

---

## Algoritmos III

Bugliot, Nicolás  
Karagoz, Filyan  
Martinez, Santiago  
Ruiz Huidobro, Matías

# Algo-Chess

## TP3

### Introducción:

El objetivo de este trabajo práctico es el desarrollo grupal de una aplicación, aplicando los conceptos aprendidos durante el curso. Para la implementación se utilizará Java, con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua. La aplicación será desarrollada en su totalidad siguiendo el modelo MVC (Modelo Vista Cotrolador), por lo que contará con una interfaz gráfica.

La aplicación consiste en un videojuego por turnos, de dos jugadores conformado de un tablero y distintas unidades sobre él. El objetivo del juego es destruir todas las unidades enemigas y gana el jugador que lo consigue primero. El jugador dispone de distintas entidades para acomodar en el tablero, pero tiene una cantidad limitada de puntos para usar (cada unidad cuenta con un costo distinto).

### Supuestos:

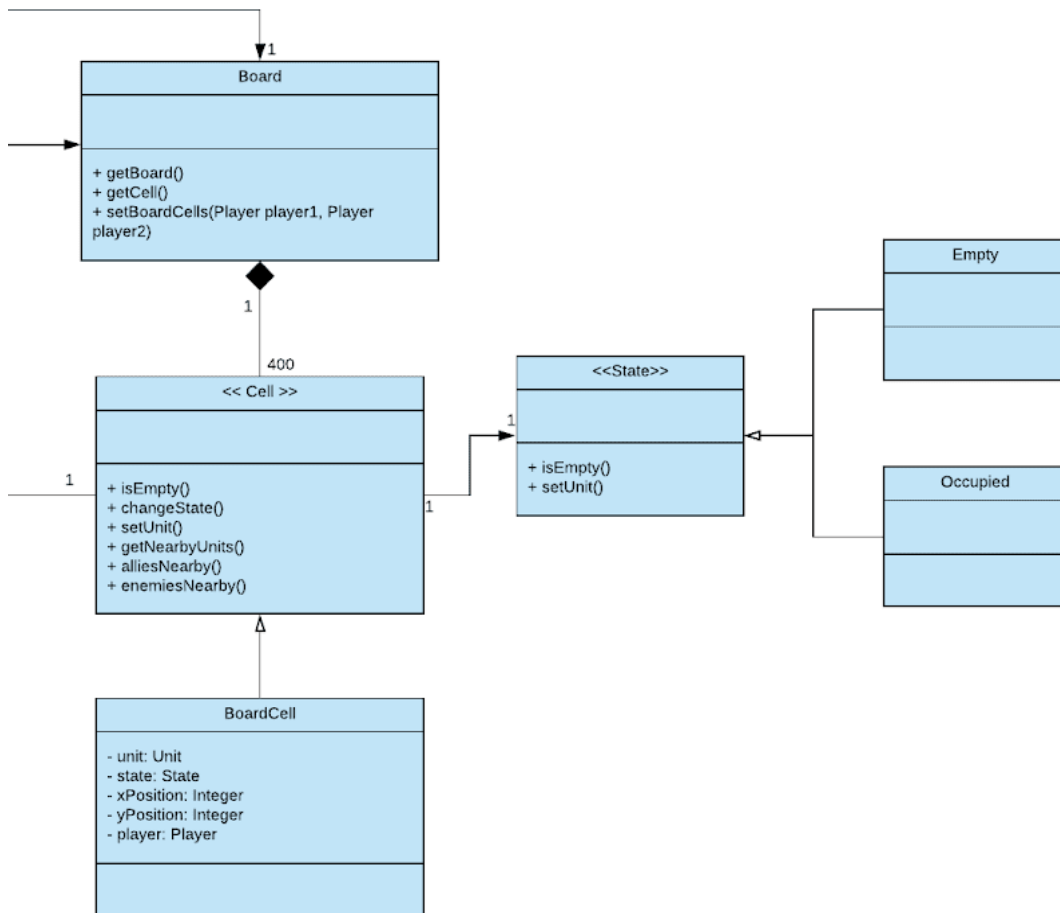
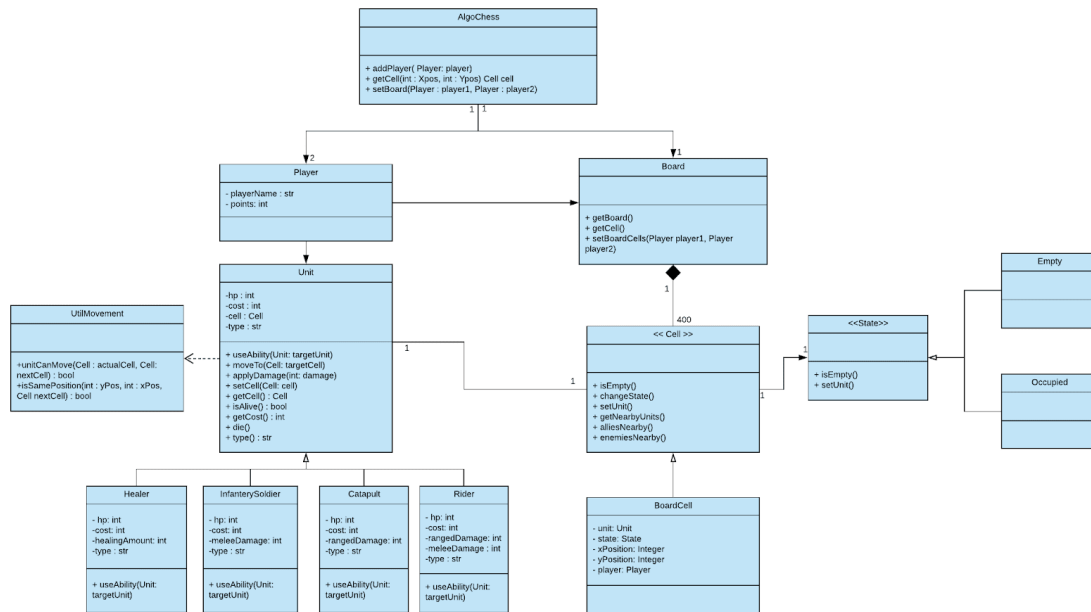
Los supuestos que tomamos para este trabajo práctico fueron:

- Permite hasta dos jugadores.

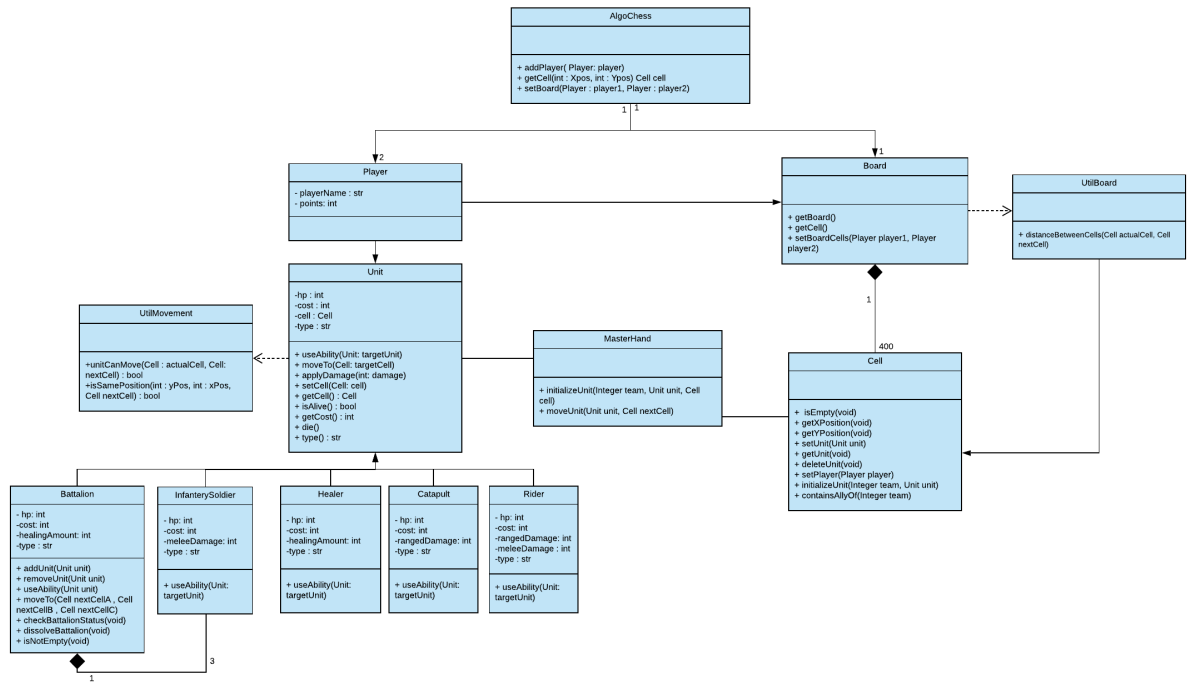
### Diagramas de Clases:

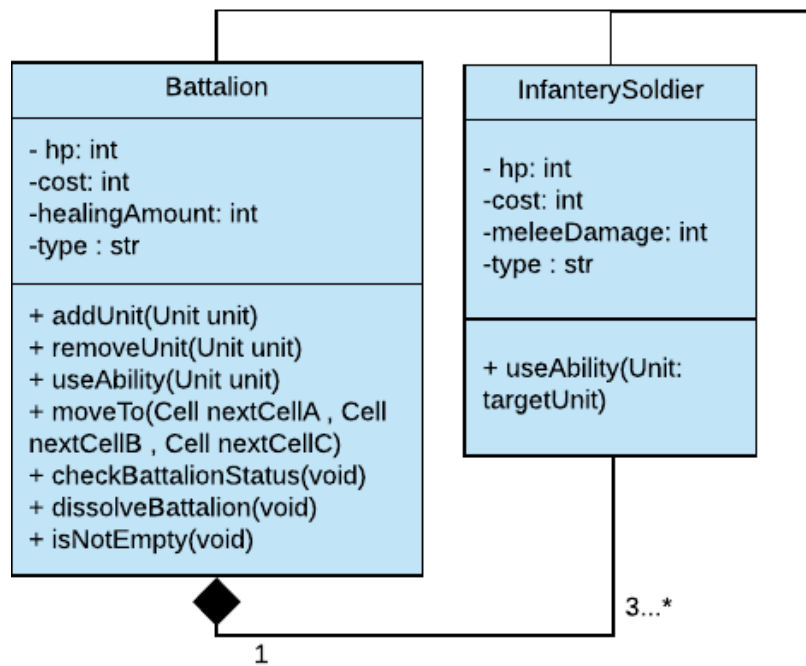
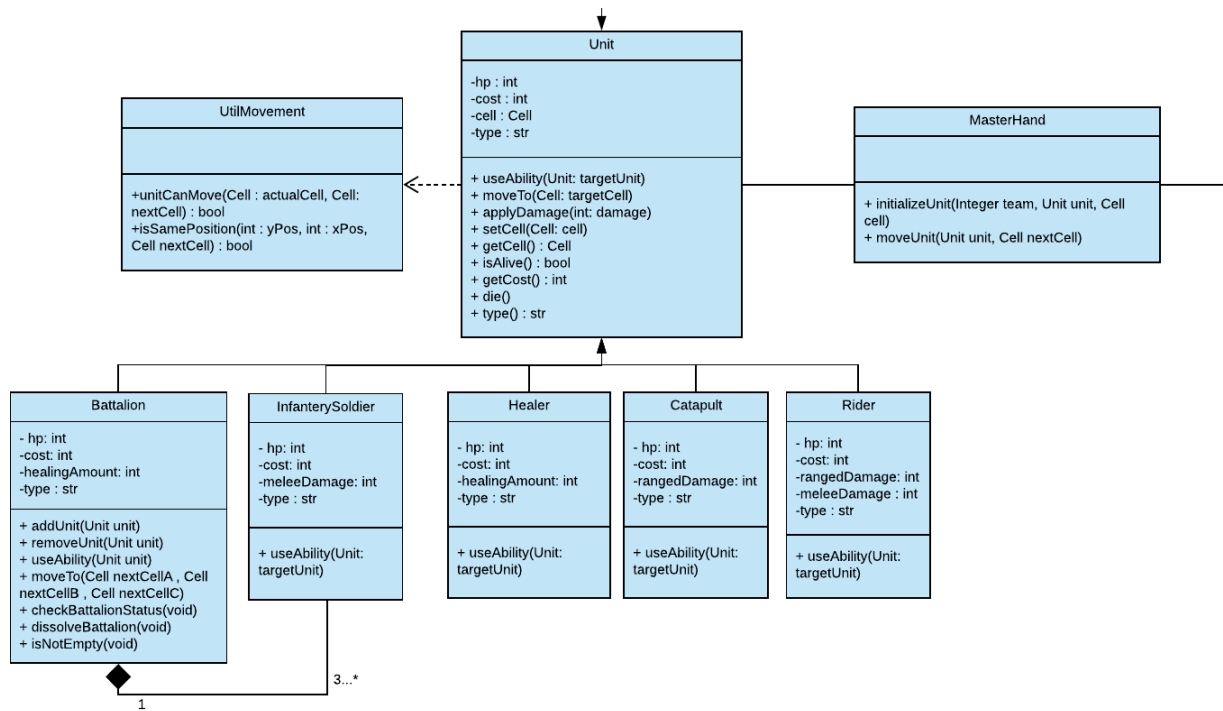
### Primer Entrega:

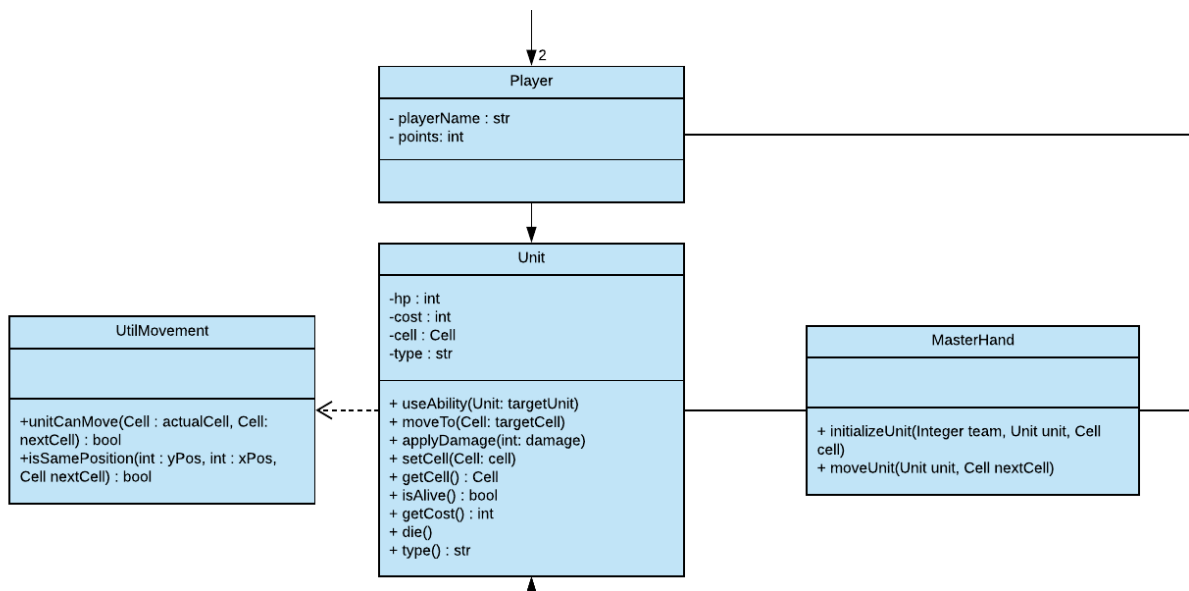
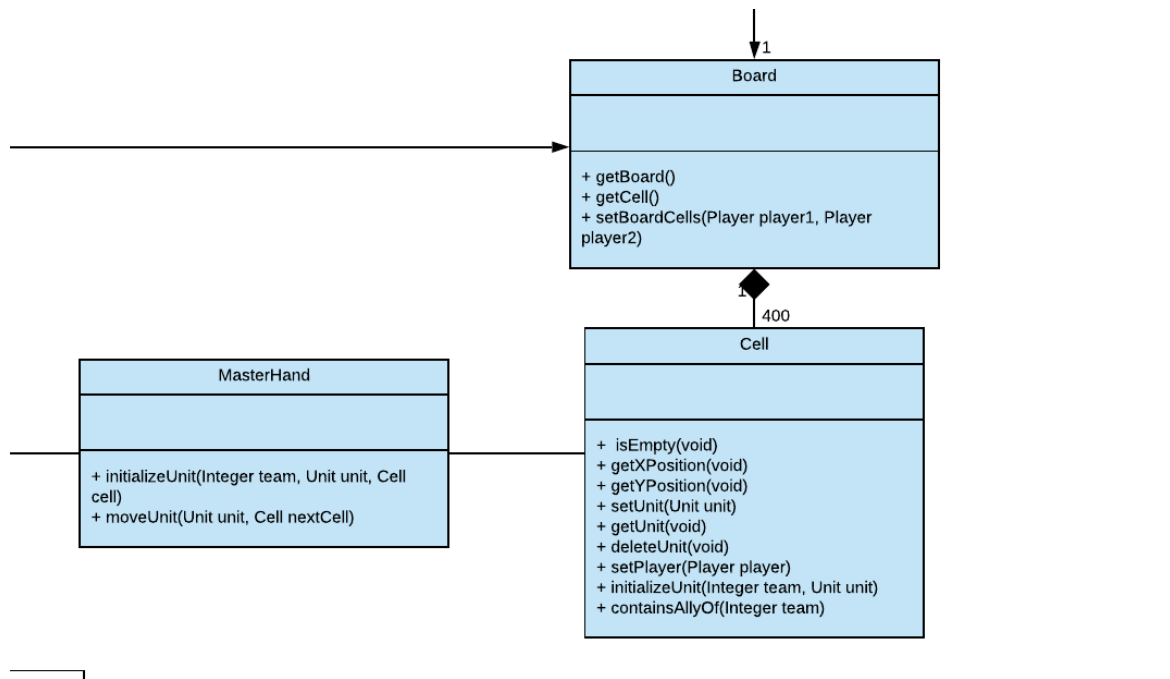
Para esta primer entrega se conceptualizaron los siguientes diagramas de clases como un primer boceto de las clases necesarias para lograr las interacciones requeridas.



