INF-253 Lenguajes de Programación
Tarea 3: Java

Profesor: José Luis Martí, Jorge Díaz Matte, Rodrigo Salas Fuentes
Ayudante Cátedras: Gonzalo Severín Muñoz, Jhossep Martínez Velásquez
Ayudante Coordinador: Álvaro Rojas Valenzuela
Ayudante Tareas: Ricardo Barriga Vera, Bryan González Ramírez, Bastián Salomón Ávalos, Cristian Tapia Llantén, Cristóbal Tirado Morales

30 de abril de 2024

1. Ultra-Java

Después del estrepitoso fracaso de FactoryOfNumbers. Tu jefe Zoddax te deja la libertad de crear un nuevo juego en lenguaje Java. Para ello, diseñas un diagrama de clases UML basado en famoso juego de matar demonios. Ahora tienes que llevar la idea a la realidad, enfrentando todas las fases del desarrollo de videojuegos.

2. Juego de Rondas y Estilo

Este juego consiste en rondas de enemigos en una gran arena. El giro interesante es que para dañar a los enemigos, estarás al merced de un generador de números aleatorio. Cada vez que dispares tu arma a una parte a elección del enemigo, la probabilidad de darle dependerá de la precisión del arma y la probabilidad de darle a esa parte del enemigo. Y cada vez que dispares o mates a un enemigo, se te recompensan con puntos de estilo llamados puntos P. Para luego ser canjeados en la terminal al centro de la arena por nuevas armas.

3. Clases a Implementar

Ultra-Java puede ser representado por la siguiente UML:

