UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Navegación en 3D en espacios indoor
Autores: Manuel De la Penna – Agustin De Nigris
Director: Dra. Silvia Gordillo
Codirector: Dra. Cecilia Challiol
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

El trabajo tiene como objetivo implementar una aplicación que permita posicionar al usuario dentro de espacios indoor, en particular, para este trabajo, se ha tomado el edificio de la Facultad de Informática de la UNLP. La misma consta de un navegador de tres dimensiones que permite al usuario desplazarse tanto física como virtualmente. Su interfaz es de tipo "Street View" y utiliza Códigos QR para posicionar al usuario, permite la lectura de los mismos sin salir de la aplicación.

Para llevar a cabo la implementación, se adaptó el prototipo de dos dimensiones propuesto por los ex-alumnos Luciano Appathie y Claudio Borre en su tesina de grado en el año 2012 y se utilizó como base para el desarrollo el estándar definido por WebGL y HTML5 del lado del cliente y PHP del lado del servidor.

Se presenta un marco teórico sobre los sistemas basados en localización, luego se hace un análisis de la aplicación tomada como base, posteriormente se muestra el Modelo de Navegación en tres dimensiones creado (con los diagramas de secuencia más relevantes). Además, se enumera y detallará los Casos de Uso para posteriormente pasar a la implementación del prototipo y finalmente mostrar un ejemplo de uso de la aplicación.

Palabras Claves

Sistemas de Posicionamiento de Interiores - Navegación en 3D indoor – WebGL – HTML5 – PHP – Javascript – código QR - Three.js – Tweenjs – KineticJS – JsQRCode – jQueryMobile – Symfony – Patrón MVC.

Trabajos Realizados

Análisis teórico sobre Posicionamiento. Instalación e interpretación de la aplicación tomada como base para comprender su funcionamiento. Definición de requerimientos funcionales y no funcionales. Búsqueda de herramientas y tecnologías disponibles y análisis de viabilidad de la aplicación a partir de herramientas y tecnologías encontradas. Definición del Modelo de Navegación en 3D, diagramas de secuencia y Casos de Uso. Implementación del prototipo y ejemplos de uso.

Conclusiones

Se comprendió con profundidad el funcionamiento de los sistemas de posicionamiento en interiores, sus usos, beneficios y problemáticas aún no resueltas, como por ejemplo, la no existencia de un estándar definido para su implementación.

Las herramientas destinadas al desarrollo de este tipo de aplicaciones avanzaron mucho, tanto en eficiencia, como en facilidad de uso. La consolidación de HTML5 y WebGL como estándares y el soporte de hoy en día casi todos los navegadores del mercado, incluidos los de dispositivos móviles, además de la creciente capacidad de procesamiento de éstos últimos, fomentan cada vez más su desarrollo.

Trabajos Futuros

A modo de trabajos futuros se propone cambiar la tecnología de Códigos QR por otra capaz de posicionar al usuario en el mapa automáticamente, sin la necesidad de leer continuamente códigos cada vez que se mueve. Implementar recorridos automáticos y navegación virtual libre. Extender la implementación a toda la facultad, agregar contenido multimedia al detalle de estructuras navegables y vistas 360 de las mismas. Agregar plantas, columnas, sillas, mesas y otros objetos para mejorar la interfaz de navegación.

Fecha de la presentación: Abril de 2014

Índice General

Índice de contenidos

Índice de Figuras 4
1. Introducción 9
1.1 Motivación 9
1.2 Objetivos 10
1.3 Estructura de la Tesis 11
2. Posicionamiento 13
2.1 Sistemas de Información Geográfica (GIS) 13
2.2 Sistemas Basados en Localización (LBS) 14
2.3 Indoor vs Outdoor 15
2.4 Sistemas de Posicionamiento de Interiores (IPS) 16
2.4.1 Beneficios de IPS 17
2.4.2 Ejemplos de IPS 17
2.5 Sistemas de Street View 19
3. Background 22
3.1 Aplicación de Navegación 2D usada de base 22
3.1.1 Descripción del funcionamiento de la aplicación original. 26
3.2 Tecnologías y Herramientas utilizadas 26
3.2.1 HTML5 26
3.2.2 KineticJS 27
3.2.3 WebGL 29
3.2.3.1 WebGL: Soporte de Navegadores. 32
3.2.4 Three.js 35
3.2.5 TweenJS 38
3.2.6 JsQRCode 40
3.2.7 jQuery Mobile 40
3.2.8 Symfony 41
4. Modelo de Navegación 3D 43
4.1 Modelo especificado para Navegación 3D 43
4.2 Interacción y Diagramas de Secuencia 45
4.3 Ejemplo de Instanciación del Modelo creado 47

5. Casos de Uso 52
5.1 Usuarios Comunes 52
5.2 Usuarios Registrados 56
6. Implementación del Prototipo 58
6.1 Arquitectura General 58
6.2 Vista 3D 59
6.3 Animaciones 63
6.4 Navegación lógica y física 65
6.5 Lectura de códigos QR 66
7. Ejemplos de uso 69
8. Conclusiones y trabajos futuros 76
8.1 Conclusiones 76
8.2 Trabajos a Futuro 77
9. Bibliografía 79
Anexo A: Casos de Uso reutilizados 82
Anexo B: Implementación del Backend 92
Anexo C: Implementación del Controlador 100
Anexo D: Implementación del Frontend 103

Índice de Figuras

Figura 1. Interpretación vectorial (izquierda) y raster (derecha) 14
Figura 2. Captura de imagen de Google Street View. 20
Figura 3. Captura de imagen de Google Street View luego de avanzar unos metros hacia adelante. 20
Figura 4. Captura de imagen de Google Street View luego de rotar a la izquierda. 21
Figura 5. Captura de la interfaz de listado de estructuras navegables. 22
Figura 6. Captura de mapa 2D con recorrido establecido. 23
Figura 7. Barra de navegación de mapa 3D. 24
Figura 8. Captura de información detallada de la Biblioteca. 24
Figura 9. Mapa en dos dimensiones con recorrido establecido. 25
Figura 10. Captura de pantalla luego de realizar un recorrido completo. 25
Figura 11. Arquitectura de la Aplicación de Navegación 2D. 26
Figura 12. Captura de código: creación escenario KineticJS. 28
Figura 13. Captura de código: creación círculo KineticJS. 28
Figura 14. Captura de código: creación rectángulo KineticJS. 28
Figura 15. Captura de código: creación rectángulo KineticJS. 29
Figura 16. Captura de código: Asignaciones KineticJS. 29
Figura 17. Modelo conceptual de la arquitectura WebGL 31
Figura 18. Cuadro de compatibilidad con WebGL de los navegadores. Año 2012. 33
Figura 19. Cuadro de compatibilidad con WebGL de los navegadores. Año 2014. 34
Figura 20. Captura de ccódigo: Definición de renderer en Three.js 35
Figura 21. Captura de código: Creación escena en Three.js 36
Figura 22. Captura de código: Agregación de escena en Three.js 36
Figura 23. Captura de código: Creación de cámara en Three.js 36
Figura 24. Captura de código: Inclusión de cámara en escena en Three.js 37
Figura 25. Captura de código: Mostrar renderer en escena en Three.js 37
Figura 26. Captura de código: Utilización requestAnimationFrame en Three.js. 38
Figura 27. Captura de código: Función renderer en Three.js. 38
Figura 28. Captura de código: Inclusión librería TweenJS 39
Figura 29. Captura de código: Constructor de Tween para librería TweenJS 39
Figura 30. Modelo de Clases IndoorNavigation3D. 43
Figura 31. Diagrama de secuencia “getCaminoDesdeHasta()” 45

Figura 32. Diagrama de secuencia “getPuntoNavegaciónDeEstructura()”.....46
Figura 33. Diagrama de secuencia “getMultimediaDeEstructura()” 46
Figura 34. Diagrama de secuencia “obtenerDistanciaDesdeHasta()” 46
Figura 35. Diagrama de secuencia “posicionFisicaDentroDeCamino()” 47
Figura 36. Ejemplo de edificio simplificado en dos dimensiones. 47
Figura 37. Ejemplo edificio simplificado en tres dimensiones. 48
Figura 38. Edificio simplificado con información agregada en dos dimensiones. 49
Figura 39. Edificio simplificado con información agregada en tres dimensiones.....49
Figura 40. Diagrama de instanciación del edificio simplificado (considerando dos y tres dimensiones). 50
Figura 41. Grafo resultante de la instanciación del edificio simplificado (igual para dos y tres dimensiones). 50
Figura 42. Grafo de puntos de navegación en dos dimensiones. 51
Figura 43. Grafo de puntos de navegación en tres dimensiones. 51
Figura 44. Arquitectura de la aplicación IndoorNavigation3D. 58
Figura 45. Imágenes de paredes simplificadas en plano 3D. 59
Figura 46. Captura de código: Función de construcción de paredes en el plano 3D. 62
Figura 47. Captura de código: Movimiento de cámara entre dos puntos 64
Figura 48. Mapa de dos dimensiones en miniatura 66
Figura 49. Íconos de mapa en dos dimensiones. 66
Figura 50. Captura de pantalla principal de la aplicación. 69
Figura 51. Captura de pantalla de lista de estructuras navegables. 70
Figura 52. Captura de pantalla interfaz de navegación 3D. 70
Figura 53. Captura de pantalla interfaz navegación. Diferencia entre posición lógica y Física en plano 2D 71
Figura 54. Captura de pantalla interfaz navegación. Posición lógica fuera de camino establecido.....72
Figura 55. Captura de pantalla. Barra de navegación 3D. 72
Figura 56. Captura de pantalla. Barra de navegación QR. 73
Figura 57. Captura de pantalla. Popup de finalización de recorrido.....74
Figura 58. Captura de pantalla. Informe detallado de estructura navegable. 74
Figura 59. Captura de pantalla. Lista de estructuras navegables. Opción de continuación. 75
Figura 60. Captura de código: Definición de la clase Estructura. 92
Figura 61. Captura de código: Definición de la clase Multimedia. 92

Figura 62. Captura de código: Definición de la clase Puntos....93
Figura 63. Captura de código: Definición de la clase PuntoNavegación....93
Figura 64. Captura de código: Definición de la clase PuntoNavegacionPuntoNavegacion....94
Figura 65. Captura de código: Definición de la clase OrientacionPared....94
Figura 66. Captura de código: Definición de la clase ParedDibujable....94
Figura 67. Actualización de base de datos....95
Figura 68. Pantalla principal de la aplicación de administración....95
Figura 69. Pantalla de edición de paredes dibujables (3D)....96
Figura 70. Captura de código: Definición de la clase Dijkstra....97
Figura 71. Captura de código: Funciones getCamino() y cargarListaDeAdyacencia()....98
Figura 72. Captura de código: Función caminoMinimoDesdeHasta() Parte 1....98
Figura 73. Captura de código: Función caminoMinimoDesdeHasta() Parte 2....98
Figura 74. Captura de código: Función Pública - executeOrigen()....100
Figura 75. Captura de código: Función Pública - executeBuscar()....100
Figura 76. Captura de código: Función Pública - executeNavegar()....101
Figura 77. Captura de código: Función Pública - executeDetalleEstructura ()....101
Figura 78. Patrón Decorator....103
Figura 79. Captura de código: layout.php de la aplicación....104
Figura 80. Captura del Bloque cabecera de la página....104
Figura 81. Captura del Bloque a utilizar para la barra de navegación....104
Figura 82. Captura del Bloque de pie de página....105
Figura 83. Captura de código: Template - barra_de_navegacion.php....105
Figura 84. Captura de barra de navegación 3D....105
Figura 85. Captura de código:Template - barra_qr.php....105
Figura 86. Captura de barra de códigos QR....105
Figura 87. Captura de código: Template - mensajes_usuario.php....106
Figura 88. Captura de código: Template - origenSuccess.php....106
Figura 89. Captura de lector de códigos QR de la página principal....107
Figura 90. Captura de código: Template - buscarSuccess.php....107
Figura 91. Captura de listado de estructuras navegables de la pantalla de búsqueda....108
Figura 92. Captura de código:Template - navegarSuccess.php....108
Figura 93. Captura de la pantalla de navegación....109
Figura 94. Captura de popup con mensaje de llegada a destino....109

Figura 95. Captura de código: Mensajes de usuario 109
Figura 96. Captura de código: Declaraciones plano 2D 110
Figura 97. Captura de código: Declaraciones plano 3D 110
Figura 98. Captura de código: invocación a la función inicio().....111
Figura 99. Captura de código: función inicio() 111
Figura 100. Captura de código: función dibujarFacu().....111
Figura 101. Captura de código: Encabezado función init() 112
Figura 102. Captura de código: cálculo de pendiente y ángulo de rotación de pared. 112
Figura 103. Captura de código: declaración del renderer 112
Figura 104. Captura de código: declaración del piso del escenario.....112
Figura 105. Captura de código: Habilitación de botonera y construcción de paredes. 113
Figura 106. Captura de código: Activación de cámara y deshabilitación de botón recalcular. .113
Figura 107. Captura de código: funciones anímate() y render(). 113
Figura 108. Captura de código: función activarCamara() 114
Figura 109. Captura de código: función dibujarPoligono().....115
Figura 110. Captura de código: función dibujarPuntoNavegacion().....115
Figura 111. Captura de código: función dibujarTodosLosPuntos() 116
Figura 112. Captura de código: función dibujarLinea() 116
Figura 113. Captura de código: función dibujarPared() 117
Figura 114. Captura de código: función irAPunto().....117
Figura 115. Captura de código: funciones getPuntoSiguiente() y getPuntoAnterior() 118
Figura 116. Captura de código: función rotar360() 118
Figura 117. Captura de código: funciones deshabilitarBotonRetrocerder() y deshabilitarBotonAvanzar() 119
Figura 118. Captura de código: funciones deshabilitarBotones360() y habilitarBotones360() .119
Figura 119. Captura de código: funciones habilitarBotonRetrocerder() y habilitarBotonAvanzar() 119
Figura 120. Captura de código: funciones chequearBotonera(), deshabilitarBotonera() y habilitarBotonera() 120
Figura 121. Captura de código: funciones tiempoRotacion() y tiempoAvanceRetroceso() 120
Figura 122. Captura de código: funciones encenderLector() y apagarLector() 121
Figura 123. Captura de código: funciones actualizarPosicionFisica() y chequearNuevaUbicacionFisica().....121

Figura 124. Captura de código: funciones habilitarBotoneraRecalcular(), deshabilitarBotoneraRecalcular() y recalcularCamino().....122

1. Introducción

1.1 Motivación

Los sistemas de posicionamiento global (GPS)[1] han evolucionado de tal modo que hoy en día casi no existe dispositivo móvil que no los contenga. Que un dispositivo móvil cuente con acceso a internet no es novedad, y que posean gran capacidad de procesamiento tampoco, a tal punto que poseen dos, tres y hasta cuatro procesadores, aceleradores gráficos, y una gran capacidad de memoria RAM. Adicionalmente han aparecido nuevas tecnologías para el desarrollo de aplicaciones web y muchas otras han sido mejoradas y/o abstraídas, a partir de frameworks. Los navegadores web actuales en su mayoría provee soporte para todas ellas, y los dispositivos móviles ya no se encuentran excluidos de este soporte.

Estos avances han permitido contar con aplicaciones móviles basadas en posicionamiento, las cuales abarban diferentes dominio y proveen diferentes servicios a los usuarios. Pero aún es un campo abierto como guiar al usuario para moverse dentro de espacios indoors.

Las aplicaciones móviles en donde el usuario navega tanto digitalmente (aplicaciones Web tradicionales o Hipermedia), como físicamente (éste se mueve por el mundo real) se denominan aplicaciones de Hipermedia Móvil [1]. En estas aplicaciones el usuario tiene la posibilidad de obtener información mediante algún dispositivo móvil de dos maneras distintas: una digitalmente es decir a través de la navegación tradicional de las aplicaciones de Hipermedia y otra de manera presencial acorde a su posición cuando un objeto del mundo real lo sensa y la aplicación automáticamente le provee información del lugar. La posición del usuario es fundamental para determinar qué información debe recibir en cada lugar. Estas aplicaciones deben considerar la movilidad del usuario como parte esencial de su funcionamiento sumando complejidad a la hora de la creación de las mismas [2].

Desplegar información del lugar donde se encuentra un usuario, calcular caminos cuando el usuario elige un lugar para visitar, (es decir, calcular el camino desde donde se encuentra ubicado hasta el destino elegido), permitir que un usuario cancele el recorrido que está realizando en cualquier momento del mismo, considerar que el usuario puede desviarse del camino pautado (en este caso además, el sistema debe poder asistirlo para retomar su camino), considerar información adicional de los sitios de interés (por ejemplo, los horarios de apertura y cierre del destino elegido), permitir al usuario volver a ver una página ya visitada como así también volver físicamente a un lugar ya visitado (en el caso de volver físicamente el sistema debe calcular el camino al lugar anteriormente visitado), son algunas de las funcionalidades que debe brindar este tipo de aplicaciones.

El soporte para HTML5 [3] y WebGL[4], en casi todos los dispositivos móviles sin duda ha permitido abrir el abanico a la hora de desarrollar aplicaciones, y las de Hipermedia Móvil sin dudas han adquirido un papel protagónico.

[1] El Sistema Global de posicionamiento (GPS) permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una precisión hasta de centímetros.

En esta tesis se busca dar una posible solución a la asistencia a la navegación en el mundo real de las aplicaciones de Hipermedia Móvil. En particular permitiendo que el usuario pueda navegar virtualmente por el ambiente pero también de manera física.

1.2 Objetivos

La tesis de grado consiste en desarrollar una aplicación web de navegación de interiores en tres dimensiones utilizando el estándar WebGL. El software funcionará en dispositivos móviles de alta gama y su fin será el de orientar al usuario tanto de manera física, como de manera virtual dentro del edificio, en particular, el prototipo va a estar desarrollado para del edificio de la Facultad de Informática de la UNLP.

El proyecto planteado tiene como objetivo continuar la implementación realizada por los ex-alumnos Luciano Appathie y Claudio Borre en su tesina de grado [5]. Dicha implementación consta de un navegador de interiores en dos dimensiones, la cual no pudo ser implementada en tres dimensiones debido a diversos inconvenientes tecnológicos de ese momento, por ejemplo, la incompatibilidad de navegadores y las librerías encargadas de abstraer el uso de WebGL, sumado al bajo rendimiento de estas librerías en dispositivos móviles. En la actualidad, gracias a los distintos avances en estas áreas, estos problemas técnicos ya no existen, por lo tanto, motivan a la propuesta planteada a continuación.

Se creará un modelo para considerar la navegación en tres dimensiones a partir del modelo existente en dos dimensiones. Este modelo permitirá al usuario moverse virtualmente por el espacio como así también físicamente. Al moverse físicamente se tendrá en cuenta la posición actual del mismo para guiarlo en dicho espacio.

La aplicación a desarrollar contará principalmente con las siguientes características:

Se adaptará la implementación existente en dos dimensiones para generar el plano en tres dimensiones.

Como prototipo a implementar, se representará de manera acotada el primer piso de la Facultad en tres dimensiones, es decir, se recreará virtualmente la misma simulando todas sus paredes, piso, aulas, etcétera. Este prototipo tendrá además las siguientes características:

Cada pared, aula o piso constará de una textura asociada a ésta, pudiendo ser una fotografía de su fachada, o bien un color, que le servirá al usuario para identificar el lugar en el que está situado dentro del edificio.

El usuario se podrá mover tanto virtualmente como físicamente. Para este último tipo de movilidad se tendrá en cuenta la posición del usuario.Se brindará al usuario la posibilidad de moverse por el edificio a partir de un punto de partida para llegar a un destino específico. Para lograr esto se representará a la facultad dentro de un eje cartesiano de tres dimensiones. Existirán distintos puntos navegables para el usuario y además otros puntos de referencia para calcular los caminos.

Se generará una interfaz de tipo “Street View” [6] en donde el usuario verá el ambiente que lo rodea con la indicación del camino a seguir.

Se podrá desplegar información de los puntos de navegación accesibles por el usuario.

Con la implementación de la presente aplicación se espera construir un software que sirva de herramienta de soporte a los usuarios que deseen conocer o recorrer un espacio físico, facilitando la búsqueda de un lugar específico del edificio y el recorrido que determina cómo llegar al mismo. En particular se mostrará un ejemplo con la Facultad de Informática,

1.3 Estructura de la Tesis

La estructura de la presente tesis se divide en 9 capítulos y un anexo abarcando los siguientes temas:

- **Capítulo 1 – Introducción.**
  Se describe la motivación y los objetivos de la presente tesis.

- **Capítulo 2 - Posicionamiento.**
  Se presenta un marco teórico sobre los sistemas basados en localización y se hace un breve estudio de los diferentes algoritmos de localización existentes, los beneficios que pueden obtenerse con ellos y sus usos.

- **Capítulo 3 - Background**
  Se detalla la aplicación utilizada como base y se mencionan las tecnologías y herramientas utilizadas durante el desarrollo, brindando las características principales de cada una de ellas y ejemplos básicos de su utilización.

- **Capítulo 4 - Modelo de Navegación 3D**
  Se presenta el modelo de clases de la aplicación y los principales diagramas de secuencia.

- **Capítulo 5 - Casos de Uso.**
  Se enumeran las modificaciones realizadas a los casos de uso ya existentes y los agregados para la implementación en tres dimensiones para describir el funcionamiento de la aplicación, mostrando el flujo normal de ejecución de la misma.

- **Capítulo 6 - Implementación.**
  Se detalla la arquitectura de la aplicación, los aspectos tenidos en cuenta durante el proceso de adaptación de la interfaz en dos dimensiones a la de tres dimensiones, es decir, problemáticas encontradas, soluciones, ventajas y desventajas ante diferentes variantes, justificaciones de decisiones tomadas y fundamentos teóricos.

- **Capítulo 7 - Ejemplos de uso.**
  Se detalla el funcionamiento de la aplicación, con ejemplos de uso, imágenes

y explicaciones detalladas.

- **Capítulo 8 - Conclusiones y trabajos futuros.**
  Se describen las conclusiones, resultados obtenidos y trabajos futuros planteados para mejorar el software de navegación.

- **Capítulo 9 - Bibliografía.**
  Se detalla toda la bibliografía utilizada para el desarrollo de este proyecto.

- **Anexos**

2. Posicionamiento

Antes de hablar de los sistemas de posicionamiento indoor se mencionarán diferentes tipos de sistemas que son anteriores a éstos y han servido de base para la creación de los mismos.

2.1 Sistemas de Información Geográfica (GIS)

Geographic Information System (Sistema de Información geográfica) son un caso especial de sistemas de información para la gestión, análisis y visualización de datos geográficos. Estos sistemas se encargan de capturar, manipular y presentar el conocimiento geográfico.

Dicho conocimiento, puede estar estructurado en diferentes conjuntos, como mapas interactivos, modelos de geoprocesamiento, modelos de datos o metadatos.

Se puede decir que un GIS es una herramienta basada en una computadora y que permite a los usuarios crear consultas interactivas y así analizar eventos que ocurren en un área geográfica, y a partir de allí obtener información para la toma de decisiones y planear estrategias.

Un GIS trabaja almacenando información acerca del mundo como una colección de capas, las que pueden relacionarse geográficamente.

Los GIS funcionan con dos tipos de almacenamiento de información: el modelo vectorial y el modelo raster.

Vectorial: Las características geográficas se digitalizan en datos vectoriales, aunque se pierde la cantidad de detalle del espacio, manteniendo las características geométricas de las figuras. El detalle perdido puede tener como resultado una pequeña diferencia con el espacio, aunque este es mínimo.

Raster: Divide el espacio en una serie de líneas y columnas verticales y horizontales, donde cada cuadrícula es llamado “Celda”, donde cada una de ellas representa un único valor representando un elemento particular en el mapa. Son muy utilizados en estudios que requieran la generación de capas continuas, necesarias en fenómenos no discretos; también en estudios medioambientales donde no se requiere una excesiva precisión espacial (contaminación atmosférica, distribución de temperaturas, localización de especies marinas, análisis geológicos, etc.).

Estos dos modelos tienen sus ventajas y desventajas, pero actualmente aún los dos son utilizados en los modelos GIS [7]. En la Figura 1, se puede apreciar un ejemplo concreto de amos modelos.

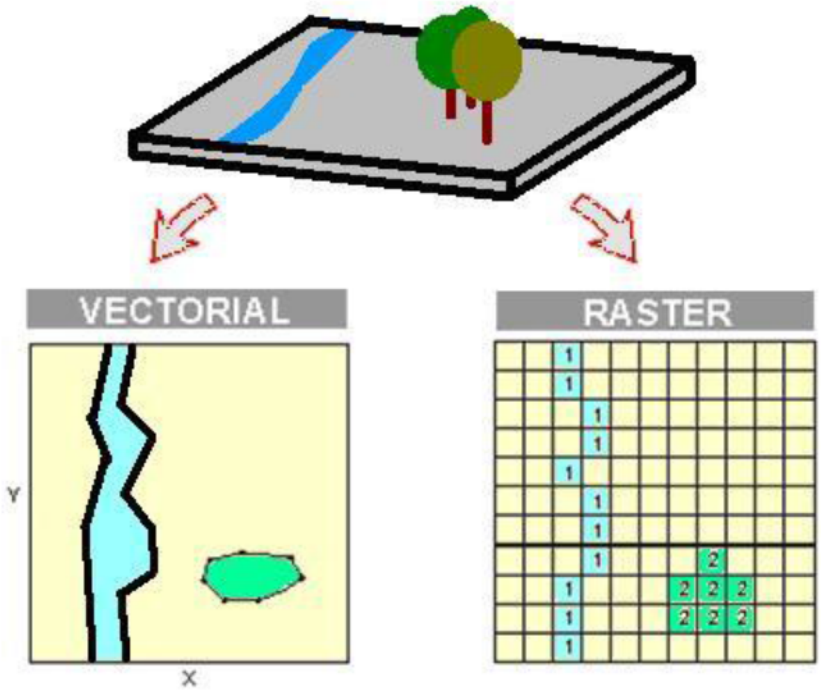


Figura 1. Interpretación vectorial (izquierda) y raster (derecha)[2]

2.2 Sistemas Basados en Localización (LBS)

En la actualidad, GIS están teniendo una fuerte implementación en los Sistemas Basados en Localización (LBS), debido al abaratamiento y masificación de la tecnología GPS integrada en dispositivos móviles (teléfonos móviles, PDAs, ordenadores portátiles, etc.). Los sistemas LBS permiten a los dispositivos con GPS mostrar su ubicación respecto a puntos de interés fijos (hoteles, restaurantes, cajeros, estaciones de servicio) o móviles (amigos, colectivos, etc.) o para transmitir su posición a un servidor para la visualización u otro tipo de tratamiento. [8]

Los Sistemas Basados en Localización ofrecen un servicio personalizado a cada usuario basado en información de su ubicación geográfica actual.

