FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — A.L.T.E.G.O.

[7507/9502] Algoritmos y Programación III
Curso 1
Primer cuatrimestre de 2021

Alumno	**Padrón**	**Email**	**Github**
Quintero, Nazareno Hernán	105296	nquintero@fi.uba.ar	NazaQuintero
Patiño, Franco	105126	fpatino@fi.uba.ar	mrinkspot
Bartocci, Camila	105781	cbartocci@fi.uba.ar	camilabartocci
Bocanegra, Eduardo Martín	106028	ebocanegra@fi.uba.ar	martinboca
Medone Sabatini, Juan Ignacio	103878	jmedone@fi.uba.ar	juanimedone

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación del juego de mesa T.E.G. en Java utilizando los conceptos del paradigma de la orientación a objetos vistos en el curso.

2. Supuestos

Debido a que ciertas especificaciones no fueron provistas por la cátedra y en ciertas ocasiones el criterio de cómo abordar alguna situación quedaba a cargo del alumno, se tuvieron en cuenta los siguientes supuestos:

- Cuando un jugador recibe ejércitos por posesión de un continente, puede colocarlos en cualquier país que le pertenezcan (no necesariamente de ese continente).
- En el momento del reagrupe, pueden realizarse ilimitados traspasos de una ficha a otros países (siempre y cuando sean países del jugador que corresponde y no quede ningún país vacío).
- Si un jugador no coloca todos los ejércitos que tiene disponibles en la ronda de colocación, los pierde (no se acumulan).
- Cuando se realiza una conquista de un país en un ataque, se pasa una sola ficha.
- Si un jugador recibe una tarjeta y no posee ese país, puede activarla en un turno posterior si conquista ese país.
- Las tarjetas de países pueden activarse en rondas de reagrupe y colocación.

3. Diagramas de clases

En esta sección se encuentran todas las entidades que fueron utilizadas y las relaciones estáticas entre ellas en los siguientes diagramas:

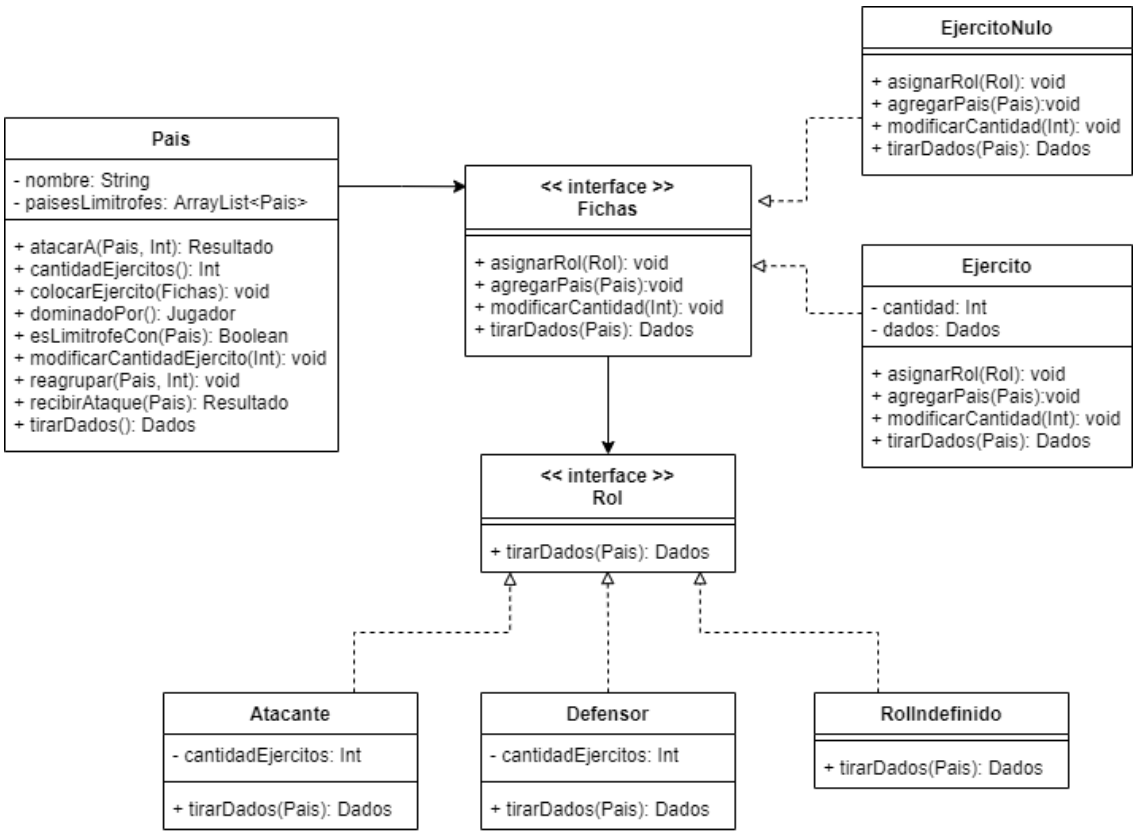

Figura 1: Entidades relacionadas con Pais

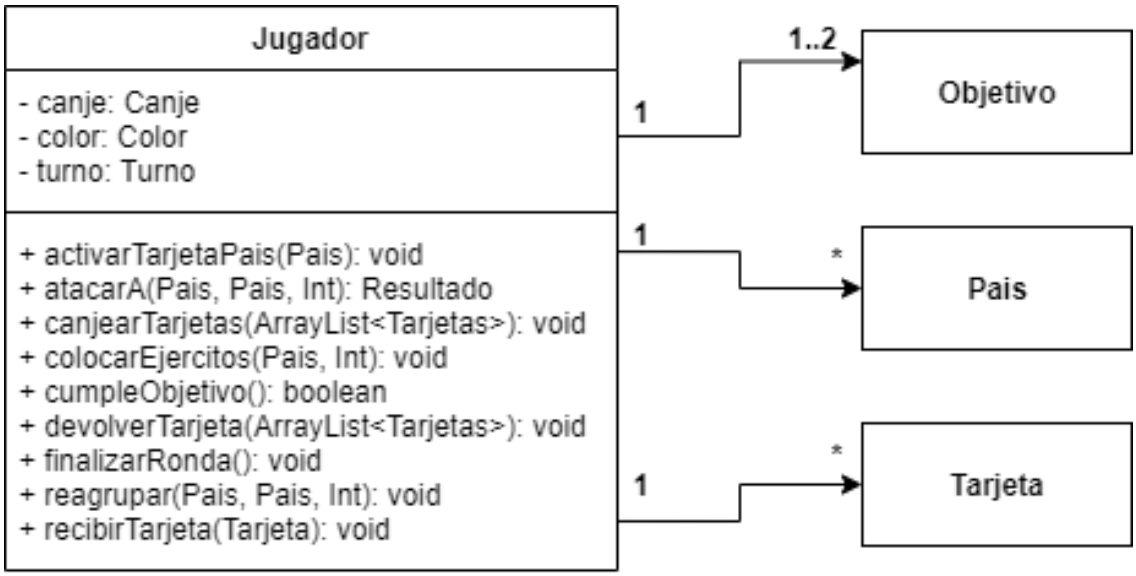

Figura 2: Entidades relacionadas con Jugador.

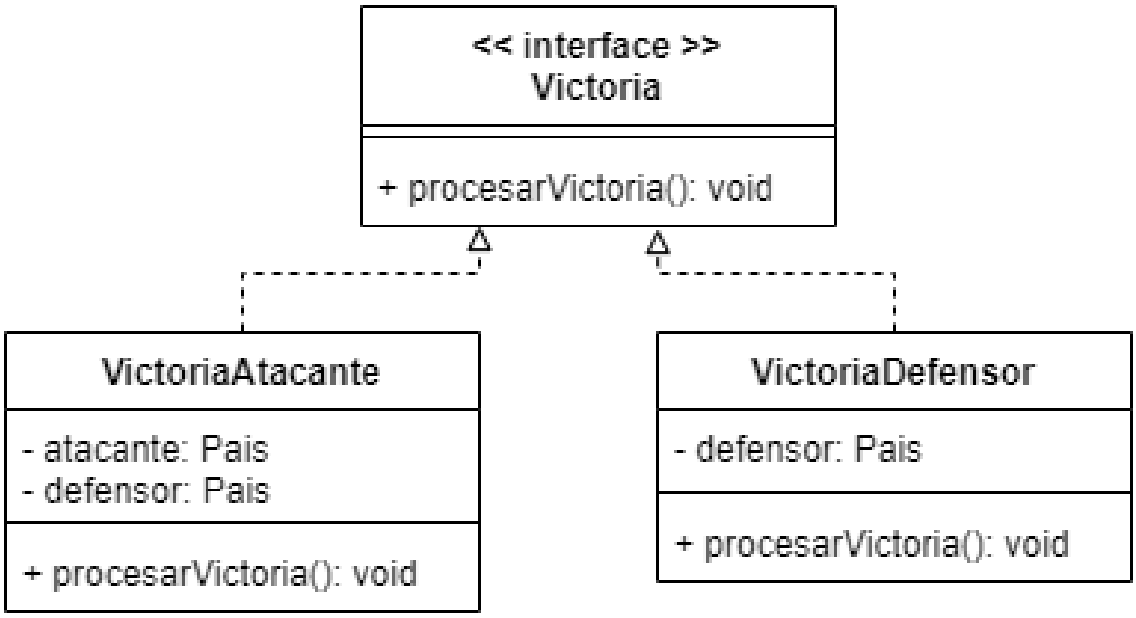

Figura 3: Interfaz Victoria.

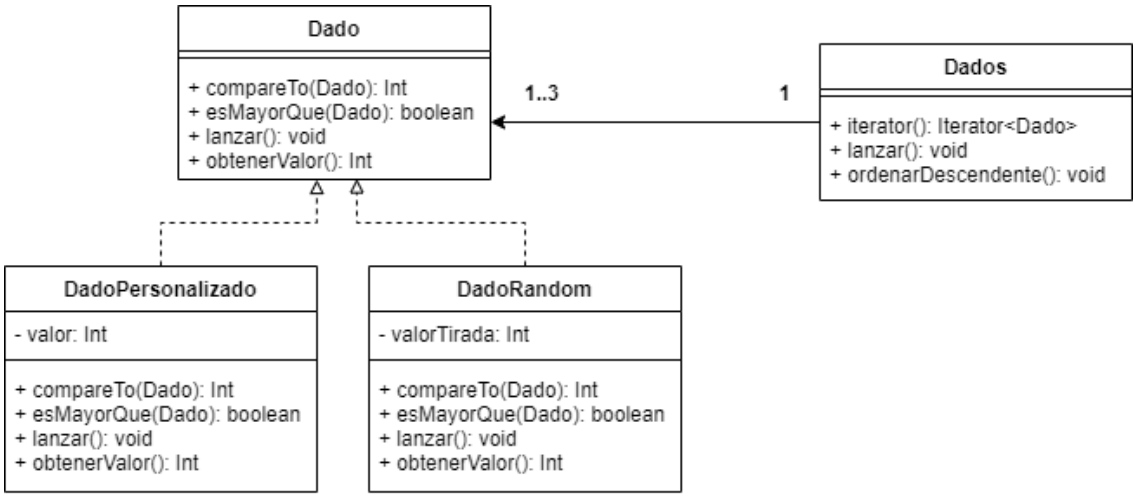

Figura 4: Entidades relacionadas con los dados.

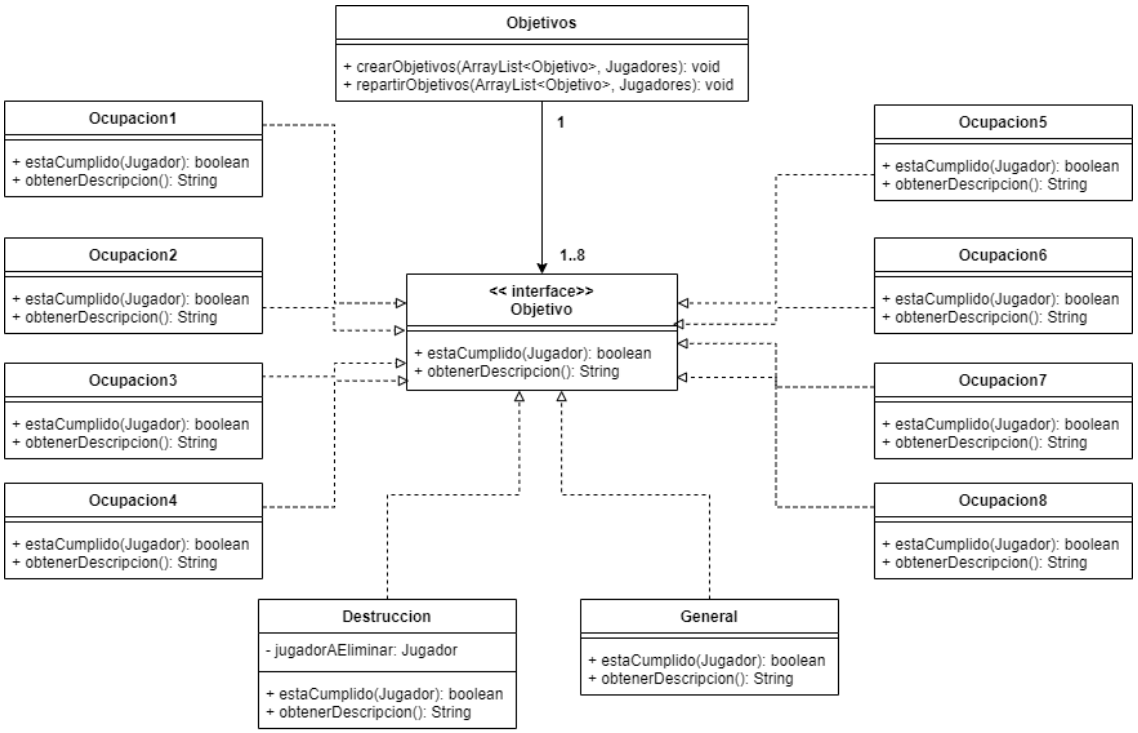

Figura 5: Distintos tipos de objetivos.

