FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Java

[7507/9502] Algoritmos y Programación III
Curso 2
Primer cuatrimestre de 2021

Alumno:	Padron:	Mail:
Jose Hernandez	106168	jhernanezm@fi.uba.ar
Agustin Gabriel Garcia	105991	aggarcia@fi.uba.ar
Christian Nahuel Rodriguez	104979	crnrodriguez@fi.uba.ar
Patricio Tomás Silva	106422	psilva@fi.uba.ar
Alexis Martin Ramos	98891	amramos@fi.uba.ar

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 el juego de mesa TEG en Java utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

- Los jugadores se asignan en orden, es decir, ninguno puede elegir su color.
- El jugador de la derecha es el del próximo turno.
- El jugador puede decidir si quiere agregar en cada uno de sus paises, al iniciar el juego del TEG, si este quiere agregar 3 o 5 ejércitos.
- No hay un máximo de ejércitos que pueda estar en un país.
- El resultado de dados iguales representan una victoria para el equipo defensor.
- Los ataques pueden realizarse aunque el numero de atacantes sea menor al de los defensores.
- No pueden hacerse pactos entre jugadores.
- Cunando un jugador gana se le pasa una tropa automaticamente al pais conquistado, y luego se le pregunta si se le quieren mandar mas tropas, en el caso de que tenga la cantidad disponibles.
- Solo se puede atacar con un maximo de 3 tropas.
- Los dados se lanzan una unica vez en cada orden de ataque, si el jugador quiere volver a atacar tiene que mandar una nueva orden.
- El jugador perdedor, no pierde una tarjeta país al ser conquistado.
- Existe un total de 10 objetivos distintos definidos en el juego.
- Los símbolos de las tarjetas pueden ser números.
- se termina el juego si algún jugador logro completar su objetivo o dominar 30 países.

3. Modelo de dominio

El juego busca implementar el juego de mesa TEG, donde 2 o varios jugadores tratan de cumplir objetos de conquista mundial para poder vencer en el juego.

Nos permite realizar los ataques únicamente a países limítrofes, osea que estén al rededor de nuestro país o conectado por un puente, dependiendo de la cantidad de ejércitos que tengamos en el país vamos a usar 1 o varios dados, donde podríamos derrotar o perder tropas dependiendo de nuestra suerte.

Mientras mas países, o continentes, vayamos conquistando en este mundo del TEG, mas oportunidades tendremos de conseguir mayor cantidad de tropas, donde podremos reagruparlas a cualquier país bajo nuestro dominio (estas solamente se pueden a países limítrofes), nuestro objetivo final sera lograr completar nuestro objetivo secreto, que sera entregado a cada jugador al iniciar el juego, o lograr dominar 30 países en total.

4. Diagramas de clase

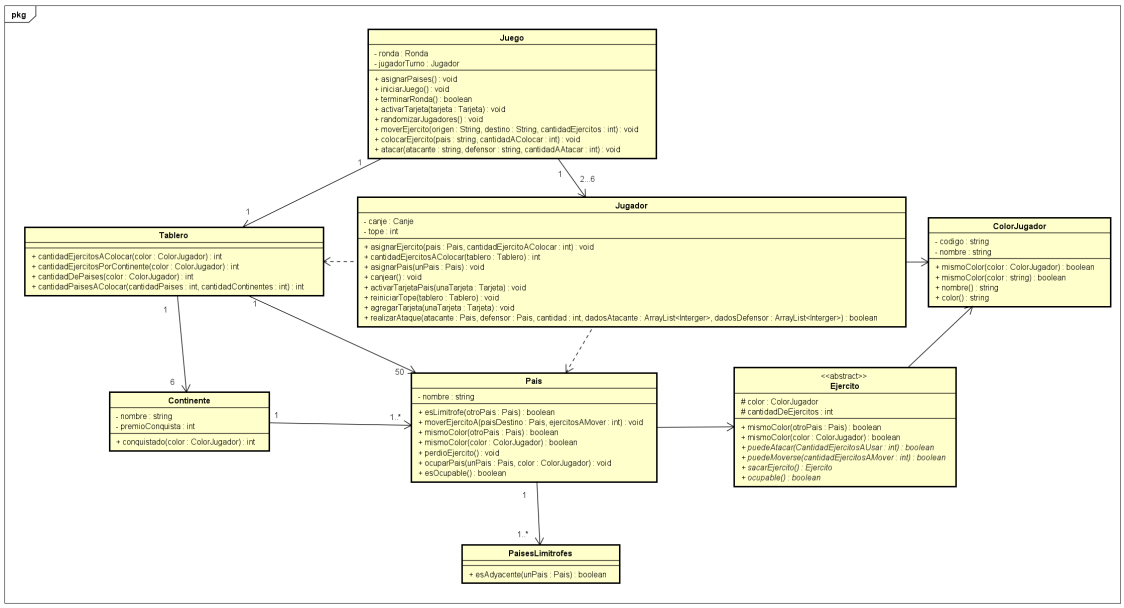

Figura 1: Diagrama de clase del juego en general.

