FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2

[7507/9502] Algoritmos y Programación III
Curso 2
Segundo cuatrimestre de 2019

Alumno:	PAUSELLI, Fabrizio Dante
Número de padrón:	103418
Email:	fpauselli@fi.uba.ar

Alumno:	YANES KAHIL, Juan Francisco
Número de padrón:	103556
Email:	franciscoyanesk@gmail.com

Alumno:	Rodriguez, Nazareno Jose Luis
Número de padrón:	101891
Email:	rodrigueznazarenojoseluis@gmail.com

Alumno:	Elias Lamanna
Número de padrón:	101758
Email:	eliaslamanna@gmail.com

Índice

1. Supuestos

- La fase inicial del juego, en la cual cada jugador posiciona sus unidades dentro del tablero, no se encuentra divida por turnos, sino que permite a ambos jugadores posicionar sus unidades sin respetar turnos.
- Si un jugador decide abandonar el juego cerrándolo, no se mostrará ganador alguno.

2. Diagramas de clases

En esta sección mostraremos diagramas de clases generales, simplificados, sin mostrar sus métodos ni atributos. En la sección 'Detalles de implementación' se podrán encontrar diagramas mas pequeños con sus correspondientes explicaciones.

2.1. Modelo

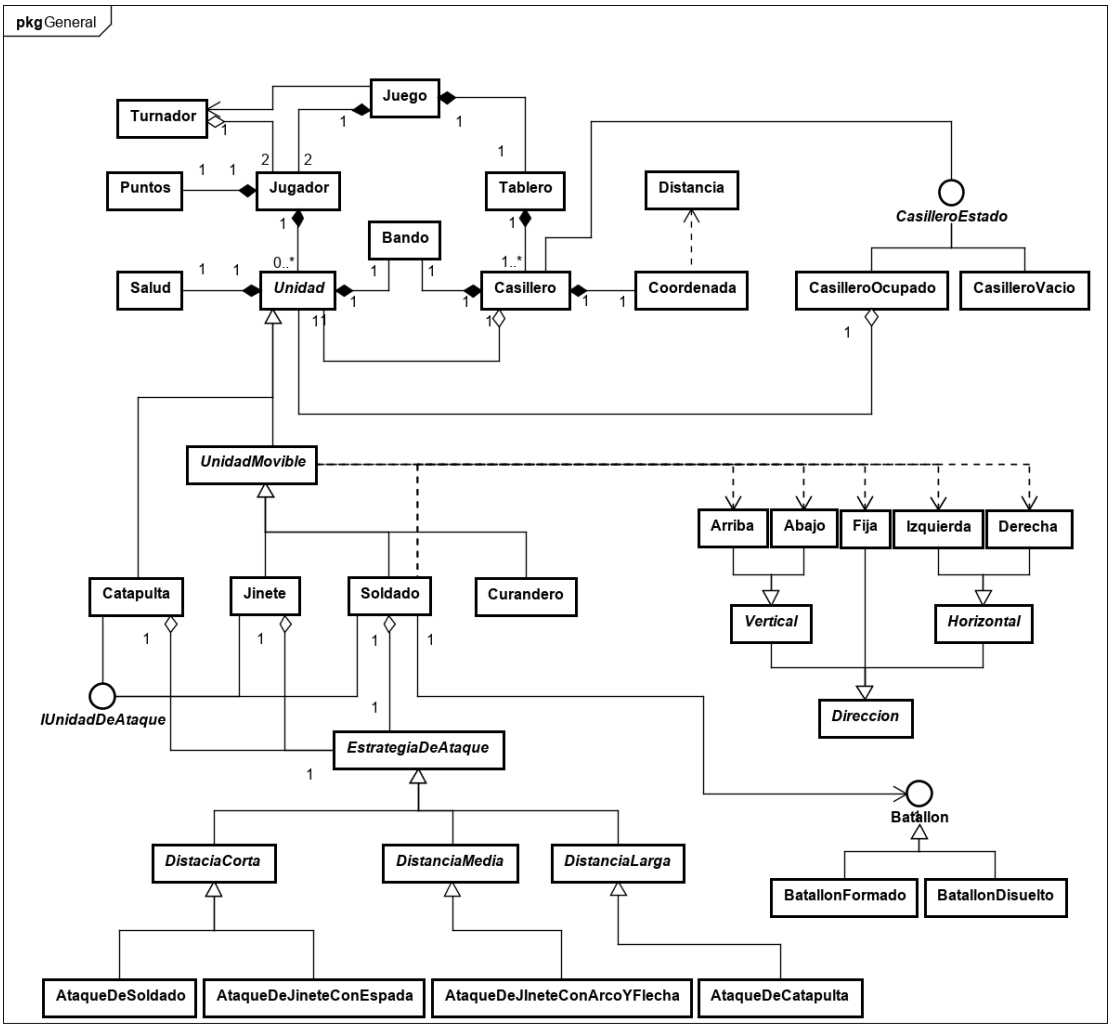

Figura 1: Diagrama general del modelo (simplificado).

2.2. Vista

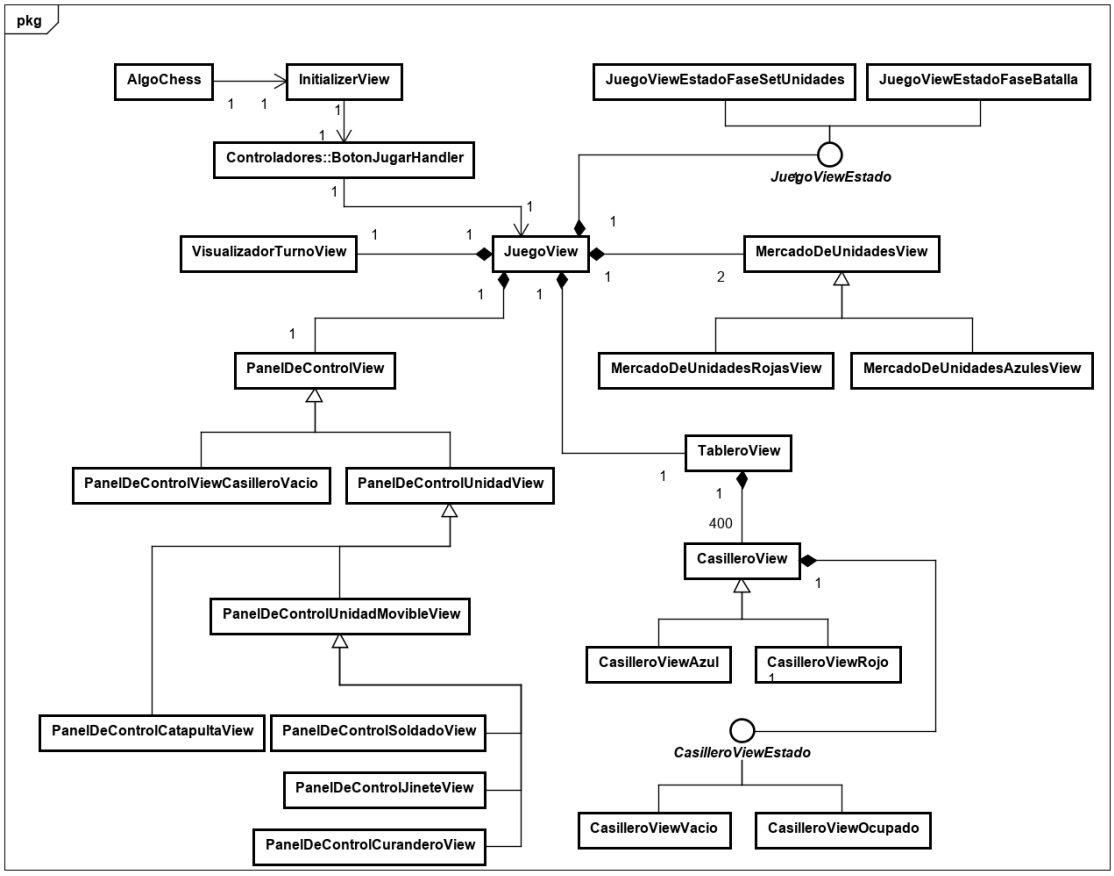

Figura 2: Diagrama general de la vista (simplificado).

3. Diagramas de secuencia

En la actual sección hemos seleccionado un conjunto de flujos de secuencias que consideramos interesante de visualizar.