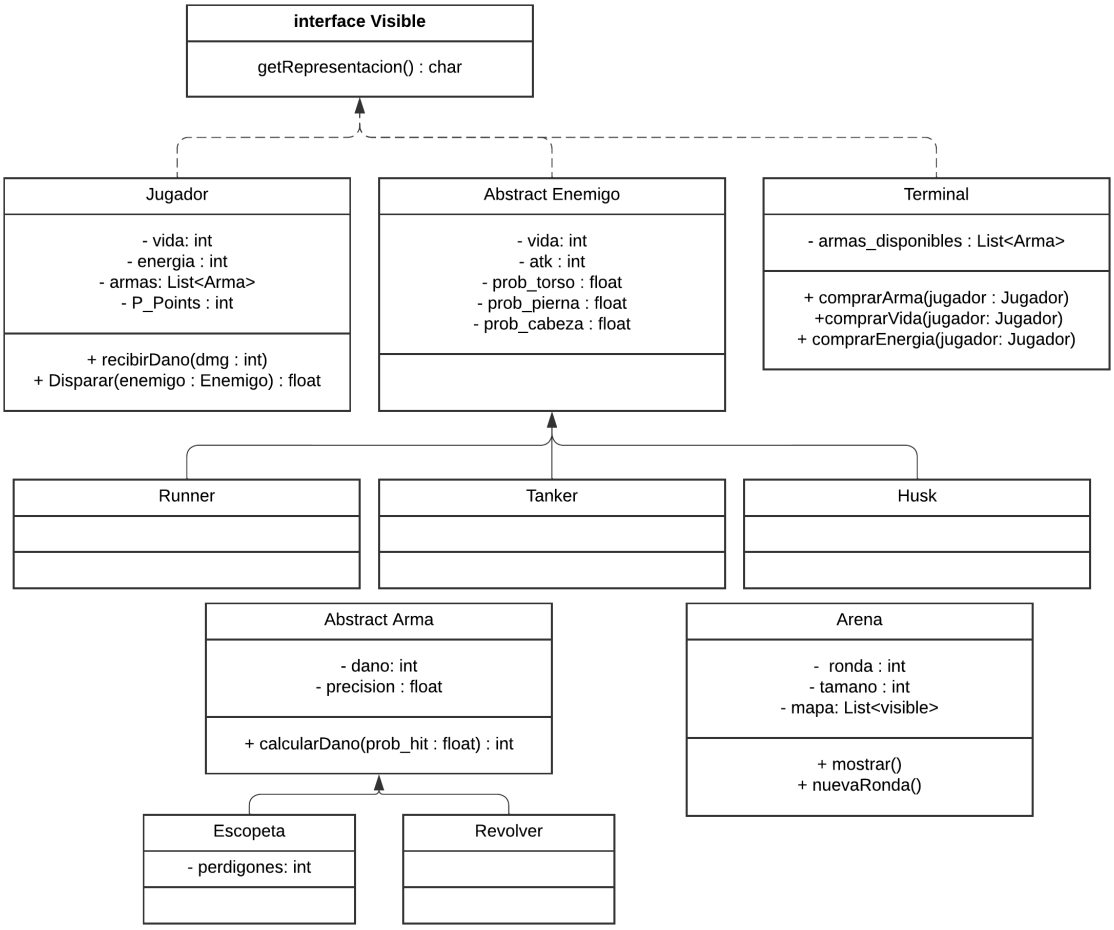

Figura 1: UML Ultra-Java

3.1. Arena

Esta clase maneja la arena y las rondas de enemigos.

- ronda: Número de la ronda actual.
- tamano: Tamaño de la arena.
- mapa: Array 1D de los elementos de la arena.
- mostrar(): Imprime el mapa por consola. Utiliza getRepresentacion() para obtener el símbolo de cada elemento.

- nuevaRonda(): Aumenta en uno el atributo ronda, reposiciona al jugador al lado derecho de la terminal y genera una nueva ronda de enemigos.

3.2. Jugador

Esta clase es el jugador.

- vida: Vida actual del jugador.
- energia: Energía actual del jugador.
- armas: Array de armas del jugador.
- P_Points: puntos P actual del jugador.
- Disparar(enemigo): Dispara un arma a elección del jugador a un enemigo.
- getRepresentacion(): Entrega una letra que represente al jugador.

3.3. Enemigo

Esta es una clase abstracta de enemigos, estos son tres tipos, Husk, Tanker y Runner.

- vida: La vida actual del enemigo.
- atk: Puntos de daño que hace el enemigo.
- prob_toso: número entre 0 y 1, representa la probabilidad de recibir daño en el torso.
- prob_pierna: número entre 0 y 1. Representa la probabilidad de recibir daño en la pierna.
- prob_cabeza: número entre 0 y 1. Representa la probabilidad de recibir daño en la cabeza.
- getRepresentacion(): Entrega una letra que represente al enemigo. Por ejemplo, 'H' Puede ser para Husk, 'K' para Tanker y 'R' para Runner.

Cada subclase de Enemigo inicia con atributos diferentes a decisión del programador. Por ejemplo, Husk puede tener menos vida que Tanker. Pero Tanker tiene más probabilidad de que reciba daño en el torso. Mientras que Runner tiene probabilidades más bajas, pero su vida es la menor de todas.

3.4. Terminal

Esta clase es persistente en todas las rondas. Este siempre está posicionado en el centro de la arena. Y permite al jugador canjear puntos P por una arma nueva, vida o energía. La clase posee lo siguiente:

- armas_disponibles: Lista de armas disponible para la venta.
- comprarArma(jugador): Permite canjear una cierta cantidad de P_Points del jugador por alguna de las armas disponibles.

- comprarVida(jugador): Permite canjear una cierta cantidad de P_Points del jugador para recuperar una cierta cantidad de vida.
- comprarEnergia(jugador): Permite canjear una cierta cantidad de P_Points del jugador para recuperar una cierta cantidad de energía.
- getRepresentacion(): Entrega una letra que represente a la terminal. Por ejemplo, puede ser 'T'.

Los precios que se les pide al jugador y la cantidad que recibe de vuelta es a disposición del programador. Decida que tan fácil o estrepitosamente difícil quera el juego. El jugador solo puede tener un tipo de arma en su arsenal. (Lista de armas del atributo de Jugador).

3.5. Arma

Esta clase abstracta corresponde a las armas disponibles en el juego. La clase posee lo siguiente:

- dano: Daño que provoca el arma.
- precision: Es un valor entre 0 y 1. Indica la precisión del arma.
- calcularDano(prob_hit): Este método calcula el daño que provoca el arma dada la probabilidad de darle al objetivo (prob_hit). Para calcular el daño del arma se tiene que obtener un número aleatorio r entre 0 y 1. Y sí $r > porb\_hit \cdot precision$ el daño del arma es el valor de **dano**. Si de lo contrario, el daño es 0.

La escopeta es un arma de perdigones. Por lo tanto, todos los perdigones tiene una misma probabilidad de darle al enemigo. Por ende, el daño provocado del arma es una multiplicación de **dano** por la cantidad de perdigones que acierta el disparo. La probabilidad de cada perdigón es un número aleatorio r entre 0 y 1, Y si $r > prob\_hit \cdot precision$, entonces este perdigón acierta.

