FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 - Algo Empires

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

Alumno	Padrón	Email
PONCE, Gastón Nicolás	99723	gastonrock8@hotmail.com
MAC GAUL, Pedro	101503	pedromacgaul@gmail.com
DE GIÁCOMO, Nicolás	99702	nicolas.de.giacomo@gmail.com

Índice

1. Introducción

El presente informe reúne la documentación de la solución del tercer Trabajo Práctico de la materia Algoritmos y Programación III. Este consiste en desarrollar un juego en Java, utilizando el paradigma orientado a objetos, basado en el clásico juego 'Age Of Empires'.

2. Supuestos

Los supuestos de este modelo son los siguientes:

- **Daño al Arma de Asedio montada**: Un Arma de Asedio montada se considera un Edificio al momento de recibir daño.
- **Construcción**: Una vez iniciada la construcción no se podrá detener.
- **Creación de Unidades**: El Usuario podrá elegir donde quiere crear la Unidad, siempre que se mantenga a una casilla de su Edificio creador.
- **Unidades Muertas**: Cuando una Unidad muere, de inmediato se elimina del Mapa.

3. Diagramas de Clase

A continuación se mostraran las clases implementadas, separadas por categoría.

3.1. Unidades

El modelo cuenta tres clases abstractas y siete concretas. Las clases concretas representan cada Unidad existente en el juego, mientras que las clases abstractas las agrupan según su comportamiento compartido.

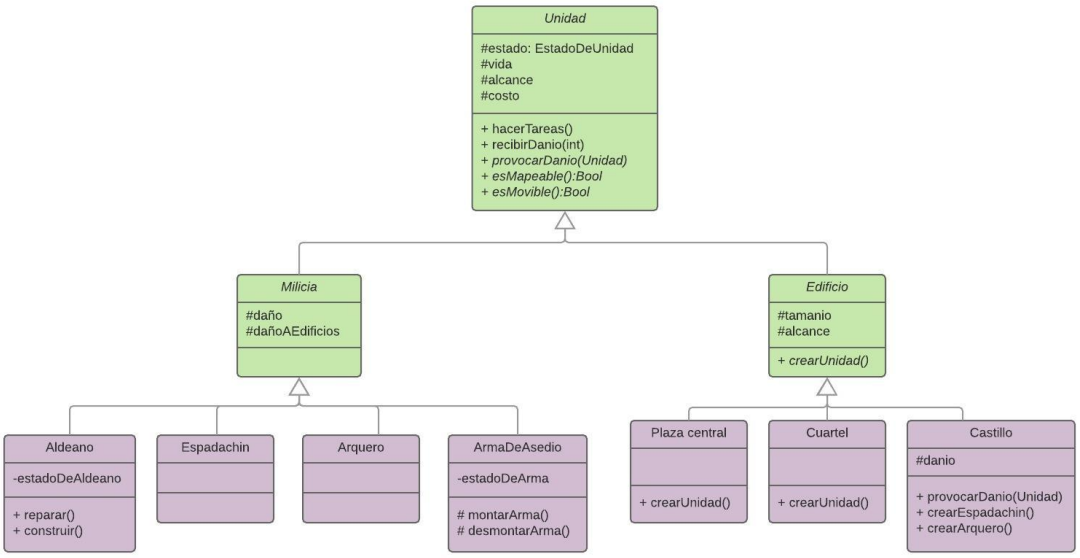

Figura 1: Diagrama de Unidades.

4. Diagrama de Paquetes

El siguiente diagrama muestra los paquetes creados, resaltando las clases mas importantes y su relación.