3.1. Ataque

A continuación se muestran dos diagramas de secuencias correspondiente al ataque de un soldado, siendo el primero en una distancia valida, mientras que el segundo en una distancia invalida.

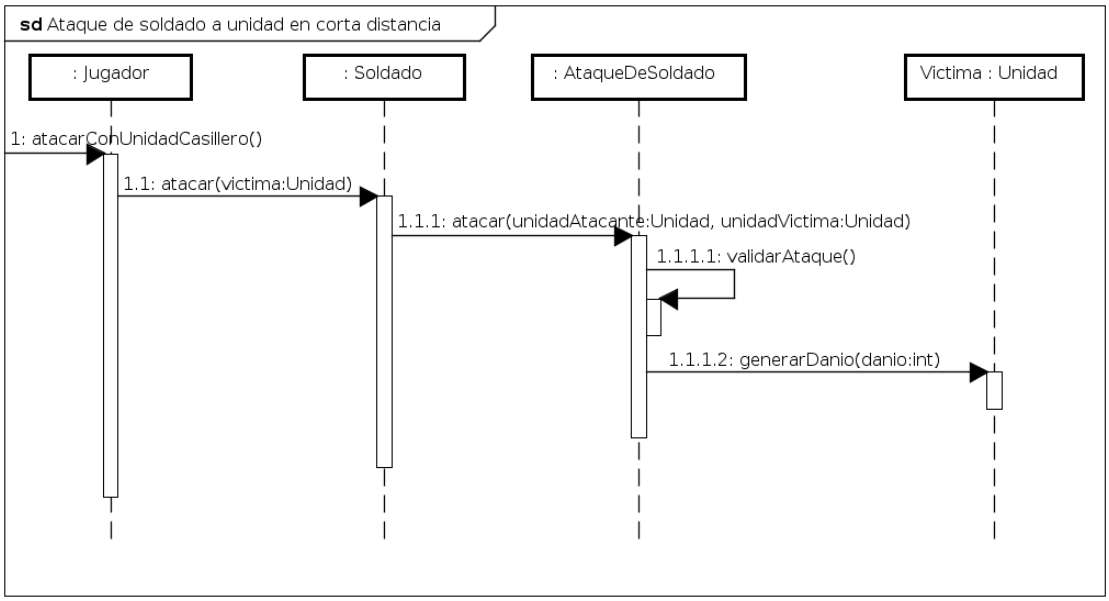

Figura 3: Diagrama de secuencia - Ataque de soldado en distancia corta (distancia válida).

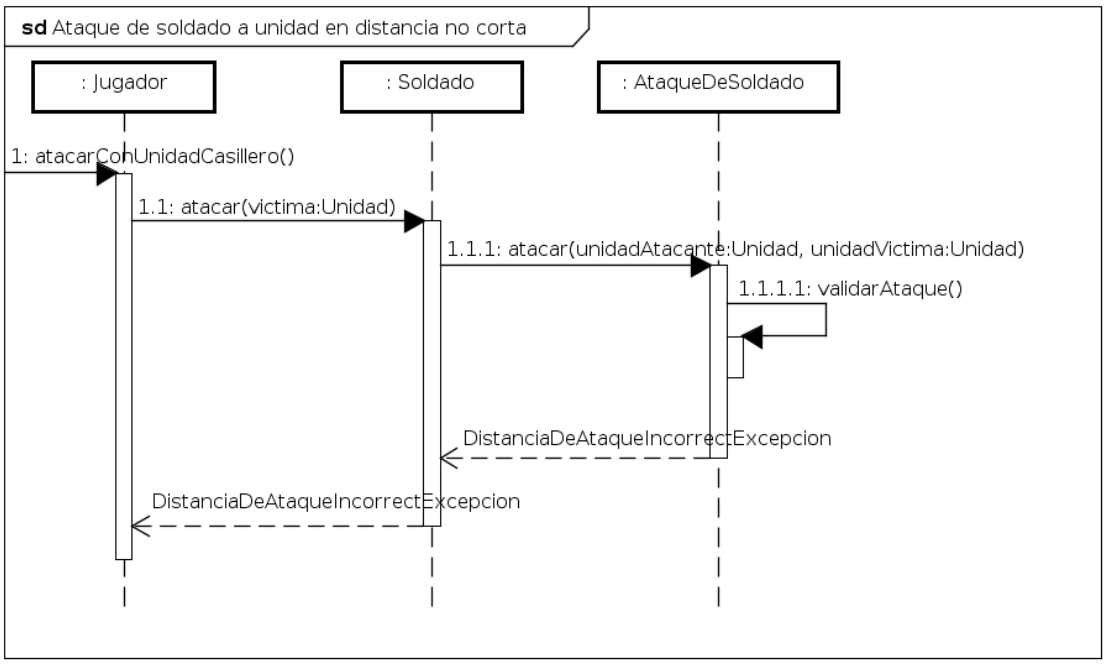

Figura 4: Diagrama de secuencia - Ataque de soldado en distancia no corta (distancia inválida).

3.2. Movimiento

Los siguientes dos diagramas muestran la secuencia al momento de realizar un movimiento, siendo el primero un movimiento válido, mientras que el segundo un movimiento inválido.

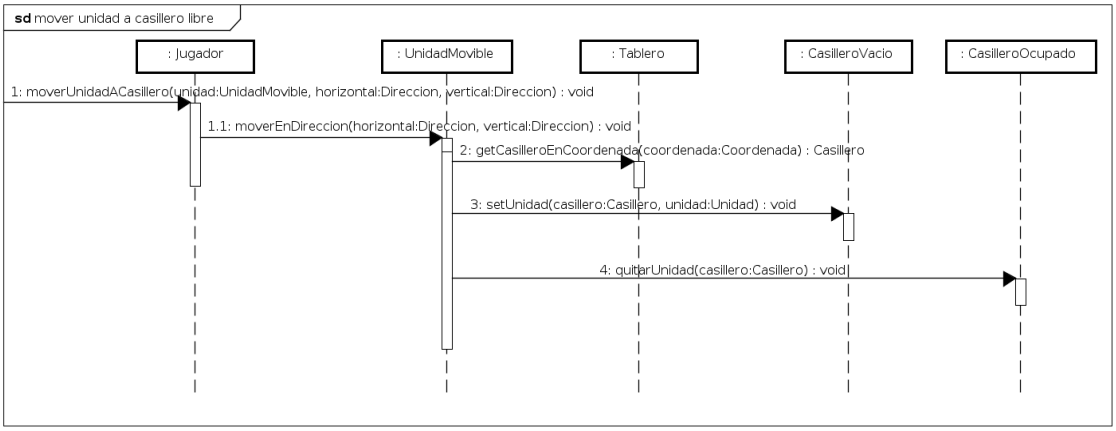

Figura 5: Diagrama de secuencia - Mover unidad a casillero vacío (movimiento válido).

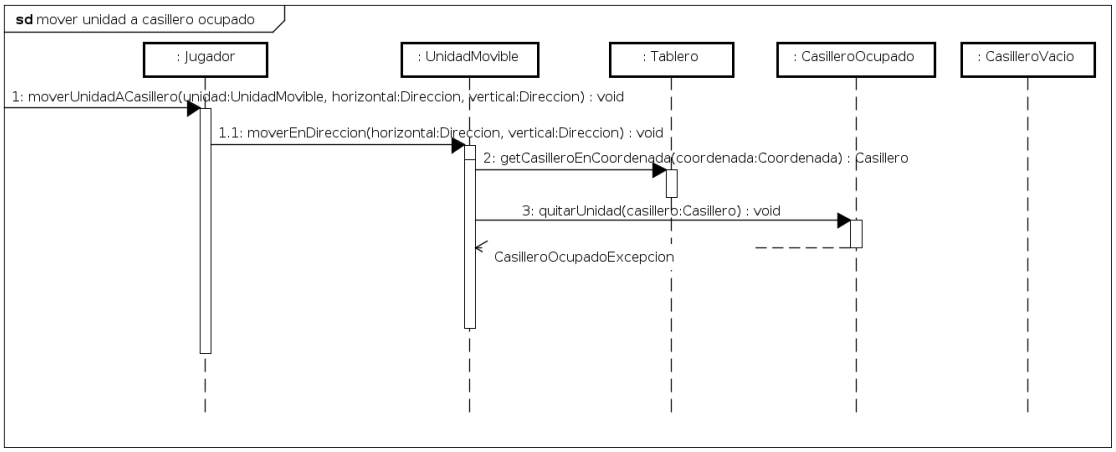

Figura 6: Diagrama de secuencia - Mover unidad a casillero ocupado (movimiento invalido).

