Alpaca Emblem

Proyecto semestral

Profesor: Alexandre Bergel
Auxiliares: Juan-Pablo Silva
Ignacio Slater
Semestre: Primavera 2019

Resumen

Un cliente quiere hacer un juego de estrategia por turnos con distintos tipos de unidades que tienen fortalezas y debilidades entre ellas. Su cliente intentó implementar el juego por si mismo pero se dio cuenta de que no tenía los conocimientos necesarios para lograrlo.

1. Descripción general del proyecto

El proyecto a realizar será programar un *juego de estrategia por turnos* donde participarán dos jugadores y se tendrán distintos tipos de unidades con características particulares. El objetivo del juego es derrotar a una unidad especial de su oponente (a partir de ahora, se referirá a esta unidad como *Héroe*).

Su cliente ya tiene una implementación base del juego, pero busca que usted lo termine, considerando que puede modificar el código original entregado tanto como quiera.

La implementación del proyecto se hará siguiendo el patrón MVC[1], donde cada entrega se referirá a un elemento de éste.

1.1. Modelo

Para la primera entrega se le solicitará que cree todas las entidades necesarias que servirán de estructura base del proyecto y las interacciones posibles entre dichas entidades. Las entidades en este caso se refieren a los elementos que componen el juego. Para el detalle de lo que se pide implementar revise el enunciado de la tarea 1.

1.2. Vista

Se le pedirá también que cree una interfaz gráfica simple para el juego que pueda responder al input de un usuario y mostrar toda la información relevante del juego en pantalla.

1.3. Controlador

Servirá de conexión lógica entre la vista y el modelo, se espera que el controlador pueda ejecutar todas las operaciones que un jugador podría querer efectuar, que entregue los mensajes necesarios a cada objeto del modelo y que guarde la información más importante del estado del juego en cada momento.

[1] **Modelo-Vista-Controlador**. Esto se verá en el curso, pero si se quiere hacer una idea del concepto refierase a este artículo

2. Requisitos adicionales

Que su programa funcione no es suficiente, se espera además que éste presente un buen diseño, siendo esta la característica a la que se le dará **mayor importancia** en la revisión de sus entregas.

Sus programas además deberán estar bien testeados, por lo que **NO SE REVISARÁN** las funcionalidades que no tengan un test correspondiente que pruebe su correcto funcionamiento. Para revisar esto, también es de suma importancia que el trabajo que entregue pueda ejecutarse (que su código compile), en caso contrario no podrá revisarse la funcionalidad y por ende no tendrá puntaje en este aspecto.

Otro aspecto que se tendrá en cuenta al momento de revisar sus tareas es que éstas estén bien documentadas, siguiendo los estándares de documentación de *Java*[2].

Todo su trabajo debe ser subido a un repositorio privado de *Github*. La modalidad de entrega será mediante un resumen en **formato PDF** entregado mediante *u-cursos* que contenga su nombre, rut, un diagrama de clases de su programa y un enlace a su repositorio (no se aceptará código recibido por este medio)[3]. Además su repositorio deberá contener un archivo `README.md`[4]que contenga todas las instrucciones necesarias para ejecutar su programa, todos los supuestos que realice y una breve explicación del funcionamiento y la lógica de su programa.

3. Plazos de entrega

No se aceptarán peticiones de extensión de plazo, pero dispondrá de 72 horas a lo largo del semestre para entregar tareas atrasadas sin que se aplique descuento. Esto significa que si la entrega de la primera tarea se hace con 72 horas de atraso, entonces las siguientes tendrán que entregarse sin atraso. No es necesario dar aviso al cuerpo docente para usar las horas, simplemente se revisará el *commit* correspondiente a la entrega de acuerdo a su *tag*[5].

Las horas de atraso serán aproximadas: para esto, se considerará cada media hora de atraso como 1 hora, y menos que eso como 0 horas, exceptuando la primera hora. Entonces, si se entrega la tarea con 1 minuto de atraso, cuenta como 1 hora de atraso; si se entrega con 1 hora y 1 minuto, también contará como 1 hora; si se entrega con 1:30 de atraso se contará como 2 horas, y así.

4. Recomendaciones

Como se mencionó anteriormente, no es suficiente que su tarea funcione correctamente. Este curso contempla el diseño de su solución y la aplicación de buenas prácticas de programación que ha aprendido en el curso. Dicho esto, no se conforme con el primer diseño que se le venga a la mente, intente ver si puede mejorarlo y hacerlo más extensible.

No comience su tarea a último momento. Esto es lo que se dice en todos los cursos, pero es particularmente importante/cierto 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 deficiente o sin usar lo enseñado en el curso.

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

Se les pide 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 hacerle llegar un correo al auxiliar o ayudantes.

[2] `https://www.oracle.com/technetwork/articles/javase/index-137868.html`
[3] No cumplir con estas condiciones implicará descuento de puntaje.
[4] `https://help.github.com/en/articles/basic-writing-and-formatting-syntax`
[5] Se explicará como agregar un *tag* a un *commit* en clases

Se le recomienda también utilizar la guía de estilos de *Google* para dar formato a su código, esta información la puede encontrar en `https://google.github.io/styleguide/javaguide.html`.

Por último, el orden en el que escriben su programa es importante, se le sugiere que para cada funcionalidad que quiera implementar:

1. Cree los *tests* necesarios para verificar la funcionalidad deseada, de esta manera el enfoque está en como debería funcionar ésta, y no en cómo debería implementarse. Esto es muy útil para pensar bien en cuál es el problema que se está buscando resolver y se tengan presentes cuales serían las condiciones de borde que podrían generar problemas para su implementación.
2. Escriba la firma y la documentación del método a implementar, de esta forma se tiene una definición de lo que hará su método incluso antes de implementarlo y se asegura de que su programa esté bien documentado. Además, esto hace más entendible el código no solo para alguna persona que revise su programa, sino que también para el mismo programador[6].
3. Por último implemente la funcionalidad pensando en que debe pasar los tests que escribió anteriormente y piense si estos tests son suficientes para cubrir todos los escenarios posibles para su aplicación, vuelva a los pasos 1 y 2 si es necesario.

[6]Entender un programa mal documentado que haya escrito uno mismo después de varios días de no trabajar en él puede resultar bastante complicado.