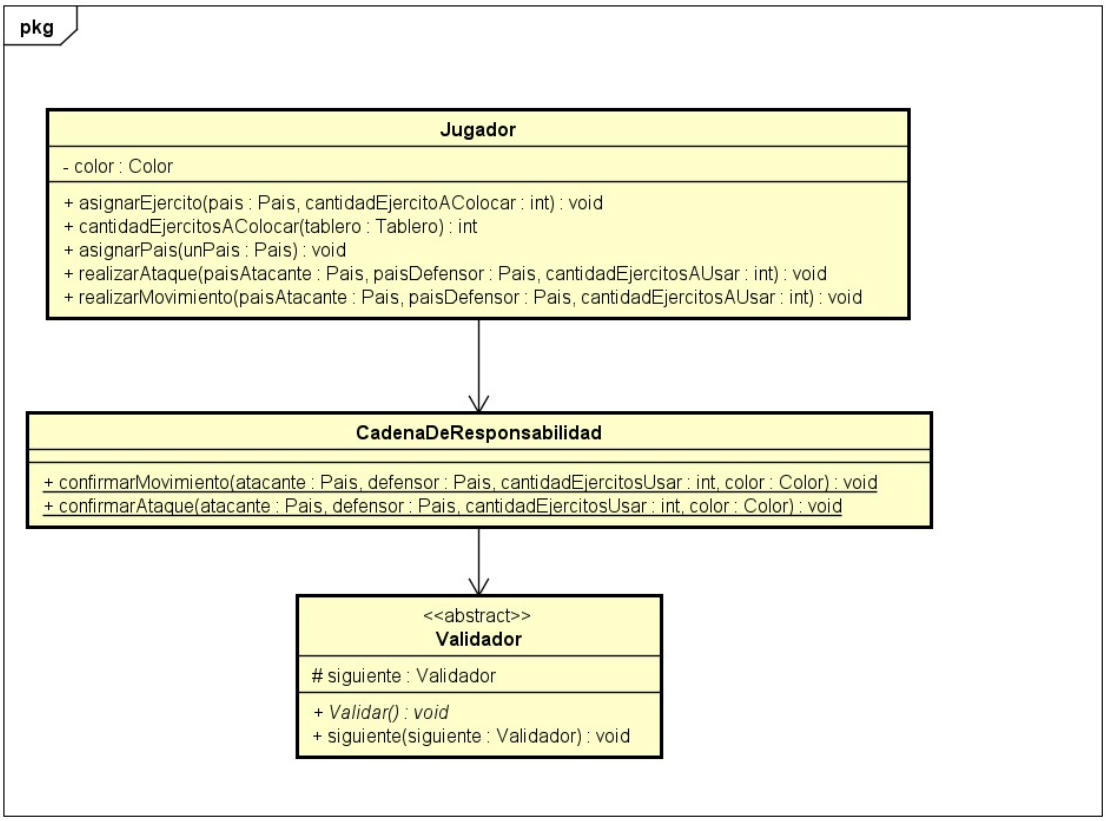

Figura 2: Diagrama de clase de la relación del jugador con la cadena de responsabilidad.

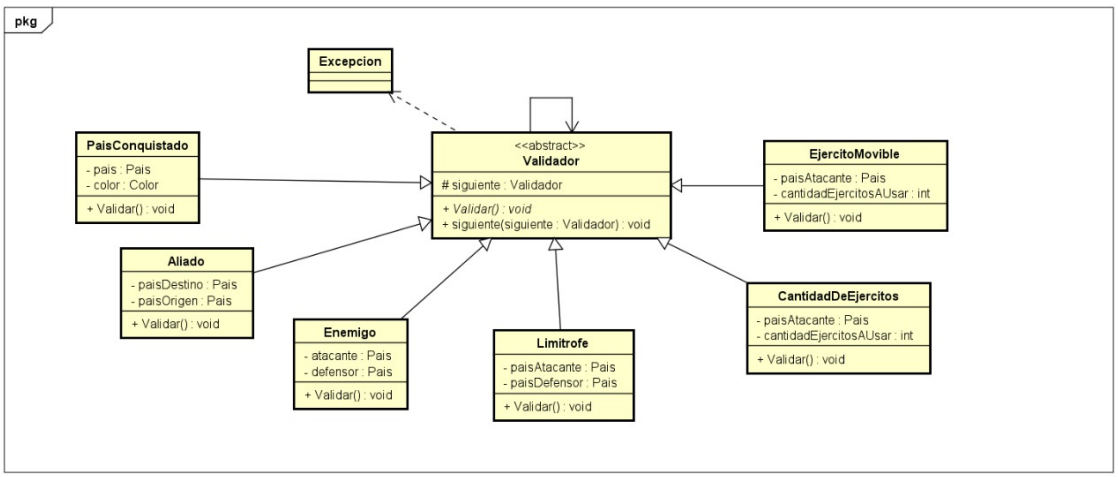

Figura 3: Diagrama de interfaces de la conección de los validadores a la interface.

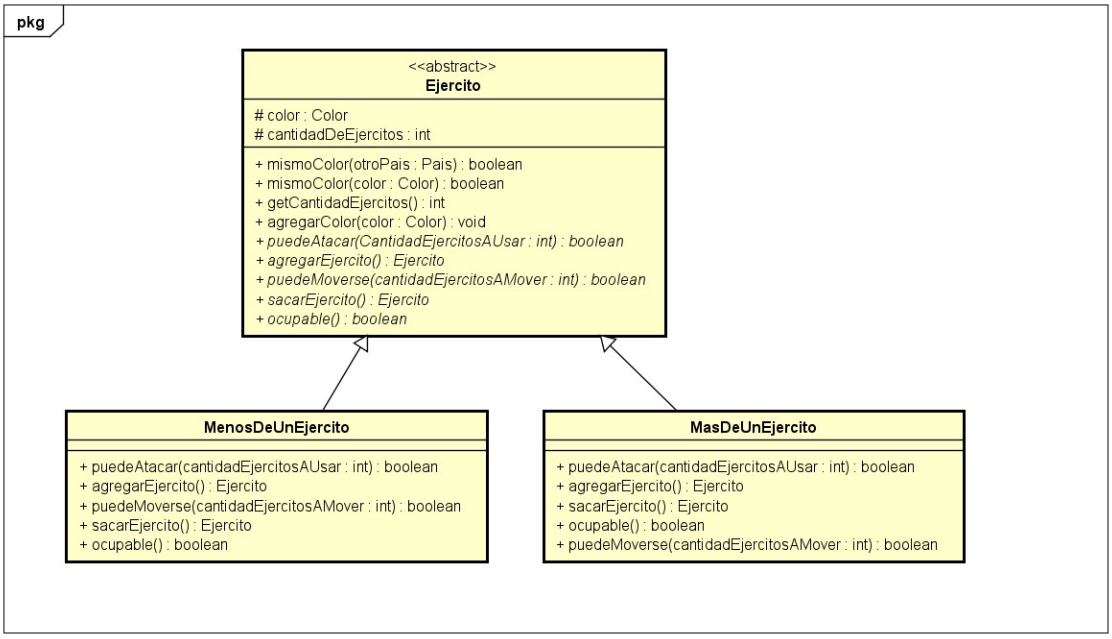

Figura 4: Diagrama de clase de la relación de Ejercito con las clases MasDeUnEjercito y UnoO-MenosEjercitos.