3.3. Inicialización

El siguiente diagrama de secuencia muestra la inicialización de juego.

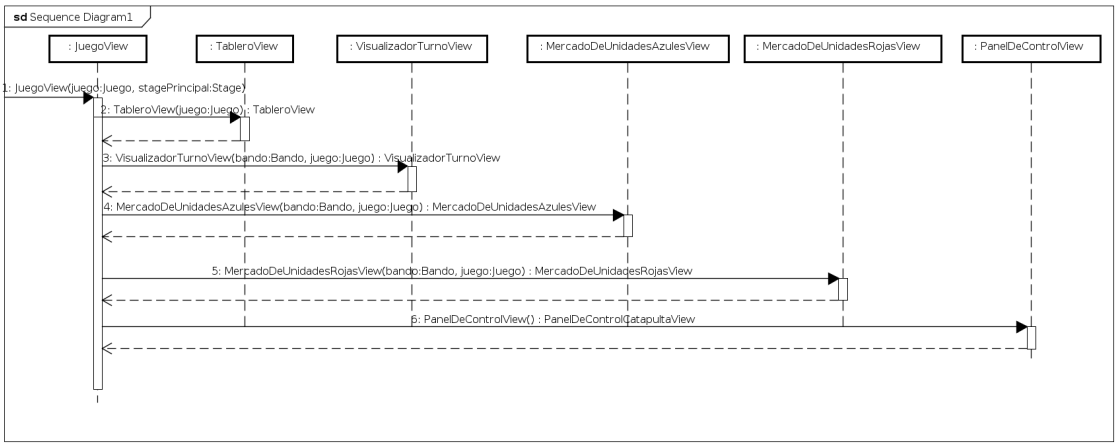

Figura 7: Diagrama de secuencia - Inicialización de juegoView.

4. Diagramas de paquetes

4.1. Paquetes principales

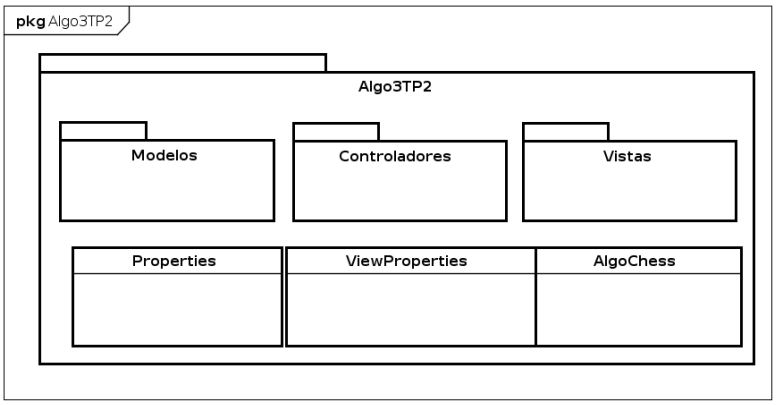

Figura 8: Paquetes principales de todo el proyecto.

4.2. Contenido de los paquetes principales

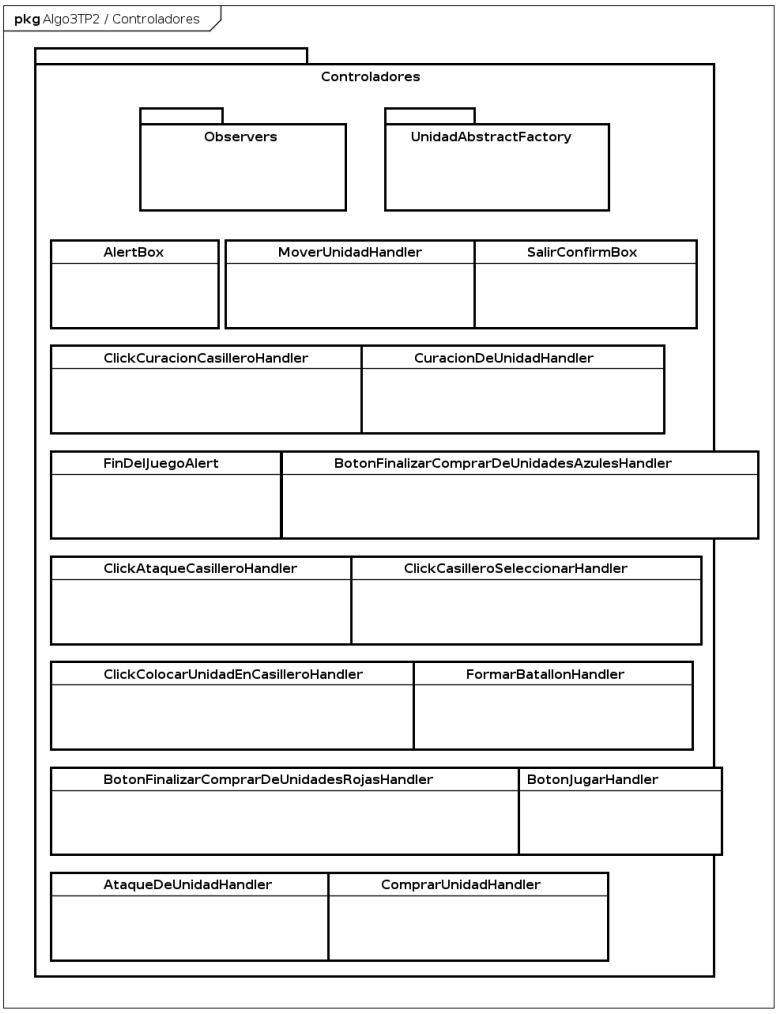

Figura 9: Contenido del paquete Controladores.

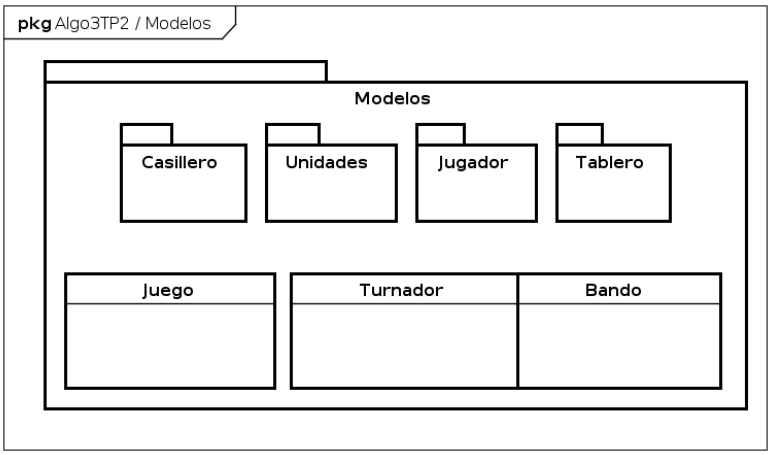

Figura 10: Contenido del paquete Modelos.

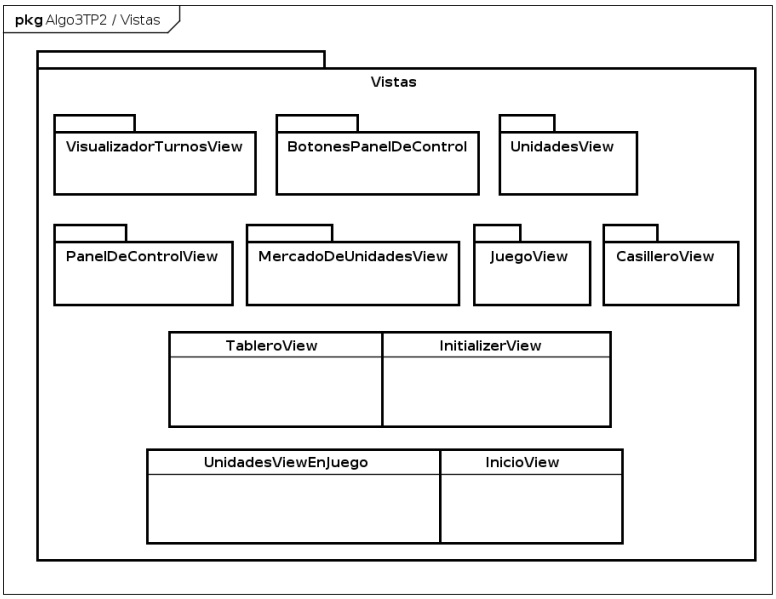

Figura 11: Contenido del paquete Vistas.