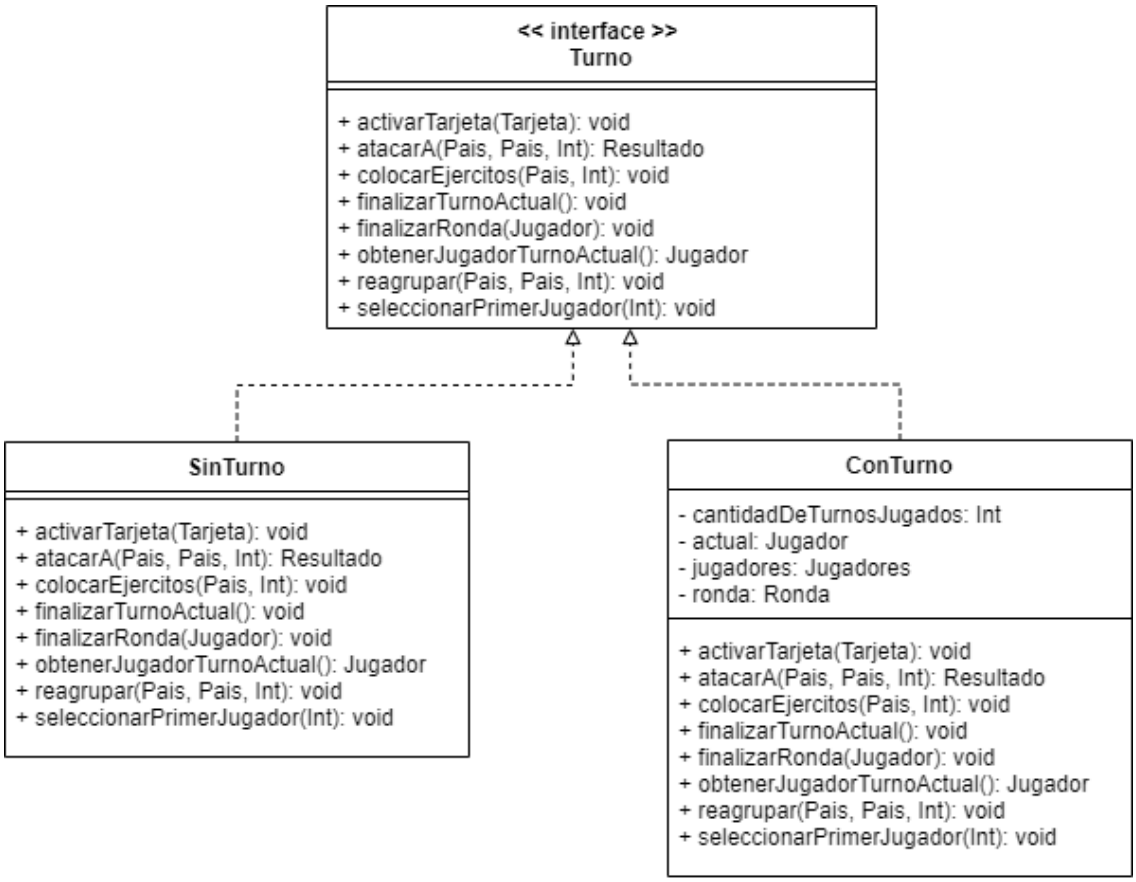

Figura 6: Interfaz Turno.

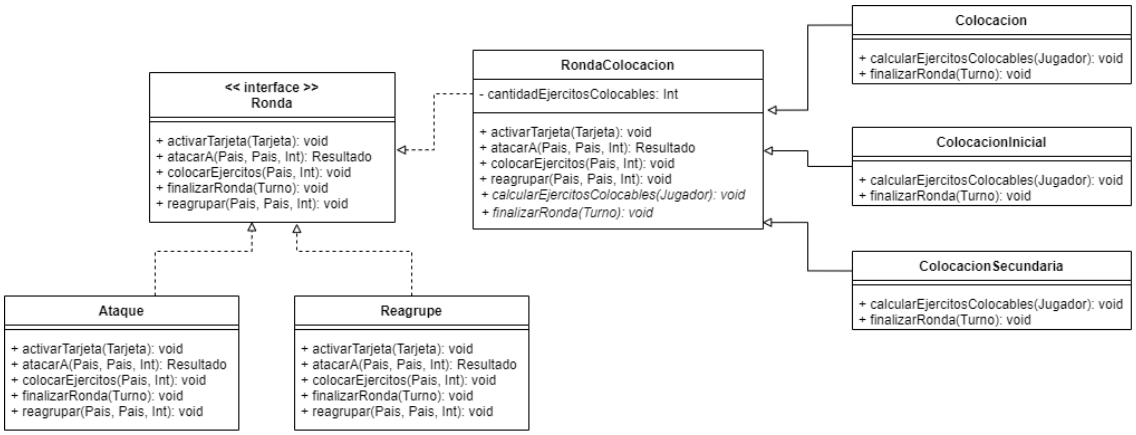

Figura 7: Entidades relacionadas a las rondas.

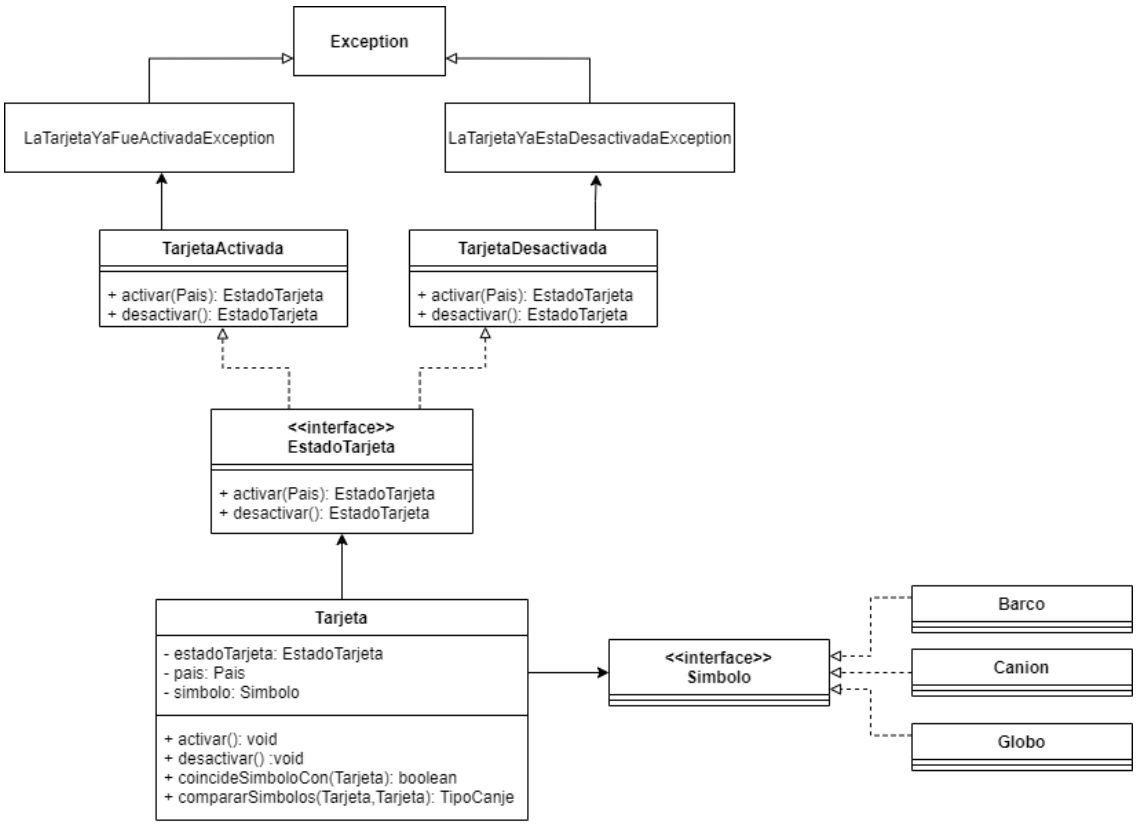

Figura 8: Entidades relacionadas a Tarjeta.

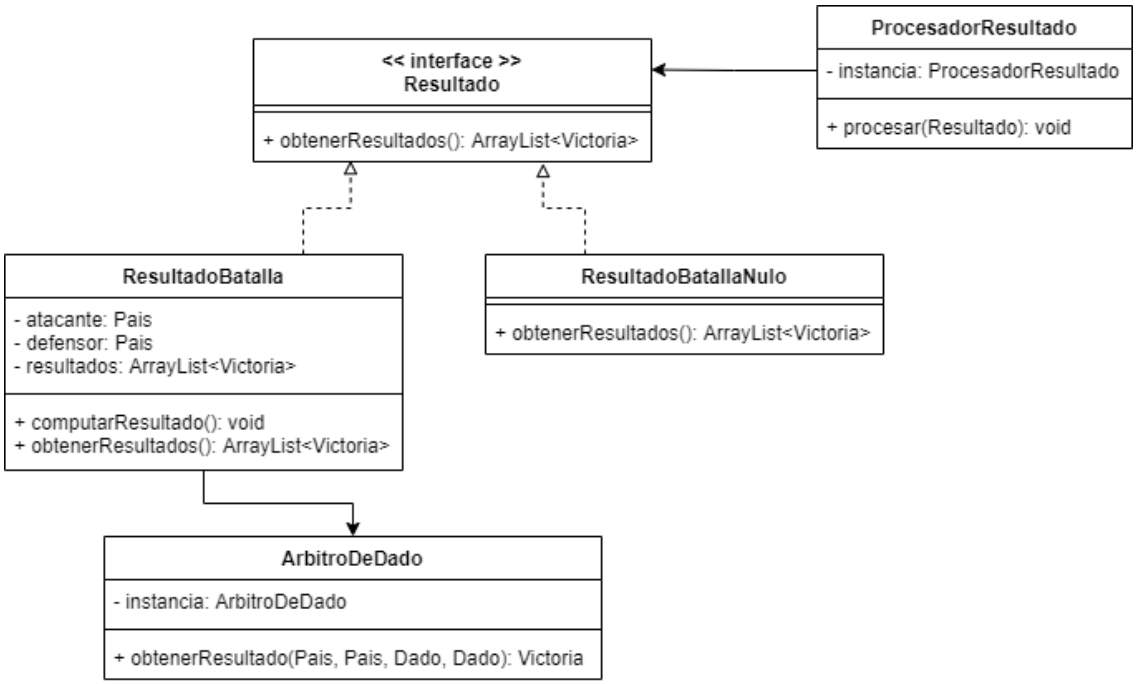

Figura 9: Entidades relacionadas a las batallas.

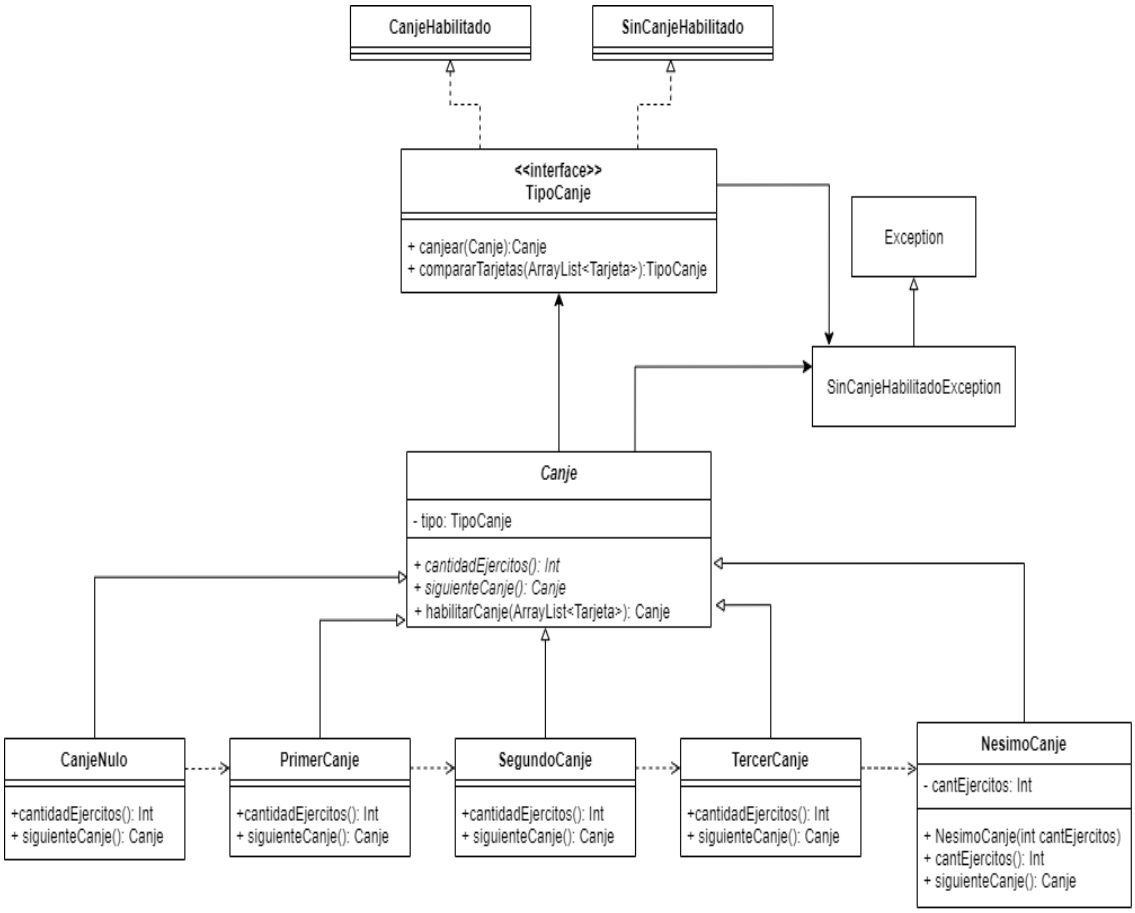

Figura 10: Entidades relacionadas a Canje.