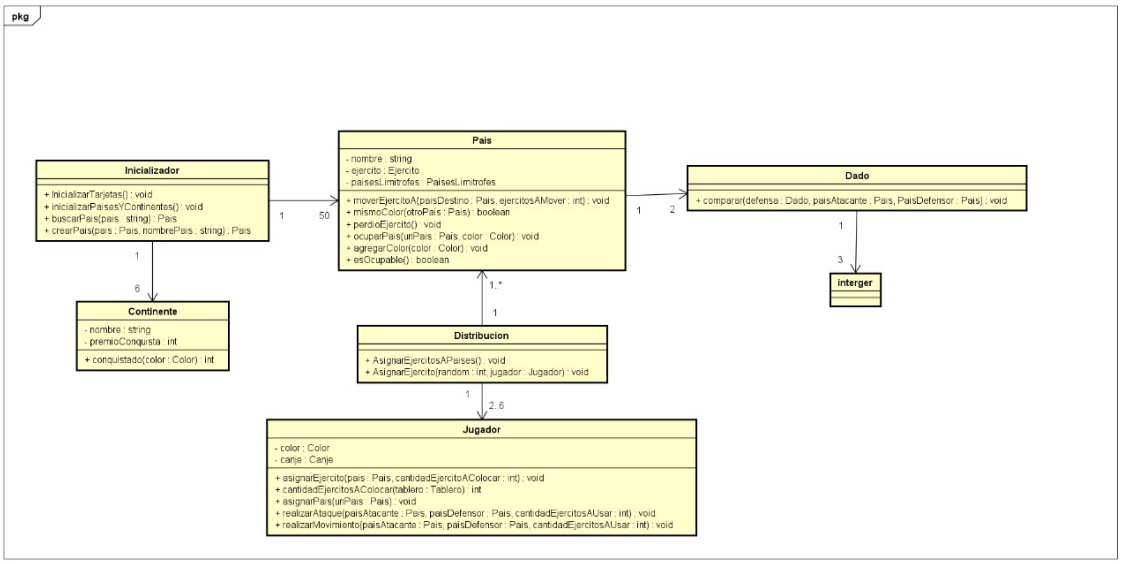

Figura 5: Diagrama de clase de la relación entre País , Dado, Inicializador y Distribución.

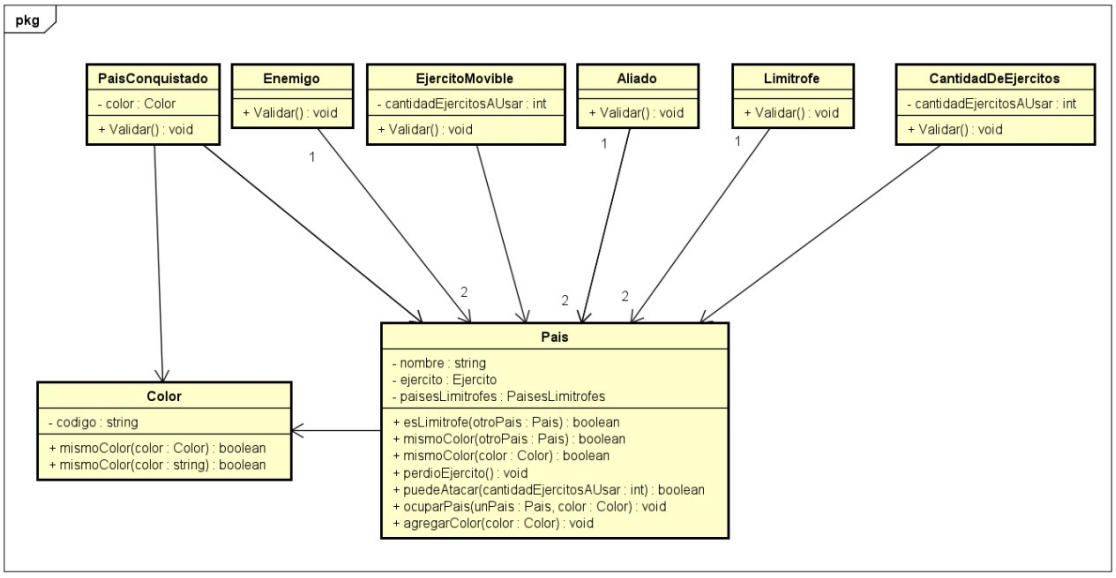

Figura 6: Diagrama de clase de la relación entre País y las clases de validador.

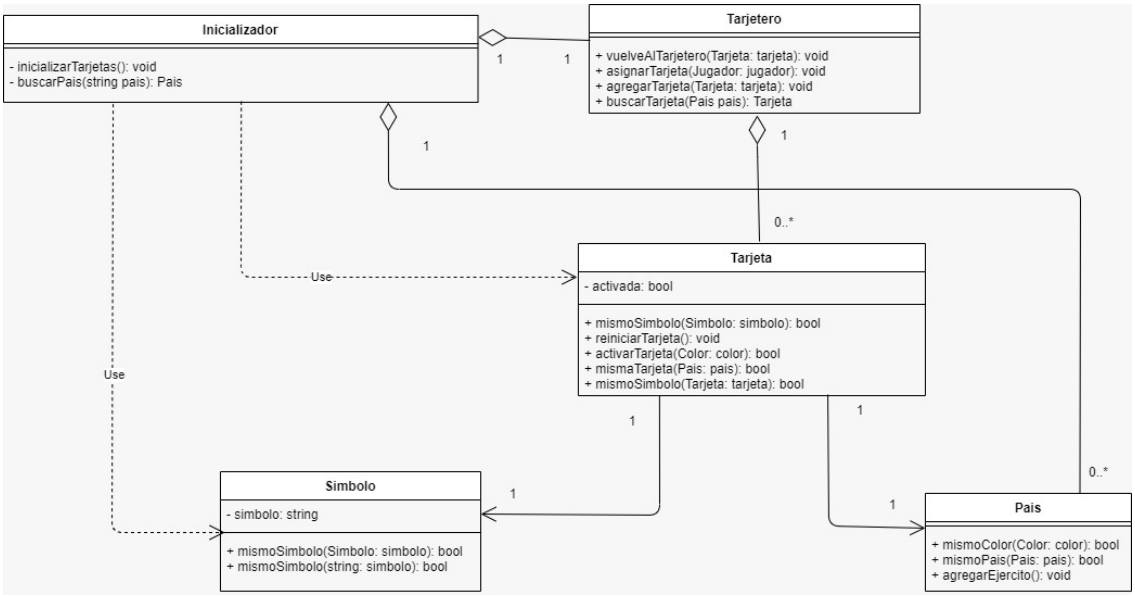

Figura 7: Diagrama de clase de la relación entre el Tarjetero y una Tarjeta, y a su vez la tarjeta con su Símbolo y País.

