Trabajo Práctico 2 — AlgoChess

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

Padron	Alumno	Email
91076	Sherly Porras	sherfiuba@gmail.com
102396	Juan Pablo Fresia	juanpablofresia@gmail.com
100589	Verónica Beatriz Leguizamón	veronicaleguizamon@outlook.es

Tutor: Eugenio Yolis.
Fecha de entrega final: 5 de Diciembre.

Índice

1. Introducción

El presente informe reune la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar un juego: AlgoChess

2. Supuestos

Jugador puede comprar la cantidad unidades que desee (incluso si son del mismo tipo). El jugador comienza la partida con 20 puntos y no hay forma de aumentar su ganancia.

Al atacar una unidad a otra, se debe verificar si es enemiga.

Por el momento se considera que al curar una unidad 'curable' no existe un máximo de puntos de vida que puedan tener. Es decir que si un soldado es creado con 100 puntos de vida podrá ser curado para tener 115. Es probable que ésto se cambie en una versión futura.

3. Diagramas de clase

El modelo se basa principalmente en la utilización de un objeto Tablero, el cual posee una lista de entidades que se encuentran en el mismo. Además, utiliza un objeto del tipo Sector para conocer los límites de sus casillas, las cuales se ven expresadas como coordenadas (a través de la clase Coordenada).

