Trabajo Práctico 2 — AlgoEmpires

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2018
Grupo T15

Alumno	Padron	Email
GIORDANO, Franco	100608	francogior98@gmail.com
ISOLA, Federico	100120	fedeisola2007@gmail.com
IRIBARREN, Alvaro	101049	alvaroiribarrenfiuba@gmail.com

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación del estilo 'Age of Empires' utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Algunas de las consideraciones que tomamos a la hora de modelar el trabajo fueron,

- Un casillero puede ser ocupado por una única entidad.
- Un jugador no puede poseer una cantidad de oro negativa.
- Los jugadores pueden ver las construcciones ajenas.
- Los edificios pueden crear mas de una unidad por turno.
- Un edificio nuevamente construido comienza con una fracción de su vida y el aldeano responsable lo construirá a medida que avanzan los turnos.
- Un nuevo edificio puede ser inmediatamente utilizado, no es necesario esperar hasta que termine su construcción.
- Si se le ordena una acción al aldeano, cancela lo que este haciendo y realiza lo ordenado.
- El tamaño del terreno es modificable, con un tamaño mínimo de 20x20 casilleros.

3. Modelo de dominio

Para generar un modelo como solución al problema se creo la clase **Jugador**, la cual debe conocer en donde está jugando y contra quién, sus unidades y edificios, su dinero disponible. Cada una de estas características es una clase. Primero el **Terreno**, lugar donde ocurre el desarrollo de la partida. Segundo las **Unidades** y **Edificios** que un jugador posee fueron modelados a partir de la clase **Entidad**. Por último cada jugador posee un **Monedero**, encargado de almacenar y manejar el dinero. Cada **Unidad** se puede decir que tiene tamaño '1' y por lo tanto ocupan un solo **Casillero**. Esto es distinto para los **Edificios**, los cuales pueden ocupar mas de un **Casillero**. Para esto se creó la clase **Región**, la cual es contenedora de todos los **Casilleros** pertenecientes a la zona. Un punto a tener en cuenta es que las **Unidades** pueden desplazarse por el mapa, pero solo de a un **Casillero** por turno. Para poder generar el movimiento, se debe indicar la **Dirección** en la cual quiere hacerlo. Evidentemente, están disponibles las 8 **direcciones** posibles, cada una con su respectiva clase.

Todas las clases se ven reunidas en **AlgoEmpires**, la clase principal del juego, que se encarga de la correcta gestión de turnos, y comunicación entre clases para que todo sea funcional.

4. Diagramas de clase

4.1. Edificios y unidades

Las clases principales de este modelo son el **Edificio** y la **Unidad**. Ambas sirven para modelar todo lo que necesitamos a partir de herencia. Así mismo estas dos, aunque no lo parezca, tienen cosas en común, por lo tanto, se modela una clase Entidad que cumple con nuestra querida regla "es un".

Primero se explicará la unidad. De esta hay dos tipos, unidades 'pacíficas', que en nuestro caso NO modelamos como un tipo aparte (ya que solo el **Aldeano** pertenece a este tipo) y las guerreras.

De la última se derivan 3 tipos de guerreros. Primero tenemos al arma de asedio(**ArmaDeAsedio**) la cual solo es capaz de atacar a los edificios. Segundo tenemos al **Arquero** el cual tiene un rango y puede atacar tanto a unidades como a edificios (con un daño distinto), y por último tenemos al **Espadachín** el cual comparte características con el arquero pero solo puede atacar de cerca.

Luego nos quedan las construcciones. Tenemos 3 clases: Plaza Central(**PlazaCentral**), **Castillo** y **Cuartel**. Todas estas cumplen con el hecho de ser un Edificio.

