# [75.07 / 95.02]

# Algoritmos y programación III

*Trabajo práctico 2: AlgoDefense*

(trabajo grupal)

Estudiantes:

| Nombre | Padrón | Mail |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

Tutor:

Nota Final:

# Índice

# 1. Objetivo

Desarrollar una aplicación de manera grupal aplicando todos los conceptos vistos en el curso, utilizando un lenguaje de tipado estático (Java) con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua.

# 2. Consigna general

Desarrollar la aplicación completa, incluyendo el modelo de clases, sonidos e interfaz gráfica. La aplicación deberá ser acompañada por pruebas unitarias e integrales y documentación de diseño.

# 3. Especificación de la aplicación a desarrollar (Parte 1 / 2)

AlgoDefense es un juego similar al famoso TowerDefense pero no es en tiempo real sino por turnos. El mismo es un juego de estrategia y se basa en la construcción de defensas

que impidan que los enemigos lleguen hasta el jugador.

### *Los enemigos*

- Hormiga:
  - Tiene 1 de velocidad
  - Causa 1 punto de daño al llegar a la meta
  - Tiene 1 punto de energia
  - Otorga 1 crédito al jugador cada vez que 1 hormiga es destruida, pero cuando se matan más de 10 hormigas, cada nueva hormiga otorga 2 al jugador.

- Araña:
  - Tiene 2 de velocidad
  - Causa 2 puntos de daño al llegar a la meta
  - Tiene 2 puntos de energía
  - Otorga créditos de forma aleatoria al jugador al ser destruída. Los créditos que otorgan pertenecen al rango 0..10

### *Gestión de recursos*

El jugador comienza con 100 créditos que le permitirán construir sus defensas. Luego a medida que éstas eliminen enemigos el jugador irá recibiendo créditos según corresponda a la unidad eliminada.

### *Vida del jugador*

El jugador comienza con 20 puntos de vida. A medida que cada unidad llega a la meta se le irá descontando puntos de vida, si llega a 0 pierde.

## Defensas

Cada torre puede atacar a 1 enemigo por turno y el mismo debe estar dentro del radio de ataque de la torre.

| Nombre | Costo | Tiempo de construcción | Rango de ataque | Daño |
|---|---|---|---|---|
| Torre Blanca | 10 créditos | 1 turno | 3 | 1 |
| Torre Plateada | 20 créditos | 2 turnos | 5 | 2 |

## Turnos

El juego es por turnos, el jugador elige si quiere construir una nueva defensa o pasar de turno. En cada paso de turno las torres disparan según sus capacidades y los enemigos avanzan según sus capacidades también.

## Jugador

La aplicación se juega con 1 jugador. Al iniciar el juego el sistema debe consultar al usuario su nombre con la validaciones que correspondan.

Validaciones:

- Nombre de jugador debe contener por lo menos 6 caracteres.

## Comienzo y fin de la partida

La partida comienza con los enemigos desfilando por la senda permitida. El usuario deberá ir armando sus defensas. Si logra sobrevivir cuando se hayan desfilado todos los enemigos, habrá ganado la partida, de lo contrario perderá.
La cantidad total de enemigos, el orden y la cantidad en que aparecen en cada turno serán oportunamente entregados por la cátedra en formato JSON.

## Mapa

El mapa es el lugar donde se lleva a cabo el juego. Hay diferentes tipos de parcelas:

- **Pasarela**: Es la parcela por donde desfilan los enemigos. Solo sobre ellas se pueden mover. No es posible construir ninguna torre de defensa sobre una pasarela. Puede haber **N** enemigos ubicados en la misma posicion, es decir en una posicion dada de una pasarela puede haber 0, 1, 2, … enemigos al mismo tiempo.
    - Pasarela largada: Es la parcela por donde si o si empiezan a aparecer los enemigos.
    - Pasarela meta: Es la parcela por donde terminan, si llegan hasta allí le proporcionan un daño al jugador.

- **Rocoso**: Es una superficie en la que no es posible construir defensas.
- **Tierra**: Es una superficie apta para la construcción de defensas.

La disposición de parcelas del mapa también vendrá dada en formato JSON.