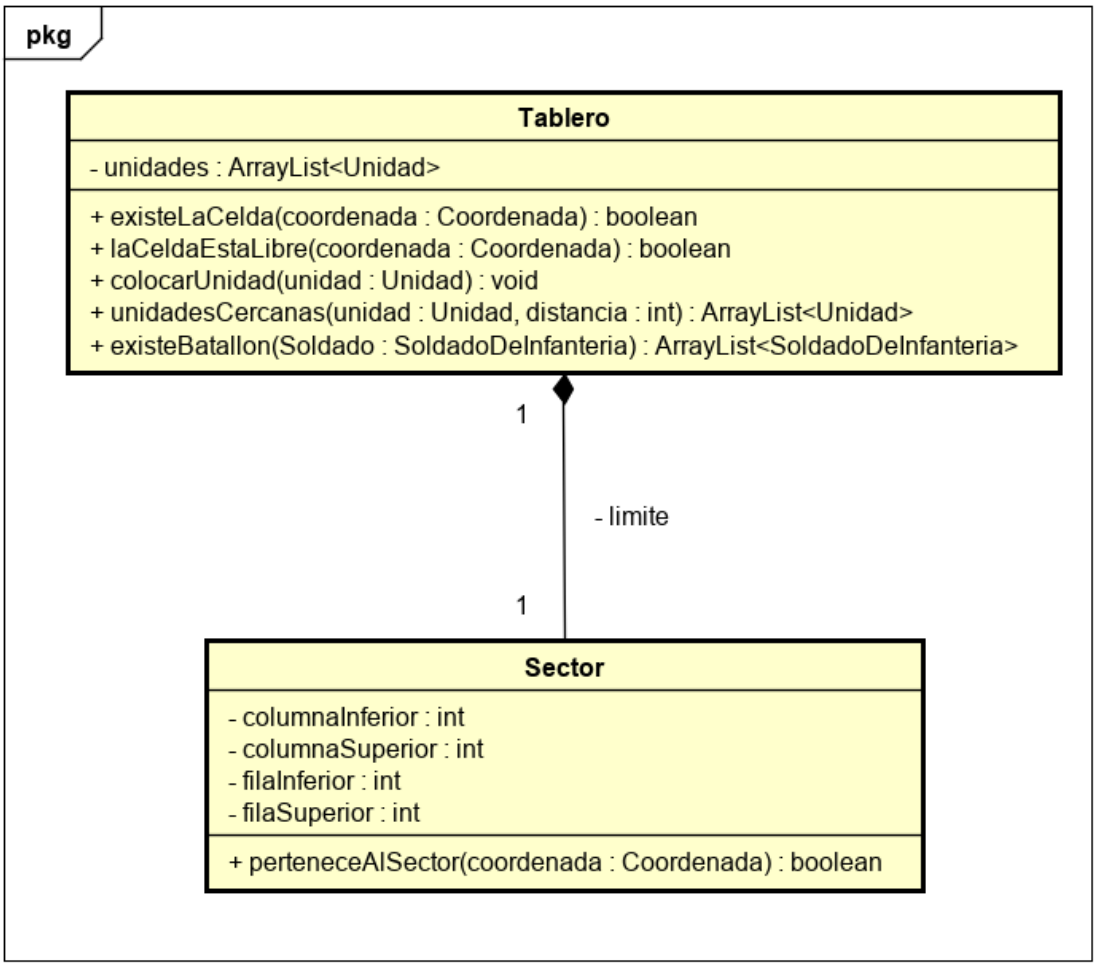

Figura 1: Diagrama del Tablero.

Por otro lado, se cuenta con objetos de la clase Jugador, los cuales poseen el nombre del mismo y llevan un recuento de los puntos que puede invertir en entidades. Además cada uno de ellos contiene un objeto de la clase Equipo que se encarga de almacenar las entidades que compró.

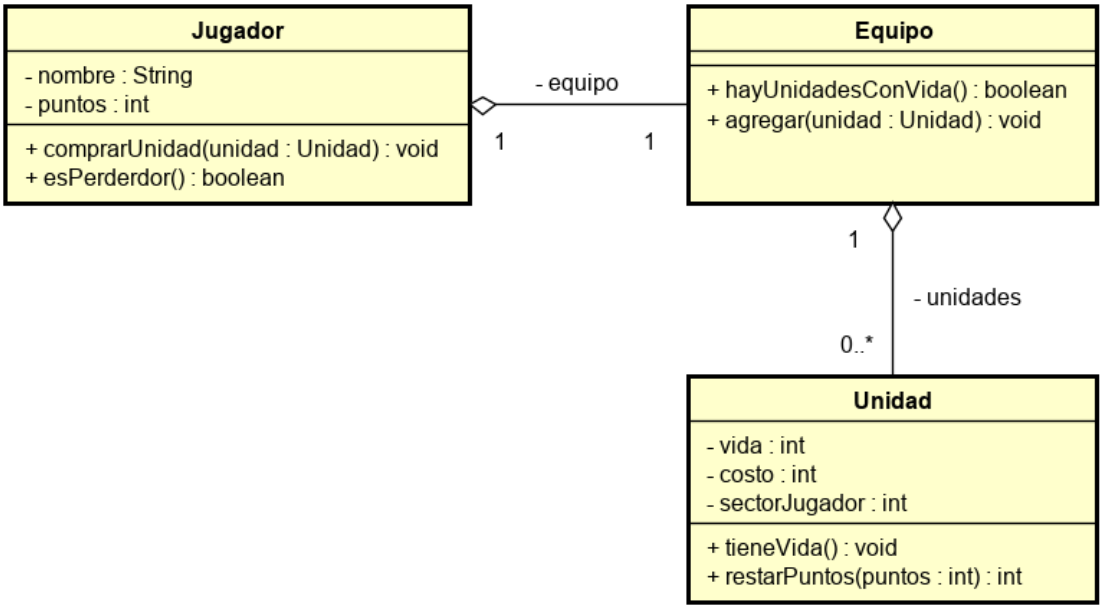

Figura 2: Diagrama del Jugador.

Utilizamos una super clase Unidad, de la que heredan UnidadMovible (que serán aquellas unidades que se puedan mover) y Catapulta. De UnidadMovible a su vez heredan las clases SoldadoDeInfanteria, Jinete y Curandero. Implementamos también dos interfaces: Atacante (unidades que pueden atacar, serán los soldados, jinetes y catapultas) y Curable (son aquellas que pueden ser curadas por un curandero, es decir soldados, jinetes y curanderos en sí).