5. Detalles de implementación

El desarrollo de juego ha sido llevado adelante a respetando lo propuesto por el patrón de diseño Modelo-Vista-Controlador (MVC). Separa así los datos de una aplicación (modelo), la interfaz de usuario (vista) , y la lógica de control (controladores) en tres componentes distintos, con la finalidad de evitar el acoplamiento entre el modelo y la vista.

5.1. Implementación del modelo

5.1.1. Implementación de las unidades

Al momento de implementar las unidades, conscientes de que las mismas podrían ser sujetas a futuros cambios requeridos por el cliente (nuevas unidades, nuevos tipos de ataques para la ya existentes, cambios en el comportamiento, etc.). Decidimos entonces buscar la implementación que mayor sencillez proporcione para la implementación de los mismo. Con dicho objetivo nos dispusimos a realizar un modelado en que cada clase posea una responsabilidad única y que permita a través del uso de herencia la utilización de las mismas.

Atributos de la unidad:

La Unidad posee dos objetos como atributos principal.

- Salud: a cargo de todo lo que respecta a la vida de la unidad.
- Bando: a cargo de identificar la unidad con su correspondiente jugador.

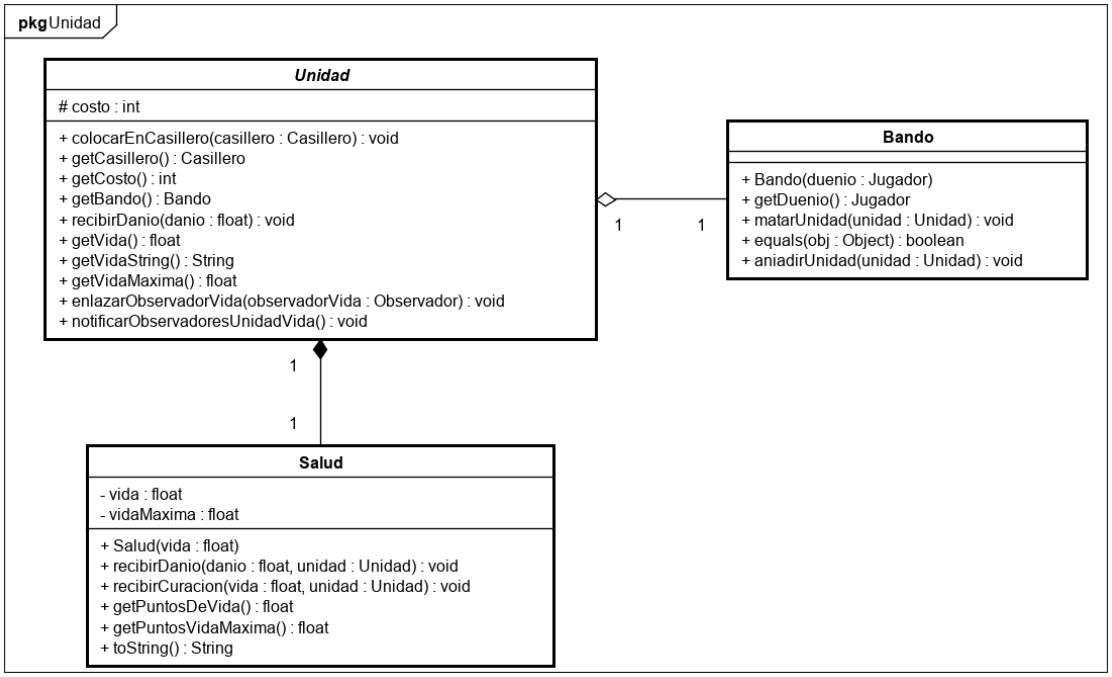

Figura 12: Diagrama de clases - Atributos de unidad.

Capacidad de movimiento:

Se han diferenciado en dos capacidades de movimientos.

- Unidad inmóvil: la unidad no puede moverse en ninguna dirección.

- Unidad móvil: la unidad puede moverse en todas las direcciones.

Siendo Unidad Móvil una clase que extiende de la clase madre Unidad, que no posee capacidad de movimiento, la misma proporciona los correspondientes métodos de movimientos. Facilitando de este modo la implementación al poder hacer que una unidad en cuestión, sencillamente extienda de la clase que se corresponda con su intención de movimiento.

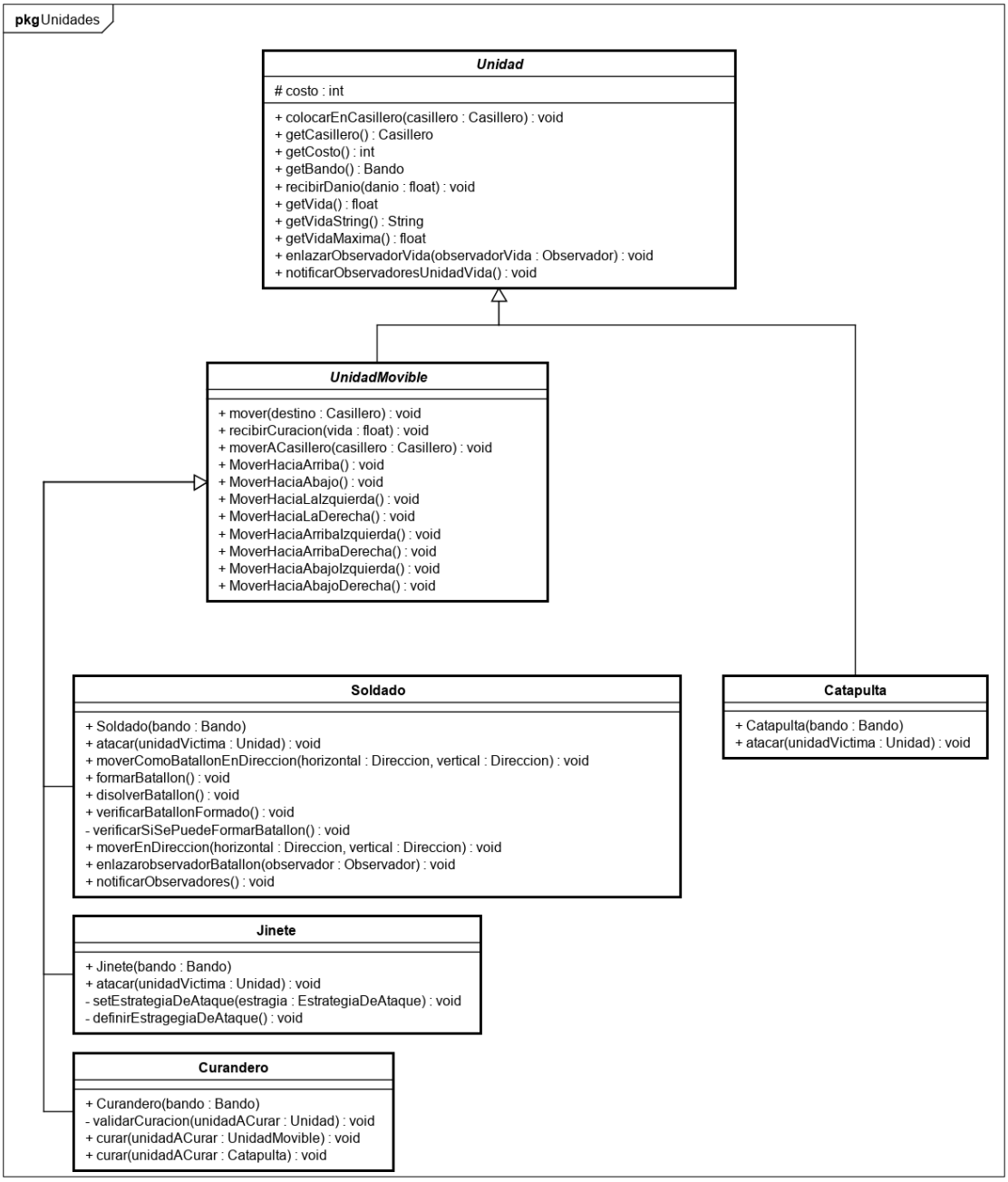

Figura 13: Diagrama de clases -Capacidad de movimiento.

Acción de ataque:

Para la misma hemos diferenciado entre 3 acciones cruciales a llevar a cabo.