Para lograr su cometido, un LBS utiliza tecnología de sistemas de información geográfica, alguna de las tecnologías de posicionamiento disponible, como por ejemplo la tecnología GPS y también tecnología de comunicación para así transmitir información hacia una aplicación LBS, la cual será la encargada de procesar la información y responder la solicitud.

La mayoría de las LBS buscan proveer servicios geográficos en tiempo real. Algunos ejemplos típicos de esto son servicios de mapas callejeros y enrutamiento. Existen básicamente dos escenarios para las aplicaciones basadas en localización: un usuario puede necesitar de un servicio de información o el centro de administración puede requerir rastrearlo

[2] 1http://1.bp.blogspot.com/-M9WnYTj9SPY/UIRi1i1yYrI/AAAAAAAAABY/1SFLMoP8OIM/s1600/modeloss.jpg

(*tracking*) en tiempo real. [9]

A partir del objetivo de un sistema LBS, existen dos posibles escenarios para este tipo de aplicaciones:

A. Un usuario necesita de un servicio de información.
B. Un centro de administración necesita rastrear un móvil en tiempo real.

Para estos dos escenarios, se puede determinar la posición del usuario a través de un dispositivo de posicionamiento integrado en el móvil del mismo.

Esta información, junto con otros datos relevantes, es transmitida a la aplicación LBS. Allí, todos los datos son analizados con el soporte de sistemas de información geográfica, y así poder entregar una respuesta al usuario.

Los sistemas LBS, tienen dos tipos de modelo de trabajo, el modelo activo y el pasivo. Un sistema LBS activo está dedicado para aquellos usuarios que cuentan con un dispositivo móvil. Su objetivo es proveer al usuario información sobre servicios. Es aquí donde se debe mencionar el poder de las aplicaciones SIG móviles, integradas en smartphones, tablets y notebooks, que permiten ver en tiempo real la posición actual del usuario y así informar a este de los servicios próximos a su ubicación, tales como entretenimiento, salud, estaciones de servicios y demás.

A diferencia de los sistemas LBS activos, los LBS pasivos son diseñados para aquellos clientes requieren administrar sus recursos móviles, como por ejemplo tener información en tiempo real de valores del mercado, localización de los vehículos de una flota, y a partir de allí realizar toma de decisiones.

2.3 Indoor vs Outdoor

Los datos que subyacen a los sistemas tradicionales de información geográfica (GIS) por lo general no contienen ni los detalles ni los sistemas de coordenadas para satisfacer las necesidades de los usuarios en un ambiente interior. Los Sistemas Basados en Localización (LBS) requieren detalles de georeferenciación más especializados para satisfacer las necesidades de los interiores. No es suficiente la georreferenciación de un edificio si las posiciones de los usuarios y de otros objetos en el interior del edificio son relevantes. Los objetos estáticos se utilizan como puntos de referencia (referencias espaciales), y las relaciones entre estos objetos son cruciales para la representación simbólica de un sistema.

Considerando como ejemplo cualquier centro comercial donde el nivel de detalle de mapeo de la ubicación del modelo subyacente determina qué funcionalidad puede ser proporcionada a la aplicación. Por ejemplo, si la aplicación se utiliza al aire libre (outdoor), para navegar entre puertas, es suficiente para volver a lugares de la granularidad de edificios, mientras que en el interior de una habitación del edificio, el usuario está en un nivel más fino de granularidad (tal como los espacios entre sillas y/o estantes). Como resultado, la información geográfica tradicional no es lo suficientemente detallada para satisfacer las necesidades del usuario en un ambiente de interior.

Un LBS necesita guiar a un usuario desde una ubicación actual a un destino específico. La navegación de interiores puede parecer diferente a la navegación para automóviles (en la red de calles). Considerando que la navegación para automóviles es un proceso relativamente

uniforme, que sigue la geometría de la carretera; los usuarios móviles caminan dentro de un edificio, por otro lado, son menos predecibles.

En Interiores, los usuarios de móviles no tienen un conjunto bien definido de vías, ya que pueden vagar en prácticamente cualquier espacio libre en el medio ambiente, mientras que los coches se circunscriben principalmente a circular sólo en la red de carreteras.

Sin embargo, los lugares de interiores no limitan a los usuarios móviles a un cierto nivel de acuerdo con la disposición del lugar, que puede ser modelado

(Explícitamente con los modelos topológicos) dentro del sistema de navegación en interiores. Los usuarios móviles pueden caminar hacia adelante, hacia atrás, y pueden cambiar el sentido de su caminar en cualquier momento, o se pueden mover arriba o abajo. Los desplazamientos verticales deben ser tenidos en cuenta debido a que los usuarios viajan arriba y abajo. [12]

2.4 Sistemas de Posicionamiento de Interiores (IPS)

Hoy en día, localizar a los usuarios a través de dispositivos móviles (que cuentan con gran capacidad de procesamiento), se ha vuelto muy importante para muchas de las aplicaciones desarrolladas en la actualidad. Estas aplicaciones necesitan o utilizan la posición actual del usuario en el mundo para brindarle diferentes tipos de información, como establecimientos de interés, sitios de comida, o simplemente compartir su posición en una red social.

Esta demanda creciente, es debida a que hace varios años, los dispositivos móviles no contaban con el nivel procesamiento de hoy en día, pero en la actualidad, el desarrollo de estos dispositivos y las nuevas aplicaciones, tienen un papel fundamental en la aparición de los sistemas de posicionamiento Outdoor e Indoor.

Un sistema de posicionamiento en interiores (IPS) es un término utilizado para una red de dispositivos que se utilizan para localizar de forma inalámbrica objetos o personas dentro de un edificio. En lugar de utilizar los satélites, un IPS cuenta con anclajes cercanos (nodos con una posición conocida). Los Sistemas de Navegación por GPS generalmente no son adecuados para ser utilizados en interiores, esto es debido a que las microondas se atenúan y dispersan por techos, paredes y otros objetos. Sin embargo, con el fin de hacer que la señal de posicionamiento se pueda conseguir en todas partes, la integración entre el GPS y el posicionamiento en interiores puede ser realizada mediante el uso de cualquier tecnología inalámbrica. [10]

Los sistemas de posicionamiento de detección indoor han tomado más relevancia en la actualidad, sin embargo tienen una tarea difícil.

Estos sistemas proporcionan una nueva capa “detección automática” de localización de objetos. Las aplicaciones en el mundo real que hoy cuentan con esta capa, son variadas, como lo puede ser una aplicación de detección de productos almacenados en una bodega, la detección de medicinas o equipamiento en una clínica, o la detección de la ubicación más exacta de un plantel profesional de bomberos en un edificio.

Los sistemas de detección indoor, son relativamente nuevos, por lo que en la actualidad, se realiza una investigación a nivel mundial por parte de industrias como también de alumnos y docentes del mundo académico.

Positivamente se le suma a estas investigaciones, el gran crecimiento en tecnologías inalámbricas, que se han sumado a distintos ámbitos, como el de logística, transporte y seguridad. A pesar de ello, existe una gran demanda hacia estas tecnologías, en base a su precisión del posicionamiento, tanto para sistemas indoor, como outdoor, que hoy en día no es nada fácil de suplir. [11]

2.4.1 Beneficios de IPS

Las nuevas aplicaciones de posicionamiento en interiores han despertado el interés tanto de los usuarios comunes, como también de aquellas personas que poseen distintos tipos de establecimientos y quieren hacer de ellos lugares conocidos comercialmente o turísticamente.

En los últimos años ha crecido notablemente la calidad y la prestación de los dispositivos móviles, pudiendo encontrar entre sus características GPS, Wi-Fi, Bluetooth y demás prestaciones que permiten a los desarrolladores de aplicaciones IPS aumentar las posibilidades de elección de tecnologías.

Sin embargo, estas tecnologías proporcionan información poco precisa sobre la ubicación de los usuarios, debido a que las señales disminuyen en lugares cerrados.

Se conocen varios sistemas, métodos y soportes para lograr el posicionamiento en interiores, basadas en redes inalámbricas como las mencionadas anteriormente, las que requieren una costosa infraestructura y complicaciones a la hora de realizar sus instalaciones.

Los sistemas IPS, aunque en la actualidad aún no pueden dar todo su potencial y son objeto de un intenso estudio e investigación, tienen y tendrá importantes beneficios para los usuarios. Estos sistemas facilitan a los usuarios la ubicación, orientación y el guiado dentro de diferentes tipos de establecimientos, como centros comerciales, parques de atracciones, cocheras cubiertas para localizar un vehículo, Hospitales, aeropuertos, tiendas, hoteles, museos y diferentes lugares turísticos.

Muchas veces en la vida diaria, conocer la posición del usuario es un requisito imprescindible. Los actuales sistemas de posicionamiento global no permiten la localización con precisión de dispositivos en interiores. Debido a esto, se están desarrollando nuevas tecnologías de sensores y perfeccionamiento de las ya tecnologías ya utilizadas para nuevas y más precisas herramientas para el posicionamiento o localización en interiores.

Actualmente las redes inalámbricas están orientadas a dar cobertura tanto a necesidades empresariales y particulares, como a necesidades públicas en estaciones, hospitales, universidades, en centros de ocio, hoteles, cafeterías, bares, y para muchas de estas actividades, que se realizan en estos entornos interiores, se necesita conocer la posición en la que se encuentran los usuarios. [13] [14]

2.4.2 Ejemplos de IPS

En esta sección se da una breve explicación de los distintos sistemas de localización desde el punto de vista de hardware necesario para su funcionamiento. Se describirán algunas de las versiones comerciales implementadas:

Active Bat: Este sistema utiliza una red de receptores ultrasónicos, que reciben la señal del dispositivo, determinando así la ubicación del usuario. La sincronización viene por parte de una señal de radio. Tiene una precisión de unos 10 centímetros en el 95% de los casos.

Cricket: Utiliza una red de sensores, señales de radio como medio de sincronización y señales ultrasónicas como medio de localización. Alcanza una precisión con un margen de error de 2 cms. Presenta una escalabilidad descentralizada y una alta privacidad, no obstante tiene altos requisitos de computación por la necesidad de precisos sincronismos.

AHLos: La principal ventaja de este sistema es su instalación dinámica, la red se despliega haciendo uso de infraestructura ya existente como GPS (funciona tanto en exteriores como interiores) para localizar sus propias balizas. Los nodos estiman su posición y distancia a los vecinos a partir de los nodos que ya conocen. El mayor inconveniente son los costos de computación.

RADAR: sistema desarrollado por Microsoft, basado en redes inalámbricas IEEE 802.11. RADAR mide la potencia y la relación señal-ruido y la emplea para posicionar el dispositivo en un mapa previamente muestreado a la base de fingerprints (huellas). Tiene un margen de error de unos 2-3 metros con solo cuatro puntos de acceso 802.11. Es un sistema de bajo coste ya que no requiere otra infraestructura más que los routers web que estén instalados.

PinPoint 3D-iD: se colocan sensores que emiten secuencialmente señales mientras el dispositivo a localizar emite su propia señal. La medida de tiempo de estas dos señales se emplea para calcular la posición.

MotionStar magnetic tracker: es un sistema de gran precisión espacial (1 mm y 0.1° de orientación de margen de error) y una gran resolución temporal (1 ms de refresco). Genera un sistema de seguimiento a través de pulsos continuos magnéticos, mide la posición y la orientación a través de una serie de antenas que reciben la respuesta en tres ejes espaciales. Su principal desventaja es que los sensores deben estar a una distancia de entre 1 y 3 metros del emisor.

Identificación por radiofrecuencia (RFID): utiliza etiquetas (tags) activas, estas almacenan y recuperan datos a través de transmisiones electromagnéticas a circuitos integrados compatibles con radio frecuencia (RF). Estos circuitos con antenas pueden ser tanto pasivas (no requieren una fuente de alimentación interna y solo emiten señal cuando reciben una en un alcance menor a dos metros) como activas (requieren fuente de alimentación propia y pueden emitir señal periódicamente a mayor distancia o realizar tareas más complejas, son de mayor costo). Se basa en un algoritmo en tres dimensiones de agregación a través de la intensidad de la señal de radio. Estos sistemas suelen ser utilizados en sistemas de localización de equipajes en aeropuertos, o seguimiento de paquetes en almacenes industriales.

Smart Floor: Consiste en una serie de sensores de presión en el suelo para capturar pisadas para localizar y seguir a peatones. Un problema que presenta es su escasa

escalabilidad, dado que requiere un sensor por cada punto donde se localiza.

QR codes[3]: Son códigos de barras en dos dimensiones los cuales pueden codificar información como una URL, y esta al ser leída desde un móvil obtiene la posición dentro del edificio. Al igual que Smart Floor su principal desventaja es la escalabilidad, ya que es necesario un QR code por cada punto referenciable, la ventaja de este método es su simplicidad y bajo costo. [15]

2.5 Sistemas de Street View

Los sistemas de Street View, o "vista de calles", son aquellos que brindan servicios en internet, desde los que es posible observar recorridos virtuales de ciudades, calles y de estructuras conocidas e importantes de todo el mundo.

El recorrido en este tipo de servicios, se realiza con el cursor de la PC, o con el teclado directamente, pudiendo desplazarse por los lugares disponibles en la vista.

Para implementar este tipo de sistemas es necesario realizar una recopilación de imágenes por medio de equipamiento adecuado para la toma de las fotografías en 360º.

Luego de tomar todas las fotografías, es necesario relacionar todas estas con su ubicación geográfica, y es allí donde toma un papel muy importante el GPS. Combinando señales, velocidad, dirección, y demás datos proporcionados por el GPS se puede reconstruir una ruta, calle, o lugar exacto.

Todas las imágenes tomadas son adyacentes pero no perfectas, por lo que para evitar espacios o superposición de imágenes, se procesan las imágenes con algoritmos especiales para reducir desperfecciones.

Por último, todas las fotografías tomadas son modificadas antes de la publicación final en el sistema, realizando la pixelacion de rostros y matrículas de vehículos, para evitar romper con la privacidad interna de los diferentes países.

Street View posee muchos beneficios, como conocer un espacio antes de visitarlo, su fachada, ver cómo es un hotel o una casa antes de hacer la reserva, evaluar diferentes destinos de viaje. También le proporciona a comercios, hoteles, inmobiliarias la capacidad de mostrar sus interiores, los servicios que brindan. Otro beneficio, y muy importante, es su utilización como servicio cultural y educativo, ya que puede mostrarse zonas y lugares en las cuales ocurrieron acontecimientos.

En las Figuras 2,3 y 4 se puede apreciar ejemplos de uso de Street View.

[3] Un código QR (*quick response code*, «código de respuesta rápida») es un módulo para almacenar información en una matriz de puntos o un código de barras bidimensional.

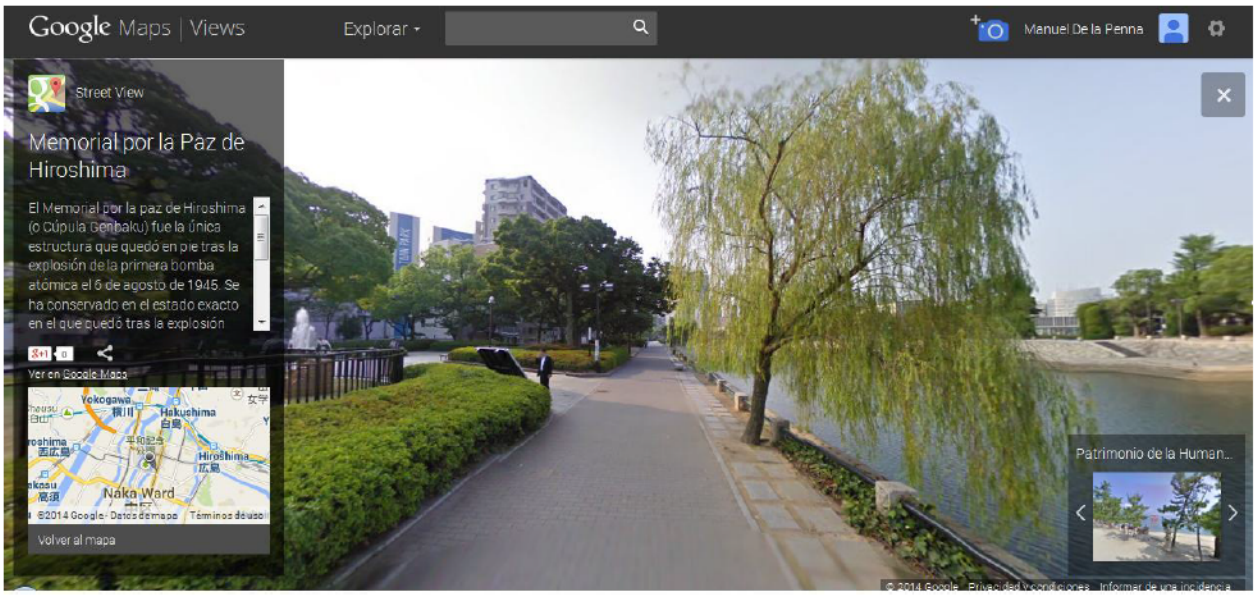


Figura 2. Captura de imagen de Google Street View.

Figura 3. Captura de imagen de Google Street View luego de avanzar unos metros hacia adelante.

Figura 4. Captura de imagen de Google Street View luego de rotar a la izquierda.

3. Background

En este capítulo se presentará la aplicación 2D que se utilizará de base en esta tesis. Además se describirán las tecnologías utilizadas para la realización de la misma.

3.1 Aplicación de Navegación 2D usada de base

La aplicación Indoor-Navigation usada de base tiene como objetivo principal guiar a los usuarios mediante el uso de dispositivos móviles, desde un punto hacia otro dentro de un edificio.

La forma de posicionar a un usuario dentro de un edificio se realiza de forma abstracta, haciendo una aplicación en la cual la tecnología de posicionamiento pueda cambiar y la aplicación siga respondiendo de la misma manera, es decir, sea capaz de representar la posición del usuario en un mapa del edificio, mostrando el camino al destino que eligió y guiándolo en el camino.

Para la captura de la posición del usuario se utiliza la tecnología de códigos QR, codificando así, ubicaciones de puntos de navegación para que los usuarios puedan leerlos con sus dispositivos móviles y así poder informarle a la aplicación su ubicación actual.

La aplicación representa una estructura navegable basada en el Edificio de la Facultad de Informática situada en 50 y 120. Construido en base a un plano de la primera etapa de construcción de la Facultad.

La ejecución de Indoor-Navigation comienza con la lectura de un código QR, dentro del cual se encuentra codificada una URL con el parámetro de la posición actual que le indica a Indoor-Navigation la ubicación actual del usuario.

Una vez ingresado el QR con la URL, el browser del dispositivo móvil despliega una interfaz, donde se encuentra un listado con todas las estructuras navegables posibles dentro de la facultad. La Figura 5 ilustra dicho listado.

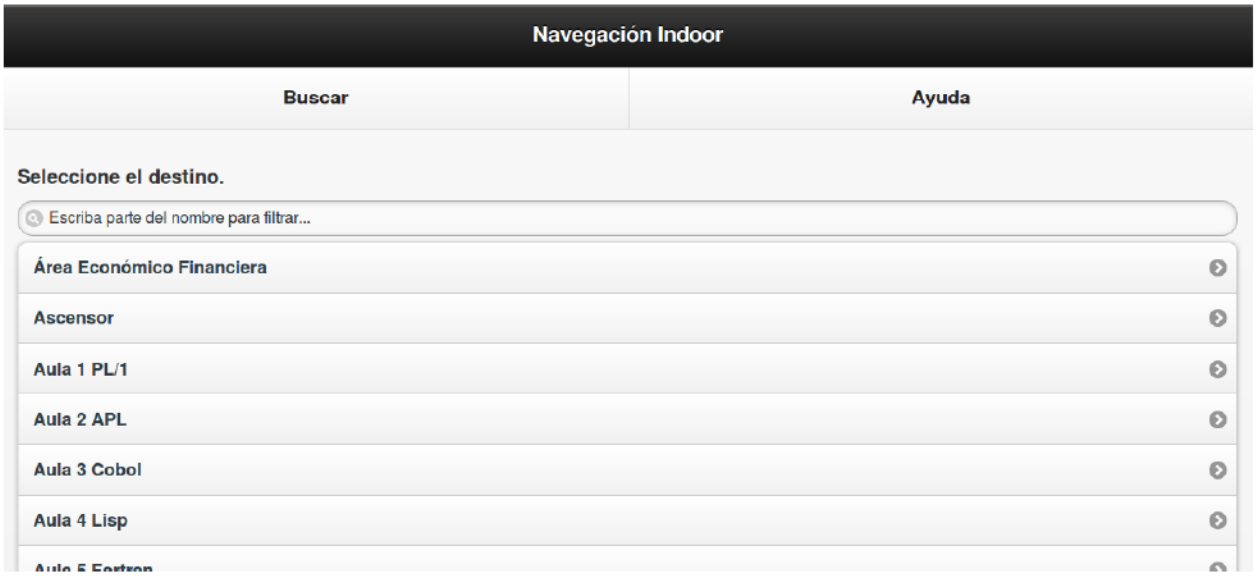


Figura 5. Captura de la interfaz de listado de estructuras navegables.

Una vez seleccionado un destino de la lista desplegada, el usuario es trasladado a la pantalla de navegación. En la Figura 6 se muestra como un usuario navega desde la puerta de la facultad (lectura del código QR) hasta la selección realizada posteriormente.

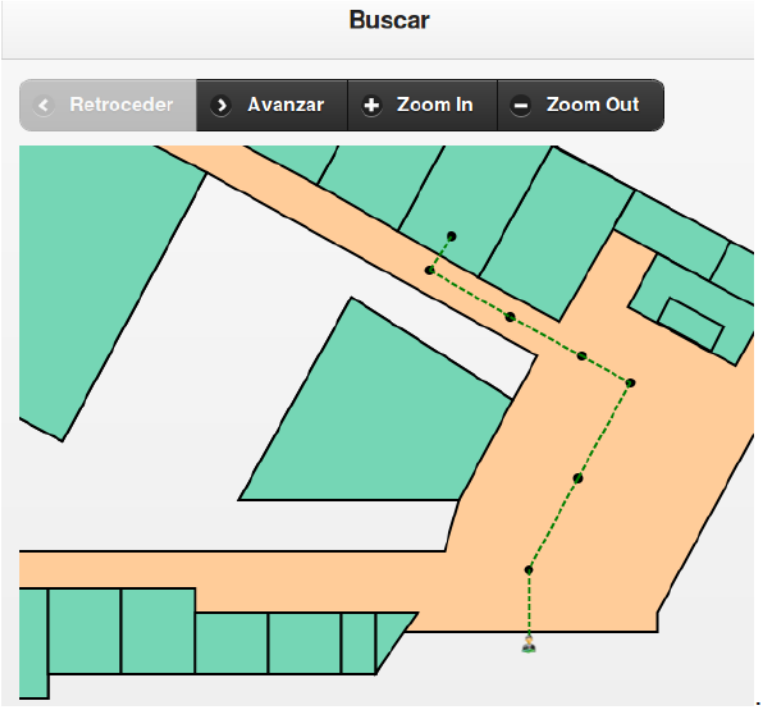


Figura 6. Captura de mapa 2D con recorrido establecido.

Para dibujar el plano del edificio, los caminos, la posición del usuario y los puntos de navegación se utiliza el tag <canvas> de HTML5, y la librería javascript KineticJS.

La aplicación obtiene los datos de cada una de las estructuras instanciadas, los puntos de navegación, calcula el camino y deja todos estos datos disponibles para que la capa de presentación los utilice.

Las estructuras se crean situando los vértices de estas con las coordenadas de los puntos asociados a la estructura a dibujar. El color de la estructura depende de si es navegable (color verde) o no (color salmón), y a cada estructura creada se le asigna un handler para manejar el evento “click” o “touch” , el cual cuando sea seleccionado pedirá al sistema información detallada sobre la estructura en cuestión.

El camino calculado por la aplicación está compuesto por una colección de puntos de navegación. Para ser representados en el plano se recorre esta colección y se dibujan utilizando KinecticJS en las coordenadas asociadas al punto de navegación y se dibuja con líneas punteadas la unión de los puntos de navegación conformando así el camino a seguir.

El primer punto de navegación de la colección representa la posición actual del usuario (el lugar donde leyó el código QR y desde donde inicia la navegación), su ubicación es representada por el icono de una persona.

El usuario a medida que avanza en el edificio real podrá seguir leyendo los códigos QR que encuentre en los puntos de navegación que atraviesa y así su posición en el mapa de la aplicación será actualizada.

Para simular la lectura de códigos QR la aplicación cuenta una barra de navegación en la parte superior del mapa que permite “Avanzar” al siguiente punto de navegación en el camino y así actualizar su posición en la aplicación. Esto se puede apreciar en la Figura 7.

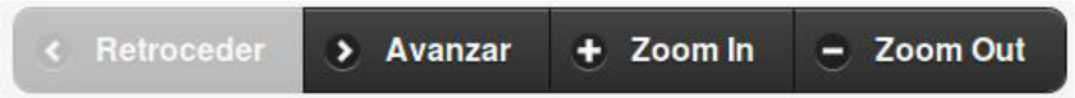


Figura 7. Barra de navegación de mapa 3D.

El botón retroceder, permite volver al punto de navegación anterior en el camino. en una primera instancia se encuentra deshabilitado ya que el usuario no puede retroceder debido a que se encuentra en el punto inicial de su navegación.

Cada vez que el usuario modifica su posición, el plano es redibujado. La barra de navegación permite hacer zoomIn y zoomOut, existiendo cinco niveles de zoom en la aplicación, por defecto se utiliza el intermedio, hay 2 niveles de zoomOut y 2 niveles de zoomIn.

Durante la navegación, es posible consultar las distintas estructuras. Al realizar un clic sobre una estructura navegable, se obtiene un detalle en el cual se pueden ver fotos, textos y demás información asociada.

La Figura 8, muestra el detalle de realizar click sobre la Biblioteca de la facultad de informática.