4. Diagramas de secuencia

En los siguientes diagramas, las entidades subrayadas representan instancias de esas mismas clases y las no subrayadas representan abstracciones (interfaces).

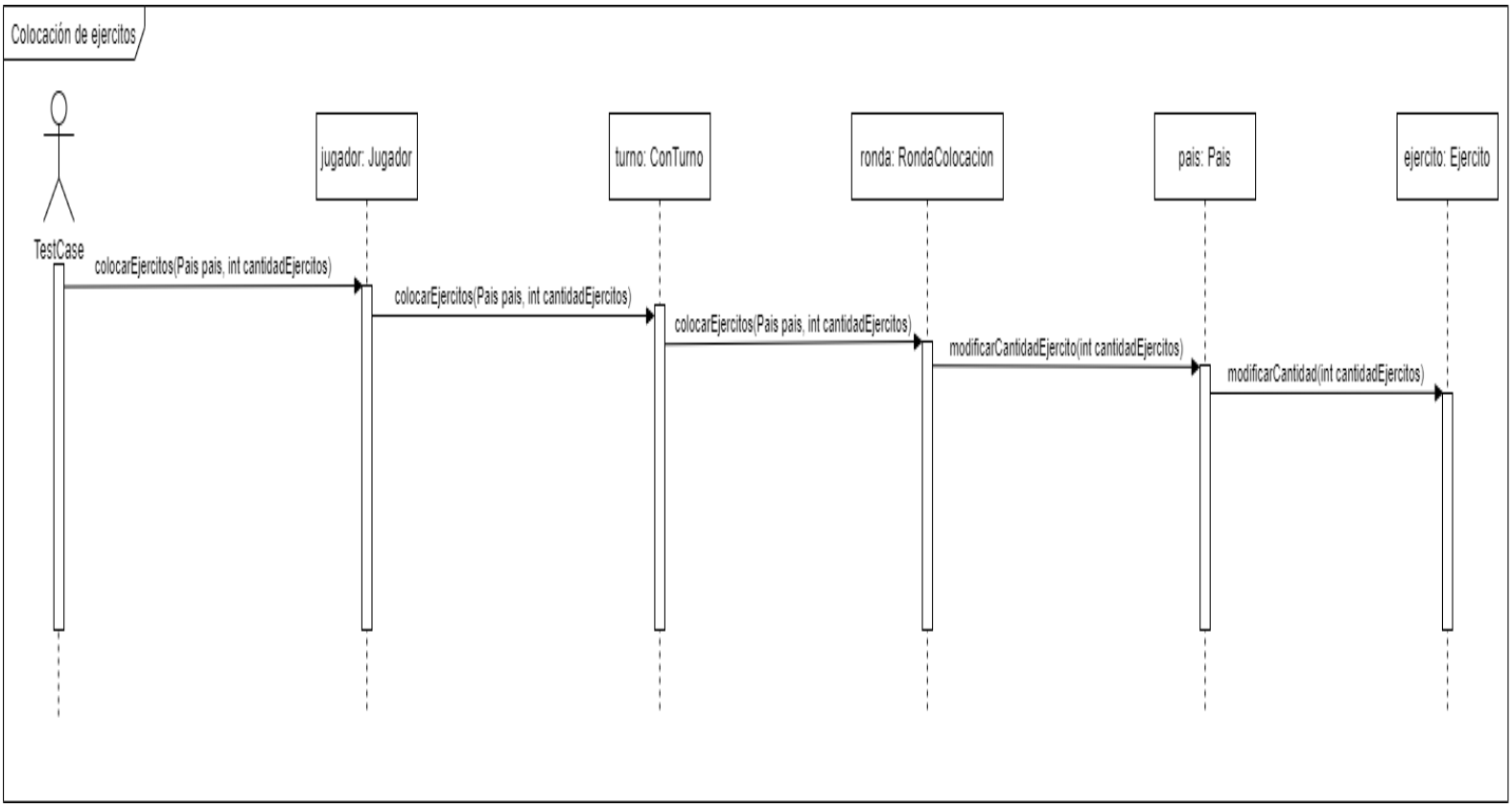

Figura 11: Diagrama de secuencia Colocación de ejércitos.

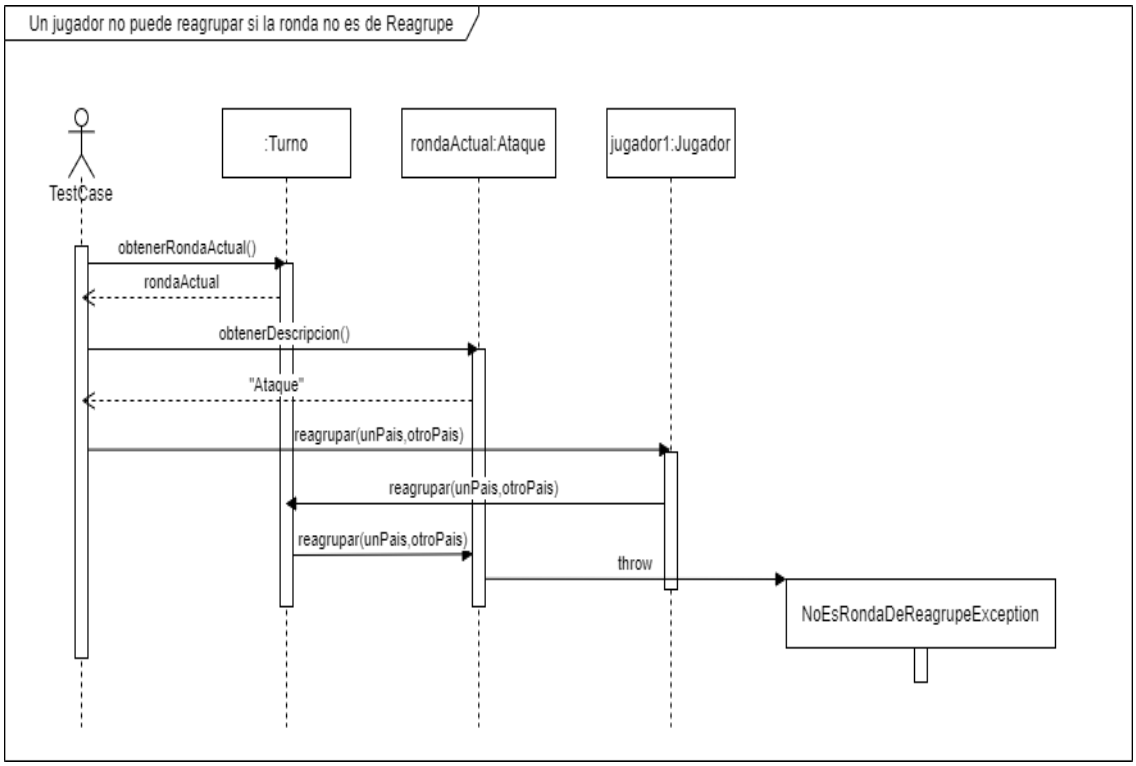

Figura 12: Diagrama de secuencia reagrupe en ronda no válida.

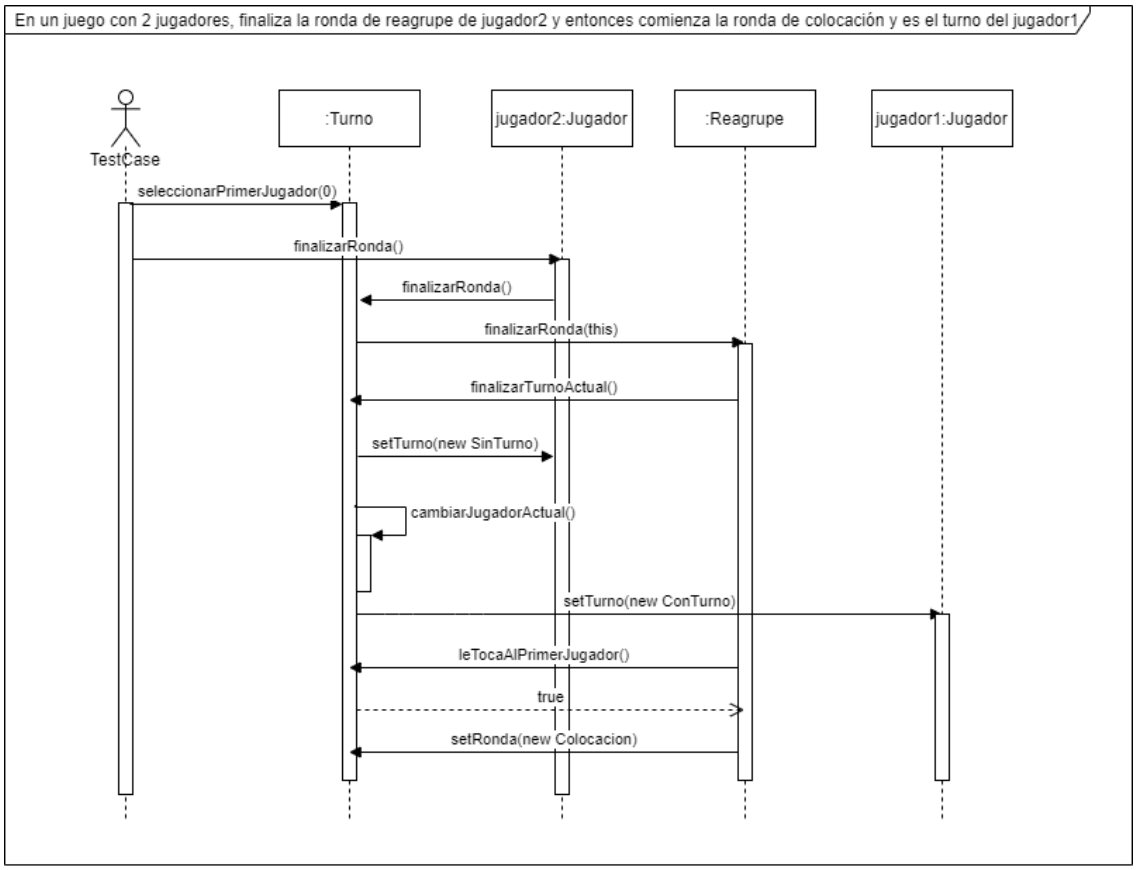

Figura 13: Diagrama de secuencia de Reagrupe a Colocación.