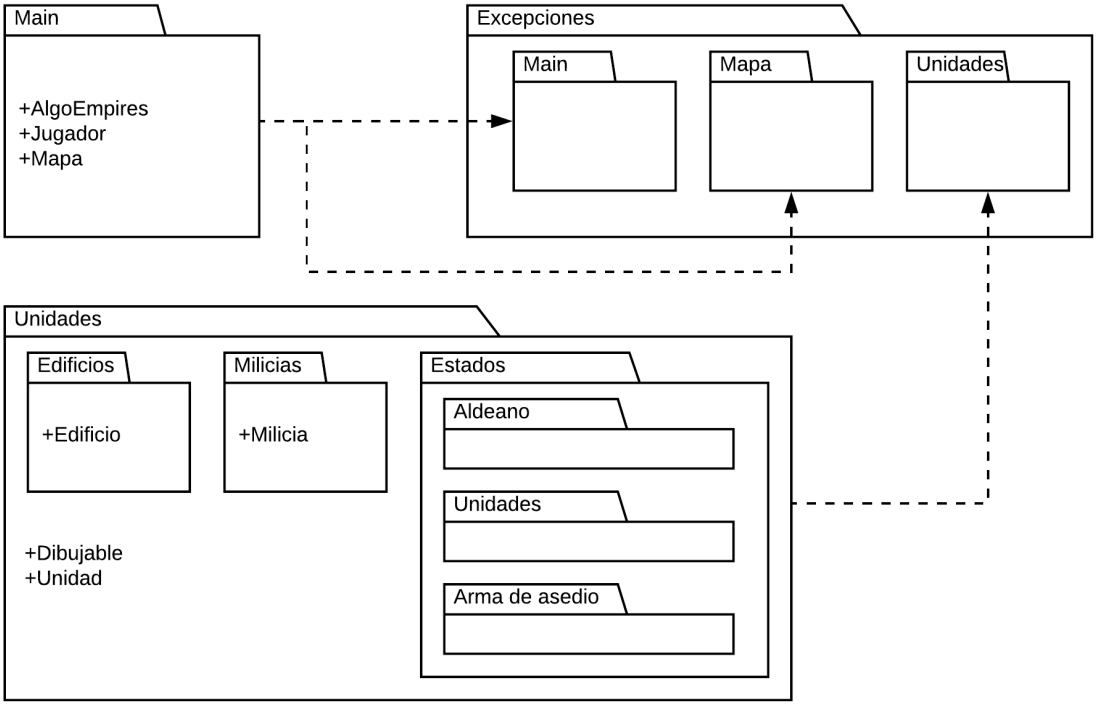

Figura 2: Diagrama de paquetes.

