# Enunciado

# Parcial 2c 2023 - Ingeniería de Software 1 - FIUBA

Nuestro cliente quiere resolver ciertas desventajas que tiene un combatiente al momento de atacar a otro. Para ello, debemos **extender** nuestro **modelo de combatientes fantásticos**.

Por ejemplo, si un combatiente está en terreno embarrado tiene desventaja. Si está confundido, ataca peor. A estas desventajas el cliente las denomina penalizadores, cuando un combatiente ataca con un penalizador inflige menos daño .

Se requiere modelar los siguientes penalizadores:

* **Terreno embarrado**: es unapenalización **por terreno,** inflige **1** menos **de daño**.
* **Confundido**: es una penalización **mental,** inflige **2** menos **de daño**.
* **Aturdido**: es una penalización **mental**, inflige **2** menos **de daño**.
* **Cansado**: es una penalización **física**, inflige **2** menos **de daño**.
* **Exhausto**: es una penalización **física**, inflige **4** menos **de daño**.

En un ataque pueden intervenir **varios** de estos **penalizadores**, pero no todos los penalizadores son “apilables”. Si un combatiente está cansado y exhausto sólo aplica el peor.

Esto sucede porque cada penalizador afecta a un aspecto del ataque: terreno, mente, cuerpo, etc. Es decir, **si los penalizadores afectan al mismo aspecto, solamente se aplica el peor, en caso contrario se suman**.

**Ejemplo 1:** si se tienen los siguientes penalizadores

* + Terreno embarrado (terreno / -1)
  + Confundido (mental / -2)
  + Aturdido (mental / -2)

El ataque produce 3 menos de daño ya que solo se toma un solo penalizador mental.

**Ejemplo 2:** si se tienen los siguientes penalizadores

* + Terreno embarrado (terreno / -1)
  + Confundido (mental / -2)
  + Cansado (fìsico / -2)
  + Exhausto (fìsico / -4)

El ataque produce 7 menos de daño ya que solo se toma un penalizador físico, en este caso estar exhausto.

### Extensibilidad

Nuestro cliente nos ha comentado que necesita agregar nuevos penalizadores por lo que es indispensable que **nuestro modelo sea extensible.** Nos ha hablado sobre dos ejes de cambios:

* Nuevos penalizadores de los aspectos mencionados (terreno, mental y físico).
* Nuevos penalizadores sobre nuevos aspectos.

Y nos ha señalado que el primero es más importante que el segundo. Pero que requiere de ambos.

Algunos ejemplos de futuros penalizadores podrían ser:

* -1 mental por estar desmoralizado
* -1 por maldición mágica
* -3 por maldición mágica

**Importante:** No hace falta modelar estos penalizadores futuros, simplemente tener un modelo que permita fácilmente agregar un nuevo penalizador.

## Modalidad de trabajo

Nuestra tarea es **modelar lo pedido mediante TDD y siguiendo las heurísticas de diseño** vistas durante toda la cursada de la materia.

En su modelo final **deben pasar todos los tests**. Tanto los que agregue en su recorrido en TDD como los ya existentes en el código inicial.

**Ayudas:**

Pueden cargar como código inicial *IngSoftI-2023-2C-Parcial-CombatientesFantasticos.Inicial.st*, ahí encontrarán un código muy similar a la solución del ejercicio de combatientes fantásticos. Deben modificar el protocolo del combatiente para que contemple la funcionalidad pedida. Tener en cuenta que se debe preservar el funcionamiento actual en los mensajes *“atacar”* y *“atacarA:”*, es decir estos mensajes realizan ataques sin penalizadores.

Prioricen la calidad del trabajo en este orden:

1. TDD (tener tests en un orden y granularidad acorde a TDD).
2. Modelo (nombres, no repetir código, claridad, etc).
3. Resolver las restricciones de extensibilidad indicadas en la segunda parte.
4. Correctitud de tests (repetición de código, claridad, etc).
5. Nombre de los tests (nombres claros de los tests)

Entrega del examen

1. Tienen tiempo para entregarlo hasta las 22hs. No serán tomadas en cuenta las entregas posteriores a ese horario (estricto).

2. Recuerden grabar la imagen con frecuencia e ir haciendo file-outs de lo que vayan haciendo. No se aceptarán explicaciones del estilo “se me colgó la máquina” o “perdí todo” como motivos de no entrega.

3. Entregar el fileout de la categoría “IngSoftI-2023-2C-Parcial-CombatientesFantasticos.st”, que debe incluir toda la solución (modelo y tests).

4. Entregar también el archivo que se llama CuisUniversity-nnnn.user.changes.

5. **Probar que el archivo generado** en el paso 1 se cargue correctamente en una imagen “limpia” (o sea, sin la solución que crearon) y que todo funcione correctamente. Verifiquen que los nombres de los colaboradores sean los correctos (y no temp1). Esto es fundamental para que no haya problemas de que falten clases/métodos/objetos en la entrega.