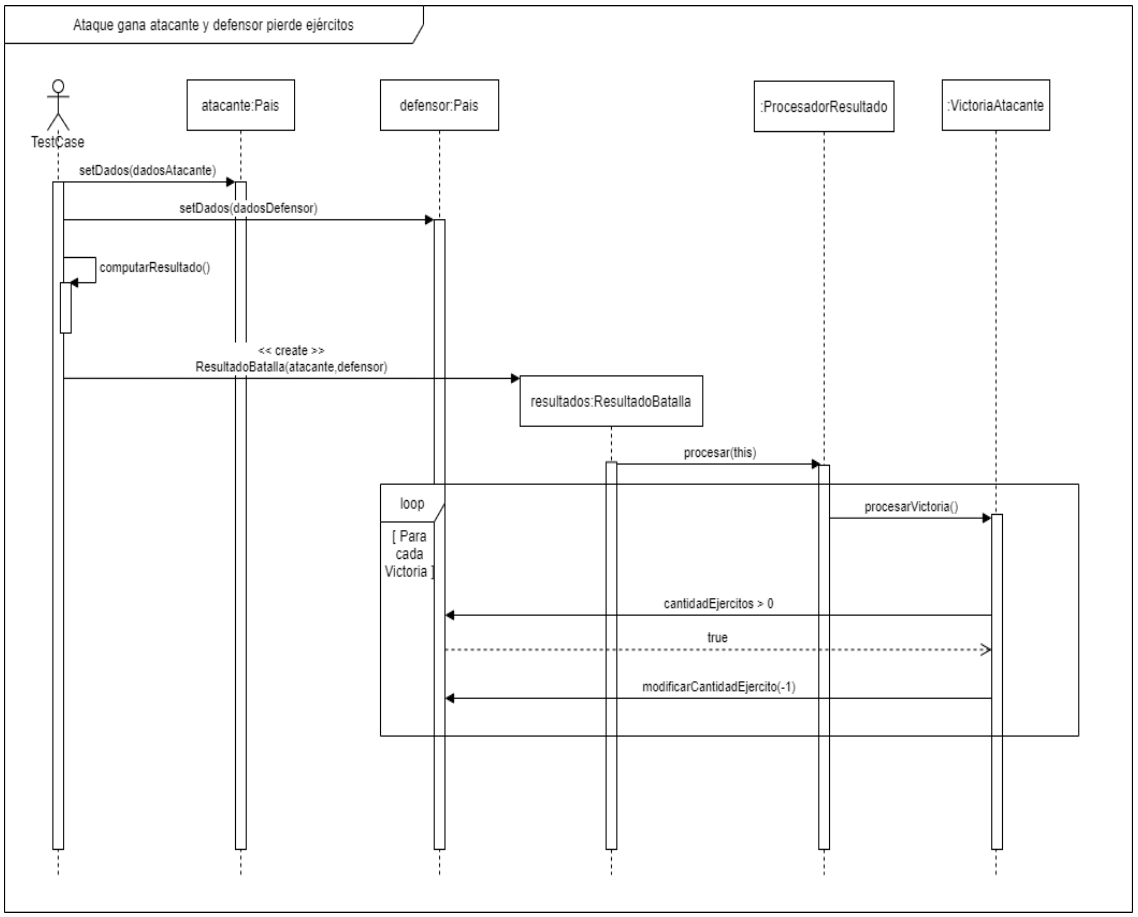

Figura 14: Diagrama de secuencia Ataque (gana atacante y el país defensor pierde ejércitos).

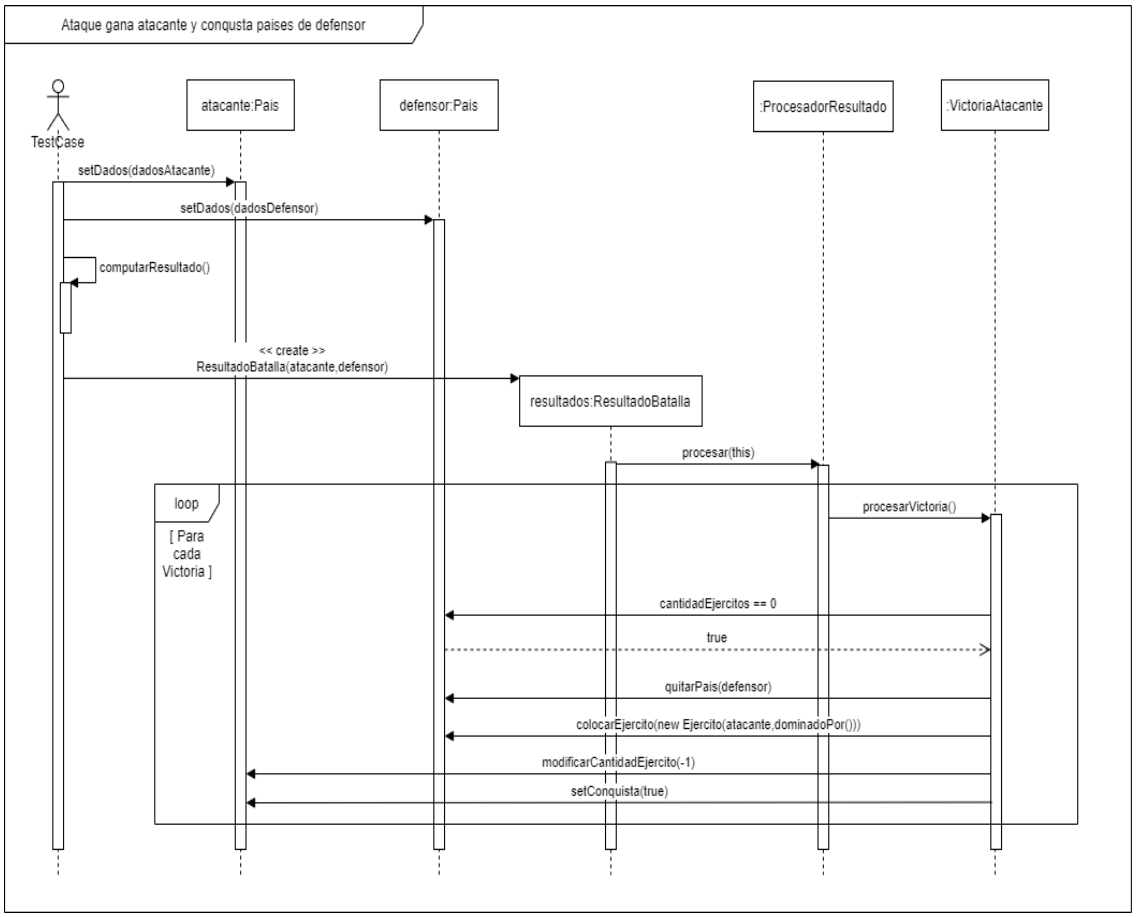

Figura 15: Diagrama de secuencia Ataque (gana atacante y conquista País del defensor).

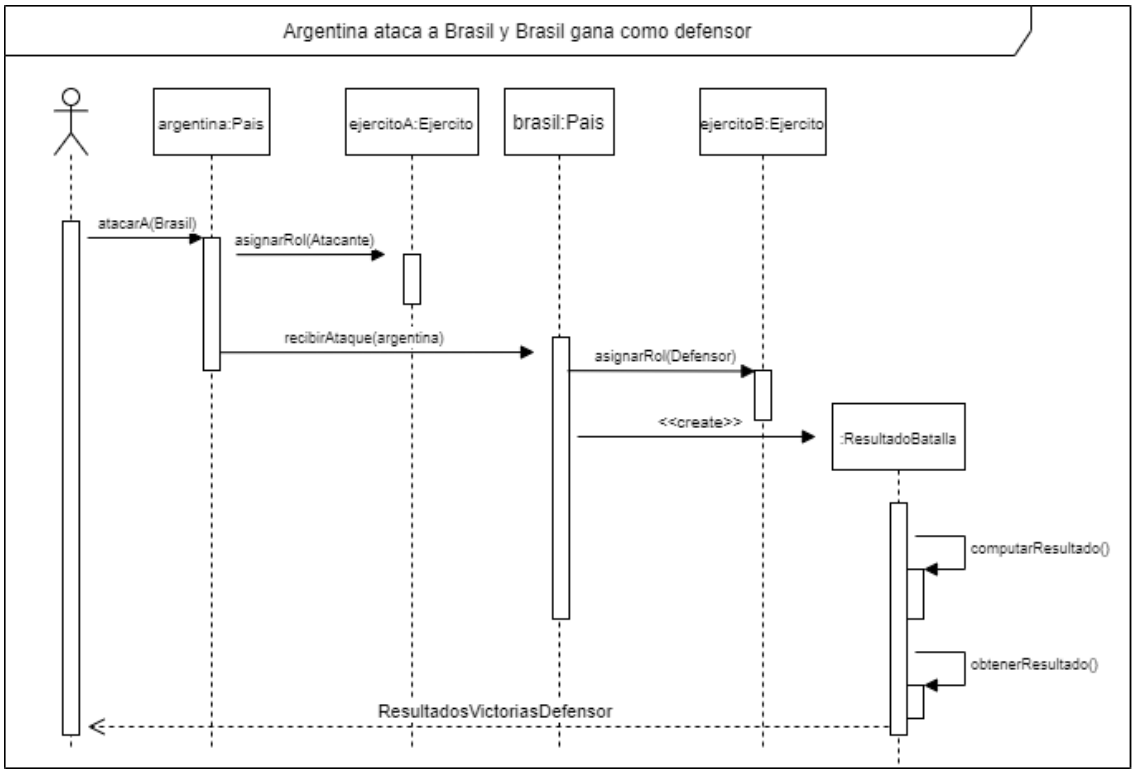

Figura 16: Diagrama de secuencia Ataque (gana atacante y conquista País del defensor).

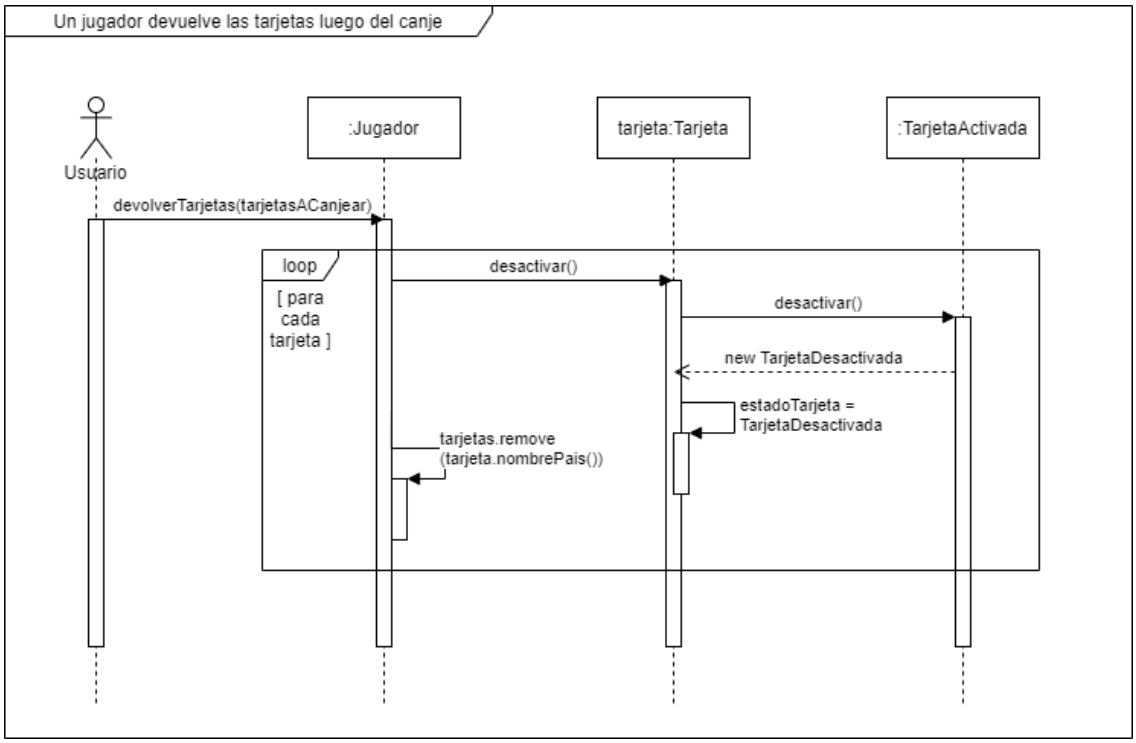

Figura 17: Diagrama de secuencia devolver Tarjetas.

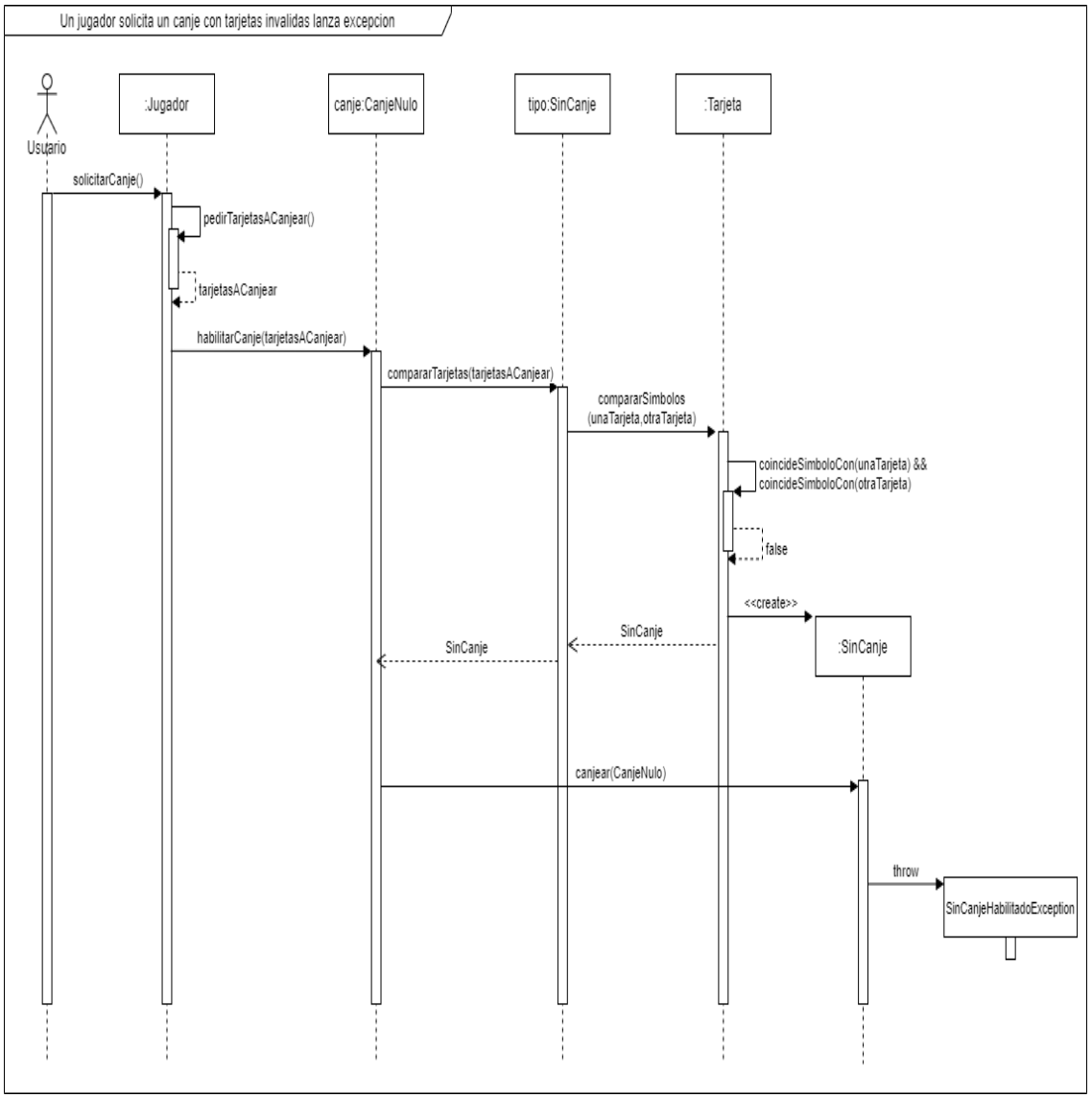

Figura 18: Diagrama de secuencia canje inválido.

