GO-Engine

por Daniel Herzog Cruz

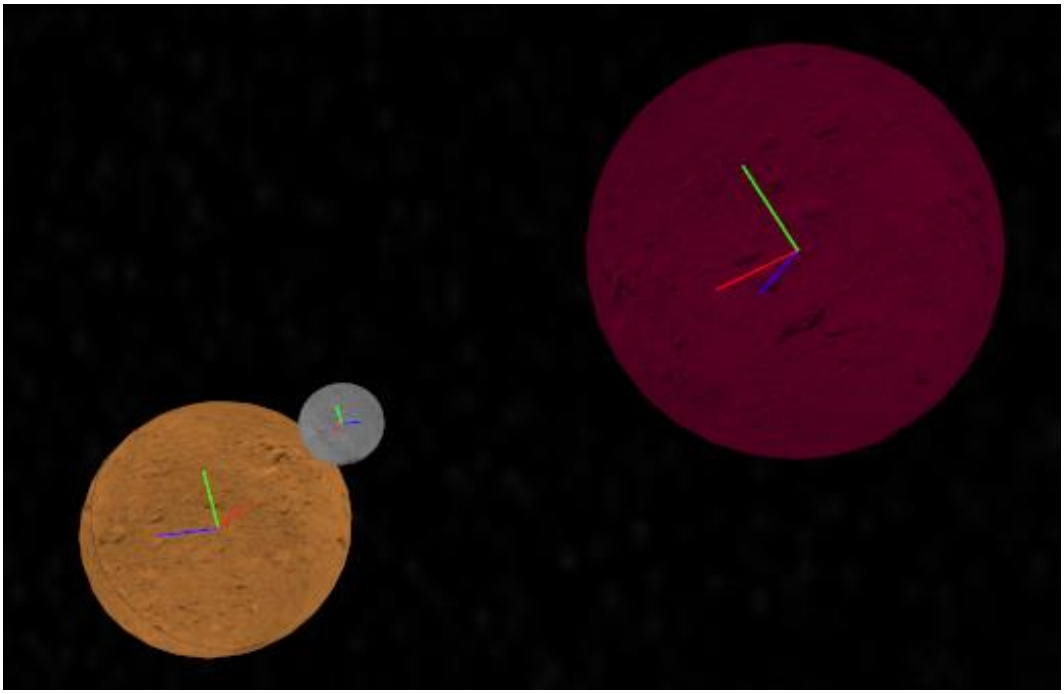
Go-Engine es un framework de un motor gráfico multiplataforma para desarrollar videojuegos, y cualquier tipo de aplicación de un carácter similar (simuladores, herramientas de investigación…). Dicho framework estará dividido en dos partes, una del usuario (carpeta *user*), usada para definir los objetos de juego y otros detalles para el *contenido* del juego, y una parte del motor (carpeta *engine*), usada para contener todo detalle relacionado con el *core* del motor gráfico, siendo aquello común a todos los proyectos desarrollados con el framework.

La idea del proyecto surgió al acabar el primer proyecto presentado el año pasado (2012-2013) al VII Concurso de Software Libre, llamado *Pong++*. Viendo el susodicho juego ya terminado, me di cuenta de que el siguiente paso era hacer algo bien hecho, algo que no fuese código cableado y difícil de mantener, algo que funcionase correctamente, algo que fuese un verdadero reto para aprender y entretenerme. De ahí surgió la idea de realizar un motor gráfico, una herramienta de utilidad para personajes que quisieran de hacer, de una manera ciertamente correcta, un proyecto como si de un *videojuego* se tratase.

La primera decisión fue elegir las características del proyecto. La primera y más importante es que se trata de software libre y multiplataforma, por lo que puede (y debe) ser funcional en la mayoría de las plataformas (Windows, Linux y Mac, básicamente). Además, el principal aspecto del motor no reside en lo anterior, sino en cómo está orientado, tal como dice su nombre, *GO-Engine* o *Game Object Engine*, que se basa en objetos de juego y componentes, que pueden ser creados, modificados y borrados en tiempo de ejecución (dinamismo).

Para entenderlo de manera sencilla, un *Game Object* u objeto de juego (o simplemente objeto) representa una abstracción de una realidad (por ejemplo, un coche), mientras que los componentes definen las características de un objeto de juego (por ejemplo, una carrocería (modelo 3d), una posición, ciertos sonidos, etc. Además, existe una jerarquía entre los objetos de juego, de tal manera que haya una relación de paternidad de uno a muchos (un objeto puede tener varios hijos, pero sólo puede tener un padre). Por ejemplo, supongamos un coche, que será el objeto padre “coche, y este tendría varios hijos, como un “eje_delantero”, un “eje_trasero”, y por consiguiente, cada eje tendría 2 ruedes. Para este caso, cada objeto tendría unas características (posición, rotación, modelo 3D, color…).

