Trabajo Práctico 2 — ALTEGO

[7507/9502] Algoritmos y Programación III
Curso 1
Primer cuatrimestre de 2021
Grupo 10

Alumna	Padrón	Email
PONT, María Fernanda	104229	mpont@fi.uba.ar
DI NARDO, Chiara Anabella	105295	cdinardo@fi.uba.ar
GADDI, María Pilar	105682	mgaddi@fi.uba.ar

Índice

1. Introducción

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

2. Supuestos

Debido a que en algunos casos la resolución de ciertos aspectos (no contemplados en la especificación del trabajo) quedaban a criterio del alumno, se tuvieron en cuenta los siguientes supuestos: En esta primera entrega se siguieron las reglas del juego establecidas por la cátedra. Entre ellas están:

- El atacante no podrá atacar con la misma cantidad de fichas que tiene el mismo en su país ni con una cantidad mayor.
- El defensor atacará con la cantidad de fichas que tiene el mismo en el país que posee y que es atacado.
- Se podrá jugar con un máximo de tres dados por ataque, y el atacante va a poder elegir cuantos dados utilizar, siempre y cuando esa cantidad no sea mayor o igual a la cantidad de fichas (ver supuesto 1).
- El jugador debe saber cuáles países son válidos. El mismo no puede inventar países nuevos.

3. Diagramas de clase

En las siguientes imagenes se tiene un diagrama de clases, en estos se muestran las clases utilizadas en el trabajo y las formas de las mismas.

El primer diagrama de clase muestra como se planteo la idea de la primera entrega del juego T.E.G. Se tiene la clase Juego que tiene una relación de agregación debido a que por más que no exista el Juego, los jugadores (personas) seguirán existiendo. Luego la clase Jugador puede tener 1 o varios paises, por eso presenta una relación de asociación con la clase Pais. Por último, Pais puede realizar ataques (que pueden ganar en su totalidad y debido a eso conquistar otros paises) y tirar dados.