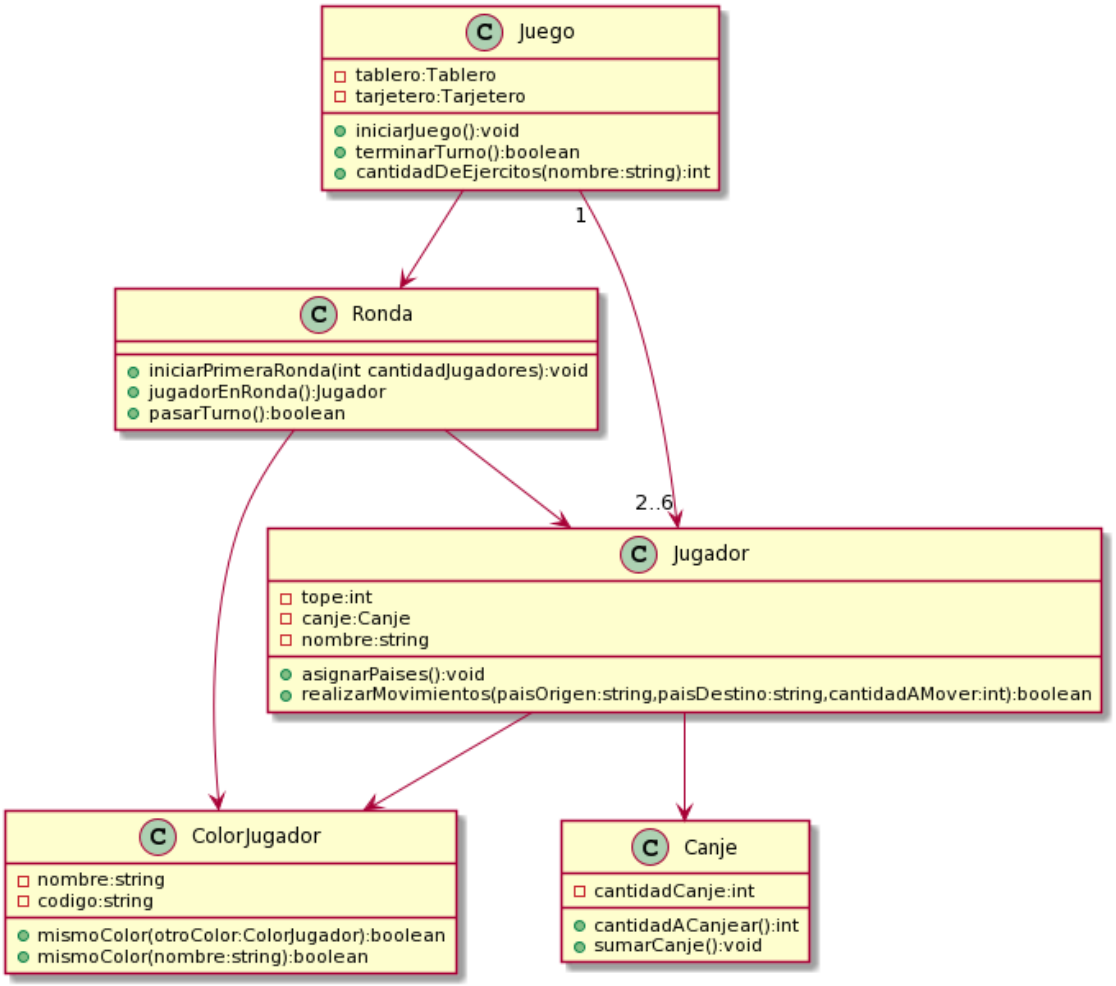

Figura 8: Diagrama de clase de la relación entre el juego y la cantidad de jugadores que participan.

5. Detalles de implementación

5.1. Detealle1

Para la clase Color se realizo un **envío doble** para los métodos mismoColor, donde primero se le llama al método que tiene por parámetro una clase Color, y luego esta llama al otro método del mismo nombre pero con un parámetro String, donde realiza la comparación de ambos colores.

Este método de **envio doble** se implementan en las clase País, y en la clase abstracta Ejercito.

```
public Boolean mismoColor(Color otroColor){
    return otroColor.mismoColor(codigo);
}

public Boolean mismoColor(String otroColor){
    return (codigo.equals(otroColor));
}
```

5.2. Detalle2

Creamos una Clase CadenaDeResponsabilidad, donde usando **Inversion de control** nosotros asignamos cual clase le sigue al antecesor, haciendo esto podemos nosotros especificar que comportamientos queremos que chequee para validar el ataque o movimiento de un país a otro.

5.3. Detalle3

La creacion de los dados para el ataque entre los paises, se realiza primero en el metodo **atacaA** donde se crea el dado del país atacante enviándole por parámetros 2 arrays con hasta 3 valores para los dados, en caso de que los valores de los array estén vacíos, se crearan automáticamente con valores randomisados.

Dentro del método anterior (atacaA), luego de crear el dado, se llama a otro método **teAtaca** donde realiza lo mismo que en el método anterior, pero para el dado defensor.

```
public void atacaA(Pais defensor, int ejercitosAtaque,
        ArrayList<Integer> valoresDadoAtacante,
        ArrayList<Integer> dadoDefensor)
    {
        Dado dadoAtacante = new Dado(ejercitosAtaque,valoresDadoAtacante);
        defensor.teAtaca(this,dadoAtacante,dadoDefensor);
    }

public void teAtaca(Pais atacante,Dado dadoAtacante,
        ArrayList<Integer> valoresDadoDefensor)
    {
        Dado dadoDefensor = new Dado(ejercito.getCantidadDeEjercitos(),
                                     valoresDadoDefensor);
        dadoAtacante.comparar(dadoDefensor,atacante,this);
    }
```

5.4. Detalle4

Al principio realizamos la implementacion de la interfaz con **scene buider** (.fxml), pero por motivos como:

- No se podía realizar Polimorfismo
- Si se producía un error afectaba a todo el código y no solo a una parte.
- No permite pruebas unitarias, porque automatizaba la asignación de métodos.
- Si se querían inicializar todas las fichas en el tablero, teníamos que implementarlas una por una, en cambio con javafx, con un archivo .csv podemos crear al país y sus coordenadas donde se encontrara en el mapa.

5.5. Detalle5

Utilizamos el patrón obsrever para actualizar la información en el tablero de forma automatica

5.6. Detalle6

Usamos alertas para avisarle al jugador/usuario, que cosas no puede realizar o si le falta alguna otra acción para continuar, todo esto sin hacer que se termine el programa.

6. Excepciones

Exepción FileNotFoundException: si hubo un error en la carga de los archivos a leer.

Exepción Exepcion: Exepcion que salta cuando la validacion entre paises falla.