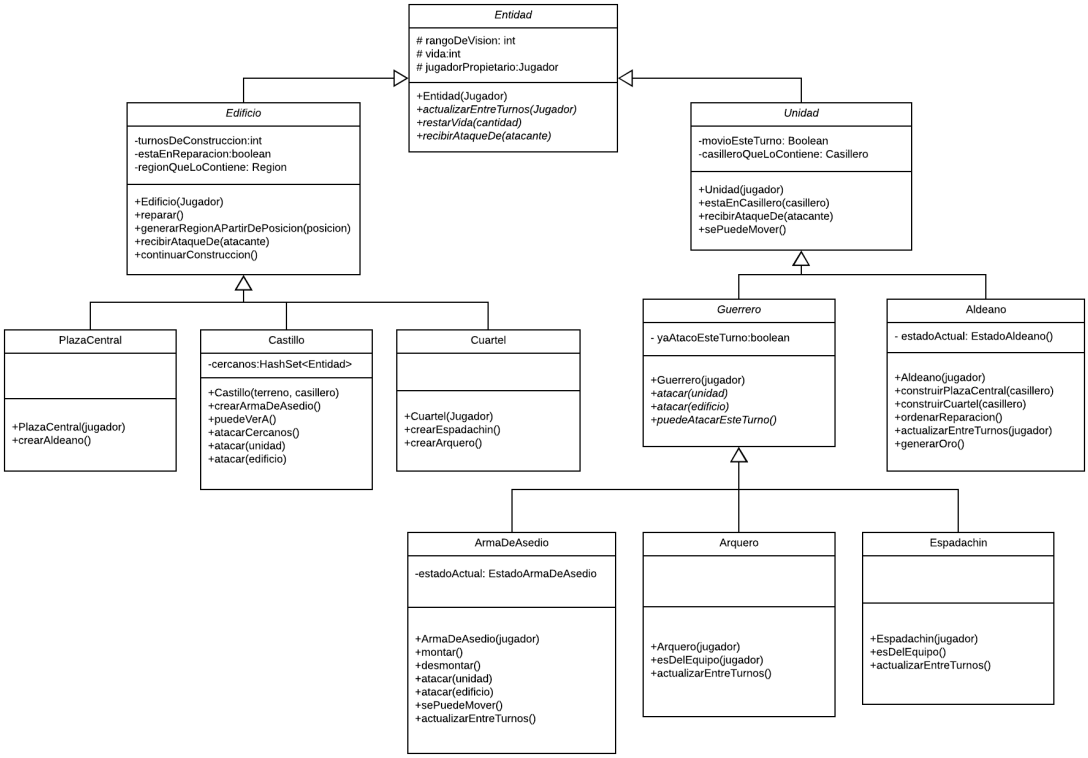

Figura 1: Edificios y unidades.

4.2. Dirección

Como se adelantó en la explicación del modelo, tenemos 8 direcciones disponibles para realizar un movimiento. Primero se creó la clase **Dirección**, madre a todos los tipos de direcciones. Lo único que varia entre las direcciones es la dirección (en coordenadas, pertenecientes a la clase **Posición**) horizontal y vertical que poseen. De esta forma se le puede indicar a una unidad de una forma muy cómoda hacia donde moverse, dejando una mayor claridad en el código.

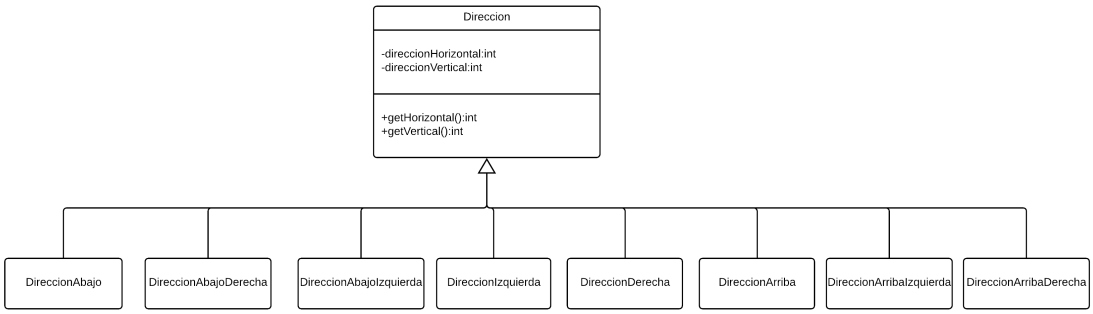

Figura 2: Direcciones.

