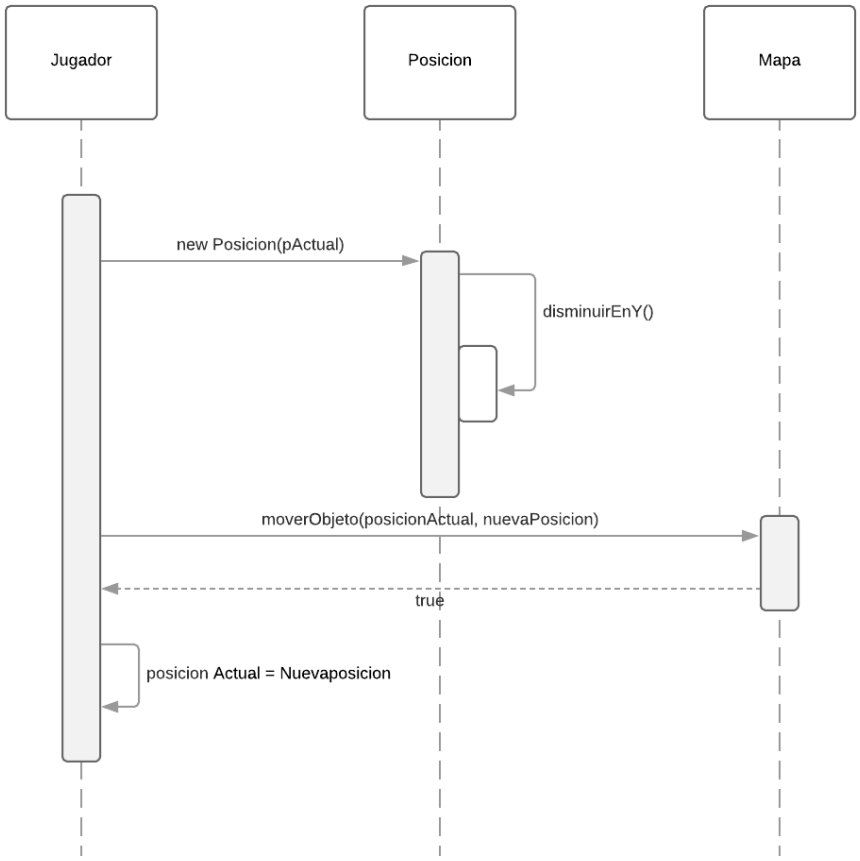

Figura 5.3: Diagrama de secuencia cuando el jugador se quiere desplazar hacia abajo en el mapa.

6. Diagramas de estado

El diagrama de estados de las herramientas se puede resumir en tres estados no cíclicos. *"Herramienta nueva"* es el estado inicial en el que se instancia cada herramienta para poder ser usada por primera vez ,y su durabilidad es la máxima de la propia herramienta. El estado *"Dañada"* es aquel en el que muta cuando es usada por primera vez, y cada vez que se vuelve a utilizar, reduciendo aún más su durabilidad.

Por último, el estado *Rota"* sucede cuando su durabilidad es igual a 0 y, por consiguiente, la herramienta es retirada del inventario dando por finalizada la secuencia de estados.

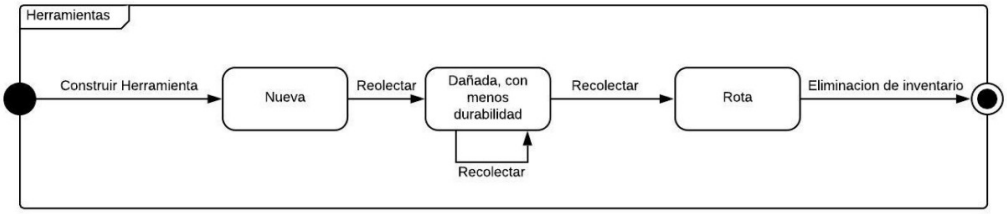

Figura 6.1: Diagrama de estado de las Herramientas.

En el caso de los materiales el diagrama es similar por el hecho de que tiene la misma dinámica que las clases herramientas. Cada material es inicializado por el juego automáticamente (a diferencia de las herramientas, que son inicializadas por el jugador), en el mapa y con su durabilidad máxima. Inicialmente tiene un estado de *"No recolectable"*. Luego, cuando el jugador intenta recolectarlo, sigue como no recolectable hasta que su durabilidad llegue a "0". Ese instante es representado por el estado -*oto"* en inventario. El usuario tiene la posibilidad de ingresar o quitar el material en la mesa de crafteo (más especificamente en la receta) y en caso que lo ingrese, este puede ser utilizado para construir herramientas, finalizando así la vida de la instancia del material.

