Tarea 1 - Batallas medievales

Profesor: Alexandre Bergel
Auxiliar: Juan-Pablo Silva

En esta tarea se le pedirá que implemente un modelo de peleas inspirado en los clásicos mundos de fantasía. Su aplicación debe estar debidamente testeada y documentada y debe utilizar buenas prácticas de diseño. Especificaciones serán dadas a continuación en este documento.

Requisitos

Tendremos 2 tipo de entidades, entidades que pueden recibir ataques (*Attackable*) y entidades que pueden atacar (*Attacker*). Las entidades *Attacker* pueden atacar y también recibir ataques. Dentro de las entidades que pueden atacar, tenemos humanos, goblins (*Goblin*), golems de hielo (*IceGolem*) y no-muertos (*Undead*). Dentro de los humanos hay profesiones especializadas como caballeros (*Knight*), hechiceros de fuego (*FireMage*) y clérigos (*Priest*). Los objetos que solo pueden recibir ataques (*Attackable*) son rocas (*Rock*) y árboles frutales (*FruitTree*).

Como en la edad medieval se pasaba en guerra, estas entidades se peleaban entre si, pero ciertas entidades tienen ventajas sobre otras. Por ejemplo, el caballero no golpea a los clérigos, pero golpea con un 50% extra a los magos ya que estos no visten armaduras. En términos generales, los clérigos no golpean a nadie, salvo cuando pelean con no-muertos, en este caso golpean 5 veces sus puntos de ataque. Los hechiceros en general pegan el doble de su poder de ataque, pero hay ocasiones en que dañan menos. Los golems también dañan el doble en la mayoría de las interacciones. Detalles respecto a cómo interactúa cada una de las distintas entidades se pueden ver en la tabla 1.

Una interacción especial ocurre cuando un humano golpea una roca, al hacerlo su golpe es devuelto con la misma fuerza con que lo hizo (es una roca especial), haciendo directamente el mismo daño que los puntos de ataque del humano. Otra interacción especial es cuando un humano o un goblin golpean a un árbol frutal: en este caso cae una manzana (o algo) y recupera un 30% del máximo de los puntos de vida (*HP*) de los humanos y un 15% del HP máximo de los goblin. Ninguna entidad puede tener más HP que el HP máximo, ni puede tener un HP menor a 0.

La tabla 1 se lee de la siguiente manera: las entidades que están en la columna “Ataca” al atacar a las entidades que están en la fila “Recibe”, golpean su número de puntos de ataque multiplicado por el factor indicado en la tabla, o visto de otra manera, las entidades en la fila “Recibe” al ser atacados por las entidades de la columna “Ataca”, reciben de daño los puntos de ataque de quien los golpeó multiplicado por el factor de la tabla correspondiente. Ejemplo: IceGolem golpea el doble de sus puntos de ataque a FireMage, pero no tiene daño extra al golpear a otro IceGolem.

Ataca/Recibe	Knight	FireMage	Priest	Goblin	IceGolem	Undead	FruitTree	Rock
Knight	1.0	1.5	–	1.25	0.5	1.0	0.3 ↑	1 ↓
FireMage	2.0	2.0	2.0	2.0	5.0	0.5	0.3 ↑	1 ↓
Priest	–	–	–	–	–	5.0	0.3 ↑	1 ↓
Goblin	0.5	1.5	1.0	–	–	–	0.15 ↑	–
IceGolem	1.5	2.0	2.0	2.0	1.0	2.0	–	–
Undead	1.0	1.0	1.0	1.0	–	–	–	–
FruitTree	/	/	/	/	/	/	/	/
Rock	/	/	/	/	/	/	/	/

Table 1: Matriz de interacciones para las entidades del programa. Note la diferencia entre – y /. – es no ataca o no hace daño, / es no aplica. $X \uparrow$ se refiere a que aumenta los puntos de vida en un X de la vida máxima de la entidad. $X \downarrow$ se refiere a que disminuye los puntos de vida un $X\%$ de los puntos de ataque del atacante (solo es $X = 1$ así que siempre se devuelve el daño de exactamente los puntos de ataque del atacante).

