Trabajo Práctico N° 2 — Java

T.E.G

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

Integrantes	Padron	Email
Rodrigo Vargas Chávez	97076	rvargas@fi.uba.ar
Lucia Lourengo Caridade	104880	llourengo@fi.uba.ar
Daniela Pareja	100648	dpareja@fi.uba.ar
Julian Galvan	104635	jhgalvan@fi.uba.ar

Índice

1. Supuestos

En el siguiente trabajo práctico, los supuestos que tuvimos en cuenta son:

- Cuando un jugador ataca a otro país y gana dicho ataque, una ficha del país ganador se borra automáticamente y se introduce al país perdedor.

- Cuando la cantidad de jugadores son tres, cuatro o seis y la totalidad de países (50) se reparten entre ellos, al no poder ser de manera equitativa, algunos jugadores van a tener un país más que otros.

- El jugador puede pasar el turno o colocar una cantidad de ejércitos menor que la que le corresponde poner. Por ejemplo, si el juego le permite poner 3 ejércitos, el jugador podría poner sólo 1 ejército y pasar de turno.

- Las tarjetas país pueden canjearse estando activadas o no, pero al canjearse (vuelven al mazo) las que estaban activadas se desactivan.

- Las tarjetas país que tienen de símbolo de comodín valen como tarjetas iguales o diferentes según le convenga al dueño de la tarjeta.

- Cada jugador puede colocar tantos ejércitos como países domine, dividido 2, pero si tiene un número impar de países, se redondea hacia abajo. Por ejemplo, si tiene 7 países, le corresponden 3 ejércitos y no 4.

2. Diagramas de clase

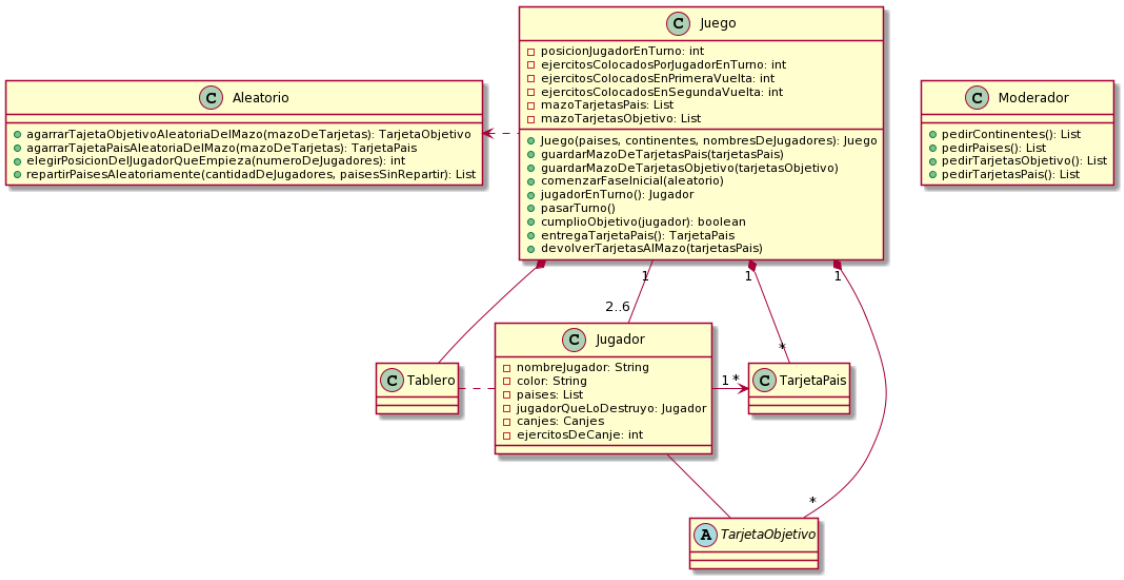

Figura 1:
En este diagrama se muestra a Juego con su estado y sus métodos, y su relación con otras clases. Los métodos de Tablero, Jugador, TarjetaPais y TarjetaObjetivo no se muestran ya que se encuentran en diagramas de clase específicos de las siguientes figuras.

