

Trabajo Práctico II — JAVA

[7507/9502] Algoritmos y Programación III

Curso 01

Primer cuatrimestre de 2021

Alumno	Padron	Email
Martin Pata Fraile	106226	mpata@fi.uba.ar
Andrés Kübler	105238	akubler@fi.uba.ar
Sofía Marchesini	105994	smarchesini@fi.uba.ar
Santiago Vaccarelli	106051	svaccarelli@fi.uba.ar

Índice

1. Introducción	2
2. Supuestos	2
3. Modelo de dominio	2
4. Diagramas de clase	2
5. Diagramas de secuencia	4
6. Diagramas de paquetes	6
7. Diagramas de estado	6
8. Detalles de implementación	6
8.1. Clase Turnos	6
8.2. Clase Teg	7
8.3. Clase Jugador	7
9. Excepciones	8

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 que simula 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

Para la realización del trabajo tuvimos en cuenta los siguientes supuestos:

- Si el objetivo corresponde a la destrucción total de un ejercito color al cual no se encuentra jugando actualmente, el objetivo cambia automáticamente a la destrucción total del ejercito con próximo turno en la ronda inicial con respecto a quien le corresponde el objetivo

3. Modelo de dominio

El modelo del trabajo practico consiste en una clase general llamada TEG que representa el juego y que lo inicia. Esta se comunica con turnos quien maneja las rondas y es la que conecta a TEG con el usuario.

Turnos guarda el tipo de ronda de la ronda actual, que puede ser colocación o ataque, la ronda de como avanzan las rondas y el turno actual.

Al comenzar el juego, TEG asigna a cada jugador las cartaPais y los objetivosTeg y a su vez TEG guarda todos los jugadores y el Tablero. El jugador para ganar tiene que cumplir su objetivoTeg. Hay dos tipos de objetivoTeg: los de conquista y los de destrucción. Para cumplir el objetivoTeg el jugador tiene que ganar Países atacando desde otro Pais. El Pais para atacar depende de la cantidad de fichas que tiene su Ejercito y según la cantidad de fichas es la cantidad de Dados que el Jugador puede tirar

4. Diagramas de clase

Un diagrama de clase es un diagrama estático en el cual se representa la estructura de un sistema compuesto por clases, reflejando así sus atributos, métodos y las relaciones con otros objetos. A continuación se presentan algunos diagramas de clase correspondientes al trabajo practico.