Volviendo con los humanos, estos se diferencian de los *Goblin*, *IceGolem* y *Undead*, en que los **humanos tienen un nombre**. Sin embargo, son similares en que todas estas entidades tienen puntos de ataque y puntos de vida (*HP)*. Queda a su criterio definir cómo se contabiliza el daño dentro de cada entidad. No está de más aclarar que una entidad muere y **no puede volver a atacar** cuando sus puntos de vida llegan a 0 o menos. Usted debe tener un método que verifique que efectivamente la entidad puede continuar peleando (puede llamarla, por ejemplo, *canFight*).

Requerimientos adicionales

Además de una implementación basada en las buenas prácticas y patrones de diseño vistos en clases, usted además debe considerar:

- **Cobertura:** Cree los tests unitarios, usando JUnit 4, que sean necesarios para tener al menos un coverage del 90% por paquete. Todos los tests de su proyecto deben estar en un paquete `test`.
- **Javadoc:** Cada interfaz, clase pública y método público deben estar debidamente documentados siguiendo las convenciones de Javadoc[1]. En particular necesita @author y una pequeña descripción para su clase e interfaz, y @param, @return (si aplica) y una descripción para los métodos.
- **Resumen:** Debe entregar un archivo **pdf** que contenga su nombre, su usuario de Github, un link a su repositorio y un diagrama UML que muestre las clases, interfaces y métodos que usted definió en su proyecto. **No debe incluir los tests** en su diagrama.

[1] `http://www.oracle.com/technetwork/articles/java/index-137868.html`

- **Git:** Debe hacer uso de git para el versionamiento de su proyecto. Luego, esta historia de versionamiento debe ser subida a Github.

Evaluación

- **Código fuente (3.5 puntos):** Este ítem se divide en 2:
  - **Funcionalidad (1 punto):** Se analizará que su código provea la funcionalidad pedida.
  - **Diseño (2.5 puntos):** Se analizará que su código provea la funcionalidad implementada utilizando un buen diseño.
- **Coverage (1 punto):** Sus casos de prueba deben crear diversos escenarios y contar con un coverage de al menos 90% por paquete. No está de más decir que sus tests deben testear algo (es decir, no ser tests vacios o sin asserts).
- **Javadoc (1 punto):** Cada clase, interfaz y método público debe ser debidamente documentado. Se descontará por cada falta.
- **Resumen (0.5 puntos):** El resumen mencionado en la sección anterior. Se evaluará que haya incluido el diagrama UML de su proyecto. **En caso de no enviarse el resumen con el link a su repositorio, su tarea no sera corregida**[2].

Entrega

Su repositorio de Github debe ser **privado**, y debe llamarse **cc3002-tarea1**, lo cual significa que si su usuario de Github es *juanpablos*, el link a su repositorio será `https://github.com/juanpablos/cc3002-tarea1`. Además, deberán invitar a la cuenta del equipo docente, *CC3002EquipoDocente*, como colaboradores de su repositorio para que podamos acceder a él. Para hacer esto entran a su repositorio, luego a *Settings*, *Collaborators*, ingresan su contraseña y finalmente escriben *CC3002EquipoDocente* en el campo para buscar por usuario, seleccionan *Add collaborator* y nos habrán invitado exitosamente.

Usted debe subir a U-Cursos **solamente el resumen**, con su nombre, usuario de Github, link a su repositorio y diagrama UML. El código de su tarea será bajado de Github directamente. El plazo de entrega es hasta el domingo 15 de abril a las 23:59 hrs. Se verificará que efectivamente el último commit se haya hecho antes de la fecha límite. No se aceptarán peticiones de extensión de plazo.

[2] porque no tenemos su código.

Recomendaciones

No comience su tarea a último momento. Esto es lo que se dice en todos los cursos, pero es particularmente importante en este. Si usted hace la tarea a último minuto lo más seguro es que no tenga tiempo para reflexionar sobre su diseño y termine entregando un diseño defectuoso o sin usar lo enseñado en el curso.

Haga la documentación de su programa en ingles (no es necesario). La documentación de casi cualquier programa open-source se encuentra en este idioma. Considere esta oportunidad para practicar su ingles.

Les pedimos encarecidamente que las consultas referentes a la tarea las hagan por el **foro de U-Cursos**. En caso de no obtener respuesta en un tiempo razonable, pueden hacernos llegar un correo al auxiliar o ayudantes.