Los valores de los atributos de escopeta y revolver está a decisión del programador.

3.6. Otros atributos y métodos

Todos los atributos y métodos previamente mencionados deben estar implementados en la tarea, pero deben incluir los constructores en todas las clases que estimes conveniente. Además, todas las clases deben incluir todos los *getters* y los *setters* según la cantidad de atributos. **No debe acceder directamente a los atributos afuera de la clase, deben estar obligatoriamente en privado**. Puede incluir otros atributos y métodos adicionales, si lo desea.

4. Inicialización del juego

Primero se crea un objeto Arena inicializándose con los siguientes datos:

- ronda = 0
- tamano = 15
- mapa = new Visible[tamano]

Luego se inicializa una instancia de Terminal. Se debe posicionar en el centro del mapa y su atributo *arma_disponible* debe contener una lista de todas las armas disponible para vender. (En este caso es solamente la escopeta). También se debe inicializar una instancia de jugador. Se debe posicionar en la celda continua a la derecha de la terminal. Y debe inicializar con los siguientes datos:

- vida = 100
- energia = 3
- P_Points = 0

Además, en el atributo *armas* debe contener una instancia de Revolver. Luego de tener todos los elementos iniciales, puede iniciar el juego.

5. Flujo de juego

Este es un juego de rondas de enemigos. Por cada ronda, el jugador es reposicionado en la celda continua a la derecha de la terminal y al menos 3 enemigos aparecen en cualquier otra parte diferente a las celdas que ocupa la terminal y el jugador. Entre rondas, por cada acción del jugador, primero se debe mostrar el mapa y luego el jugador puede hacer alguna de estas opciones:

- **Ver estadísticas:** El jugador puede ver su vida actual, su energía actual, los Puntos P actuales y los valores de su arsenal actual.
- **Mover:** El jugador se debe mover a una casilla adyacente dentro del mapa o quedarse en la misma. Si en ese lugar lo ocupa un enemigo, deben ocurrir estos tres eventos:
  - Se le da la opción de disparar y a que parte del enemigo con un arma de su arsenal a elección, provocando daño al enemigo. Si la vida de este llega a 0, el enemigo deja de existir.
  - El enemigo daña al jugador (independientemente si este muere primero), restando puntos de vida según los puntos de ataque de este. Pero si el jugador tiene al menos un punto de energía, se le da la opción de esquivar el ataque, denegando todo el daño.
  - Cada vez que acierte o mate a un enemigo, al jugador se le es recompensando con puntos P. La cantidad de estos puntos puede depender, por ejemplo, si mata a un enemigo de un tiro a la cabeza se le da más punto a que solo le reste vida en un disparo a la pierna. Sé creativo en esta parte para mostrar el increíble estilo de acabar con los enemigos.

  Si en el lugar ocupa la Terminal, el jugador puede decidir si va a comprar vida, energía o una nueva arma canjeando una cierta cantidad de puntos P.

Una vez que no existan enemigos, inicia la siguiente ronda. Y cuando el jugador supere todas las rondas, este gana. Pero si la vida del jugador llega a 0, este pierde y el juego termina.

6. Datos importantes

- El formato de lo que se muestra por consola es **libre**. Pero este debe ser clara para el entendimiento del juego.

- El mapa puede estar conectado en los extremos.
- Se deja libre al programador establecer la dificultad del juego, se le pide que en cada ronda aparezca al menos 3 enemigos, uno de cada tipo, puedes agregar más.
- Puedes aumentar el tamaño de la arena si lo deseas, como mínimo se pide de 15.
- Puedes crear más variaciones de armas y enemigos a gusto, con nuevas mecánicas y estadísticas. Expresa tu desarrollador de videojuegos interno. (Asegura de probar las mecánicas, que el juego no se cierre espontáneamente por un error).
- Se debe crear un archivo llamado **Ultrajava.java** donde contiene la función main.
- Puede decidir si una clase Enemigo bloquea el paso al jugador, pero no es el caso para la Terminal.

