NUEVOS PARADIGMAS DE INTERACCIÓN

PRÁCTICA 1: Interfaces con sensores

AlhambraApp

Autores:
Miguel Ángel Pérez Díaz
Juan Carlos Ortegón Aguilar
Lukas Haring García

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada,2019

ÍNDICE

1. Descripción general sobre el proceso de desarrollo de la aplicación

En nuestro caso hemos utilizado un entorno de desarrollo diferente a Android Studio, nosotros hemos elegido trabajar con Javascript como lenguaje de programación, obviamente apoyado en HTML y CSS. Por lo que el código generado y las librerías utilizadas van a variar con respecto al resto de compañeros.

Inicialmente se creó la escena haciendo uso de clase Renderer y sus diferentes atributos y métodos, de modo que se generó una perspectiva 360º utilizando imágenes estáticas apropiadas para ello. Una vez construido el mundo donde moverse y trabajar se trató de implementar el multitouch. La idea que se tuvo era utilizar doubletouch para movernos entre los diferentes lugares (escenas) de la Alhambra. De este modo se desarrolló el multitouch para movernos entre los diferentes sitios de la Alhambra, además de lo anterior se tuvo en cuenta la orientación de la cámara en todo momento, de modo que al realizar el movimiento del doubletouch los diferentes lugares que comunicaban con el sitio actual iban cambiando su aparición en pantalla dependiendo de la orientación.

Más tarde, se pensó el integrar el sensor de giroscopio para utilizarlo como brújula en nuestro mundo virtual. De este modo se añadió una pequeña brújula en la esquina superior izquierda que nos mostraba en cada caso con cualquier movimiento del teléfono móvil la orientación hacia la que estábamos mirando.

Casi de forma simultánea a lo anterior, se introdujo otro nuevo sensor que en este caso fue la cámara. Este sensor fue aprovechado para la posterior lectura e interpretación de códigos QR. De esta forma podríamos interpretar los diversos códigos QR del reciento y obtener información a través de ellos. Posteriormente se añadió una nueva funcionalidad a estos códigos tal como usarlos para localizarnos dentro del mundo virtual, de modo que tendríamos dos usos diferentes para los códigos QR: obtener información y localización.

Finalmente, se añadió otra nueva funcionalidad a la aplicación. Ésta se trata de un mapa del mundo virtual donde se pueden

observar los diferentes lugares de la Alhambra de los que disponemos. De forma que podemos observar en todo momento el lugar en el que nos encontramos, además de esto con un simple touch sobre el marcador de otro lugar del mapa podemos movernos hacia éste. El mapa además muestra qué lugares están relacionados entre sí, es decir, se encuentran comunicados.

Como nueva y última funcionalidad se añadió el uso de GPS para localizarnos dentro del mapa creado, de este modo de acuerdo a nuestras coordenadas reales podemos situarnos dentro del mapa comprobando que lugar del mapa se encuentra más cercano a nosotros.

2. Descripción de las bibliotecas empleadas.

En este caso solos hemos trabajado con Leaflet, biblioteca de código abierto de JavaScript para mapas interactivos fáciles de usar para dispositivos móviles. Leaflet está diseñado pensando en la simplicidad, el rendimiento y la facilidad de uso. Funciona eficientemente en todas las principales plataformas de escritorio y móviles, se puede ampliar con muchos plugins, fácil de usar y bien documentada y un código fuente sencillo y legible al que es un placer contribuir.

También permite a los desarrolladores sin un fondo GIS mostrar fácilmente mapas web en mosaico alojados en un servidor público, con superposiciones opcionales en mosaico. Puede cargar datos de características de archivos GeoJSON, darle estilo y crear capas interactivas, como marcadores con ventanas emergentes cuando se hace clic en ellas.

En nuestro caso hemos utilizado dicha biblioteca para la creación de nuestro mapa, de modo que hemos utilizado una imagen como fondo para nuestro mapa y Leaflet ya integra el zoom en éste. Más tarde nosotros hemos incorporado los marcadores que corresponden a los diferentes lugares de la Alhambra, además de las etiquetas para mostrar el nombre

del lugar. Aprovechando la funcionalidad de Leaflet hemos incorporado líneas que relacionen los diferentes lugares, además se ha añadido la opción que si pulsas sobre uno de los marcadores del mapa te da la opción de moverte hacia dicha zona a través de una ventana modal.