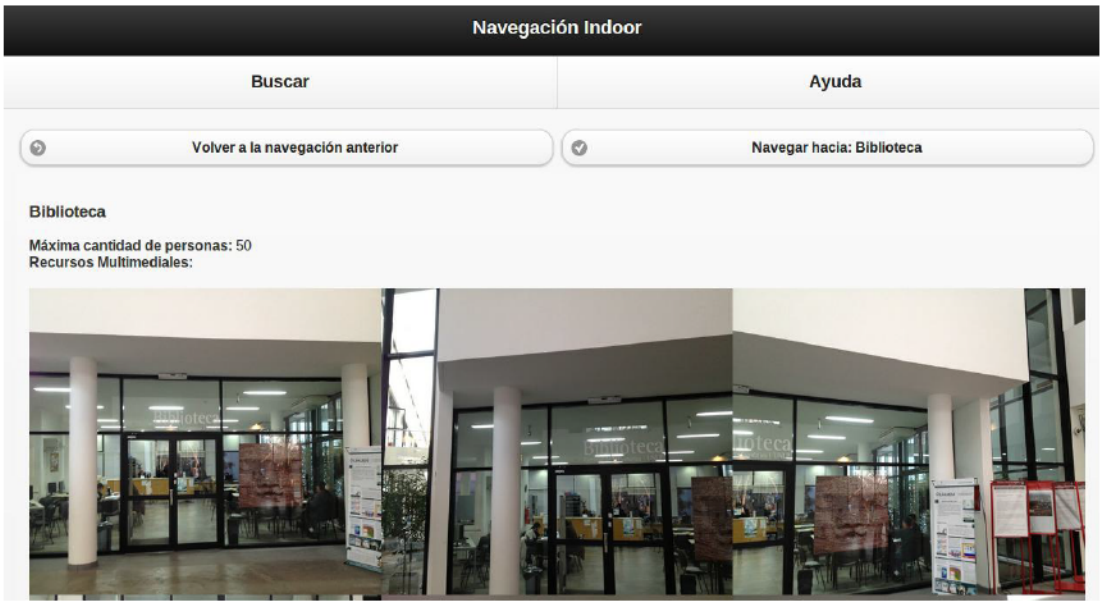


Figura 8. Captura de información detallada de la Biblioteca.

La Figura 9 muestra la opción de seleccionar una estructura como nuevo destino desde su detalle, haciendo que el sistema realice el cálculo del camino nuevamente entre la estructura actual seleccionada y la posición actual del usuario.

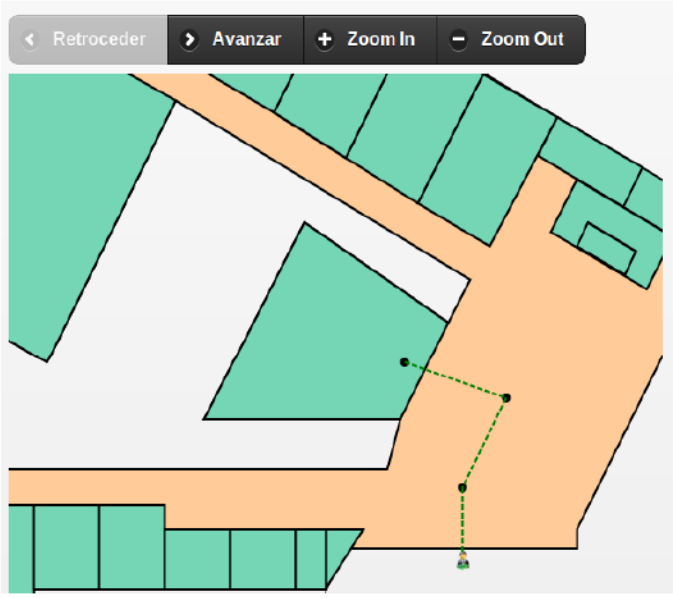


Figura 9. Mapa en dos dimensiones con recorrido establecido.

Avanzando por el camino trazado por la aplicación, al llegar al punto de navegación más cercano a la estructura de destino la aplicación muestra un mensaje que indica la llegada al destino antes seleccionado:

Cerrar el popup y ver el mapa para elegir un nuevo destino en el mapa, abriendo el detalle de una estructura y marcando la misma como nuevo destino, o yendo a la sección de Búsqueda de destino y así comenzar una nueva navegación, donde la posición actual del usuario será la estructura a la que llegó recientemente. Como se puede apreciar en la Figura 10.

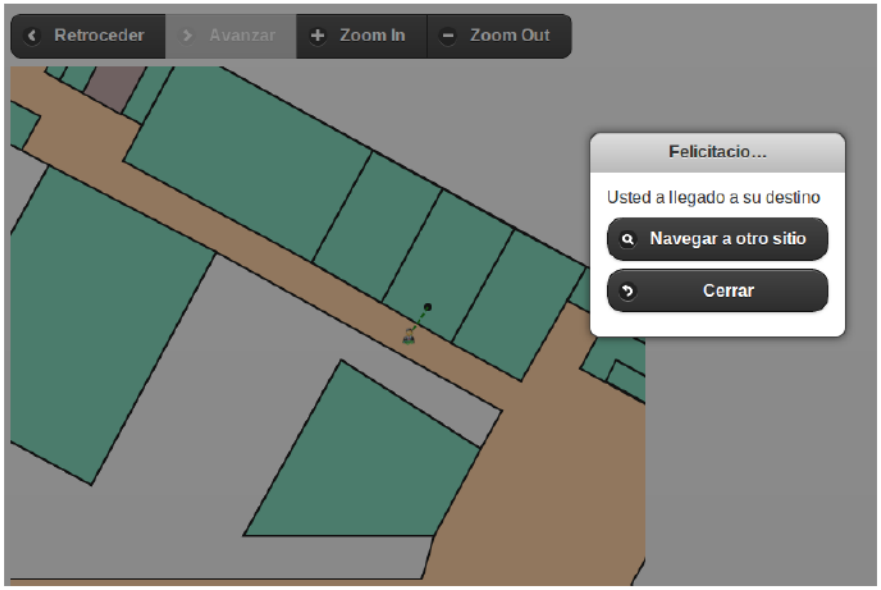


Figura 10. Captura de pantalla luego de realizar un recorrido completo.

3.1.1 Descripción del funcionamiento de la aplicación original.

La aplicación original fue planteada íntegramente bajo la arquitectura que se presenta en la Figura 11, es decir, funciona por medio de peticiones del cliente que posteriormente se procesan en el servidor. Luego el servidor le responde al cliente, quien, por medio de la librería KineticJS, sumado a HTML5 y Javascript muestra de manera precisa en pantalla la respuesta del servidor. Como se puede observar, todos los cálculos se realizan en el servidor y no en el cliente, lo cual permite que la aplicación corra con relativamente bajos requerimientos.

La aplicación original no utiliza AJAX, con lo cual la página se recarga cada vez que se hace una petición, simplificando el proceso de desarrollo pero por otra parte obligando al dispositivo móvil a recargar una y otra vez la página en cada petición. Este es un punto clave a tener en cuenta en el desarrollo en tres dimensiones que se explicará con mayor profundidad en el siguiente punto.

La aplicación original no proporciona ningún mecanismo de lectura de códigos QR integrado en la aplicación, éstos deben ser leídos mediante software adicional instalado en el dispositivo móvil.

Finalmente, la aplicación original no distingue entre posicionamiento virtual y físico, únicamente posee posicionamiento físico.

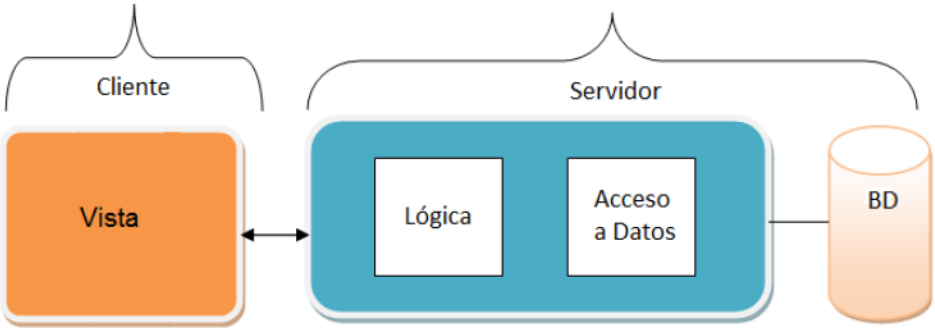


Figura 11. Arquitectura de la Aplicación de Navegación 2D.

3.2 Tecnologías y Herramientas utilizadas

3.2.1 HTML5

HTML5 es la última evolución de la norma que define HTML, dicha versión posee nuevos elementos, atributos y comportamientos con respecto a la anterior y dispone de un conjunto más amplio de tecnologías que permite que los sitios Web y las aplicaciones sean más diversas y de gran alcance.

Las mejoras de HTML5 se clasifican en:

- **Semántica**: permite describir con mayor precisión cuál es su contenido.
- **Conectividad**: permite comunicarse con el servidor de formas nuevas e

innovadoras.

- **Desconectado y almacenamiento**: permite a páginas web almacenar datos, localmente, en el lado del cliente y operar fuera de línea de manera más eficiente.
- **Multimedia**: permite reproducir vídeo y audio en la Web abierta.
- **Gráficos y efectos 2D/3D**: permite una gama mucho más amplia de opciones de presentación.
- **Rendimiento e Integración**: proporciona una mayor optimización de la velocidad y un mejor uso del hardware del equipo.
- **Dispositivo de Acceso**: admite el uso de varios dispositivos de entrada y salida.
- **Styling**: deja a los autores escribir temas más sofisticados. [16]

Para el presente trabajo se aprovecharán al máximo las mejoras en lo que respecta a gráficos y efectos 2D y 3D (para dibujar y representar la facultad, tanto en dos como en tres dimensiones), rendimiento e integración (para que la aplicación funcione lo más rápido posible utilizando recursos de hardware del cliente como por ejemplo la GPU y la cámara del dispositivo) y dispositivos de acceso (para lograr conectar la cámara del dispositivo y así lograr leer códigos QR).[4]

3.2.2 KineticJS

KineticJS es una librería de JavaScript que extiende el contexto 2D para Canvas (propuesto en HTML5), permitiendo dibujar formas o imágenes con la API existente, añadir detectores de eventos, moverlos, escalarlos, y girar de forma independiente de otras formas o imágenes. Este framework está licenciado bajo las licencias MIT y GPL. y puede descargarse tanto desde el sitio oficial[5] como de los repositorios de GitHub.[6]

Esta librería fue utilizada en el desarrollo anterior para dibujar el plano en dos dimensiones de la Facultad de Informática. En el trabajo actual se conservará esta vista en un segundo plano para orientar mejor la navegación en tres dimensiones.

El objeto principal que utiliza KineticJS se llama Stage (traducido como escenario), que se asocia directamente con el Canvas. Luego están los objetos Layer (capas) que se insertan en el Stage y que a su vez contienen los elementos gráficos, por ejemplo: círculos, rectángulos, polígonos, imágenes, etc.

Ejemplo de instanciación de KineticJS

1. Creación de un escenario (Stage) y dos capas (Layers), se muestra en la Figura 12.

---

[4] Para mayor información acerca de HTML5 se puede consultar el siguiente enlace: http://www.w3.org/html/wg/drafts/html/master/
[5] http://kineticjs.com
[6] https://github.com/ericdrowell/KineticJS

```
window.addEvent('domready',function() {

    // creacion del escenario
    var escenario = new Kinetic.Stage({
        container: 'miCanvas',
        width: 640,
        height: 480,
    });

    var principal = new Kinetic.Layer();
    var fondo = new Kinetic.Layer();
```

Figura 12. Captura de código: creación escenario KineticJS.

2. Creación de un círculo (Circle) dentro de la capa llamada “principal” (ver Figura 13)

```
// creacion de un circulo
var circle = new Kinetic.Circle({
    x: 75,
    y: 75,
    radius: 50,
    fill: "blue",
    stroke: "#000",
    strokeWidth: 2
});
principal.add(circle);
```

Figura 13. Captura de código: creación círculo KineticJS.

3. Creación de un rectángulo(Rect) dentro de la capa llamada “principal” (ver Figura 14)

```
// creacion de un rectangulo
var rect = new Kinetic.Rect({
    x: 150,
    y: 100,
    width: 120,
    height: 80,
    fill: "green",
    stroke: "#000",
    strokeWidth: 2,
    draggable: true
});

principal.add(rect);
```

Figura 14. Captura de código: creación rectángulo KineticJS.

4. Creación de una imagen (Image) dentro de la capa llamada “fondo” (ver Figura 15)

```
1   // creacion de una imagen
2   var imagen = new Image();
3   imagen.src = 'playa.jpg';
4
5   var imgFondo = new Kinetic.Image({
6       image: imagen,
7       x: 0,
8       y: 0,
9       width: 640,
10      height: 480
11  });
12
13  fondo.add(imgFondo);
```

Figura 15. Captura de código: creación rectángulo KineticJS.

5. Finalmente se asignan las capas al escenario y se le da la orden al escenario de dibujarse para ver el resultado en el navegador. Como se muestra en la Figura 16.

```
1
2
3       escenario.add(fondo);
4       escenario.add(principal);
5       escenario.draw();
6
7   });
```

Figura 16. Captura de código: Asignaciones KineticJS.

Como se puede observar, al momento de crear círculos, rectángulos, etc., entre llaves se pueden pasar muchos parámetros para configurar los. x es la coordenada horizontal sobre la que se ubicará; y es la coordenada vertical (las verticales se incrementan hacia abajo). radius es el radio (en el caso de círculo). fill es el color de relleno. stroke, el color del borde. strokeWidth,el grosor del borde. draggable permite drag and drop.

3.2.3 WebGL

Antes de hablar de WebGL se darán a conocer otros estándares necesarios para comprender qué es y cómo funciona WebGL.

OpenGL (Open Graphics Library) es una especificación estándar que define una API (application programming interface) multiplataforma para escribir aplicaciones que contengan gráficos 2D y 3D. OpenGL está manejado por el grupo tecnológico Khronos Group, el cual es un consorcio industrial sin fines de lucro encargado de crear estándares abiertos para permitir la creación y aceleración de la computación paralela, gráficos y medios dinámicos en una variedad de plataformas y dispositivos.

OpenGL ES 2.0 (OpenGL for Embedded Systems) es una variante simplificada de OpenGL para dispositivos integrados, tales como smartphones, PDAs, consolas, entre otros. Consiste

de un subconjunto bien definido de OpenGL. Permite la programación completa de gráficos 3D. Al igual que OpenGL, esta manejado por Khronos Group. Toda la familia de OpenGL tiene una característica muy importante: la aceleración por hardware, esta consiste en el uso del hardware para desempeñar algunas funciones mucho más rápido de los que es posible en software corriendo en la CPU de propósito general. De esta manera se utiliza la GPU de la tarjeta gráfica para procesar grandes cargas de gráficos.

La GPU (Graphics Processing unit) o unidad de procesamiento de gráficos es un procesador dedicado al procesamiento de gráficos u operaciones de coma flotante.

WebGL fue creado inicialmente por Mozilla, y más tarde estandarizado por el grupo tecnológico Khronos Group, el mismo grupo responsable de OpenGL y OpenGL ES. El primer prototipo fue diseñado por Mozilla en el año 2006 y a principios del 2009, Mozilla y Khronos Group comenzaron el WebGL Working Group. Además de los ya mencionados, actualmente los principales fabricantes de navegadores, Apple (Safari), Google (Chrome) y Opera (Opera), así como algunos proveedores de hardware son miembros del grupo de trabajo WebGL o WebGL Working Group. Todos ellos están interesados en verificar que el contenido WebGL pueda correr tanto en desktop y hardware de dispositivos móviles.

WebGL (Web-based Graphics Library) es un estándar web multiplataforma para una API de gráficos 3D de bajo nivel basado en OpenGL ES 2.0 y expuesto a través del elemento canvas de HTML5 como interfaces DOM (Document Object Model). Esta API provee enlaces de JavaScript a funciones OpenGL haciendo posible proveer contenido 3D acelerado en hardware a las páginas web. Esto hace posible la creación de gráficos 3D que se actualizan en tiempo real, corriendo en el navegador.

Podemos decir también que WebGL, es una librería de software que extiende al lenguaje de programación javascript para permitir generar gráficos interactivos 3D dentro de cualquier navegador web compatible. WebGL es un contexto del elemento canvas que provee un API de gráficos 3D sin la necesidad de plugins. Decir que WebGL es un contexto del elemento canvas, podría no entenderse hasta que se muestre como trabajan juntos en la siguiente sección.

WebGL trae el API de OpenGL ES 2.0 al elemento canvas. El contenido 3D está limitado a canvas. Se hicieron cambios en la API de WebGL en relación con la API de OpenGL ES 2.0 para mejorar la portabilidad a través de varios sistemas operativos y dispositivos.

En la Figura 17 se muestra un modelo conceptual de la arquitectura de WebGL.

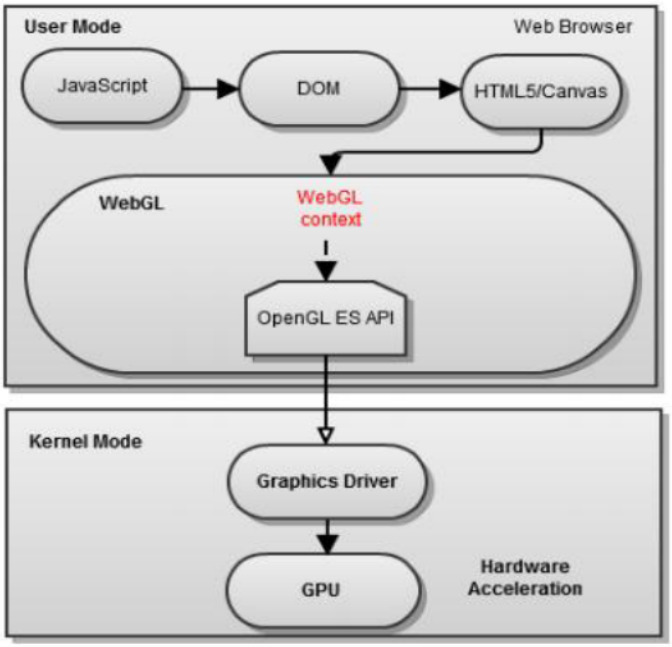


Figura 17. Modelo conceptual de la arquitectura WebGL

En este modelo se utiliza JavaScript para obtener a través del DOM el elemento Canvas de HTML5. Una vez obtenido el elemento Canvas, se define el contexto WebGL, por medio del cual accedemos a la API de WebGL, la cual está basada en la API de OpenGL ES. Por eso se dice que, técnicamente es un enlace (binding) para JavaScript para usar la implementación nativa de OpenGL ES 2.0.

Este último, se encarga de comunicarse con el driver de la tarjeta gráfica y así poder realizar la aceleración por hardware en la GPU. Como vemos, se diferencian los eventos que ocurren en el espacio de usuario y en el kernel.

La API de WebGL interactúa bien con el resto de las plataformas web; específicamente se proporciona apoyo para la carga de texturas 3D a partir de imágenes HTML o video, y la entrada del teclado y mouse se manejan mediante los conocidos eventos DOM. Este API poderoso es de muy bajo nivel y su uso requiere buenos conocimientos acerca de la programación 3D y matemática 3D. Una de las principales desventajas de WebGL es el alto consumo de CPU.

Una característica importante de WebGL es que, brinda la posibilidad de contenidos 3D a la web sin la necesidad de utilizar plugins, ya que se encuentra implementado en el navegador.

WebGL es un estándar web. Un estándar web, es un conjunto de especificaciones técnicas en constante evolución y de buenas prácticas para construir sitios web. Con ello se facilita el mantenimiento, la usabilidad, la interoperabilidad y la calidad de los trabajos. Utilizar un estándar asegura una larga vida a los proyectos, ya que provee por lo menos una pequeña estructura de mantenimiento. También asegura de que la mayoría de las personas puedan visitar el sitio web, sin importar qué navegador se esté utilizando. La compatibilidad hacia adelante y hacia atrás (forward and backward compatibility) es posible.

En el caso particular de WebGL, su API está basado en un estándar de gráficos 3D familiar

y ampliamente aceptado. No utilizar un estándar tiene varias desventajas, como poco soporte, mayor exposición a problemas de seguridad y problemas de performance. Además la curva de aprendizaje suele ser muy alta, con un costo-beneficio menor. Otro problema que se puede dar es el del vendor lock in, en donde se tiene una dependencia absoluta del proveedor, el cual puede tener un costo elevado.

El lanzamiento (release) de la especificación de la versión final de WebGL 1.0 se dio el 3 de marzo del 2011 en la conferencia de desarrolladores de juego (Game Developers Conference - GDC), en San Francisco, Estados Unidos.

Debido a que WebGL posee un API de muy bajo nivel, ha provocado el rechazo por parte de algunos programadores. Gracias a su creciente uso, muchos desarrolladores han trabajado duro para proveer una API de alto nivel, a través de la creación de varios frameworks. Estos frameworks son sólidos y garantizan que corren en cualquier dispositivo, haciendo la programación más fácil. Esto permite que los desarrolladores web no tengan que lidiar con las complejidades WebGL y la matemática 3D.

Algunos frameworks son: C3DL, CopperLicht, CubicVR, EnergizeGL, GammaJS, GLGE, GTW, Jax, O3D, Oak3D, PhiloGL, SceneJS, SpiderGL, TDL, Three.js y X3DOM. La mayoría son librerías JavaScript.[17]

Por cuestiones como la popularidad, disponibilidad de documentación y funcionalidades, se eligió la librería Three.js para el desarrollo de la aplicación.

3.2.3.1 WebGL: Soporte de Navegadores.

En la etapa de desarrollo de la tesis “IndoorNavigation 2D”, el grupo de trabajo se encontró con una API WebGL en pleno desarrollo, contando con herramientas a muy bajo nivel, y una compatibilidad de los navegadores mínima.

En la Figura 18 se puede apreciar el grado de compatibilidad con WebGL de los principales navegadores en los años 2011/2012.

= Supported = Not supported = Partially supported = Support unknown

WebGL - 3D Canvas graphics - Other

Method of generating dynamic 3D graphics using JavaScript, accelerated through hardware

Resources: Firefox blog post Polyfill for IE Instructions on enabling WebGL Webkit blog post Tutorial

Global user stats*:	
Support:	28.76%
Partial support:	24.78%
Total:	53.54%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
19 versions back			4.0									
18 versions back			5.0									
17 versions back		2.0	6.0									
16 versions back		3.0	7.0									
15 versions back		3.5	8.0									
14 versions back		3.6	9.0									
13 versions back		4.0	10.0									
12 versions back		5.0	11.0									
11 versions back		6.0	12.0									
10 versions back		7.0	13.0		9.0							
9 versions back		8.0	14.0		9.5-9.6							
8 versions back		9.0	15.0		10.0-10.1							
7 versions back		10.0	16.0		10.5							
6 versions back		11.0	17.0		10.6							
5 versions back	5.5	12.0	18.0	3.1	11.0			2.1				
4 versions back	6.0	13.0	19.0	3.2	11.1	3.2		2.2		10.0		
3 versions back	7.0	14.0	20.0	4.0	11.5	4.0-4.1		2.3		11.0		
2 versions back	8.0	15.0	21.0	5.0	11.6	4.2-4.3		3.0		11.1		
Previous version	9.0	16.0	22.0	5.1	12.0	5.0-5.1		4.0		11.5		
Current	10.0	17.0	23.0	6.0	12.1	6.0	5.0-7.0	4.1	7.0	12.0	18.0	15.0
Near future		18.0	24.0		12.5				10.0	12.1		
Farther future		19.0	25.0									

Figura 18. Cuadro de compatibilidad con WebGL de los navegadores. Año 2012.[7]

Como se puede observar en esta Figura, los navegadores con mayor compatibilidad eran sólo aquellos navegadores de pertenecientes a PCs, mientras que los navegadores pertenecientes a dispositivos móviles como Safari, Opera Mini, o Chrome aún no eran compatibles.

Con el correr de los años, el desarrollo y la importancia de WebGL aumentó notablemente, acompañado por el gran desarrollo producido en tecnologías móviles. Tanto los smartphones, como tablets y otros dispositivos móviles, actualmente cuentan con soporte para HTML5, que recién en el año 2012 comenzó a introducirse en este tipo de dispositivos.

Los procesadores de los dispositivos móviles actuales, cuentan con un aumento de su velocidad muy grande comparada con años anteriores, además de agregar tarjetas gráficas dedicadas, por lo que al WebGL hacer uso de aceleración por hardware, los dispositivos móviles actuales dan un soporte óptimo, siempre dependiendo, de las especificaciones particulares de cada dispositivo.

En la Figura 19 se puede ver la compatibilidad de los navegadores con la API WebGL del presente año 2014.

[7] http://caniuse.com/webgl (Año 2012)

= Supported = Not supported = Partially supported = Support unknown

WebGL - 3D Canvas graphics - **Other**

Method of generating dynamic 3D graphics using JavaScript, accelerated through hardware

Resources: Firefox blog post Polyfill for IE Instructions on enabling WebGL Webkit blog post Tutorial

Global user stats*:	
Support:	40.83%
Partial support:	21.8%
Total:	62.63%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
29 versions back			4.0										
28 versions back			5.0										
27 versions back		2.0	6.0										
26 versions back		3.0	7.0										
25 versions back		3.5	8.0										
24 versions back		3.6	9.0										
23 versions back		4.0	10.0										
22 versions back		5.0	11.0										
21 versions back		6.0	12.0										
20 versions back		7.0	13.0										
19 versions back		8.0	14.0										
18 versions back		9.0	15.0										
17 versions back		10.0	16.0										
16 versions back		11.0	17.0										
15 versions back		12.0	18.0		9.0								
14 versions back		13.0	19.0		9.5-9.6								
13 versions back		14.0	20.0		10.0-10.1								
12 versions back		15.0	21.0		10.5								
11 versions back		16.0	22.0		10.6								
10 versions back		17.0	23.0		11.0								
9 versions back		18.0	24.0		11.1								
8 versions back		19.0	25.0		11.5								
7 versions back		20.0	26.0	3.1	11.6			2.1					
6 versions back	5.5	21.0	27.0	3.2	12.0			2.2		10.0			
5 versions back	6.0	22.0	28.0	4.0	12.1	3.2		2.3		11.0			
4 versions back	7.0	23.0	29.0	5.0	15.0	4.0-4.1		3.0		11.1			
3 versions back	8.0	24.0	30.0	5.1	16.0	4.2-4.3		4.0		11.5			
2 versions back	9.0	25.0	31.0	6.0	17.0	5.0-5.1		4.1		12.0			
Previous version	10.0	26.0	32.0	6.1	18.0	6.0-6.1		4.2-4.3	7.0	12.1			
Current	11.0	27.0	33.0	7.0	19.0	7.0	5.0-7.0	4.4	10.0	16.0	33.0	26.0	10.0
Near future		28.0	34.0		20.0								
Farther future		29.0	35.0		21.0								
3 versions ahead		30.0	36.0										

Figura 19. Cuadro de compatibilidad con WebGL de los navegadores. Año 2014.[8]

Se puede observar como con el pasar de los años, ha aumentado con gran notoriedad el soporte a WebGL por parte de los navegadores, tanto el PCs mejorando el rendimiento, como de los dispositivos móviles.