- Validar la rivalidad de la unidad a atacar: responsabilidad de la clase EstrategiaDeAtaque.

- Validar distancia de ataque: responsabilidad a cargo de las clases DistanciaCorta/DistanciaMedia/DistanciaLarg según corresponda.
- Realizar el ataque con el daño correspondiente al mismo: responsabilidad la clase ataqueUnidadArma.

Siendo entonces que las primeras dos acciones ya de encuentran implementadas, tan solo resulta necesario llevar a cabo la implementación de la última acción. Cabe mencionar que el patrón de diseño utilizado ha sido Strategy.

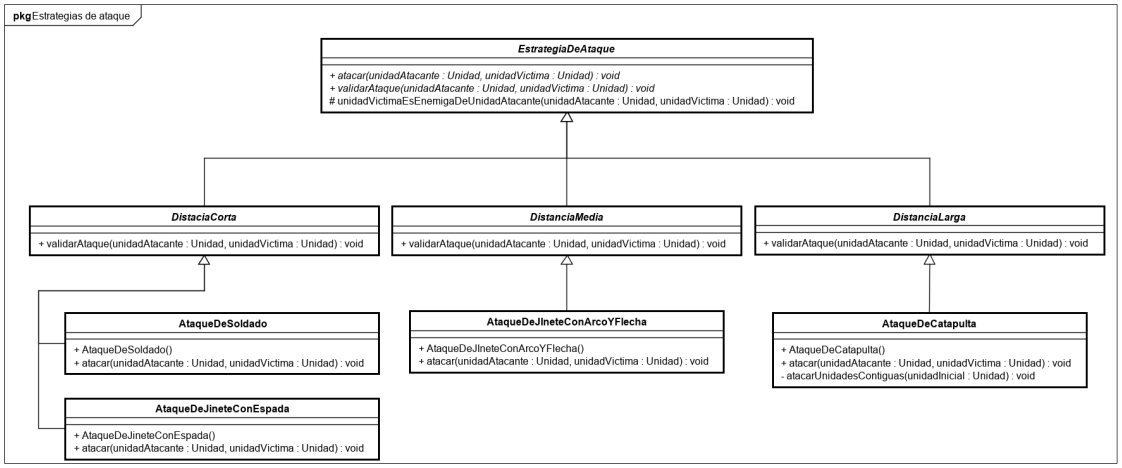

Figura 14: Diagrama de clases - Estrategia de ataque.

A continuación, mostraremos un ejemplo de cómo introducir un nuevo requisito para una unidad.

Supongamos que el cliente desea que el Soldado de infantería que tan solo posee la capacidad de atacar a distancia corta con una espada, ahora quiere que también disponga de una lanza con la cual pueda atacar a enemigos a distancia media.

La introducción de este nuevo requisito sería sumamente simple:

- Creamos una clase para el ataque pertinente, la misma la llamaremos `AtaqueDeSoldadoConLanza`.
- La misma heredará de `DistanciaMedia`.
- A continuación en la primer línea llamemos al metodo de la clase madre `validarAtaque`:

  ```
  this.validarAtaque(unidadAtacante, unidadVictima);
  ```

- Si el ataque no es válida retorna la excepción correspondiente al motivo, de lo contrario se continuará con la ejecución de la segunda linea del método.
- Finalmente solo debemos atacar la unidad con el valor de daño correspondiente

  ```
  unidadVictima.recibirDanio(Properties.danioSoldadoConLanza);
  ```

  (Se aconseja el uso del archivo `Properties` para guardar las constantes de daño de cada tipo de ataque)

5.1.2. Implementación del tablero y sus casilleros

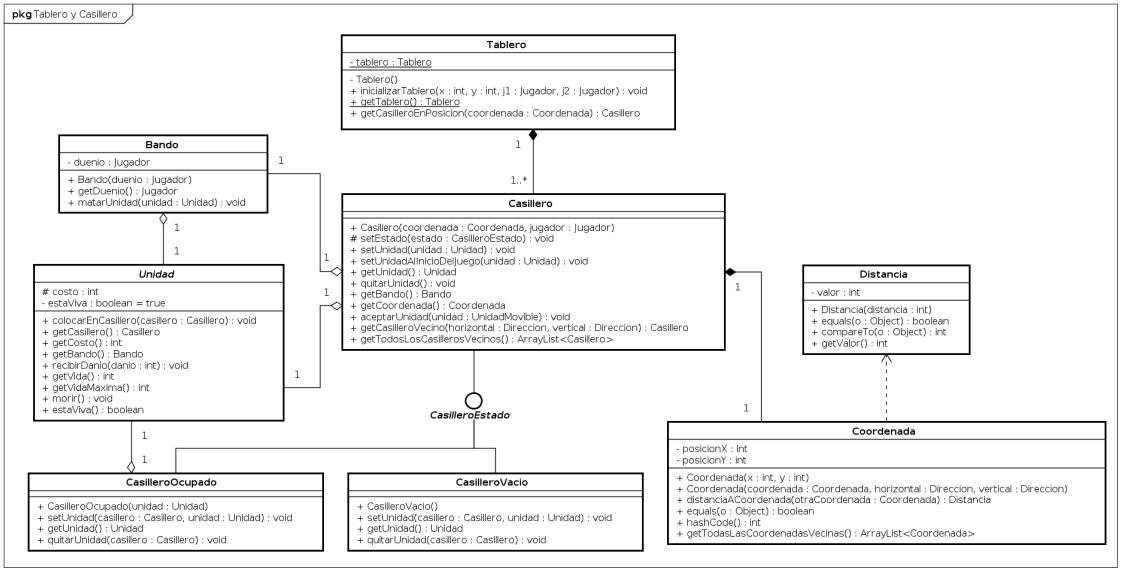

Figura 15: Diagrama de clases - Tablero y Casillero.

El tablero se encuentra compuesto por casilleros, los almacena en un `Hashmap<Coordenada, Casillero>` de modo tal de facilitar el acceso a los casilleros. La coordenada posee redefinido su método `hashCode()` de modo que al buscar un coordenada en el Hashmap de tablero, la búsqueda ignora si se trata de instancias distintas de Coordenada sino, que tan solo lo haga en relación a sus atributos "posicionXposicionY". Permitiendo entonces:

```
tablero.posicionarUnidad(unidadEnCuestion, new Coordeanda(posicionX, posicionY));
```

Para los casilleros hemos optado por el Patrón de diseño `State`, diferenciado asi en dos estados:

- `CasilleroOcupado`: Posee una unidad.
- `CasilleroVacio`: No posee una unidad.

5.2. Implementación de la vista

Como se ha mencionado anterior mente, se ha hecho uso del patrón de diseño Modelo Vista Controlador (MVC) de modo tal que la visualización no lleva adelante acciones sobre el modelo, tan solo sirve de vista y ciertos objetos poseen la capacidad de observar al modelo a través del patrón `Observer` con la finalidad de mantenerse actualizados.

Para la vista del juego, decimos implementar aquellos objetos del modelo que poseían una visualización, el tablero, los casilleros y las unidades son ejemplos de estos. Estos se han implementado del modo mas básico posible, ya que su única responsabilidad es la de visualizar los objetos del modelo y actualizar la vista de acuerdo a los cambios que surgen en el mismo a través del patrón `Observer`. También se han implementado aquellas vistas que resultaban necesarias para la interacción de usuario con el juego, como lo son el mercado de unidades, el visualizador de turnos y el panel de control.