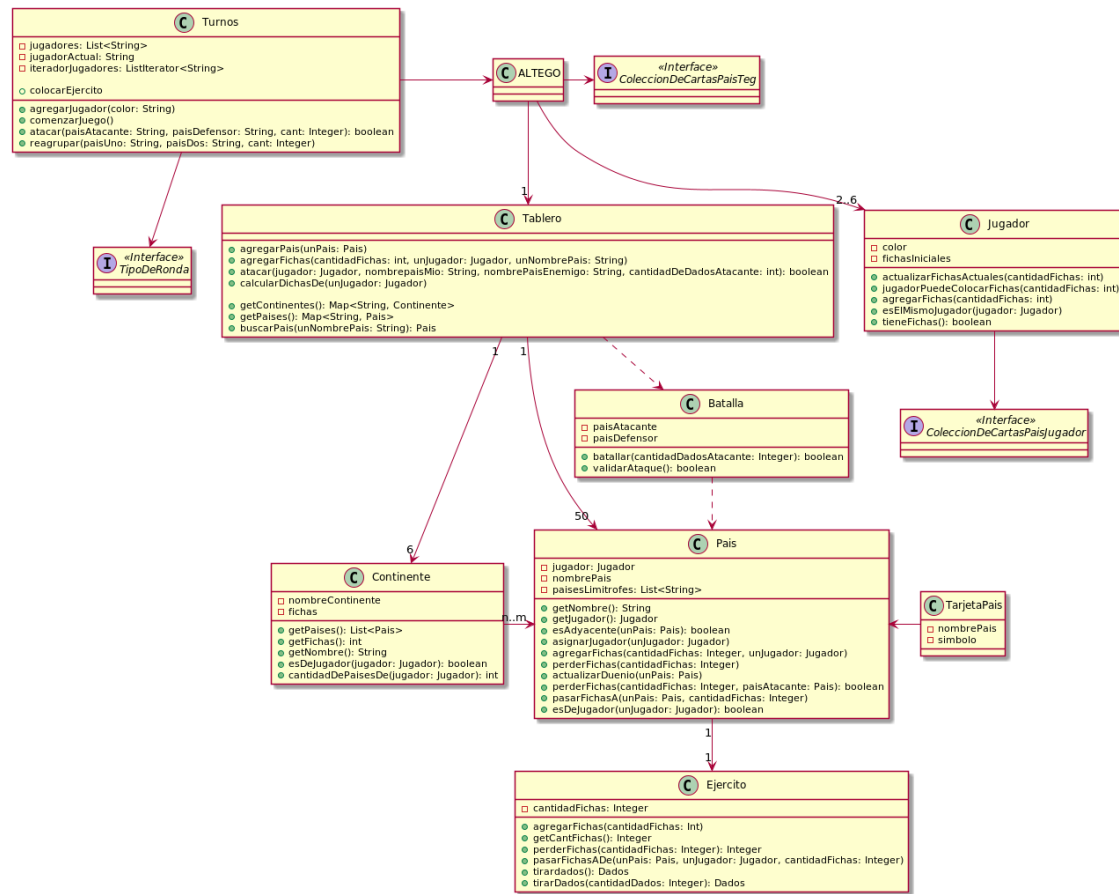


Figura 1: Modelo inicial TEG.

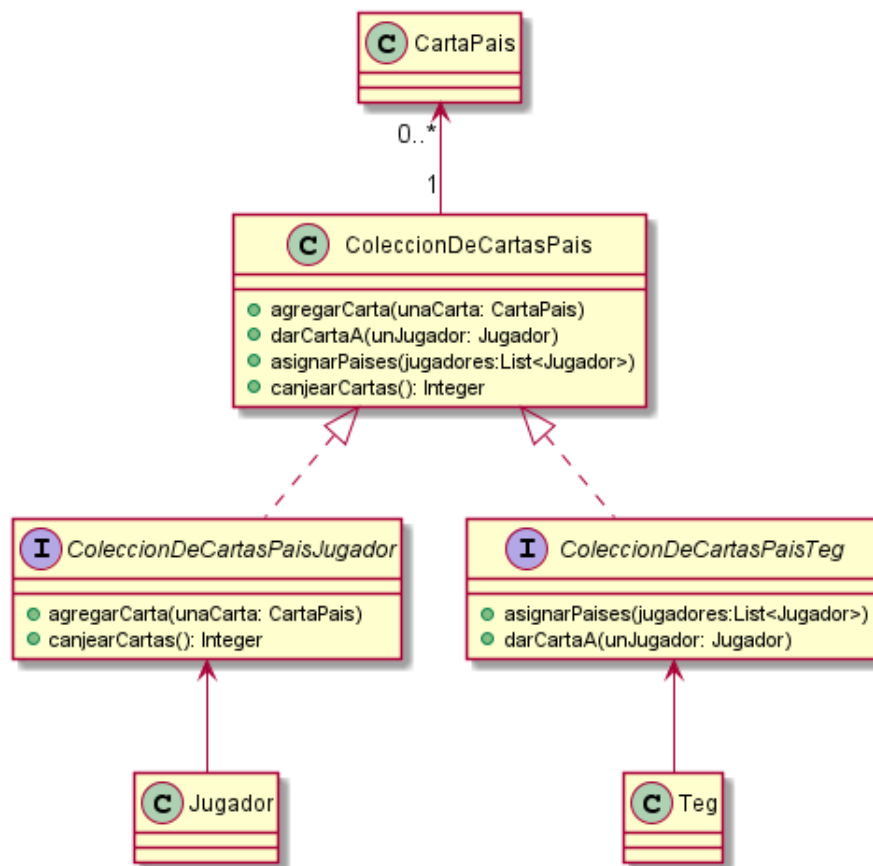


Figura 2: Interfaz de CartaPais.