5. Detalles de implementación

5.1. Movimiento de unidades

Para poder mover una unidad primero se le debe indicar en que dirección hacerlo, como se explico en el segundo diagrama. La unidad podrá realizar este movimiento siempre y cuando no se haya movido en el turno y el casillero al cual se intenta desplazar sea válido. Una vez chequeadas estas condiciones, el terreno donde se desarrolla el juego, del cual la unidad posee una referencia, mueve a la unidad al casillero deseado. Cada vez que se hace un movimiento se activa una flag la cual indica que la unidad ya realizó su respectivo movimiento en el turno.

```
public void desplazarHacia(Direccion direccion)
throws CasilleroInvalidoException, UnidadYaMovioEnEsteTurnoException {

        if (movioEsteTurno) {
            throw new UnidadYaMovioEnEsteTurnoException();
        }

        terrenoDeJuego.moverUnidad(casilleroQueOcupo, direccion);
        this.casilleroQueOcupo = casilleroQueOcupo.generarMovimientoHacia(direccion);
        movioEsteTurno = true;
    }
```

5.2. Redefinición en HashMap

Para implementar el terreno necesitamos una estructura capaz de darnos la entidad que ocupa un Casillero. Perfectamente pudo haber sido una lista, pero la búsqueda de un simple elemento puede tomar mucho tiempo si es mucha la cantidad. Decidimos solucionarlo con un HashMap, de casilleros y posiciones.

Pero surgieron otros problemas. Primero, las keys que este posee no pueden mutar a lo largo de la ejecución ya que cuando se agregan los elementos se les asigna un 'HashCode', el cual es fijo. Esto se solucionó creando todos los casilleros previamente y agregarlos al terreno. Segundo, el mapa no sabe comparar Posiciones, por lo tanto se necesitó "Overridear" el método 'equals' para que fuera capáz de hacerlo. Finalmente queda algo del tipo:

```
@Override
    public boolean equals(Object otro) {
        boolean result = false;
        if (otro instanceof Casillero) {
```

```
            Casillero that = (Casillero) otro;
            result = (this.posHorizontal == that.getHorizontal()
            && this.posVertical == that.getVertical());
        }
        return result;
    }

    @Override
    public int hashCode() {
        return (41 * (41 + posHorizontal) + posVertical);
    }
```

6. Excepciones

AleanoOcupadoException Se lanza si el aldeano todavía tiene turnos restantes para finalizar alguna tarea.

ArmaDeAsedioNoPuedeAtacarSinEstarMontadaException Se lanza si se le ordena un ataque a una arma de asedio sin estar antes montada.

ArmaDeAsedioNoPuedeAtacarUnidadesException Comportamiento esperado según la especificacion.

DimensionesInvalidasException Al momento de crear un terreno, si las dimensiones son inferiores a 20x20.

EdificioNoFuncionlException Aparece cuando se quiere realizar una acción con un edificio que está en construcción o no finalizado.

EntidadFueraDeRangoException Encontrada cuando un atacante intenta dañar otra entidad fuera de su alcance.

GuerreroYaAtacoEsteTurnoException Por especificación, una unidad guerrera puede atacar una sola vez por turno.

NoSePuedeInteractuarConEntidadesEnemigasException Cuando un jugador quiere jugar con las entidades enemigas.

NoSeToleraFuegoAmigoException Una unidad/castillo no puede atacar a otra entidad del mismo equipo..

OroInsuficienteException Lanzada cuando no se tenga el dinero suficiente para concretar una acción (compra unidad/edificio).

PosicionDeCreacionFueraDeRangoException Se lanza cuando desee crearse un edificio o unidad mas lejos de lo permitido por la entidad responsable.

PosicionInvalidaException Hallada cuando se quiera crear una entidad en una posición no perteneciente al terreno, o se mueva una unidad a una posición no perteneciente/ya ocupada.

PosicionInvalidaException Cuando se elige una posición invalida para cierta tarea.

SeIntentoSuperarPoblacionMaximaException Comportamiento pedido por especificación, hay establecido un máximo de unidades por jugador y no puede superarse.

SoloUnAldeanoReparaALaVezException Especificado por consigna, un edificio no puede ser reparado por mas de un aldeano.

