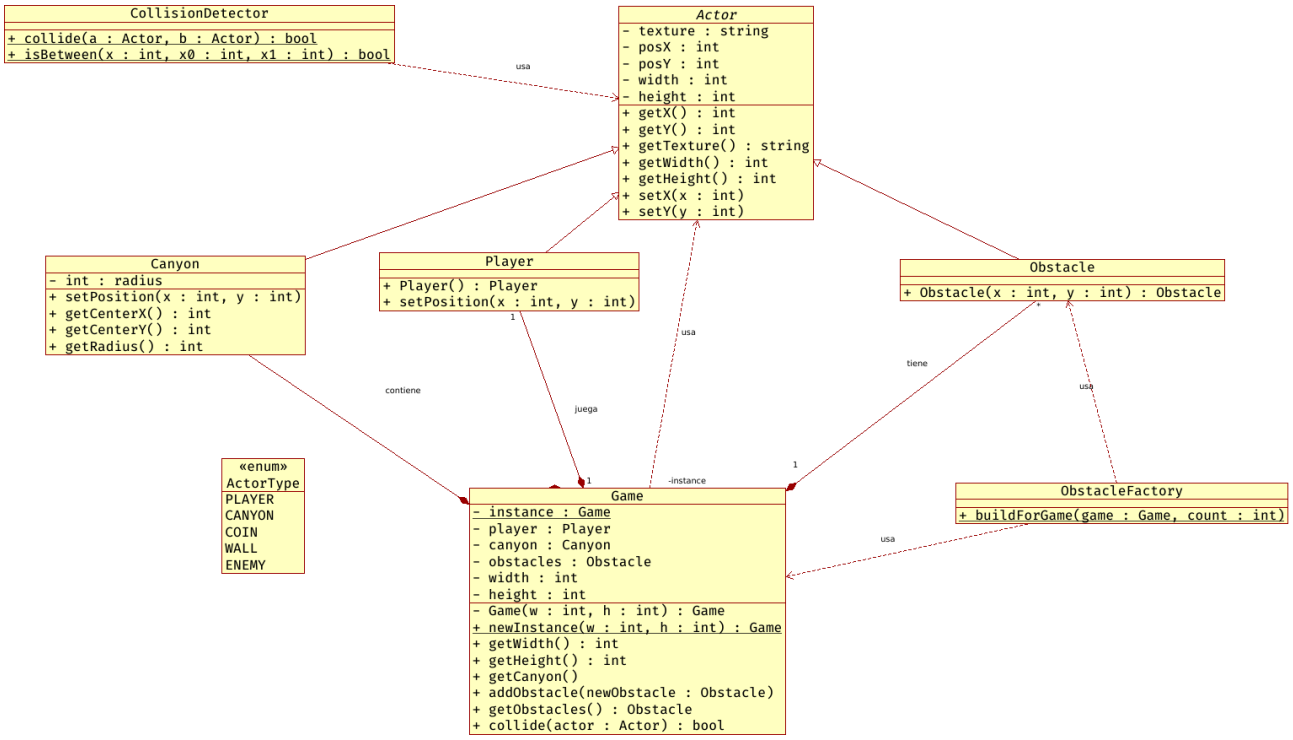
Diagrama de clases

Implementación

Enunciado

En esta práctica tienes que terminar de implementar un juego escrito con Java y JXGL. Esta parte por terminar la tienes modelizada en el siguiente UML:



El código a completar lo tienes en el repositorio siguiente: https://github.com/rgmf/eed-tema4-p2

Haz un *fork* del mismo y empieza a trabajar sobre esa base.

La práctica se entregará vía GitHub dentro del plazo indicado en Aules. El repositorio de GitHub en la que esté tu práctica se tiene que llamar **`eed-tema4-p2`**.

En los siguientes apartados te doy los detalles necesarios para implementar los métodos de todas las clases.

Clase Game

Esta clase representa el juego en sí. Como ves tiene algo "especial" que es un atributo estático llamado `instance` del mismo tipo que la clase. Dicha clase implementa el patrón de diseño *Singleton*.

Te explico brevemento qué hace cada método y el constructor de esta clase. Omito los métodos *getter* y *setter* pues ya sabes lo qué hacen.

a) Constructor

En el constructor tienes que inicializar los atributos `width`, `height`, `player`, `canyon` y `obstacles`.

Los atributos `width` y `height` se inicializan con los valores pasados al constructor: `w` y `h`.

El objeto `player` se crea en la posición `x=20` e `y=20`.

El objeto `canyon` se crea con un radio de `50` y su posición será: `x=width / 2 – 50` e `y=height / 2 – 50`

Por último, el atributo `obstacles` será un `array` de `50` objetos de la clase `Obstacle`.

b) newInstance

El método `newInstance` inicializa el atributo `instance` (solo si fuera `null`).

Además, devuelve como resultado dicho atributo `instance`.

c) getObstacles

Este método *getter* es un tanto especial porque no va a devolver el objeto `obstacles` sino un `array` con los `Obstacle` que no son `null`.

d) addObstacle

Este método tiene que añadir en `newObstacle` que viene como parámetro en el array `obstacles` de la clase. Se añade, si hay sitio, en la última posición libre.