5. Diagramas de secuencia

Sed scelerisque est at augue finibus, at faucibus erat venenatis. Phasellus euismod magna mi, nec malesuada quam pretium id. Donec vel diam eleifend, lobortis leo nec, semper sapien. Nunc ultricies mauris augue, id iaculis erat vehicula in. Nam molestie metus vel mi tincidunt lacinia. Nunc a cursus nisl, id sollicitudin mauris. Donec sit amet condimentum dolor, eget rutrum augue.

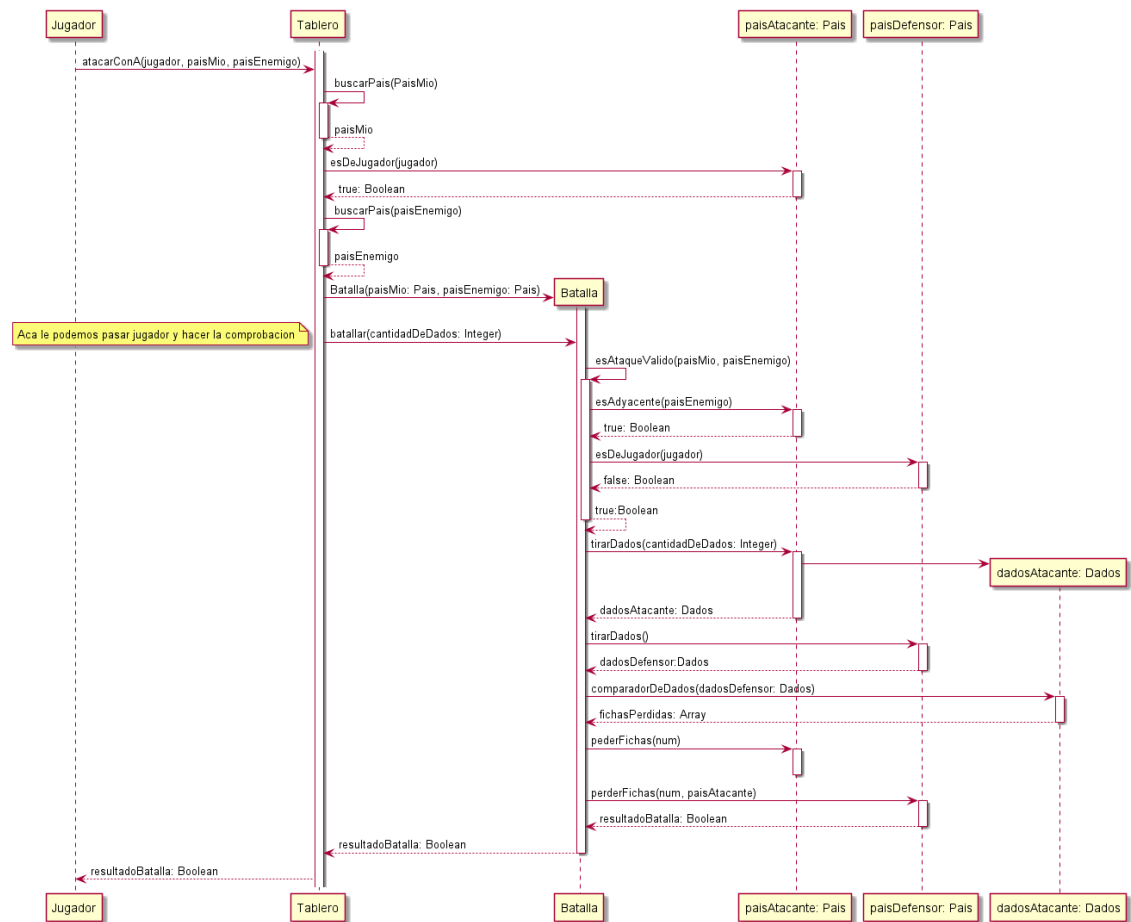


Figura 3: Un pais ataca a otro.