7. Diagramas de secuencia

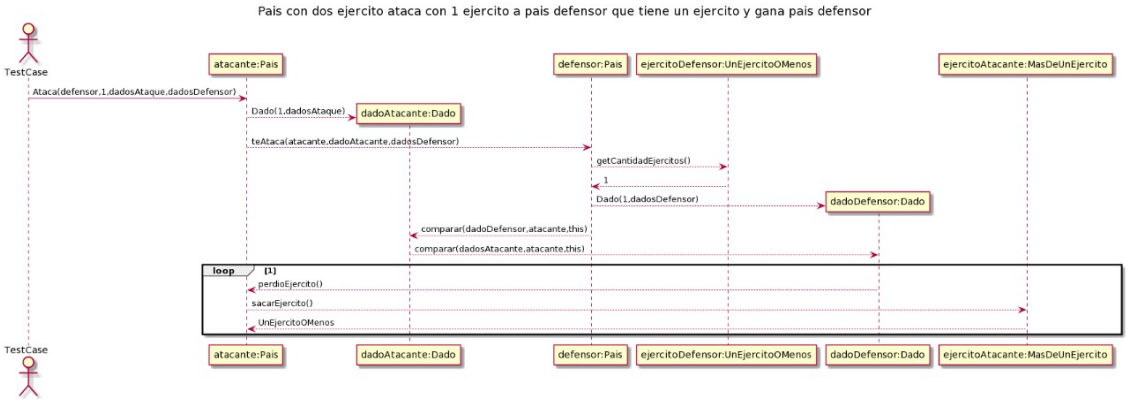

Figura 9: Diagrama de secuencia que muestra como un pais ataca a otro, donde el atacante pierde la batalla.

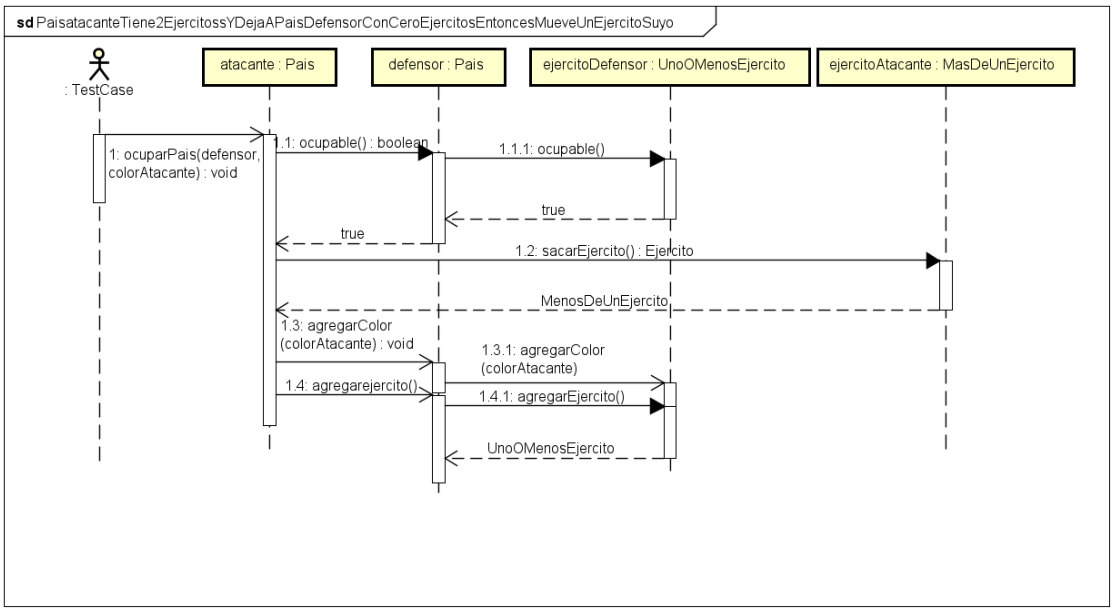

Figura 10: Diagrama de secuencia que muestra como un pais atacante despues de ganar, ocupa a su rival.

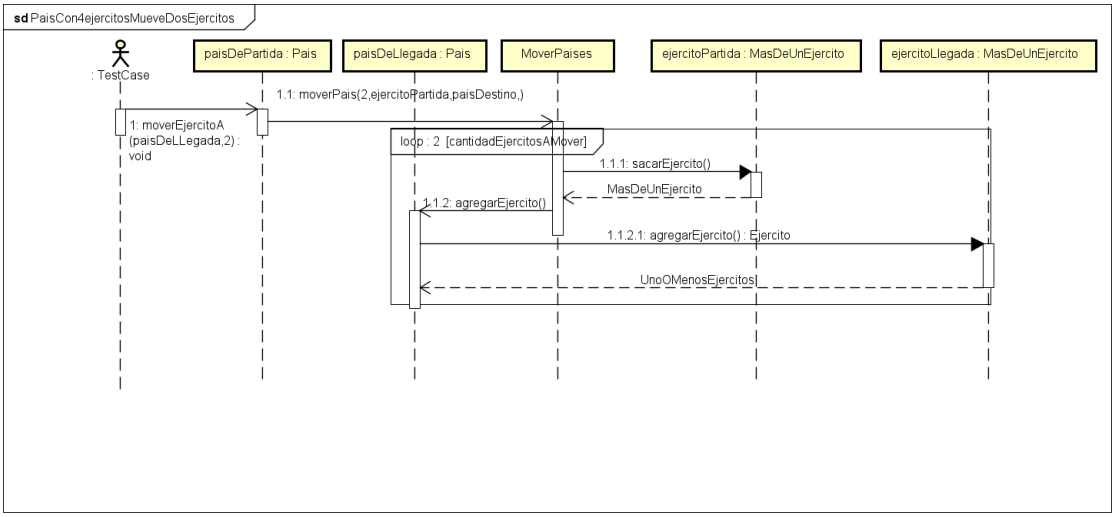

Figura 11: Diagrama de secuencia que muestra como un pais atacante mueve 2 de sus ejercitos al pais rival derrotado.