WebGL provee una URL (http://get.webgl.org/), en la cual solo ingresando a ella, se da conocimiento si el navegador soporta o no WebGL.

[8] http://caniuse.com/webgl (Año 2014)

3.2.4 Three.js

Three.js es un framework de JavaScript, dedicado a facilitar la programación en cuanto a la creación de elementos, terrenos, figuras, tanto en el plano 2D, como en el 3D, abstrayendo la especificación WebGL, y funciona bajo los estándares de HTML5.

Básicamente, la librería maneja tres conceptos importantes que se explicarán a continuación para poder comprenderla.

- El Renderer.
- La escena.
- La cámara.

El Renderer

El renderer es el objeto principal de WebGL, es donde la GPU de la PC ubicará, dibujara y pintará todos los elementos que se deseen. Para simplificar la explicación, podría verse como el “canvas” de la aplicación.

Un terreno, o mundo de tres dimensiones puede ser enorme, infinito, pero los usuarios sólo verán todo aquello que se vea desde la "cámara" y su perspectiva en la pantalla. Ésta que nosotros vemos, la perspectiva 3D, es conocida como Renderer.

Para crear un nuevo render, es necesario seterarle como mínimo tres atributos: su ancho, su alto y su color inicial. Este color podría verse como el color del “espacio” que rodea la escena que queremos dibujar en el canvas. Un ejemplo puede observarse en la Figura 20.

```
1  var renderer;
2
3  var canvasWidth = 500;
4  var canvasHeight = 500;
5
6  renderer = new THREE.WebGLRenderer();
7  renderer.setClearColorHex(0x000000, 1);
8  renderer.setSize(canvasWidth,canvasHeight);
9
```

Figura 20. Captura de ccódigo: Definición de renderer en Three.js

La Escena

En Three.js, la escena es la que se encarga de contener todos los elementos y objetos que se dibujaran en el momento de correr la aplicación.

Dentro de una escena de Three.js, se pueden encontrar diferentes tipos de objetos, entre las cuales, las figuras y las cámaras son las más utilizadas.

En toda aplicación de Three.js, se debe contar con al menos una escena, donde se encontrarán todos los objetos, y serán visualizados dependiendo el enfoque de la cámara en cuestión. Esto no significa que solo se deba utilizar una escena, sino que pueden utilizarse todas las que se requieran, para poder darle toda la realidad posible a un entorno en tres

dimensiones.

Crear una escena en esta librería es muy simple, solo se necesita escribir el código presentado en la Figura 21.

```
1  var nombreescena;
2  nombreescena = new THREE.Scene();
```

Figura 21. Captura de código: Creación escena en Three.js

Para añadir luego objetos a la escena creada, es necesario el nombre de la escena seguido de .add, y enviando el objeto como parámetro, como se detalla en la Figura 22.

```
4  nombreEscena.add(objeto);
```

Figura 22. Captura de código: Agregación de escena en Three.js

La Cámara

Las cámaras toman un papel fundamental dentro de cualquier mundo 3D creado con Three.js, o de cualquier otro framework de rendering 3D. Esto se debe a que la perspectiva o ubicación que obtenga la cámara utilizada en un momento dado, mostrará una parte del mundo 3D, y no el todo.

En Three.js, una escena puede tener el número que deseemos, pero al momento de renderizar la escena, solo una de ellas podrá ser utilizada. Es posible avanzar, retroceder, girar, elevar, y realizar todo tipo de movimientos con la cámara, permitiendo cambiar el enfoque del mundo 3D.

Si bien existen varios tipos de cámara, PerspectiveCamera es la más utilizada. La Figura 23 muestra un ejemplo de cómo crear una en Three.js.

```
1  var camera = new THREE.PerspectiveCamera( 45, width / height, 1, 1000 );
```

Figura 23. Captura de código: Creación de cámara en Three.js

El constructor de la PerspectiveCamera posee 4 parámetros que deben enviarse para su correcta instanciación, los cuales se detallan a continuación:

FOV: Ángulo del campo de visión grados.
Aspect: Radio del aspecto, relación WIDTH/HEIGHT del canvas.
Near: Distancia mínima de dibujado
Far: Distancia máxima de dibujado

Una vez creada la cámara, es posible incluirla a la escena, como se muestra en la Figura 24.

```
3   nombreescena.add(camera);
```

Figura 24. Captura de código: Inclusión de cámara en escena en Three.js

Para mostrar en el renderer la captura de la cámara se ejecuta código presentado en la Figura 25.

```
12  renderer.render(escena, camara);
13
```

Figura 25. Captura de código: Mostrar renderer en escena en Three.js

Actualización de la visión de la cámara – RequestAnimationFrame

Para actualizar la vista de la cámara no alcanza con llamar una única vez al método render() del renderer ya que de esta manera la cámara mostrará únicamente una parte del movimiento dado por una fracción de tiempo. Para mantener actualizada continuamente la visión de la misma hay que llamar al método render() continuamente cada un período de tiempo fijo, de esta manera, cualquier movimiento que se produzca en la cámara será visualizado en pantalla.

Para esto es necesario implementar un método, el cual podría llamarse por ejemplo animate(), encargado de gestionar las actualizaciones de la vista de la aplicación. Javascript provee una función nativa que permite ejecutar asincrónicamente cada un lapso determinado de tiempo una función o una sección de código llamada setInterval(). Sin embargo, Three.js provee una nueva función llamada requestAnimationFrame() que funciona de manera más óptima.

Esta nueva función se desarrolló debido a que setInterval realiza el envío de peticiones para ejecutar loops cada una cantidad determinada de milisegundos sin contemplar si el sistema está ocupado o no, haciendo que este último almacene la petición en una cola de espera. SetInterval también emite peticiones aunque el sistema no esté en foco, es decir que aunque se minimice el navegador, la función continua enviando peticiones. Esta cantidad de peticiones en memoria realizaban un gasto totalmente innecesario de procesamiento y memoria.

La nueva función requestAnimationFrame espera recibir un sólo parámetro: La función que se quiere ejecutar, y a diferencia de setInterval, una vez finalizado el loop es necesario volver a llamar a la función requestAnimationFrame pasándole nuevamente la función a ejecutar.

Esta nueva función optimiza el uso de la CPU, actualizando de forma automática como la CPU le permite, a 60 cuadros por segundo en Pcs y notebooks (60 FPS), mejorando así la capacidad del manejo de información en animaciones y consumiendo menos recursos.

Para la utilización de la función resquestAnimationFrame, es necesario llamar a esta como primera línea de una función, enviando como parámetro la función que se necesita ejecutar como se muestra en las Figuras 26 y 27

```
function animate()
{
    requestAnimationFrame( animate );
    render();
}
```

Figura 26. Captura de código: Utilización requestAnimationFrame en Three.js.

```
function render()
{
    renderer.render( scene, camara );
}
```

Figura 27. Captura de código: Función renderer en Three.js.

RequestAnimationFrame contiene animaciones suaves, ya que utiliza una frecuencia de imagen consistente. La CPU en lugar de ser sobrecargada con tareas de representación, permite que el procesador sea capaz de manejar otras tareas al mismo tiempo que la animación enviada.

Otra de las ventajas de usar esta función es que funciona con una tasa de refresco de la pantalla, que define la rapidez con la pantalla puede mostrar una nueva imagen.

Por último, si la pestaña actual del navegador donde se produce la animación pierde foco, requestAnimationFrame detendrá la animación, lo cual produce importantes ahorros de energía y aumenta el rendimiento de la CPU. [18] [19] [20]

3.2.5 TweenJS

Desde la creación de Javascript, muchos usuarios utilizan esta librería para realizar distintos tipos de animaciones, algo complicado a la hora de programar. Este tipo de animaciones, para mover objetos o diferentes elementos, debía realizarse mediante interpolaciones de movimiento entre partes distintas de cada fotograma.

A partir de la aparición de la librería Tween.js, resulta más sencillo realizar todo tipo de animaciones, tales como mover un objeto, cambiar su tamaño, desvanecerlo, y muchas otras animaciones mejorando notablemente su rendimiento.

Tween.JS, es una librería, que actúa como motor de animaciones, cuyo fin es facilitar a los usuarios la creación de animaciones en la web. Su tarea es realizar la interpolación automática de las propiedades de los objetos, lo que en un principio debía realizarse manualmente.

Esta librería es sencilla de aprender y utilizar, pero a su vez es muy potente en cuanto a su rendimiento, permitiendo realizar animaciones complejas aprovechando las posibilidades que nos brinda JavaScript. Tween.js nos brinda muchas ventajas en cuanto a performance, precisión, velocidad de fotogramas, y como lo mencionamos anteriormente, una notable facilidad en su uso.

En la Figura 28 se muestra cómo utilizar esta librería. Lo primero que se debe realizar, es incluir la biblioteca en nuestro proyecto:

```
1  <script type='text/javascript' src='tween.js'></script>
```

Figura 28. Captura de código: Inclusión librería TweenJS

La manera de construir un Tween se realiza como se muestra en la Figura 29.

```
1  var tween1 = new Tween(objeto,propiedad,easing,inicio,fin,duración,sufijo);
```

Figura 29. Captura de código: Constructor de Tween para librería TweenJS

Dónde:

Objeto: Este argumento contiene el objeto al que se quiere aplicar la animación.
Propiedad: La propiedad que será modificada en el objeto.
Easing: El tipo de suavizado que utilizará la animación.
Inicio: El valor inicial de la propiedad.
Fin: El valor final de la propiedad.
Duración: El tiempo en segundos que durará la animación.
Sufijo: Agrega al final de la propiedad modificada, una cadena de caracteres.

Para especificar una función de easing deberemos especificar, cuál es su nombre. Se dispone de 6 métodos de suavizado:

Back: Extiende la animación cuando se supera el límite de la transición en uno o ambos extremos para producir el efecto de ser empujado hacia atrás desde el límite.
Bounce: Añade un efecto de rebote dentro del rango de la transición en uno o ambos extremos. El número de rebotes depende de la duración: cuanto mayor sea la duración, producirá un mayor número de rebotes.
Elastic: Añade un efecto elástico que está fuera del rango de la transición en uno o ambos extremos. La duración no afecta al grado de elasticidad.
Regular: Añade un movimiento más lento en uno o ambos extremos. Esta función permite añadir un efecto de aceleración, desaceleración o ambos.
Strong: Añade un movimiento más lento en uno o ambos extremos. Este efecto es similar al suavizado Regular, pero más pronunciado.
None: Añade un movimiento uniforme de principio a fin sin efectos, ralentización ni aceleración. Esta transición también se denomina transición lineal. [21]

Una vez instanciado Tween, el siguiente paso es realizar la acción correspondiente para este cambio tenga efecto y podamos verlo. Para realizar dicha acción, se utiliza el método start(), el cual se encargará de comenzar la animación.

Una vez terminada la animación, Tween.js permite a través del método onComplete() poder ejecutar inmediatamente otro código, una nueva animación, o directamente concluir con la animación que se realizó recientemente.

Esta librería permite, en el marco de la presente tesis, facilitar notablemente la creación de animaciones, ya que la aplicación IndoorNavigation3D, consta de numerosas animaciones para

poder realizar los movimientos de la cámara principal, permitiendo que ésta recorra toda la facultad de informática representada de manera suave, correcta y utilizando de manera eficiente la CPU.

3.2.6 JsQRCode

JsQRCode es un escáner y a su vez un creador de códigos QR, implementado en íntegramente en javascript, permitiendo su ejecución en el navegador del cliente sin necesidad de instalar algún plugin ya que funciona bajo los estándares de HTML5.

A diferencia de otros frameworks, este no solo permite subir y procesar un archivo con una imagen de un código QR, sino también leer los códigos QR desde la cámara web del dispositivo cliente.

La implementación de dicho framework está basada en la librería ZXing implementada en Java, diseñada para procesar imágenes de códigos de una y dos dimensiones.

El software está licenciado bajo la licencia Apache y su código fuente está disponible en GitHub[9]. Además en el sitio oficial (www.webqr.com) existe una demo del framework.

Para el desarrollo de la aplicación se utilizó una variante del framework llamada jsqrcode-scanner la cual es una adaptación de la librería anteriormente mencionada e implementa un scanner de códigos QR. Esta variante se puede descargar desde https://github.com/asbjornenge/jsqrcode-scanner y utilizarla bajo las mismas condiciones que jsqrcode.

3.2.7 jQuery Mobile

jQuery Mobile es un framework de interfaz de usuario de desarrollo de web que permite desarrollar aplicaciones web para celulares o móviles que trabajen en cualquier Smartphone y Tablet.

El jQuery Mobile framework está basado en jQuery y proporciona una serie de herramientas, incluyendo el manejo del DOM de HTML y XML, el control de evento, la comunicación con el servidor a través de Ajax, así como los efectos de animación y de imágenes para páginas web.

El framework jQuery Mobile se descarga de manera independiente desde su sitio oficial[10] y adicionalmente es necesario descargar la librería jQuery[11]. Tanto jQuery Mobile como jQuery son frameworks gratis, bajo una doble licencia (MIT y GPL).

Posee un amplio conjunto de componentes, es por eso que jQuery Mobile se ha convertido en un framework muy utilizado para el desarrollo de aplicaciones web para móviles.

Las características principales de jQuery Mobile son:

Simplicidad: El framework es simple de usar. Permite desarrollar páginas utilizando principalmente el marcado con un escaso código JavaScript.

Mejora progresiva: Mientras jQuery Mobile aprovecha las últimas tecnologías de HTML5, CSS3 y JavaScript, no todos los dispositivos móviles brindan este soporte. La filosofía de jQuery Mobile es soportar tanto la gama alta de los dispositivos, como aquellos que no

[9] https://github.com/LazarSoft/jsqrcode
[10] http://jquerymobile.com/download
[11] http://jquery.com/download

soportan grandes funcionalidades. Se trata de proveer la mejor experiencia posible.

Accesibilidad: jQuery Mobile ha sido diseñado pensando en la accesibilidad. Brinda soporte a las aplicaciones de internet enriquecidas y accesibles "Accessible Rich Internet Applications" para ayudar a hacer páginas web más accesibles a los visitantes con discapacidad que usen tecnologías de asistencia.

Tamaño pequeño: El tamaño total de jQuery Mobile framework es relativamente pequeño. Unos 12 KB de la biblioteca JavaScript, 6kb de CSS y algunos íconos.

Tematización: El framework también proporciona un sistema de temas que ofrece una aplicación a nuestro propio estilo.

Como se mencionó, jQuery Mobile es compatible tanto con la gama alta como con la baja de dispositivos, incluyendo los que no soportan JavaScript y actualmente provee. En el sitio oficial (http://jquerymobile.com/gbs/1.4) se detalla a que plataformas móviles brinda soporte. [22]

3.2.8 Symfony

Symfony es un framework Open Source para desarrollar aplicaciones web en PHP. Originalmente fue concebido por la agencia interactiva SensioLabs para el desarrollo de sitios web para sus propios clientes. Symfony fue publicado por la agencia en el año 2005, bajo la licencia MIT, de código abierto y hoy se encuentra entre los frameworks líderes de PHP.

Apoyado por SensioLabs y también por una gran comunidad, Symfony tiene una amplia gama de recursos que facilitan su aprendizaje: detallada documentación, soporte de la comunidad (listas de e-mail, IRC, etc.) y soporte profesional (consultoría, entrenamientos, etc.).

Cuenta, entre los miles de sitios y aplicaciones desarrolladas con su plataforma, prestigiosas referencias, tales como Yahoo!, Dailymotion, Opensky.com, Exercise.com, phpBB, o Drupal. [23]

Symfony es un framework que optimiza el desarrollo de las aplicaciones web. Está basado en el patrón MVC (Modelo Vista Controlador), es decir, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja.

Además, automatiza las tareas más comunes, por ejemplo la creación de módulos CRUD de una clase, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. La premisa del framework es el de no reinventar la rueda cada vez que se crea una nueva aplicación web.

Symfony está desarrollado completamente en PHP 5.3 y es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows.

Fue diseñado para ajustarse a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares).
- Independiente del sistema gestor de bases de datos. Su capa de abstracción y el uso de Doctrine o Propel como ORM, permiten cambiar con facilidad de SGBD en

cualquier fase del proyecto.

- Utiliza programación orientada a objetos.
- Sencillo de usar en la mayoría de casos, aunque es preferible para el desarrollo de grandes aplicaciones Web que para pequeños proyectos.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador sólo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con las bibliotecas de otros fabricantes.
- Una potente línea de comandos que facilitan generación de código, lo cual contribuye a ahorrar tiempo de trabajo.
- Las características más comunes para el desarrollo de proyectos web están automatizadas en symfony, tales como:
- Permite la internacionalización para la traducción del texto de la interfaz, los datos y el contenido de localización.
- La presentación usa templates y layouts que pueden ser construidos por diseñadores de HTML que no posean conocimientos del framework.
- Los formularios soportan la validación automática, lo cual asegura mejor calidad de los datos en las base de datos y una mejor experiencia para el usuario.
- El manejo de caché reduce el uso de banda ancha y la carga del servidor.
- La facilidad de soportar autenticación y credenciales facilita la creación de áreas restringidas y manejo de seguridad de los usuarios.
- El enrutamiento y las URLs inteligentes hacen amigable las direcciones de las páginas de la aplicación.
- Las listas son más amigables, ya que permite la paginación, clasificación y filtrado automáticos.
- Los plugins proveen un alto nivel de extensibilidad.
- La interacción con AJAX es mucho más sencilla.

Symfony se distribuye bajo licencia Open Source MIT, que no impone restricciones y permite el desarrollo de código abierto, así como aplicaciones propietarias. Puede descargarse desde su sitio oficial[12] de manera gratuita. [24]

Si bien Symfony se encuentra en la versión 2.4 que posee varias mejoras y cambios estructurales con respecto a las versiones 1.x, para el actual proyecto se utilizó la versión 1.4 dado que el sistema original estaba implementado en esta versión.

[12] http://symfony.com/download

4. Modelo de Navegación 3D

4.1 Modelo especificado para Navegación 3D

El modelo de clases se presenta en la Figura 30, se basó en el de la aplicación original, modificando únicamente aspectos relacionados con la navegación en tres dimensiones.

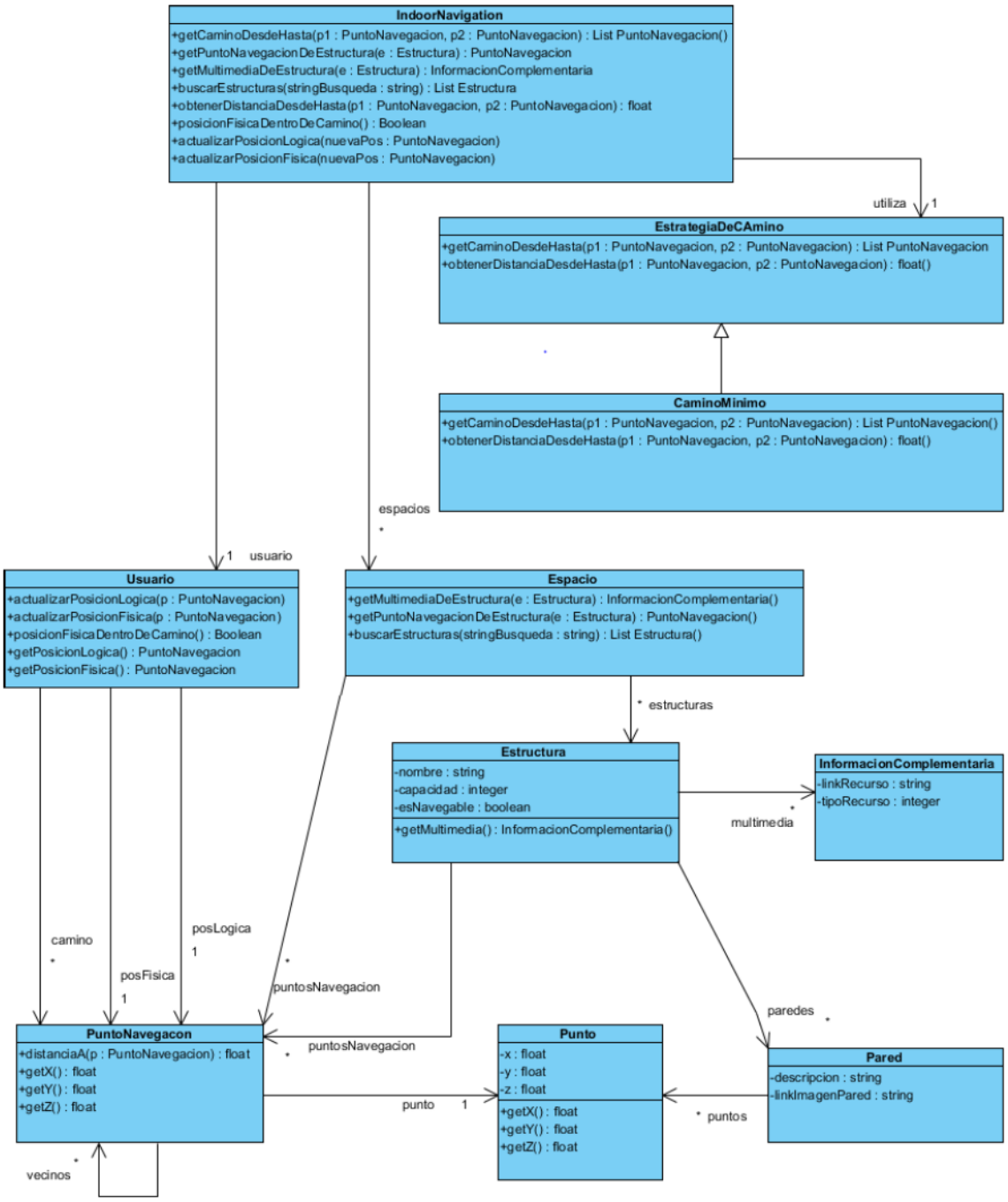


Figura 30. Modelo de Clases IndoorNavigation3D.

A continuación se detallan una por una las clases definidas en el Modelo de Clases.

IndoorNavigation: Esta clase representa a la aplicación en sí y es la encargada de responder a las peticiones realizadas por los usuarios.

Conoce a un Espacio, sus estructuras y todos los puntos navegables dentro de él. Tiene conoce una Estrategia de Camino, la cual determinará el algoritmo a utilizar para el cálculo de caminos dentro del espacio.

Usuario: Representa al usuario del sistema, el cual mediante peticiones al sistema podrá determinar su ubicación actual física y lógica. También conoce el camino para realizar, compuesto por diferentes puntos de navegación.

Espacio: Es la abstracción del edificio navegable. Conoce los puntos de navegación y estructuras que lo componen.

Estructura: Representa a cada aula o habitación dentro de un edificio. Conoce las paredes que lo forman, tendrá un Punto por cada vértice de la estructura que deseemos representar, permitiendo así representar cualquier forma.

Una estructura puede ser navegable o no, si es navegable tiene un punto de navegación asociada a la misma para que pueda ser seleccionada como destino u origen de la navegación. Una estructura también tiene datos asociados a la misma, como fotos, videos y texto, que ayudarán al usuario a crear una relación entre la estructura dibujada en un plano y la estructura real.

Pared: Representa a las paredes que conforman una estructura, a la que se le puede asociar una descripción y una imagen.

Punto: están formados por las coordenadas x,y,z que ubican al punto en un plano. Esto será utilizado para poder representar un vértice de estructuras, o la posición de un punto de navegación.

PuntoNavegacion: Representan puntos por donde un usuario puede moverse y ser posicionado. Algunos puntos de navegación están asociados a estructuras convirtiéndolos en posibles destinos. Los puntos de navegación se relacionan entre sí, guardando como dato de esa relación la distancia a ese punto de navegación, formando así una red o grafo, esto es lo que determina los caminos dentro del edificio y hace posible el cálculo del camino de un punto de navegación de origen a uno de destino.

InformaciónComplementaria: Representan información visual asociada a una estructura, de esta manera el usuario no solo verá indicaciones de ruteo al destino sino que podrá ver información relacionada al lugar donde se dirige o está parado.

EstrategiaDeCamino: Es una clase abstracta que representa los conceptos generales de de algoritmos de cálculos de caminos. A partir de la misma se puede subclasificar para tener varias estrategias concretas de cálculo de camino instanciadas y cambiar de estrategia en

tiempo de ejecución. De esta forma podemos tener diferentes criterios para determinar un camino. Esto cumple con el patrón de diseño Strategy[13].

CaminoMinimo: Estrategia de cálculo de caminos concreta, en este caso esta estrategia determinará el camino de menor distancia de un punto de navegación a otro. Esta es la estrategia que implementaremos en la aplicación.

4.2 Interacción y Diagramas de Secuencia

A continuación se presentan los diagramas de secuencias que se disparan a partir de invocar los métodos presentes en el usuario.

+getCaminoDesdeHasta(origen : PuntoNavegacion, destino : PuntoNavegacion) : List PuntoNavegacion()

En el diagrama de secuencia de la Figura 31 podemos ver como el usuario interactúa con el sistema para conocer el camino mínimo entre puntos de navegación enviados como parámetro. El usuario inicia la interacción pidiendo al sistema el cálculo de caminos de un origen a un destino. La búsqueda del camino mínimo se delega a la estrategia de cálculos de caminos mínimos para que esta determine el camino, el cual será representado en un plano en la interfaz del sistema.

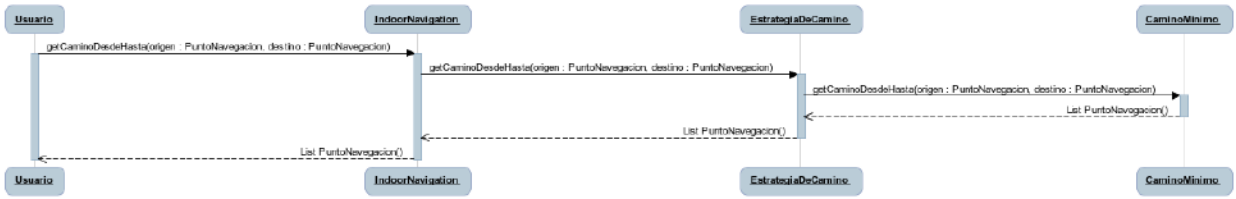


Figura 31. Diagrama de secuencia "getCaminoDesdeHasta()"

+getPuntoNavegacionDeEstructura(estructura :.Estructura) : PuntoNavegacion

El sistema, a partir de una estructura dada, recupera el punto de navegación asociado a esta estructura para su selección o no como posible destino. Como se puede apreciar en la Figura 32.

[13] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: elements of reusable object-oriented software. Pearson Education.