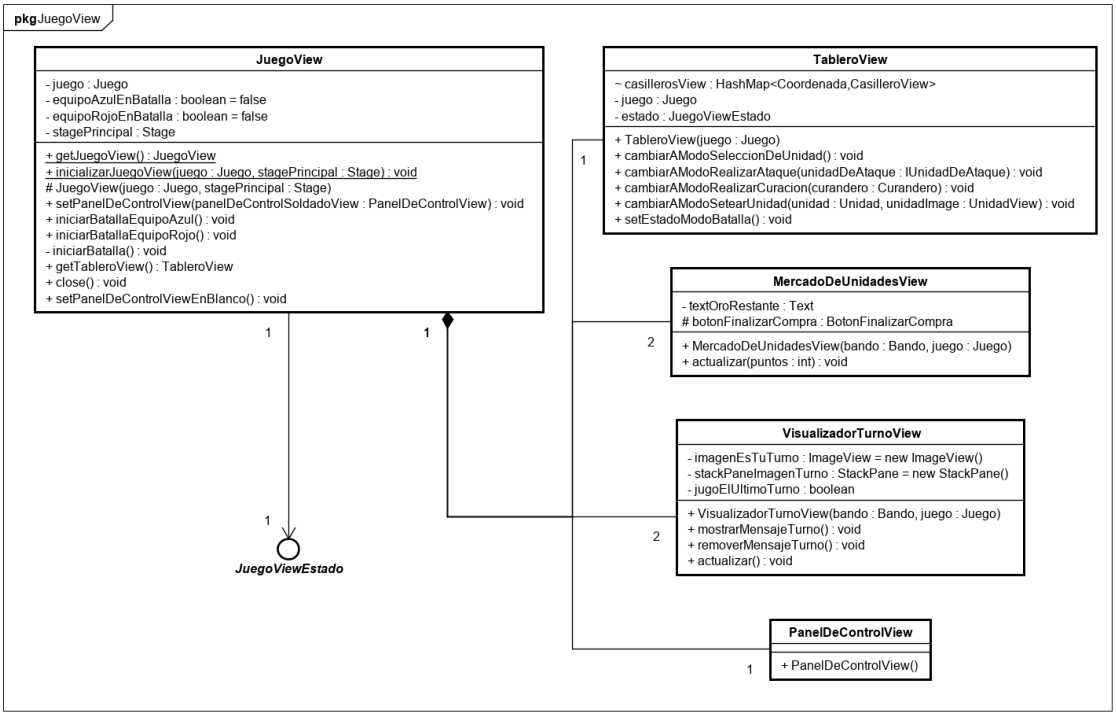

Figura 16: Diagrama de clases - vista general del juego.

El juego posee un estado asociado, implementado través del patrón `State`:

- `JuegoViewEstadoFaseSetUnidades`: visualiza a los costados del tablero el mercado de unidades y tan solo permite colocar unidas en el tablero.
- `JuegoViewEstadoFaseBatalla`: visualiza a los costados el nombre de los jugadores y la leyenda "¡Es tu turno!", según corresponda. Permite realizar movimiento, ataques y demás acciones propias de la batalla.

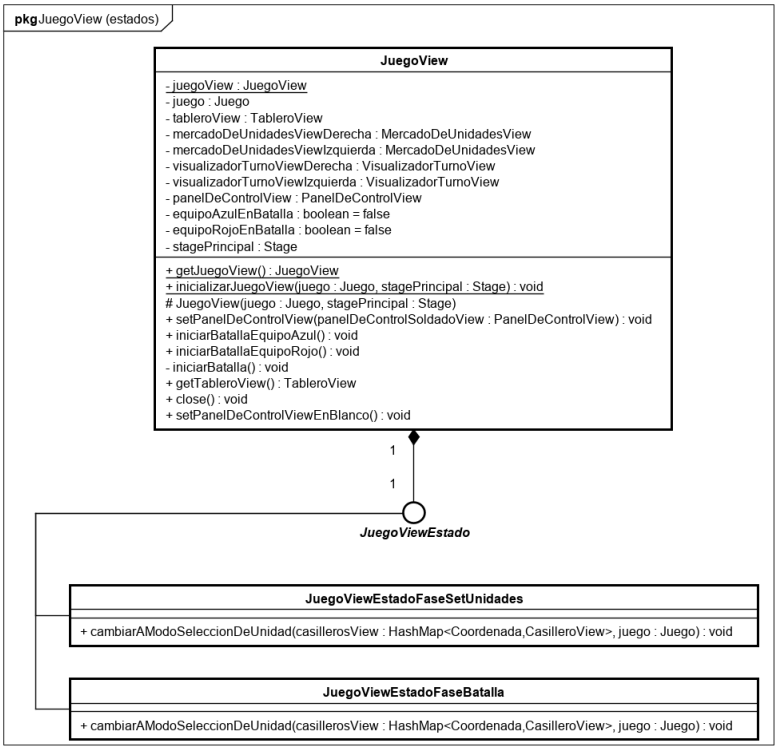

Figura 17: Diagrama de clases - estados de `JuegoView`.

5.2.1. Vista del tablero

La vista del tablero se encuentra conformada una lista de vistas de `Casillero`, los cuales poseen un enlace con el casillero del modelo correspondiente a su coordenada. Dicho enlace es realizado a través del patrón `Observer`, la relación es `CasilleroView`: objeto observador; `Casillero`: objeto observado. Un cambio de estado del `Casillero` en el modelo será comunicado a `CasilleroView`, generando que este también lleve adelante un cambio de estado. A igual que en el modelo se diferencia entre dos estados, haciendo uso del patrón `State`:

- `CasilleroViewOcupado`: visualiza la uniad que tiene dentro.
- `CasilleroViewVacio`: no visualiza ninguna unidad.

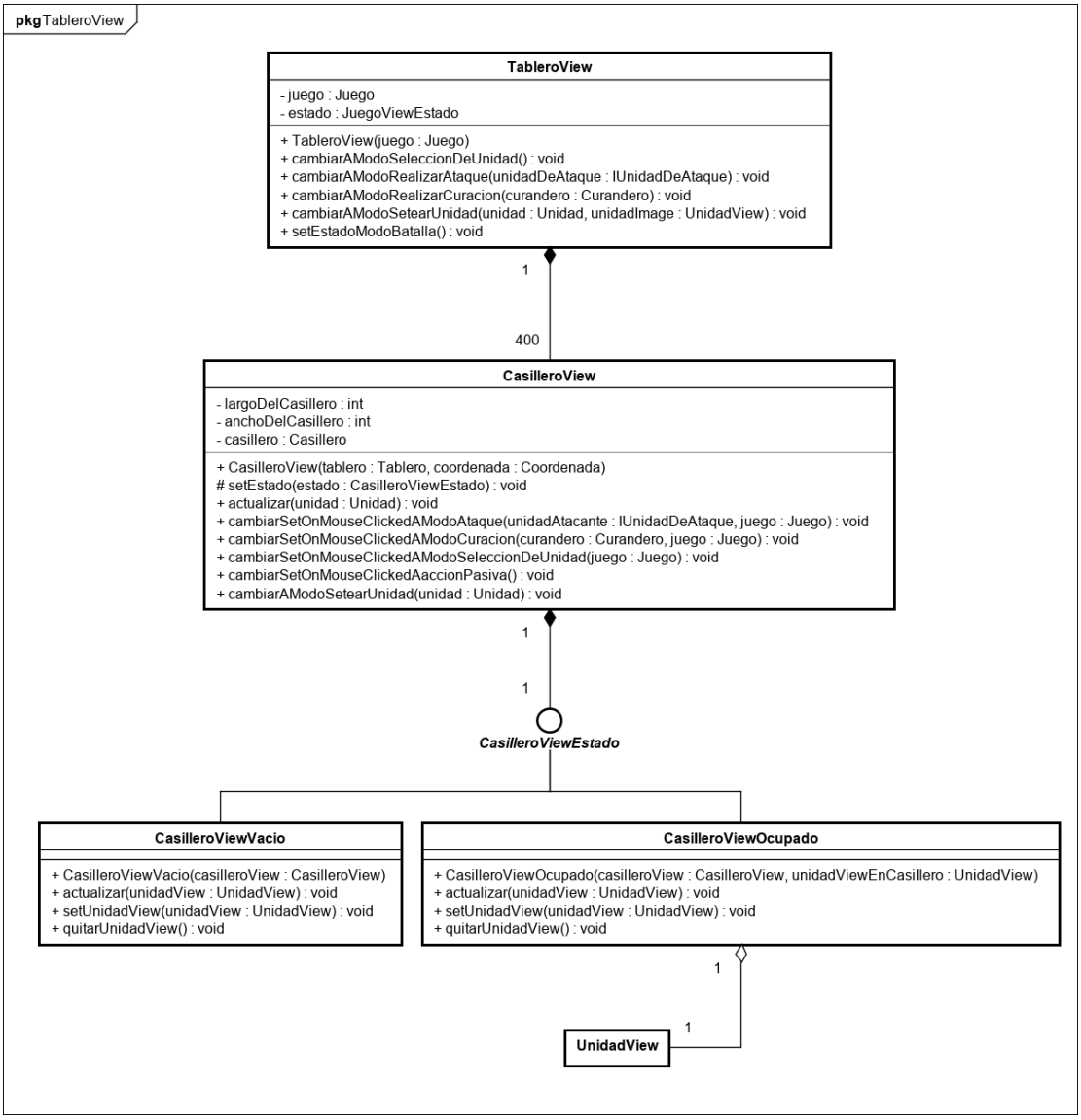

Figura 18: Diagrama de clases - vista tablero.