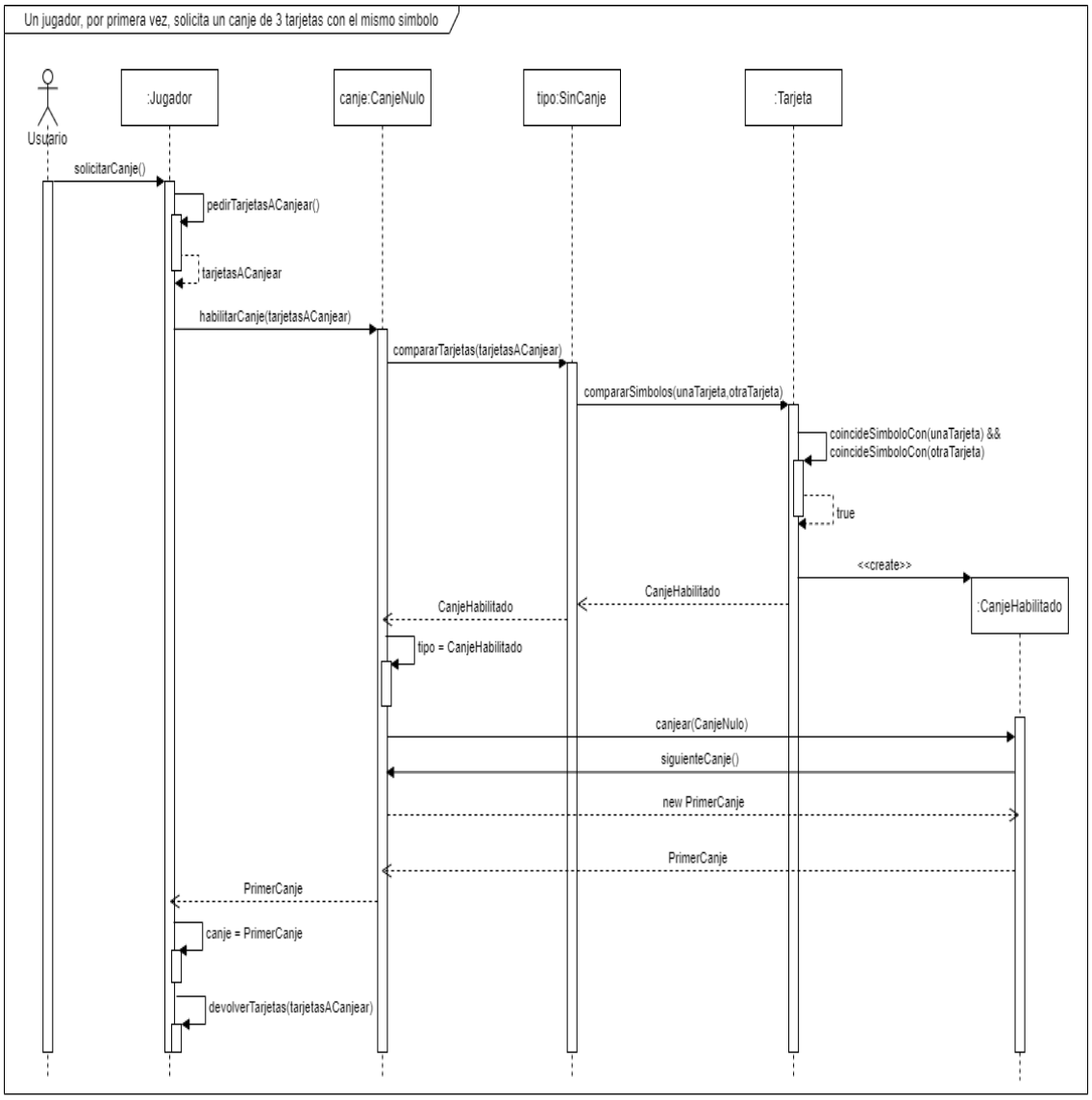

Figura 19: Diagrama de secuencia canje válido.

5. Diagramas de paquetes

En este apartado se muestra la organización del proyecto.

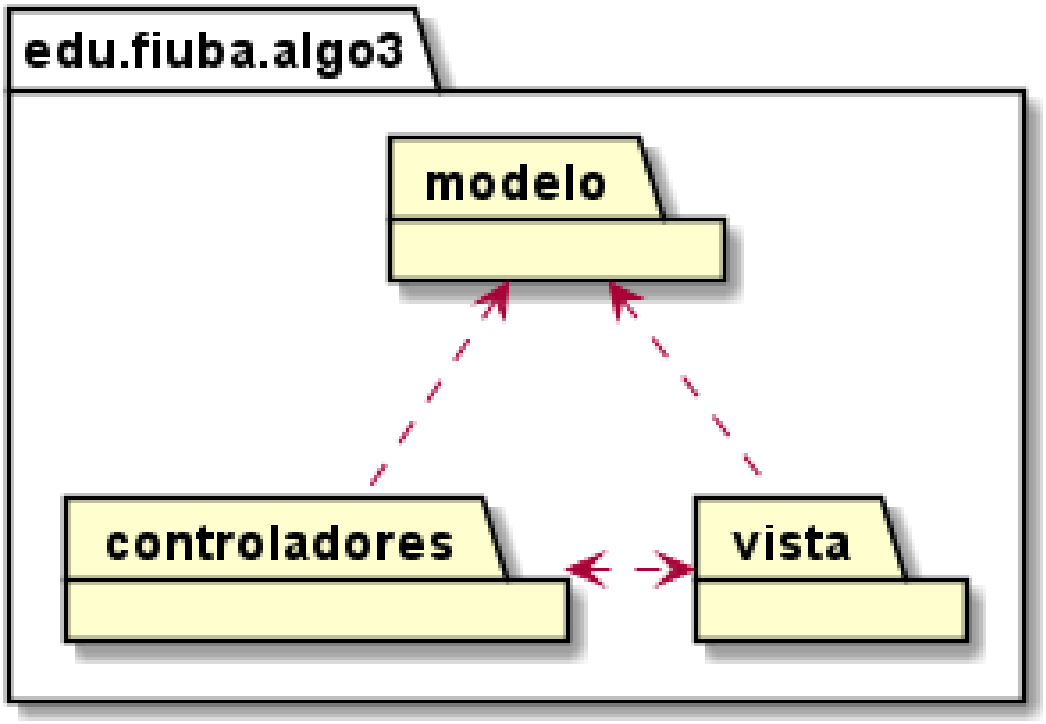

Figura 20: Patrón Modelo-Vista-Controlador (MVC)

