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
87039	Zoraida Flores	z__flores@hotmail.com
100589	Verónica Beatriz Leguizamón	veronicaleguizamon@outlook.es

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

Índice

1. Introducción **2**

2. Supuestos **2**

3. Diagramas de clase **2**

4. Detalles de implementación **4**
4.1. Estrategia de trabajo 4

5. Excepciones **4**

6. Diagramas de secuencia **5**

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

Nuestro modelo fue planteado en base a una clase Tablero, la cual se encarga de asociar objetos de la clase Celda con objetos de la clase Coordenada. Las celdas tienen como atributos si se encuentran ocupadas y a que sector del tablero pertenecen. El Tablero además, posee una colección de las entidades que se encuentran en el mismo.

Los objetos de la clase Jugador poseen el nombre del mismo y llevan un recuento de los puntos que puede invertir en entidades. Además contiene un objeto de la clase Equipo que se encarga de almacenar las entidades que compró.

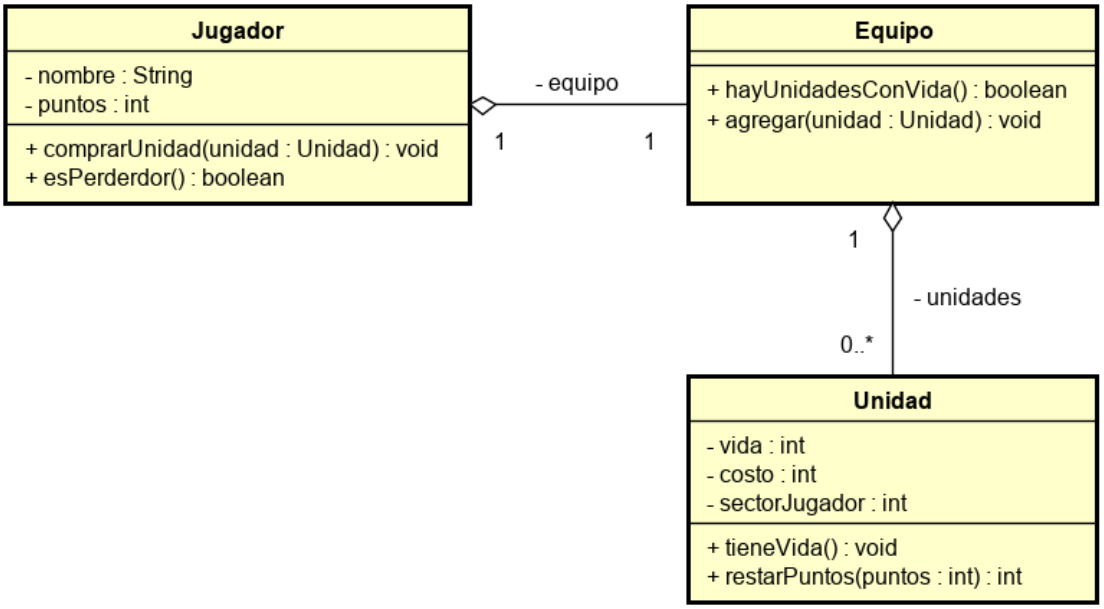

Figura 1: Diagrama del Jugador.

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).