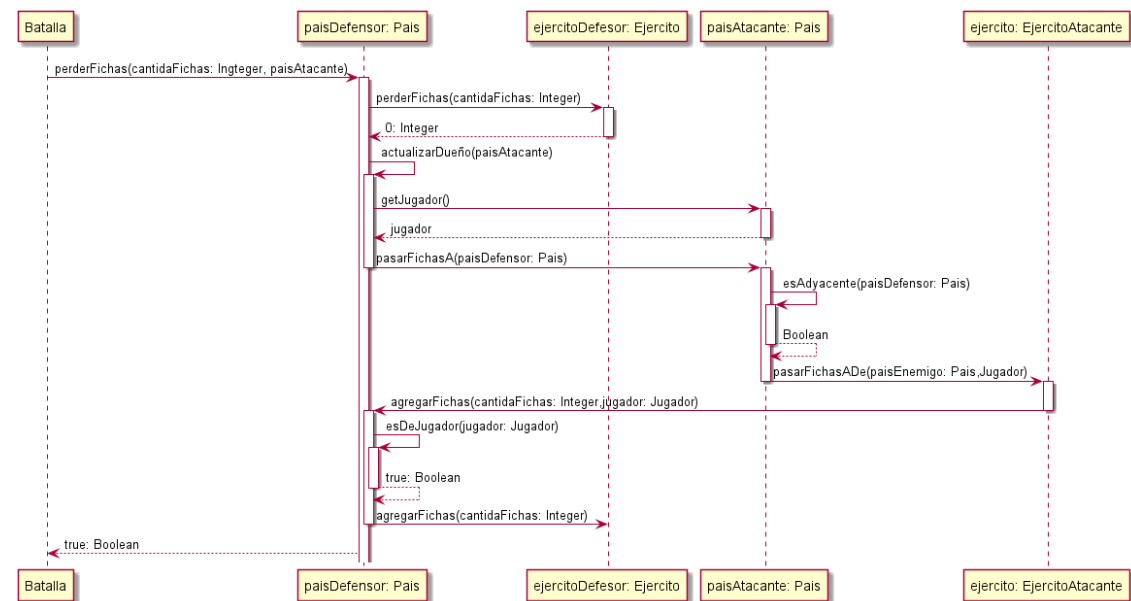


Figura 4: pais defensor pierde todas las fichas.

6. Diagramas de paquetes

7. Diagramas de estado

8. Detalles de implementación

8.1. Clase Turnos

Esta es la clase principal del programa y la encargada de comenzar el juego. Esta compuesta por una instancia de la misma clase Turnos, una lista de jugadores, un tipo de ronda y una clase Teg. Se utilizó el patrón Singleton para lograr la instancia de la clase. Los métodos pertenecientes a esta clase son: getInstance(), Turnos(), comenzarJuego(), atacarACon(), colocarFichas(), entre otros.

```

public static Turnos getInstance() {
    if(turnos == null){
        turnos = new Turnos();
    }
    return turnos;
}

public void comenzarJuego(List<String> listaJugadores){
    this.jugadores.addAll(listaJugadores);
    if(this.jugadores.size() < 2 || this.jugadores.size() > 6){
        throw new LimiteDeJugadoresException();
    }
    Collections.shuffle(jugadores); //Mezcla los jugadores como si tiraron Dados
    this.teg.comenzarJuego(jugadores);
    this.tipoDeRonda = new RondaColocacion(jugadores, teg) ;
}
  
```

Asimismo esta clase posee sobrecarga de metodos, lo cual consiste en la creación de varios métodos con el mismo nombre pero con diferente lista de tipos de parámetros.

```
public Turnos() {
    this.teg = new Teg();
}

public Turnos(Teg teg, List<String> jugadores){
    this.tipoDeRonda = newRondaColocacion(jugadores, teg) ;
    this.teg = teg;
    this.jugadores = jugadores;
}
```

