Introducción a la Programación de Videojuegos y Gráficos

GALACTIC STRIKE

Realizado por:
Eduardo Radío Gallego, eduardo.radio@estudiante.uam.es
Borja Mauricio Fourquet Maldonado, borja.fourquet@estudiante.uam.es

Fecha: 18/12/2015

Introducción a la Programación de Videojuegos y Gráficos

1. Descripción general del proyecto

En esta sección se deben especificar los objetivos generales del proyecto, así como una descripción del equipo disponible y el entorno de desarrollo que se va a emplear. Se describen las principales técnicas a emplear. Se describirá de forma general la funcionalidad del proyecto, posibles requisitos hardware, así como cualquier otra descripción o recurso adicional que sea de interés a un nivel más general.

Este punto se podría subdividirse (entre otras) en las siguientes secciones:

1. Introducción

El objetivo principal del proyecto es el de desarrollar un videojuego de navegador en línea, novedoso y divertido, aunando características de diversos tipos de juegos.

2. Descripción general del producto (en este caso un videojuego).

Galactic Strike es un juego multijugador de peleas por equipos en el espacio. Los jugadores elegirán o crearán un escenario con un número determinado de jugadores, seleccionarán uno de los distintos personajes disponibles, se unirán a un equipo y competirán contra otros en distintos modos de juego. Durante el combate aparecerán distintos objetos aleatoriamente que los personajes podrán utilizar a su favor para dominar la partida.

3. Funcionalidad (reglas del juego, objetivos, mundo, etc).

El juego será esencialmente **PVP** *(Player Versus Player).*

En el principal modo de juego el objetivo de cada equipo será aniquilar a los miembros del equipo contrario. A medida que se avance con el desarrollo del juego, podrían incluirse distintos formatos de partida (aniquilación por rondas, capturar la bandera, todos contra todos, etc.).

El mundo en el cual se llevarán a cabo estas batallas estará compuesto por diversos astros con distintas características. Los jugadores tendrán que saltar de un planeta a otro para recolectar objetos, alcanzar a sus enemigos, huir, etc. Un escenario estará compuesto por un conjunto de astros estáticos, que normalmente generarán un campo gravitatorio que atraerá a cada objeto o personaje que entre en él, además de otro posible conjunto de astros móviles espontáneos, como meteoritos o agujeros negros, que jugarán un papel de enemigo neutral.

4. Dominio, o género, de desarrollo del proyecto.

Se trata de un videojuego plataformas en 2D, con ideas extraídas del clásico Counter Strike [3], pero utilizando personajes más propios de un juego de rol, y todo ello en un escenario más propio de un "Space Ship Game" [4] o juego de naves espaciales.

5. Entorno y herramientas de desarrollo.

Los entornos de producción del software son nuestros ordenadores personales de sobremesa, con **Windows 10 Home** y **Ubuntu 15.10**. Usaremos el IDE ***Brackets***, y probaremos el software en el navegador **Google Chrome** (en la última versión estable que exista en el momento subida de cada uno de los entregables. El juego se codificará en **JavaScript** sobre **HTML5**. Se usará el *framework* **Phaser** tanto en el lado del cliente como en el del servidor para gestionar las físicas, los gráficos, etc. Además, para gestionar las conexiones, las llamadas a métodos remotos y demás se utilizará el *framework* **Socket.io** que utiliza **NodeJS**.

6. Descripción del hardware.

El videojuego se desarrollará principalmente para los ordenadores de la EPS, los cuales en su mayoría son de gama media-baja y cuentan con procesadores de doble núcleo.

El cliente debería de poder ejecutarse en cualquier ordenador portátil o sobremesa que tenga instalado el navegador, independientemente de la arquitectura hardware del mismo.

Para el servidor actualmente se está utilizando para las pruebas un PC con un procesador Intel i7 4790K

7. Equipo y lugar de trabajo.

El equipo está compuesto por dos personas, que realizarán labores tanto de diseño como de implementación:

Eduardo Radío Gallego: eduardo.radio@estudiante.uam.es
Borja Mauricio Fourquet Maldonado: borja.fourquet@estudiante.uam.es

8. Técnicas a emplear.

Utilizaremos una metodología ágil basada en el archivo adjunto GS-CR-vX.xlsx, que se irá completando a lo largo del desarrollo.