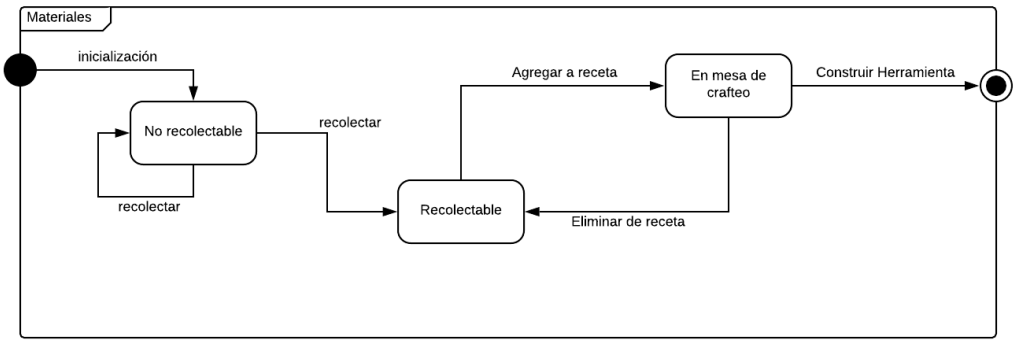

Figura 6.2: Diagrama de estado de los materiales.

A continuación podemos observar el diagrama de estados de las recetas. Ellas son las encargadas interpretar lo que el jugador le pide, usando los recursos que le ofrece para la creación de alguna de las herramientas disponibles, teniendo como intermediario entre la receta y el jugador a la mesa de crafteo. Sus estados son muy simples. Se inicializa en el estado por defecto "Patrón vacío con constructor vacío". A medida de que el usuario va ingresando o eliminando materiales en un patrón personalizado (métodos encapsulados en el evento *.ªlterar receta"*), la receta puede cambiar al estado *"Patrón personalizado válido"* o *"patrón personalizado inválido"* dependiendo de si hubo coincidencia o no con alguna herramienta disponible. Sólo en el caso de que la receta sea válida, se puede acceder a *çonstruir herramienta"* volviendo al estado inicial vacío. Desde cualquier estado, se finaliza la secuencia de estados cerrando el juego.

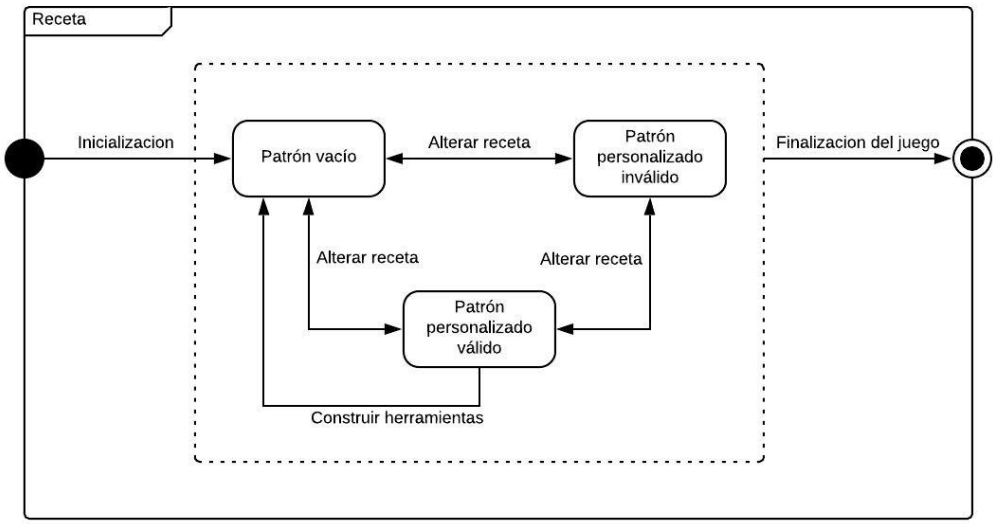

Figura 6.3: Diagrama de las recetas.

7. Diagramas de paquete

En la figura 7.1 se muestra el diagrama de paquetes de la aplicación con sus dependencias internas más importantes. En general, los paquetes de excepciones suelen depender de todos los demás dentro de un paquete, ya que engloban todas las excepciones que se pueden lanzar para un dado paquete, por lo que no se especifica dicha dependencia con una flecha. Se puede ver como la vista y el controlador estan relacionados, conociendose entre si, y estos a la vez conocen al modelo.