¿Cómo sabes cuando una posición está libre? Porque es igual a `null`.

¿Cómo sabes si está lleno el array? Porque no hay posiciones iguales a `null`.

e) collide

Este método comprueba si hay colisión entre el `actor` (de la clase `Actor`) y cualquiera de los actores de esta clase: `player`, `canyon` y `obstacles`.

Si hay colisión con alguno devuelve `true`, en otro caso devuelve `false`.

Puedes comprobar si un actor colisiona con otro haciendo uso del método `collide` de la clase `CollisionDetector`: `CollisionDetector.collide(actor, player);`

Clase Actor

Esta clase es abstracta y es la base de cualquier actor del juego. Todos los objetos tienen una textura (el “disfraz” que no es más que una imagen PNG), un tamaño (ancho y alto) y se encuentra en una posición (x, y).

Debes tener clase que el juego es un eje cartesiano (un cuadrado de píxeles) donde la posición (0, 0) está arriba a la izquierda.

Sus métodos son *getter* y *setter* que no necesitan más explicación.

Clase Player

Esta clase representa el jugador.

a) Constructor

El constructor inicializa `posX` y `posY` a `0` y el atributo `texture` al valor `"player.png"`.

b) setPosition

Este método modifica `posX` y `posY` con los valores (x, y) recibidos y el ancho y alto (`width` y `height`) con el número `50`.

Clase Obstacle

Esta clase representa un obstáculo en el juego. Solo tiene un constructor que recibe los valores (x, y) con los que inicializar los atributos `posX` y `posY`.

Además, en el constructor da el valor `40` a los atributos `width` y `height`.

Por último, el atributo `texture` inicialízalo con el valor `"brick.png"`.

Clase Canyon

Esta clase representa el cañón desde el que se lanzan las bolas de fuego (enemigos).

Esta clase añade a sus atributos un radio (radius) porque es un círculo.

a) Constructor

El constructor inicializa el atributo `radius` con el parámetro recibido.

El `width` y el `height` es igual al doble del `radius`.

b) setPosition

Este método sirve para dar valor a los atributos `posX` y `posY` con los parámetros (x, y) recibidos.

c) getRadius

Un simple *getter* para devolver el atributo `radius`.

d) getCenterX

Devuelve la posición x del cañón a partir del `radius`. ¿Cómo se obtiene? Sumando a `posX` el valor del `radius`. Así pues, solo devuelve la suma de estos dos atributos.

e) getCenterY

Igual que `getCenterX` pero para `posY`. Aquí también hay que devolver la suma de los atributos `posY` y `radius`.

Clase ObstacleFactory

Es una clase muy sencilla con un único método estático que se usa para crear obstáculos (objetos de la clase Obstacle).

El método, como ves en el UML, se llama `buildForGame` y recibe el objeto `Game` en sí y un número (`count`) que indica cuántos objetos `Obstacle` hay que crear.

Dicho método tiene que crear dicho número de obstáculos pero llevando cuidado que no colisionen. Las posiciones de los obstáculos serán al azar.

Para crear un `Obstacle` tienes que pasar la posición `x` y la posición `y` del obstáculo. Estos valores (x, y) los generás al azar.

Estos valores (x, y) tienen que estar dentro del escenario del juego. Puedes saber el tamaño máximo para `x` porque está en `game.getWidth()` y el máximo para `y` que está en `game.getHeight()`.

Puedes saber si el obstáculo creado colisiona con otro actor del juego llamando a `game.collide(obstacle)`.

Te escribo el código que genera números al azar dentro de un bucle:

```
int x;
int y;
Obstacle obstacle;

long seed = System.currentTimeMillis();
int seed2 = 1;
Random random;

while (...) {
    random = new Random(seed + seed2);
    x = random.nextInt(game.getWidth() - (int)(game.getWidth() * 0.1));
    y = random.nextInt(game.getHeight() - (int)(game.getHeight() * 0.1));

    obstacle = new Obstacle(x, y);

    seed2++;
}
```

Este es, básicamente, el código a completar.

Clase CollisionDetector

Esta clase te la doy yo implementada:

```
public class CollisionDetector {

     private static boolean isBetween(int x, int x0, int x1) {

           return x >= x0 && x <= x0 + x1;

     }


     public static boolean collide(Actor a, Actor b) {

           return (isBetween(a.getX(), b.getX(), b.getWidth()) &&
           isBetween(a.getY(), b.getY(), b.getHeight())) ||

                 (isBetween(a.getX() + a.getWidth(), b.getX(),
     b.getWidth()) && isBetween(a.getY(), b.getY(), b.getHeight())) ||

                 (isBetween(a.getX(), b.getX(), b.getWidth()) &&
isBetween(a.getY() + a.getHeight(), b.getY(), b.getHeight())) ||

                 (isBetween(a.getX() + a.getWidth(), b.getX(), b.getWidth())
&& isBetween(a.getY() + a.getHeight(), b.getY(), b.getHeight()));

     }

}
```

Enumeración

Crea la enumeración indicada en el UML.