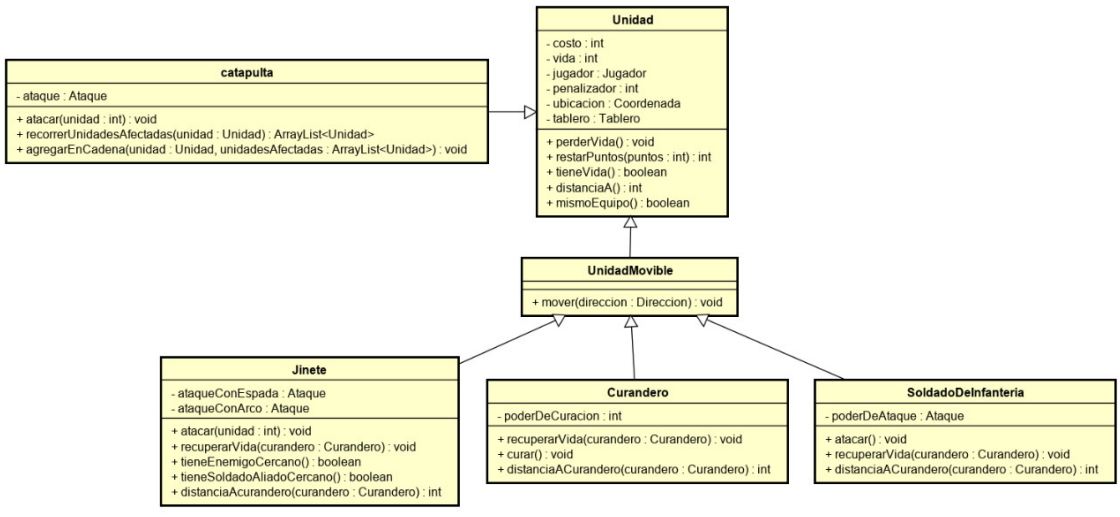

Figura 3: Diagrama de herencia de Unidades.

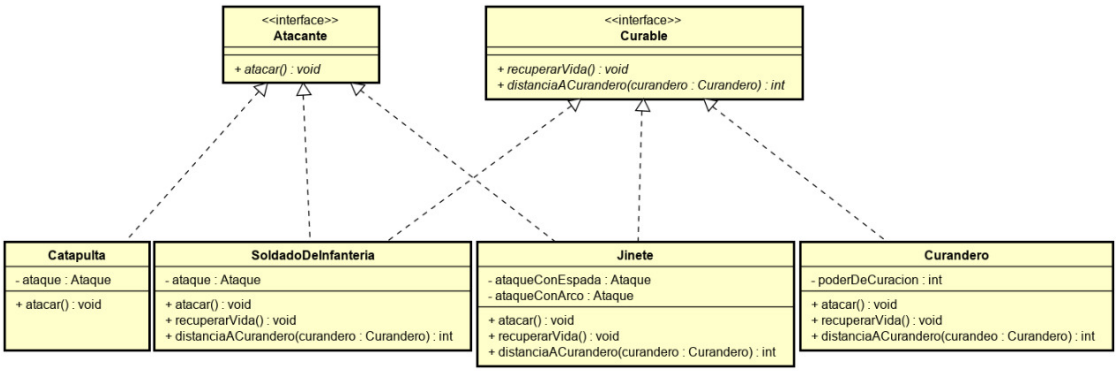

Figura 4: Diagrama de interfaces de Unidades.

Para el funcionamiento de los movimimientos y ataques instruidos directamente sobre las unidades, vimos necesario que Unidad conozca su Coordenada y al Tablero del juego. Las unidades atacantes poseen como atributo un objeto del tipo Ataque, quien se encarga de verificar condiciones respectivas al daño a realizar.

4. Detalles de implementación

4.1. Estrategia de trabajo

El primer gran problema que nos encontramos al trabajar con el código fue la división de tareas, supimos comprender que solamente se necesitaba la firma de los métodos y que devolvería cada uno para comenzar a trabajar, haciendo las pruebas unitarias de los mismos con el uso del mocks.

Mediante la utilización del repositorio en GitHub, fuimos armando el código en partes, siempre procurando tener pruebas unitarias acompañando las clases y luego desarrollando pruebas de integración.