Finalmente, como se mencionó anteriormente, se añadió un botón de localización en dicho mapa que utilizando las coordenadas GPS reales podemos triangular la posición y situarnos dentro del mapa.

Para el resto de sensores utilizados se han utilizado bibliotecas nativas de JavaScript, de modo que para el sensor multitáctil se ha detectado el número de dedos en pantalla con la función touches.lenght , si estamos tocando la pantalla con dos dedos se activan las funciones multitouch de nuestra aplicación. De esta forma si tocamos con dos dedos la pantalla nos aparece en ambos lados diferentes lugares a los que podemos ir deslizando los dos dedos hacia cualquiera de éstos.

Con respecto al código QR hemos utilizado la API InstaScan, la cual hemos importado. Se ha creado un objeto de la clase InstaScan.scanner con el cual hemos trabajado a la hora de grabar o pausar video, y través de ello hemos podido leer los códigos QR por cámara.

Para el desarrollo de la brújula se ha obtenido la rotación del móvil y mediante html.style.webkitTransform y html.style.transform hemos modificado la rotación de la imagen de la brújula, más concretamente se ha modificado la rotación de la flecha que marca la orientación. De esta forma obteniendo la rotación mediante el giroscopio del móvil podemos simular una brújula utilizando dicha rotación para rotar la aguja de ésta.

Para la geolocalización en el mapa se ha utilizado la variable nativa navigator.geolocation la cual nos proporciona la longitud y la latitud actual , es decir, utilizamos la posición que nos proporciona el navegador. Utilizando dicha posición podemos triangular y asociarnos un marcador del mapa que se encuentre más cercano a nuestra posición.

3. Descripción de las clases utilizadas

En este apartado vamos a comentar las diferentes clases utilizadas, además de sus diferentes atributos y métodos. Para ello vamos a seguir un esquema descendente desde las clases más principales (más genéricas) hasta las más secundarias.

Inicialmente vamos a comentar la clase ***App***, la clase más general la cual posee atributos de otras clases:

ATRIBUTOS:

- El nombre de la aplicación
- Un objeto de la clase Renderer, el cual se encargará de cargar toda la escena 360º utilizada en la aplicación y de gestionar dichos cambios de escena.
- Un objeto de la clase MiniMap, el cual se encargará de cargar el mini mapa con los diversos marcadores y de gestionar la localización GPS.
- Un objeto de la clase QR, el cual se encargará de activar o desactivar la cámara para la lectura de códigos QR. Dicha clase gestiona los diferentes tipos de QR utilizados durante la práctica: localización e información.
- Un objeto de la clase Compass, el cual se encarga de manejar el sensor de giroscopio utilizado en la brújula. Dependiendo de la orientación del móvil los valores de nuestra brújula irán cambiando.
- Un objeto de la clase Cache, el cual se encarga de cargar todos los elementos necesarios durante la ejecución.
- Una variable llamada _start_touch que va a almacenar la fecha en la que se detecta un toque sobre la pantalla del dispositivo.
- Una tupla llamada _touch , que almacena una posición (x,y) y que va a ser utilizada cuando tocamos la pantalla con dos dedos para almacenar la posición de un dedo imaginario creado en medio de ambos.

MÉTODOS:

- *Load( )* : dicho método se encarga de cargar todos los nodos (lugares) del mini mapa en la calle, para ello llenamos un array con las texturas de todos los nodos y finalmente llamamos al método load de atributo cache pasándole dicho array como argumento.

- *Init ( )* : Se encarga de cargar la pantalla de inicio de la aplicación y de iniciar los atributos de la clase. Además, se encarga de detectar el movimiento de los dedos sobre el entorno.

- *TouchStart (Event event):* se encarga de actualizar la variable _start_touch, guarda la posición de un dedo sobre la pantalla, en caso de detectar dos dedos se encarga de crear un dedo imaginario con la posición media de ambos.