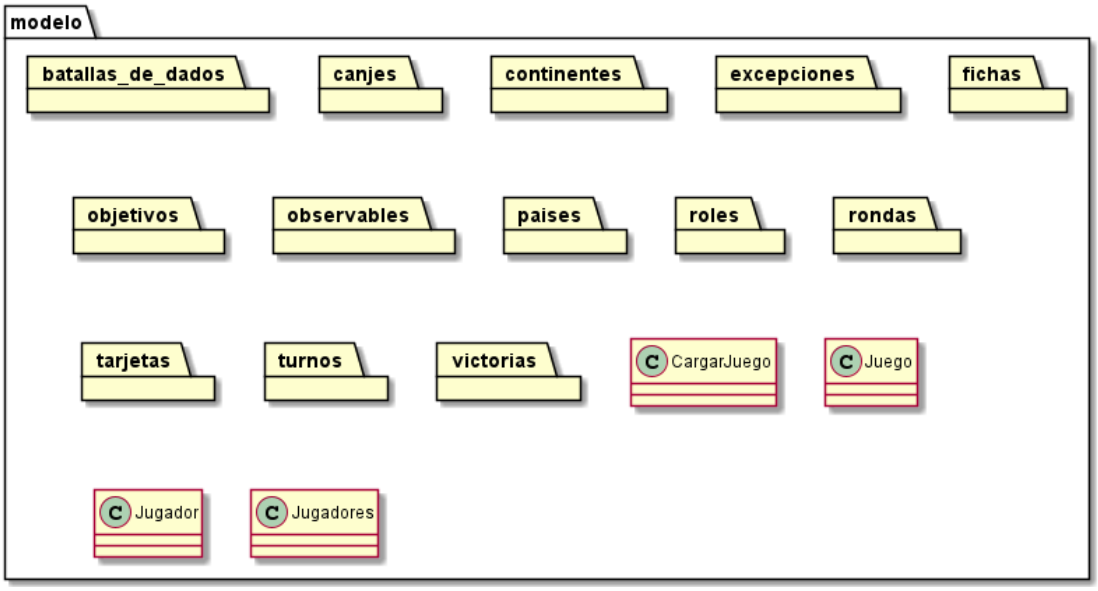

Figura 21: Subpaquetes del modelo

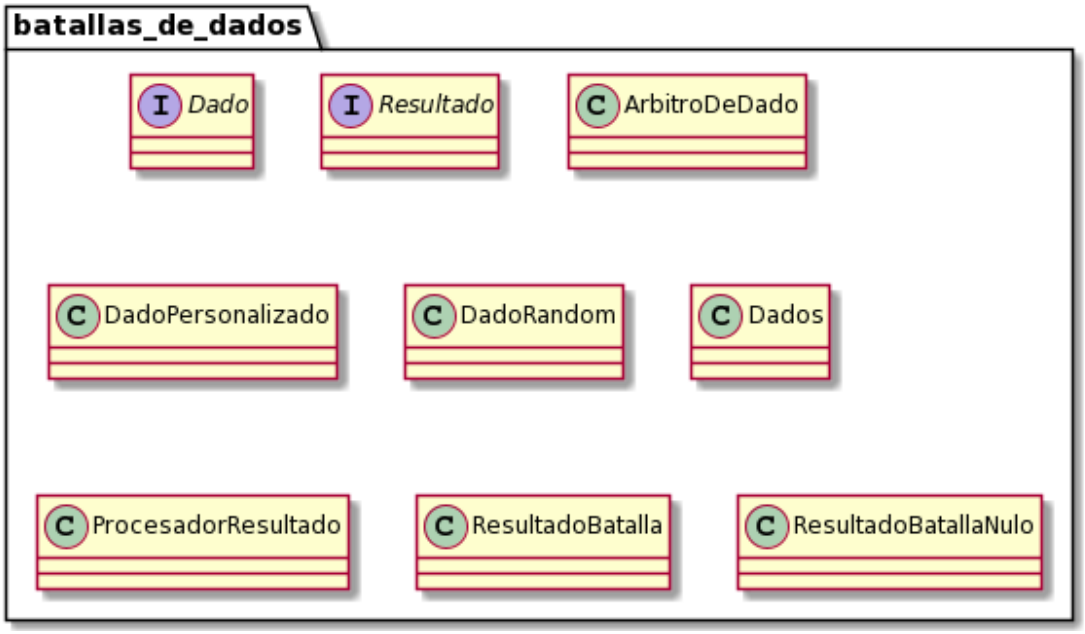

Figura 22: Paquete batallas_de_dados

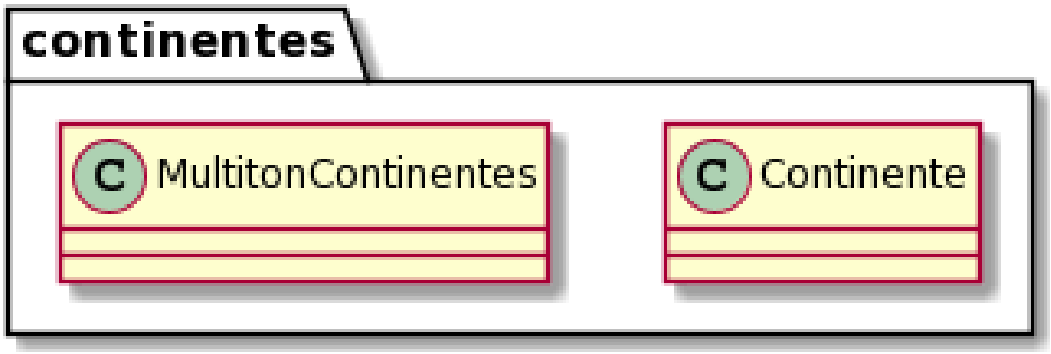

Figura 23: Paquete continentes

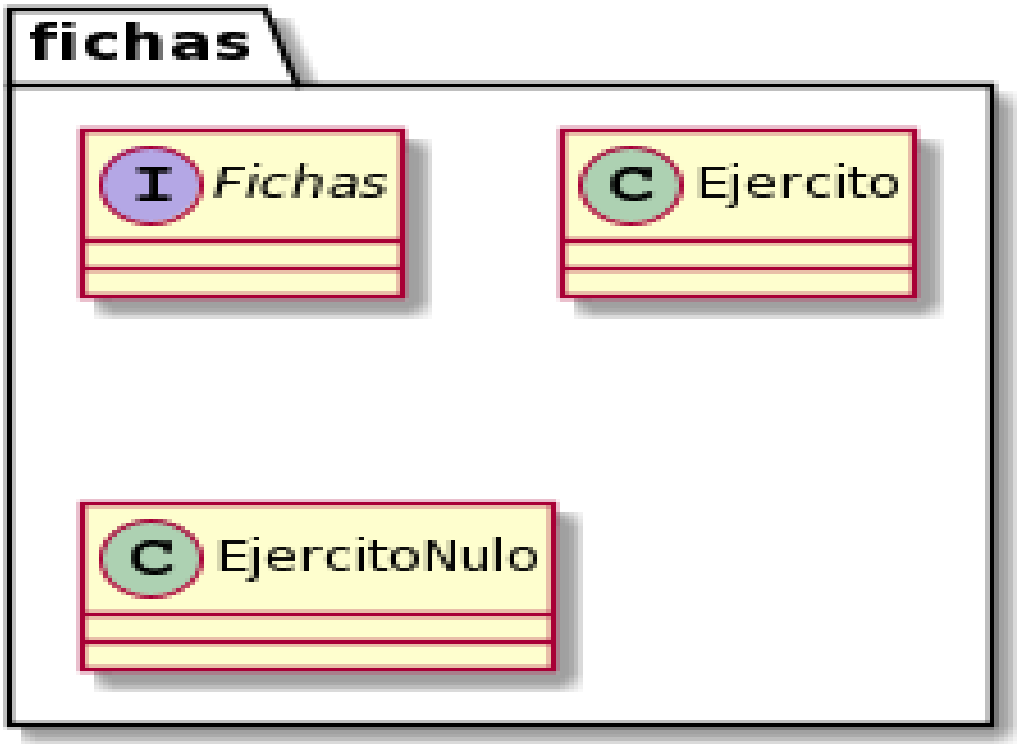

Figura 24: Paquete fichas

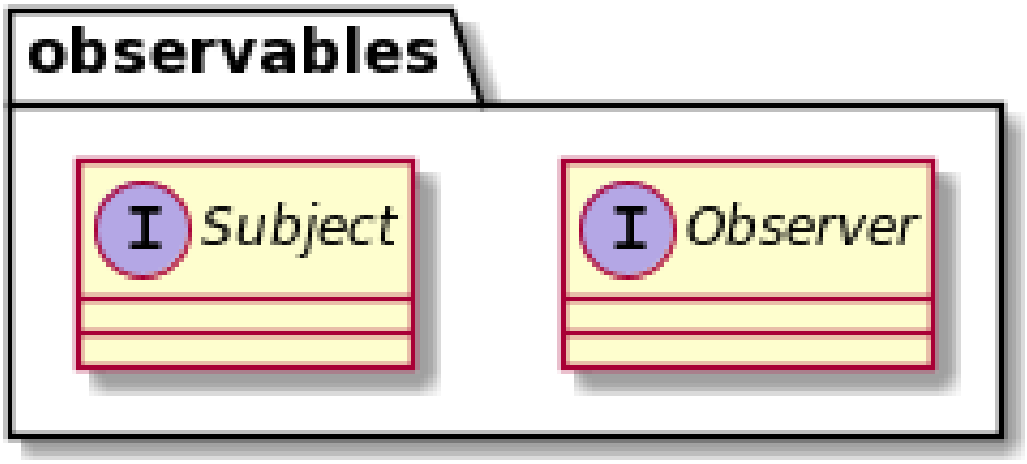

Figura 25: Paquete observables

Figura 26: Paquete objetivos

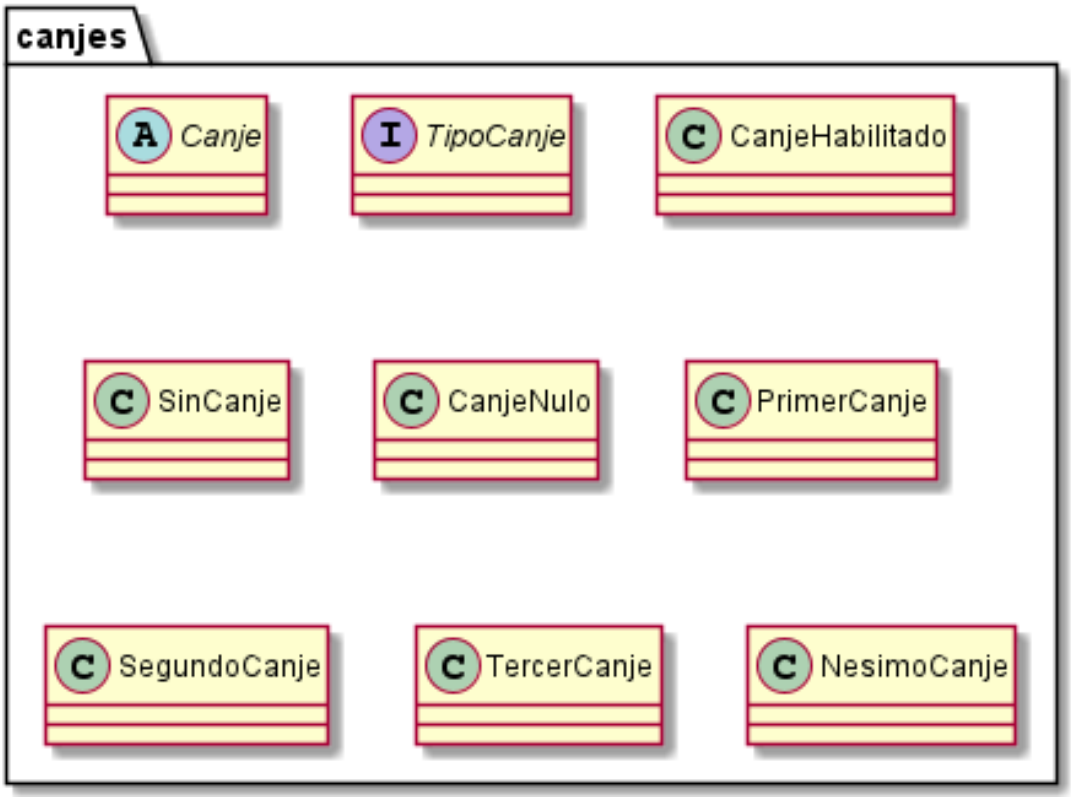

Figura 27: Paquete canjes

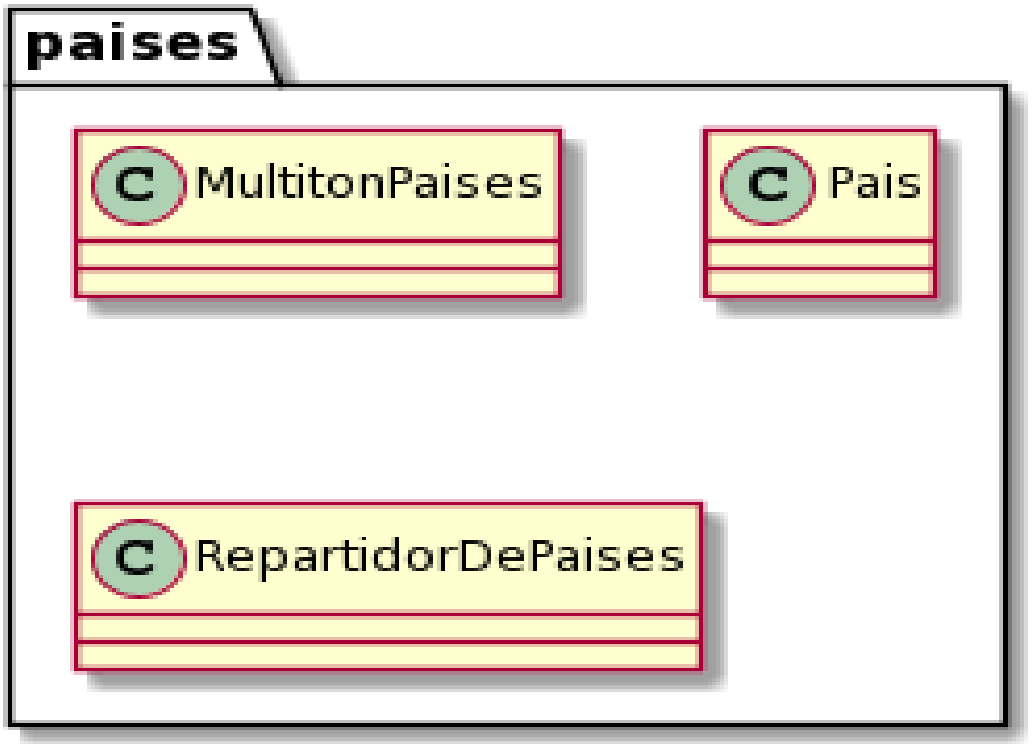

Figura 28: Paquete paises

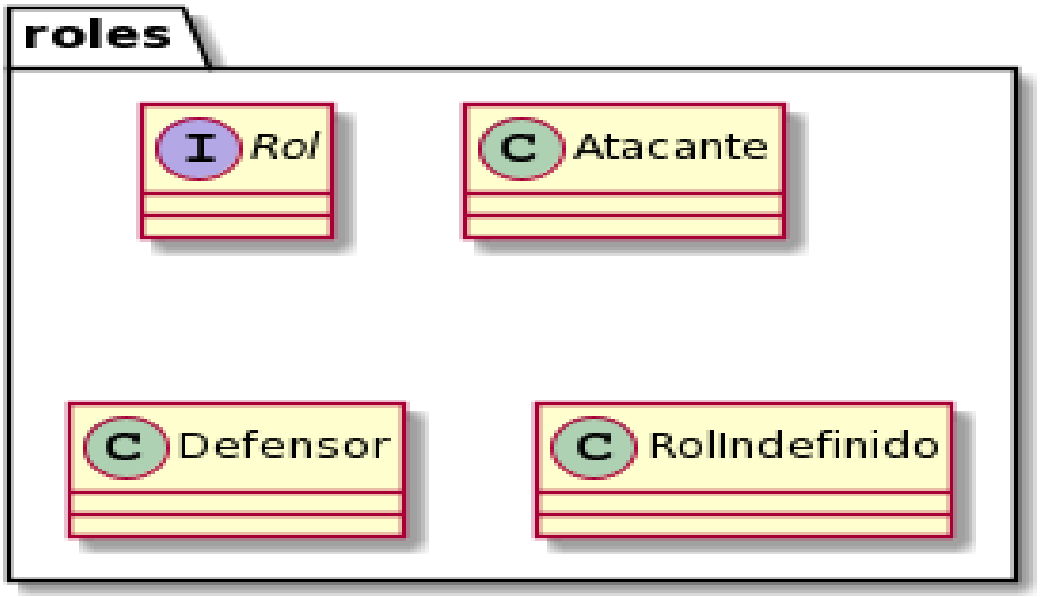

