

# - Reconquistando la tierra de fantasía

Algoritmos y programación II

**UNTREF**

Equipo:	Turing Team
Carrera:	Ingenieria en Computacion
Materia:	Algoritmos y programacion II
Comisión:	334
Fecha de Entrega:	02-07-21



# Reconquistando la tierra de fantasía

Algoritmos y programación II



Integrantes:

Pedro Villar  
Franco Spataro  
Juan Elías Ribeiro  
Juan Manuel Calviño

Tutor:

Leandro Doctors López

Repositorio:

<https://github.com/juancalvino/TuringTeam>



# UNTREF


UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

## - Introducción

Una raza de guerreros debe reconquistar sus tierras que han sido arrebatadas. Para ello, deben atravesar diferentes pueblos hacia su destino y así poder recuperar lo que es suyo.

En esta odisea, los guerreros tendrán que atravesar diversos caminos. En algunos se toparán con pueblos aliados, los cuales se sumarán a la travesía de reconquistar la tierra perdida, en otros se encontrarán con pueblos enemigos, quienes se interpondrán en su camino y lucharán hasta desmayarse para evitar que lleguen a su destino.

No obstante, estos guerreros cuentan con la ventaja de tener al hechicero del código de su lado, quién puede predecir si esta reconquista es factible y, en caso de serlo, cuántos guerreros lograrán sobrevivir y cuánto tiempo durará la travesía.



# Reconquistando la tierra de fantasía

Desafío, contexto y desarrollo

## Desafío

Construir un programa que, recibiendo un archivo '.txt' con la información necesaria, instancie un escenario deseado y realice una simulación que prediga si es factible concretar la reconquista y, en caso de serlo, qué camino deben tomar los guerreros, cuánto tiempo demorarán y cuántos sobrevivirán a la travesía.

## Contexto

El contexto está dado por un conjunto de pueblos que están conectados entre sí por medio de caminos, y cada camino demora un cierto tiempo en ser recorrido.

A su vez, cada pueblo posee un ejército. Y cada ejército puede ser propio, aliado, o enemigo. Además, cada ejército está compuesto por guerreros, los cuales pueden pertenecer a cuatro tipos diferentes de razas: **Wrives, Reralopes, Radaiteran y Nortaichian**.

Todas las razas pueden atacar, recibir ataques y descansar, pero cada una lo hace de una forma particular.

## Desarrollo

En primera instancia, se interpretó la consigna de forma individual y luego de forma grupal. Llevamos a cabo reuniones dos veces por semana con el fin de avanzar hacia una solución final. Con el objetivo de mejorar la dinámica y los procesos de desarrollo concluimos en dividir el trabajo en problemas más pequeños y seleccionar las herramientas necesarias para abordar el desarrollo.

- El trabajo se dividió en cinco partes:
  - ▶ Creación e implementación de los guerreros.
  - ▶ Implementación del algoritmo de Batalla
  - ▶ Desarrollo del Escenario
  - ▶ Procesamiento y carga del archivo '.txt'.
  - ▶ Desarrollo del algoritmo predictivo.

# Reconquistando la tierra de fantasía

Creación e implementación de los guerreros

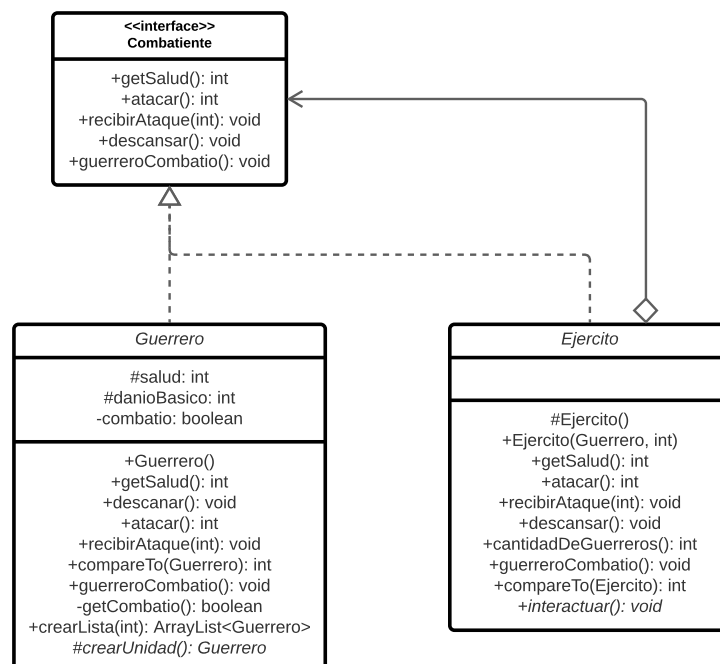
## Creación e implementación de los guerreros

- Se descompuso el problema en tres partes:
  - Desarrollo de las razas.
  - Desarrollo de los ejércitos.
  - Desarrollo de la batalla.