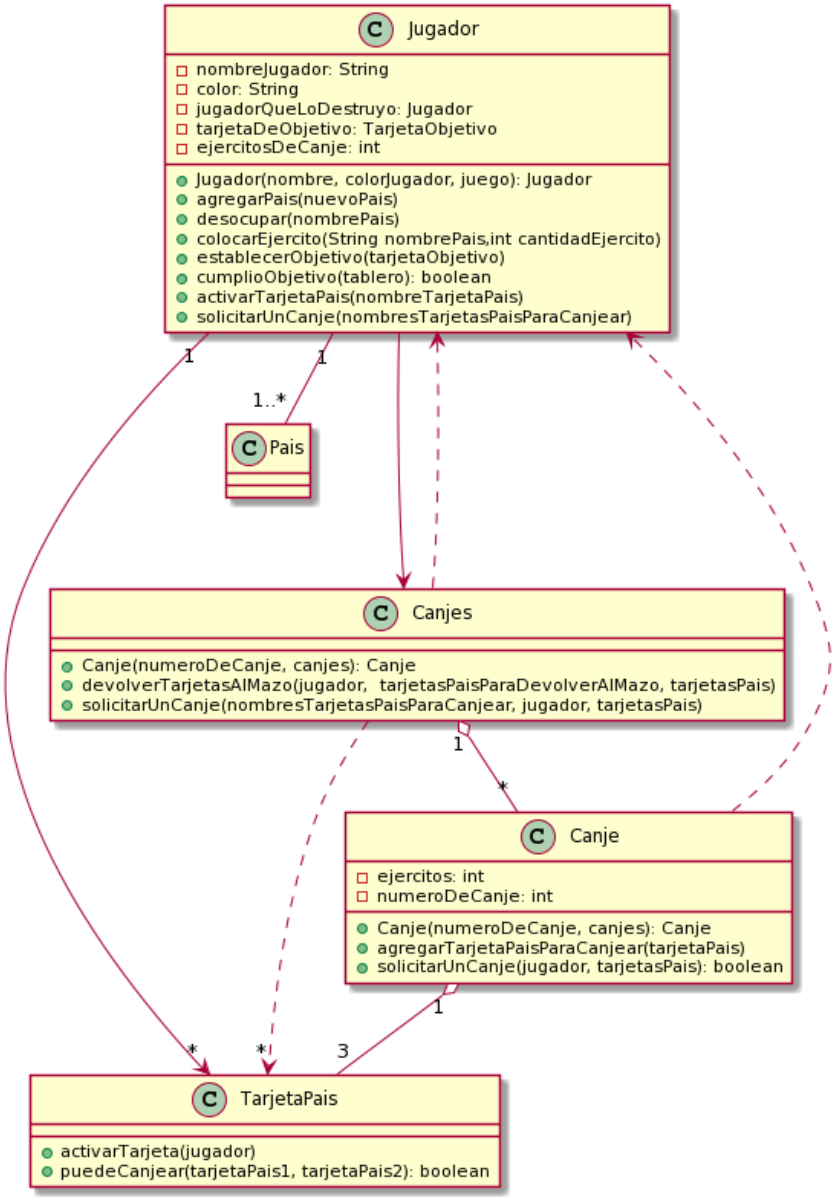

Figura 2:
En este diagrama toma relevancia la clase Jugador y se describen las relaciones con otras clases más específicas relacionadas al mismo. El estado y métodos de la clase Pais se detallan en la figura siguiente (figura 3).

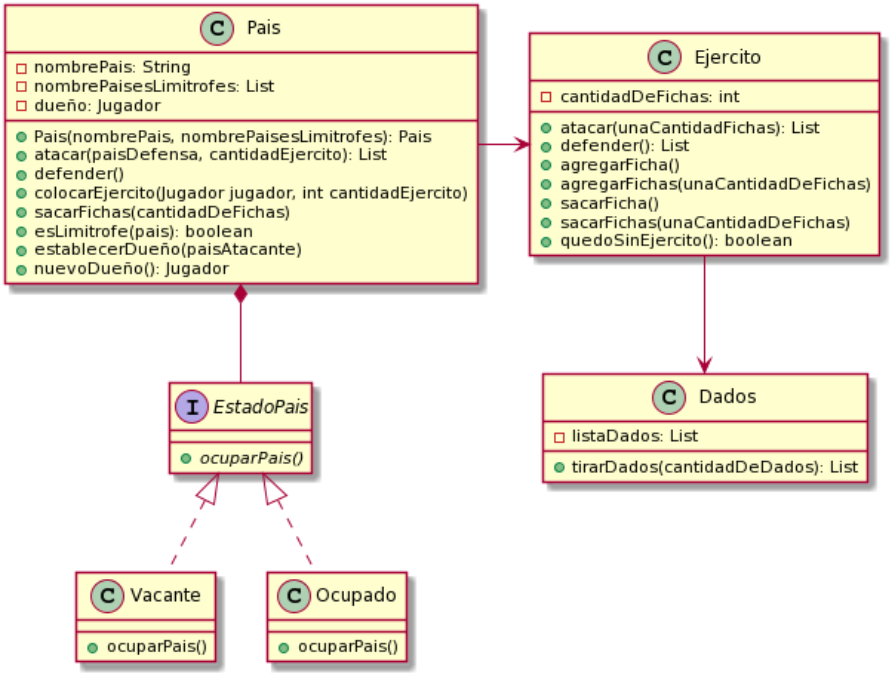

Figura 3:
En este diagrama se representa a la clase Pais con su estado y sus métodos y se describen las relaciones con otras clases.

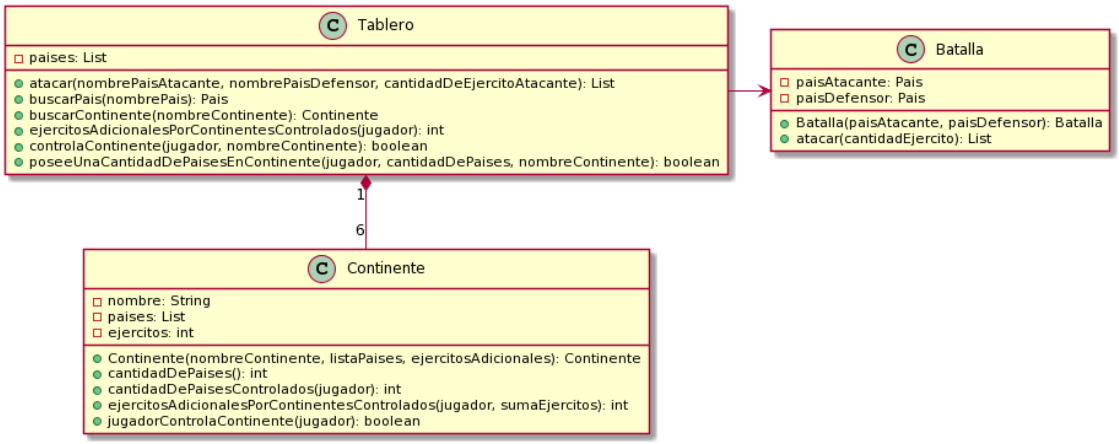

Figura 4:
En este diagrama toma relevancia la clase Tablero y se describen las relaciones con las clases Continente y Batalla.

