

Comisión 11

<https://github.com/maxi512/ProyectoTDP2018.git>

Integrantes

- Romina, Vitacca
- Maximiliano Agustin, Montenegro
- Joaquin Leonel Rau

Errores encontrados

- × Si se presionan las teclas de movimiento en el menú principal al comienzo del juego se lanza una excepción *NullPointerException*.
- × En la clase Juego ustedes tienen un atributo `int nivel actual`, eso es equivalente a hacer un `instance of`. Lo mejor sería que cada nivel conozca al siguiente, y que no se creen todos los niveles al principio si no que se creen a medida que se pasa de nivel.
- × Para llevar a cabo el power up de detener tiempo se intenta utilizar un visitor pero lo usan incorrectamente.
- × El cambio de inteligencia de los enemigos lo realiza el método `mover` o `disparar` de inteligencia, no me parece que fuera responsabilidad de este método. Debería ser responsabilidad de "recibir daño" (o su equivalente) ya sea en la inteligencia o en el enemigo. Al realizar este cambio deben considerar la interacción con el power up de congelar el tiempo porque por ejemplo algunos enemigos pierden su arma cuando tienen menos de 20% de su vida y cambian su inteligencia a kamikaze, si este cambio se produce en el momento que está congelado el tiempo los enemigos mencionados anteriormente se descongelarían antes de tiempo.
- × En `InteligenciaEnemigoSinArma` sobrescriben el método `mover` pero hace exactamente lo mismo (repiten el código).
- × El arma debería crear el disparo con su posición ya inicializada.
- × Se debería tratar de evitar el uso de valores hard-codeados, por ejemplo, la decisión de cuando los disparos desaparecen no debería depender de un valor hard-codeado.
- × En la clase `Disparo` los métodos: `golpearJugador(Jugador j)`, `golpearEnemigo(Enemigo enem)`, `golpearObstaculoEnemigoYJugador(Obstaculo o)`, `golpearObstaculoJugador(Obstaculo o)` deberían ser abstract.
- × En la clase `"ColisionadorObstaculoRompeJugador"` el método `"afectarDisparo"` debería recibir por parámetro un `DisparoJugador` porque en caso contrario los disparos enemigos también lo afectan.
- × No es necesario que todos los power-ups conozcan el juego, lo único para el cual sería necesario es el que detiene el tiempo.
- × En `ArmaMejorada`, `ArmaMisil`, `ArmaRapida` el método `generarDisparo()` podría hacer uso de `super.generarDisparo()` para llevar a cabo una parte de su tarea y no repetir tantas veces el mismo código.
- × Lo mismo del anterior sucede en la inteligencia.
- × Si no entiendo mal el método `actualizar` de la clase `arma` no es necesario. En todo caso debería ser privado y llamado al principio de `generarDisparo`.
- × ¿Por qué hay un hilo solo para disparos? ¿No sería mejor que el juego solo conociera entidades y a lo sumo al jugador?
- × El boss debería tener un arma, y esta generar los disparos.
- × Todos los métodos de la clase `Colision` deberían ser abstractos.

- × En la clase GeneradorPowerUp en lugar de tener un atributo cantidadPowerUpTiempo sería mejor que la clase PowerUpTiempo fuera singleton.
- × ¿La clase PrimerBoss no la usan para nada? Creo que la intención de ustedes era usar en algún momento el método manage() de la clase Juego, pero según pude ver no lo hacen. Igualmente me parece que la responsabilidad de hacer esto es del último nivel.
- × La detección de colisiones debería realizarse para ambas entidades involucradas: entidad1.colisionar(entidad2) y entidad2.colisionar(entidad1). Por ejemplo la colisión del disparo enemigo con el jugador hace que el jugador reciba daño, y la colisión del jugador con el disparo enemigo hace que el disparo desaparezca (muera).
- × En la clase jugador los métodos golpearObstaculo, y golpearJugador deberían ser abstractos.
- × Deberían tener un atributo de "fuerza de impacto" para las clases que producen daño.
- × Si agregan el "FinalBoss" al juego deberían también agregarlo al UML.

Recomendaciones

- Para determinar si el nivel está completo verifican si hay entidades todavía, esto hace que si eliminan todas las entidades pero se continúa disparando o no se le eliminó una barricada por ejemplo no sea posible completar el nivel. Me parece que sería mejor que se verifique si ya no hay enemigos.

Lineamientos del proyecto

- Mapa
 - Escenario
Correcto
 - *Plano secuencia*: al llegar un enemigo a la parte inferior de la pantalla, deberá volver a aparecer en la parte superior, o en su defecto en su posición original
Correcto: los enemigos traspasan la parte inferior del mapa.
- Nivel
 - Dos niveles existentes
Correcto: se identificaron 4 niveles.
 - Obstáculos en un nivel
Correcto: existen obstáculos fijos.
- Enemigos
 - Fuerza de impacto
Parcialmente Correcto: Lo modelan usando valores "hard-codeados", lo mejor sería tener un atributo "fuerza de impacto" para las entidades.
 - Diferente velocidad
Correcto: Si bien en los diferentes niveles usan la misma velocidad es fácilmente configurable.
 - Diferente cantidad de vida
Correcto: Si bien en los diferentes niveles usan la misma vida es fácilmente configurable.
 - Tres kamikazes
 - Búsqueda de jugador
Correcto: la inteligencia Kamikaze se dirige al Jugador.
 - Aleatorio
Correcto: tienen una inteligencia Kamikaze que se mueve aleatoriamente.
 - Búsqueda hasta 50% y luego mareado (aleatorio)

- Además han incorporado otras inteligencias para enemigos
- Pérdida del arma

- Obstáculos

- Parcialmente Correcto:** según lo que se pudo observar existen dos tipos de obstáculos. Aquellos que se pueden destruir y los que no. Falta implementar correctamente para que solamente el jugador puede destruir algunos obstáculos.

- Magia temporal

- Correcto:** Las entidades, a excepción del jugador, se detienen cuando se captura el power up.

- Correcto:** El jugador cambiar de arma y puede disparar super misiles cuando se captura el power up.

- Campo de protección contra kamikaze

- Poción de vida

Aspectos destacados

- 3