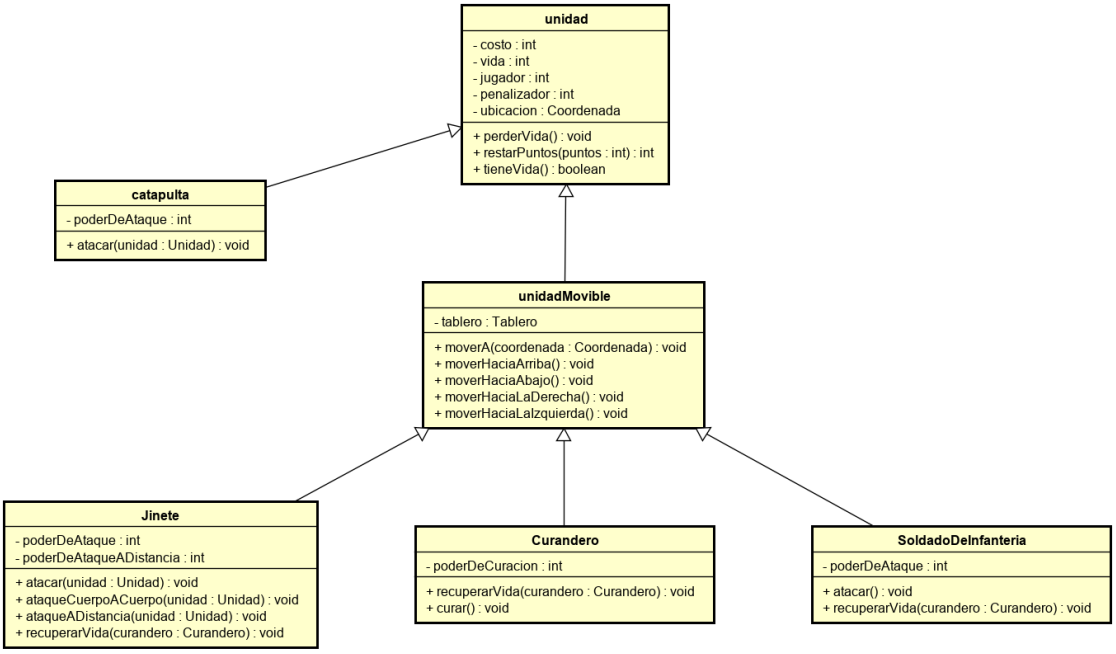

Figura 2: Diagrama de herencia de Unidades.

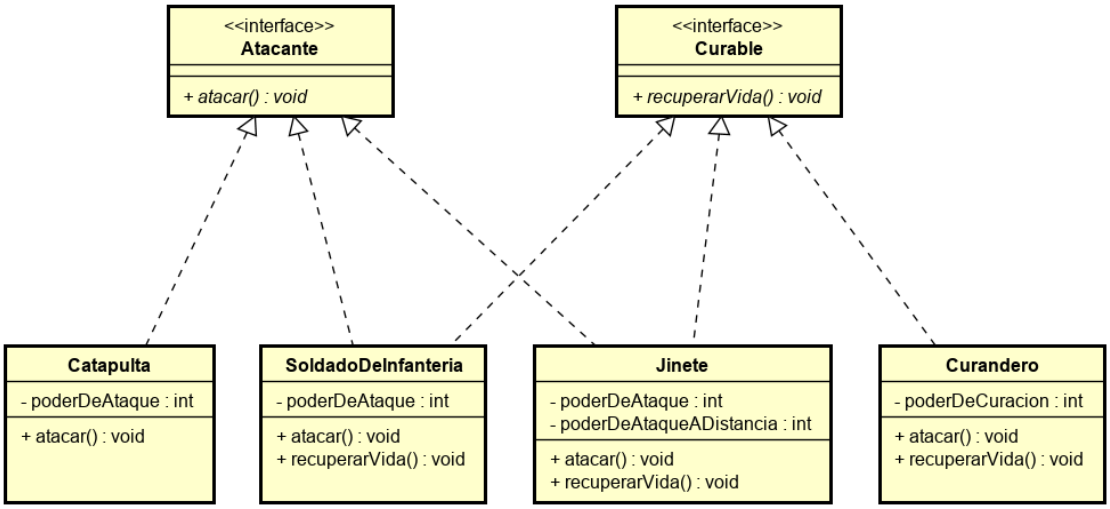

Figura 3: Diagrama de interfaces de Unidades.

Para el funcionamiento de los movimimientos instruidos directamente sobre las unidades, vimos necesario que Unidad conozca su Coordenada. Además las UnidadesMovibles deben conocer al Tablero del juego.