# 4. Interfaz gráfica

La interacción entre el usuario y la aplicación deberá ser mediante una interfaz gráfica intuitiva. Consistirá en una aplicación de escritorio utilizando JavaFX y se pondrá mucho énfasis y se evaluará como parte de la consigna su usabilidad. (*en el anexo I se explicarán algunas buenas prácticas para armar la interfaz gráfica de usuario o GUI*)

# 5. Herramientas

1. JDK (Java Development Kit): Versión 1.8 o superior.
2. JavaFX
3. JUnit 5 y Mockito: Frameworks de pruebas unitarias para Java.
4. IDE (Entorno de desarrollo integrado): Su uso es opcional y cada integrante del grupo puede utilizar uno distinto o incluso el editor de texto que más le guste. Lo importante es que el repositorio de las entregas no contenga ningún archivo de ningún IDE y que la construccnos de los IDEs más populares son:ión y ejecución de la aplicación sea totalmente independiente del entorno de desarrollo.
    a. Eclipse
    b. IntelliJ
    c. Netbeans
5. Herramienta de construcción: Se deberán incluir todos los archivos XML necesarios para la compilación y construcción automatizada de la aplicación. El informe deberá contener instrucciones acerca de los comandos necesarios (preferentemente también en el archivo README.md del repositorio). Puede usarse Maven o Apache Ant con Ivy.
6. Repositorio remoto: Todas las entregas deberán ser subidas a un repositorio único en GitHub para todo el grupo en donde quedarán registrados los aportes de cada miembro. El repositorio puede ser público o privado. En caso de ser privado debe agregarse al docente corrector como colaborador del repositorio.
7. Git: Herramienta de control de versiones
8. Herramienta de integración continua: Deberá estar configurada de manera tal que cada *commit* dispare la compilación, construcción y ejecución de las pruebas unitarias automáticamente. Algunas de las más populares son:
    a. Travis-CI
    b. Jenkins
    c. Circle-CI
    d. GitHub Actions (recomendado)

Se recomienda basarse en la estructura del proyecto base armado por la cátedra.

# 6. Entregables

Para cada entrega se deberá subir lo siguiente al repositorio:

1. Código fuente de la aplicación completa, incluyendo también: código de la prueba, archivos de recursos.
2. Script para compilación y ejecución (Ant o Maven).
3. Informe, acorde a lo especificado en este documento (en las primeras entregas se podrá incluir solamente un enlace a Overleaf o a Google Docs en donde confeccionen el informe e incluir el archivo PDF solamente en la entrega final).

No se deberá incluir ningún archivo compilado (formato .class) ni tampoco aquellos propios de algún IDE (por ejemplo .idea). Tampoco se deberá incluir archivos de diagramas UML propios de alguna herramienta. Todos los diagramas deben ser exportados como imágenes de manera tal que sea transparente la herramienta que hayan utilizado para crearlos.

# 7. Formas de entrega

Habrá 5 entregas formales que tendrán una calificación de APROBADO o NO APROBADO en el momento de la entrega. Además, se contará con una entrega 0 preliminares.

Aquel grupo que acumule 2 no aprobados, quedará automáticamente desaprobado con la consiguiente pérdida de regularidad en la materia de todos los integrantes del grupo. *En cada entrega se deberá incluir el informe actualizado*.

# 8. Evaluación

El día de cada entrega, cada ayudante convocará a los integrantes de su grupo, solicitará el informe correspondiente e iniciará la corrección mediante una entrevista grupal. Es imprescindible la presencia de todos los integrantes del grupo el día de cada corrección.

Se evaluará el trabajo grupal y a cada integrante en forma individual. El objetivo de esto es comprender la dinámica de trabajo del equipo y los roles que ha desempeñado cada integrante del grupo. Para que el alumno apruebe el trabajo práctico debe estar aprobado en los dos aspectos: grupal e individual (se revisarán los commits de cada integrante en el repositorio).

Dentro de los ítems a chequear el ayudante evaluará aspectos formales (como ser la forma de presentación del informe), aspectos funcionales: que se resuelva el problema planteado y aspectos operativos: que el TP funcione integrado.

# 9. Entregables para cada fecha de entrega

Cada entrega consta de las pruebas + el código que hace pasar dichas pruebas.

## Entrega 0 (Semana 11 del calendario)

- Planteo de modelo tentativo con diagramas de clases.
- Repositorio de código creado según se explica aquí.
- Servidor de integración continua configurado.
- Al menos un commit realizado por cada integrante, actualizando el README.md

## Entrega 1 (Semana 12 del calendario - 30 de Mayo / 1 de Junio)

Pruebas (sin interfaz gráfica)

Caso de uso 1

- Verificar que jugador empieza con la vida y los créditos correspondientes.