Para la vista de unidades optamos por implementarlas tan solo como un objeto `ImageView`. De modo que tan solo posee la imagen correspondiente asociada al tipo de unidad y al color de su bando (azul o rojo).

5.2.2. Vista unidades

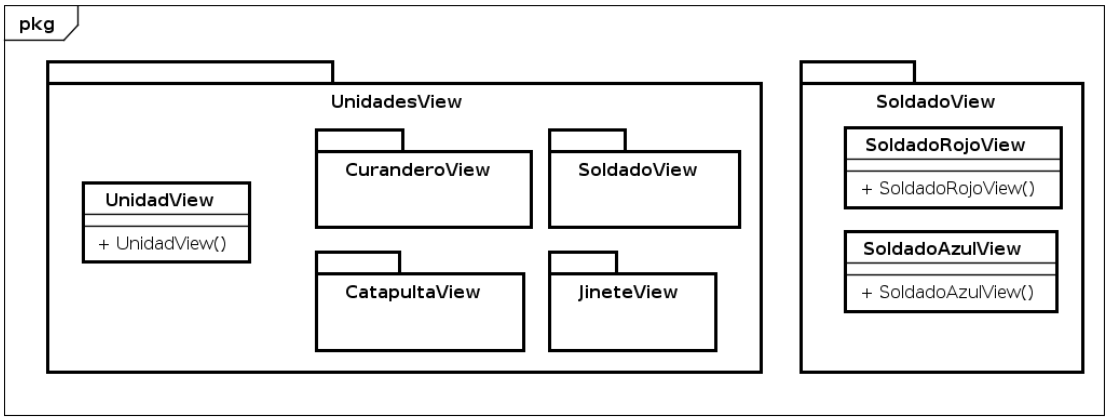

Figura 19: Diagrama de paquete - vista unidades con paquete soldado como ejemplo.

5.2.3. Vista mercado de unidades

Para el posicionamiento de unidades al inicio del juego, optamos por crear el objeto `MercadoDeUnidades`, uno por cada bando. Posee la visualización de las unidades con todas sus características. Su implementación permite seleccionar la unidad que se desee y posicionarla en el tablero. A través del patrón `Observer`, se mantienen actualizados los puntos para compra de unidades que posee el jugador, y va restando dichos puntos a medida que los gasta.

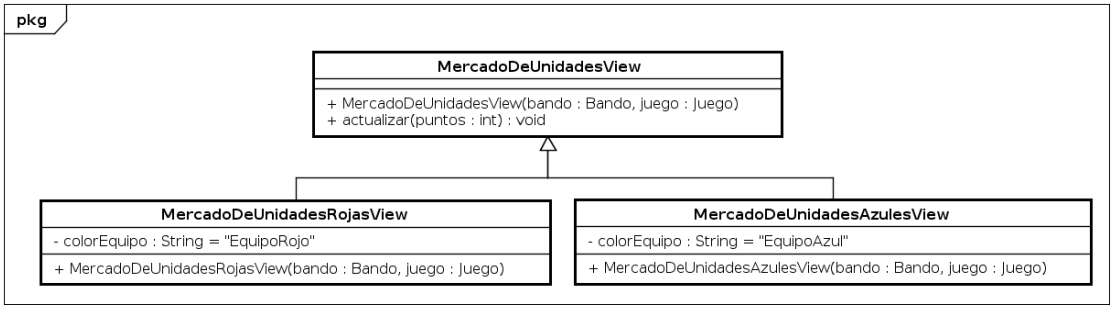

Figura 20: Diagrama de clases - vista mercado de unidades.

5.2.4. Vista panel de control

Finalmente para el otorgarle al jugador el control sobre las unidades, decidimos crear el objeto `PanelDeControl`, el mismo permite una visualización de la unidad y su vida, en conjunto a los controles disponibles para la misma, sobre la parte baja del juego.

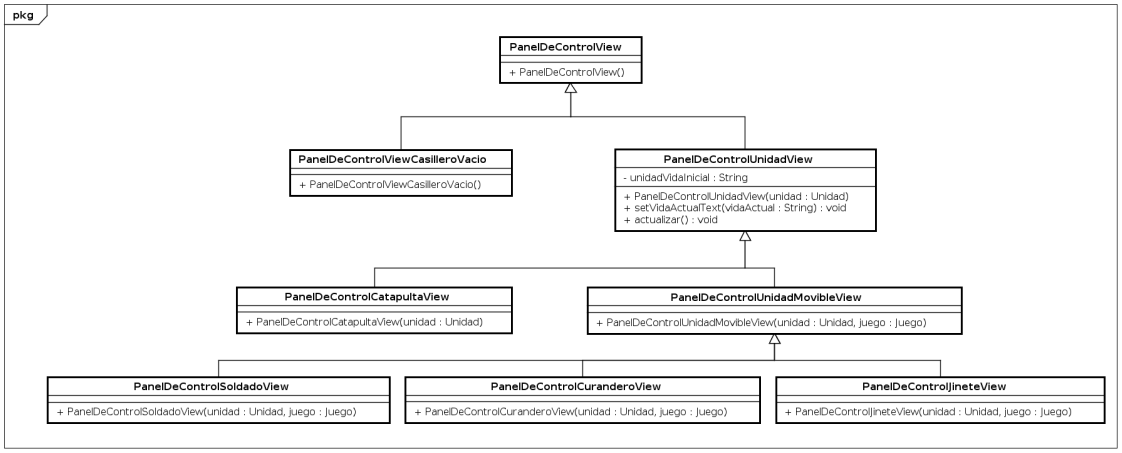

Figura 21: Diagrama de clases - Vista PanelDeControl.

- `PanelDeControlView`: clase base, permite implementar un estilo en común.
- `PanelDeControlViewCasilleroVacio`: visualiza un panel de control que no posee unidad ni botones de control.
- `PanelDeControlUnidadView`: visualiza un panel de control que posee tan solo la imagen de la unidad, con su vida correspondiente (utilizado un patrón `Observer`). Sirve tanto para visualizar unidades enemigas como clase madre para las aliadas, ya que no posee botones de control.
- `PanelDeControlUnidadMovibleView`: proporciona una botonera para realizar movimientos en todas las direcciones.
- Panel de control para cada unidad en específico: para cada unidad se posee un un panel de control que añade los botones partículas para la unidad en cuestión.

5.3. Implementación de los controladores

5.3.1. Controladores que afectan al tablero

Los presentes controladores son utilizados para el `setOnMouseClicked()` de los casilleros, llevando adelante la acción que se encuentra implícita en el nombre de cada cual sobre el modelo.

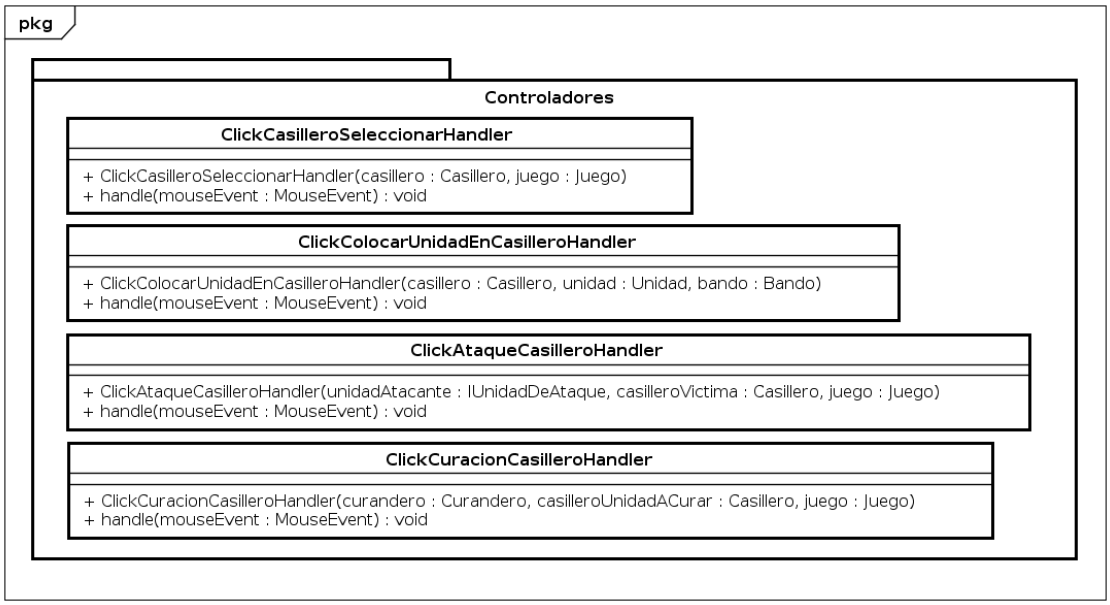

Figura 22: Diagrama de clases - Controladores que afectan al tablero.