SoloUnidadesSePuedenDesplazarException Cuando se emite la orden de terreno.mover(entidad) a un edificio.

UnidadNoPuedeMoverseException Se intento desplazar una unidad que no se encuentra en un estado movible (ya se movió este turno, por ejemplo).

7. Diagramas de secuencia

En estos diagramas se mostrarán solamente los casos exitosos, es decir, casos en los que el usuario realiza las acciones correctamente evitando la excepciones.

7.1. Desplazamiento de Unidad

Para mover una unidad el encargado es el Terreno, ya que el es que maneja posiciones. Por lo tanto se le pide que obtenga el casillero al que me quiero desplazar, a partir de una dirección. Una vez obtenido el casillero al que nos queremos desplazar, es cuestión de decirle informarle al casillero actual de la unidad que nos queremos trasladar al nuevo casillero.

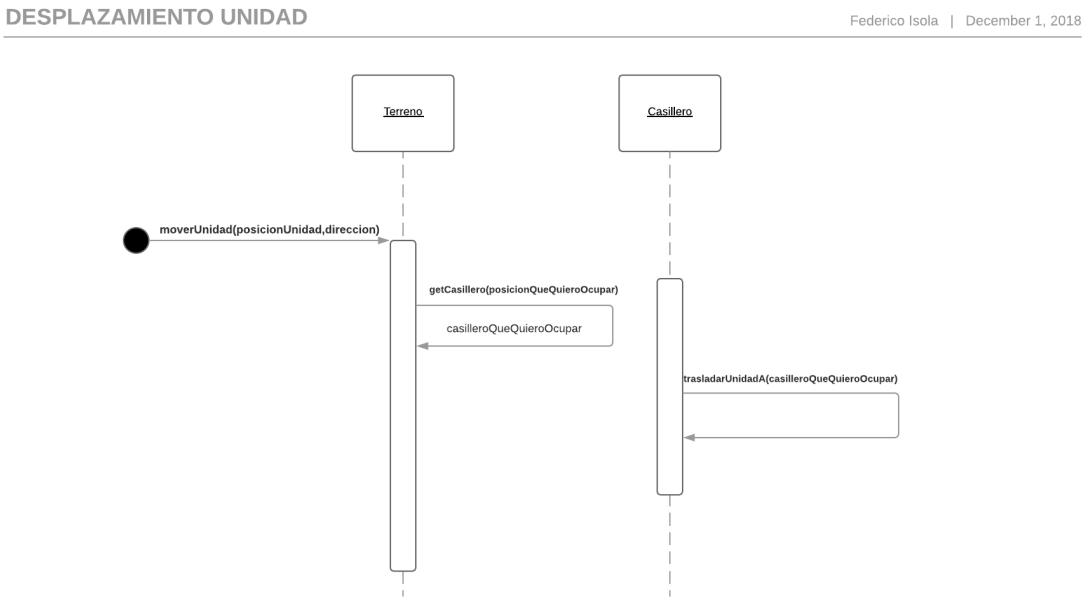

Figura 3: Desplazamiento de la unidad

7.2. Reparación de Aldeano

Los responsables de la reparación de los edificios son los aldeanos, para pedirles que realicen la tarea es necesario decirles en qué posición realizarla. El llamado lo hace el propio jugador. Internamente, el aldeano le delega la tarea a la clase terreno, ya que conoce cada uno de los casilleros. Esta busca la posición en la cual debe realizar la tarea y luego le pide a ese mismo edificio que se repare. Para simplificar el diagrama se agregó el 'buscarEdificioAReparar', que si bien no es un método explícito, es útil para agregar comprensión.