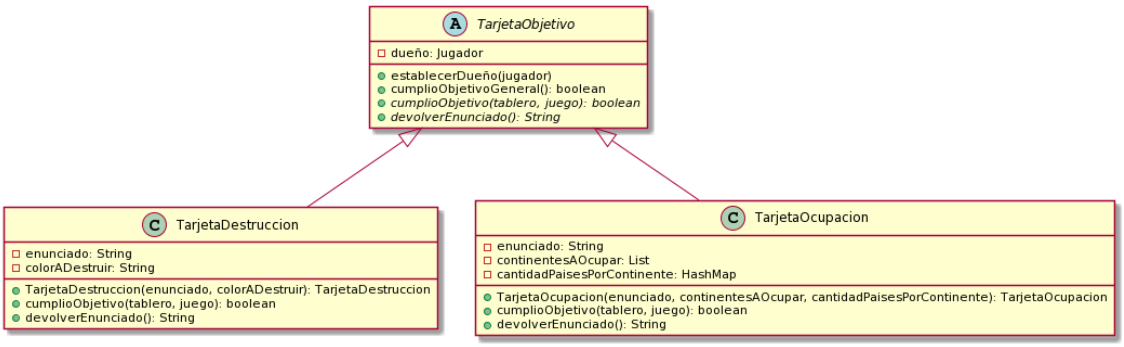

Figura 5:
En este diagrama se muestra la representación de la herencia que se describe en los detalles de implementación. Se observa a TarjetaObjetivo (clase madre) con sus clases hijas.

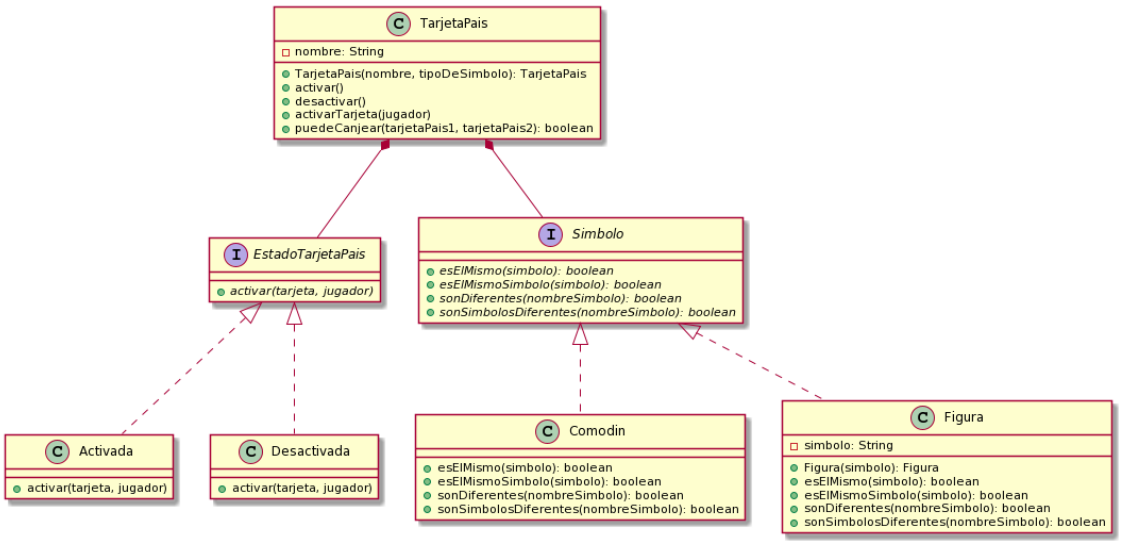

Figura 6:
En este diagrama se muestra en detalle a la clase TarjetaPais y su estructura.