6. Realizar la entrega enviando mail **A LA LISTA DE DOCENTES**

[**fiuba-ingsoft1-doc@googlegroups.com**](mailto:fiuba-ingsoft1-doc@googlegroups.com)con el Asunto: "Nro Padrón: nnnn - Solución del Parcial 2023 2c"

7. Al enviar la solución deben esperar a recibir una **confirmación de recepción ANTES de retirarse** del aula. Recién una vez recibida la confirmación, puede retirarse.

---------------------------------------------------------------------------------------------------

# Enunciado V3 - Original bonificadores

# Primer Parcial 2c 2023 - Ingeniería de Software 1 - FIUBA

Debemos extender nuestro modelo. Nuestro cliente quiere resolver ciertas ventajas que tiene un combatiente al momento de atacar a otro. Por ejemplo, si un combatiente está en terreno elevado tiene ventaja. Si está lleno de furia, ataca mejor. A estas ventajas el cliente las denomina bonificadores, cuando un combatiente ataca con un bonificador inflige más daño .

Se requieren los siguientes bonificadores:

* **Terreno elevado**: es unabonificación **por altura** e inflige **1** más **de daño**.
* **Estado de furia**: es una bonificación **de moral** e inflige **2** más **de daño**.
* **Inspirado**: es una bonificación **de moral** e inflige **2** más **de daño**.
* **Ataque de flanco**: es una bonificación **por posición**, cuando el atacante arremete desde el costado. Este bonificador hace **2** más **de daño**.
* **Ataque por retaguardia**: es una bonificación **por posición**, cuando el atacante arremete desde atrás. Este bonificador hace **4** más **de daño**.

En un ataque pueden intervenir **varios** de estos **bonificadores**, pero no todos los bonificadores son “apilables”. Si un combatiente está inspirado y con furia sólo aplica el mejor.

Esto sucede porque cada bonificador afecta a un aspecto del ataque: posición, moral, altura, etc. Es decir, **si los bonificadores afectan al mismo aspecto, solamente se aplica el mejor, en caso contrario se suman**.

**Ejemplo 1:** si se tienen los siguientes bonificadores

* + Terreno elevado (altura / +1)
  + Por furia (moral / +2)
  + Por inspirado (moral / +2)

El ataque produce 3 más de daño ya que solo se toma un bonificador por moral.

**Ejemplo 2:** si se tienen los siguientes bonificadores

* + Terreno elevado (altura / +1)
  + Inspirado (moral / +2)
  + Ataque de flanco (posición / +2)
  + Ataque por retaguardia (posición / +4)

El ataque produce 7 más de daño ya que solo se toma un bonificador por posición, en este caso el ataque por retaguardia.

## Segunda parte

Nuestro cliente nos ha comentado que necesita agregar nuevos bonificadores por lo que es indispensable que **nuestro modelo sea extensible.** Nos ha hablado sobre dos ejes de cambios:

* Nuevos bonificadores de los aspectos mencionados (altura, moral, posición).
* Nuevos bonificadores sobre nuevos aspectos.

Y nos ha señalado que el primero es más importante que el segundo. Pero que requiere de ambos.

Algunos ejemplos de nuevos bonificadores pueden ser:

* +1 por moral por estar bien descansado y nutrido
* +1 por velocidad
* +3 por velocidad

**Importante:** No hace falta modelar estos modificadores de ejemplo (obviamente si los mencionados para la primera parte :)), simplemente tener un modelo que permita fácilmente agregar un nuevo modificador.

**Importante:** en el código inicial exploramos otro bonificador de ataque que es *bonificadorDeFuerza*, por ahora no hace falta integrarlo con la nueva funcionalidad pedida, ni modificar nada de esto.

## Modalidad de trabajo

Nuestra tarea es **modelar lo pedido mediante TDD y siguiendo las heurísticas de diseño** vistas durante toda la cursada de la materia.

En su modelo final **deben pasar todos los tests**. Tanto los que agregue en su recorrido en TDD como los ya existentes en el código inicial.

**Ayudas:**

Pueden cargar como código inicial *IngSoftI-2023-2C-1Parcial-AtaquesFantasticos.Inicial.st*, ahí encontrarán un código muy similar a la solución del parcial anterior. Deben modificar el protocolo del combatiente para que contemple la funcionalidad pedida. Tener en cuenta que se debe preservar el funcionamiento actual en los mensajes *“atacar”* y *“atacarA:”*, es decir estos mensajes realizan ataques sin bonificadores.