5. Diagramas de Secuencia

5.1. Castillo atacando unidades cercanas

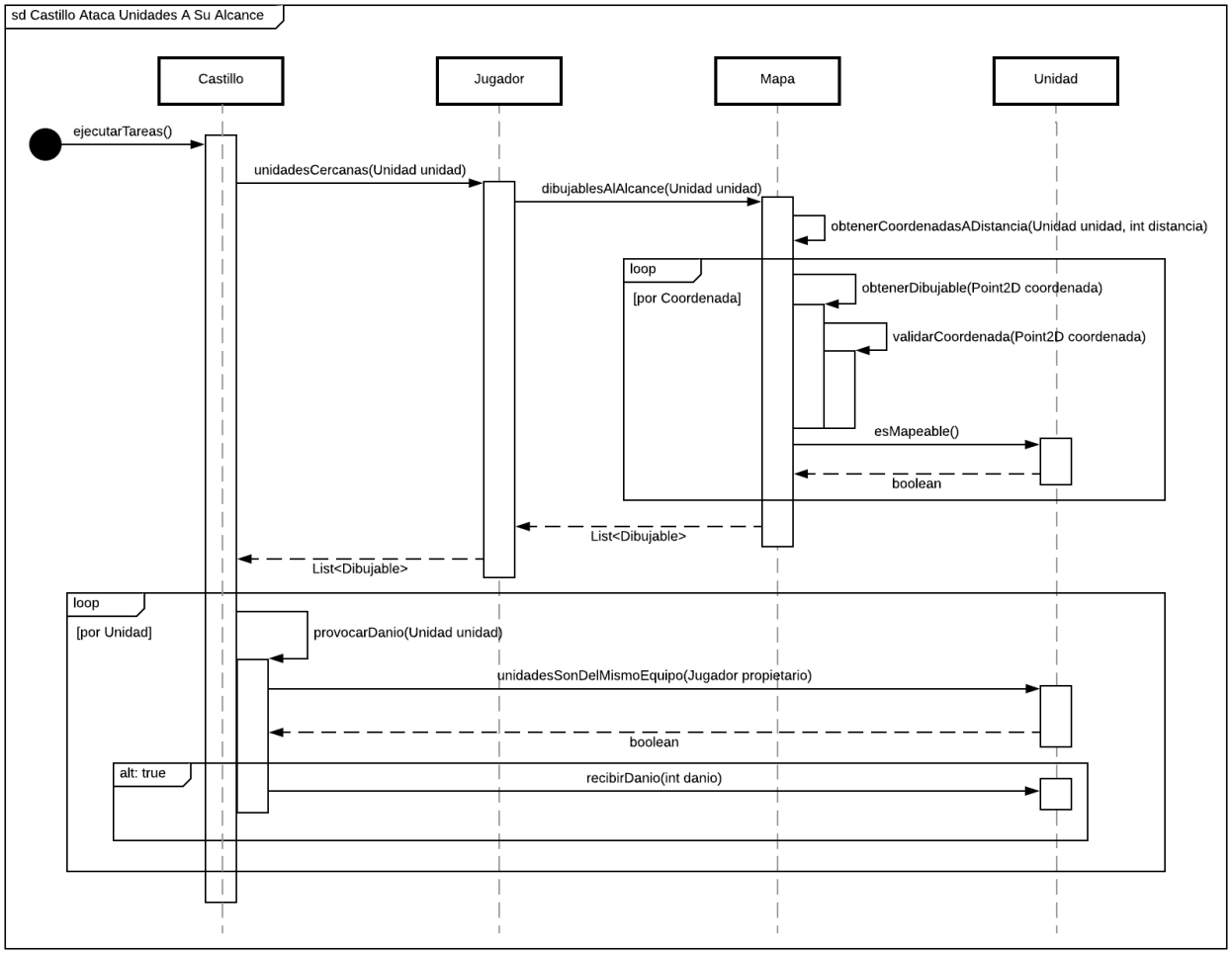

Figura 3: Diagrama de secuencia del ataque del Castillo.

5.2. Edificio creando Milicia

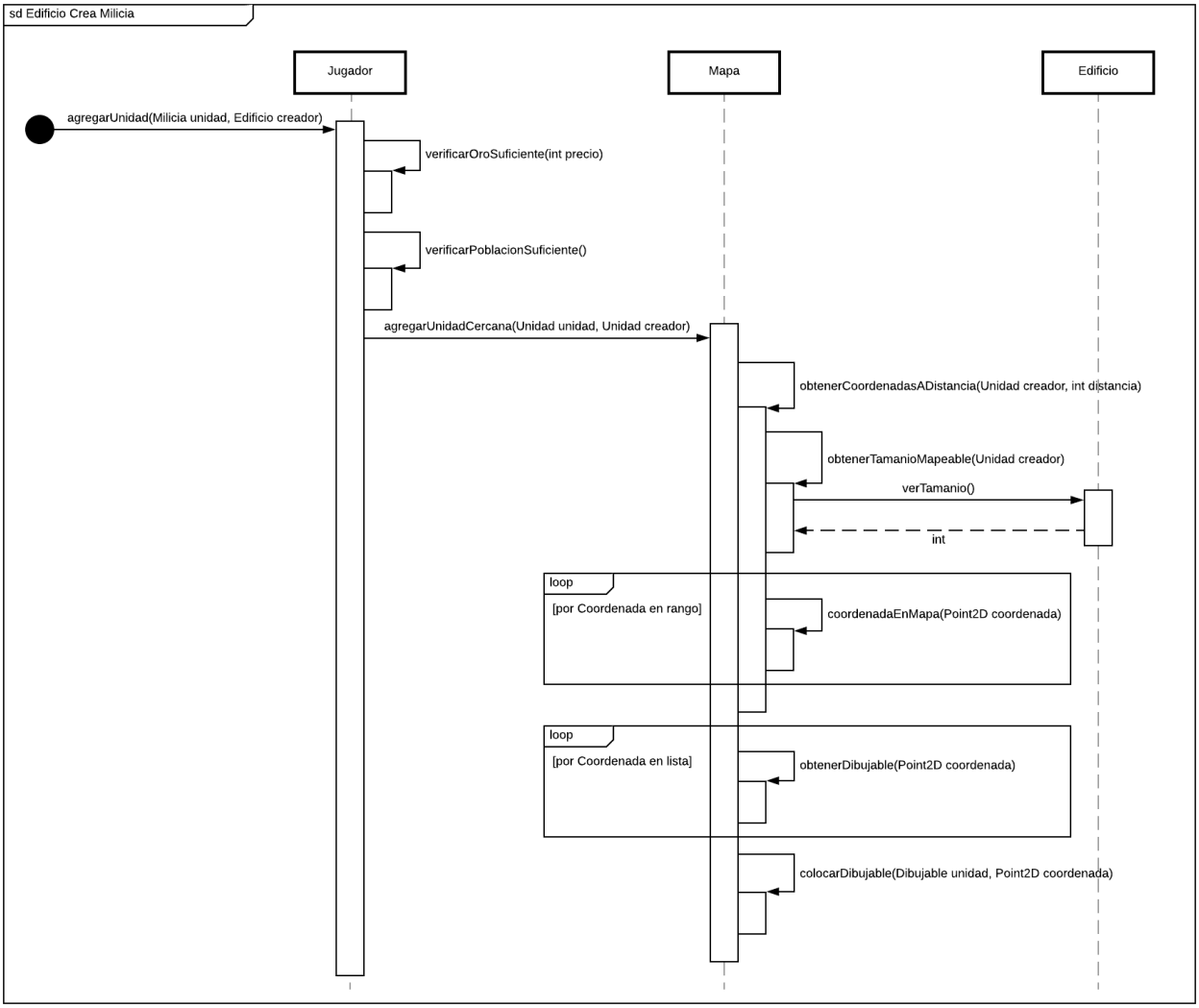

Figura 4: Diagrama de secuencia la creación de Milicia.

