Sistemas Gráficos

PRÁCTICA 2
OLYMPUS RACING: RUSSIAN EDITION
DISEÑO DE LA APLICACIÓN

Tercer Curso - Grado en Ingeniería Informática
Curso 2019-2020

JAVIER LORENZO GARCÍA
PEDRO SERRANO PÉREZ

Contents

1 Introducción

Olympus Racing: Russian Edition, la aplicación que hemos decidido desarrollar, se trata de un juego de carreras reducido, en el que el jugador podrá elegir entre una serie de circuitos y vehículos (chasis y ruedas) para realizar una carrera (de dos vueltas) por el circuito elegido. Se cronometrará el tiempo que el jugador necesite para terminar la carrera.

En este documento vamos a explicar los aspectos más importantes sobre el diseño de la aplicación y los distintos elementos de esta.

Antes de presentar el diseño general de la aplicación, y de explicar los diferentes componentes de este, se debe mencionar que los circuitos, chasis y ruedas han sido realizados mediante **blender**, por los integrantes del grupo. En secciones posteriores explicaremos de forma más detallada este punto.

2 Diseño de la aplicación

En esta sección podemos encontrar el diagrama de clases de diseño de la aplicación y algunos aspectos de las clases más importantes que han de tenerse en cuenta para comprender el diseño que se seguido para el desarrollo de esta.

2.1 Diagrama de clases

En esta sección presentamos el diagrama de clases de diseño de la aplicación (Figura: 1). Como se puede ver en el diagrama hay 3 clases principales que representan la escena (MyPhysijsScene), el circuito (Circuito) y el vehículo (Coche). La clase MyModels actua como almacen de datos de los distintos modelos (y una imagen de vista previa que se mostrará en el menú de selección).

Como se trata de un diagrama de clases, solo se han representado las clases en sí y las relaciones entre ellas, pero se deben tener en cuenta los siguientes archivos/elementos de la aplicación que no son clases:

- Script.js - Se encarga de crear, y renderizar la escena, y de obtener los parametros *get* de la url.
- Cronometro.js - Permite iniciar un contador que actualizará un cronómetro en la partida que representa el tiempo que ha pasado el jugador en la carrera desde que esta comenzó.

2.2 Estructura general de la aplicación

La aplicación se encuentra dividida en tres partes o pantallas:

- La primera de ellas es un menú de selección. En este menú podremos un circuito, un chasis, y unas ruedas, entre las distintas alternativas de cada uno (Actualmente hay 2 circuitos, 2 chasis y 2 tipos de ruedas).

  Como hemos visto en el diagrama de clases la clase MyModels sirve como almacen de modelos, por lo que, para añadir un nuevo modelo (circuito, coche o ruedas), solo es necesario añadir la ruta del modelo en esta clase, junto con una imagen de vista previa que se mostrará en el menú de selección. Esto aporta una gran facilidad para la integración de nuevos circuitos y elementos de personalización del vehículo (chasis y ruedas).

- Una vez se seleccionen las distintas opciones (si en algún caso no se selecciona una opción, por ejemplo el circuito, se tomará por defecto la primera de las opciones) se redireccionará al jugador a la carrera en sí. Esto último se realiza medienta una petición http de tipo get(utilizando formularios html y javascript), en la que se envían los datos