Los miembros del equipo se irán autoasignando requisitos de este documento según su prioridad, y al final volverán a seleccionar otro requisito que llevar a cabo. Los requisitos "extra" los proponemos para la entrega final en el caso de que hubiésemos completado todos los requisitos anteriores ("prototipo" y "final").

9. Recursos adicionales (por ejemplo, gráficos, sonidos, etc).

Para la primera versión utilizaremos mayormente sprites y sonidos de terceros, aunque si se da bien para la última entrega, se pretende incorporar gráficos y audios de elaboración propia.

Los efectos de sonido del prototipo se han creado modificando y añadiendo efectos al archivo *public/assets/sound/pingas.mp3*

2. Gestión del Proyecto

En esta sección se deben mostrar las planificaciones temporales de desarrollo del proyecto en su fase de inicio y de elaboración, así como el diario de ejecución del proyecto, junto con el diario de construcción de la aplicación y cumplimiento de los plazos estimados.

El proyecto se encuentra en un repositorio privado creado en *BitBucket* [2], que utiliza *Git* como software de control de versiones. Para acceder a este repositorio desde nuestros entornos de producción, hemos utilizado el plug-in *Brackes Git* para *Brackets*, que proporciona una GUI cómoda e intuitiva para ejecutar la mayoría de los comandos de *Git*.

BitBucket proporciona además un sistema de gestión de incidencias para cada repositorio. Se ha utilizado este sistema para crear informes de bugs, de mejoras posibles, tareas pendientes, etc, y se han ido marcando como resueltas cada una de estas incidencias a medida que se han llevado a cabo.

En el archivo **README.md** se encuentran los detalles sobre cómo desplegar el servidor y acceder al juego.

1. ***Entregables*** (en el caso de la asignatura 3, incluyendo este documento[1]) y fechas de los mismos.

   - ***Práctica 1, Domingo 1 de Noviembre de 2015:***

     - *Documento de Análisis y Diseño (este mismo documento): GS-DAyD-v1.0.pdf*
     - *Diagrama de Clases: GS-DC-v1.0.pdf*
     - *Diagrama de Estados: GS-DE-v1.0.pdf*
     - *Catálogo de Requisitos: GS-CR-v1.0.xlsx*

1 En el caso de este proyecto, la fecha límite del primer entregable será el **1 de Noviembre de 2015**

- ***Práctica 2, Viernes 18 de Diciembre de 2015:***
  - *Documento de Análisis y Diseño (este mismo documento): GS-DAyD-v2.0.pdf*
  - *Diagrama de Clases: GS-DC-v2.0.pdf*
  - *Diagrama de Estados: GS-DE-v2.0.pdf*
  - *Manual de Usuario: Manual-v1.0.html*
- ***Práctica 3:***

2. ***Contenido*** de cada entregable. Tanto en forma de documentación, como de software o prototipos a mostrar en cada uno de ellos.

Matriz de trazabilidad de cambios

Fecha	Identificación del Cambio	Documento de Análisis y Diseño	Diagrama de Clases	Diagrama de Estados	Catálogo de Requisitos	Manual de Usuario
1/11/2015	Entrega práctica 1	GS-DAyD-v1.0.docx	GS-DC-v1.0.html	GS-DE-v1.0.html	GS-CR-v1.0.xlsx	
18/12/2015	Entrega práctica 2	GS-DAyD-v2.0.docx	GS-DC-v2.0.html	GS-DE-v2.0.html		UserGuide/Manual-v1.0.html
	Entrega práctica 3					

3. Análisis/Diseño

Para este proyecto se utiliza programación orientada a objetos, la cual es soportada por JavaScript con ciertas dificultades.

De momento se ha decidido que no exista persistencia de datos sobre los usuarios ni las partidas, por lo que no se requiere del uso de una base de datos para este proyecto.

Como se ha explicado antes, el juego es on-line. Los jugadores podrán unirse a una sala, esperar a que lleguen más jugadores y comenzar la partida. Por motivos de simplicidad de la infraestructura de red, se ha decidido que (de momento) sólo pueda haber una sala y una partida. Por tanto, los usuarios podrán unirse a la sala, lo que les llevará al lobby y se quedarán allí chateando con los jugadores que haya en la sala hasta que el host empiece la partida o crear una sala si no hay ninguna creada en el momento, lo que hará que el jugador que ha creado la sala sea el host.