Caso de uso 2

- Verificar que cada defensa tarde en construirse lo que dice que tarda y que recién están “operativas” cuando ya se terminaron de construir.

Caso de uso 3

- Verificar que se disponga de credito para realizar las construcciones.

Caso de uso 4

- Verificar solo se pueda construir defensas sobre tierra (y verificar lo contrario)

Caso de uso 5

- Verificar que las defensas ataquen dentro del rango esperado (y verificar lo contrario)

Caso de uso 6

- Verificar que las unidades enemigas son dañadas acorde al ataque recibido.

Caso de uso 7

- Verificar que las unidades enemigas solo se muevan por la parcela autorizada.

Caso de uso 8

- Verificar que al destruir una unidad enemiga, el jugador cobra el crédito que le corresponde.

Caso de uso 9

- Verificar que al pasar un turno las unidades enemigas se hayan movido según sus capacidades.

Caso de uso 10

- Verificar que al eliminar todas la unidades enemigas el jugador gana el juego

Caso de uso 11

- Verificar que sin eliminar todas la unidades enemigas, pero las pocas que llegaron a la meta no alcanzan para matar al jugador, este también gana el juego.

Caso de uso 12

- Verificar que si las unidades enemigas llegadas a la meta matan al jugador, este pierde el juego

## Entrega 2 (Semana 13 del calendario, 6 / 8 de Junio)

Pruebas (sin interfaz gráfica). Refactor de todas aquellas pruebas de la entrega 1 que hayan sido puros getters para verificar valores y no hayan verificado **comportamiento**.

Caso de uso 13

- Verificar el formato valido del JSON de enemigos.

Caso de uso 14

- Verificar el formato valido del JSON del mapa.

Caso de uso 15

- Verificar la lectura y posterior conversión a unidades del modelo de dominio del JSON de enemigos

Caso de uso 16

- Verificar la lectura y posterior conversión a unidades del modelo de dominio del JSON del mapa.

Caso de uso 17

- Verificar que el juego se crea acorde a ambos JSON.

Caso de uso 18

- Simular y verificar que el jugador gana una partida.

Caso de uso 19

- Siuilar y verificar que el jugador pierde una partida.

Caso de uso 20

- Verificar el sistema de log a utilizar necesario para la entrega 3. El log puede ser una implementación propia, casera y simple del grupo o utilizar alguna librería.

## *Entrega 3 (Semana 14 del calendario, 13 / 15 de Junio)*

**Interfaz gráfica inicial básica**: Pantallas iniciales, comienzo del juego y visualización del mapa e interfaz de usuario básica.

Finalización del modelo junto a todas las pruebas unitarias y de integración. Se tiene que poder jugar una partida completa a través de las pruebas e ir viendo en la consola todos los eventos que están ocurriendo, por ejemplo:

```
...
Torre Blanca ataca una hormiga en la posicion (X,Y)
Araña llega a la meta, produce X daño al jugador
Jugador Construye Torre Plateada en posicion (X,Y)
...
Jugador Gana la Partida
```

## Entrega 4 (Semana 15 del calendario, 20 / 22 de Junio)

Interfaz gráfica completa acorde al enunciado.

## Entrega 5 - Final: (Semana 16 del calendario, 27 / 29 de Junio)

Trabajo Práctico completo funcionando, con interfaz gráfica final, **sonidos** e informe completo.

Tiempo total de desarrollo del trabajo práctico:

6 semanas

# 10. Informe

El informe deberá estar subdividido en las siguientes secciones:

## Supuestos

Documentar todos los supuestos hechos sobre el enunciado. Asegurarse de validar con los docentes.

## Diagramas de clases

Varios diagramas de clases, mostrando la relación estática entre las clases. Pueden agregar todo el texto necesario para aclarar y explicar su diseño de manera tal que el modelo logre comunicarse de manera efectiva.

## Diagramas de secuencia

Varios diagramas de secuencia, mostrando la relación dinámica entre distintos objetos planteando una gran cantidad de escenarios que contemplen las secuencias más interesantes del modelo.

## Diagrama de paquetes

Incluir un diagrama de paquetes UML para mostrar el acoplamiento de su trabajo.

## Diagramas de estado

Incluir diagramas de estados, mostrando tanto los estados como las distintas transiciones para varias entidades del modelo.

## Detalles de implementación

Deben detallar/explicar qué estrategias utilizaron para resolver todos los puntos más conflictivos del trabajo práctico. Justificar el uso de herencia vs. delegación, mencionar que principio de diseño aplicaron en qué caso y mencionar qué patrones de diseño fueron utilizados y por qué motivos.