## Segunda Entrega:



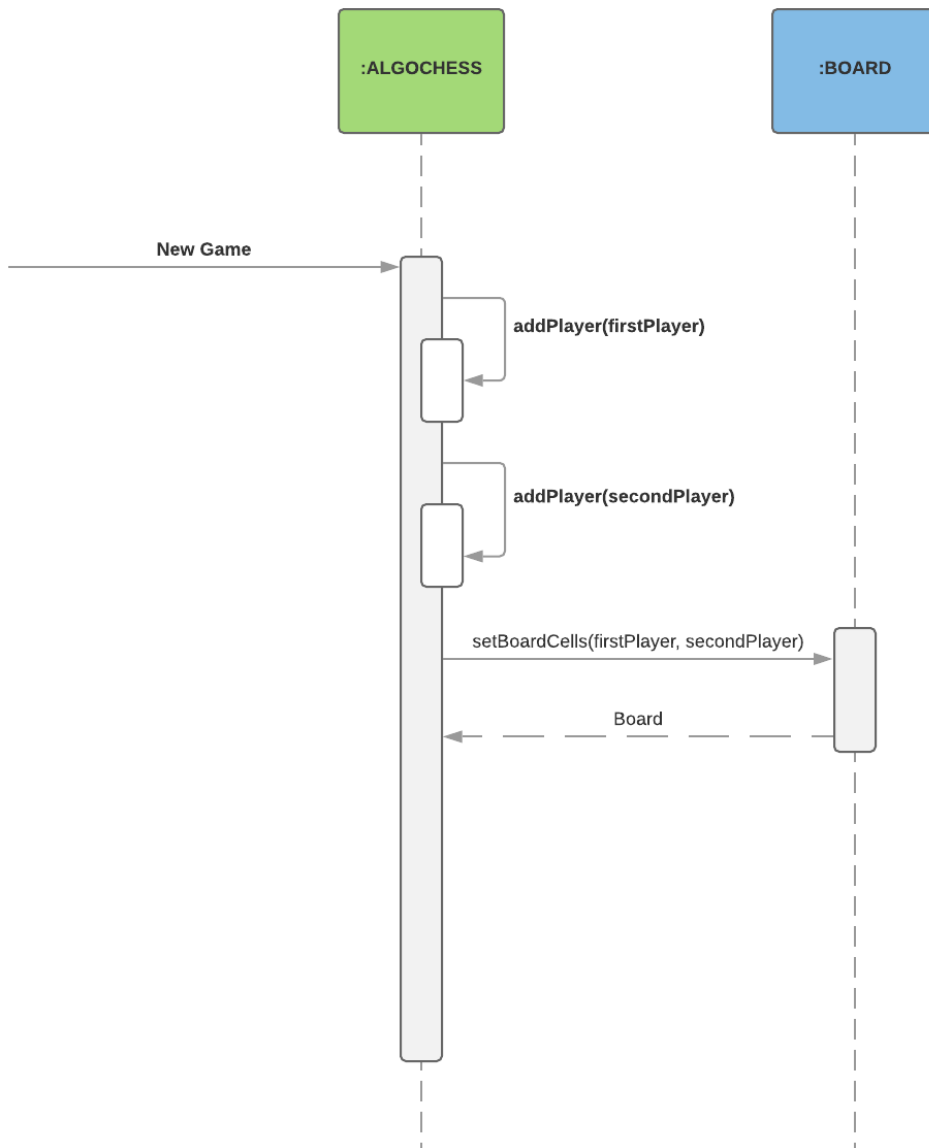




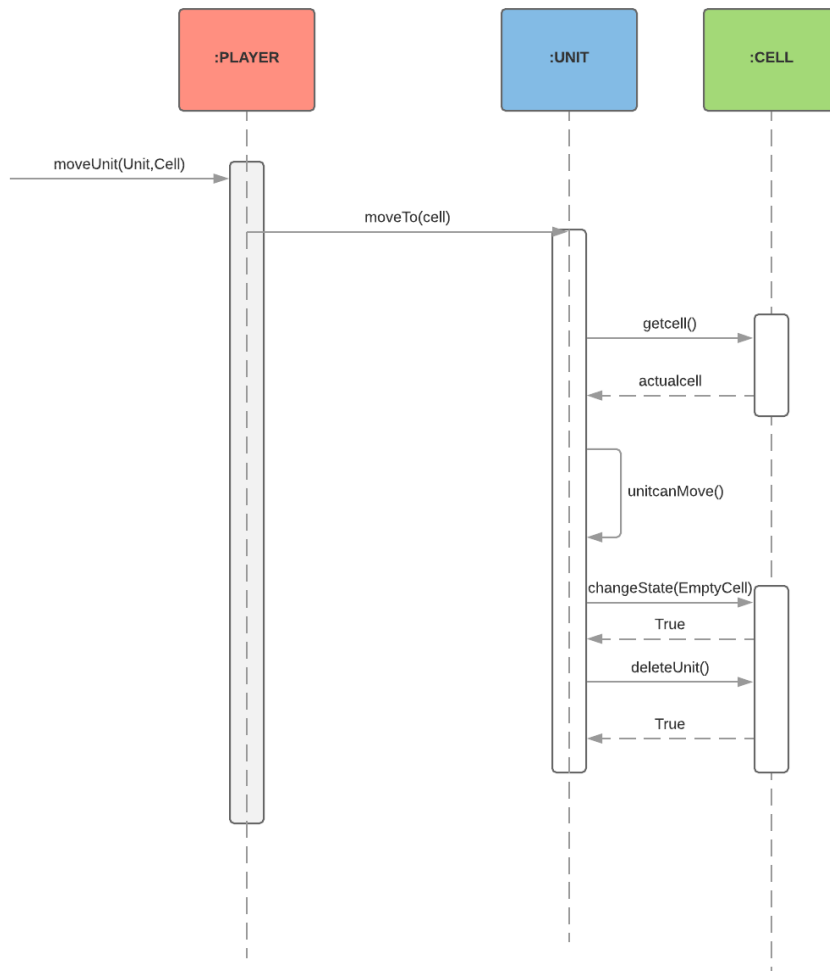
## Diagramas de Secuencia:

### Primera Entrega:

*Creación del tablero*



## Mover una pieza



---

## Segunda Entrega:

**Diagramas de Paquetes:**

**Diagramas de Estado:**

**Detalles de Implementación:**



---

---

## Primer Entrega:

Para esta primer entrega se planteó una estructura que intente imitar el diagrama de clases indicado previamente. Optamos por hacer una clase de Unidad (Unit) abstracta la cual concentraba el comportamiento general a todas las unidades y polimórficamente tratar las interacciones entre unidades. Esto se realizó optando por qué en vez de una unidad atacar o curar, cada unidad tenga una o varias habilidades. Entonces a la unidad se le indicará que utilice su habilidad (método UseAbility) sobre otra, y luego en las implementaciones hijas de Unidad, se defina como comportarse ante este método.

También creamos una unidad extra para las pruebas, un “Muñeco de Prueba” (TestDummy) el cual nos permitió realizar pruebas de ataques sobre él, obteniendo valores específicos de vida y daño, sin comprometer el encapsulamiento de las otras unidades.

## Segunda Entrega:

En esta fase, requerimos realizar pruebas en las que varias clases interactúan entre sí. Es uno de estos casos el del jinete, para el cual tuvimos que implementar una técnica de Mocking para simular la interacción con el tablero de manera controlada. Esto nos permitió tanto trabajar en las clases de tablero y unidades a la vez, ya que no requerimos la terminación del tablero para poder verificar el funcionamiento de las unidades, y también la corroboración de que los métodos específicos de ataque del jinete estaban funcionando de manera esperada, ya que el comportamiento de las otras clases era simulado y controlado.

También utilizamos una técnica de Espionaje (Spy) para simular el comportamiento específico de uno de los métodos de las unidades con la que interactúan. De esta forma al preguntarle a una unidad a que distancia se encontraba, podíamos controlar el resultado de ese método, pero mantener el resto del comportamiento normal de la unidad.

## Excepciones:

### Primer Entrega:

Las siguientes excepciones fueron creadas con el fin de obtener mayor información en caso de error y poder también abstraer los errores de las implementaciones propias de Java.

---

Excepción de Celda Vacía: **EmptyCellException**

Excepción de Celda Ocupada: **OccupiedCellException**

Excepción de Fin De Juego: **GameOverException**

Excepción de Puntos Insuficientes: **InsufficientPointsException**

Excepción de Movimiento: **MovementException**

Fueron creadas de Manera Preemptiva para disponer de ellas a futuro durante la siguiente fase de desarrollo, donde más entidades estarán interactuando entre sí y consideramos nos serán útiles.

## Segunda Entrega:

Las previamente mencionadas excepciones fueron adaptadas, utilizandolas como casos de error dentro de las clases particulares a las que hacen referencia sus nombres. Estas excepciones dan un mensaje particular dependiendo del caso en el que fueron lanzadas, pero su nombre general hace referencia a la clase de la que provienen. De este modo podemos prevenir y adaptarnos mejor a los posibles errores que desencadenan en excepciones en tiempo de ejecución, previniendo la terminación del programa.