Prioricen la calidad del trabajo en este orden:

1. TDD (tener tests en un orden y granularidad acorde a TDD).
2. Modelo (nombres, no repetir código, claridad, etc).
3. Resolver las restricciones de extensibilidad indicadas en la segunda parte.
4. Correctitud de tests (repetición de código, claridad, etc).
5. Nombre de los tests (nombres claros de los tests)

Entrega del examen

1. Tienen tiempo para entregarlo hasta las XXhs. No serán tomadas en cuenta las entregas posteriores a ese horario (estricto).

2. Recuerden grabar la imagen con frecuencia e ir haciendo file-outs de lo que vayan haciendo. No se aceptarán explicaciones del estilo “se me colgó la máquina” o “perdí todo” como motivos de no entrega.

3. Entregar el fileout de la categoría “IngSoftI-2023-2C-1Parcial-AtaquesFantasticos.st”, que debe incluir toda la solución (modelo y tests).

4. Entregar también el archivo que se llama CuisUniversity-nnnn.user.changes.

5. **Probar que el archivo generado** en el paso 1 se cargue correctamente en una imagen “limpia” (o sea, sin la solución que crearon) y que todo funcione correctamente. Verifiquen que los nombres de los colaboradores sean los correctos (y no temp1). Esto es fundamental para que no haya problemas de que falten clases/métodos/objetos en la entrega.

6. Realizar la entrega enviando mail **A LA LISTA DE DOCENTES**

[**fiuba-ingsoft1-doc@googlegroups.com**](mailto:fiuba-ingsoft1-doc@googlegroups.com)con el Asunto: "Nro Padrón: nnnn - Solución del 2do Parcial 2023 2c"

7. Al enviar la solución deben esperar a recibir una **confirmación de recepción ANTES de retirarse** del aula. Recién una vez recibida la confirmación, puede retirarse.

---------------------------------------------------------------------------------------------------

## Cambios para el código inicial

### Renombrar “usarXXX” a “equiparXXX”

El nombre “usar” es ambiguo. Se puede confundir con “usar un arma es utilizarla, es decir, atacar con ella”.

La misma categoría de “equipo” ya nos indicaba que son cosas que equipamos.

### Agregar tests usando al Orquestador

Tests donde se crean combates entre varios combatientes, se usan varios modificadores (ningún caso obvio y directo de TDD) y se compruebe la duración esperada hasta que haya un ganador.

Esto sirve para entender cómo se va a usar y que la funcionalidad es la esperada.

Pero no ayuda en el recorrido de TDD, ni en el nombrado ni en descubrir el protocolo.

Estos tests estarían incompletos y con un método que “hace atacar a un combatiente en condiciones de niebla, humo y terreno elevado”. Por dar un ejemplo.

### Agregar nuevo test08

Falta agregar un “hermano” del test test08CuandoUnCombatienteEstaVencidoNoAtaca que envie el mensaje atacarA:conBonificadores: en el caso de que el combatiente atacante ya esté fuera de combate. Porque hay dos IF que contemplan no atacar estando fuera de combate. Falta ejercitar, cubrir, ese otro IF.

# Enunciado V2 - Descartado

Queremos modelar las ventajas que tiene un combatiente al atacar a otro.

Por ejemplo, si un combatiente está en terreno elevado tiene ventaja. Si un combatiente está lleno de furia, ataca mejor.

Al mismo tiempo, no todos los bonificadores son “apilables”. Si un combatiente está inspirado y con furia sólo aplica el mejor.

Esto sucede porque cada bonificador afecta a un aspecto del ataque: posición, moral, velocidad, tamaño, etc.

Queremos modelar:

* La bonificación moral (+2) que te da estar en estado de furia
* La bonificación moral (+2) que te da estar inspirado
* La bonificación por posición (+1) que te da estar en terreno elevado
* La bonificación por posición (+2) que te da atacar de flanco, desde el costado
* La bonificación por posición (+4) que te da atacar por retaguardia, desde atrás

Queremos entonces modificar el protocolo del Combatiente para que considere estas cuestiones. Para que cuando le enviemos a un combatiente el mensaje “atacar”, indicarle esas circunstancias.

Los mensajes actuales de “atacar” y “atacarA:” debemos preservarlos. Y suponer que cuando atacamos con esos mensajes no hay bonificadores.

# Enunciado V1- Descartado

Queremos modelar cómo ciertas circunstancias modifican los ataques.

Por ejemplo, si un combatiente está en terreno elevado tiene ventaja. Si un combatiente tiene poca visibilidad porque hay niebla, atacá peor. Si está lleno de furia, ataca mejor.

Queremos entonces modificar el protocolo del Combatiente para que considere estas cuestiones. Para que cuando le enviemos a un combatiente el mensaje “atacar”, indicarle esas circunstancias.

Al mismo tiempo, no todas las circunstancias son “apilables”. Si hay humo y hay niebla, no se acumulan las penalizaciones, se usa la peor. Si un combatiente está inspirado y con furia, lo mismo, sólo aplica la mejor.