8.2. Clase Teg

La clase Teg es la encargada de administrar las piezas del juego y la lógica de las mismas. Posee como atributos una clase Tablero, un Map de jugadores almacenando su nombre y valor, una colección de CartasPais y una lista de objetivos. Algunos de los metodos pertenecientes a esta clase son comenzarJuego(), asignarObjetivos(), colocarFichas(), atacarConA(), getGanador(), entre otros. En esta clase tambien se realizo la sobrecarga de metodos al momento de crear la clase mediante el metodo Teg().

```
public Teg() {
    this.tablero = ConstructorTablero.create(PATHJSON.concat("Teg-Tablero.json"));
    this.cartas = new MazoDeCartasPais(CartasPais.create(PATHJSON.concat("Teg-Cartas.json"), this));
    this.objetivos.addAll(Objetivos.objetivosConquista(PATHJSON.concat("Teg-Objetivos.json")));
}

public Teg(Tablero tablero, Map<String, Jugador> jugadores, MazoDeCartasPais mazoDeCartasPais) {
    this.tablero = tablero;
    this.jugadores = jugadores;
    this.cartas = mazoDeCartasPais;
}
```

8.3. Clase Jugador

Esta clase es la encargada de representar a cada uno de los jugadores durante la partida. Se encuentra compuesta por un color la cual la representa, un mazo de Cartas Pais, una cantidad de fichas desplegadas en el tablero y un objetivo de juego. Los metodos pertenecientes a esta clase son sacarFichas(), puedeColocarFichas(), agregarFichas(), darCartaPais(), esElMismoJugador(), tieneFichas(), hacerCanje(), conquistoPais(), darObjetivo(), gano(), activarCartas() y sacarConquista().

```
public int sacarFichas(int cantidadFichas){
    if(this.puedeColocarFichas(cantidadFichas)){
        this.fichas = this.fichas - Math.abs(cantidadFichas);
    }else{
        this.fichas = 0;
    }
    return this.fichas;
}
```


9. Excepciones

AtaqueNoValidoException Esta excepción tiene como finalidad limitar el ataque al momento de que un jugador quiera realizar un ataque con un país que no le pertenece o si el país a atacar no es adyacente a este.

EjercitoConUnaFichaNoPuedeAtacarException Esta excepción busca restringir un ataque a que el país con el cual el jugador ataca tenga mas de una ficha.

EjercitoNoPuedeTirarEsaCantidadDeDadosException Esta excepción busca restringir...

NoSePuedenCrearCeroDadosException Esta excepción tiene como finalidad encontrar un posible error en la creación de dados, el cual consiste en crear una clase dados con una cantidad de dados menor o igual a cero.

MazoNoTieneSuficientesCartasException Esta excepción tiene como finalidad limitar el reparto de cartas si estas ya fueron todas repartidas y el mazo se encuentra vacío.

JugadorSigueTeniendoFichasException Esta excepción busca solventar la ruptura de la logística del juego al momento en el cual un jugador quiera finalizar su turno en la ronda colocación pero le siguen quedando fichas por colocar.

LimiteDeJugadoresException Esta excepción busca restringir la cantidad de jugadores por partida dentro de un mínimo de 2 y un máximo de 6 jugadores.

NoSePuedeHacerEstaAccionEnEstaRondaException Esta excepción busca restringir los métodos de las distintas clases de tipo rondas, ya que cada ronda tiene un comportamiento diferente y algunas acciones no están permitidas.

PasajeDeFichasNoValidoEnAtaqueException Esta excepción tiene como finalidad restringir el pasaje de fichas de un país a otro en ronda ataque, solamente si el país fue conquistado en el ataque.

JugadorNoPoseePaisException Esta excepción tiene como finalidad evitar problemas al momento de agregar fichas a un país que no corresponde al jugador o pasar fichas de un país a otro país el cual no pertenece al jugador, ambos casos corresponden a los métodos **agregarFichas** y **pasarFichasA** de la clase País.

JugadorNoTieneSuficientesFichasException Esta excepción restringe la colocación de fichas en un país propio de un jugador si este jugador no posee ficha alguna disponible para poder colocar.

PaisNoEsLimitrofeException Esta excepción busca limitar el pasaje de fichas entre países si los dos países involucrados en el pasaje de fichas no son limítrofes.

PaisSinSuficientesFichasParaPasarException Esta excepción tiene como finalidad restringir el pasaje de una cierta cantidad de fichas a un país si la cantidad de fichas indicadas por el jugador supera a las que dispone el ejercito durante el ataque.