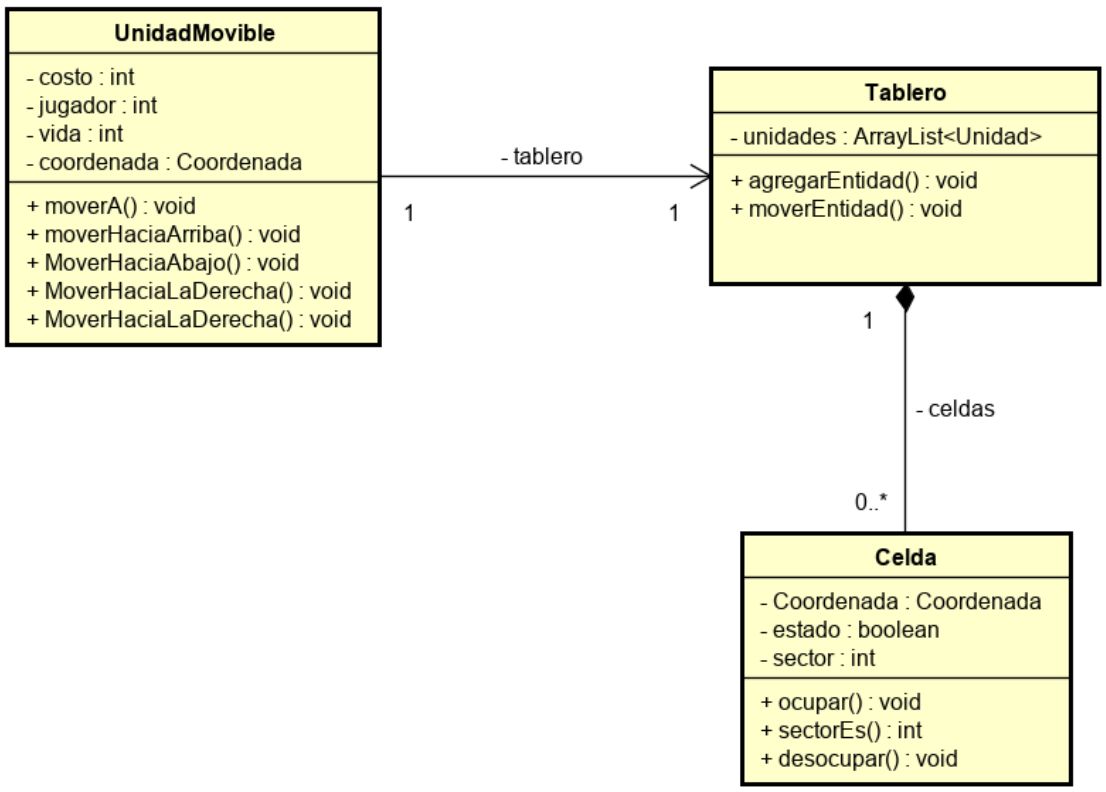

Figura 4: Diagrama de UnidadMovible con Tablero.

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.

CeldaNoExisteExcepcion La Celda objetivo no está en el tablero.

CeldaNoAdyacenteExcepcion La Celda objetivo no es adyacente a la Unidad.

CeldaEstaOcupadaExcepcion La Celda objetivo se encuentra ocupada por otra Unidad.

MovimientoInvalido Excepción generalizada ante un eventual error de movimiento.

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.