6. Diagramas de Estado

A continuación se mostraran la variación en los estados de las clases, separadas por categoría.

6.1. Aldeano

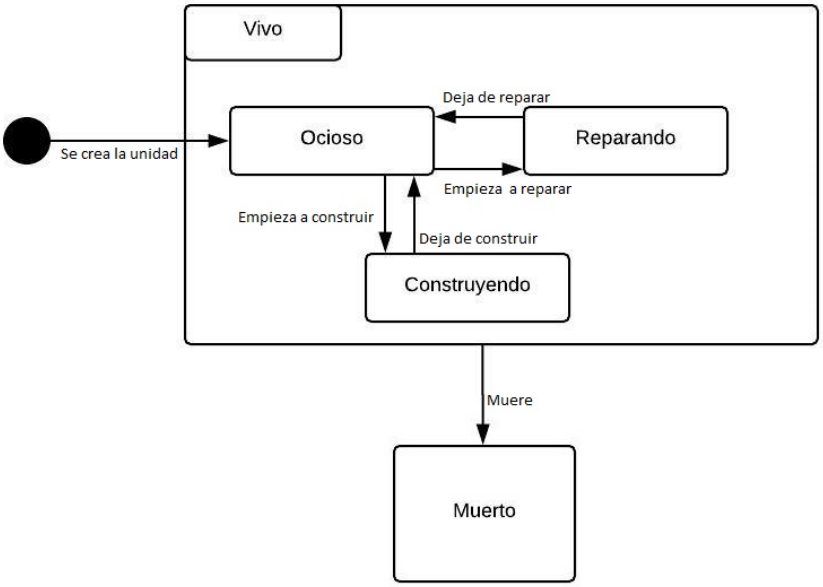

Figura 5: Diagrama de estado del Aldeano.

6.2. Edificio

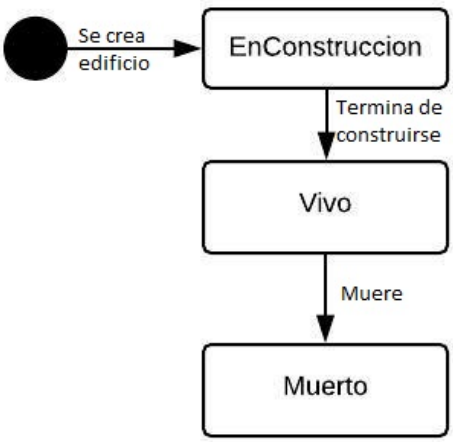

Figura 6: Diagrama de estado del Edificio.

6.3. Arma de Asedio

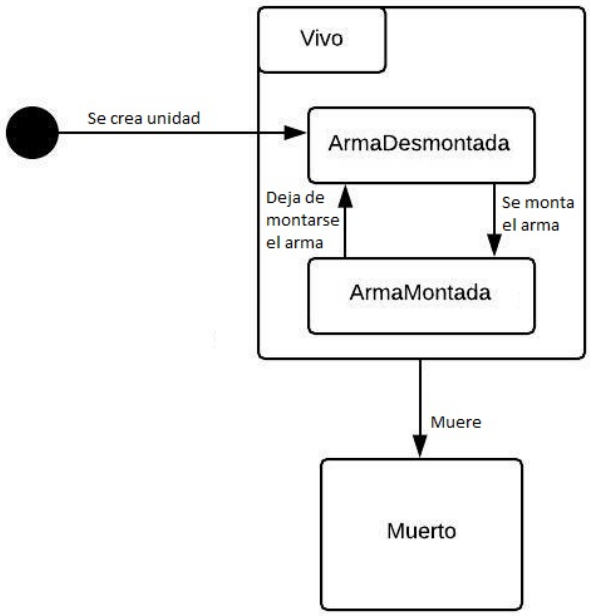

Figura 7: Diagrama de estado del Arma de Asedio.

7. Detalles de Implementación

7.1. Clase Mapa

Se eligió llevar a cabo el trabajo con una clase Mapa que tenga la responsabilidad de todo lo que esté relacionado a los movimientos, colocación y posiciones de las Unidades a lo largo del Juego. Además tomando la decisión de que solo la clase AlgoEmpires sea la que se comunique con el Mapa.