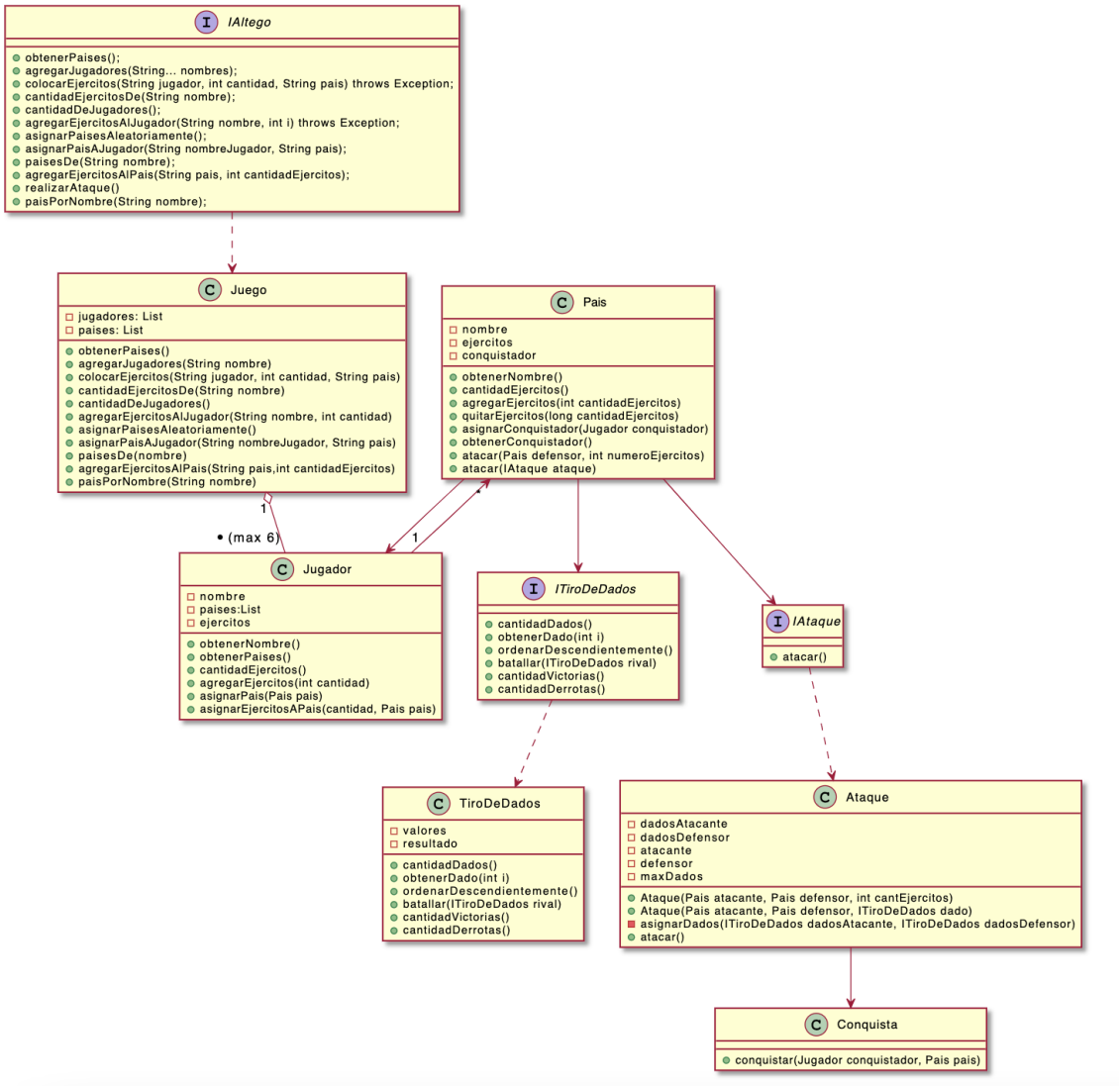

Figura 1: Diagrama de Juego (ALTEGO).

4. Diagramas de secuencia

En la siguiente imagen se muestra la secuencia de atacar un pais y que como resultado el pais defensor salga victorioso.

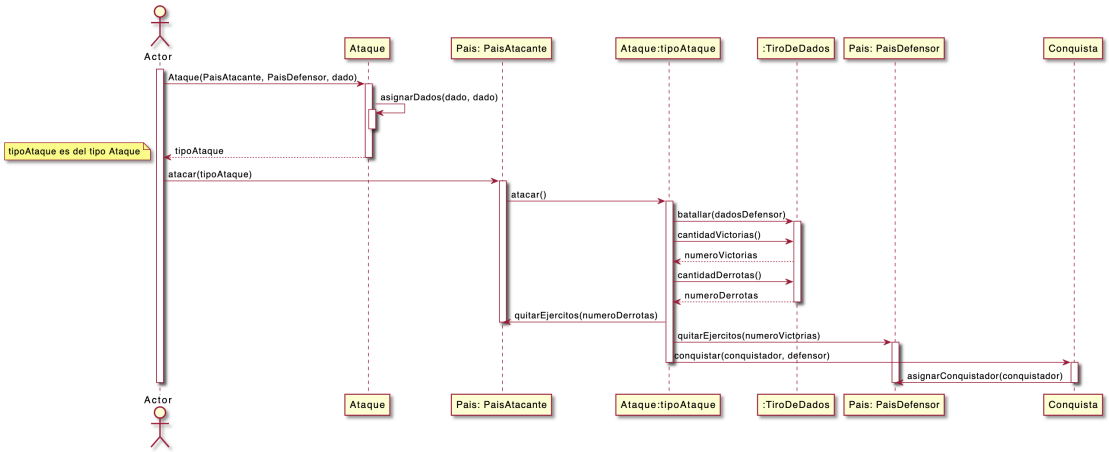

Figura 2: Atacar pais, gana defensor.

En la siguiente imagen se muestra la secuencia de asignar ejércitos en un país.