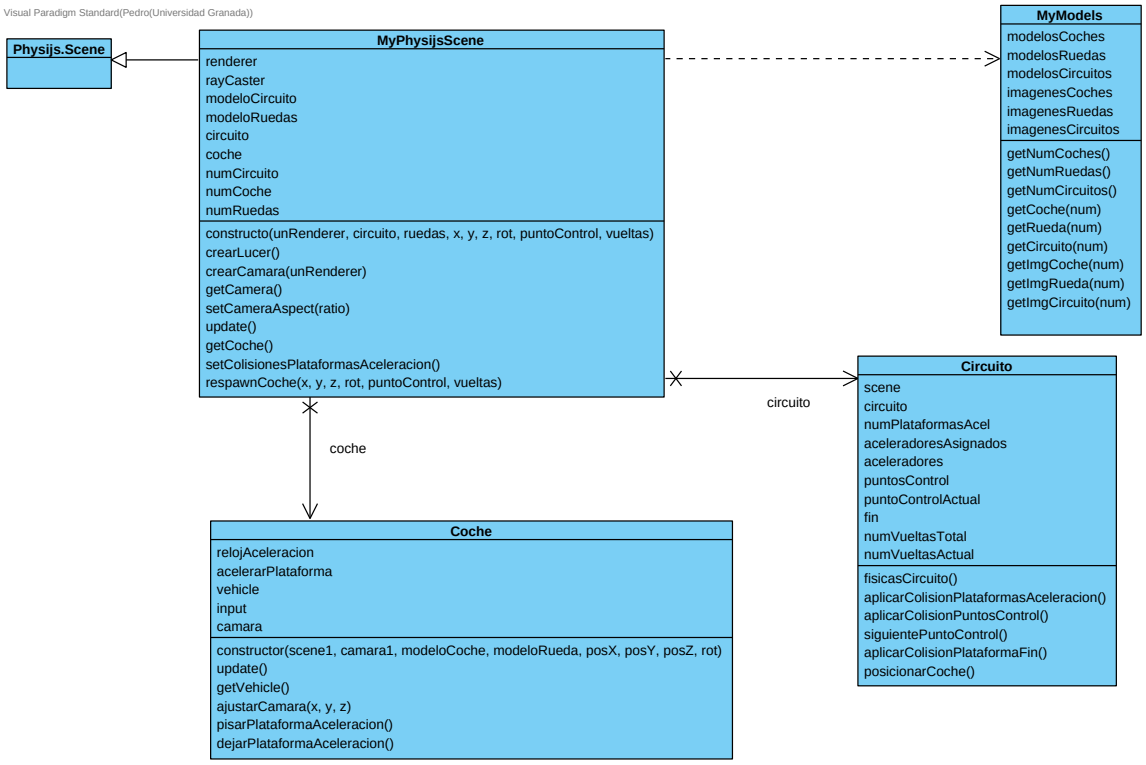

Figure 1: Diagrama de clases de diseño

necesarios sobre las elecciones del jugador, que se obtendrán en el fichero "script.js" mediente la lectura de la *url*.

- Una vez el jugador finaliza la carrera este será redirigido a una pantalla de resultados, donde podrá ver el tiempo que ha empleado para completar las dos vueltas al circuito. La redirección se realiza de forma similar a la comentada en el punto anterior. En esta pantalla de resultados se podrá volver, pulsando sobre un botón, al menú de selección.

NOTA: Se han utilizado peticiones http de tipo "GET" debido a que el servidor no admite peticiones http de tipo "POST".

2.3 Clase MyPhysijsScene

Esta clase representa la escena. Principalmente se encargará de crear, una vez se seleccionen los elementos de la partida (circuito, chasis y ruedas), el circuito, coche, luces y cámara.

La cámara sigue al vehículo desde atrás, por lo que en algunas rampas, estas se interponian entre la cámara y el vehículo, por lo que, para evitar esto, se ha reducido la propiedad "near" de la camara, por lo que, aunque siempre falte una pequeña porción en la parte inferior de la pantalla, la visión no se ve reducida en las rampas.

El método setColisionesPlataformasAceleracion sirve como método de ayuda para que, mientras el vehículo pise una plataforma de aceleración, aumente la aceleración del coche. El método respawnCoche servirá para hacer que el vehículo vuelva al último punto de control que ha pisado. Estas funcionalidades hacen uso, o son llamadas por métodos de otras clases, por lo que las funcionalidades se explicará de forma completa más adelante.

2.4 Modelos

Antes de profundizar en las clases de coche y circuito debemos comentar algunos aspectos sobre los modelos, debido a que estas clases hace uso de los modelos para obtener la geometría y material.

Los modelos los hemos creados de forma manual mediate blender (en la carpeta de modelos, a parte de los modelos en formato .glb, se adjuntan los modelos de blender que hemos creado). Para cargar los modelos hemos utilizado GLTFLoader, de Three.js, debido a que nos encontramos con dificultades, para cargar de forma correcta, la geometría de los modelos en formato .obj, debido a que el orden de los vértices no se mantenía una vez cargados mediante OBJLoader de Three.js. GLTFLoader carga la escena que hay en un archivo (.glb en nuestro caso), por lo que, de la escena cargada, obtenemos la geometría y el material que se le aplica, además de otra información, como el nombre de un mesh, que será muy útil para identificar las distintas partes de un circuito, como veremos en una de las siguientes secciones.

2.5 Clase Coche

Para la creación de un coche, junto con sus físicas y controles, hemos utilizado **Physijs.Vehicle**, que recibiría un chasis, que se añade a la escena, y cuatro ruedas, por lo que separa el chasis de las ruedas (estos elementos, como se ha mencionado, se obtienen cargando un modelo .glb. Una de las principales ventajas de Physijs.Vehicle es la posibilidad de especificar, cuando se crea el vehículo, aspectos como la suspensión, fricción de las ruedas, etc. Además, posee facilidades para establecer la aceleración, giro de las ruedas, frenos o la tracción del vehículo.