5. Excepciones

Cada excepción posee un nombre autodescriptivo acorde a la situación. Por el momento, cada clase se encarga de lanzarlas al encontrarse con una situación excepcional que no pueden controlar por si mismas. Éstas serán resueltas más adelante cuando contemos con una interfaz gráfica para dar a entender al usuario que errores surgieron ante sus acciones.

PuntosInsuficientes Un Jugador no posee los puntos suficientes para adquirir una Unidad.

ObjetivoAliado La Unidad objetivo de un ataque es del mismo bando de la Unidad atacante.

UnidadNoPerteneceAlSector La Unidad creada no pudo colocarse en una Celda del Jugador contrario.

6. Diagramas de secuencia

A continuación se mostrarán algunas ejemplos de secuencias en la implementación.

En este diagrama observamos cómo ante el mensaje esPerdedor(), el Jugador pregunta a su Equipo si todavía queda alguna Unidad que se encuentre con vida. Como alguna de ellas dice que sí, el Jugador no pierde.

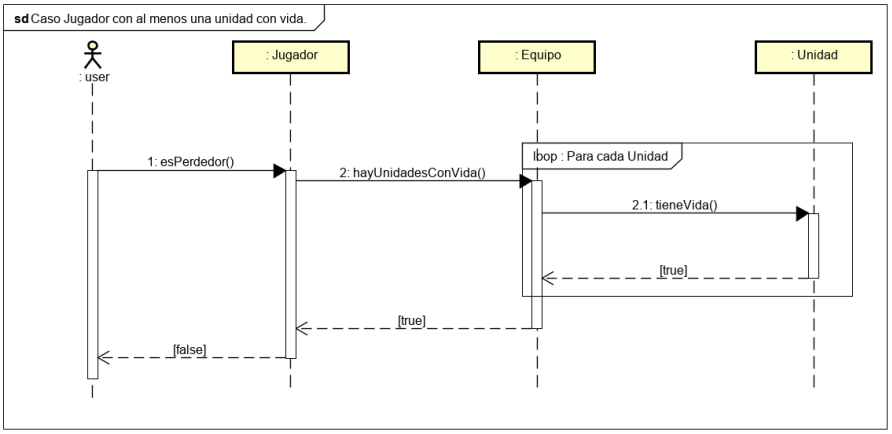

Figura 5: Jugador no ha perdido si todavía tiene alguna unidad viva.

En este caso, se le indica a un Jinete que se mueva hacia la Coordenada que tiene directamente arriba. Éste recibe la Direccion del movimiento, la cual se encarga de calcular la nueva Coordenada. Luego se le consulta al Tablero si es posible moverse hacia esa posición, a lo cual contesta que sí, por lo que Jinete actualiza su Coordenada.

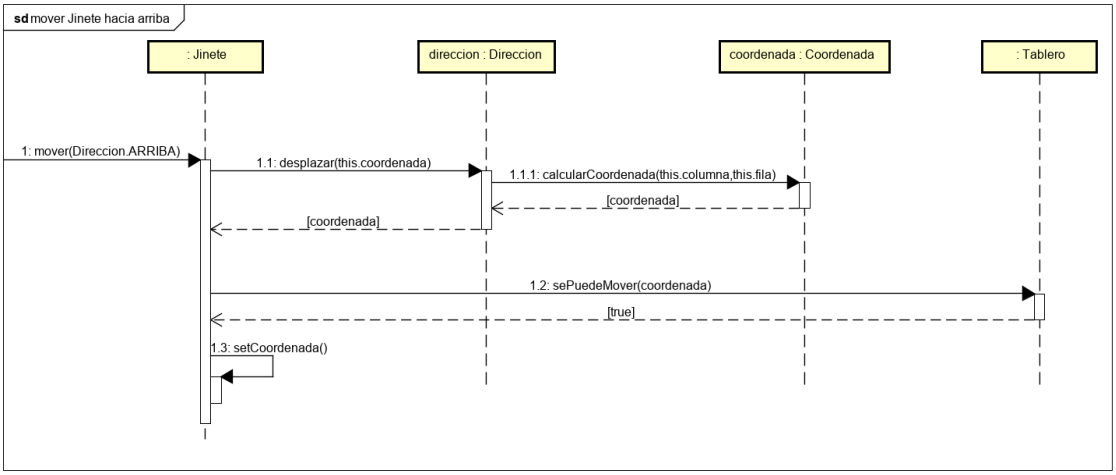

Figura 6: Jinete se mueve hacia arriba.