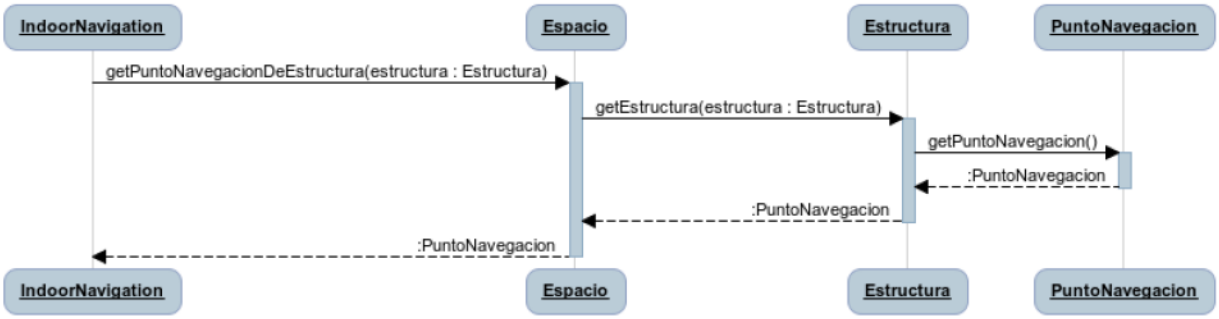


Figura 32. Diagrama de secuencia "getPuntoNavegaciónDeEstructura()"

+getMultimediaDeEstructura(estructura :.Estructura) : InformacionComplementaria

El usuario selecciona una estructura y elige la opción "ver detalles de la estructura" (ver Figura 33). IndoorNavigation recibe la estructura y a partir de esta, recupera los recursos multimedia que tiene asociados para luego ser representados en la interfaz.

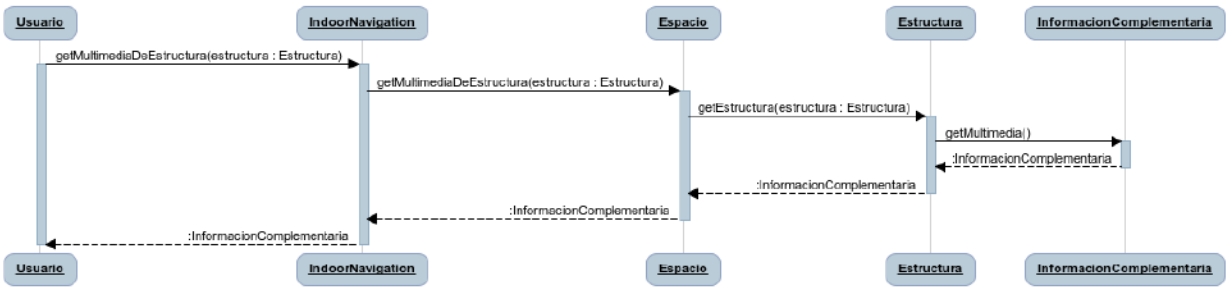


Figura 33. Diagrama de secuencia "getMultimediaDeEstructura()"

+obtenerDistanciaDesdeHasta(origen : PuntoNavegacion, destino : PuntoNavegacion) : float

A partir de dos puntos de navegación ya preestablecidos, el sistema calcula la distancia que existe entre estos dos puntos, para luego utilizar la misma en el cálculo del camino mínimo entre dos puntos. Como se muestra en la Figura 34.

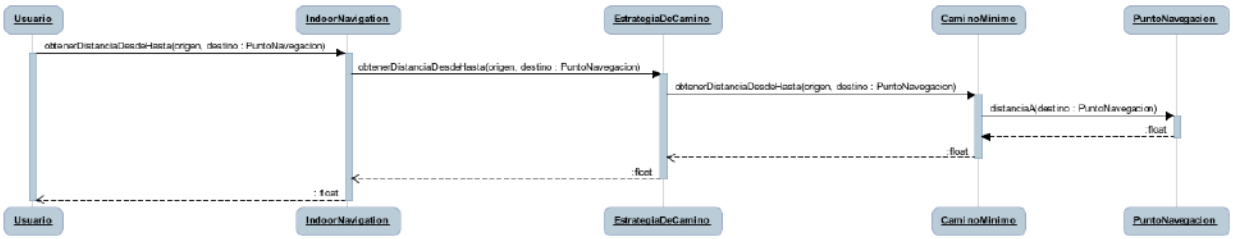


Figura 34. Diagrama de secuencia "obtenerDistanciaDesdeHasta()"

+posicionFisicaDentroDeCamino() : Boolean

El sistema a partir de la posición actual física de un usuario determinado, devuelve verdadero o falso dependiendo si esta posición física actual se encuentra dentro del camino mínimo actual. Esto se puede apreciar en la Figura 35.

Figura 35. Diagrama de secuencia "posicionFisicaDentroDeCamino()"

4.3 Ejemplo de Instanciación del Modelo creado

La instanciación del edificio navegable es uno de los puntos más importantes de la aplicación, ya que es donde se definen las estructuras que actuarán como posibles destinos del usuario y el grafo que formarán los puntos de navegación. La instanciación es un paso complejo ya que se crean partes del edificio navegable individualmente para luego ser asociadas, por este motivo para ver el edificio resultante se necesita tener todas las estructuras con sus puntos creados y asociadas a sus respectivos puntos de navegación.

El modelo permite representar los datos para luego poder tener una vista general en tres dimensiones y una en dos dimensiones. Para poder ejemplificar la instanciación, se tomarán dos planos de un edificio simplificado, uno en dos dimensiones (Figura 36) y otro en tres dimensiones (Figura 37), los cuales cuentan con tres estructuras navegables elegibles como posibles destinos.

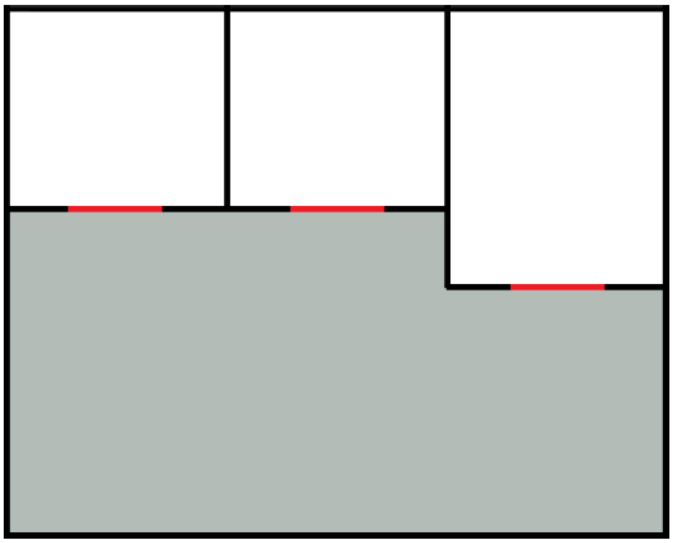


Figura 36. Ejemplo de edificio simplificado en dos dimensiones.

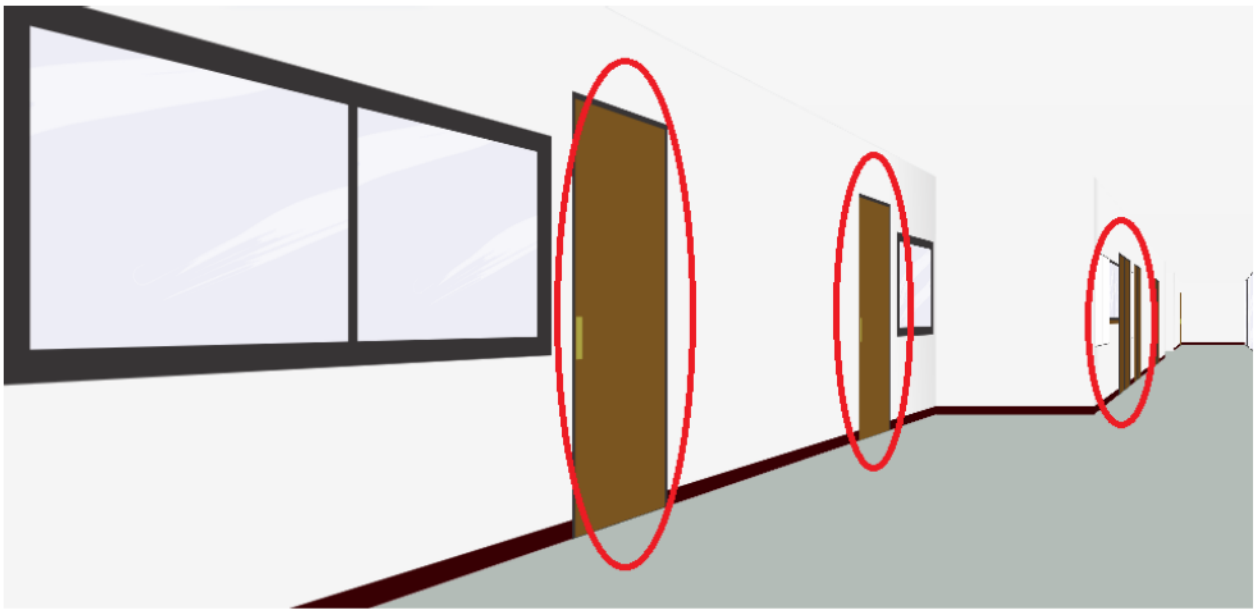


Figura 37. Ejemplo edificio simplificado en tres dimensiones.

Para poder representar este plano con el modelo de clases creado es necesario definir más elementos que permitan instanciar el edificio:

Puntos: los puntos representan los vértices de cada estructura (sea navegable o no) y la posición de los puntos de navegación. A cada uno de los puntos definidos en el plano le fue asignado una coordenada (x,y,z) que la aplicación luego usará para determinar su posición en el plano.

Puntos de navegación: los puntos de navegación determinan los caminos posibles dentro del edificio. Estos también tienen un punto asociado para situarlos en el plano. Cada punto de navegación está asociado con los puntos de navegación cercanos y esta relación guarda como información la distancia entre estos, creando así un grafo que luego la aplicación utilizará para determinar el camino. Se pueden ver que las tres estructuras tienen un punto de navegación asociado, esto las convierte en navegables, y de esta manera, elegibles como destinos al usuario de la aplicación.

Estructuras: Se pueden ver las 3 estructuras navegables en el plano, cada una con su nombre que permitirá al usuario hacer la selección del destino. Se pueden ver los Puntos que determinarán su forma y dimensiones y el punto de navegación asociado a estas.

A continuación en la Figura 38 se muestra el plano 2D resultante luego de definir estos datos mencionados. En la Figura 39 muestra el plano en 3D.

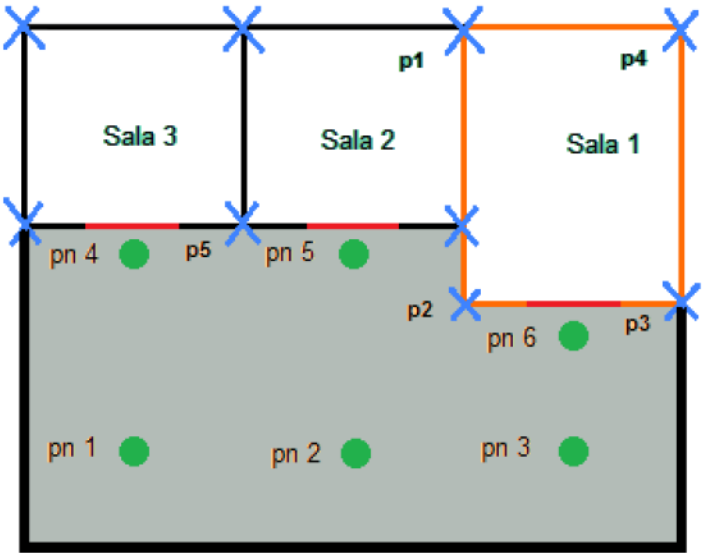


Figura 38. Edificio simplificado con información agregada en dos dimensiones.

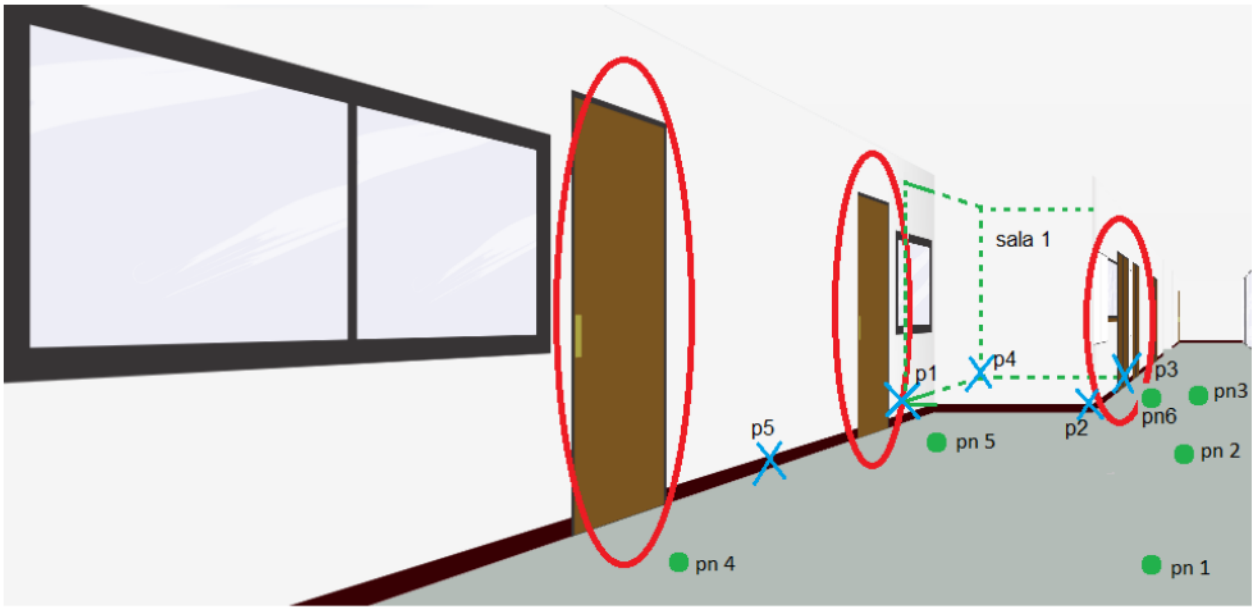


Figura 39. Edificio simplificado con información agregada en tres dimensiones

En las Figuras 38 y 39, se puede ver la información más relevante que se necesita para instanciarlo. Por ejemplo se puede ver como sala.1 está formado por los puntos p1,p2, p3, p4 , y que tiene asociado el punto de navegación pn.6. Como se puede observar, la sala 1, está formada por 4 puntos situados en la base de las paredes, por simplicidad todas las paredes de la aplicación tienen la misma altura

En el diagrama de instancias de la Figura 40, se puede ver cómo se representa el plano con el modelo de clases y cada una de las instancias de las clases Estructura, PuntoDeNavegacion

y Punto, y sus relaciones.

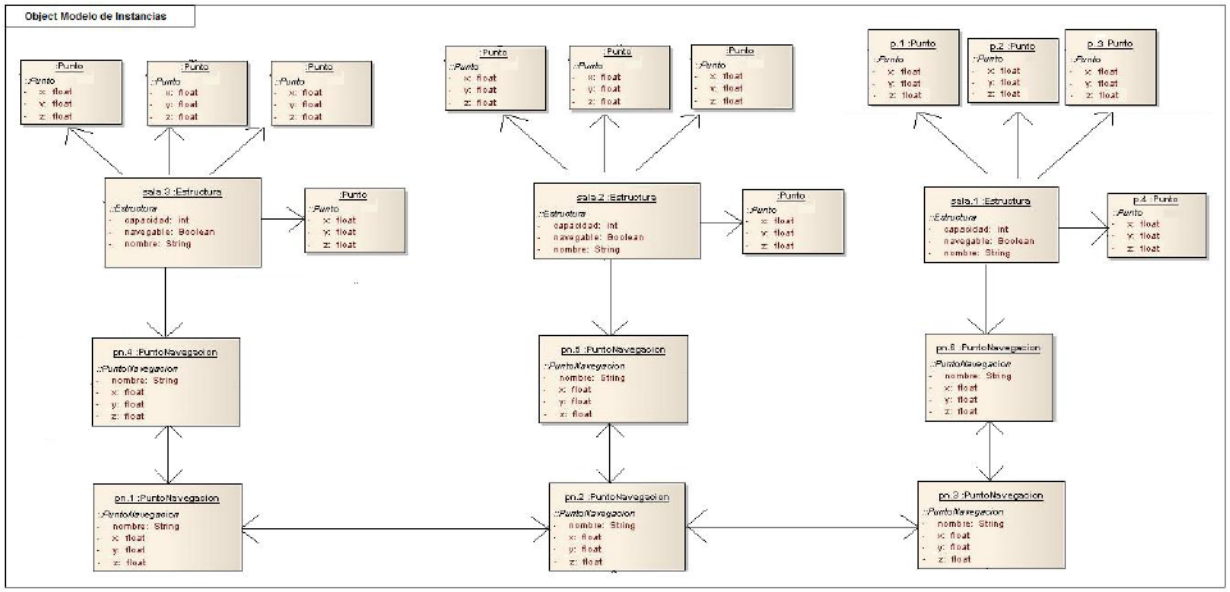


Figura 40. Diagrama de instanciación del edificio simplificado (considerando dos y tres dimensiones).

El cálculo de camino utiliza el grafo que se forma de las relaciones entre los puntos de navegación, cada relación con otro punto de navegación contiene la distancia (hasta este), esta distancia es utilizada por el algoritmo de cálculo de caminos (clase *CaminoMinimo*) para determinar el camino más corto entre dos puntos. En la Figura 41 se puede ver el grafo de navegación resultante de la instanciación de puntos de navegación definidos en el edificio de ejemplo.

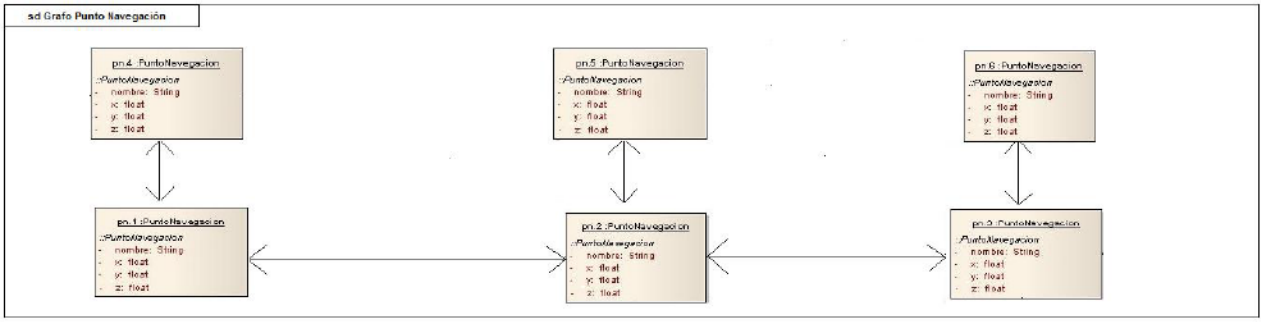


Figura 41. Grafo resultante de la instanciación del edificio simplificado (igual para dos y tres dimensiones).

Ahora se mostrará (Figuras 42 y 43) el grafo marcado con azul en el plano del edificio, las instancias entre puntos de navegación determinan los costos de pasar de un nodo a otro. Es importante aclarar que los costos se calculan en base a la distancia entre cada punto de navegación.

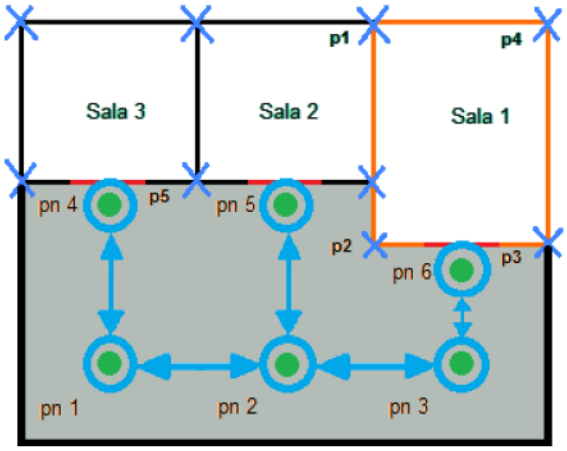


Figura 42. Grafo de puntos de navegación en dos dimensiones.

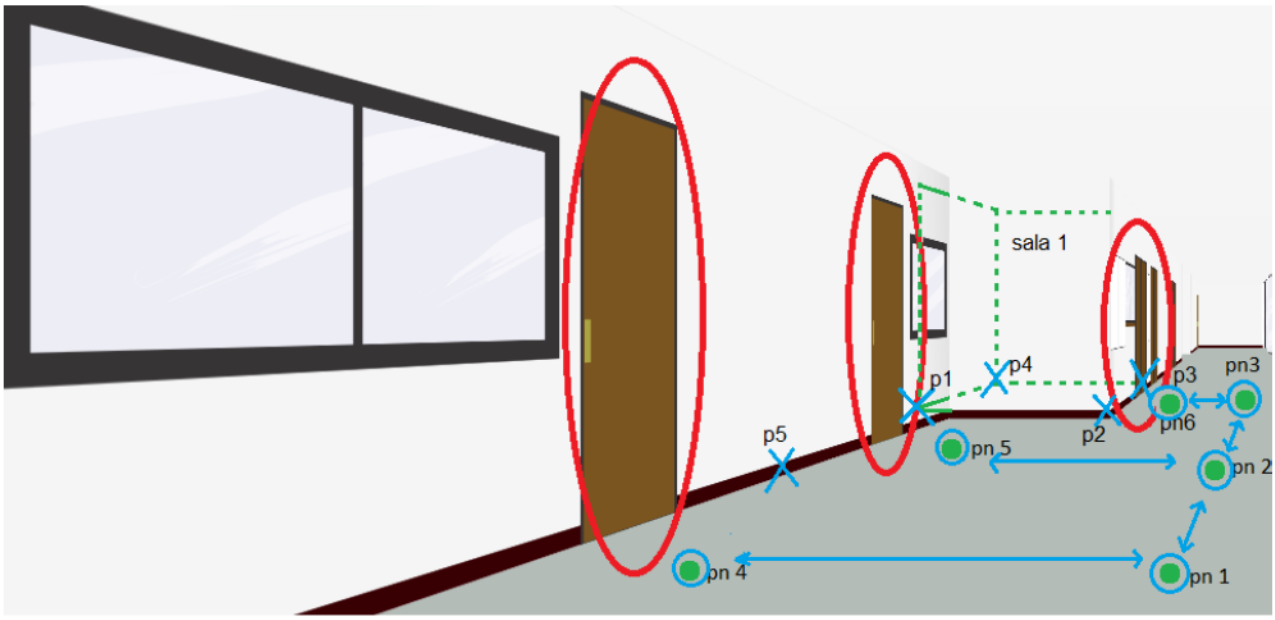


Figura 43. Grafo de puntos de navegación en tres dimensiones.

5. Casos de Uso

A continuación se detalla la funcionalidad que provee el prototipo desarrollado. En el *Anexo A* se listan los casos que se reutilizaron de la aplicación base para navegación 2D.

5.1 Usuarios Comunes

Nombre: Avanzar en el plano 3D.
Descripción: El usuario avanza lógicamente hacia su destino.
Actores: Usuario, Sistema.
Precondición: El usuario debe tener seleccionado el punto de origen y el punto destino dentro de la aplicación.
Curso de Eventos: 1. El usuario selecciona desde la interfaz de la aplicación la opción Avanzar. 2. El sistema procesa la información enviada por el usuario realizando el cálculo del siguiente punto del camino y realiza una animación hacia el mismo. Bloquea la barra de navegación. 3. El usuario visualiza en su interfaz una animación hacia su nueva posición. 4. El sistema desbloquea la barra de navegación. Si el siguiente punto calculado por el sistema es el último del camino, el sistema informa al usuario de la acción desplegando un menú con opciones para: Navegar a otro sitio, Ver información o Cerrar ventana.
Post condición: El usuario avanza en su posición lógica al siguiente punto del camino.

Nombre: Retroceder en el plano 2D y 3D
Descripción: El usuario retrocede lógicamente en el camino.
Actores: Usuario, Sistema.
Precondición: El usuario debe tener seleccionado el punto de origen y el punto destino dentro de la aplicación.
Curso de Eventos: 1. El usuario selecciona desde la interfaz de la aplicación la opción Retroceder. 2. El sistema procesa la información enviada por el usuario realizando el cálculo del punto anterior del camino y realiza una animación hacia el mismo. Bloquea la barra de navegación. 3. El usuario visualiza en su interfaz una animación hacia su nueva posición.

4. El Sistema desbloquea la barra de navegación.
Post condición: El usuario retrocede al punto anterior del camino.

Nombre: Girar 360º izquierda/derecha en el plano 3D.
Descripción: El usuario selecciona la opción Girar 360º izquierda/derecha en la aplicación.
Actores: Usuario, Sistema.
Precondición: El usuario debe tener seleccionado el punto de origen y el punto destino dentro de la aplicación.
Curso de Eventos: 1. El usuario selecciona desde la interfaz de la aplicación la opción Girar 360º izquierda/Derecha. 2. El sistema procesa la información enviada por el usuario y realiza una animación de un giro de 360 grados. Bloquea la barra de navegación. 3. El usuario visualiza en su interfaz una vista panorámica con respecto a su posición actual. 4. Desbloquea la barra de navegación.
Post condición: El usuario queda parado en la misma posición luego de visualizar una vista panorámica.

Nombre: Buscar Origen y Destino.
Descripción: El usuario selecciona la opción Buscar y lee un código QR para determinar un punto de origen y luego selecciona un destino.
Actores: Usuario, Sistema.
Precondición: -
Curso de Eventos: 1. El usuario lee un código QR para determinar el origen. 2. El sistema guarda en una variable la opción seleccionada por el usuario y despliega un listado con las estructuras navegables disponibles, omitiendo en dicho listado la estructura seleccionada como origen. 3. El usuario selecciona desde la interfaz de la aplicación una de las opciones disponibles como punto de destino. 4. El sistema procesa la información enviada por el usuario realizando el cálculo del camino mínimo entre el punto de origen actual y el destino.

5. El usuario visualiza en su interfaz un plano con un camino desde el punto de origen hasta el destino seleccionado.
Post condición: El usuario visualiza un plano en 3D y 2D con el camino a navegar.

Nombre: Activar QR.
Descripción: El usuario activa el lector QR para indicar su posición física actual.
Actores: Usuario, Sistema.
Precondición: El usuario debe tener seleccionado el punto de origen y el punto destino dentro de la aplicación. El lector QR debe estar desactivado.
Curso de Eventos: 1. El usuario selecciona el botón "Activar QR" desde la aplicación. 2. El sistema procesa la información recibida y realiza la activación del lector de códigos QR de la aplicación.
Post condición: El sistema activa el lector de códigos QR ubicado en la interfaz de la aplicación.

