Es un videojuego programado en Java que se me pidió en algoritmos y programación 3 como proyecto final para evaluar mis conocimientos respecto a la programación orientada a objetos, principios SOLID, metodología scrum y realización de diagramas UML de clases, secuencia, estados y paquetes.

Esta era la consigna:

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**
   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.
   <u>Los enemigos:</u>
   ● Hormiga: ○ Tiene 1 de velocidad ○ Causa 1 punto de daño al llegar a la meta ○ Tiene 1 punto de energía ○ 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 destruida. Los créditos que otorgan pertenecen al rango 0..10.
   ● Topo: ○ En sus primeros 5 movimientos tiene 1 de velocidad, luego en los próximos 5 su velocidad es 2, a partir del movimiento 11 su velocidad queda definitivamente en 3. ○ Si el número de turno en que llega a la meta es impar, entonces causa 5 puntos de daño, sino sólo 2. ○ El topo no camina por la superficie, sino que va enterrado lo que hace que ninguna torre lo pueda atacar.
   ● Lechuza: ○ Tiene 5 de velocidad. ○ Tiene 5 puntos de energía. ○ No causa daño al jugador al llegar a la meta, sino que destruye automáticamente la primera torre construida por el jugador. Si llega una segunda lechuza, destruirá la 2da torre construida y así sucesivamente. Si el jugador no tiene torres entonces la lechuza no produce ningun efecto al llegar a la meta. ○ Al volar, la lechuza puede moverse por cualquier tipo de parcela, no tiene restricciones. Siempre busca llegar a la meta. Cuando arranca la Lechuza vuela en forma de L hacia la meta (Hace los 2 catetos). PERO cuando la lechuza tiene menos del 50% de vida, cambia drásticamente su forma de volar y pasa a volar en linea recta directamente a la meta (va por la hipotenusa) 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

Trampa arenosa:
● Sólo puede ser construida sobre una pasarela que no puede ser ni la de llegada ni la meta.
● Ralentiza el avance en un 50% a todos los enemigos que estén en la misma posición que la trampa arenosa. (No afecta la lechuza, pero sí topo, hormiga y araña) ● Tiene una vida util de 3 turnos. Se construye de forma inmediata, es decir al turno siguiente que se la decidió construir ya está operativa por 3 turnos.
● Cuesta 25 créditos.

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: Primer cuatrimestre 2023 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.

5. **Entregables para cada fecha de entrega**

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

   Entrega 0 (Semana 1):

   ● Planteo de modelo tentativo con diagramas de clases. ● Repositorio de código creado según se explica aquí.

   ● Servidor de integración continua configurado.

   Entrega 1 (Semana 2): 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 3): 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 ● Simular 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 4):

Interfaz gráfica inicial básica:

● Pantallas inicial de comienzo del juego donde se le pide el nombre al usuario. Validación del mismo y boton para dar inicio al juego.

● Visualización del mapa según JSON. Finalización del modelo junto a todas las pruebas unitarias y de integración incluyendo la lechuza, el topo y la trampa arenosa. 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 5): Interfaz gráfica completa acorde al enunciado.

Entrega 5 - Final (Semana 6): Trabajo Práctico completo funcionando, con interfaz gráfica final, sonidos e informe completo.