Aquí se puede observar que si la casilla a la que Jinete se quiso mover está ocupada, Tablero contestaría que el movimiento no es posible y Jinete no actualizaría su posición.

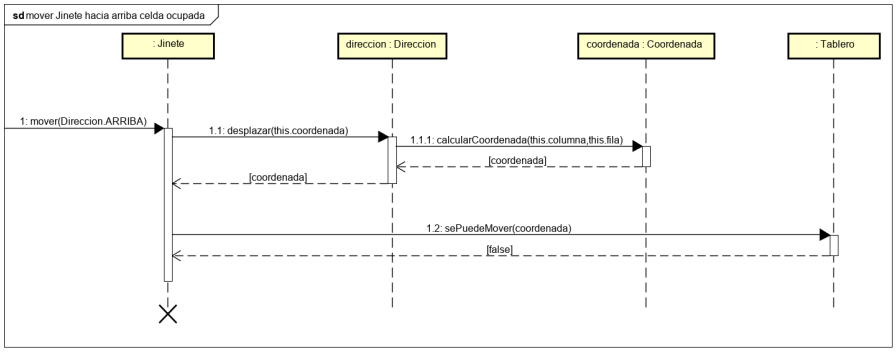

Figura 7: Jinete trata de moverse hacia arriba pero no puede.

En la siguiente figura se muestra la secuencia para agregar una Unidad (es decir, cualquier instancia de sus clases hijas) al Tablero durante el principio del juego. El Tablero verifica en primer lugar si existe la Coordenada donde se quiere colocar la Unidad. Luego revisa si ya existe alguna otra Unidad en esa posición. Por último, le pregunta a la Unidad misma si la casilla donde va a ser colocada es de su mismo Jugador. De cumplirse todas estas condiciones la Unidad es agregada correctamente al Tablero.

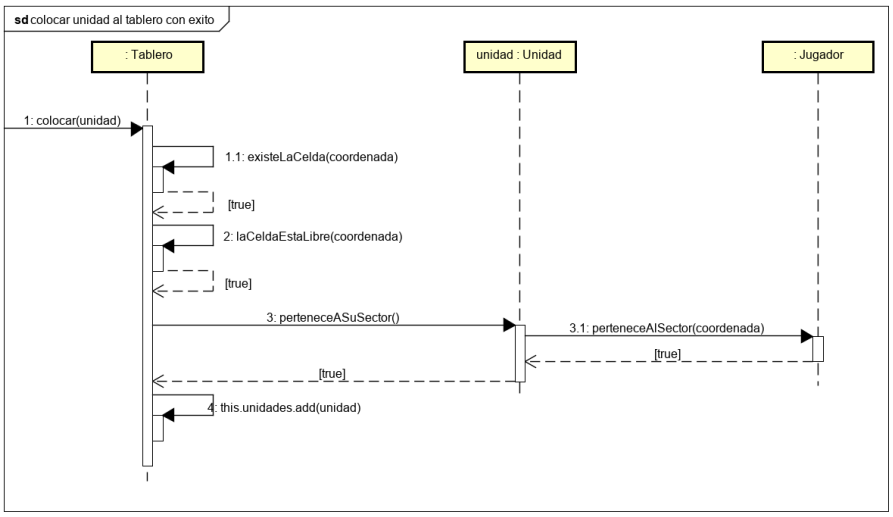

Figura 8: Colocar una pieza en el Tablero.