Figura 29: Paquete roles

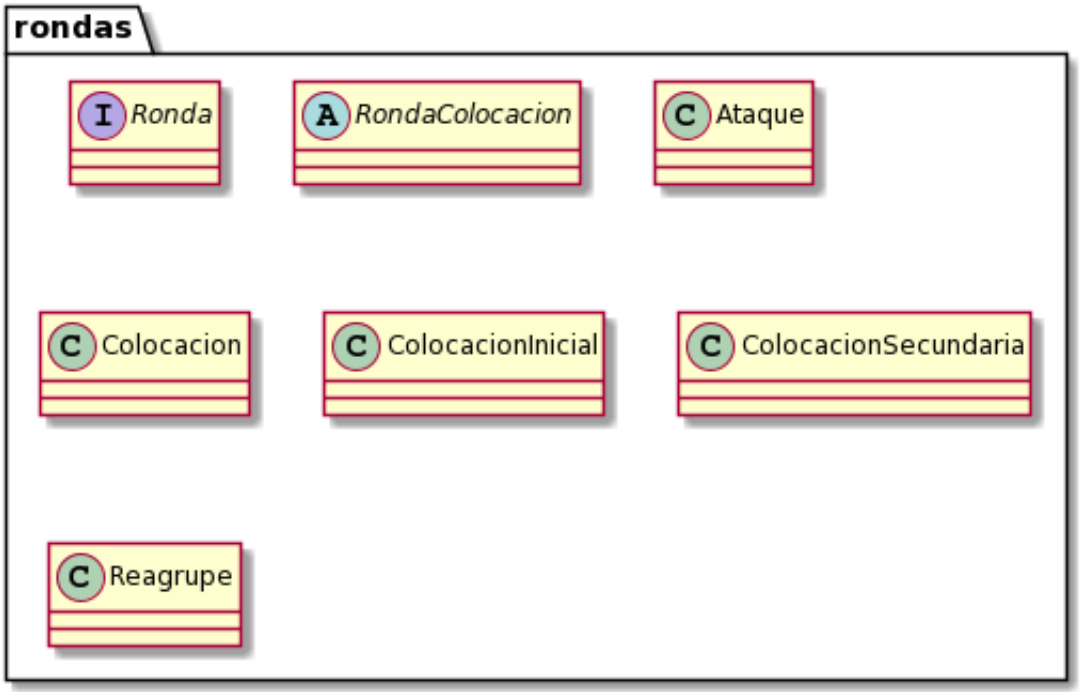

Figura 30: Paquete rondas

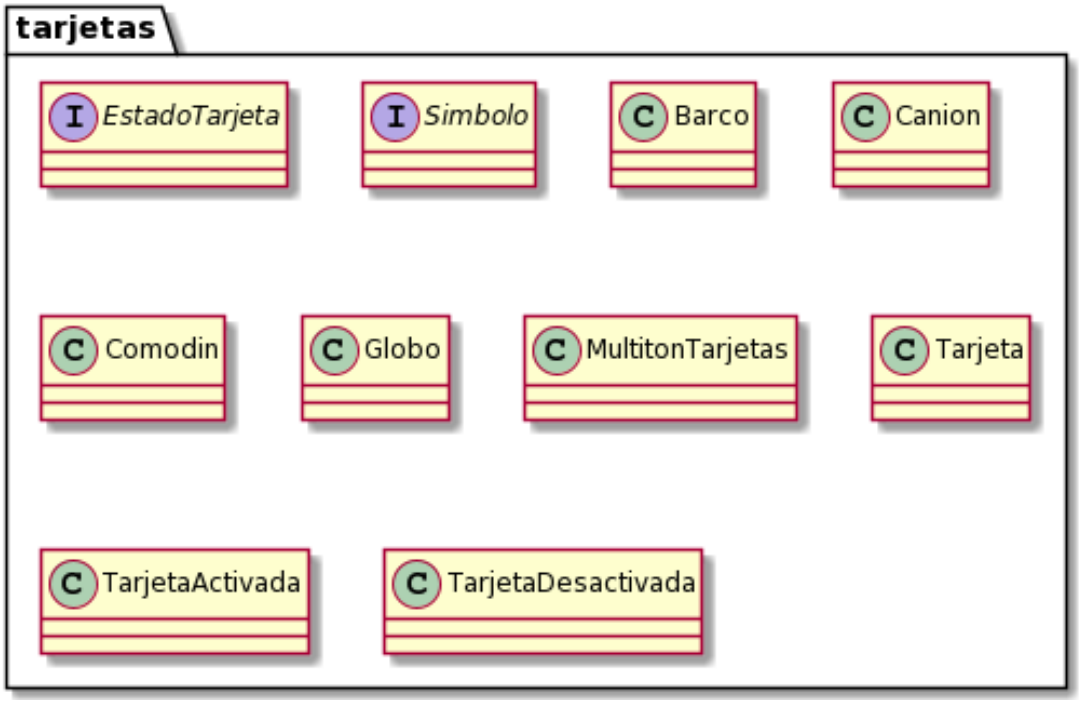

Figura 31: Paquete tarjetas

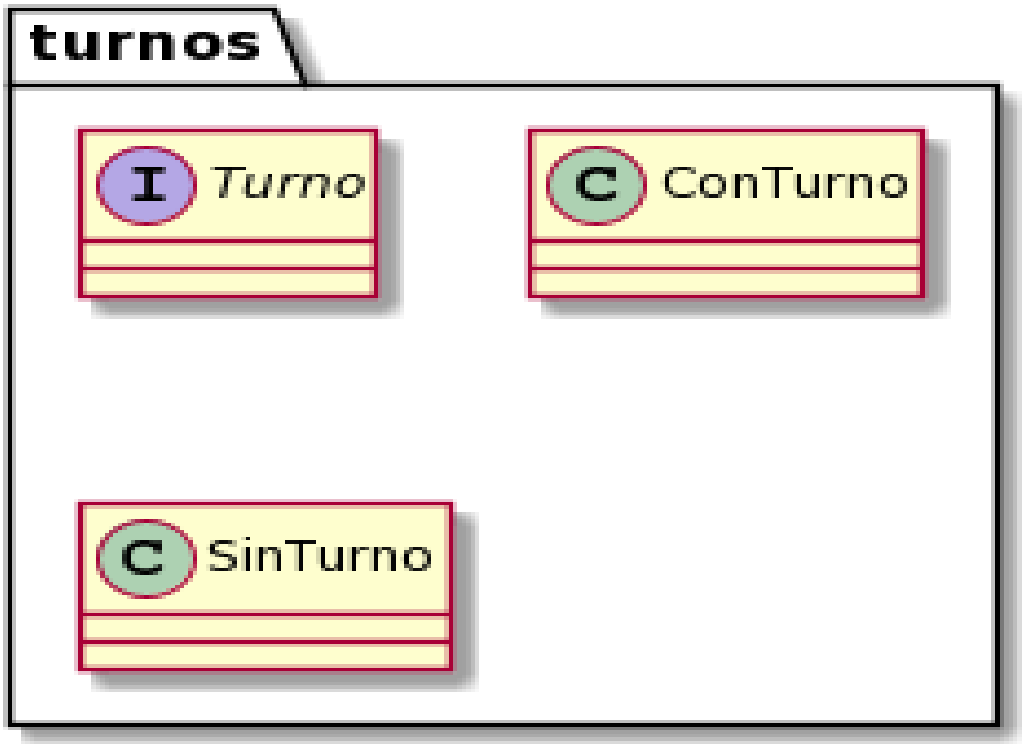

Figura 32: Paquete turnos

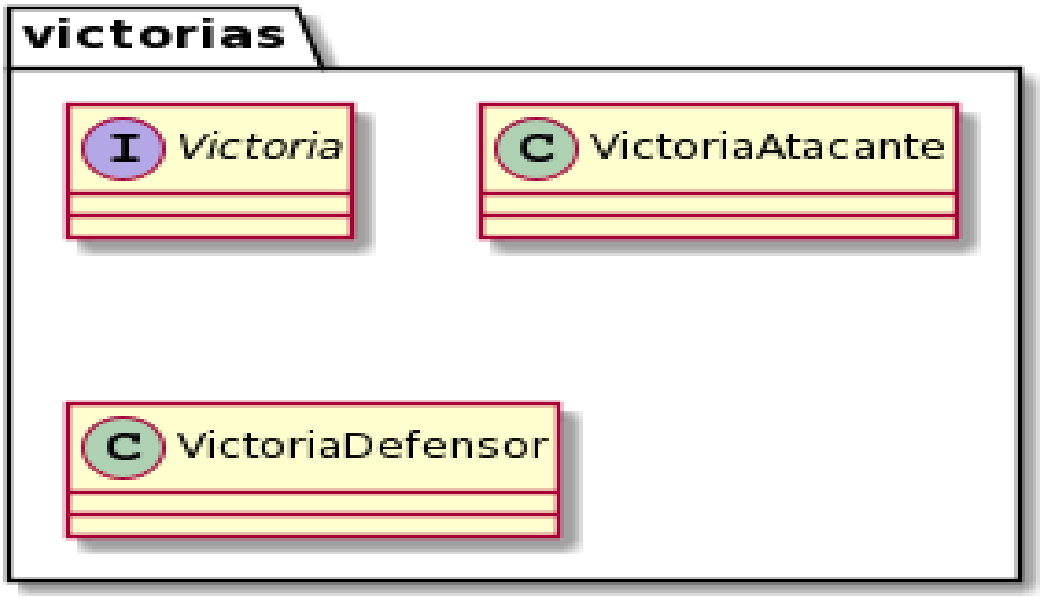

Figura 33: Paquete victorias

6. Diagramas de estado

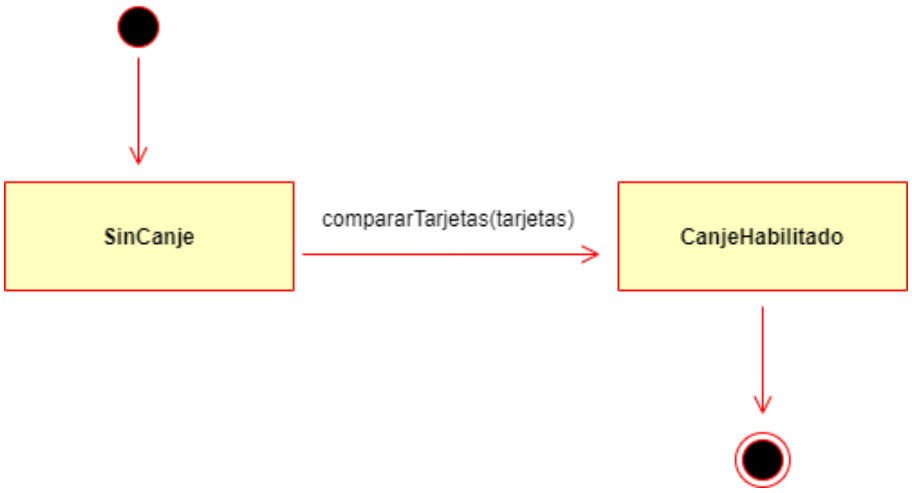

Figura 34: Diagrama de estado Canje.

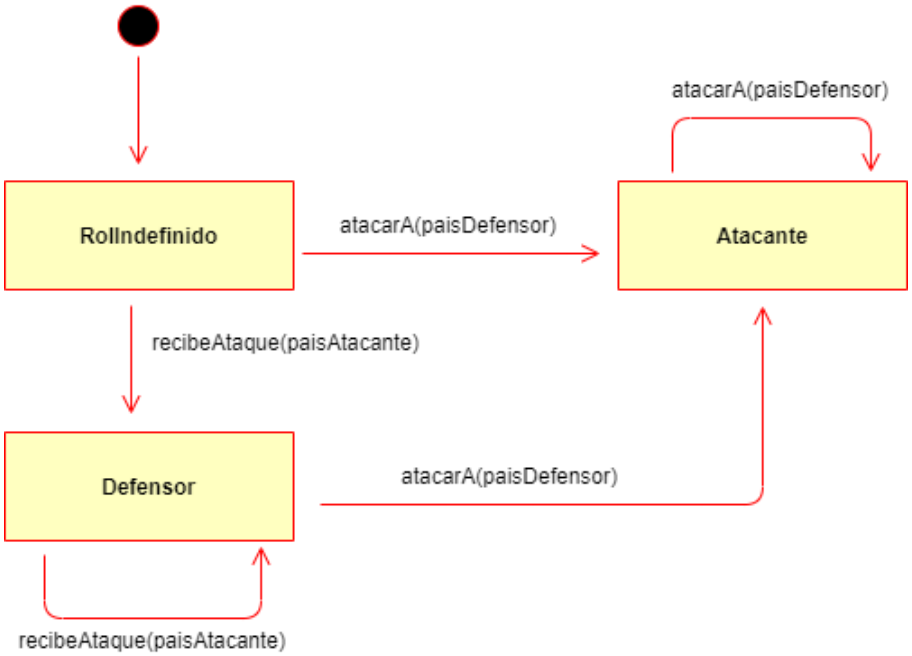

Figura 35: Diagrama de estado Rol de ejercito.

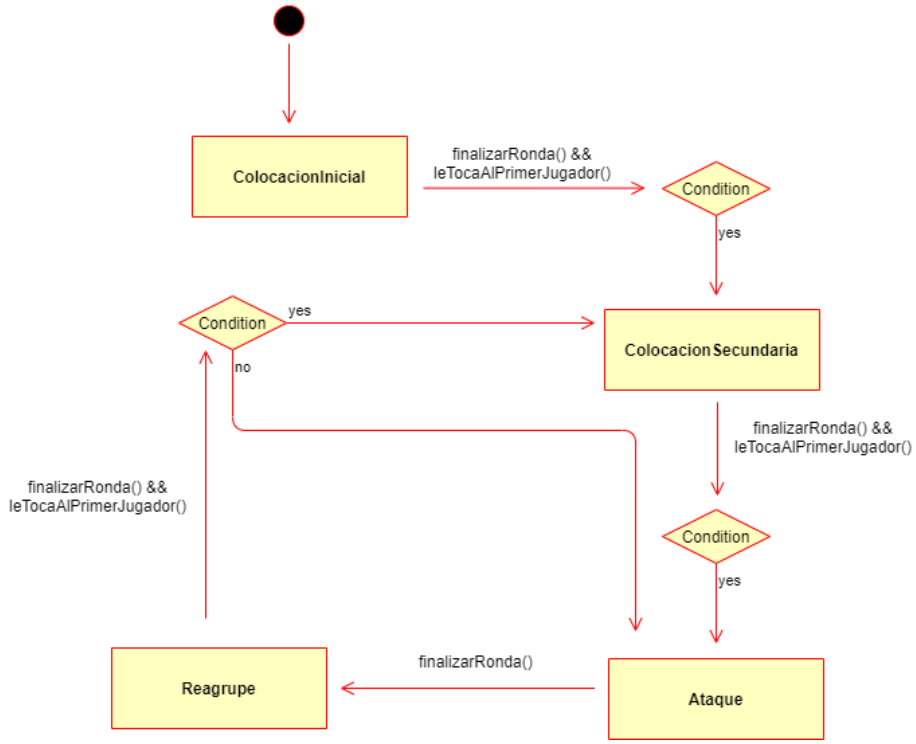

Figura 36: Diagrama de estado Rondas.