- *TouchMove (Event event):* En caso de tener dos dedos sobre la pantalla se encarga de obtener la nueva posición, calcula la distancia de arrastre de los dedos por la pantalla y actualiza los elementos HTML en base a ello.

- *TouchEnd (Event event):* Comprueba la fecha actual, y si seguimos con dos dedos sobre la pantalla. Si terminamos el movimiento sobre uno de los paneles que aparecen sobre la pantalla conseguimos cambiar de escena con el método move del minimap. En caso de realizar un movimiento de arrastre rápido los iconos se desplazan, pero vuelven a su posición enseguida.

A continuación, seguimos detallando la clase ***Cache*** que solo tiene relación con la clase anterior.

ATRIBUTOS:

- Un entero que define el total de todo lo que puede almacenar la caché.
- Un entero que define el total cargado de los datos de aplicación.
- Un Map con un string como clave y un objeto como valor que vamos a utilizar para guardar las distintas imágenes y texturas a cargar.

MÉTODOS:

- *Add (String prefix, String id, Object obj)* : dicho método se encarga de añadir al map un elemento dependiendo del tipo de proceso. Además, se encarga de actualizar la barra de cargado de la pantalla inicial.

- *Get (String name)* : devuelve el elemento de la cache que tenga name como nombre.
- *Image (String path)* : se encarga de procesar una imagen de una ruta, añade dicha imagen a la caché.

- *Texture (String path)* : se encarga de procesar una textura de una ruta, añade dicha textura a la caché.

- *Load (Object array)* : Carga desde el array {tipo, id, ruta} en la caché dependiendo del tipo procesará una imagen o una textura

Vamos a detallar las principales clases utilizadas en la aplicación, de este modo vamos a iniciar explicando la clase ***Renderer****:*

ATRIBUTOS:

- Un elemento HTML que guarda el elemento del HTML que define el entorno.
- Un elemento HTML que guarda las etiquetas HTML.
- Un objeto de la clase THREE.Raycaster. Esta clase está diseñada para ayudar con el raycasting. El Raycasting se utiliza, entre otras cosas, para seleccionar los objetos en el espacio 3d sobre los que se encuentra el ratón.
- Una variable booleana para controlar a la hora de dibujar la escena.
- Un entero que representa el ángulo acimut que se refiere al ángulo de la orientación sobre la superficie de la esfera virtual.

MÉTODOS:

- *Init ( )* : dicho método se encarga de crear el Renderer , las cámaras , los controles y la escena. Además, se encarga de actualizar la función de movimiento de dos dedos a la nueva rotación (brújula) si el ángulo acimut ha cambiado.

- *Azimuth_connection(Int azimuth)* : Devuelve desde el nodo actual el siguiente nodo caminando en un ángulo acimutal de 0 - 3 y desde el dispositivo de rotación actual.

- *Update_direction ( ) :* Actualiza la función de movimiento de dos dedos a la nueva rotación (brújula). Para cada dirección posible desde el nodo actual. Toma la dirección de la brújula y ponla en sentido contrario a las agujas del reloj desde la dirección de conexión. Obtiene la nueva imagen de vista previa y el elemento html de ese id. Modifica su html y eliminarlo de la lista de no visitantes. Para los ítems html no visitados, añada una clase para evitar que se muestre.
- *Create_renderer ( ) :* Crea el renderizador THREE JS.
- *Create_controls ( ) :* Crea los controles THREE JS Orbit .
- *Create_camara ( ) :* Crea la cámara THREE JS.

- *Create_scene ( ) :* Crea la escena THREE JS. Inicialmente crea la escena , añadimos una luz blanca a ésta. Creamos una escena de fondo y un oscurecedor para el material del Skybox. Ademas creamos una skybox y fijamos el material a ésta y lo añadimos a la escena de fondo creada anteriormente.

- *Cast (Float x, Float y) :* Aplicamos Ray Casting usando THREE JS.

- *Change_scene ( ) :* Utilizada para cambiar la escena. Cambia el material del oscurecedor al nuevo skybox.Pone los controles a 0 y actualiza. Pone nuevos elementos del nodo y actualiza las cajas de direcciones de dos dedos.

- *Resize (Event event):* Redimensiona el evento de ventana que fija el ratio de aspecto. Actualiza la matriz de proyección de la cámara.