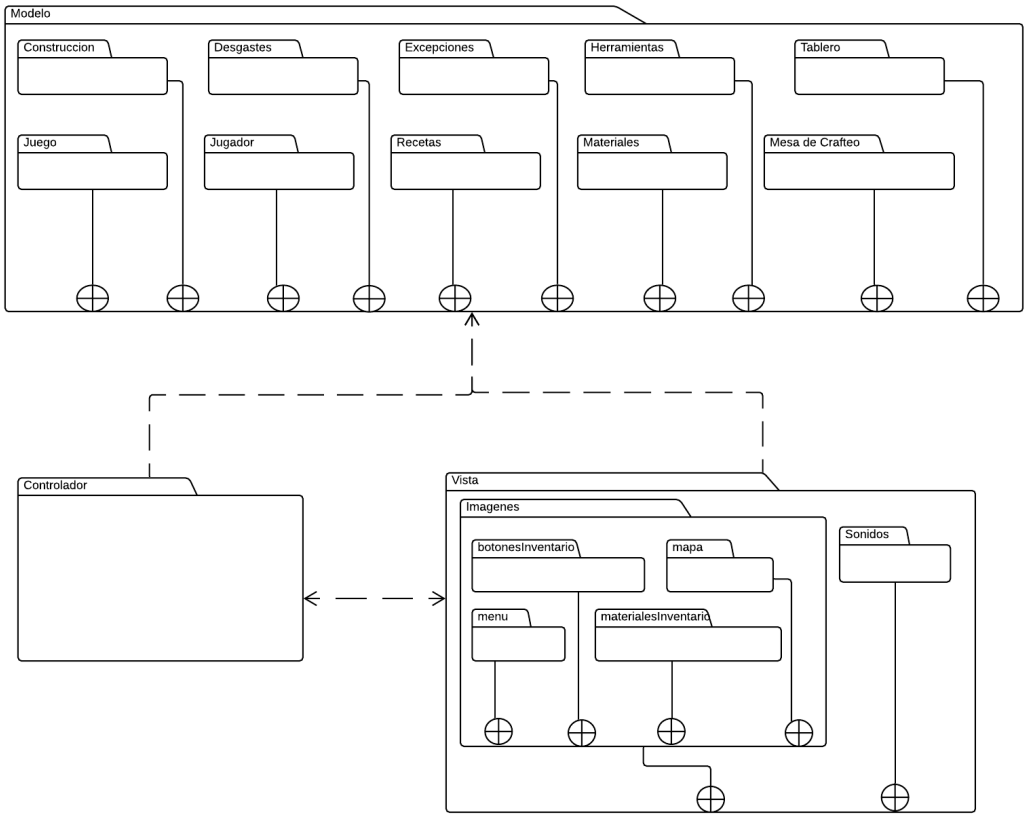

Figura 7.1: Diagrama de paquetes.

8. Excepciones

En esta sección hablaremos de las diferentes excepciones que fueron usadas en el programa, listándolas y explicando cuando ocurren cada una de ellas ya que varías son utilizadas para diferentes casos a través del programa. Cada uno de estos errores se manifiesta con una "alerta" en la vista, avisándole al usuario que ocurrió. De la manera que se manejó, dependiendo el caso en donde apareció la excepción, la alerta mostrada es diferente.

- **CasilleroOcupado**
  Este error se lanza cuando se quiere acceder, o colocar un material, a una posicion ya ocupada.
- **ExcedeLimiteDeMapa**
  Ocurre cuando se quiere acceder mas allá del mapa ya establecido.
- **MaterialIngresadoEnPosicionInvalida**
  Es lanzada cuando a una receta se le intenta ingresar un material en una posicion del patron que ya contiene materiales.
- **RecetaEliminarMaterial**
  Excepcion lanzada cuando a una receta se le intenta remover un material en una posicion del patron que no contiene materiales.
- **RecetaInvalida**
  Es lanzada cuando, en la mesa de crafteo, se tiene una receta, con sus respectivas herramientas, y se presiona el boton para crear herramienta, pero esta no coincide con ninguna de las recetas de las herramientas que se pueden crear.
- **SinHerramientaEquipada**
  Se lanza cuando el usuario intenta utilizar una herramienta pero no tiene ninguna equipada en ese momento.

9. Patrones de Diseño

A continuación mencionamos los patrones utilizados:

- **Singleton**
- **Abstract Factory**
- **Strategy**
- **State**

10. Conclusión

Se desarrolló una aplicación que implementa de manera el juego Minecraft utlizando el lenguaje de programación Java y la plataforma JavaFX para la interfaz gráfica. Se estudiaron diferentes patrones de diseño que resultaron de utilidad para modelar varias partes del programa.
Por otro lado, el uso del patrón de arquitectura MVC nos permitió desarrollar independientemente el controlador y la vista a pesar de que se realizaban en paralelo varias refactorizaciones del modelo. Por último se logró aprender cómo la programación orientada a objetos permite identificar entidades independientes minimizando el acople entre ellas y facilitando el desarrollo de aplicaciones de complejidad media a elevada.