7. Sobre la Entrega

- El código debe venir indentado y ordenado.
- Se debe entregar como mínimo los siguientes archivos:
  - Ultrajava.java
  - Arena.java
  - Arma.java, Escopeta.java, Revolver.java
  - Visible.java
  - Jugador.java, Terminal.java
  - Enemigo.java, Runner.java, Tanke.java, Husk.java
  - Makefile, Readme.txt
- Los métodos y funciones deberán ir comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y los que devuelve (en caso de que devuelva algo). Se deja libertad al formato del comentario.
- En caso de que agregue una nueva clase, debe comentar brevemente una descripción general de esta.
- Debe estar presente el archivo Makefile para que se efectúe la revisión. Este debe compilar todos los archivos.
- Se debe trabajar de forma individual obligatoriamente.
- **La entrega debe realizarse en un archivo comprimido en tar.gz, y debe llevar el nombre: Tarea3LP_RolAlumno.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno, e instrucciones detalladas para la correcta utilización, compilación y ejecución del programa**.
- La entrega será vía aula y el plazo máximo es hasta el **17 de mayo a las 23:55 horas**.

- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro de primera hora).
- Las copias serán evaluadas con 0 y se les avisará a las respectivas autoridades.
- Se va a utilizar el compilador javac versión 11.

8. Clasificación

8.1. Entrega

Para la clasificación de su tarea, la entrega debe cumplir con los requerimientos mínimos que le permitan obtener 30 puntos base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

8.1.1. Entrega Mínima

Para obtener el puntaje mínimo en la entrega, el programa de cumplir con los siguientes requerimientos:

- Implementa la clase Arena de tal forma que solamente muestre por pantalla el mapa del juego.
- Instancia un Jugador en el mapa y este se puede mover libremente en el mapa sin problemas.
- Instancia una Terminal al centro del mapa, no es necesario que el jugador interactúe con ella.

8.1.2. Entrega

Luego de cumplir con la Entrega Mínima, puede obtener más puntaje cumpliendo con los siguientes puntos (puede haber puntaje parcial por cada punto):

- Implementación de Interfaz Visible: (Max 8 pts)
  - Crea correctamente la interfaz.
  - Las clases Jugador, Enemigo y Terminal implementan correctamente la interfaz.
- Implementación de Jugador: (Max 15 pts)
  - Implementa recibirDano(), restando el atributo de la vida cuando corresponde y la posibilidad de esquivar el daño usando una cantidad de energía del jugador.
  - Implementa Disparar() correctamente.
  - Constructor, getters y setters de la clase correspondiente.
- Implementación Enemigo: (Max 10 pts)
  - Implementa correctamente las subclases de Runner, Tanker y Husk como tres tipos de enemigos y con atributos iniciales diferentes.
  - Constructores, getters y settes de las subclases correspondientes.
- Implementación Terminal: (Max 10 pts)

- Implementa comparArma() correctamente, restando una cantidad de P Points del jugador y entregando una nueva arma.
- Implementa comprarVida() correctamente, restando una cantidad de P Points del jugador y aumentando el atributo vida.
- Implementa comparEnergia() correctamente, restando una cantidad de P Points del jugador y aumentando el atributo energía.
- Constructores, getters y setters de la clase.

- Implementación Arma: (Max 12 pts)
  - Implementa correctamente las subclases Escopeta y Revolver.
  - Implementa calcularDano() diferentes para las subclases correctamente.
  - Constructores, getters y setters de las subclases correspondientes.
- Flujo de juego: (Max 15 pts)
  - El flujo de juego inicia con una nueva ronda cada vez que el jugador acabe con todos los enemigos, reposicionando al jugador donde corresponde.
  - Hay al menos 3 rondas, con al menos 3 enemigos diferentes cada una de las rondas.
  - Por cada acción del jugador se muestra el mapa y puede ver sus estadísticas e interactúa con los enemigos y la terminal.
  - El juego acaba cuando el jugador muere o supera todas las rondas.
- **Puntaje extra:** Crea una interfaz de usuario creativa y/o crea nuevas armas/enemigos con mecánicas nuevas que haga la experiencia del juego mejor. (10 puntos)

8.1.3. Descuentos

Luego de cumplir con la Entrega Mínima, puede sufrir descuentos por los siguientes motivos:

- Atributo con visibilidad en público (-5 pts c/u)
- Falta de orden (-20 pts)
- Código no compila (-100 pts)
- Warning (-5 pts c/u)
- Falta de comentarios (-10 pts c/u, Max 30 pts)
- Falta de Readme (-20 pts)
- Falta de Makefile (-100 pts)
- Falta de alguna información obligatoria en el README (-5 pts)
- Día de atraso (-20 pts por día, -10 dentro de la primera hora)
- Mal nombre en algún archivo entregado (-5 pts)

En caso de existir nota negativa, esta será remplazada por un 0.