> IMPORTANTE
>
> No describir el concepto de herencia, delegación, principio de diseño o patrón de diseño. Solo justificar su utilización.

## Excepciones

Explicar las excepciones creadas, con qué fin fueron creadas y cómo y dónde se las atrapa explicando qué acciones se toman al respecto una vez capturadas.

# Anexo 0: ¿Cómo empezar?

## Requisitos, análisis

¿Cómo se empieza? La respuesta es lápiz y papel. No código!.

1. Entiendan el dominio del problema. Definir y utilizar un lenguaje común que todo el equipo entiende y comparte. Ej.: Si hablamos de “X entidad”, todos entienden que es algo ... Si los conceptos son ambiguos nunca podrán crear un modelo congruente.
2. Compartan entre el grupo, pidan opiniones y refinen la idea en papel antes de sentarse a programar. Pueden escribir en ayudante-virtual si tienen dudas

# Anexo I: Buenas prácticas en la interfaz gráfica

El objetivo de esta sección es recopilar algunas recomendaciones en la creación de interfaces gráficas de usuario o GUI. La idea no es exigir un diseño refinado y prolijo, sino que pueda ser usado por el grupo de docentes de Algo3 sin impedimentos “lo mejor posible”.

### Prototipo

Venimos escribiendo código, integrales y UML todo el cuatrimestre. Me están pidiendo una interfaz gráfica. ¿Cómo se empieza? La respuesta es lápiz y papel. No código!.

1. Armen un dibujo o prototipo en papel (pueden usar google docs o cualquier herramienta también) de todas las “pantallas”. No tiene que ser perfecto.
2. Compartan entre el grupo, pidan opiniones y refinen la idea en papel antes de sentarse a programar. Pueden escribir en ayudante-virtual si tienen dudas

### JavaFX

Entender cómo funciona JavaFX es clave para implementar la GUI correctamente. No subestimen el tiempo que lleva implementar y modificar la UI o interfaz de usuario. Lean todo lo que ofrece y las buenas prácticas de la tecnología. Hay plugins específicos para el IDE, pero por más que usen herramientas WYSIWYG, siempre conviene entender la API para mejorar el código autogenerado que casi nunca es óptimo.

## Recomendaciones visuales

### Tamaño de elementos

- Se recomienda un tamaño de tipografía de al menos a 10 puntos al mayor contraste negro contra blanco para asegurar la legibilidad, o bien 12 puntos.
- Para los botones o elementos interactivos, el tamaño mínimo del área debería ser de 18x18.

- Extra: Los elementos más importantes deberían ser más grandes y estar en posiciones más accesibles (poner ejemplos)

### Contraste

- Aseguren que el texto y los elementos tengan buen contraste y se puedan leer bien
- Se sugiere un contraste cercano a 3 entre textos y fondos para texto grande e imágenes.
- Herramientas para verificar contraste: https://colourcontrast.cc/ https://contrast-grid.eightshapes.com

### Uso del color

- La recomendación es no utilizar más de 3 colores para la UI y los elementos
- En el enunciado se ejemplifica con una paleta accesible, pero pueden usar cualquiera para las fichas
- Accesibilidad: Si usan para las fichas rojo y verde, o verde y azul, aseguren que las personas con daltonismo puedan distinguirlas usando letras o símbolos sobre las mismas.
- Dudas eligiendo paletas? https://color.adobe.com

### Tipografía

- No se recomienda usar más de 2 tipografías para toda la aplicación
- Asegurarse de que esas tipografías se exporten correctamente en en TP
- Evitar tipografías “artísticas” para texto, menú y botones, ya que dificultan la lectura

## Recomendaciones de interacción

### Manejo de errores

- No escalar excepciones a la GUI. Es un No absoluto. Enviar a la consola.
- Si muestran errores al usuario, el mensaje de error debe estar escrito sin jerga técnica y permitir al usuario continuar y entender lo que está pasando
- Siempre optar por validar y prevenir errores, a dejar que el usuario ejecute la acción y falle.

### Confirmaciones

- Antes de cerrar o ejecutar cualquier operación terminal, una buena práctica es pedir confirmación al usuario (para el alcance del tp no sería necesario)

### Visibilidad del estado y otros

- Mostrar el estado en el cual está el juego. Siempre debería estar accesible
- Permitir al usuario terminar o cerrar en cualquier momento