- *Start ( ) :* Función para empezar a dibujar la escena. Cambia el valor del atributo booleano stop.

- *Stop ( ) :* Función para parar de dibujar la escena. Cambia el valor del atributo booleano stop.

- *Put_items ( ) :* Pone todos los elementos del nodo actual en primer plano. Para ello borrar los elementos y etiquetas anteriores y dibujar todos los elementos del nodo actual.Pone el item en primer plano y finalmente pone las etiquetas.

- *Render_scene (Time time) :* Renderiza la escena. Para cada elemento de la escena principal lo redibuja.

- *Render (Time time) :* Renderiza el visualizador: Fondo, primer plano y elementos. Coloca el centro de la malla de fondo en la posición de la cámara y renderiza fondo y escena de primer plano.

A continuación, vamos a seguir explicando la clase ***QR****:*

ATRIBUTOS:

- Un objeto scanner de la clase InstaScan.Scanner que utilizaremos inicializar la cámara, pararla y leer los códigos QR.

MÉTODOS:

- *Init ( )* : Método que asigna al botón QR una función un evento para que cuando se haga click en él, se abra el popup y se inicialice la cámara.

- *Start_scanner ( )* : Método para inicializar la cámara. Obtenenemos la cámara trasera si es posible. En caso de que el dispositivo no disponga de cámaras, salimos. Una vez obtenida la cámara llamamos al método read_qrcode para empezar a leer

- *Stop_scanner ( )* : Método para parar la cámara.

- *Read_qrcode ( )* : étodo para leer el código QR. El código contendrá de manera codificada un json que parsearemos y obtendremos un objeto con dos atributos:
  - El tipo de QR (localización o información).
  - La localización o la información.

  Una vez obtenido esto, saltará un mensaje por pantalla que nos informará del tipo de QR que hemos leido y que podremos aceptar o rechazar si queremos que se realice la acción. Si es de tipo localización, nos moverá a dicho lugar y si, es de tipo información, nos mostrará dicha información.

- Read_location_qr ( ) : método que se llama cuando se ha leído un QR de localización y muestra un mensaje en una ventana modal para poder movernos a una zona distinta del mini mapa.

- Read_information_qr ( ) : método que se llama cuando se ha leído un QR de información y muestra un mensaje en una ventana modal para poder visualizar información sobre el lugar correspondiente al QR.

A continuación, vamos a seguir explicando la clase ***Compass****:*

Para esta clase no cabe destacar ningún atributo importante, ya que se trata de una clase bastante simple y todo el trabajo se hará en sus métodos con variables locales.

MÉTODOS:

- *Init ( )* : Método que sirve para iniciar nuestra brújula. Realiza una petición para otorgar permisos para la brujula.

- *Compass_permission (Bool granted):* Función que comprueba si tenemos permisos, en caso afirmativo llama a la función compass_orientation pasándole el evento de orientación del dispositivo recibido.

- *Compass_orientation (Event event)* : Función que obtiene la orientación del evento recibido y actualiza la orientación del puntero de nuestra brújula , de modo que dependiendo de la orientación que recibamos modificaremos la posición del puntero simulando una brújula.

Seguimos en este caso con la clase ***MiniMap****:*

ATRIBUTOS:

- Un objeto image de la clase L.imageOverlay de Leaflet que representa la imagen que utilizamos como mini mapa en la cual trabajaremos.
- Un objeto map de la clase L.Map de Leaflet en el que indicamos el sistema de referencia de coordenadas que en nuestro caso representa una cuadrícula cuadrada e indicamos el mínimo y el máximo de zoom.
- Un objeto group de la clase L.Group que se ha utilizado para agrupar los diferentes marcadores (lugares) del mini mapa.
- Un objeto current_node de la Clase Node que representa el nodo actual en el que nos encontramos.

MÉTODOS:

- *Init ( )* : Método que sirve para iniciar nuestra mini mapa. Inicialmente situamos la escena en el nodo actual, dibujamos los marcadores sobre el mapa y si poseemos geolocalización mediante el navegador se muestra en el mini mapa un botón de localización para localizarnos dentro del mundo virtual utilizando coordenadas reales GPS.