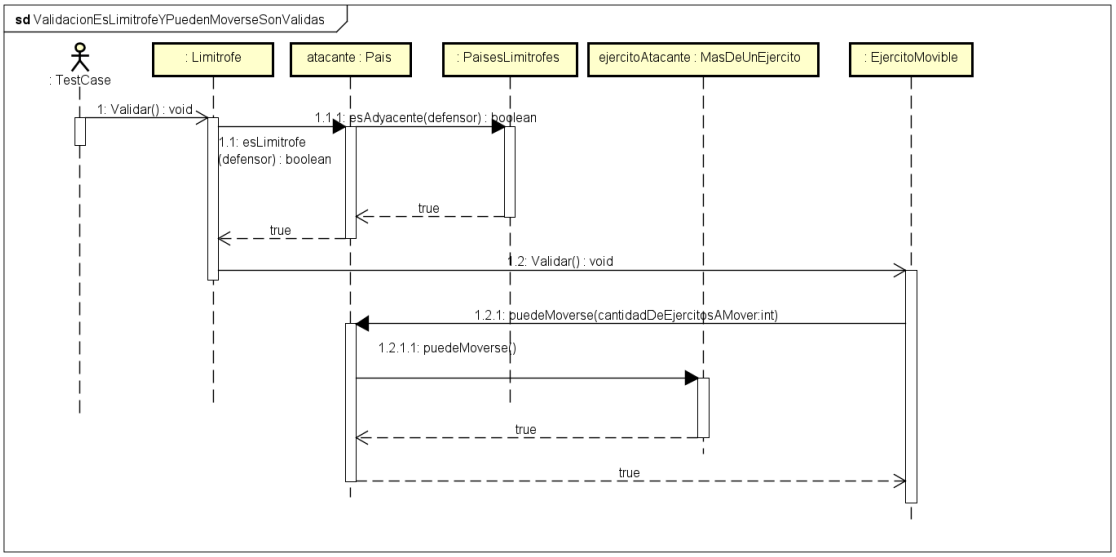

Figura 12: Diagrama de secuencia que muestra como valida si un pais puede moverse a un pais limitrofe.

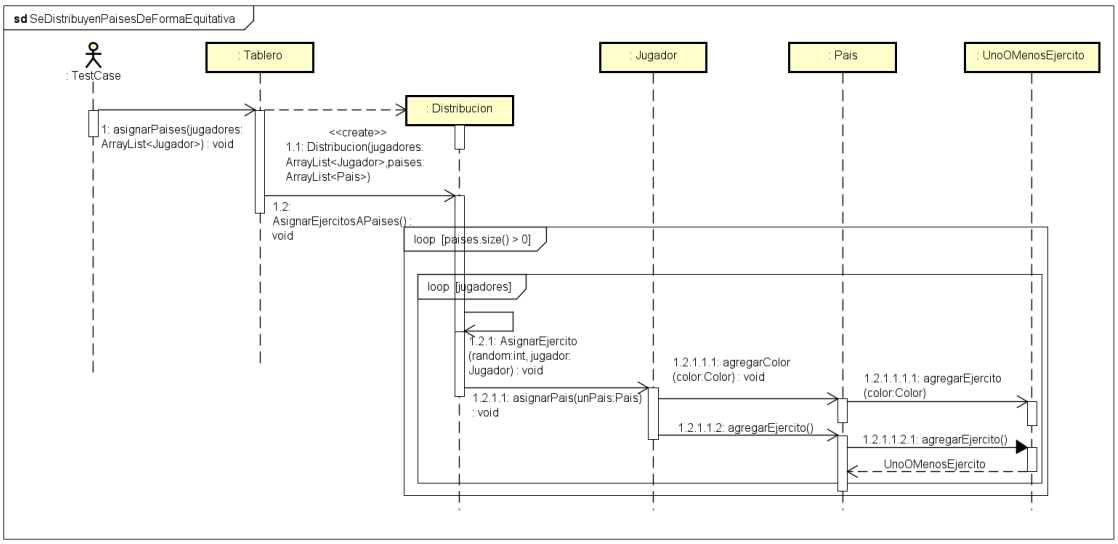

Figura 13: Diagrama de secuencia que muestra como se distribuyen los paises del TEG entre los jugadores que participaran.

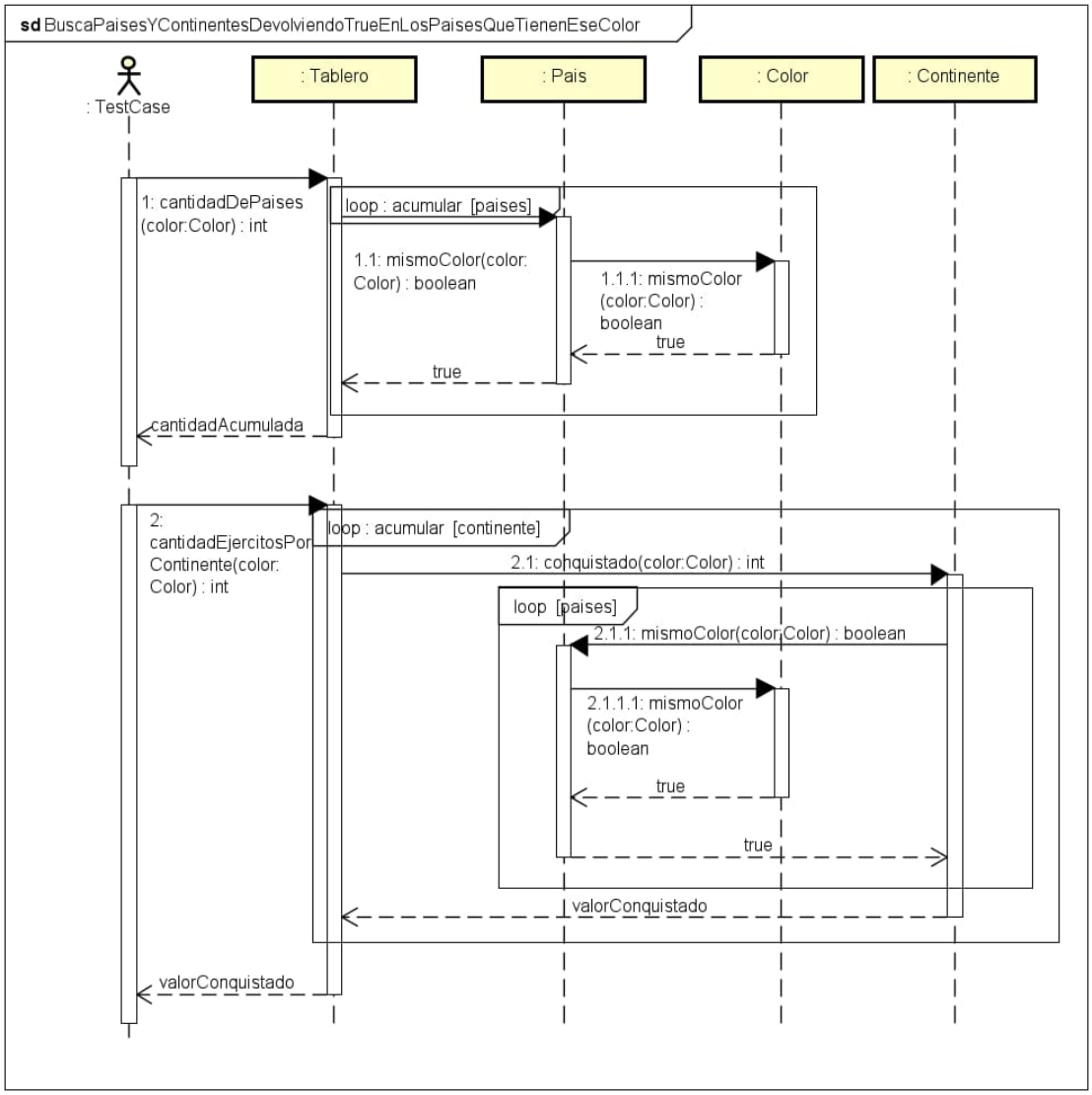

Figura 14: Diagrama de secuencia que muestra la comprobacion de un Color al tener todos los paises de un continente.