Nombre: Desactivar QR.
Descripción: El usuario desactiva el lector QR de la aplicación.
Actores: Usuario, Sistema.
Precondición: El usuario debe tener seleccionado el punto de origen y el punto destino dentro de la aplicación. El usuario debe tener activado el lector de códigos QR.
Curso de Eventos: 1. El usuario selecciona el botón "Desactivar QR" desde la aplicación. 2. El sistema procesa la información y deja leer códigos QR desde el lector.
Post condición: El sistema desactiva el lector de códigos QR de la aplicación.

Nombre: Leer código QR desde navegación.
Descripción: El usuario realiza la lectura de un código QR desde el lector QR de la aplicación.
Actores: Usuario, Sistema.
Precondición: El usuario debe tener activado el lector QR de la aplicación.
Curso de Eventos: 1. El usuario lee desde la cámara de su dispositivo un código QR que indica su posición

física actual.

2. El sistema procesa la información, detecta la posición actual física del usuario y dibuja un icono representativo del mismo en el mapa 2d de la interfaz.

Post condición: El sistema actualiza la posición física del usuario en la interfaz.

Nombre: Recalcular camino.

Descripción: El usuario selecciona la opción para que el sistema re-calcule el camino desde su nueva posición física actual.

Actores: Usuario, Sistema.

Precondición: El usuario debe haber leído un código QR que indique su posición física actual fuera del camino establecido. El botón “recalcular camino” debe encontrarse activado.

Curso de Eventos:

1. El usuario selecciona la opción “Recalcular camino” desde la interfaz del sistema.

2. El sistema procesa la información, detecta la posición física actual, y despliega en la interfaz la lista de destinos disponibles para navegar.

Post condición: El sistema muestra en la interfaz los posibles los recorridos disponibles.

Nombre: Navegar a otro sitio desde destino.

Descripción: El usuario selecciona un nuevo destino desde el destino alcanzado.

Actores: Usuario, Sistema.

Precondición: El usuario llegó físicamente al destino.

Curso de Eventos:

1. El usuario selecciona navegar a otro sitio desde el destino.

2. El sistema procesa la información enviada por el usuario y guarda el nuevo origen en una variable.

3. El sistema despliega en su interfaz un listado con los destinos navegables posibles.

Post condición: El usuario obtiene un listado de los destinos navegables posibles.

5.2 Usuarios Registrados

Nombre: Crear Pared Dibujable.
Descripción: El Usuario registrado da de alta una Pared Dibujable.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, crear una Pared Dibujable. 4. El usuario registrado selecciona del menú la función, crear una pared Dibujable. 5. El usuario registrado ingresa los atributos: Punto1, Punto2, Link imagen, Descripción y Orientación pared.
Post condición: Se crea una nueva Pared Dibujable en el sistema.

Nombre: Editar Pared Dibujable.
Descripción: El Usuario registrado selecciona una Pared Dibujable para ser editada.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, editar una Pared Dibujable. 4. El usuario registrado busca en el listado de Paredes Dibujables la que desea modificar y selecciona la opción de editar. 5. El usuario registrado modifica los atributos necesarios. 6. El sistema valida la información enviada por el usuario y crea la nueva Pared Dibujable.
Post condición: Se modifican los atributos de una Pared Dibujable en el sistema.

Nombre: Borrar Pared Dibujable.
Descripción: El Usuario registrado selecciona una Pared Dibujable para ser eliminada.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, borrar una Pared Dibujable. 4. El usuario registrado busca en el listado de Paredes Dibujables la que desea eliminar y selecciona la opción de borrar. 5. El sistema pregunta al usuario si realmente desea realizar el borrado físico de la pared dibujable. 6. El usuario registrado confirma que desea eliminar la Pared Dibujable. 7. La Pared Dibujable es eliminada del sistema.
Post condición: Se elimina una Pared Dibujable en el sistema.

6. Implementación del Prototipo

6.1 Arquitectura General

Para el desarrollo en tres dimensiones se utilizó WebGL, abstraída por la librería Three.js, la cual como se mencionó permite trabajar con la unidad de procesamiento gráfico (GPU) del cliente por medio de código javascript. La intención siempre fue la de mostrar una animación de la navegación virtual en lugar de recargar la página, con lo cual, fue necesario no solo agregar la librería Three.js sino también parte de la arquitectura de la aplicación.

Al tratarse de una tecnología que requiere de un dispositivo móvil con buena capacidad de procesamiento y video, se optó por modificar la arquitectura original de la aplicación de navegación 2D, delegando al cliente los cálculos en los que no era viable refrescar la pantalla, por ejemplo la actualización de la posición virtual que requiere de una animación. Como se puede apreciar en la Figura 44.

El motivo de la elección de realizar los cálculos en el cliente en lugar de utilizar AJAX, es decir, realizar los cálculos en el servidor y pasarlos luego al cliente sin recargar la página, fue puramente de rendimiento. Resultó más rápido realizar los cálculos de una petición en el cliente, teniendo en cuenta que el dispositivo requerido para que la aplicación funcione cuenta con una gran capacidad de cálculo, que realizar una petición vía AJAX al servidor para que éste haga los cálculos y le retorne los resultados al cliente.

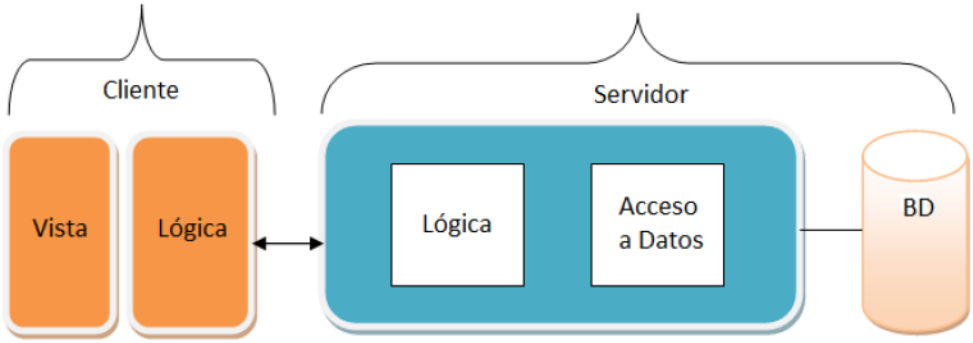


Figura 44. Arquitectura de la aplicación IndoorNavigation3D.

Otro aspecto muy importante a tener en cuenta para realizar estas modificaciones fue la lectura de códigos QR para actualizar la posición física. Fue indispensable integrar un lector de códigos QR dentro de la aplicación que se ejecutará en el dispositivo cliente, caso contrario, no se podría actualizar la posición física sin salir de la aplicación. Se hablará de esto en presente capítulo.

De esta manera quedó planteada la arquitectura de la aplicación. A continuación se mencionan las ventajas y desventajas del cambio de arquitectura.

Ventajas

Utilizando esta arquitectura no se recarga la página en cada cambio de posición física (lectura de código QR) o lógica (animación) y los cálculos se realizan en el cliente, con lo cual la aplicación responde de manera inmediata al ejecutarse otorgando al usuario un tiempo de respuesta deseable.

Desventajas

Se complejiza un poco el código, ya que ahora la lógica no está únicamente en el servidor, sino que también el cliente. Es necesario disponer de dispositivos móviles con gran capacidad de procesamiento y video.

6.2 Vista 3D

Sin dudas uno de los desafíos más grandes dentro de los cambios propuestos fue la vista de la aplicación. Como se mencionó en la sección de desarrollo propuesto, se propuso implementar una vista basada en Street View. A continuación se hablará un poco de los sistemas de Street View para entender el funcionamiento de los mismos.

Para implementar la facultad de informática en formato Street View se utilizó la librería javascript Three.js que abstrae la implementación del estándar definido por WebGL. El primer inconveniente encontrado fueron las fotografías, ya que no se contaba con equipamiento adecuado para tomar este tipo de imágenes. Con lo cual, se optó por generar objetos en tres dimensiones simulando paredes y asignándole a cada uno una imagen que identificara la pared virtual con la real. Estas imágenes fueron procesadas a partir de fotografías tomadas con una cámara digital por un diseñador gráfico quien se encargó de transformarlas en imágenes simplificadas, facilitando el proceso de desarrollo y por otra parte reduciendo el tamaño de las mismas. Gracias a esto, las fotos pasaron de pesar no menos de 4MB cada una, a pesar no más de 50 KB cada una, mejorando de manera drástica el rendimiento de la aplicación. En la Figura 45 se muestra las paredes simplificadas en el plano 3D.

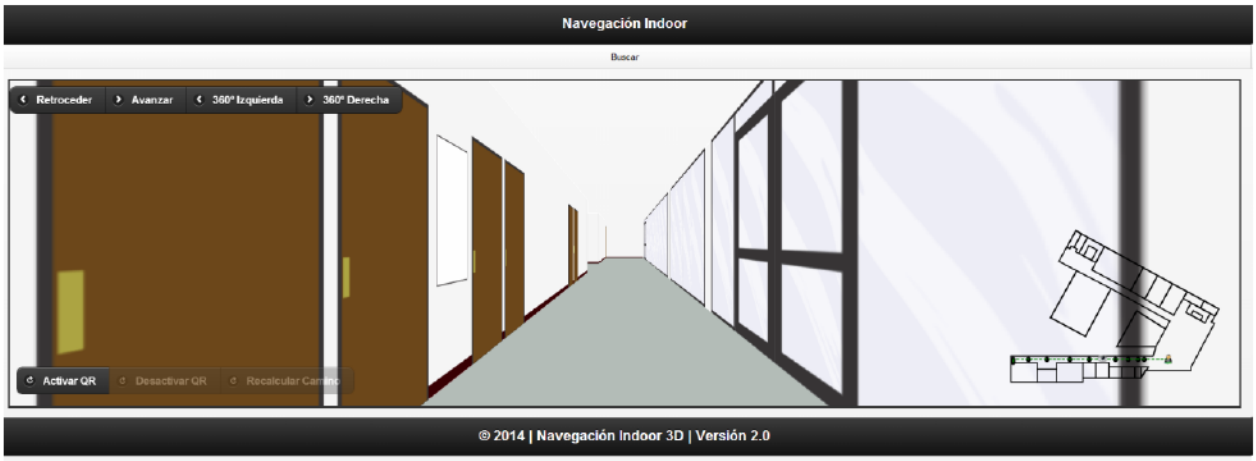


Figura 45. Imágenes de paredes simplificadas en plano 3D.

Para generar las paredes se utilizó como base los datos disponibles de las estructuras implementadas en dos dimensiones, lo cual agilizó muchísimo el desarrollo de la vista tridimensional de la Facultad ya que no fue necesario pasar un plano del edificio a un sistema de coordenadas. Si bien no se disponía de la altura de las paredes, ya que únicamente existían puntos que representaban los vértices del perímetro de la base de las diferentes estructuras (una estructura posee al menos tres puntos), se mantuvo una altura constante para las paredes para simplificar la construcción del escenario.

Por otra parte, para mejorar el rendimiento de la aplicación se optó por no construir todas las paredes de la facultad ya que muchas de ellas se superponían y muchas otras no iban a ser visualizadas por el usuario navegando el interior de la facultad. De esta manera, muchos puntos quedaron excluidos para la construcción en tres dimensiones (no así en la de dos dimensiones). Adicionalmente se creó una nueva tabla en la base de datos llamada "pared_dibujable", que contiene los siguientes datos:

- **punto_1_id:** representa a uno de los extremos de la pared a dibujar.
- **punto_2_id:** representa al otro extremo de la pared a dibujar.
- **link_imagen:** representa al link de la imagen a cargar como textura.
- **descripción:** describe la pared para orientar mejor al desarrollador durante la implementación
- **orientacion_pared_id:** define la orientación de la pared (divididas en "Norte/Oeste" y "Sur/Este") para cargar la imagen de la pared únicamente del lado que corresponda, permitiendo optimizar el tiempo de carga.

Para poder dibujar efectivamente las paredes con Three.js, fue necesario realizar diferentes cálculos detallados a continuación.

Cálculo de punto medio de la pared

Three.js necesita conocer el punto medio de la pared, ya que desde ese punto se construye la misma. Como se dispone de dos puntos que representan a la pared, es fácil calcularlo.

El Punto medio está determinado por:

$$M = (x1 + x2) / 2 ; (y1 + y2) / 2$$

Ejemplo: A(7,5) y B (4,1)

$$M = (7 + 4) / 2 ; (5 + 1) / 2$$
$$M = 11/2 ; 6/2$$
$$M = (5,5 ; 3)$$

Cálculo de ángulo de rotación de la pared.

Adicionalmente, Three.js necesita conocer el ángulo de rotación que posee la pared. Como tenemos dos puntos que la representan, es posible obtener el ángulo realizando los siguientes

cálculos.

Suponiendo que se tienen los puntos A=(x1,y1) y B=(x2,y2). Al unirlos mediante una recta, esta recta va a estar inclinada o paralela con respecto al eje x. Si la recta está inclinada va a tener una pendiente denominada m, si la recta es vertical, no tiene pendiente y si la recta es horizontal entonces tiene pendiente(m=0).

La pendiente se calcula de la siguiente manera:

$$m = (y1-y2) / (x1-x2)$$

A partir de la pendiente se puede calcular el ángulo, ya que la pendiente es igual a la tangente del ángulo que nos interesa saber. Despejando se puede obtener fácilmente el ángulo.

$m = TAN(\alpha)$
$\alpha = ATAN(m)$

donde ATAN=arcotangente
α es el ángulo que forman esos puntos

Ejemplo: A(7,5) y B (4,1)

$m = (5-1) / (7-4) = 4/3 = 1,33$
$1,33 = TAN (\alpha)$
$\alpha = ATAN(1,33)$
$\alpha \approx 0.926$ radianes $\approx 53.06^{o}$

En el caso de que (x1 - x2) = 0, es decir, no exista pendiente, utiliza 90º como ángulo de rotación.

Cálculo de distancia de la pared

Por último, Three.js necesita conocer la distancia de la pared. Como se dispone de un par de puntos por pared ubicados en distintos lugares del sistema de coordenadas, la distancia queda determinada por la relación:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Para demostrar esta relación se deben ubicar los puntos A(x1,y1) y B(x2,y2) en A(x1,y1) y B(x2,y2) en el sistema de coordenadas, luego formar un triángulo rectángulo de hipotenusa AB y emplear el teorema de Pitágoras.

Ejemplo: Calcular la distancia entre los puntos A(7,5) y B (4,1)

$$d = \sqrt{(4-7)^2 + (1-5)^2}$$

$$d = \sqrt{(-3)^2 + (-4)^2}$$

$$d = \sqrt{9+16}$$

$$d = \sqrt{25}$$

$$d = 5 \ unidades$$

La Figura 46 muestra el código con la función encargada de construir una pared en base a lo explicado anteriormente.

```
function dibujarPared(distancia,puntoMedioX,puntoMedioY,anguloRotacion,orientacion,link) {

    var border = 0;
    var elevation = 150;

    // se crea la pared
    var materialArray = [];
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));

    if (orientacion == 'Norte/Oeste'){
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('../../../../images/imagenes-navegador/'+link) }));
    }else{
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('../../../../images/imagenes-navegador/'+link) }));
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    }
    var MovingCubeMat = new THREE.MeshFaceMaterial(materialArray);
    var MovingCubeGeom = new THREE.CubeGeometry(distancia, elevation, border, 1, 1, 1, materialArray );

    pared = new THREE.Mesh( MovingCubeGeom, MovingCubeMat );
    pared.position.set(puntoMedioX, 75, puntoMedioY);
    pared.rotation.y = anguloRotacion;
    scene.add( pared );
}
```

Figura 46. Captura de código: Función de construcción de paredes en el plano 3D.

Como se puede observar, la función recibe como parámetros la distancia, punto medio, ángulo de rotación, calculados anteriormente y la orientación y link de imagen de la pared a construir.

Como se puede ver el borde de la pared siempre es 0 y la altura es siempre 150. Luego se crean las texturas del objeto a construir que va a representar a la pared, en este caso un rectángulo de tres dimensiones, el cual posee seis caras, (las texturas se definen antes de crear el objeto) y se almacenan en un arreglo. Como se puede observar son 6 seis texturas en total, alterando el orden de las últimas dos dependiendo de la orientación de la misma.

Las siguientes líneas de código construyen el objeto que representa a la pared. Luego se posiciona la misma según el punto pasado por parámetro (la altura se mantiene fija siempre). Ya casi en el final del código, se define el ángulo de rotación de la misma y finalmente se agrega la pared a la escena.

6.3 Animaciones

Como ya se mencionó, el modelo del plano tridimensional, se instancio para mostrar el posicionamiento virtual dentro del edificio de la Facultad de Informática. Tanto para avanzar como para retroceder de un punto al otro, es decir, el anterior o el siguiente dentro del camino estipulado para llegar a un destino, era necesario mostrar de alguna manera ese desplazamiento (si no se mostraba el desplazamiento, el cambio era muy brusco provocando que el usuario pudiera perderse siguiendo la navegación virtual). Para avanzar o retroceder virtualmente dentro del camino disponible para llegar al destino seleccionado se decidió colocar dos botones, cada uno llamando a una función con su implementación correspondiente.

El problema surgió al intentar implementar la lógica de las funciones anteriormente mencionadas. Primero era necesario girar la cámara que gestiona la vista hasta el punto anterior o siguiente según correspondiera, y luego hacerla mover linealmente hasta el punto destino. Por más que fuera sencillo calcular el ángulo de giro y la distancia a recorrer, la implementación no era simple ya que las herramientas disponibles (básicamente funciones como setInterval y setTimeout nativas de javascript) no garantizaban que el giro fuera siempre igual en distintas ejecuciones, y esto mismo se repetía con los avances.

A partir de esta problemática, se buscó una solución y tras investigar sobre el tema, se dio con un framework implementado en javascript que funciona como motor de animaciones, llamado Tweenjs. El mismo se integra de manera muy simple (sólo alcanza con agregar el archivo js al proyecto) y provee una simple interfaz para gestionar las animaciones. A continuación se muestra y explica detalladamente cómo se resolvieron los movimientos de un punto a otro dentro del plano tridimensional.

A continuación se muestra en la Figura 47 cómo se realizó el traslado de la cámara, desde un punto de navegación a otro (anterior o siguiente), dentro del camino mínimo resultante:

```
function irAPunto(destinoX,destinoY,destinoZ,cam) {

    //calcula pendiente y angulo de rotacion
    var pendiente = (cam.position.x - destinoX) / (cam.position.z - destinoZ);
    var angulo_rotacion = Math.atan(pendiente);

    angulo_rotacion = normalizarAnguloRotacion(cam,destinoZ,angulo_rotacion);

    var tiempo = tiempoRotacion(cam,angulo_rotacion);

    //rotacion
    var animacionRotacion = new TWEEN.Tween(cam.rotation).to({
         x: 0,
         y: angulo_rotacion,
         z: 0,},tiempo);
     animacionRotacion.easing(TWEEN.Easing.Linear.None).onUpdate(function () {
         //mientras rota no hace nada
     });
     animacionRotacion.onComplete(function () {
         //termina de rotar y avanza


        tiempo = tiempoAvanceRetroceso(cam,destinoX,destinoZ);
        var animacionAvance = new TWEEN.Tween(cam.position).to({
              x: destinoX,
              y: destinoY,
              z: destinoZ
         },tiempo);
        animacionAvance.easing(TWEEN.Easing.Linear.None).onUpdate(function () {
           //mientras avanza no hace nada
         });
        animacionAvance.onComplete(function () {
           //termina de avanzar y habilita la botonera
           habilitarBotonera();
        });
        //comienza a avanzar
        animacionAvance.start();



     });
```

Figura 47. Captura de código: Movimiento de cámara entre dos puntos

Como se observa en la imagen anterior, lo primero que se realiza, es la carga las variables que serán necesarias al momento de realizar la animación, que consta de girar la cámara hasta que su enfoque se dirija al punto siguiente de navegación, para luego realizar el recorrido hasta este.

La variable pendiente, contiene el valor de la pendiente entre el punto en el cual se ubica la cámara, y el punto al cual debe dirigirse. Esta variable, luego será utilizada para el cálculo del valor del ángulo que debe girar la cámara para enfocar el punto a donde se dirige, que se alojara en la variable angulo_rotacion.

El valor de angulo_rotacion luego es normalizado, para realizar un giro en el cual no supere una rotación de 360 grados para simplificar la vista de la cámara.

Como último valor para realizar correctamente esta función, tenemos el valor del tiempo, el cual dependerá del ángulo de giro en el caso de la rotación y de la distancia entre los puntos,

en el caso del desplazamiento, recreando una velocidad constante que se asemeja al caminar de una persona.

Y ahora lo importante de este código, la creación de la animación esperada a través de dos instancias de Tween, el primero, realizando la rotación necesaria para que el punto actual de la cámara haga foco en la del punto siguiente, para luego realizar el desplazamiento entre los dos puntos.

La primera instancia de Tween, se almacena en la variable animacionRotacion, la cual una vez cargada se procederá a utilizar la función Tween.start() para comenzar con la animación.

El parámetro enviado a la función constructor del Tween, es en este caso cam.rotation (cam es un parámetro que recibe la función irAPunto(), que hace referencia a la cámara de la librería Three.js que simula a su vez a la vista virtual de la aplicación), que indica la variable que será modificada durante la ejecución del algoritmo.

Luego en la función .to, se realiza la modificación de los valores de la rotación de la cámara, manteniendo los ángulos que no cambian, X y Z, y modificando el valor de Y, dando como valor la rotación resultante en el cálculo de valores realizado antes de instanciar Tween (angulo_rotacion). Al terminar la modificación, al final dentro de la misma función se le da el valor de tiempo en el cual se requiere que se realice la animación, en nuestro caso el valor de la variable tiempo.

En la función easing, se carga el valor Tween.Easing.Linear.None, para que la animación se realice normal, sin ningún tipo de efecto ni suavizado.

En la función onUpdate(), es decir mientras se produce la animación, no se realiza ningún otro tipo de modificación, por lo que se omite y se deja vacía.

El la función onComplete(), se incluye el código que se ejecutará luego de terminada la animación de la primera instancia de Tween. Para que la animación que deseamos realizar se produzca de manera correcta, es decir, primero la rotación y luego el desplazamiento, dentro de esta función se creó una nueva instancia de Tween, la cual se encarga de realizar la animación que desplaza la cámara de su punto actual hasta el siguiente punto del camino.

Esta instancia de Tween ubicada dentro de la función onComplete(), es muy similar al antes creado para realizar la rotación de la cámara, modificando en este caso, la posición de la cámara (cam.position). En este caso se modificarán los tres valores de la posición de la cámara, utilizando como valor los parámetros de la función irAPunto.

Al igual que el anterior función, no se utiliza ningún suavizado ni efecto, y tanto en onUpdate como en onComplete no se realiza ninguna acción adicional.

6.4 Navegación lógica y física

Como se mencionó anteriormente, se reutilizó casi completamente el plano de dos dimensiones para obtener una vista en miniatura simplificada de la navegación. Se le agregó un pequeño ícono con la imagen de un ojo para representar la posición lógica durante la navegación y se continuó utilizando el ícono con la imagen de una persona para representar la posición física del usuario.

En la Figura 48 se puede apreciar el plano 3D y en el sector inferior derecho el plano 2D en miniatura (el cual es ampliado en la Figura 49).

Figura 48. Mapa de dos dimensiones en miniatura

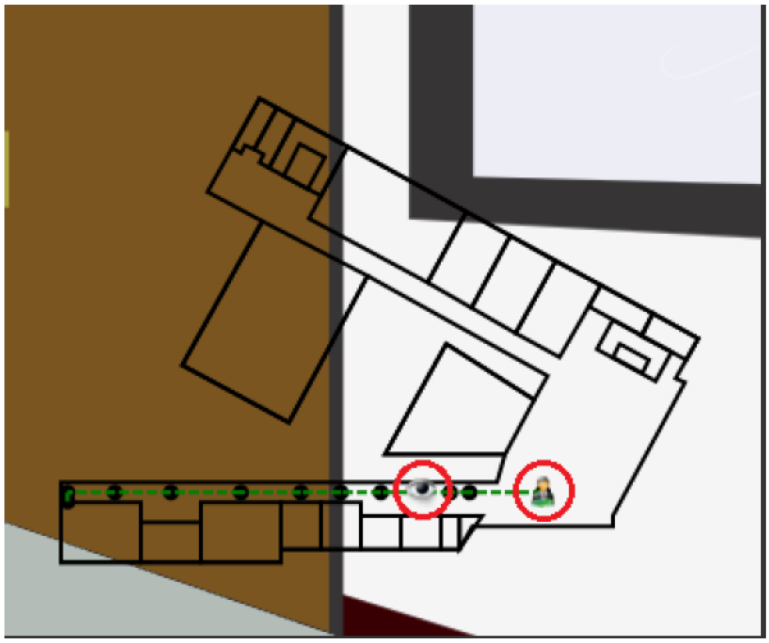


Figura 49. Íconos de mapa en dos dimensiones.

6.5 Lectura de códigos QR

La lectura de códigos QR se utiliza para reconocer la posición física de la persona que está utilizando la aplicación. Como se mencionó en los casos de uso, en primer lugar el usuario lee un código QR para determinar su posición física y luego selecciona un destino. Luego aparece el plano de tres dimensiones con la posición física del usuario como punto de partida y un dibujo con la forma de una persona en el mapa de dos dimensiones indicando la posición física del usuario dentro del plano de la Facultad. A medida que el usuario se desplaza lee un código QR para actualizar su posición física. Es importante aclarar que los cambios deben ser mostrados en el plano de dos dimensiones y no así en el de tres dimensiones que representa la posición y navegación lógica. La aplicación original no dispone de un lector de códigos QR integrado y en su lugar utiliza una aplicación externa que posiciona al usuario en un plano en

dos dimensiones únicamente en el punto de partida, con lo cual, el uso de la aplicación no varía demasiado. En el caso de la navegación en tres dimensiones, la posición física se actualiza constantemente, con lo cual llevó a analizar la forma en la que se leían los códigos QR y plantear la idea de integrar un lector de códigos QR dentro de la aplicación.

Lectura de códigos QR fuera de la aplicación

Ventajas

La aplicación consume menos recursos (memoria RAM y procesador) ya que no tiene que encender la cámara del dispositivo ni analizar códigos QR.

Desventajas

Es necesario salir de la aplicación para utilizar otra que lea códigos QR, lo cual produce que sea tedioso su uso.

Se necesita software adicional para correr la aplicación.

Es necesaria una forma de resguardar los datos con las posiciones física y lógica, destino, etc. actuales al salir de la aplicación y de alguna manera restaurarlas al leer la nueva posición física y volver a la misma.