A lo largo de la duración del Juego, el Mapa se encarga de comprobar que todas las acciones que realiza, y todas las coordenadas que se le pasan y que calcula, sean válidas. Verificando así que no se pisen Unidades, que no se realicen acciones fuera del Rango del Mapa, y que se vea reflejado de manera correcta el estado del Juego.

Por ejemplo: Si una Unidad es atacada y se muere, su estado cambia y deja de ser 'Mapeable', por lo que la próxima vez que el Mapa devuelva esa referencia en la celda, la debe quitar y devolver en NULL. De la manera:

```
Dibujable obtenerDibujable(Point2D coordenada) throws FueraDeRangoException {
    validarCoordenadaEnMapa(coordenada);

    Dibujable dibujable = mapa[coordenada.getX()][coordenada.getY()];

    if (dibujable == null || !((Unidad) dibujable).esMapeable()) {
        quitarUnidad((Unidad) dibujable);
        return null;
    }
```

```
    return dibujable;
}
```

Estos controles son esenciales para el correcto flujo del Juego, y definen un marco para la representación gráfica que requiere el Jugador para planear y ejecutar sus turnos.

7.2. Unidades

Nuestro modelo de Unidades cuenta con una interfaz (*Dibujable*), tres clases abstractas (*Unidad*, *Edificio* y *Milicia*) y siete clases concretas (*Aldeano*, *ArmaDeAsedio*, *Arquero*, *Espadachin*, *Castillo*, *Cuartel*, *PlazaCentral*). Este modelo permite que todo comportamiento compartido quede ubicado en alguna de las clases abstractas, mientras que la interfaz dibujable permite que las unidades sean utilizadas en el mapa.

La razón de la existencia de la interfaz *Dibujable* es la siguiente: Supongamos que se quiere agregar al mapa del juego un obstáculo (ej: una piedra, un lago), estos obstáculos no se comportan como unidades pero sí deben aparecer en el mapa. Esto nos llevó a crear una interfaz para ser implementada por cualquier objeto que necesite mostrarse en el mapa.

La clase abstracta *Unidad* contiene el comportamiento compartido entre milicias y edificios, esto es es el manejo de la vida, y de los estados principales (*Vivo*, *Muerto*).

La clase abstracta *Milicia* contiene el comportamiento compartido entre las milicias, esto es el manejo del tamaño, del alcance y de la capacidad de atacar a otras unidades.

La clase abstracta *Edificio* contiene el comportamiento compartido entre los edificios, esto es el manejo del estado *EnConstruccion* y fallar cuando se intenta atacar con un edificio (el *Castillo* es el único edificio que ataca, pero lo hace automáticamente).

Las clases abstractas implementan comportamiento particulares, por ejemplo la construcción en los aldeanos o el estado *ArmaMontada*/*ArmaDesmontada* en las armas de asedio.

8. Excepciones

8.1. Excepciones del programa principal

ComienzoDePartidaException Utilizado cuando hay un error al comenzar la partida (ej: La partida ya fue comenzada).

LimiteDePoblacionException Utilizado cuando se intenta crear una Milicia pero la población llegó a su límite.

NombreRepetidoException Utilizado cuando se intenta agregar un jugador con un nombre ya utilizado.

NumeroDeJugadoresException Utilizado cuando se quieren agregar más jugadores de los permitidos, o menos de los necesarios.

OroInsuficienteException Utilizado cuando un jugador no posee el oro necesario para la acción requerida.

8.2. Excepciones del mapa

UnidadNoMovibleException Utilizado cuando se intenta mover una unidad fija.

CoordenadaInvalidaException Utilizado cuando se quiere realizar una acción sobre una coordenada inválida (ej: Construir sobre un lugar ya ocupado).

8.3. Excepciones del las unidades

AldeanoOcupadoException Utilizado cuando se intenta reparar o contruir con un aldeano que ya está realizando una acción.

ArmaDeAsedioYaDesmontadaException Utilizado cuando se intenta desmontar un arma de asedio ya desmontada.

ArmaDeAsedioYaMontadaException Utilizado cuando se intenta montar un arma de asedio ya montada.

AtaqueIncorrectoException Utilizado cuando se intenta realizar un ataque que no es válido (ej: Atacar un aldeano con un arma de asedio).

CreacionDeCastilloException Utilizado cuando se quiere crear un castillo.

UnidadNoEspecificadaException Utilizado cuando no se especifica la unidad que se quiere crear (ej: El castillo puede crear dos unidades).