- *Gps_localization ( ) :* Encuentra el nodo actual mediante la localización GPS. De modo que utilizando las coordenadas reales GPS triangula nuestra posición y la asocia al nodo más cercano del mini map utilizando, obviamente, las coordenadas reales de los lugares.

- *Find_node (String id) :* Encuentra de todos los nodos, el que tiene como identificador id.

- *Nodes ( ) :* Devuelve todo el nodo del grafo, empezando por el nodo actual.

- *Move (v) :* Se desplaza a un nuevo nodo y cambia la escena, puedes pasar
  - String : Encuentra un Nodo de Clase de Objeto con ese id
  - Nodo : Clase de objeto a mover

  Para ello llama a la función change_scene de la clase Renderer.

- *Current ( ) :* Devuelve el nodo actual

- Draw_markers ( ) : Elimina el grupo anterior y dibuja todos los marcadores y el camino en un nuevo grupo finalmente lo imprime en el mapa.

Finalmente vamos a concluir explicando las clases mas secundarias o clases auxiliares que son utilizadas por otras. Para ello primero a comenzar explicando la clase ***Node***:

ATRIBUTOS:

- Un string title que representa el nombre del lugar que representa dicho nodo.
- Un string id que representa el identificador que vamos a asociar a dicho nodo.
- Una tupla marker que representa las coordenadas ficticias de dicho nodo en el mini mapa.
- Una tupla gps que representa las coordenadas reales de dicho nodo en el mundo real.
- Un vector de ítems donde guardaremos la información relativa cada nodo.

MÉTODOS:

- *Add (Item ítem)* : Para añadir dicho ítem al vector de ítems.
- *Set_gps (Float lat, Float lng)* : Para poner a dicho nodo sus coordenadas reales GPS.
- *Get_gps ( )* : Función que nos devuelve las coordenadas reales GPS de dicho nodo.
- *Set_marker (Float x, Float y)* : : Para poner a dicho nodo sus coordenadas ficticias para situarlo en el mini mapa.
- Get_marker ( ) : Función que nos devuelve las coordenadas ficticias de dicho nodo.
- Id ( ) : Función que nos devuelve el identificador de dicho nodo.
- Title ( ) : Función que nos devuelve el título de dicho nodo.
- Distance (Float lat, Float long) : Devuelve la distancia al nodo actual dado la latitud y la longitud.
- Connect (Node node, int angle) : Se conecta a otro nodo mediante un ángulo
  0 -> Norte (000 - 090) grados
  1 -> Este (090 - 180) grados
  2 -> Sur (180 - 270) grados
  3 -> Oeste (270 - 360) grados
- Connections ( ) : Devuelve todas las conexiones que tiene dicho nodo.
- To (id) : Devuelve el nodo desde un ángulo (dirección)

A continuación, vamos a seguir explicando la clase ***Connection****:*

ATRIBUTOS:

- Un entero angle_walk que representa el angulo a partit del cual se comunican dos nodos.
- Un nodo A y un nodo B que indican que dos nodos están conectados.

MÉTODOS:

- *From ( )* : Devuelve el nodo inicial de la conexión.
- *To ( ) :* Devuelve el nodo final de la conexión.
- *Angle ( ) :* Devuelve el ángulo a partir del cual se comunican ambos nodos.

Finalmente vamos a terminar la descripción de las diferentes clases comentando la clase ***Item****:*

ATRIBUTOS:

- Un string _title que representa el título del ítem.
- Un string _description que representa la información que aporta dicho ítem.
- Un string _image que asocia una imagen de dicho lugar al ítem.
- Un entero_ angle que representa la dirección del bloque.
- Un entero _size que representa el tamaño del bloque que aparece en el ítem.
- Una variable booleana _visible que representa la visibilidad de los diferentes objetos en el mapa.

MÉTODOS:

- *Title ( )* : Devuelve el título del ítem.
- *World (float angle , int size)* : Crea un elemento para la escena y posición. Crea un THREE Cube para el ítem.
- *Html ( ) :* Devuelve código HTML para cargar el ítem en el fichero principal.

4. Diagrama de clases de la aplicación