Lectura de códigos QR dentro de la aplicación

Ventajas

No es necesario salir de la aplicación para actualizar la posición física, con lo cual no se debe realizar un resguardo de las posiciones física y lógica, destino, etc.

El uso de la aplicación es mucho más cómodo e intuitivo para el usuario.

No se necesita de software soporte para utilizar la aplicación.

Mayor facilidad para agregar nuevas funcionalidades a los códigos QR leídos.

Desventajas

Poco conocimiento por parte de los desarrolladores acerca de la disponibilidad de librerías que operen nativamente, sin necesidad de instalar plugins, respetando estándares, de código abierto y gratuitas para controlar la cámara del dispositivo y analizar códigos QR.

Con respecto a los recursos que podía llegar a consumir la integración de un lector de códigos QR dentro de la aplicación, teniendo en cuenta que ésta corre sobre dispositivos móviles de alta gama, se creyó conveniente no poner este ítem como desventaja.

La opción de integrar un lector de códigos QR fue la elegida, entonces se realizó una exhaustiva búsqueda de librerías que lograran concretar el objetivo.

Afortunadamente se encontró la librería llamada jsqrcode (mencionada en la sección de herramientas utilizadas), que cumplía con los aspectos anteriormente mencionados.

La integración con el sistema fue muy simple dado que la librería trae varios ejemplos de cómo instanciar la cámara del dispositivo, cómo leer un código y qué hacer luego de

concretarse la lectura.

El único inconveniente encontrado durante la utilización del scanner de códigos QR fue durante la lectura, ya que el lector para reconocer un código QR, ejecuta código para reconocer las imágenes capturadas por polling, con lo cual la aplicación desperdiciaba recursos constantemente, afectando notoriamente el rendimiento de la aplicación (sobre todo al momento de realizar las animaciones correspondientes a los cambios de posición lógica) en lugar de gastarlos únicamente al momento de actualizar la posición física. Para solucionar este problema, buscaron y encontraron los método encargados de activar/desactivar la cámara y se agregó un botón encargado de gestionar la lectura de los códigos QR. De esta manera la aplicación arranca con la cámara del dispositivo encendida pero inhabilitada para reconocer códigos QR. Al pulsar el botón anteriormente mencionado, se habilita el reconocimiento de códigos, y cuando el mismo es leído, se actualiza la posición física en el mapa de dos dimensiones y se desactiva el lector nuevamente.

De esta manera se logró solucionar la lectura de los códigos QR y logrando optimizar y mejorar el uso de la aplicación sin perder de vista el rendimiento de la misma ni la utilización innecesaria de recursos.

7. Ejemplos de uso

En este capítulo se explicará el funcionamiento del prototipo creado de navegación indoor 3D.

El acceso a la aplicación se realiza mediante el empleo de un navegador WEB. Cabe destacar que esta aplicación es compatible con todos los navegadores disponibles.

Una vez que hayamos accedido a la aplicación, la misma se presentará con un mensaje que pide la confirmación de uso de la cámara del dispositivo. Una vez confirmada la utilización de la cámara, la aplicación iniciará con la interfaz mostrada en la Figura 50.

Figura 50. Captura de pantalla principal de la aplicación.

Como podemos observar, la interfaz de la pantalla de inicio cuenta con un texto, el cual nos indica que debemos mostrar a la cámara activa, un código QR, que en esta aplicación es la codificación de una URL que identifica alguna de las estructuras disponibles.

Al ingresar un código correcto que represente a una estructura, la aplicación guardará esta estructura como la posición actual del usuario, para luego desplegar en la interfaz una lista con las estructuras navegables disponibles, obviando la estructura elegida anteriormente.

En la Figura 51 podemos ver la interfaz mencionada.

Figura 51. Captura de pantalla de lista de estructuras navegables.

Además de contar con la lista de estructuras disponible, la interfaz anterior también cuenta con un buscador por filtro de caracteres, que facilita la búsqueda de una estructura específica.

Una vez seleccionado el destino, la aplicación se encarga de calcular el camino mínimo entre los dos puntos preseleccionados conocidos como posición actual y estructura destino. El camino mínimo está conformado por el punto de la posición actual, los puntos intermedios, y el punto de navegación de la estructura destino. La aplicación despliega el mapa en tres dimensiones, ubicando la cámara principal en el punto de navegación actual del usuario, con la misma haciendo foco al siguiente punto del camino navegable.

En la Figura 52 podemos ver la interfaz desplegada luego de seleccionar los puntos de navegación inicio y destino.

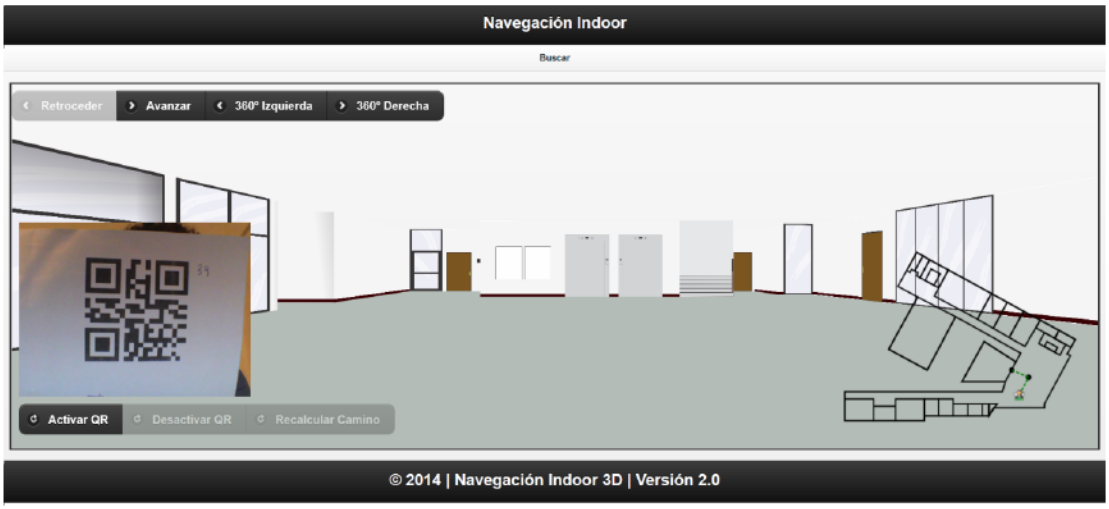


Figura 52. Captura de pantalla interfaz de navegación 3D.

En esa interfaz de la aplicación muestra el plano en tres dimensiones, compuesto por un piso en color material, y las distintas paredes que forman la facultad, cada una con una textura que representa la realidad.

Hacia la parte inferior derecha de la interfaz se encuentra el mapa en dos dimensiones, el cual nos ayuda a conocer la posición en la cual nos ubicamos físicamente a través del dibujo de una persona en el punto de navegación, y con un dibujo de un ojo, para conocer la ubicación lógica dentro del plano 3D. Cuando hablamos de ubicación física diferenciándola de la lógica, se debe a que la posición física dentro del edificio, se actualiza solamente cuando el usuario desde su dispositivo móvil lee un QR de algún punto navegable de la facultad. Mientras, la posición lógica, indicada con el icono del ojo en el camino, indica la posición de la cámara dentro de la aplicación en tres dimensiones desplegada en la interfaz, y al seleccionar un avance o un retroceso, solamente se actualiza el mapa de dos dimensiones.

La Figura 53 muestra un ejemplo donde la posición lógica se diferencia de la posición física del usuario.

Figura 53. Captura de pantalla interfaz navegación. Diferencia entre posición lógica y Física en plano 2D.

En el caso que la nueva posición física ingresada desde la lectura de un código QR, la modificación solo se ve reflejada en el mapa de dos dimensiones. A diferencia del caso anterior, en el caso que la nueva posición física ingresada no se encuentre en el camino, además de actualizar el plano 2D, se activa el botón “Recalcular camino”, como podemos ver en la Figura 54.

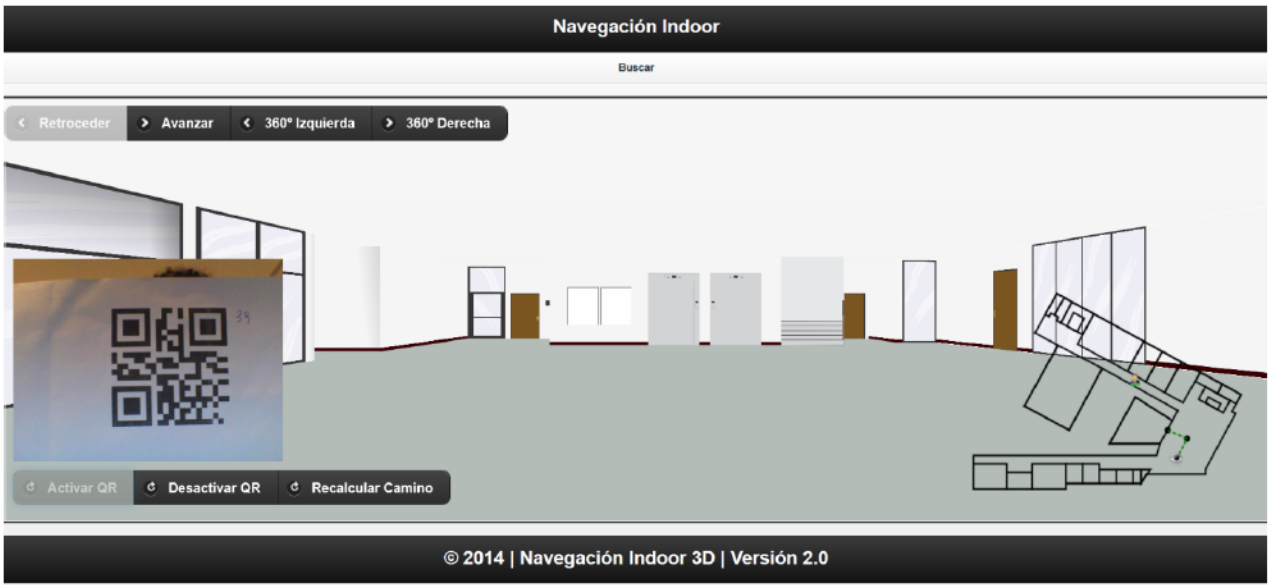


Figura 54. Captura de pantalla interfaz navegación. Posición lógica fuera de camino establecido.

En el caso de que el usuario seleccione la opción de "Recalcular Camino", el sistema almacena como posición actual la última posición física ingresada por código QR, y despliega nuevamente la interfaz de selección de destino como se vio anteriormente en este capítulo.

Volviendo a la interfaz de navegación, dentro del mapa de tres dimensiones nos encontramos con la botonera de la Figura 55.

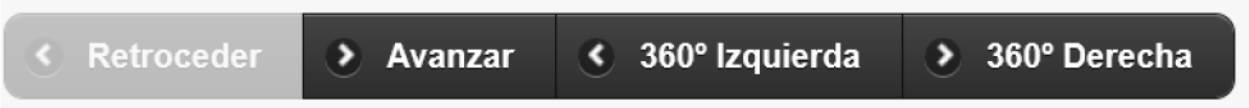


Figura 55. Captura de pantalla. Barra de navegación 3D.

En esta botonera podemos encontrar diferentes opciones para la navegación de la cámara (o posición lógica de usuario), que serán descritas a continuación:

Retroceder: Este botón se encuentra deshabilitado siempre que la cámara se encuentre en la primera posición del camino, siendo habilitando en todos los demás puntos. Su tarea una vez habilitada la opción, es de tomar el punto anterior a su posición actual, realizar el giro necesario para que el foco de la cámara se dirija hacia dicho punto, para luego desplazarse hacia él.

Avanzar: El botón avanzar se encuentra habilitado al comienzo del camino, y estará disponible siempre y cuando no se encuentre la cámara en el último punto del camino a recorrer. Su función es la misma que el botón retroceder, cambiando el punto al cual avanzar luego de hacer el giro necesario, siendo el punto siguiente en el camino.

Los botones 360° Izquierda y 360° derecha, realizan un movimiento especial dentro del mapa en tres dimensiones. Al presionar uno de estos dos botones, dependiendo si es derecha

o izquierda, la cámara realiza un giro en 360 grados para su lado respectivo, haciendo ver al usuario un panorama completo del punto en el cual se encuentra ubicada la cámara y su panorama completo.

Otro elemento importante en esta interfaz puede encontrarse en la parte inferior izquierda del mapa en tres dimensiones, estamos hablando de la vista de la cámara del dispositivo, y de la botonera con las opciones de “Activar QR” y “Desactivar QR”:

Como se puede ver en la Figura 56, dentro del cuadrado superior podemos ver la vista de la cámara del dispositivo, en la cual se mostrarán los códigos QR para que sean leídos.

Figura 56. Captura de pantalla. Barra de navegación QR.

Debido a que para la lectura de un código QR es necesario realizar un loop que cada cierta cantidad de segundos intente leer un código y esto consume una cantidad de recursos no deseados para que la aplicación se ejecute de una manera eficiente, se agregaron los dos botones de la parte inferior de la Figura.

El botón Activar QR, lo que realiza es habilitar la lectura de códigos, realizando el loop antes mencionado, mientras que la opción de Desactivar QR, detiene este loop de lectura, permitiendo solamente a la cámara del dispositivo mostrar la imagen actual de la vista de su cámara.

Volviendo a en concepto de posición lógica y física, debemos destacar que la forma para completar un camino mínimo desde su inicio hasta su final, es ingresar por código QR, el código que representa al punto de navegación de la estructura elegida como destino.

En caso de que esto suceda, la interfaz desplegará un popup que dará al usuario diferentes opciones a elegir.

En la Figura 57 podemos ver que la interfaz nos propone tres opciones diferentes:

- La opción cerrar, que cierra el popup y nos permite continuar navegando con el mismo camino que se estaba realizando anterior a la llegada al destino
- La opción navegar a otro sitio, en la cual la aplicación almacena la posición de destino como nueva posición actual, y despliega la interfaz de seleccionar destino, con la lista de las nuevas estructuras navegables posibles.
- Por último, la opción ver información despliega la interfaz detallada en la Figura 59.

Figura 57. Captura de pantalla. Popup de finalización de recorrido.

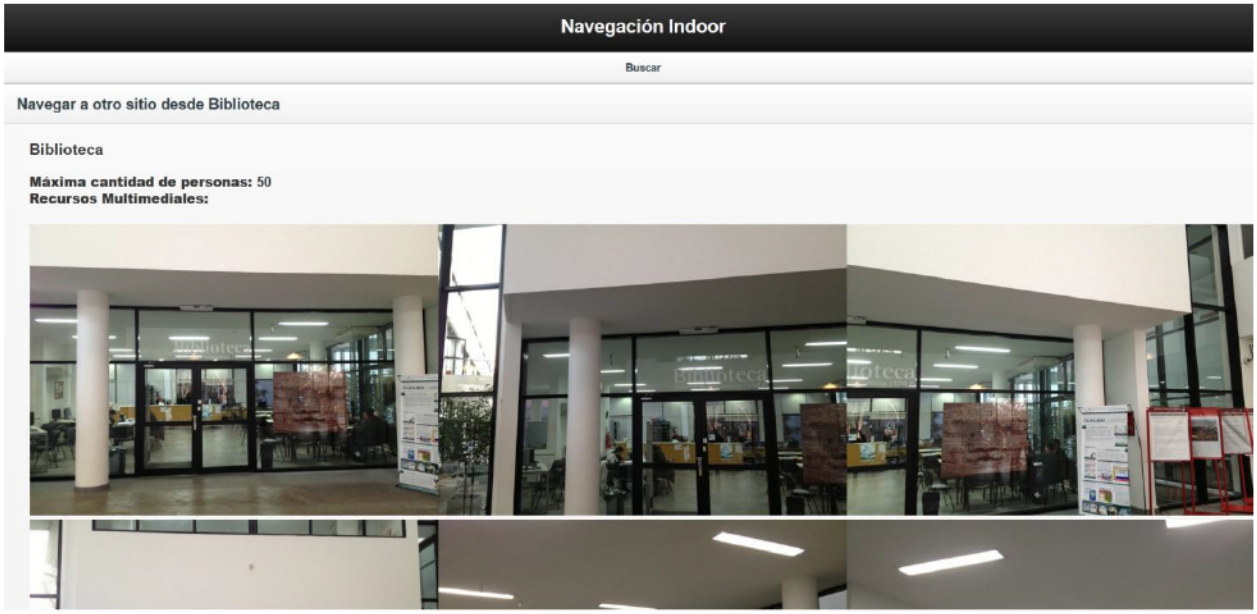


Figura 58. Captura de pantalla. Informe detallado de estructura navegable.

En esta pantalla se muestra la información multimedia de la estructura destino a la cual se llegó. Podemos encontrar datos, fotos y videos.

También encontramos la opción Navegar hacia otro destino desde Biblioteca (estructura actual), lo que nos dirige a la pantalla de seleccionar destino para realizar un nuevo camino desde la estructura actual.

Tanto en las pantallas de inicio, de seleccionar destino, como también en la pantalla de navegación, en la parte superior de la interfaz podemos encontrar el botón Buscar, el cual en cualquier de las pantallas nos dirige nuevamente a la pantalla de inicio de la aplicación.

Otro punto importante de la aplicación, es que almacena en una variable interna, siempre el último punto de navegación de destino, por lo que luego del primer ingreso a la pantalla de seleccionar destino, nos dará la opción de navegar directamente hacia el último destino seleccionado.

Como se observa en la parte superior izquierda de la lista de destinos ilustrada en la Figura 59, se encuentra la opción de navegar al último destino seleccionado, en este caso el Ascensor.

Figura 59. Captura de pantalla. Lista de estructuras navegables. Opción de continuación.

8. Conclusiones y trabajos futuros

8.1 Conclusiones

Luego de llevar a cabo el presente trabajo de tesis, se logró entender con profundidad el funcionamiento de los sistemas de posicionamiento en interiores, sus usos, beneficios y problemáticas aún no resueltas.

Mientras que los sistemas de posicionamiento en espacios en exteriores utilizando sistemas basados en satélite se consideran una tecnología madura, por el contrario, los sistemas de posicionamiento en interiores están aún en una fase de desarrollo.

Aunque existen diversas soluciones que aproximen una solución para este tipo de sistemas, la no existencia de un estándar definido para su implementación los convierte en un problema aún no resuelto.

Generalmente, salvo alguna excepción, la implementación de este tipo de sistemas depende de una inversión en infraestructura para que funcionen de manera precisa, siendo posible el uso de distintas tecnologías y algoritmia en función de la necesidad que se quiera cubrir.

Sin embargo, en los últimos años los dispositivos móviles han adquirido nuevas características y han mejorado otras tantas. Hoy en día casi todos los dispositivos cuentan con GPS, WiFi, Bluetooth, gran capacidad de procesamiento, aceleradores gráficos, capacidad de almacenamiento y buena capacidad de memoria RAM entre otras, lo cual motiva a que las inversiones anteriormente mencionadas sean llevadas a cabo.

Adicionalmente, las herramientas destinadas al desarrollo de este tipo de aplicaciones han mejorado mucho, en particular las que facilitan el uso de WebGL. No sólo se dispone de muchos más frameworks que hace no más de dos o tres años, sino que también son simples de usar y cada vez más eficientes y con más funcionalidades, lo cual alienta y facilita a los programadores. A su vez se cuentan con más documentación disponible, al menos las más populares, con ejemplos y con continuas mejoras. Sin dudas, la consolidación de HTML5 como estándar y el soporte de hoy en día casi todos los navegadores del mercado, incluidos los de dispositivos móviles, ha fomentado este avance.

En lo que respecta al desarrollo, se logró reutilizar casi por completo el modelo de clases (Modelo) y la lógica del sistema original (Controlador), concentrando la mayoría del tiempo en la navegación 3D (sobre todo en parte gráfica), que siempre fue el objetivo principal de este trabajo. Si bien se encontraron algunas trabas durante el desarrollo (por ejemplo, la integración del lector de códigos QR en la aplicación), se lograron sortear permitiendo completar el trabajo, utilizando buenas prácticas de programación y diseño, satisfaciendo tanto los requerimientos funcionales, como los no funcionales mencionados en la introducción.

En esta tesis se presentó un prototipo de navegación 3D en espacios indoor, en particular para el edificio de la Facultad de informática. Sin embargo, el modelo planteado es general y puede ser usado para instanciarse en cualquier otro espacio. Esto le brinda flexibilidad a la solución planteada.

Los avances tecnológicos permiten llevar a la práctica prototipos como el que se planteo en esta tesis. Cabe destacar que esta funcionalidad no había podido ser puesta en práctica en 2012. Esto da un panorama de las posibilidades que se van ampliando a medida que la tecnología evoluciona.

8.2 Trabajos a Futuro

A continuación, se detallan algunas funcionalidades que se podrían mejorar y agregar al prototipo desarrollado.

Cambiar la tecnología de Códigos QR por otra que sea capaz de posicionar al usuario en el mapa automáticamente, es decir, sin la necesidad de tener que decirle a la aplicación donde se encuentra a cada momento. Es decir, combinar otros mecanismos de sensado de posicionamiento.

Agregar otro tipo de sensores, por ejemplo, la brújula, y acorde a eso, se le ajuste el plano 2D para decirle para donde está mirando actualmente el usuario. Marcando, por ejemplo, el ángulo de visión.

Extender la facultad implementando la parte nueva y el primer y segundo piso. La aplicación cuenta con una interfaz para poder agregar, quitar o editar estructuras, paredes, caminos, etc. Esto implica resolver como posicionar al usuario en los diferentes pisos, más aun si se usan diferentes tipos de censados como parte del sistema.

Vistas 360 de interior de aulas que permitan observar con mayor detalle las mismas al llegar a un destino, aprovechando que Three.js ya provee esta funcionalidad. Se podría además combinar con información de lo que se está dictando en ese momento en cada aula.

Agregar videos y audios en la sección de detalles.

Recorridos interactivos, en los cuales la cámara se dirija sola desde el inicio hasta el destino.

Tours guidados, por ejemplo, ayudar a los ingresantes a recorrer la facultad indicándole que es cada lugar, que tramites se hacen, etc.

Agregar objetos del interior como por ejemplo, sillones, mesas, plantas columnas, etc. Considerar estos obstáculos involucra crear puntos de navegación que los “esquiven”, así los caminos son más reales.

Agregar navegación libre, programando las colisiones contra paredes y demás objetos. Esto podría servir para asistir, por ejemplo, a disminuidos visuales.

Además de las mejoras mencionadas al prototipo el modelo podría ser instanciado y crearse aplicaciones para otros dominios, por ejemplo, museos, shoppings, etc.

9. Bibliografía

[1] Grønbæk, K., Kristensen, J.F., Ørbæk, P., and Eriksen, M.A. Physical Hypermedia: Organising Collections of Mixed Physical and Digital Material (2003)
In proceedings of the Fourtenth ACM conference on Hypertext and Hypermedia (Hypertext 03) Nottingham UK, August 26-30, pp 10-19

[2] Tesina de Doctorado de Cecilia Challiol "Desarrollo Dirigido por Modelos de Aplicaciones de Hipermedia Móvil" - Septiembre de 2011 - Doctor en Ciencias Informáticas - Facultad de Informática - Universidad Nacional de La Plata. - http://sedici.unlp.edu.ar/handle/10915/4208

[3] W3C - Estándar HTML5 - http://www.w3.org/html/wg/drafts/html/master/

[4] WebGL Working Group - http://www.khronos.org/registry/webgl/specs/1.0/

[5] "Navegación Indoor 2D/3D en dispositivos móviles" - Tesina de Grado – Facultad de Informática de la Universidad Nacional de La Plata – Luciano Appathie, Claudio Borre – Año 2012.

[6] Street View - https://www.google.com/maps/views/home?gl=es&hl=es

[7] Sistemas de Información Geográfica - http://www.uv.mx/cienciahombre/revistae/vol18num3/articulos/informacion%20geografica/index.htm

[8] ¿Que es un SIG? http://www.latitude92.com/es/ventajasesp/que-es-un-sig-esp

[9] Sistemas Basados en Localización - http://lbspro.com/?q=que-son-servicios-localizacion-LBS

[10] Terje Midtbø, Alexander S. Nossum, Are indoor positioning systems mature for cartographic tasks?. 2012.

[11] Posicionamiento en Interiores basado en dispositivos Móviles – Miguel Ruiz Cuesta http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20120507MiguelRuizCuesta.pdf

[12] Local Positioning Systems. LBS Applications and Services. Krzysztof Kolozdziej and Johan Hjelm. Ed. Taylor & Francis. Capítulo 5. Publicado en 2006 por CRC Press. Taylor & Francis Group.

[13] Diseño y desarrollo de un sistema de posicionamiento en interiores basado en Wi-Fi con tecnología Android http://e-archivo.uc3m.es/bitstream/handle/10016/8819/PFC_Adolfo_Chico_Ciprian.pdf?sequence=3

[14] Posicionamiento con una red de sensores inalámbricos (WSN): implementación para una aplicación real. - Daniel Egea Roca
http://www.recercat.net/bitstream/handle/2072/97337/PFC_DanielEgeaRoca.pdf?sequence=1

[15] Universidad Autonoma de Madrid / Posicionamiento en interiores basado en Dispositivos Moviles - Miguel Ruis Cuesta - Año 2012
http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20120507MiguelRuizCuesta.pdf

[16] HTML5
https://developer.mozilla.org/es/docs/HTML/HTML5

[17] WebGL: Biblioteca gráfica basada en Web - Carlo Samuel Cantero González – Año 2011
http://books.openlibra.com/pdf/WebGL.pdf

[18] Request Animation Frame for Better Performance
http://html5hub.com/request-animation-frame-for-better-performance/#i.symcir77vdpqr5

[19] requestAnimationFrame y setInterval
http://www.project-cyan.com/2012-05-requestanimationframe-adios-al-setinterval/

[20] requestAnimationFrame
http://juegoscanvas.blogspot.com.ar/2012/11/requestanimationframe.html

[21] La clase Tween (TweenJS)
http://webintenta.com/la-clase-tween.html

[22] Introducción a jQueryMobile
http://www.baluart.net/articulo/introduccion-a-jquery-mobile

[23] ¿Qué es Symfony?
http://symfony.com/es/what-is-symfony

[24] Symfony
http://es.wikipedia.org/wiki/Symfony

Bibliografía adicional consultada

A. Butz, J. Baus, A. Krüger, and M. Lohse. A hybrid indoor navigation system. In IUI2001: International Conference on Intelligent User Interfaces, New York, 2001.ACM.

Chang, Y.J., Tsai, S.K., Chang, Y.S.,Wang, T.Y.: A NovelWayfinding System Based on Geo-Coded QR-Codes for Individuals With Cognitive Impairments. In: Assets '07: Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility, New York, NY, USA, ACM (2007) 231–232.

