# Ejercicio 3. Entidades.

La idea del ejercicio es añadir un grado mayor de estructura al código del ejercicio anterior introduciendo clases y managers, el cual servirá como base para la realización del próximo ejercicio el cual tendrá como finalidad la inclusión de componentes. Básicamente es crear clases que incorporen el código ya visto en el anterior ejercicio.

Ejercicio 3.1.

La idea es transformar las partes del código que tengan entidad propia en clases.

La aplicación muestra una serie de bolas chocando entre sí, por lo que la entidad principal será la Bola. Transformar el código del anterior ejercicio y crear una clase que contengan las variables miembros y métodos asociados a la entidad Bola.

Las variables miembros contendrán características de las Bolas como el radio, la posición, etc.

Las funciones miembros han de tener funciones de acceso y asignación de las variables miembro cuando sean necesarias (p. ej. SetRadius y GetRadius), así como una función Slot, la cual se llamará desde el bucle de lógica para cada bola, y que actualizará la posición y la velocidad de la bola dependiendo de colisiones tal cual se ha visto en el anterior ejercicio.

Una vez transformado comprobar que el resultado de ejecutar el código no ha variado.

Ejercicio 3.2.

La idea es crear una clase que contenga la funcionalidad asociada a la gestión de tiempo. La clase contendría las funciones y variables vistos en el anterior ejercicio vinculadas a control de tiempo. Dicha clase podría incorporarse a posteriori en cualquier subsistema para controlar cuando se ha de ejecutar éste.

El código de la función LogicSlot() del anterior ejercicio debería transformarse en algo como:

void LogicSlot()

{

// Calls to Timer to process logic slot.

m\_Timer.InitSlotsToProcess();

while (m\_Timer.ProcessSlots()) {

// Call to world logic.

LogicWorldSlot(g\_fFixedTick);

}

}

Como se observa hay dos métodos miembro principales en la clase Timer. InitSlotsToProcess() se encargaría del cálculo ha realizar en cada frame. ProcessSlots() se encargaría de determinar cuando se ha de ejecutar la lógica.

Como variables miembro se tendrían las variables necesarias del anterior ejercicio como g\_fElapsedTime o g\_PreviousTime.

Ejercicio 3.3.

Crear el manager de lógica. El manager será una clase Singleton (una clase en la que sólo se permite una instancia) que contendrá los elementos del juego como p. ej. la lista de bolas. Básicamente contendrá las funciones y variables del fichero logic.cpp que no hayamos movido en los anteriores ejercicios a otras clases.

Ejercicio 3.4.

Crear el Render Engine. Será un Singleton que contendrá la funcionalidad de pintado en pantalla así como toda la interacción con OpenGL. Básicamente contendrá la funcionalidad vista en render.cpp.

El Render Engine es un sistema genérico de renderizado que no depende de un juego concreto, por lo que no tiene que acceder a funciones propias de la lógica. La idea básica es que el Render Engine proporciona clases que le son propias y que la lógica tendrá que registrar, derregistrar y actualizar (p.ej. la clase Sprite).