5.3.2. Controladores que afectan al panel de control

Los presentes controladores son utilizados para el `setOnMouseClicked()` de los botones del panel de control, llevando adelante la acción que se encuentra implícita en el nombre de cada cual sobre el modelo.

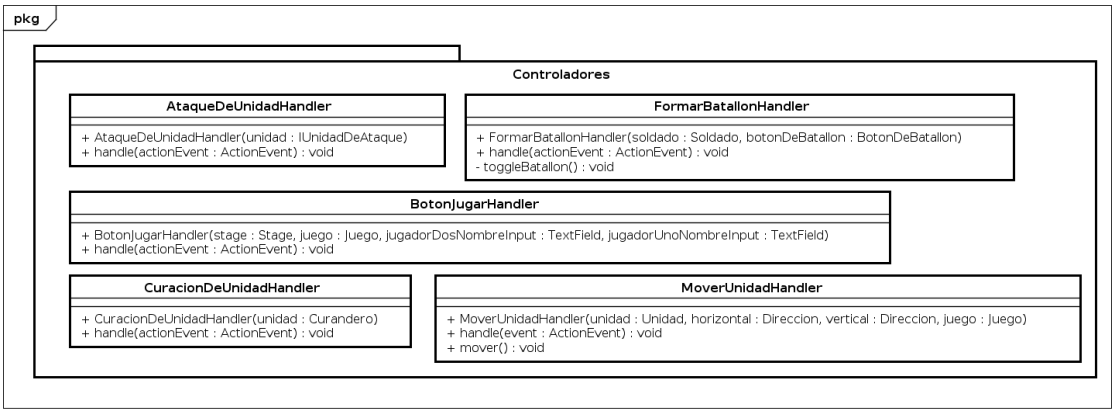

Figura 23: Diagrama de clases - Controladores que afectan al panel de control.

5.3.3. Controladores que afectan al mercado de unidades

Los presentes controladores son utilizados para el `setOnMouseClicked()` del Mercado de unidades, llevando adelante la acción que se encuentra implícita en el nombre de cada cual sobre el modelo.

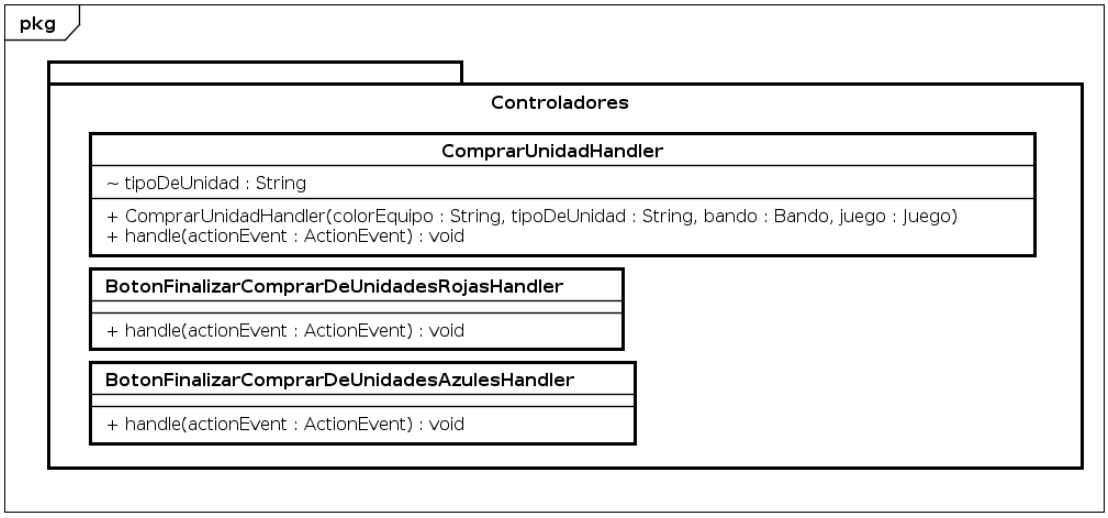

Figura 24: Diagrama de clases - Controladores que afectan al mercado de unidades.

Cabe mencionar que el controlador `ComprarUnidadHandler` hace uso de un `AbstractFactory` de unidades, la misma posee una `Factory` para unidades azules y otra para unidades rojas. Estas ultimas, se encargan de crear la `Unidad`, la `UnidadView` y el `PanelDeControl` para la unidad en cuestión.

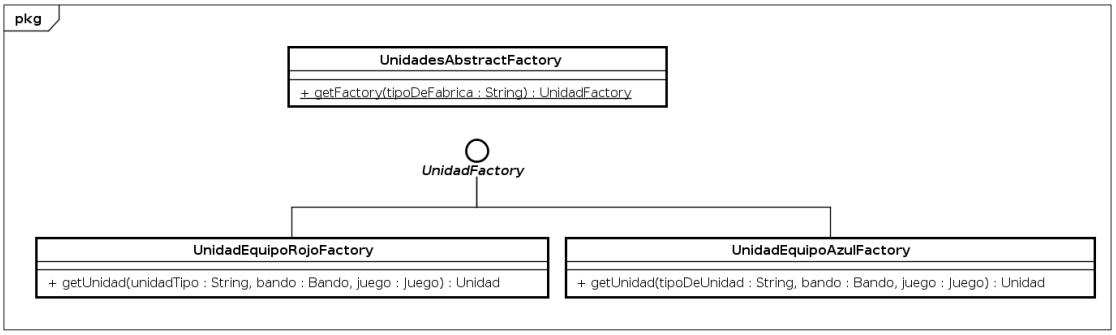

Figura 25: Diagrama de clases - `AbstractFactory` de unidades.

6. Excepciones

6.1. Excepciones Casillero

Excepciones que afectan a las acciones sobre un casillero.

- `CasilleroEnemigoExcepcion`: no se puede colocar la unidad en un casillero enemigo.
- `CasilleroOcupadoExcepcion`: el casillero se encuentra ocupado.
- `CasilleroVacioExcepcion`: el casillero se encuentra vacío.

6.2. Excepciones Jugador

Excepciones que afectan al jugador.

- `JugadorSinUnidadesExcepcion`: El jugador se queda sin unidades.

- `PuntosInsuficientesExcepcion`: El jugador no tiene suficientes puntos para comprar la unidad.
- `UnidadInvalidaExcepcion`: El jugador intenta realizar una acción sobre una unidad que no soporta dicha acción.

6.3. Excepciones Tablero

- `CasilleroFueraDeLosLimitesDelTableroExcepcion`: Se intenta acceder a un casillero a través de unas coordenadas que no existen en el tablero

6.4. Excepciones de Unidades

6.4.1. Excepciones Estrategia de Ataque

- `DistanciaDeAtaqueIncorrectaExcepcion` La unidad intenta atacar a una posición que no se encuentra a una distancia correcta.
- `UnidadAtacadaEsAliadaExcepcion` La unidad intenta atacar a una unidad aliada.

6.4.2. Excepciones Batallón

- `BatallonIncompletoExcepcion` Se intenta formar un batallón sin la cantidad necesaria de soldados.
- `BatallonNoFormadoExcepcion` Se intenta realizar una acción de batallón cuando este no esta formado.

6.4.3. Excepciones Curar

- `AlidadoConSaludCompletaNoSePuedeCurarExcepcion` Se intenta curar a una unidad con salud completa.
- `CatapultaNoPuedeSerCuradaExcepcion` Se intenta curar a una catapulta.
- `DistanciaParaCurarIncorrectaExcepcion` Se intenta curar a una unidad que no se encuentra a la distancia correcta.
- `UnidadCuradaEsEnemigaExcepcion` Se intenta curar a una unidad enemiga.

6.4.4. Excepciones Movimiento

- `MovimientoNoContiguoExcepcion` Se intenta mover a un casillero que no es contiguo al actual.