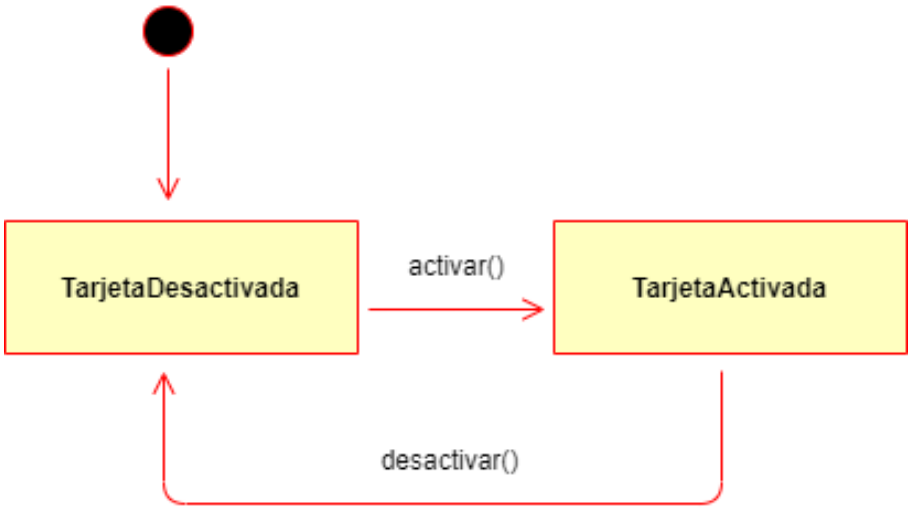

Figura 37: Diagrama de estado activación Tarjetas.

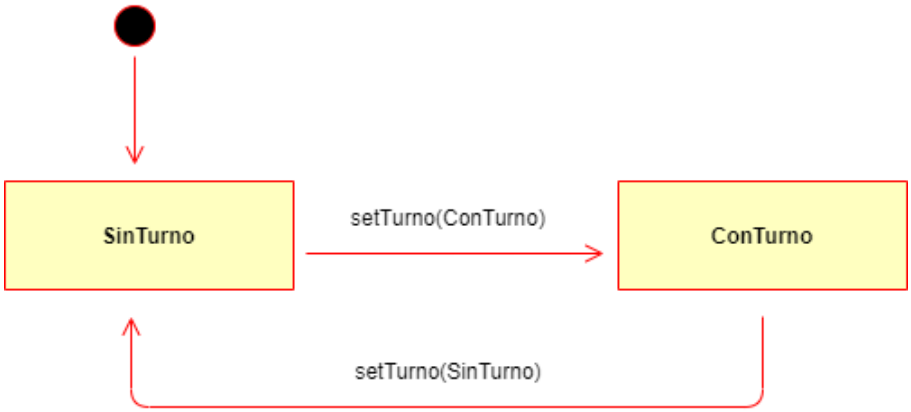

Figura 38: Diagrama de estado Turno.

7. Detalles de implementación

7.1. Patrones de diseño

7.1.1. Strategy

Este patrón de comportamiento lo utilizamos para separar la responsabilidad de la Victoria o Derrota como resultado de un ataque.

La clase ProcesadorResultado recibe como referencia a un objeto que cumple con la interfaz Resultado y que encapsula el comportamiento de procesar, minimizando a su vez la relación de dependencia y permitiendo que sean intercambiables.

7.1.2. Null Object

En varias ocasiones fue necesaria la creación de objetos ficticios que simulen ser como los reales, es decir, que sepan responder a todos los mismos mensajes.

- EjercitoNulo
- CanjeNulo
- CanjeIndefinido
- ResultadoBatallaNulo
- SinTurno

7.1.3. State

La manera que encontramos para mutar el comportamiento de un objeto en tiempo de ejecución, fue utilizando el patrónpatrón State. Definimos un grupo de clases que cumplen con la misma interfaz, para poder aplicar polimorfismo, buscando a su vez, no romper con el principio Tell don't ask. Permite también prepararnos a posibles cambios de requerimiento, puesto que garantiza el principio Open / Closed.

- Dados - Dado
- Pais - Fichas
- Canje - TipoCanje
- Jugador - Objetivo

- Jugador - Turno
- Ejercito - Rol
- ConTurno - Ronda
- Tarjeta - EstadoTarjeta
- Tarjeta - Simbolo
- ResultadoBatalla - Victoria

7.1.4. Singleton

Este patrón de diseño, fue utilizado a fines de garantizar el creado de una única instancia y facilitando a su vez, el punto de acceso a las mismas.

Abajo, ejemplos de las clases que lo implementan:

- ReproductorDeSonido
- ArbitroDeDado
- ProcesadorResultado

7.1.5. Multiton

Utilizamos este patrón para los continentes, los países y las tarjetas como una mezcla del patrón Singleton y un HashMap. De esta forma nos aseguramos de que exista una sola instancia de cada objeto y también de que podamos acceder a ella en cualquier lugar del proyecto mediante llamados a métodos estáticos.

Ejemplos: MultitonContinentes, MultitonPaises, MultitonTarjetas.

7.1.6. Iterator

Este patrón de diseño, nos permitió encapsular la iteración sobre la colección de jugadores al momento de manejar los turnos. A su vez, pudimos desligarnos del manejo de índices de la colección de jugadores tratándola como un arreglo circular.

7.1.7. Observer

Utilizamos este patrón de diseño de comportamiento, para definir un mecanismo de subscripción por parte de las vistas a distintas entidades del modelo, tales como:

Subjects:

- Fichas
- Pais
- Turno

Observers:

- MenuLateralDerecho
- VentanaTarjetas
- VistaObjetivo
- VistaPais

7.2. Helper classes

Utilizamos dos clases utilitarias para repartir responsabilidades y asegurar el cumplimiento del principio de responsabilidad única (S de los SOLID) de las clases de nuestra aplicación. Estas son:

CargarJuego: se desliga a Juego de la responsabilidad de cargar los archivos json necesarios para poder jugar.

RepartidorDePaises: para desligar a Juego de la responsabilidad de tener que repartir los países a los jugadores.

8. Excepciones

ActivacionTarjetaEnRondaEquivocadaException: Se intenta activar una tarjeta en ronda de ataque. Es lanzada desde *Ataque#activarTarjeta* y se la atrapa en *ActivarTarjetasEventHandler#activarTarjetas*. Se muestra un mensaje de error en la GUI.

ArchivoDeContinentesYPaisesNoEncontradoException: Se produce un error en el procesamiento del archivo json de continentes y países. Es lanzada desde *CargarJuego#cargarContinentesYPaises* y se la atrapa en el constructor de CreacionJugadores cuando se instancia el juego. Se imprime por consola el 'stack trace' y se sale de la plataforma (no es posible jugar sin los continentes y países cargados).

ArchivoDePaisesLimitrofesNoEncontradoException: Se produce un error en el procesamiento del archivo json de países limítrofes. Es lanzada desde *CargarJuego#cargarPaisesLimitrofes* y se la atrapa en el constructor de CreacionJugadores cuando se instancia el juego. Se imprime por consola el 'stack trace' y se sale de la plataforma (no es posible jugar sin los países limítrofes cargados).

ArchivoDeTarjetasNoEncontradoException: Se produce un error en el procesamiento del archivo json de las tarjetas. Es lanzada desde *CargarJuego#cargarTarjetas* y se la atrapa en el constructor de CreacionJugadores cuando se instancia el juego. Se imprime por consola el 'stack trace' y se sale de la plataforma (no es posible jugar sin las tarjetas cargadas).

CantidadDeEjercitosInValidaException: Se quiere atacar o reagrupar con una cantidad de ejércitos mayor a la máxima posible o menor que uno. Es lanzada desde *Pais#atacarA* y *Pais#reagrupar*. Se la atrapa en *AtaqueEventHandler#handle* y se muestra un mensaje de error en la GUI.

CantidadDeJugadoresInsuficienteException: Se inicia la partida con menos de dos jugadores. Es lanzada desde *Juego#comenzar* y es atrapada en *FormularioJugadoresEventHandler#handle*. En este caso la excepción es ignorada ya que la GUI no permite que el usuario comience el juego con una cantidad de jugadores inválida.

ContinenteInvalidoException: Se pide por un continente que no existe o no ha sido cargado aún. Es lanzada desde *MultitonContinentes#obtenerInstanciaDe* y es atrapada en *Jugador#cumpleObjetivo*. Se imprime por consola el 'stack trace', aunque cabe aclarar que con el modelo actual de la aplicación es imposible que se lance esta excepción ya que los nombres

de continentes que recibe el método que la lanza están hardcodeados (en los objetivos de ocupación) y son todos válidos.

ElJugadorNoTieneTurnoException: Un jugador realiza alguna acción cuando no es su turno. Es lanzada desde *SinTurno#atacarA*, *SinTurno#reagrupar*, *SinTurno#colocarEjercitos*, *SinTurno#activarTarjeta* y *ConTurno#finalizarRonda*. Se la atrapa en *ActivarTarjetasEventHandler#activarTarjetas* y se la ignora ya que la GUI no permite que juegue un jugador sin turno.

ElPaisNoEsLimitrofeException: Se intenta atacar o reagrupar entre dos países que no son limítrofes. Es lanzada desde *Pais#atacarA* y *Pais#reagrupar*. Se la atrapa en *AtaqueEventHandler#handle* y se la ignora ya que la GUI no permite que se intente realizar ataques o reagrupes entre países no limítrofes.

JugadorNoExisteException: Se quiere obtener un jugador de la lista de Jugadores mediante un índice inválido. Es lanzada en *Jugadores#obtenerJugador* y se la atrapa en *RepartidorDePaises#repartirPaises*. Es ignorada ya que está asegurado que siempre se va a consultar por un jugador con un índice válido.

JugadorNoPoseePaisDeLaTarjetaException: Se intenta activar una tarjeta de la cual el jugador no posee el pais. Es lanzada en *Jugador#activarTarjeta* y es atrapada en *ActivarTarjetasEventHandler#activarTarjetas*. Se muestra un mensaje de error en la GUI.

JugadorNoTieneTodasLasTarjetasException: Se quiere hacer un canje sin que el jugador tenga todas las tarjetas. Es lanzada desde *Jugador#canjearTarjetas* y es atrapada en *CanjearTarjetasEventHandler#canjear*. Es ignorada ya que la GUI no permite que se intente un canje con tarjetas que no sean del jugador.

LaTarjetaYaEstaDesactivadaException: Se desactiva una tarjeta que ya fue desactivada previamente. Es lanzada en *TarjetaDesactivada#desactivar* y se la atrapa en *Tarjeta#desactivar*. Se la ignora ya que el estado de la tarjeta ya era el de *TarjetaDesactivada*.

LaTarjetaYaFueActivadaException: Se intenta activar una tarjeta que ya fue activada previamente. Es lanzada en *TarjetaActivada#activar* y es atrapada en *ActivarTarjetasEventHandler#activarTarjetas*. Se muestra un mensaje de error en la GUI.

NoEsRondaDeAtaqueException: Se intenta realizar un ataque en una ronda que no corresponde. Es lanzada desde *Reagrupe#atacarA* y *RondaColocacion#atacarA*.

NoEsRondaDeColocacionException: Se intenta colocar ejércitos en una ronda que no corresponde. Es lanzada desde *Ataque#colocarEjercitos* y *Reagrupe#colocarEjercitos*. Se la atrapa en *PaisEventHandler#handle* y es ignorada ya que la GUI no permite que se intenten colocar ejércitos en una ronda que no sea de colocación.

NoEsRondaDeReagrupeException: Se intenta realizar un reagrupe en una ronda que no corresponde. Es lanzada desde *Ataque#reagrupar* y *RondaColocacion#reagrupar*.

NoQuedanMasEjercitosPorColocarException: Se intenta seguir colocando ejércitos una vez superado el límite por ronda. Es lanzada en *RondaColocacion#colocarEjercitos*. Se la atrapa en *RepartidorDePaises#repartirPaises* y se la ignora ya que es la colocación inicial de una ficha por país que nunca lanzará esta excepción. También es atrapada en *PaisEventHandler#handle* y se muestra un mensaje de error en la GUI.

PaisOcupadoPorOtroJugadorException: Un jugador intenta colocar ejércitos en un país que no le pertenece. Se lanza en *Jugador#colocarEjercitos* y es atrapada en *RepartidorDePaises#repartirPaises*. Es ignorada ya que solo se llama al método para países vacíos cuando se están repartiendo.

SinCanjeHabilitadoException: Se intenta hacer un canje con tres tarjetas no válido. Se lanza en *SinCanje#canjear* y se atrapa en *CanjearTarjetasEventHandler#canjear*. Se muestra un mensaje de error en la GUI.

TarjetaNoEncontradaException: Se intenta activar una tarjeta que el jugador no posee. Es lanzada en *Jugador#activarTarjeta* y se la atrapa en *ActivarTarjetasEventHandler#activarTarjetas*. Se la ignora ya que la GUI no permite que un jugador intente activar una tarjeta que no sea suya.