Más detalles sobre la transición de estos estados en el correspondiente ***Diagrama de Estados*** indicado en la matriz de trazabilidad de cambios.

Se ha elaborado un diagrama de clases que involucra a las referencias que se hacen dentro del propio estado de la partida. Se muestra **Match** como clase principal, aunque en realidad una clase con ese nombre (como más adelante se comenta, esta clase se corresponde con el estado *'Play'* añadido a los estados de *Phaser*).

Una partida pertenece a una sala (**Room**), la cual tiene una serie de equipos (**Team**), que a su vez tienen jugadores (**Player**). Cada jugador tiene asignado un personaje al cual controla desde su cliente.

Una partida también tiene un modo de juego (**GameMode**), que es la clase en la que se definen las reglas del juego (cuándo finalizan las rondas, cuándo los equipos ganan puntos, cuándo finaliza el juego, quién es el ganador del juego, etc).

También tiene un escenario o nivel (**Stage**), en el cual están definidos las posiciones de ciertos elementos propios de cada uno. También se encarga de gestionar la posición en la que “nacen” cada uno de los personajes al empezar cada una de las rondas y del escalado del mundo al hacer zoom. Durante la partida, los datos sobre transcurso de la partida se muestran en pantalla gracias a la clase **HUD,** que se encarga de acceder a los datos el modelo y representarlos en la pantalla.

Los elementos que se encuentran en la partida heredan de Phaser.Sprite, y a su vez se especializan, de momento, en tres clases:

Character : sprites que se mueven, atacan, reciben golpes(por lo que tienen stamina), etc.

Item : distintos objetos del juego. Algunos son recolectables y utilizables, otros hacen daño al tocarlos…

Star : hace referencia a cualquier astro que pueda haber en el juego, tales como planetas, estrellas, cometas, etc.

El ***Diagrama de Clases*** actualizado se encuentra en el fichero indicado en la matriz de trazabilidad de cambios.

En esta sección se describirán los modelos de análisis/diseño (diagrama de clases, si se usa programación orientada a objetos, patrones de diseño) como el modelo de datos (e.g. modelo entidad - relación), desde los cuales se puede consultar la especificación de los métodos de clase más relevantes o las especificaciones de atributos.

También es deseable proporcionar un diseño inicial (aunque no sea definitivo) de una propuesta de diseño de clases (interfaces, clases abstractas, patrones de diseño a utilizar,…) que pudiesen emplearse para el posterior desarrollo del proyecto.

4. Implementación

En el directorio ***docs/*** se encuentran todos los archivos sobre la documentación del proyecto, incluido este archivo.

El directorio ***node_modules/*** se guardan las dependencias del proyecto de *NodeJS*. Por simplicidad, se conserva para no tener que instalarlas cada vez que se hace un clone de este proyecto.

El directorio ***public/*** del proyecto contiene todos los archivos que se envían al cliente cuando hace una solicitud HTTP al servidor desde el navegador.

A su vez, este directorio tiene dos subdirectorios

En ***public/assets/*** se encuentran todos los recursos multimedia que se utilizan dentro del juego.

En ***public/js/*** está todo el código JavaScript utilizado para el proyecto.

public/js/lib/ : aquí está *Phaser v2.4.4* y *Box2D*, que son los dos frameworks que estamos utilizando en el proyecto.

public/js/model/ : aquí se encuentran todos los archivos en los que se definen las clases que aparecen en el diagrama de clase adjunto.

public/js/states/ : aquí se definen todos los estados *(con game.states.add)* que tiene el juego, y que se muestran en el diagrama de estados adjunto.

public/js/network/ : en estos ficheros están implementados los handlers para los distintos eventos que pueden ser enviados por el servidor y recibidos en el cliente.

public/js/data/ : los archivos anteriores proporcionan un motor de juego. Cada uno de los archivos de este directorio define un objeto de JavaScript (al estilo *JSON*). Estos objetos tienen definidos dentro de sí los distintos elementos que hay en el motor de juego, indiciando sus atributos (mediante valores) y sus comportamientos (mediante funciones). En los constructores de las clases del modelo, se pasa como parámetro se pasa uno de estos objetos (*items[‘spikeball’]*, *stages[‘map1’]*, *gameModes[‘deatchmatch’]*, etc.), y se accede a estos valores y funciones para almacenarlas en un objeto de la clase Item, Stage, GameMode, etc.