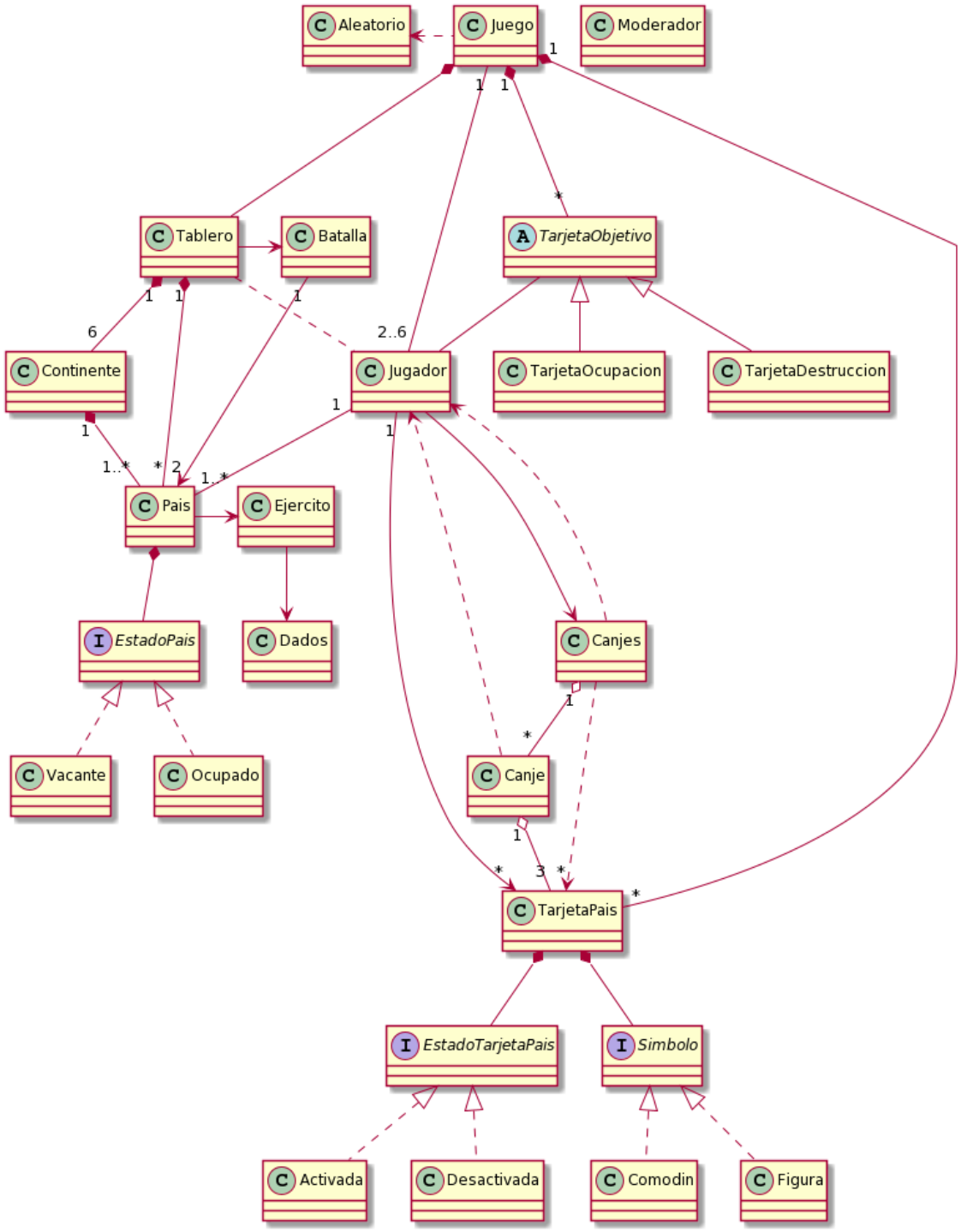

Figura 7:
Este último diagrama de clases corresponde a un esqueleto del modelo de dominio, para observar las relaciones entre clases en forma muy general.

3. Diagramas de secuencia

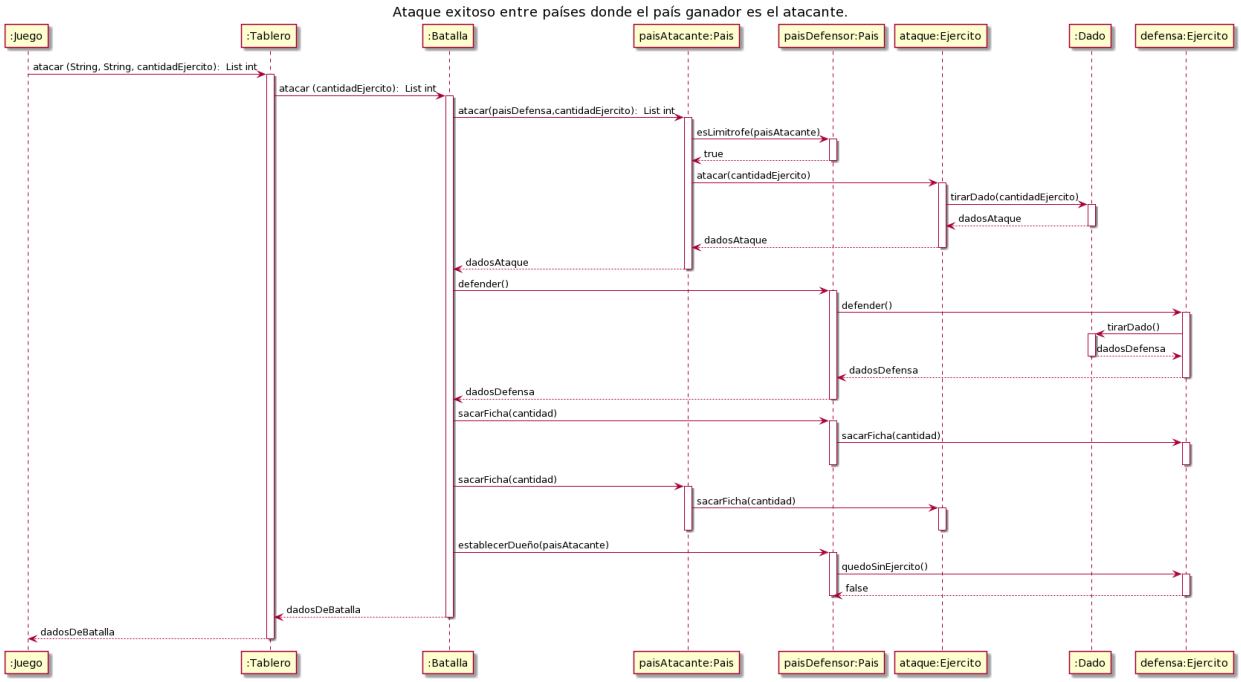

Figura 8:

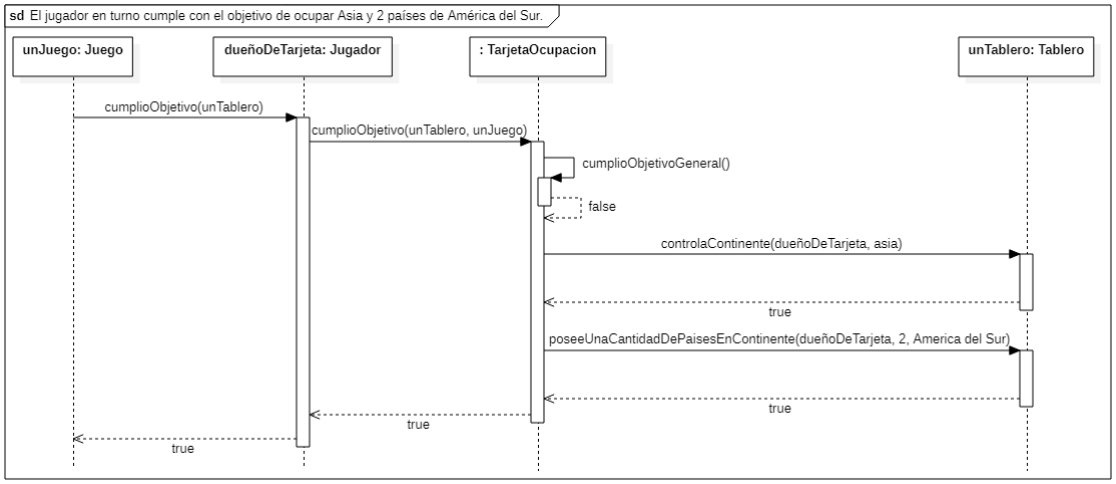

Figura 9:

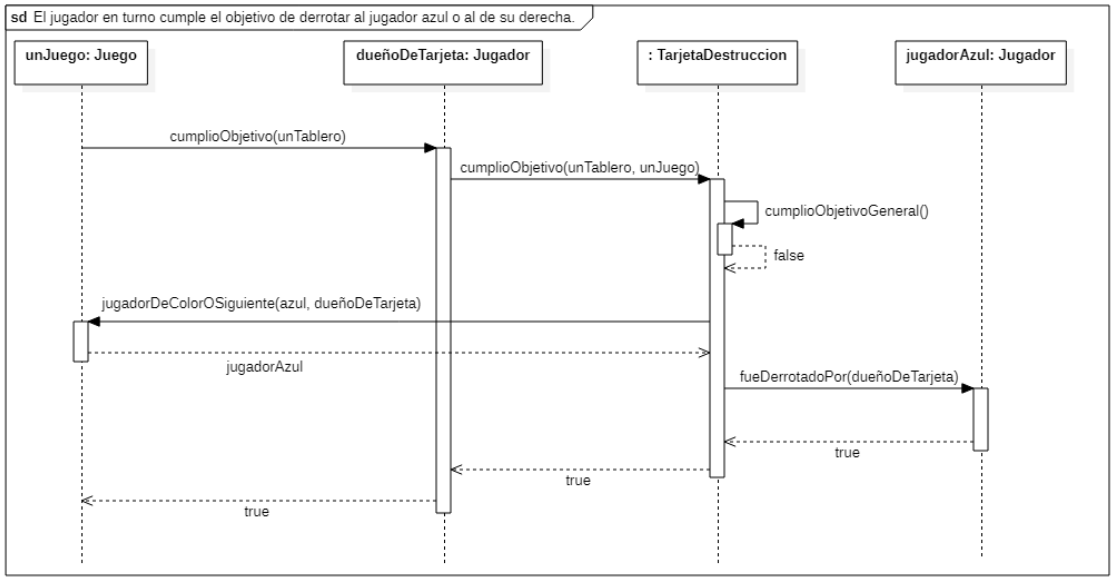

Figura 10:

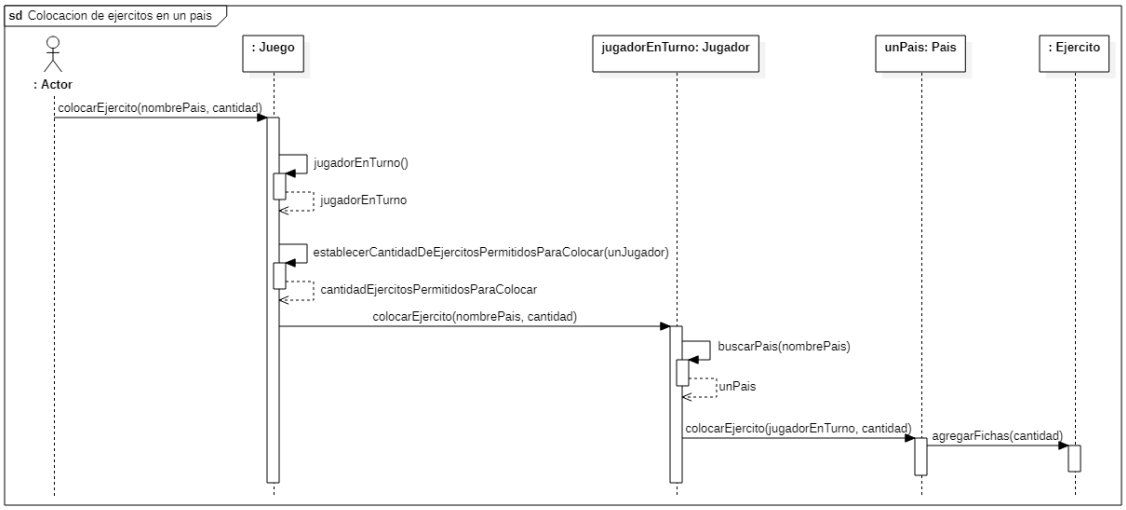

Figura 11:

4. Diagramas de paquetes

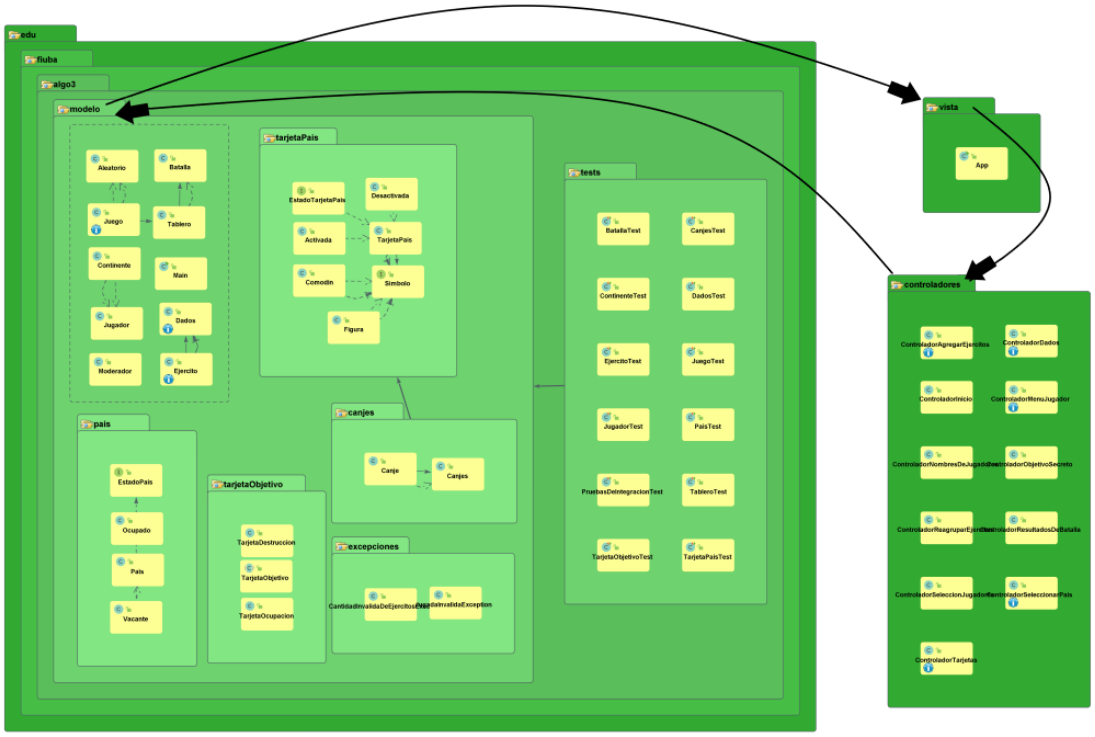

Figura 12:

5. Diagramas de estado

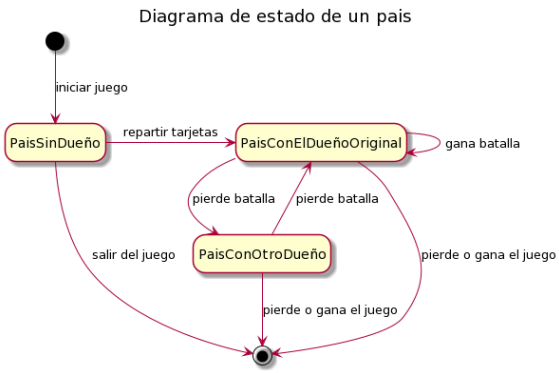

Figura 13:
Diagrama de estado de un pais.

Si bien dentro del modelo, Pais tiene estados de ocupado o desocupado, este diagrama quiere comunicar algo mas complejo. Una vez creado un país y asignándole un dueño; la única forma en que un jugador pierde el pais es perdiendo batallas, y que sólo puede recuperarlo si lo vuelve a ganar a través de batallas.

6. Detalles de implementación

A continuación se mencionan los puntos más conflictivos del trabajo práctico y las estrategias utilizadas para resolverlos:

6.1. Objetivo General y Objetivos Secretos

Cuando se inicia el juego, cada jugador recibe una tarjeta con un objetivo secreto (y también puede cumplir el objetivo general de dominar 30 o más países).

En este punto, utilizamos la herencia. Hay una clase abstracta TarjetaObjetivo y dos clases que heredan de ésta llamadas TarjetaDestruccion y TarjetaOcupacion (ver figura 5).

La TarjetaObjetivo tiene dos métodos que se relacionan a si un jugador ganó el juego: el de cumplioObjetivoGeneral() que está implementado en la misma TarjetaObjetivo y que devuelve true o false si logró dominar 30 o más países, y el de cumplioObjetivo(Tablero, Juego) que no está implementado en TarjetaObjetivo sólo está definido.

Este último, se implementa en las clases hijas TarjetaDestruccion y TarjetaOcupacion. Cada una de estas clases hijas lo implentan de manera distinta. TarjetaDestruccion envía una serie de mensajes para saber si un jugador perdió todos sus países y TarjetaOcupacion, en cambio, para saber si un jugador conquistó determinados países o continentes. Cuando se inicia el juego los jugadores reciben un objeto de TarjetaDestruccion o de TarjetaOcupacion, por lo tanto, se puede ganar el juego, de las dos formas posibles que pedía el enunciado, ya que en la herencia las clases hijas reutilizan el código de la clase madre.

6.2. Canjes de Tarjetas Pais

En las rondas de juego, un jugador que logra conseguir tres tarjetas con el mismo símbolo o tres símbolos distintos puede canjearlas para obtener ejércitos extra según una tabla de canjes detallada en el enunciado, en la cual se daban diferentes números de ejércitos para cada vez que se canjeaba.