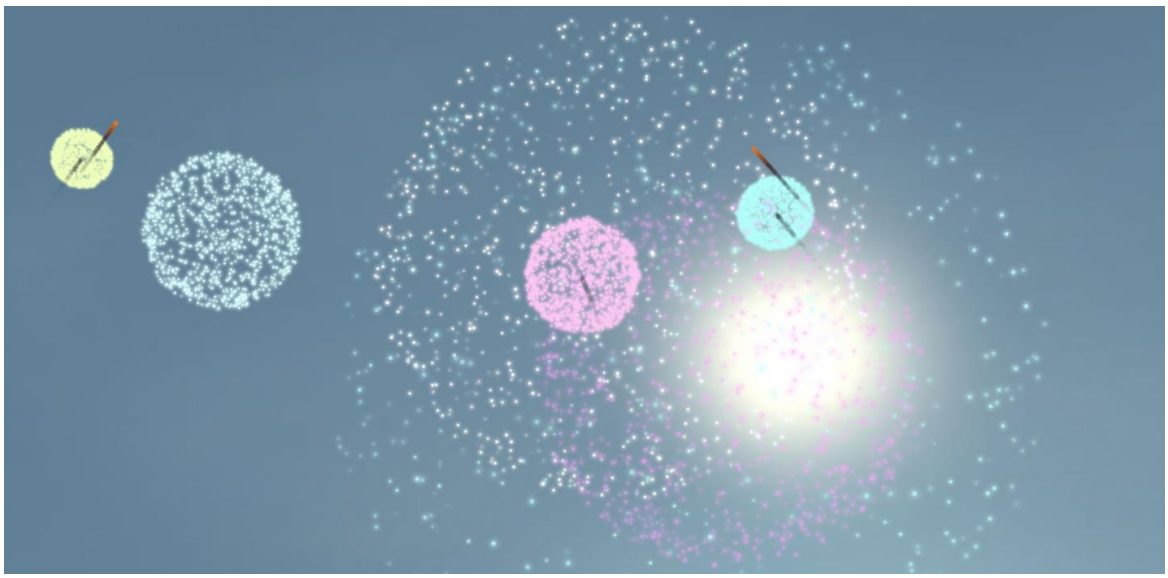
Otro caso podría ser el de un sistema solar, en el que gira una estrella o planeta, y a su alrededor giran varios planetas, y a su vez, giran en torno a estos otros planetas (además de girar sobre sí mismos).



Jerarquía sencilla de planetas.

La utilidad de esta jerarquía es, por ejemplo, al cambiar la posición del coche, se moverán automáticamente todo sus hijos, de manera recursiva, tal que se muevan todos en conjunto. También permite activar y desactivar un objeto y todos sus hijos, creando agrupaciones lógicas de objetos para poder trabajar con conjuntos, y no con objetos sueltos y desperdigados.

A un nivel superior a los objetos, se encuentran los sistemas, cuyo objetivo puede ser trabajar con objetos (sistemas de control, como el *render*, el *manager* de objetos..) o proporcionar un cierto servicio a los objetos o a otros sistemas (sistemas auxiliares, como el sistema *time*, *math*, *data storage*…). Se usan principalmente para mantener, de manera modular, las distintas características que debe cumplir el motor gráfico (gráficos 3D, sonido, redes, shaders, persistencia de datos…).



Sistema Render dibujando unos fuegos artificiales (partículas), mientras el sistema Mixer emite los sonidos de las explosiones.

Por supuesto, habrá una serie de estancias o niveles definidas en el juego, que tendrán una serie de recursos (texturas, modelos, sonidos) y un conjunto de objetos de juego, con sus respectivos componentes, lo que estructurará el juego según quiera el usuario (un escenario para un menú, otro escenario para la pantalla de juego, otro escenario para el menú dentro del juego…)

Una vez definida esta característica primordial (basado en objetos, componentes y sistemas), se definieron algunas propiedades que el proyecto debería cumplir, siendo gráficos 3D, sonido 3D, gestor de ventanas, redes, física en tiempo real y una interfaz gráfica de usuario para editar fácilmente los escenarios.

Para ello, se han utilizado librerías como *OpenGL* (gráficos 3D), *OpenAL* (sonido), *SDL2* (ventanas y eventos de teclado, ratón, joysticks…), *GLM* (álgebra matricial), entre otras (mencionadas en la página principal de la forja del proyecto).

Por desgracia, no todos los objetivos pudieron ser cumplidos (faltan los 3 últimos por realizar), puesto que el proyecto sufrió algunos problemas con respecto al equipo de trabajo, y al final acabó trabajando un único participante.

No obstante, lo verdaderamente importante ha sido la experiencia del trabajo realizado, durante un total de 7 meses (desde octubre de 2013 hasta abril de 2014), con todo tipo de conocimientos adquiridos en torno a este campo, y, por si fuera poco, el proyecto es funcional de por sí, y todo lo implementado funciona correctamente, por lo que se podría utilizar para hacer proyectos sencillos, o incluso maquetas.

Con respecto al futuro, me haría gran ilusión poder continuar el proyecto, el único inconveniente es que es demasiado trabajo para una única persona, por lo que me gustaría disponer de un pequeño equipo de trabajo, o colaboradores, capaces de aportar componentes, sistemas, demos, documentación… al proyecto.

Para más información sobre el proyecto desarrollado, puede visitar:

- **Blog:** http://goengine.wordpress.com/
- **Forja:** http://github.com/wikiti/go-engine
- **Página CUSL:** http://www.concursosoftwarelibre.org/1314/proyectos/20