Esto sucede porque cada circunstancia afecta a un aspecto particular del ataque: posición, visibilidad, moral, etc.

Queremos modelar:

* La bonificación por posición (+1) que te da estar en terreno elevado
* La bonificación moral (+2) que te da estar en estado de furia
* La bonificación moral (+2) que te da estar inspirado
* La penalización por visibilidad (-1) que se sufre atacando en niebla
* La penalización por visibilidad (-2) que se sufre atacando humo

## Tip

Cuando hagan que un combatiente ataque a otro, elijan un arma que haga 4 o más de daño. No usen los puños. Necesitan infligir ese mínimo

Si hay modificadores repetidos, no pasa nada. Termina valiendo uno de ellos.

# Desafíos y criterios

* TDD. Debería haber como mínimo:
  + 5 tests. Uno por cada modificador. Que pruebe solo ese bonificador y el valor extra que da al daño infligido
  + 1 test para probar la suma de dos bonificadores de distinto aspecto
  + 1 test para probar que dos bonificadores de mismo tipo no se suman
  + 1 test para probar que dos bonificadores de mismo tipo y diferente valor se usa el más alto. Idealmente debería haber dos. Pero 1 es el mínimo.
* Codigo repetido.
  + Los tests hacen todos lo mismo. Se puede capturar en una única abstracción. Solo cambian dos cosas:
    - Bonificadores aplicados
    - Puntos de vida esperados
  + (extra) Una abstracción superadora en los tests es un método que no testea los puntos de vida finales, esperados, sino “el delta”. La cantidad de puntos de vida que deberían cambiar entre los que inflige el arma y los que efectivamente se descuentan. De esa forma el test se desacopla de qué arma se utiliza para hacer el ataque.
  + #atacar y #atacarA: deberian llamar a #atacarConBonificadores: y #atacarA:conBonificadores: respectivamente con una colección vacía de bonificadores. Con esto se evita código repetido.
* Nombres
  + Las clases a modelar son bonificadores, no modificadores, no bonuses u otra cosa. Nombre dado en el enunciado.
  + Lo que junta a los bonificadores y hace que no se sumen es su **aspecto**. Ese nombre está dado en el enunciado.
  + Los aspectos a modelar fueron nombrados en el enunciado. Son:
    - Altura
    - Moral
    - Posicion
  + El objeto que guarda el mejor modificador para cada aspecto durante la iteración sobre los bonificadores aplicados a un ataque
  + Abstracción y nombrado del calculo del daño. Calcular los bonificadores está separado en un método propio? O en algo mejor?
* Reificar
  + El conjunto de bonificadores
    - El objeto que guarda el mejor modificador para cada aspecto durante la iteracion o el objeto (method object) que hace todo el cálculo
    - No parece ser necesario crear un method object para esta parte del cálculo. No es tan grande ni la complejidad ni los métodos auxiliares.
* Segunda parte - Escalabilidade del modelo
  + Bonificadores como única clase con aspecto y valor como parámetros
    - Detectar y sacar codigo repetido. Remover clases anémicas.
  + Inicialización del objeto que guarda el mejor bonificador por aspecto
    - Protocolo de colecciones: asSet, collect, Dictionary>>withAll:, ->
  + Suma de bonificadores de cada aspecto para dar el total
    - Protocolo de colecciones: values; sum:ifEmpty:

# Sugerencias para las correcciones :

No pueden aprobar si:

* no hicieron TDD
* no llegaron a quitar los IFs en el cálculo de los bonificadores

## TDD:

**Importante:** no tomar literal esto, intentar analizar el recorrido que hizo el alumne.

**Camino regular:** (2,25) un test para cada bonificador, un test para más de un bonificador donde se suman (aspectos diferentes), un test para mas de un bonificador donde se aplica el mayor.

**Camino bueno**: (3) mismos casos que el camino regular, y además hicieron el caso sin bonificadores, caso donde el daño extra son todos iguales (caso previo al mayor)

**Camino muy bueno:** (3,5) mismos casos que el camino bueno, pero además tiene los mismo daño donde no importa el orden y mismo aspecto donde se aplica el mayor sin importar el orden.

Cosas a evaluar: cada test debe estar testeando una única cosa, debe ser simple de entender. Debe ser el test más simple posible.

Modelo:

**Regular**: (1,5) resuelve el cálculo sin IFs

**Bueno**: (3) Regular y ademas el modelo es extensible para nuevos bonificadores para los aspectos del enunciado.

**Muy bueno**: (4) bueno y ademas el modelo es extensible para bonificadores con nuevos aspectos

TDD regular y modelo regular => 3,75 (mas codigo repetido)

TDD bueno y modelo regular y codigo repetido regular => 4/5