En este punto, utilizamos la delegación. Un objeto de la clase Jugador guarda un objeto de la clase Canjes (ver figura 2). Cuando Juego le solicita a Jugador un canje, Jugador delega en Canjes (y además le pasa por parámetro a sí mismo y su lista de TarjetasPais acumuladas hasta el momento).

Canjes se encarga de buscar las tarjetas elegidas para el canje en la lista de tarjetas del jugador y hacer las verificaciones de que el canje efectivamente se puede realizar. Luego, crea un objeto Canje y delega en éste a través del método solicitarUnCanje(Jugador, listaDeTarjetasPais).

Canje delega en TarjetaPais si puede canjear pasándole por parámetro las otras dos tarjetas por el método puedeCanjear(TarjetaPais tarjetaPais1, TarjetaPais tarjetaPais2).

La TarjetaPais delega en el objeto que guarda en su estado “simbolo”, envía un mensaje que devuelve true o false de si son el mismo símbolo o si son símbolos diferentes. El objeto que guarda en el estado “simbolo” puede ser de la clase Figura (si es un globo, un barco o un cañón) o de la clase Comodin. Cada uno de estos va a responder de manera diferente (que se llama Polimorfismo). Figura realiza las comparaciones, mientras que Comodin siempre devuelve true (si es el mismo o distinto) (ver figura 6).

Si el resultado final de toda la serie de mensajes es true entonces, Canje desactiva primero todas las tarjetas y delega en Juego “devolverlasAMazo(tarjetasPais)” (pero primero pasando por Canjes y por Jugador). Luego, envía un mensaje a Jugador para darle los ejércitos extra a través del mensaje “entregarEjercitosDeCanje(ejercitos)”.

Por último, Juego, el encargado de hacer la suma final para calcular los ejércitos que puede colocar cada jugador en cada turno, le va a sumar esos ejércitos extra. (En realidad lo que hace es restar esa cantidad a la variable ejercitosColocadosPorJugadorEnTurno, dándole así los ejércitos extra que le corresponden).

6.3. Activación de Tarjetas Pais

Si el jugador posee una tarjeta de país aún no activada de un país que controla puede activarla, lo cual le brinda dos ejércitos adicionales en dicho país. Estas tarjetas pueden activarse una sola vez, a menos que sean devueltas a través de un canje.

Aquí utilizamos también la delegación. En este caso Jugador busca la tarjeta correspondiente en su lista de tarjetas y le delega a ella a través de activarTarjeta(jugador). La tarjeta delega en su "estadoTarjeta" que guarda un objeto que puede ser de la clase Activada o Desactivada a través del método "activar(tarjetaPais, jugador)".

Si es Activada, no hace nada, pero si es Desactivada, entonces envía un mensaje a jugador para que coloque dos ejércitos en el país de la tarjeta. Esta diferencia en la respuesta a un mismo mesaje se llama Polimorfismo.

Luego, envía el mensaje activar() a TarjetaPais, el cual cambia el estado borrando el objeto Desactivada y cambiándolo por Activada. Si la tarjeta seleccionada tiene de nombre un país que no pertenece al jugador, entonces Jugador al no encontrar el país en su lista lanza una excepción de JugadaInvalida (ver item 7.2 del TP). Cuando las tarjetas de países son devueltas a través del canje explicado en el item 6.2 del TP, son desactivadas en TarjetaPais como respuesta a un mensaje de Canje.

6.4. Ejércitos Adicionales por Continente

Si el jugador controla un continente completo coloca ejércitos adicionales según una tabla continentes y número de ejércitos provista en el enunciado.

En nuestro modelo, como el resultado final del número de ejércitos que puede colocar un jugador por turno depende de varios factores, Juego va a delegar parte del cálculo en algunas clases para llegar a este resultado.

Para determinar cuántos ejércitos puede colocar un jugador con respecto a los continentes, Juego delega en Tablero que a su vez delega en Continente el cálculo de ejércitos extra por continente controlado. Juego le pasa por parámetro el jugador en turno a un objeto Tablero a través del método "ejercitosAdicionalesPorContinentesControlados(Jugador)". Y Tablero delega en cada Continente de su lista, la suma de ejércitos adicionales si corresponde o no.

Cada objeto Continente, primero verifica si el jugador lo controla y en caso de que si, suma los ejércitos adicionales que corresponden, y en caso de que no, no suma nada. Y delvuelve ese valor. El siguiente continente de la lista de Tablero repite el mismo procedimiento. De esta forma, los objetos Continente están bien encapsulados.

6.5. Principios de Diseño

Para este trabajo práctico los principios de diseño utilizados corresponden a los principios SOLID:

1) El principio de responsabilidad única. Cada una de las clases de nuestro modelo cumple una función única, sencilla y concreta. Este principio se cumple en todo el modelo pero se pueden mencionar algunos ejemplos específicos. La clase Dados, tiene la función de devolver lista de enteros random de 1 a 6. La clase Batalla se encarga de elegir al ganador en un ataque. La clase Pais se encarga de saber si otro país es su límitrofe o no. La clase Ejercito se encarga de conocer su cantidad de fichas y atacar o defender en base a eso.

2) El principio de abierto / cerrado. Las clases deben estar abiertas para la extensión pero cerradas para la modificación. En nuestro modelo esto se ejemplifica en que el juego ahora es para máximo seis jugadores, pero podría ser fácilmente para más jugadores, solamente agregando un nuevo color a la constante COLORES de la clase Juego. O también, se podría agregar un nuevo tipo de tarjeta para ganar, agregando otra clase hija a la clase madre TarjetaObjetivo. O también, cambiando el archivo de Fronteras se podría agregar más países fácilmente sin tener que hacer ninguna modificación en el modelo.