En primer lugar, asignamos el desarrollo de una raza diferente para cada integrante. Para luego cruzar las coincidencias en cuanto a atributos y métodos y formar una clase padre más general.

El uso de TDD en esta instancia fue clave para poder cumplir en las pruebas con una cobertura mayor al 92%.

Una vez implementadas las razas, concluimos que los ejércitos podrían estar compuestos tanto de otros ejércitos como de guerreros. Al ser el comportamiento de los ejércitos y de los guerreros, similar, el patrón de diseño **Composite** es una excelente alternativa para diseñar estas relaciones.



En este caso, la interfaz '*Combatiente*' indica los comportamientos básicos que deben poder realizar tanto la clase '*Ejercito*', que es el elemento compuesto, como la clase '*Guerrero*', que es la clase hoja.

# Reconquistando la tierra de fantasía

## Creación e implementación de los guerreros

Durante el proceso de desarrollo de las pruebas, se implementó la instancia '*GuerreroHack*' (una extensión de la clase Guerrero que puede ser utilizada únicamente en el ámbito de las pruebas) con la finalidad de facilitar los resultados e interacciones de las clases anteriormente mencionadas.

Para la implementación de las instancias de ejércitos, dirigimos nuestro enfoque en entender qué nos pedía la consigna.

Se solicitaba que existieran ejércitos aliados y enemigos. A su vez, se indicaba que el 'jugador' debía poseer un ejército único, debiendo ser el mismo también capaz de incorporar ejércitos aliados a sus filas.

Debido a esto, se desarrollaron cuatro clases de ejércitos:

- ▶ ***EjercitoUnico***

teniendo en cuenta que solo puede existir un único ejército perteneciente al jugador, se implementó esta clase utilizando el patrón de diseño Singleton, que garantiza que exista una única instancia para dicha clase.

- ▶ ***EjercitoPropio***

contiene y ordena los ejércitos pertenecientes al jugador. Es la clase responsable de invocar y modificar al EjercitoUnico. Puede incorporar al EjercitoUnico y también tiene la posibilidad de incorporar más de un EjercitoAliado. Ordena la prioridad en que estos ejércitos interactúan con el EjercitoEnemigo.

- ▶ ***EjercitoEnemigo***

lo único que puede hacer es interactuar con el EjercitoUnico

- ▶ ***EjercitoAliado***

puede interactuar con EjercitosEnemigos.

# Reconquistando la tierra de fantasía

Implementación del algoritmo de batalla  
y desarrollo del escenario

## Implementación del algoritmo de Batalla

Luego de haber verificado el correcto funcionamiento de los guerreros y los ejércitos, comenzamos a analizar cómo debía comportarse la interfaz *'Combatiente'* en relación a la batalla y qué características tendrían los agentes en esta interacción.

La clase *'Batalla'*, se encarga de posicionar el *'EjercitoPropio'* contra un *'EjercitoEnemigo'* e ir llamando (polimórficamente) a sus respectivos elementos base (un guerrero) y hacerlos batallar.

### Interacción de Batalla

En un principio, el *'EjercitoEnemigo'* llama al método *recibirAtaque()*, pasándole como parámetro el método *'atacar()'* del *'EjercitoPropio'*.

Luego, se irán alternando los ejércitos hasta que uno de los mismos ya no contenga ningún *'Combatiente'* disponible.

Finalmente, retorna el ejército del *'Combatiente'* sobreviviente.

## Desarrollo del Escenario

Para modelar el *'Escenario'*, decidimos utilizar la experiencia adquirida durante las clases de Grafos. Modelando el mismo con la ayuda de un mapa, que contiene todos los pueblos

Cada *'Pueblo'* posee también un mapa, que contiene todos los caminos adyacentes hacia otros pueblos a partir de sí mismo, y contiene un ejército designado. De esta forma, podemos tener a todos los pueblos conectados de una forma eficaz.

Dicho *'Escenario'* también debía ser único y accesible desde cualquier clase que lo requiera, por lo que fue menester aplicar el patrón de diseño **Singleton**.

# Reconquistando la tierra de fantasía

Procesamiento y carga del archivo de entrada

## Procesamiento y carga del archivo de entrada

Teniendo en cuenta la consigna, nosotros sabemos que el programa recibirá un archivo de tipo '.txt' como entrada de datos y que, a su vez, cada línea del mismo representa un dato que nos indica cómo debe ser instanciado el Escenario.

Y sabemos que la suma de ellos, procesados por el programa que hemos desarrollado, nos permitirá verificar si existe un camino hacia la reconquista del pueblo de fantasía.

Es por ello que necesitamos que el programa reciba todos los datos de forma correcta, ordenada y completa para evitar cualquier falla que provenga de los datos ingresados.