El problema de Physijs.Vehicle viene de la poca documentación y mantenimiento de esta, por ejemplo hay un error al añadir un coche a la escena debido a que aún no posee ruedas, pero las ruedas deben añadirse una vez el vehículo está en la escena. Además la funcionalidad que elimina el vehículo de la escena no funciona de manera adecuada (más adelante comentaremos en profundidad este problema, debido a los cambios que se han tenido que realizar una vez detectado este fallo).

2.6 Clase Circuito

La creación del circuito, como se ha mencionado anteriormente, se realiza cargando un modelo glb. El uso de blender para la creación de los circuitos, permite, además de facilitar la creación de un circuitos algo "complejos" (en cuanto a forma), establecer nombres para los distintos *mesh*. Teniendo en cuenta este último punto tenemos varios tipos de mesh en nuestros circuitos, por lo que, para añadir un nuevo circuito, este debe crearse teniendo en cuenta los siguientes puntos:

- El rozamiento base es de **0.1**, por lo que, si no se especifica otro para el "tipo de mesh", se le aplicará el rozamiento base, por ejmplo para los mesh que incluyan la subcadena "Circuito".
- Los mesh que incluyan la subcadena "Terreno" tendrán una fricción mayor, de **0.5**.
- Los mesh que incluyan la subcadena "Aceleracion" no tendrán rozamiento. Además, se tendrán en cuenta las colisiones para incrementar la aceleración del vehículo, como se ha mencionado anteriormente. Para aplicar las colisiones, como afectan de forma directa al vehículo, tanto el circuito, como el vehículo se deben haber cargado, por lo que se utiliza la función, relacionada con las colisiones de este tipo de plataformas de la clase que representa la escena, que es la clase intermedia que conecta el circuito y un vehículo.
- Los mesh que incluyan la subcadena "Fin" indicarán el fin del mundo en el que se desarrolla la carrera, por lo que si el vehículo que controla el jugador cae una plataforma fin este será enviado al último punto de control que ha pisado (en los dos que se adjuntan solo hay una plataforma de este tipo, a la cual el jugador puede caerse, pero podría haber varias que actuaran como obstáculos, etc).
- Los mesh que incluyan la subcadena "PuntoControl", como su nombre indica, servirán para marcar el progreso del vehículo en la carrera. El primer punto de control que se cargue será la meta (Si hay dos puntos de control, por ejemplo, PuntoControl0 y PuntoControl1, debido a que PuntoControl0 es el menor de los dos, en blender, estará posicionado antes en el vector de mesh, por lo será considerado como meta).

  Si el jugador se cae (mesh tipo "Fin") será enviado al último punto de control. Además, el jugador podrá volver al último punto de control pulsando la letra "B".

La aplicación la físicas del circuito se realiza mediante el método físicasCircuito(...) de la clase Circuito. Las físicas se realizan por el método propuesto por el profesor para aplicar físicas a modelos cargados. El principal inconveniente de esto es que el circuito está compuesto por rectángulos de distintas dimensiones que forman el circuito, debido a que esto permite una aplicación más exacta de físicas, y una mayor generalización para cargar distintos circuitos.

2.7 Problemas debido al uso de Physijs.vehicle

Como ya se ha mencionado en la parte de diseño de la clase Coche, se ha utilizado Phisijs.Vehicle para la creación de un vehículo compuesto por un chasis y cuatro ruedas. La principal dificultad que hemos encontrado debido al uso de Phisijs.Vehicle ha sido en el desarrollo de la funcionalidad de los "Puntos de control", los cuales permiten al jugador volver al último punto de control si se cae del circuito o no puede continuar el circuito, por ejemplo si el vehículo vuelca.

Inicialmente tratamos de desplazar el objeto físico que devuelve el constructor de Phisijs.Vehicle, pero no es posible posicionarlo, por lo que tratamos de hacer lo mismo con el mesh de este, pero la actualización no se producía de forma correcta, debido a que el chasis se separaba de las ruedas.

En este punto tratamos de eliminar el vehiculo de la escena (limpiando el mayor número de referencias posibles) y creando uno nuevo en la posición deseada, pero aún haciendo esto el heap no se liberaba por completo.

Por estas razones, y a que, debido al tiempo restante, ya no era posible dejar de utilizar Physijs.Vehicle para implementar nuestro propia estructura de vehiculos (similar a la que nos proporciona phisijs), decidimos volver a ejecutar el script que crea y renderiza la escena cuando se activaba la vuelta a un punto de control. Como ya se ha explicado en la estructura de la aplicaición hacemos uso de peticiones http de tipo "GET" para el envío de cierta información, por lo que, para volver al anterior punto de control se realiza una petición de tipo get que envía la información mínima para mantener el estado de la partida (número de vueltas y tiempo empleado), junto con la posición del punto donde debe reaparecer el vehículo. Una vez realizada la petición, el archivo script.js es el encargado de obtener los parametros y generar la escena que mantiene el progreso.