En este diagrama, se le ordena a un SoldadoDeInfanteria que ataque a un Jinete que se encuentra a distancia corta (1 a 2 casillas). En primer lugar verifica que la distancia al Jinete sea adecuada para realizar el ataque, luego utiliza su propio Ataque para realizar el daño. El objeto Ataque revisa si Jinete tiene penalizador por estar en el sector enemigo. Ante su respuesta, hace los cálculos necesarios y le resta los puntos de vida correspondientes.

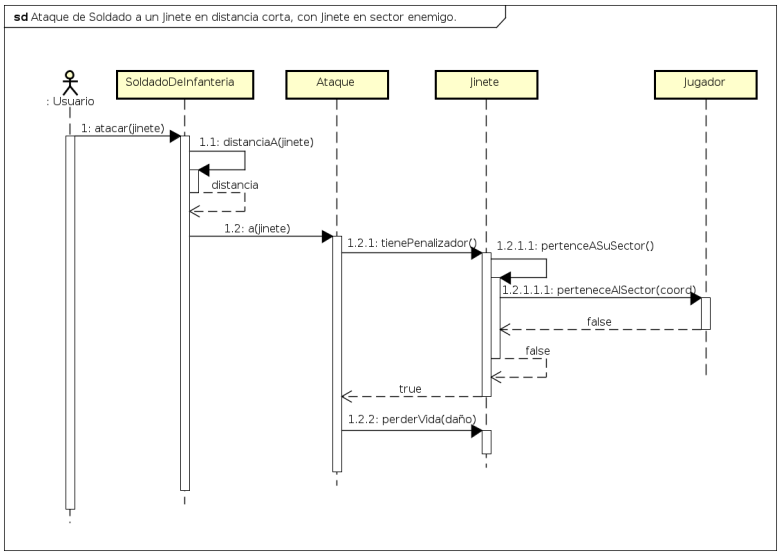

Figura 9: Soldado ataca a un Jinete.

En este otro caso, un Jinete está atacando a un Soldado enemigo a distancia media, teniendo otras unidades enemigas y por lo menos un soldado aliado a una distancia corta. Primero, al igual que en la figura anterior, se comprueba que la distancia de ataque sea correcta. Luego el Jinete le pide al Tablero que le indique las unidades que tiene en su cercanía. Con esta información, Jinete verifica si posee algun enemigo cerca, así como la existencia de algún soldado aliado que le permita atacar a distancia. Como se cumplen las condiciones, Jinete puede utilizar su ataque con arco en contra del Soldado enemigo objetivo. (a partir de que es envíado el mensaje a(soldado) al objeto Ataque, la secuencia se hace igual a la mostrada en la figura 9)

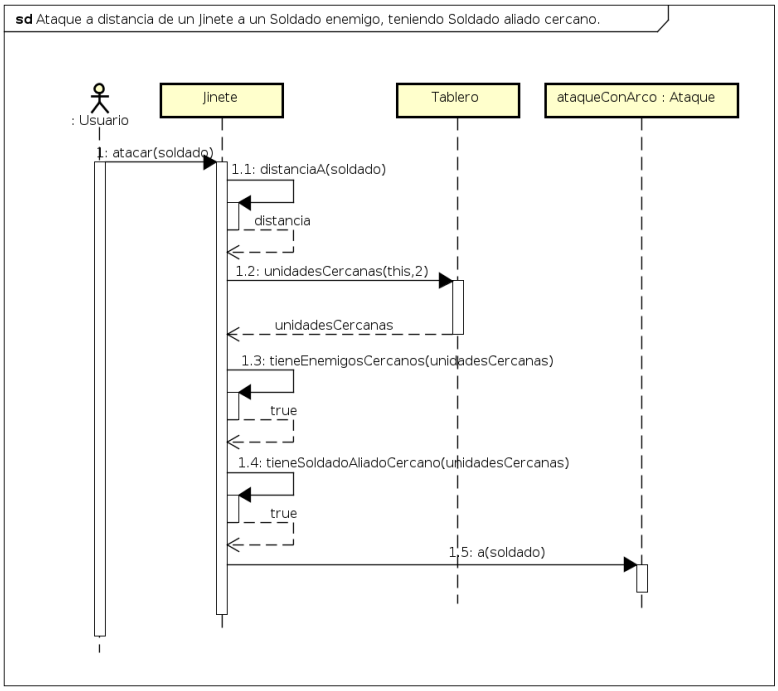

Figura 10: Jinete ataca a un Soldado enemigo utilizando su arco.