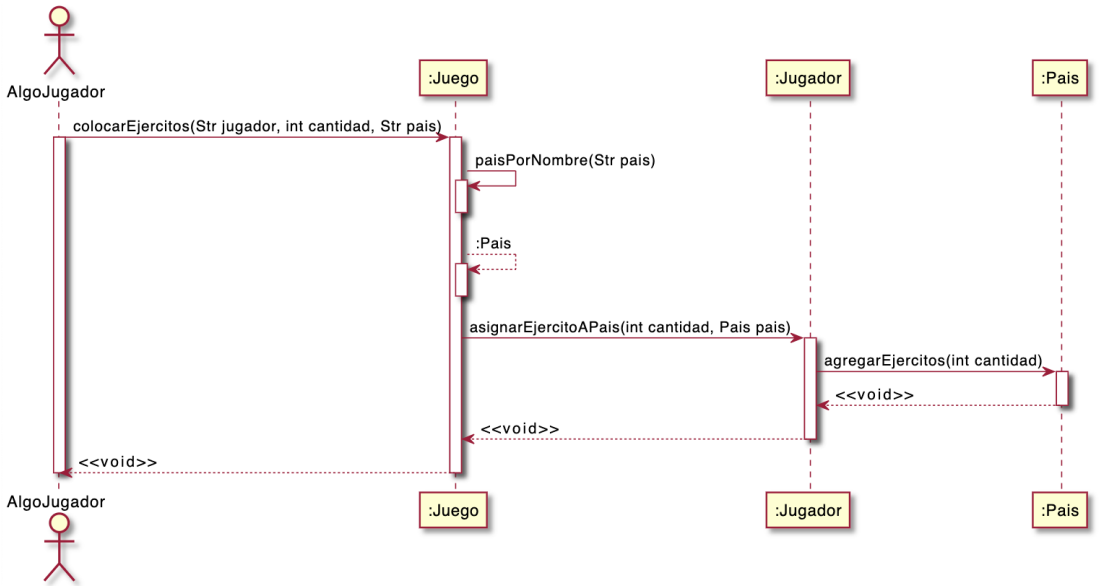

Figura 3: Jugador coloca ejércitos en un país.

5. Diagrama de Paquetes

En la siguiente imagen se muestra el diagrama de paquetes UML:

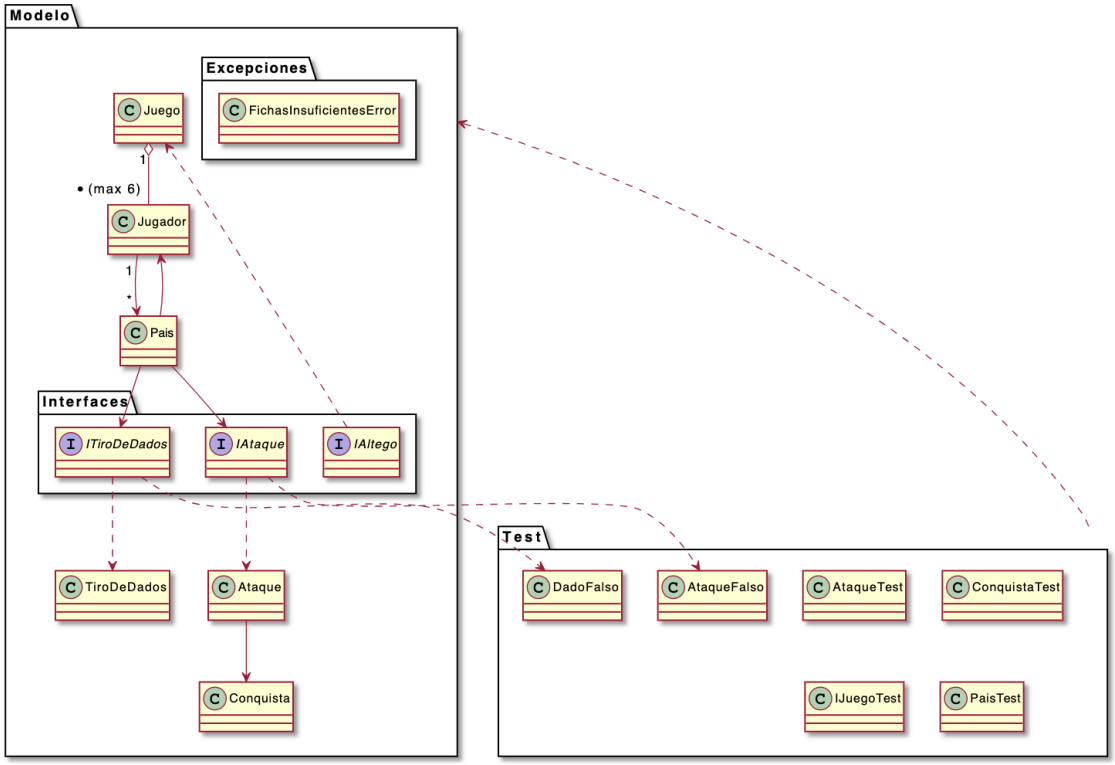

Figura 4: Diagrama de Paquetes UML.

6. Diagramas de Estado

En la siguiente imagen se muestra el diagrama de estado UML, en relación a cuando el Jugador se inicializa y luego se le agregan ejercitos y paises:

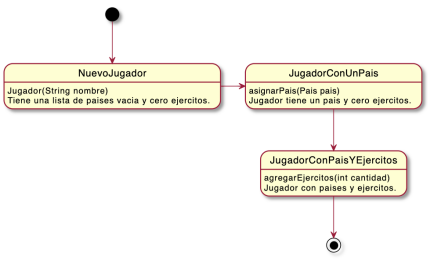

Figura 5: Diagrama de Estado Jugador Nuevo.

En la siguiente imagen se muestra el diagrama de estado UML, en relación a cuando el Pais se inicializa y luego se le agregan ejercitos:

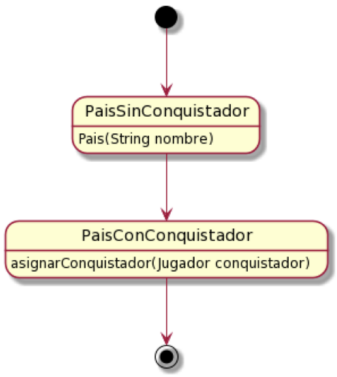

Figura 6: Diagrama de Estado Pais Nuevo.

7. Detalles de implementación

Las partes esenciales del modelo son TEGEngine (Juego) y ALTEGO. TEGEngine es el 'engine' del juego y no sabe nada sobre interacción del usuario, sino que recibe mensajes de juego como 'colocar ejércitos' o 'terminar fase'. Luego, ALTEGO es una clase que utiliza el engine de acuerdo a eventos de UI, es decir es el único que sabe de JavaFX y los eventos de JavaFX.

TEG está compuesto de 2 fases:fase de inicio y fase de juego. Además, la fase de juego es un ciclo de 3 etapas: atacar, reagrupar y colocar ejércitos.

Cada iteración es un turno de un jugador. TEGEngine es quien define las reglas del juego. Durante la fase de inicio TEGEngine no acepta mensajes como atacar(), durante la fase de juego no acepta repartirPaises(). Además no se puede agregar más jugadores después de repartir los paises, en ese sentido existen 'etapas' dentro de la fase de inicio.

A continuación se detallará la implementación interna de algunas clases interesantes para lograr un mejor entendimiento del código.

