Prueba Backend

2019

Beezy

Contenidos

Introducción

El propósito de esta prueba es evaluar cómo organizas e implementas código.

La prueba consiste en entregar **dos ejercicios de código fuente** basados en el caso descrito a continuación. No hay una única solución para los ejercicios (puede haber muchas formas distintas de interpretarlos y de plasmarlos en código); simplemente queremos ver cómo planteas la estructura de código para representar un escenario concreto.

Al final del documento tienes una sección “Requisitos sobre la entrega” que detalla los lenguajes de programación admitidos y el formato de la entrega.

Descripción del escenario

El escenario planteado es un videojuego de naves espaciales. La prueba no consiste ni mucho menos en implementarlo, simplemente imaginar que hay un equipo de desarrollo que está implementando dicho juego a través de una aplicación y al que han descrito los requerimientos funcionales que debe cumplir. Este equipo de desarrollo ha planteado la estructura principal de la aplicación y ahora necesita implementar ciertas partes clave de ella.

La dinámica general del juego es:

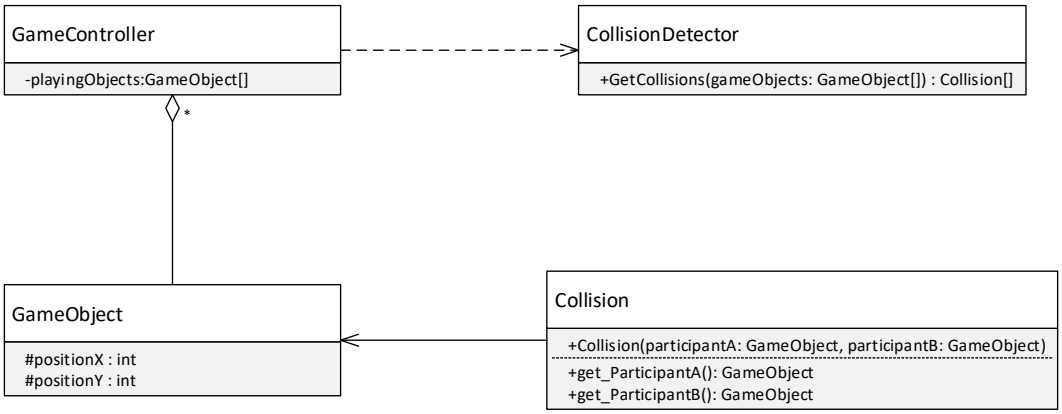
- El juego se desarrolla en un escenario de 2 dimensiones. La coordenada 0,0 se localiza en la esquina inferior izquierda.
- El jugador maneja una nave espacial que puede desplazarse en cualquier dirección del plano.
- La nave espacial puede destruir o esquivar al resto de elementos hostiles visibles en pantalla. Estos aparecerán por la parte superior de la pantalla.
- El juego finaliza cuando la nave espacial es alcanzada por un disparo o colisiona con otro objeto.
- Se produce una colisión cuando dos objetos comparten exactamente las mismas coordenadas.

Tipos de objetos voladores:

- Nave enemiga
    - Movimiento: Se desplazan desde la parte superior de la pantalla hacia abajo mientras barren la pantalla de un lado a otro tratando de localizarnos.
    - Ataque: Si nuestra nave espacial se encuentra en la misma columna y en una fila inferior a ella creará un objeto misil dirigido hacia nosotros (zona inferior de la pantalla).
- Misil
    - Movimiento: Manteniendo la posición vertical en la que fueron creados se desplaza hacia la parte superior o inferior de la pantalla (en función de si fueron lanzados por nosotros o por una nave enemiga).
    - Ataque: No soportado.
- Meteorito:

    - Movimiento: Se desplazan desde la parte superior de la pantalla hacia abajo con un ángulo y una velocidad aleatorias.
    - Ataque: No soportado.
- Agujero de gusano:
    - Movimiento: No soportado. Se crean en una posición aleatoria y permanecen en esa posición hasta que colisionan con otro objeto.
    - Ataque: Si nuestra nave entra en su radio de acción será atraída hacia él. Si no nos alejamos seremos destruidos al colisionar con él.
- Nuestra nave espacial:
    - Movimiento y ataque: No manejado por el controlador del juego.
    - Ataque: Crea un objeto misil dirigido hacia la zona superior de la pantalla.

Estructura base del código



- GameController:
  - Mantiene la colección de objetos voladores presentes en pantalla.
  - Periódicamente recorre esta colección e
    - Invoca acciones sobre cada objeto volador en función de sus capacidades (mover o atacar).
    - Invoca al componente de detección de colisiones para determinar los objetos que han colisionado y eliminarlos. Si nuestra nave espacial ha colisionado con otro elemento finaliza la partida.

- CollisionDetector:
  - Recibe un array de GameObjects a evaluar y retorna otro array con las colisiones (objetos Collision) entre objetos detectadas.
  - Las colisiones mantienen una referencia a cada uno de los elementos (GameObject) participantes en la colisión.
  - Dos colisiones con los mismos objetos participantes se consideran iguales.

Objetivos ejercicio 1

- Implementar una **librería de clases** que represente el escenario descrito.
- No se trata de implementar una aplicación entera, sino sólo el **dominio** que representa a los objetos voladores.
- Buscamos ver reflejada la estructura de clases, propiedades y métodos y no tanto la implementación de los propios métodos, estos pueden estar vacíos, simplemente con un comentario que diga lo que deberían hacer.
- Queremos ver especialmente reflejados en ese dominio los **principios básicos de la programación orientada a objetos**.
- Si quieres, opcionalmente puedes entregar:
    - Modelado del dominio en UML (si lo has utilizado para plantear la estructura de clases).
    - Comentarios adicionales (ya sean inline en el código o en un documento aparte) justificando los planteamientos y principios utilizados.

Objetivos ejercicio 2

Basado en el dominio implementado en el ejercicio 1, **implementar el método principal del componente de detección de colisiones** *GetCollisions* según lo explicado en el escenario, y cuya firma deberá ser:

GetCollisions(gameObjects: GameObject[]) : Collisions[]

Basado en las posiciones de cada objeto volador, deberá devolver las colisiones entre ellos.

Puedes crear una clase CollisionDetector vacía que sólo contenga ese método. Simplemente nos interesa ver la implementación concreta de ese algoritmo de detección de colisiones, no hace falta hacer más métodos ni clases que lo utilicen.

Requisitos sobre la entrega

- El lenguaje utilizado puede ser **Java** o **C#**.
- Los dos ejercicios pueden entregarse en el mismo proyecto de código o en proyectos separados.
- Código que compile correctamente (no hace falta ejecutarlo, sólo compilarlo).
- El código puede entregarse en ZIP o subirse a algún repositorio online.

Ponte en contacto con nosotros si necesitas resolver alguna duda sobre el escenario o cómo plantear algún aspecto de la prueba.