Título del proyecto

Descripción del proyecto

Profesores: Profesorx 1
Profesorx 2
Auxiliares: Auxiliar 1
Auxiliar 2
Semestre: Semestre

Resumen

Resumen del proyecto.[1]

1. Contexto

Descripción detallada del proyecto.

2. Modelo de la solución

Para guiar la solución de este problema, se plantearan objetivos pequeños en forma de *mini-tareas* que buscarán enfrentar una problemática a la vez. Dichas tareas tendrán fecha de entrega pero será opcional entregar en la fecha señalada y no serán evaluadas con nota. Solamente será evaluado que al momento de entregar la tarea se cumplan todos los objetivos planteados en las *mini-tareas*.

La resolución de este proyecto se hará siguiendo el patrón arquitectónico *Modelo-Vista-Controlador*[2], donde primero se implementará el *modelo*, luego el *controlador* y por último la *vista*. Este patrón se explicará en más detalle en el transcurso del curso, pero en el contexto del proyecto estos componentes serán como se explica a continuación.

Modelo Para la primera parte 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.

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.

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] Que las mini-tareas sean opcionales significa que no serán evaluadas individualmente ni se tomará en cuenta la fecha en que sean entregadas, sino que será evaluado que se hayan realizado los requerimientos acumulados de estas en las 3 fechas de entrega del proyecto.

[2] https://www.github.com/CC3002-Metodologias/apunte-y-ejercicios/wiki/Modelo-Vista-Controlador

3. Código base

Explicación del código base.

4. Evaluación

4.1. Bonificaciones

Además del puntaje asignado por cumplir con los requisitos anteriores, puede recibir puntos adicionales (*que se sumarán a su nota total*) implementando lo siguiente:

- **Mini-tareas** (0.4 pts.): De acuerdo a las entregas que haya hecho de las mini-tareas puede recibir una bonificación de hasta 0.4 pts. Entregue las mini-tareas solamente si considera que están completas o altamente completas. Entregar mini-tareas demasiado incompletas puede significar un descuento en el puntaje. Para los casos de mini-tareas que estén completas pero contengan errores pueden optar a una fracción del puntaje.
- **Exclusivamente para la tarea 3:**
  - **Interfaz gráfica avanzada** (0.5 pts.): Si su interfaz gráfica cumple más de los requisitos mínimos podrá recibir una bonificación de hasta 0.5 pts. (esto quedará a criterio del ayudante que le revise).
  - **Manejo de excepciones** (0.3 pts.): Se otorgará puntaje por la correcta utilización de excepciones para manejar casos de borde en el juego. Recuerde que atrapar *runtime exceptions* es una mala práctica, así como arrojar un error de tipo `Exception` (esto último ya que no es un error lo suficientemente descriptivo). Bajo ninguna circunstancia una de estas excepciones debiera llegar al usuario.

4.2. 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 estas estén bien documentadas, siguiendo los estándares de documentación para *Java* de *Google*.[3]

Todo su trabajo debe ser subido al repositorio privado que se le entregó a través de *Github Classroom*. 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).[4] Además su repositorio deberá contener un archivo `README.md`[5]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.

Para la tarea deberá crear su propia rama para trabajar y subir todo su código en esta y, cuando tenga una entrega lista para ser revisada, realizar un ***pull request***[6], esta será la versión que será revisada. Tenga en cuenta que si hace *push* de otros *commits* en esa rama se actualizará el *PR*, por lo que se le recomienda crear una nueva rama para seguir trabajando.

[3] https://github.com/CC3002-Metodologias/apunte-y-ejercicios/wiki/Convenciones
[4] No cumplir con estas condiciones implicará descuento de puntaje.
[5] https://help.github.com/en/articles/basic-writing-and-formatting-syntax
[6] https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/creating-a-pull-request

4.3. 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 *Pull request*.

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.

La entrega de las mini-tareas se hará también mediante *u-cursos*, para esto basta entregar un documento de texto plano con su nombre, *rut*, y un enlace al *Pull request* que cuente como entrega.

4.4. Evaluación

- **Código fuente (4.0 puntos):** Este ítem se divide en 2:
  - **Funcionalidad (1.5 puntos):** Se analizará que su código provea la funcionalidad pedida. Para esto, se exigirá que testee las funcionalidades que implementó[7]. **Si una funcionalidad no se testea, no se podrá comprobar que funciona y, por lo tanto, NO SE ASIGNARÁ PUNTAJE por ella.**
  - **Diseño (2.5 puntos):** Se analizará que su código provea la funcionalidad implementada utilizando un buen diseño.
- ***Coverage* (1.0 puntos):** Sus casos de prueba deben crear diversos escenarios y contar con un *coverage* de las líneas de al menos 90 % por paquete. No está de más decir que sus tests deben testear algo (es decir, no ser tests vacíos o sin *asserts*).
- ***Javadoc* (0.5 puntos):** 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 será corregida**.[8]

4.5. 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.

[7] Se le recomienda enfáticamente que piense en cuales son los casos de borde de su implementación y en las fallas de las que podría aprovecharse un usuario malicioso.

[8] Porque no tenemos su código.

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.

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[9].
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.

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