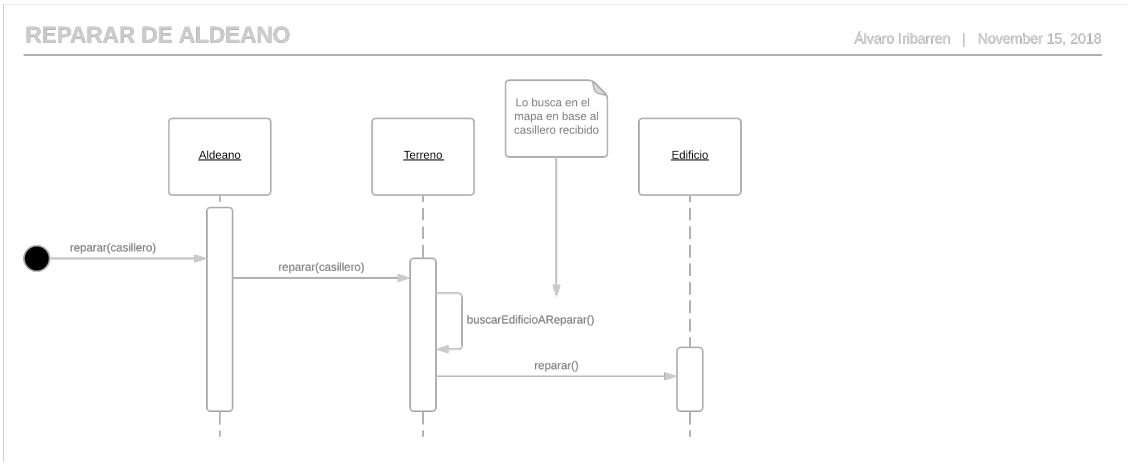

Figura 4: Reparación de edificios.

8. Diagrama de estado

En el siguiente diagrama se muestran las transiciones que atraviesa el aldeano a lo largo del juego a media que el jugador le ordena la realización de una acción.

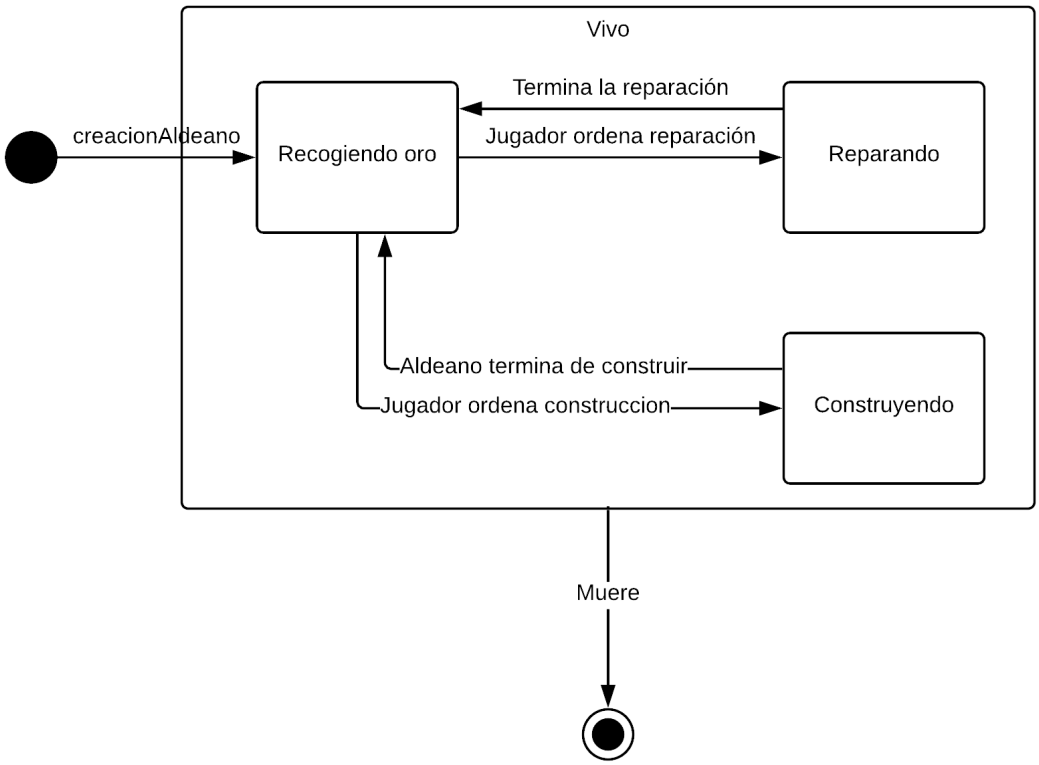

Figura 5: Estados del aldeano.