En esta otra figura se observa como una Catapulta ataca a un Jinete, y ésto provoca una sucesión de ataques en cadena a las unidades próximas al objetivo. Comienza, como siempre, verificando que la distancia sea la adecuada. Luego llama a un método para recorrer todas las unidades contiguas al Jinete. Éste utiliza un método recursivo (agregarEnCadena) que va agregando cada Unidad a una lista de ünidadesAfectadas", analizando también las unidades adyacentes a las mismas (las cuales son pedidas al Tablero). Cuando ya se recorrieron todas las unidades afectadas, la Catapulta itera sobre la lista realizando una serie de ataques "sin distinción de equipo"(dado que daña a aliados y enemigos por igual), los cuales delega a su propio objeto Ataque.

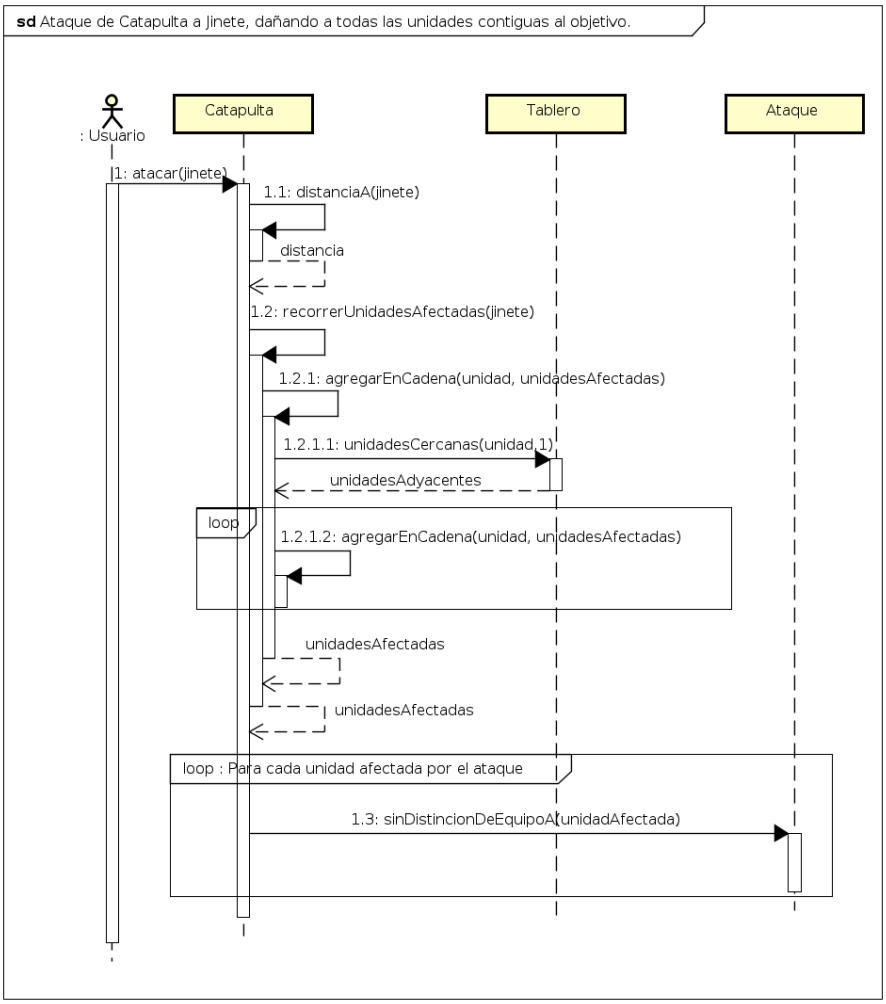

Figura 11: Catapulta ataca a un Jinete y produce daño en cadena a las unidades contiguas.