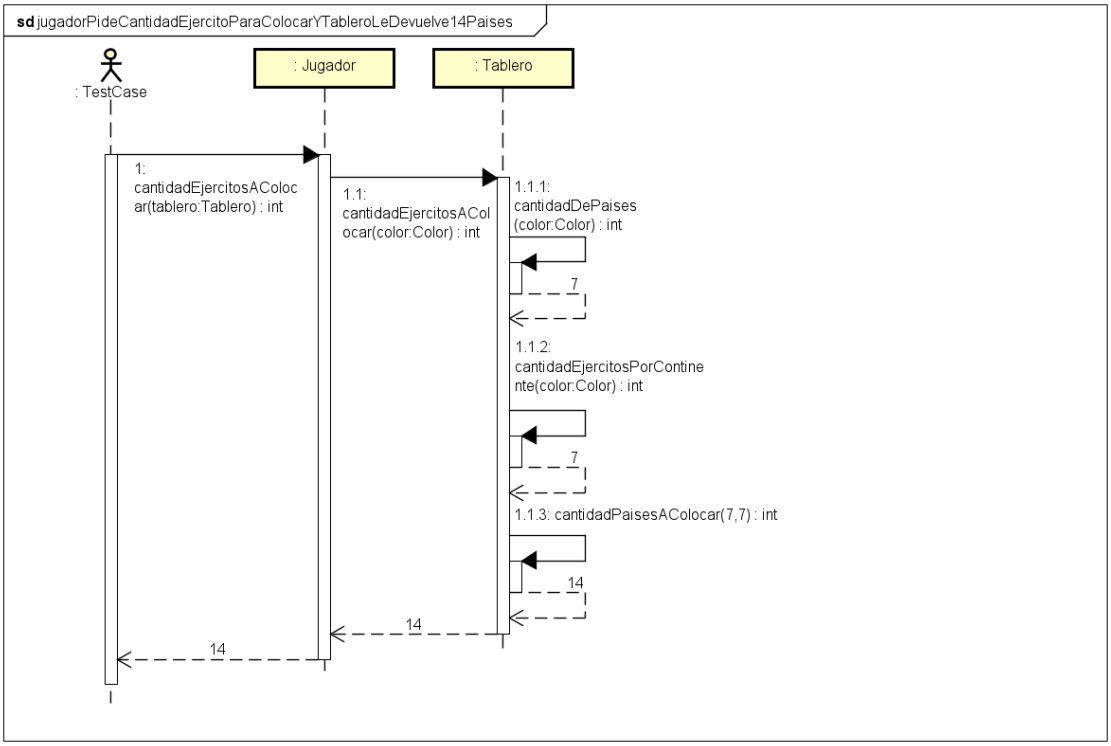

Figura 15: Diagrama de secuencia que muestra como calcula la cantidad de fichas que podra colocar, segun la cantidad de paises o continentes tenga.

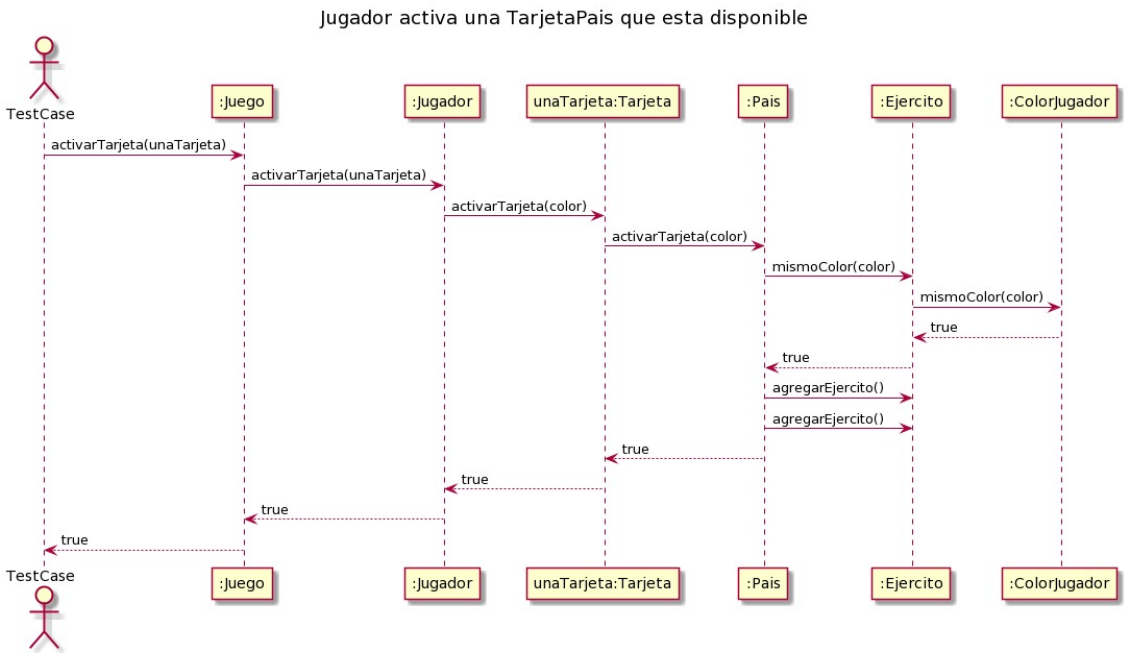

Figura 16: Como un Jugador coloca 2 ejercitos en un Pais de su color durante el turno de colocacion.