3) Principio de sustitución de Liskov. En nuestro programa no sustituímos clases hijas por una clase madre, pero podría hacerse, por ejemplo, en el caso de la TarjetaObjetivo (clase madre) y TarjetaDestrucción y TarjetaOcupación (clases hijas). Si se hiciera esto, es decir, si se usara sólo la clase TarjetaObjetivo, se podría ganar el juego conquistando 30 países, sin que el usuario sepa la diferencia o que el programa falle. De esta forma se utilizó la herencia para agregar comportamiento a las clases hijas y se cumple la condición TarjetaOcupacion "es una" TarjetaObjetivo (e igual para la otra).

4) Principio de segregación de la interfaz. Cada clase va a implementar la interfaz que va a utilizar. En nuestro modelo, tenemos tres interfaces: Interfaz EstadoPais (implementada por Ocupado y Vacante), Interfaz Simbolo (implementada por Comodin y Figura) e Interfaz EstadoTarjetaPais (implementada por Activada y Desocupada). Cada una de estas clases utiliza los métodos de su interfaz.

5) Principio de inversión de dependencias. Se depende de las abstracciones y no de las implementaciones. Por ejemplo, cuando se ataca en el juego, se envía mensaje atacar a través de una serie de clases y delegando sucesivamente hasta que Ejercito tira unos dados. Se podría cambiar Dados por otra clase y que Ejercito ataque de otra forma. En este caso, atacar no depende de la implementación.

6.6. Patrón de Diseño: MVC

Con respecto al patrón de diseño, se utilizó Modelo-Vista-Controlador que divide el programa en tres grupos según las tareas que realiza.

Por un lado el Modelo, presenta las clases y objetos del dominio. La Vista corresponde a la interfaz, todo lo que se le muestra al usuario. Es una representación del estado del Modelo en un momento determinado. Y el Controlador actúa como intermediario entre el usuario y el modelo. Capta los cambios en la vista y los traslada al modelo y viceversa. Elegimos este patrón de diseño para nuestro TP porque separar según estas funciones es una forma muy ordenada y conveniente para realizar un software completo.

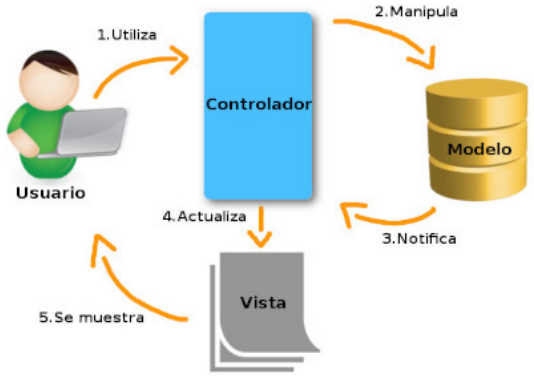

Figura 14:
Esquema del patrón de diseño MVC utilzado en nuestro juego TEG.

7. Excepciones

7.1. CantidadInvalidaDeEjercitosException

Esta excepción se puede lanzar en dos clases:

1) En Ejercito, en el caso de que se quiera atacar con una cantidad de fichas menor a uno.

2) En Juego, si los ejércitos que se quieren agregar superan el máximo permitido en la primera vuelta (cinco) y en la segunda vuelta (tres) de la fase inicial. Y en la fase de juego, si se quiere colocar mayor cantidad de ejércitos a los disponibles por turno para un jugador determinado.

En el primer caso, no es necesario atrapar la excepción porque la vista tiene un botón con un dibujo de un dado que cuando el usuario hace click empieza a contar a partir de 1.

En el segundo caso, si, se atrapa el error en el controlador AgregarEjercitos, la ventana muestra un mensaje para el usuario que dice que "No hay ejércitos suficientes". Y el juego continua.

7.2. JugadaInvalidaException

Esta excepción se puede lanzar en tres clases:

1) En Pais. Se lanza en el caso de que el país que se quiere atacar o reagrupar ejércitos no es limítrofe.

2) En Tablero. Se lanza en el caso de que no encuentra algunos de los países (atacante o defensor) en su listaPaises.

3) En Jugador. Se lanza en el caso de que se desea colocar ejércitos en algún país que no está en su listaPaises (es decir, que no le pertenece).

En el primer caso no es necesario atraparla porque en la vista, el usuario, la única opción que tiene es la de seleccionar el país de una lista, y luego elige a través de botones Atacar o Reagrupar. De esta forma se filtran los países, dejando sólo los que son válidos en una nueva ventana para dichas acciones. Además, no son los mismos los limítrofes para atacar (limítrofes de jugadores contrarios) que de reagrupar (limítrofes propios). Aquí tuvimos en cuenta la recomendación de que es mejor optar por validar y prevenir errores, a dejar que el usuario ejecute la acción y falle.

En el segundo caso, tampoco es necesario atrapar esta excepción ya que el usuario solo puede seleccionar paises que sí existan en Tablero.

Por último, el único caso donde ésto sí puede ocurrir es cuando el usuario quiere activar una tarjetaPais de un país que no le pertenece entonces a través de la delegación se puede llegar a que se lanze esta excepción. Esta excepción se atrapa abriéndose una nueva ventana que le avisa al usuario que no puede activar la tarjeta porque el país no es de su propiedad. Y continúa el juego.