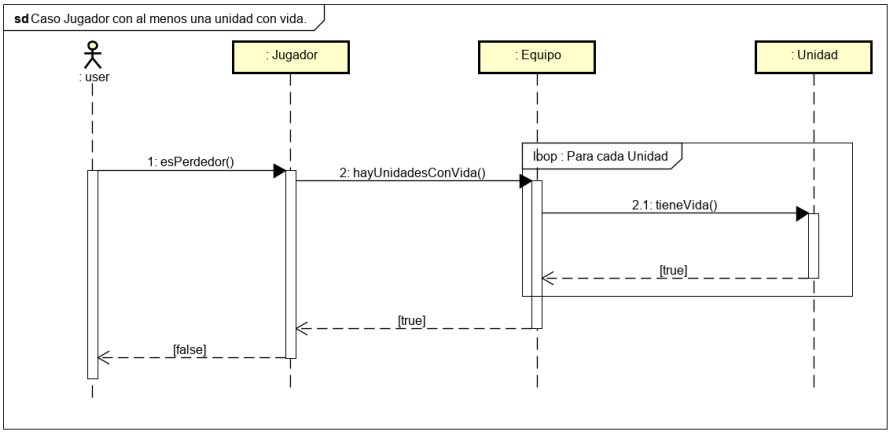

Figura 5: Jugador no ha perdido si todavía tiene alguna unidad viva.

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.

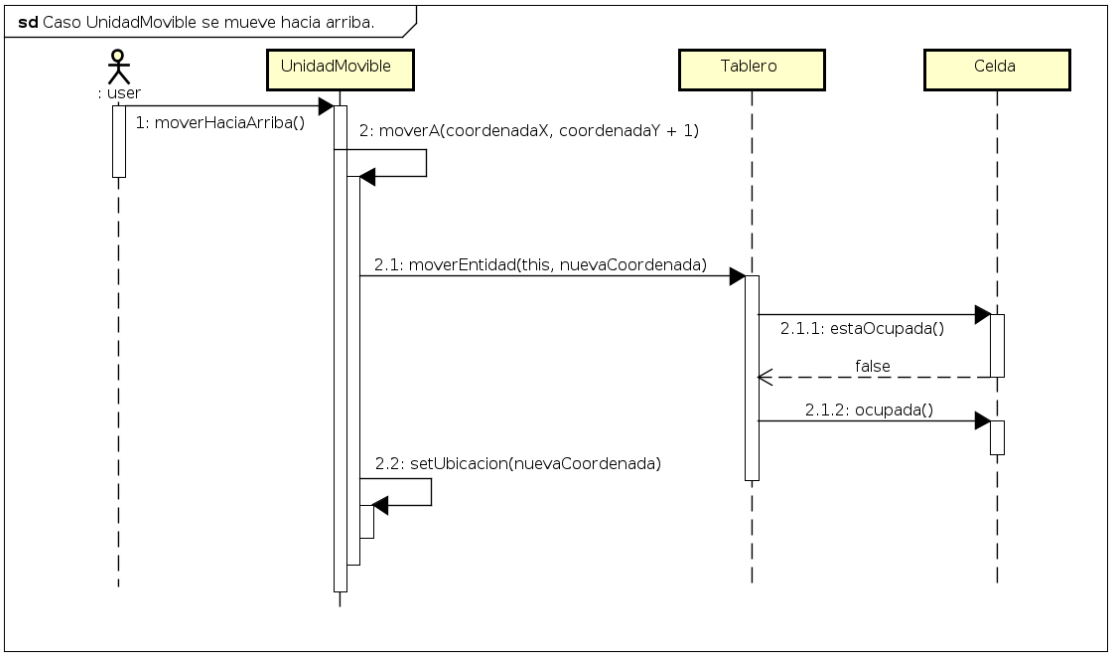

Figura 6: UnidadMovible se mueve hacia arriba.

En este caso, se le indica a UnidadMovible que se mueva hacia la Celda que tiene directamente arriba. La unidad llama a su propio método "moverA" donde pasa como parámetros sus propias coordenadas, sólo que sumándole 1 a la coordenada Y (para moverse hacia arriba). Luego le pide a Tablero que consulte a la Celda y, como ésta indica que no está ocupada, UnidadMovible realiza el movimiento y Tablero se encarga de indicarle a la Celda de que ahora dejo de estar libre.