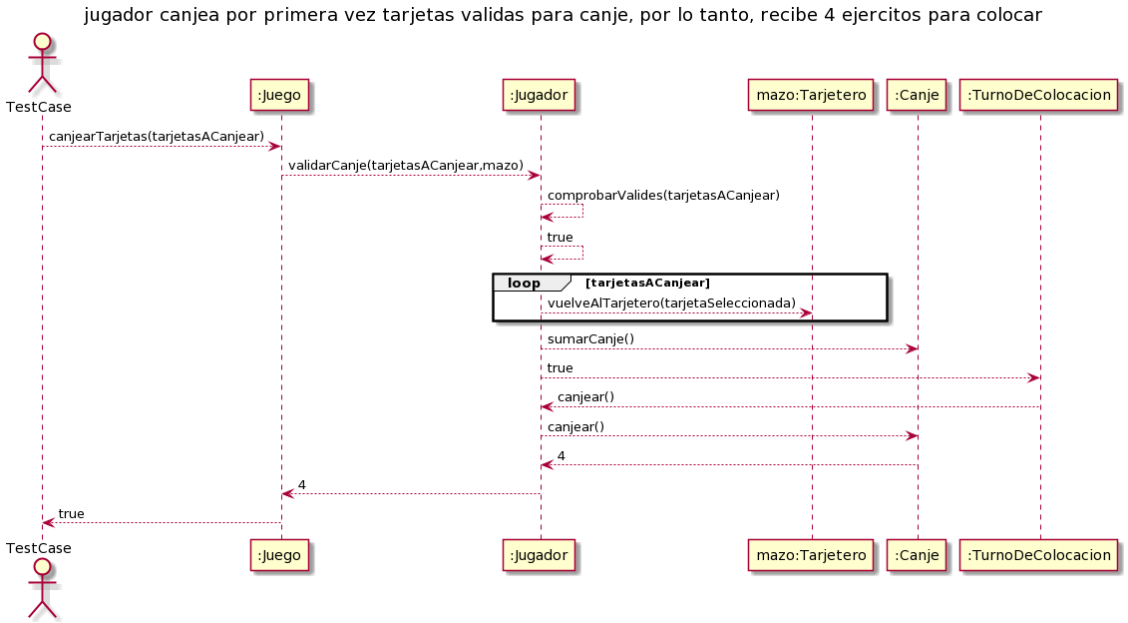

Figura 17: Un Jugador canjea 3 cartas y recibe 4 ejercitos (fichas).

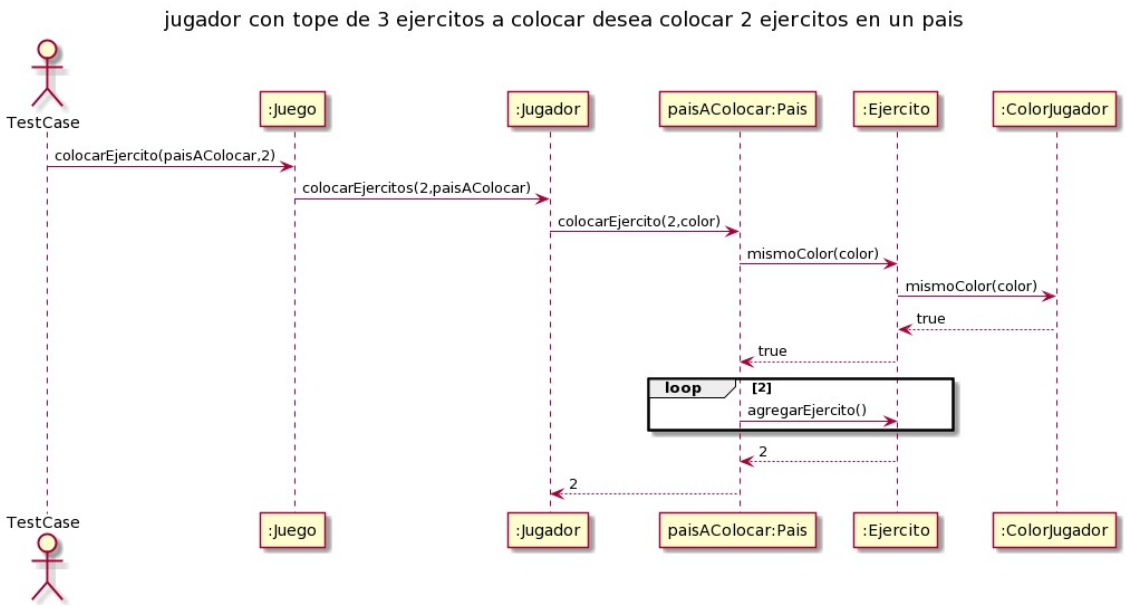

Figura 18: Como un Jugador coloca 2 ejercitos en un Pais de su color durante el turno de colocacion.

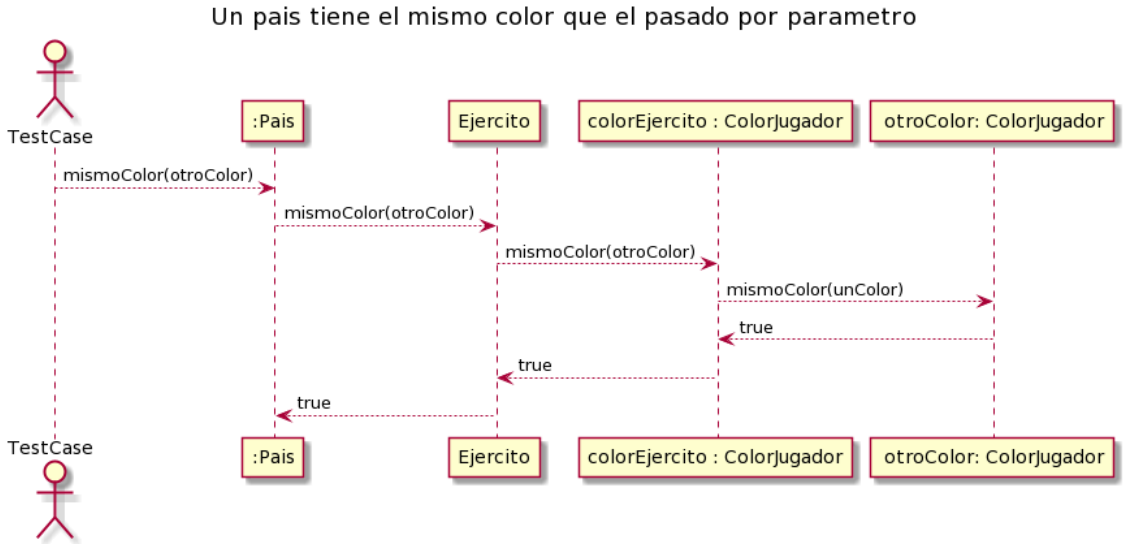

Figura 19: Comprueba que un pais tenga el color esperado.