7.1. Juego

La clase Juego implementa la interfaz IAltego. Conoce a los jugadores y a los países participes del juego. Tendrá el siguiente constructor, que agrega a todos los jugadores al juego:

```
public Juego(String[] nombresJugadores) throws Exception {
    if ( nombresJugadores.length <= 0 )
        throw new Exception();
        jugadores = Arrays.asList(nombresJugadores)
            .stream()
            .map(n -> new Jugador(n))
            .collect(Collectors.toList());
}
```

7.2. Jugador

La clase Jugador tiene como atributos un nombre, una lista de paises y una cantidad de ejercitos (que serían las fichas totales que posee). Para asignar una cantidad de ejercitos a un pais en el que un jugador X es dueño se utiliza el siguiente método:

```
public void asignarEjercitosAPais(int cantidad, Pais pais) throws Exception {
    if(cantidad > ejercitos) throw new FichasInsuficientesError("El jugador
    no tiene suficientes fichas.");
    if (!paises.contains(pais)) throw new Exception();
}
```

7.3. Pais

La clase Pais es la responsable de atacar a otros países. tiene un jugador que lo controla, y una cantidad de soldados. Cuando hay un ataque el pais NO recibe soldados que no sean de su jugador. Cuando es conquistado por otro jugador B, B tiene que establecer al menos uno de sus soldados en el país.

Tiene como atributos un nombre y una determinada cantidad de ejercitos (que serían las fichas totales que tiene el país). También tiene un dueño, que sería su conquistador (un jugador).

El pais podrá atacar estableciendo un tipo de ataque personalizado (para que distintas circunstancias puedan ser testeadas) de la siguiente manera:

```
public void atacar(IAtaque ataque) {
    ataque.atacar();
}
```

También se podrá optar por un ataque predeterminado, sin necesidad de establecer un ataque específico:

```
public void atacar(Pais defensor, int numeroEjercitos) throws Exception {
    IAtaque ataque = new Ataque(this, defensor, numeroEjercitos);
    atacar(ataque);
}
```

7.4. TiroDeDados

La clase TiroDeDados implementa la interfaz ITiroDeDados. Tiene como atributos una lista de valores que son los valores de los dados que se van tirando durante la partida, y una lista de resultados que son aquellos que van resultando de la comparación de valores de los dados, entre el atacante y el defensor, para luego poder determinar al ganador. Además de la clase TiroDeDados, se implementó un mock llamado DadoFalso, y este es utilizado para poder realizar las pruebas correspondientes a la clase Ataque, descripta a continuación.

7.5. Ataque

La clase Ataque implementa la interfaz IAtaque. Conoce a un pais atacante y a un pais defensor, como así también a los dados de ambos. Se podrán realizar dos tipos de ataques, un 'Ataque Predeterminado' y un 'Ataque Falso'. Este último fue utilizado principalemnte para que las pruebas unitarias puedan ser consistentes y para que se pueda asegurar que no cambien de resultado.

Para lograr un Ataque Predeterminado se utiliza el siguiente constructor:

```
public Ataque(Pais atacante, Pais defensor, int cantEjercitos) throws Exception {
    this.atacante = atacante;
    this.defensor = defensor;

    if(atacante.ejercitos <= cantEjercitos || cantEjercitos > maxDados)
    throw new FichasInsuficientesError("El jugador sólo puede atacar con"
    + (atacante.ejercitos - 1) + "ejércitos.");
    asignarDados(
        new TiroDeDados(cantEjercitos),
        new TiroDeDados(Math.min(defensor.ejercitos, maxDados))
    );
}
```

Para lograr un AtaqueFalso se utiliza el siguiente constructor:

```
public Ataque(Pais atacante, Pais defensor, ITiroDeDados dado) throws Exception{
    this.atacante = atacante;
    this.defensor = defensor;
    asignarDados(dado, dado);
}
```

Para este último se debe usar el mock DadoFalso mencionado anteriormente. AtaqueFalso también sería un mock para poder probar la clase Pais con un ataque personalizado. La misma fue comentada anteriormente.

7.6. Conquista

La clase Conquista presenta únicamente al método conquistar y, en caso de que se lo llame, se encarga de cambiar al dueño del país que se le pasa por parámetro. Esto lo hace de la siguiente manera:

```
public void conquistar(Jugador conquistador, Pais pais) {
    pais.asignarConquistador(conquistador);
}
```

8. Excepciones

FichasInsuficientesError Es lanzada cuando el jugador intenta agregar ejércitos a un país pero no tiene las fichas suficientes y, también, cuando se quiere atacar un país con más ejércitos que en el país atacante.