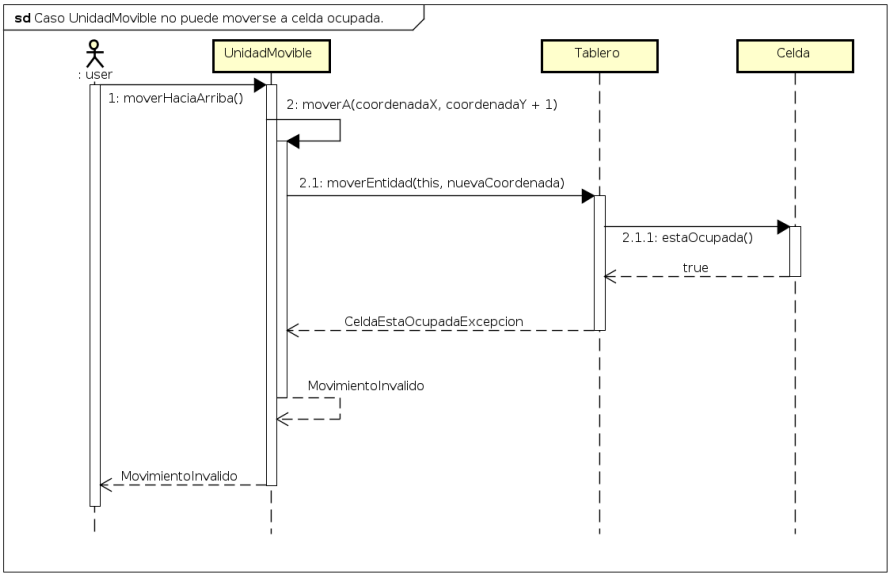

Figura 7: UnidadMovible trata de moverse hacia arriba pero no puede.

Esta otro caso es similar al anterior, sólo que cuando se le pregunta a Celda si está ocupada y

éstas indica que sí, Tablero lanza una excepción CasillaEstaOcupadaExcepcion que es atrapada y elevada por los métodos de la UnidadMovible.

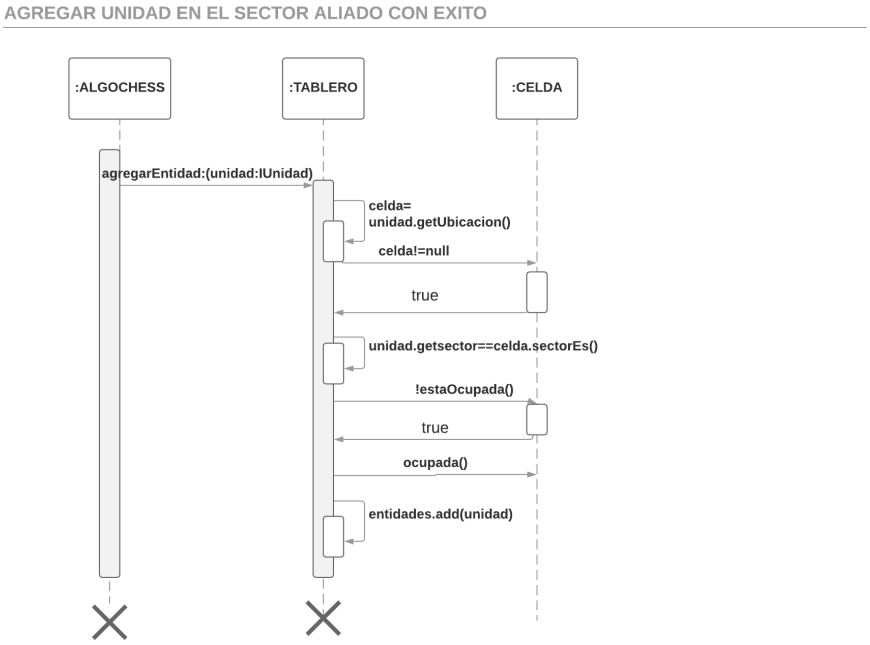

Figura 8: Tablero coloca las unidades en sus respectivos sectores.

En este caso mostramos como tablero coloca las unidades que el jugador a elegido, entonces el tablero valida si se pretende colocar una unidad que no pertenece al sector, que le corresponde, además verifica si el lugar donde colocar esta disponible para albergar a la unidad.