Hemos utilizado como **motor de físicas** Box2D. Originalmente, este motor fue desarrollado para ser usado en C++, y la versión que incluimos es un port que se ha hecho recientemente. Es bastante parecido a P2, aunque la falta de documentación hace que su uso sea complicado. En un principio elegimos este motor porque encontramos un ejemplo que implementaba unas físicas de planetas bastante parecidas a las que nosotros pensamos [5]. A medida que hemos ido desarrollando, nos hemos encontrado con bastantes problemas con este motor, entre otros, de rendimiento. Estamos estudiando muy seriamente la posibilidad de **cambiar el motor de físicas por P2** una vez entregado el prototipo, ya que son bastantes parecidos y muy posiblemente éste último esté mejor optimizado para ser usado junto con Phaser.

Para gestionar la comunicación entre los distintos clientes y el servidor, hemos elegido utilizar **Socket.io**. Socket.io es un framework para NodeJS bastante sencillo de utilizar, que nos abstrae bastante de la parte de programación y nos proporciona un sistema de **comunicación basada en eventos**. Ya que no es el objetivo de este asignatura, se explica a continuación brevemente el uso de las distintas funciones del framework y dónde se encuentran estas llamadas:

- Cliente:

socket = io ();

Esta instrucción se encarga de conectarse al servidor y devolver un socket por el cual comunicarse con él. Esta línea de código se ejecuta en *Loader.js*, cuando se cargan todos los recursos.

socket.emit('event', data);

Envía al servidor el evento *'event'*, mandando además los datos almacenados dentro del objeto *data*. Estas llamadas se encuentran a lo largo de todos los estados, siempre que un jugador ejecuta una acción que debe ser notificada al resto de usuarios.

socket.on('event', handler);

Esta línea de código 'prepara' al cliente para recibir el evento *'event'* del servidor. A partir de este momento, cuando el cliente reciba este evento, la función *handler* se ejecutará ininterrumpidamente. Las llamadas a *socket.on* sólo se hacen una vez por cliente, y se encuentran en el directorio *public/js/network/*.

- Servidor:

La implementación servidor se comentará en la entrega final, junto con un diagrama de red en el que se explicará detalladamente el protocolo de eventos que se ha creado sobre el framework de NodeJS y la relación de cada uno de estos eventos con el juego.

> En esta sección se muestran, o se describen, los prototipos de interfaces de usuario de la aplicación, así como un diagrama de la funcionalidad añadida en cada fase de producción.

5. Pruebas

Se han realizado pruebas on-line cada 3 o 4 días al estilo *Early Access* gracias a la colaboración de algunos de nuestros compañeros de clase. En estas pruebas nos hemos conectado desde 3 hasta 6 jugadores para probar el correcto funcionamiento del protocolo de red y el rendimiento del proceso servidor por un lado, pero por otro también ha servido para probar el cliente Phaser en sí con distintos hardwares y sistemas operativos. Estas pruebas nos han servido para arreglar numerosos bugs que de otra forma probablemente habrían tardado más días en aparecer (si hubiesen aparecido), y también nos ha servido para concienciarnos de la importancia de la jugabilidad y del rendimiento del código. Por estos dos últimos motivos, estudiamos la posibilidad como se comenta anteriormente de cambiar el motor de físicas, ya que hay ciertos aspectos de Box2D que no nos permiten hacer unos ataques demasiado decentes y también sospechamos que el motor de físicas actual no es demasiado eficiente.

> Por último, se describirán las pruebas a realizar en cada iteración de la aplicación.

6. Referencias

[1]: https://es.wikipedia.org/wiki/Videojuego_de_plataformas
[2]: https://bitbucket.org/
[3]: https://en.wikipedia.org/wiki/Counter-Strike
[4]: http://es.y8.com/tags/spaceship - Ejemplos de Spaceship Games
[5]:http://www.emanueleferonato.com/2015/06/19/simulate-planet-gravity-with-phaser-box2d-as-seen-on-angry-birds-space/