En un principio, se pensó en ir agregando los datos a medida que se leían las líneas del archivo, pero llegamos a la conclusión que sería más apropiado y robusto recopilar los datos, verificarlos y luego cargarlos en el '*Escenario*'.

Tomando en cuenta esto, decidimos que la clase '*CargaDeArchivos*' debía recopilar la información ingresada, ordenarla y saber tratar cualquier falta o error que pudiera existir en el ingreso de datos para luego poder subir los datos de forma correcta.

Al ser esta clase la encargada de recopilar y ordenar los datos, evitamos los problemas que podrían derivarse de que un usuario ingrese los datos de forma desordenada. Ya que, si la información se encuentra desordenada pero completa, esta clase resolverá este inconveniente.



## Reconquistando la tierra de fantasía

Procesamiento y carga del archivo de entrada  
y desarrollo del algoritmo predictivo.

Al introducir un archivo de texto, la clase *'CargaDeArchivos'* deberá leer línea por línea e interpretar si corresponde a uno de los siguientes casos:

- ▶ Cantidad total de pueblos.
- ▶ Un pueblo, con sus respectivos datos.
- ▶ Ruta del objetivo.
- ▶ Camino que conecta dos pueblos.
- ▶ Línea invalida.

Una vez decodificada la información la clase *'CargaDeArchivos'* procede a cargar los datos en la clase *'Escenario'*.

## Desarrollo del algoritmo predictivo

Una vez instanciado el *'Escenario'*, con sus respectivos *'Pueblos'* y *'Caminos'*, y realizada la correcta carga de datos de los mismos, sólo restaba realizar la predicción de la travesía.

Para este fin, resolvimos implementar una clase *'Hechicero'*, que debía encargarse de:

- ▶ Buscar el camino más corto desde un pueblo origen hacia los demás pueblos.
- ▶ Luego, armar una ruta a seguir a partir únicamente de los caminos necesarios para llegar al destino elegido.
- ▶ Recorrer dicha ruta e interactuar de forma apropiada con los posibles Pueblos.
- ▶ Devolver si dicho recorrido fue satisfactorio (alcanzó el destino) o fallido (el ejército fue derrotado).

## Reconquistando la tierra de fantasía

Desarrollo del algoritmo predictivo.

Teniendo en cuenta lo aprendido en las clases de grafos, sabíamos que teníamos que hacer uso del Algoritmo de Dijkstra para poder hallar el camino más corto a cada vértice del grafo dado uno en particular.

Por lo que decidimos probar con la misma implementación vista en clase, realizando las modificaciones pertinentes para adaptarlo a nuestro problema y así poder obtener un mapa con las distancias más cortas hacia cada pueblo desde el origen.

Luego, buscamos una manera de “discriminar” la información innecesaria que resultaba de este mapa, ya que, como se dijo anteriormente, nos brindaba el camino más corto hacia todos los pueblos del mapa. Para esto, resolvimos desarrollar un método llamado *‘calcularRecorrido()’*, que recibe dicho mapa y se encarga de devolver otro que sólo contenga las direcciones relevantes para el viaje.

Una vez resuelto ese problema, sólo nos quedaba realizar el recorrido en cuestión. Con este fin, desarrollamos el método *‘avanzarAlSiguientePueblo()’*, cuya función es la de interactuar con el siguiente pueblo indicado por la ruta. Devolviendo un valor lógico que indica si nuestro ejército sigue en pie o no.

Finalmente, con todas las piezas listas, implementamos el método *‘realizarPrediccionDeAventura()’*, que haciendo uso de todo lo anteriormente desarrollado se encarga de indicarnos la factibilidad de la travesía.

# Reconquistando la tierra de fantasía

## Conclusión

### Conclusión

Durante la realización de este proyecto se emplearon varios conceptos vistos durante la cursada como TDD (Test-driven Development), herencia y polimorfismo, programación orientada a objetos, lectura de archivos, manejo de excepciones, estructuras, patrones de diseño, grafos y algoritmos de recorrido mínimo. Siendo esta una forma práctica de aplicar e integrar dichos conocimientos.

También fue un desafío organizar las tareas y ponerse de acuerdo con respecto a las implementaciones, ya que nos encontramos con diferencias a la hora de encarar las diversas problemáticas en las distintas etapas del proyecto.

La virtualidad a su vez nos impulsó a utilizar herramientas como Git, GitHub, GitKraken y herramientas de visualización gráfica para diseñar los UML y poder comprender gráficamente como sería el funcionamiento de ciertos escenarios. Una vez incorporadas dichas herramientas, nos facilitaron y permitieron dinamizar el trabajo en equipo, acercándonos más a una dinámica de trabajo profesional.

## Diagrama UML