E. W. Dijkstra. A note on two problems in connection with graphs. Numerische Mathematik, (51):161–166, 1950.

Gartner, G., Frank, A., Retscher, G.: Pedestrian navigation system in mixed indoor outdoor environment the navio project (2004)

Fraunhofer. Messe Navigator. <http://www.iis.fraunhofer.de/ec/navigation/indoor/projekte/messe/index_d.html>, accessed at April 2011

Ran, L., Helal, S., Moore, S.: Drishti: An integrated indoor/outdoor blind navigation system and service. In: PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), Washington, DC, USA, IEEE Computer Society (2004)

M. Sorrows and S. Hirtle. The Nature of Landmarks for Real and Electronic Spaces. In C. Freksa and D. M. Mark, editors, COSIT '99, volume 1661, pages 37–50. Springer, 1999

Streeter, L.A., Vitello, D., Wonsiewicz, S.A.: How to Tell People Where to Go: Comparing Navigational Aids. International Journal of Man-Machine Studies 22(5) (1985) 549–562

Tsetsos, V., Anagnostopoulos, C., Kikiras, P., Hasiotis, P., Hadjiefthymiades, S.: A Human-Centered Semantic Navigation System for Indoor Environments. Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on (July 2005) 146–155

Anexo A: Casos de Uso reutilizados

Usuarios Comunes

Nombre: Buscar Destino (dentro del listado de destinos).
Descripción El usuario busca un destino (estructura) a donde desea navegar.
Actores: Usuario, Sistema.
Precondición: El usuario tiene que haber leído un código QR.
Curso de Eventos: 1. El sistema presenta al usuario un buscador de estructuras y una lista con todas las estructuras disponibles y navegables. 2. El usuario ingresa parte del nombre de la estructura a la cuál desea navegar. 3. El sistema acota el listado de estructuras basándose en el nombre ingresado por el usuario
Post condición: El usuario obtiene la estructura a la cual desea dirigirse.

Nombre: Navegar a Destino.
Descripción El usuario navega desde el origen al destino.
Actores: Usuario, Sistema.
Precondición: El usuario filtro y encontró el destino al que desea navegar.
Curso de Eventos: 1. El usuario selecciona el destino al que desea navegar. 2. El sistema procesa la información enviada por el usuario realizando el cálculo del camino mínimo entre el punto de origen actual y el destino. 3. El usuario visualiza en su interfaz un plano con un camino desde su posición actual y hasta el destino seleccionado.
Post condición: El usuario obtiene un mapa con una ruta desde su posición actual hasta el lugar de destino.

Nombre: Ver detalle de Estructura.
Descripción El usuario selecciona la opción ver información.
Actores: Usuario, Sistema.
Precondición: El usuario tiene que haber llegado al destino navegado.
Curso de Eventos: 1. El usuario selecciona la opción ver información luego de llegar al destino. 2. El sistema procesa la petición del usuario y busca toda la información asociada con la estructura. 3. El usuario visualiza en su interfaz toda la información de la estructura seleccionada.
Post condición: El usuario obtiene un detalle (recursos multimediales, nombre, capacidad, etc.) de la estructura seleccionada.

Usuarios registrados

Nombre: Crear Estructura.
Descripción: El Usuario registrado da de alta una estructura en el sistema.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, crear una estructura. 4. El usuario registrado selecciona del menú la función, crear una estructura. 5. El usuario registrado ingresa los atributos: Nombre, Tipo de Estructura y Capacidad. 6. El sistema valida la información enviada por el usuario y crea la nueva estructura.
Post condición: Se crea una nueva estructura en el sistema.

Nombre: Editar Estructura.
Descripción: El Usuario registrado selecciona una estructura para ser editada.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, editar una estructura. 4. El usuario registrado busca en el listado de estructuras la que desea modificar y selecciona la opción de editar. 5. El usuario registrado modifica los atributos necesarios. 6. El sistema valida la información enviada por el usuario y modifica los atributos de la estructura.
Post condición: Se cambian los atributos de una estructura.

Nombre: Borrar Estructura.
Descripción: El Usuario registrado selecciona una estructura para ser eliminada.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, borrar una estructura. 4. El usuario registrado busca en el listado de estructuras la que desea eliminar y selecciona la opción borrar. 5. El sistema pregunta al usuario si realmente desea realizar el borrado físico de la estructura. 6. El usuario registrado confirma que desea eliminar la estructura.

7. La estructura es eliminada del sistema.
Post condición: Se elimina la estructura del sistema.

Nombre: Crear Punto.
Descripción: El Usuario registrado da de alta un punto en el sistema.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado registrado. Entre ellas, crear un punto. 4. El usuario registrado selecciona del menú la función, crear un punto. 5. El usuario registrado ingresa los atributos: Punto de origen X y Punto de Origen Y. También puede seleccionar una estructura para ser asociada con el punto. 6. El sistema valida la información enviada por el usuario y crea un nuevo punto en el sistema
Post condición: Se crea un nuevo punto en el sistema.

Nombre: Editar Punto.
Descripción: El Usuario registrado selecciona un punto para ser editado.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, editar un punto. 4. El usuario registrado busca en el listado de puntos el que desea modificar y selecciona la opción de editar.

5. El usuario registrado modifica los atributos necesarios.
Post condición: Se cambian los atributos de un punto.

Nombre: Borrar Punto.
Descripción: El Usuario registrado selecciona un punto para ser eliminado.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, borrar un punto. 4. El usuario registrado busca en el listado de puntos el que desea eliminar y selecciona la opción borrar. 5. El sistema pregunta al usuario si realmente desea realizar el borrado físico del punto. 6. El usuario registrado confirma que desea eliminar el punto. 7. El punto es eliminado del sistema.
Post condición: Se elimina el punto del sistema.

Nombre: Crear Punto de Navegación.
Descripción: El Usuario Registrado da de alta un punto de navegación en el sistema.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, crear un punto de navegación. 4. El usuario registrado selecciona del menú la función crear punto de navegación.

5. El usuario registrado ingresa los atributos: Nombre y selecciona el punto con el que el punto de navegación está asociado.

6. El sistema valida la información enviada por el usuario y crea el nuevo punto de navegación.

Post condición: Se crea un nuevo punto de navegación en el sistema.

Nombre: Editar Punto de Navegación.

Descripción: El Usuario registrado selecciona un punto de navegación para ser editado.

Actores: Usuario registrado, Sistema.

Precondición: El usuario debe encontrarse logueado en el sistema.

Curso de Eventos:

1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña.

2. El sistema valida la información del usuario.

3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, editar un punto de navegación.

4. El usuario registrado busca en el listado de puntos de navegación el que desea modificar y selecciona la opción de editar.

5. El usuario registrado modifica los atributos necesarios.

Post condición: Se cambian los atributos de un punto de navegación.

Nombre: Borrar Punto de Navegación.

Descripción: El Usuario registrado selecciona un punto de navegación para ser eliminado.

Actores: Usuario registrado, Sistema.

Precondición: El usuario debe encontrarse logueado en el sistema.

Curso de Eventos:

1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña.

2. El sistema valida la información del usuario.

3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, borrar un punto de navegación.

4. El usuario registrado busca en el listado de puntos de navegación el que desea eliminar y selecciona la opción borrar. 5. El sistema pregunta al usuario si realmente desea realizar el borrado físico del punto de navegación. 6. El usuario registrado confirma que desea eliminar el punto de navegación. 7. El punto de navegación es eliminado del sistema.
Post condición: Se elimina el punto de navegación del sistema.

Nombre: Crear Relación entre Puntos de Navegación.
Descripción: El Usuario registrado da de alta una relación entre dos puntos de navegación.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, crear una relación entre puntos de navegación. 4. El usuario registrado selecciona del menú la función, crear un relación entre puntos de navegación 5. El usuario registrado ingresa los atributos: Punto de Navegación 1, Puntos de Navegación 2 y la distancia que hay entre los mismos. 6. El sistema valida la información enviada por el usuario y crea la relación entre los puntos de navegación seleccionados.
Post condición: Se crea una nueva relación entre dos puntos de navegación.

Nombre: Editar Relación entre Puntos de Navegación.
Descripción: El Usuario registrado selecciona una relación entre puntos de navegación para ser editada.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos:

1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, editar una relación entre puntos de navegación. 4. El usuario registrado busca en el listado de relaciones entre puntos de navegación la que desea modificar y selecciona la opción de editar. 5. El usuario registrado modifica los atributos necesarios. 6. El sistema valida la información enviada por el usuario y modifica los atributos de la relación entre puntos de navegación
Post condición: Se modifican los atributos de una relación entre dos puntos de navegación.

Nombre: Borrar Relación entre Puntos de Navegación.
Descripción: El Usuario registrado selecciona una relación entre puntos de navegación para ser eliminada.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, borrar una relación entre puntos de navegación. 4. El usuario registrado busca en el listado de relaciones entre puntos de navegación y elige la que desea eliminar seleccionando la opción borrar. 5. El sistema pregunta al usuario si realmente desea realizar el borrado físico de la relación entre puntos de navegación. 6. El usuario registrado confirma que desea eliminar la relación entre puntos de navegación. 7. La relación entre puntos de navegación es eliminada del sistema.
Post condición: Se elimina la relación entre puntos de navegación del sistema.

Nombre: Crear Multimedia.
Descripción El Usuario registrado da de alta un recurso multimedia en el sistema.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, crear multimedia. 4. El usuario registrado selecciona del menú la función, crear multimedia. 5. El usuario registrado ingresa los atributos: Nombre, tipo de multimedia y selecciona la estructura con la cual el recurso multimedia estará asociado. 6. El sistema valida la información enviada por el usuario y crea un recurso multimedia en el sistema.
Post condición: Se crea un nuevo recurso multimedia en el sistema.

Nombre: Editar Multimedia.
Descripción El Usuario registrado selecciona un recurso multimedia para ser editado.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario registrado. Entre ellas, editar un recurso multimedia. 4. El usuario registrado busca en el listado de recursos multimediales el que desea modificar y selecciona la opción de editar. 5. El usuario registrado modifica los atributos necesarios. 6. El sistema valida la información enviada por el usuario y modifica los atributos del recurso multimedia.
Post condición: Se cambian los atributos de un recurso multimedia.

Nombre: Borrar Multimedia.
Descripción El Usuario registrado selecciona un recurso multimedia para ser eliminado.
Actores: Usuario registrado, Sistema.
Precondición: El usuario debe encontrarse logueado en el sistema.
Curso de Eventos: 1. Un usuario realiza inicio de sesión utilizando su nombre de usuario y contraseña. 2. El sistema valida la información del usuario. 3. El sistema despliega las funciones disponibles al usuario. Entre ellas, borrar un recurso multimedia. 4. El usuario registrado busca en el listado de recursos multimediales el que desea eliminar y selecciona la opción borrar. 5. El sistema pregunta al usuario si realmente desea realizar el borrado físico del recurso multimedia. 6. El usuario registrado confirma que desea eliminar el recurso multimedia. 7. El recurso multimedia es eliminado del sistema.
Post condición: Se elimina el recurso multimedia del sistema.

Anexo B: Implementación del Backend

A continuación se presenta como se fue desarrollando la aplicación 3D, mencionando los cambios y adaptaciones producidos en la aplicación original como así también las funcionalidades agregadas, tanto en el backend como en el frontend de la aplicación.

Una vez importado el proyecto al IDE Netbeans, se procedió a modificar el **schema** de la misma, es decir, las entidades y las relaciones de la aplicación, se siguió utilizando Doctrine como ORM de Symfony. La estructura del archivo schema.yml quedó definido de la siguiente manera:

Definición de la clase Estructura (sin modificaciones)

```
1   #Modelo de datos Indoor Navigation 2D/3D
2   Estructura:
3     columns:
4       nombre:
5         type:     string(50)
6         notnull:  true
7       tipo:
8         type: integer
9         notnull: true
10      capacidad:
11        type:     integer
12        notnull:  true
13      es_navegable:
14        type:     boolean
15        default:  true
16
```

Figura 60. Captura de código: Definición de la clase Estructura.

Definición de la clase Multimedia (sin modificaciones)

```
17  Multimedia:
18    columns:
19      nombre:
20        type:     string(255)
21        notnull:  true
22      tipo:
23        type: integer
24        notnull: true
25      estructura_id:
26        type:     integer
27    relations:
28      Estructura:
29        refClass:   Estructura
30        onDelete:   cascade
31        local:      estructura_id
32        foreign:    id
33
```

Figura 61. Captura de código: Definición de la clase Multimedia.

Definición de la clase Puntos (sin modificaciones)

```
34  Puntos:
35    columns:
36      punto_origen_x:
37        type:     integer
38        notnull:  true
39      punto_origen_y:
40        type:     integer
41        notnull:  true
42      estructura_id:
43        type:     integer
44    relations:
45      Estructura:
46        refClass:   Estructura
47        onDelete:   cascade
48        local:      estructura_id
49        foreign:    id
50
```

Figura 62. Captura de código: Definición de la clase Puntos.

Definición de la clase PuntoNavegacion (sin modificaciones)

```
51  #-----------------------------------------------
52  #    modelo que define la union de puntos      #
53  #-----------------------------------------------
54  PuntoNavegacion:
55    columns:
56      nombre:
57        type:     string(50)
58        notnull:  true
59      punto_origen_x:
60        type:     integer
61        notnull:  true
62      punto_origen_y:
63        type:     integer
64        notnull:  true
65      estructura_id:
66        type:     integer
67    relations:
68      Estructura:
69        refClass:   Estructura
70        onDelete:   restrict
71        local:      estructura_id
72        foreign:    id
73      PuntoNavegacion:
74        class: PuntoNavegacion
75        local: punto_navegacion_1_id
76        foreign: punto_navegacion_2_id
77        refClass: PuntoNavegacionPuntoNavegacion
78        equal: true
79
```

Figura 63. Captura de código: Definición de la clase PuntoNavegación.

Definición de la clase PuntoNavegacionPuntoNavegacion (sin modificaciones)

```
80  PuntoNavegacionPuntoNavegacion:
81    columns:
82      punto_navegacion_1_id:
83        type:     integer
84        primary: true
85      punto_navegacion_2_id:
86        type:     integer
87        primary: true
88      distancia:
89        type:     float
90        notnull:  true
```

Figura 64. Captura de código: Definición de la clase PuntoNavegacionPuntoNavegacion.

Definición de la clase OrientacionPared (nueva)

```
92  OrientacionPared:
93    columns:
94      nombre:
95        type:     string
96        notnull:  true
97
```

Figura 65. Captura de código: Definición de la clase OrientacionPared.

Definición de la clase ParedDibujable (nueva)

```
98  ParedDibujable:
99    columns:
100     punto_1_id:
101       type:     integer
102       notnull:  true
103     punto_2_id:
104       type:     integer
105       notnull:  true
106     link_imagen:
107       type:     string
108       notnull:  true
109     descripcion:
110       type:     string
111       notnull:  true
112     orientacion_pared_id:
113       type:     integer
114       notnull:  true
115   relations:
116     OrientacionPared:
117       class:   OrientacionPared
118       onDelete:   restrict
119       local:      orientacion_pared_id
120       foreign:    id
121     Punto1:
122       class:   Puntos
123       onDelete:   restrict
124       local:      punto_1_id
125       foreign:    id
126     Punto2:
127       class:   Puntos
128       onDelete:   restrict
129       local:      punto_2_id
130       foreign:    id
```

Figura 66. Captura de código: Definición de la clase ParedDibujable.

Como se puede observar, el modelo original no se modificó, simplemente se agregaron dos clases para lograr la implementación en tres dimensiones. ParedDibujable, como se mencionó en la sección de implementación, representa a las paredes que se van a utilizar durante la construcción del escenario que representa la facultad, la misma posee una orientación de pared para determinar de qué lado se agrega la textura durante la construcción de la misma.

Una vez actualizado el schema, fue necesario actualizar la base de datos, generar el modelo de las clases nuevas, como así también los filtros y formularios que gestiona Symfony por medio del comando ***"php symfony doctrine:build --all".***(Figura 67)

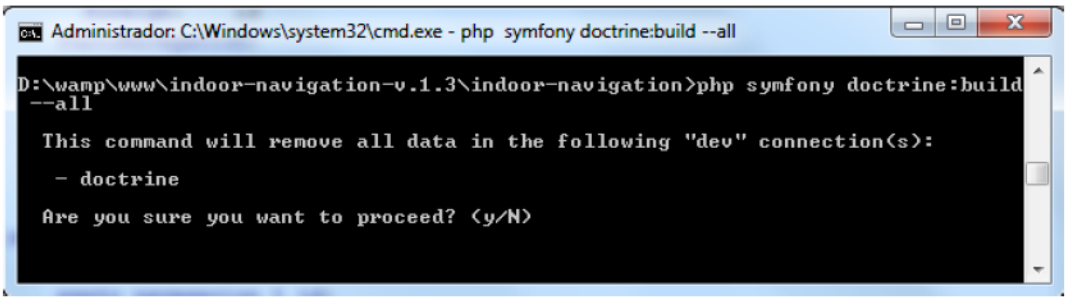


Figura 67. Actualización de base de datos.

3D Indoor Navigation - Administración - Pantalla de Login (Figura 68)

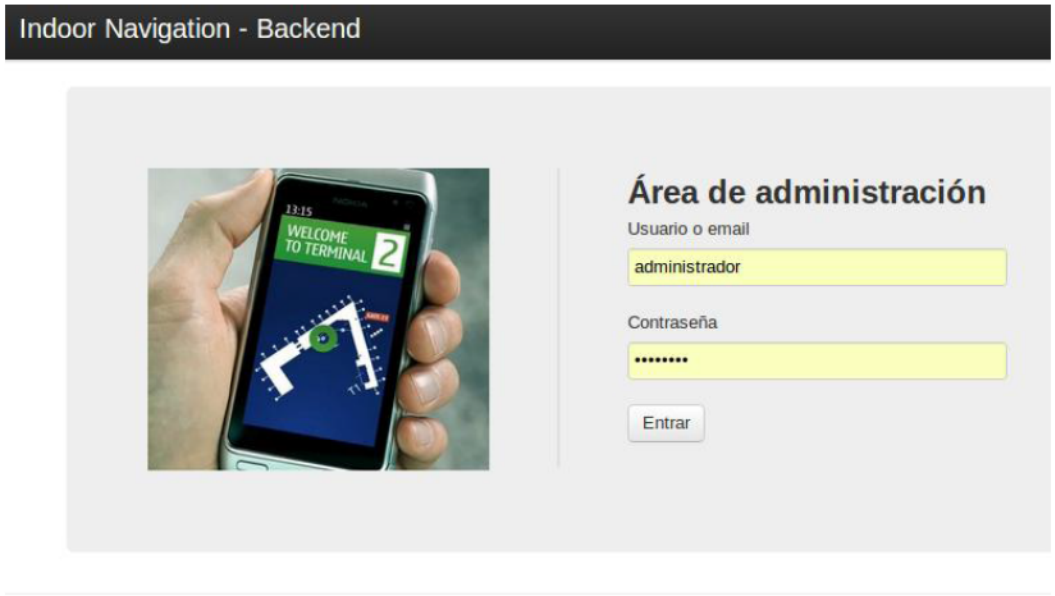
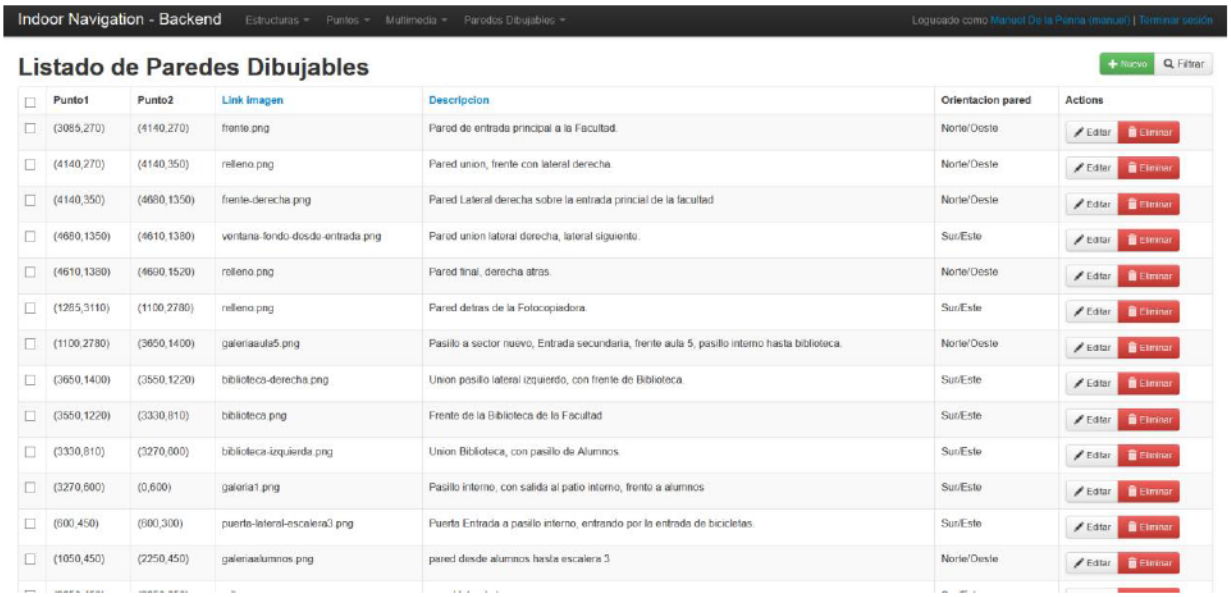


Figura 68. Pantalla principal de la aplicación de administración.

En esta aplicación no se modificó nada, únicamente se agregó un CRUD de la clase ParedDibujable. (Figura 69)



Punto1	Punto2	Link imagen	Descripcion	Orientacion pared	Actions
(3085,270)	(4140,270)	frente.png	Pared de entrada principal a la Facultad.	Norte/Oeste	Editar Eliminar
(4140,270)	(4140,350)	relleno.png	Pared union, frente con lateral derecha.	Norte/Oeste	Editar Eliminar
(4140,350)	(4680,1350)	frente-derecha.png	Pared Lateral derecha sobre la entrada principal de la facultad	Norte/Oeste	Editar Eliminar
(4680,1350)	(4610,1380)	ventana-fondo-desde-entrada.png	Pared union lateral derecha, lateral siguiente.	Sur/Este	Editar Eliminar
(4610,1380)	(4690,1520)	relleno.png	Pared final, derecha atras.	Norte/Oeste	Editar Eliminar
(1285,3110)	(1100,2780)	relleno.png	Pared detras de la Fotocopiadora.	Sur/Este	Editar Eliminar
(1100,2780)	(3650,1400)	galeriaaula5.png	Pasillo a sector nuevo, Entrada secundaria, frente aula 5, pasillo interno hasta biblioteca.	Norte/Oeste	Editar Eliminar
(3650,1400)	(3550,1220)	biblioteca-derecha.png	Union pasillo lateral izquierdo, con frente de Biblioteca.	Sur/Este	Editar Eliminar
(3550,1220)	(3330,810)	biblioteca.png	Frente de la Biblioteca de la Facultad	Sur/Este	Editar Eliminar
(3330,810)	(3270,600)	biblioteca-izquierda.png	Union Biblioteca, con pasillo de Alumnos.	Sur/Este	Editar Eliminar
(3270,600)	(0,600)	galeria1.png	Pasillo interno, con salida al patio interno, frente a alumnos	Sur/Este	Editar Eliminar
(600,450)	(600,300)	puerta-lateral-escalera3.png	Puerta Entrada a pasillo interno, entrando por la entrada de bicicletas.	Sur/Este	Editar Eliminar
(1050,450)	(2250,450)	galeriaalumnos.png	pared desde alumnos hasta escalera 3	Norte/Oeste	Editar Eliminar

Figura 69. Pantalla de edición de paredes dibujables (3D)

Clase Dijkstra

La implementación en dos dimensiones tenía implementada una clase llamada Dijkstra para calcular el camino mínimo de un punto a otro, dicha clase fue reutilizada ya que la lógica en tres dimensiones sigue siendo la misma.

Definición de la clase Dijkstra (Figura 70)

```php
<?php

class Dijkstra {

    const valor_maximo = '9999999';
    public $lista_de_adyacencias = array();
    public $cantidad_nodos;
    public $camino_a_recorrer = array();

    public $distancia = array();
    public $camino = array();

    /*
     * carga la lista de adyacencia con todos los puntos de navegacion,
     * y para cada uno agrega un arreglo.
     * Con sus vecinos y su distancia a ellos. Ejemplo.
     * 0 =>
      array
        0 =>
          array
            'punto_navegacion_1_id' => string '1' (length=1)
            'punto_navegacion_2_id' => string '2' (length=1)
            'distancia' => string '200.00' (length=6)
     */

```

Figura 70. Captura de código: Definición de la clase Dijkstra

- **valor_maximo**: es una constante utilizada para la inicialización del arreglo de distancias entre 2 puntos, definiendo un valor alto para luego actualizarla con la distancia mínima entre estos dos nodos.
- **lista_de_adyacencia**: en esta variable tendremos cada uno de los nodos y cada uno de sus vecinos.
- **cantidad_nodos**: como el nombre de la variable lo indica, contiene la cantidad de nodos del grafo.
- **camino_a_recorrer**: es un arreglo en que va guardando la distancia mínima entre los nodos y una vez finalizado el cálculo, se tiene todos los nodos a recorrer para llegar en la menor distancia al un destino.
- **distancia**: en este arreglo se guarda cual es la distancia para llegar a cada uno de los nodos, son inicializados con un valor alto para que a medida que se ejecuta el cálculo del camino mínimo se pueda actualizar con el valor más bajo.

Funciones getCamino() y cargarListaDeAdyacencia()

```
26    public function getCamino(){
27      return $this->camino_a_recorrer;
28    }
29
30    public function cargarListaDeAdyacencia(){
31
32      $puntos_navegacion = Doctrine::getTable('PuntoNavegacion')->findAll();
33      foreach ($puntos_navegacion as $pd){
34        $this->lista_de_adyacencias[] = $pd;
35      }
36
37      for ($i=0;$i<count($this->lista_de_adyacencias);$i++){
38        $this->lista_de_adyacencias[$i] =
          PuntoNavegacionPuntoNavegacionTable::retrieveById($this->lista_de_adyacencias[$i]->getId());
40      }
41
42      $this->cantidad_nodos = count($this->lista_de_adyacencias);
43    }
44
```

Figura 71. Captura de código: Funciones getCamino() y cargarListaDeAdyacencia()

Función caminoMinimoDesdeHasta()

```
58    public function caminoMinimoDesdeHasta($global_id_inicio, $id_destino){
59      $distancia = array();
60      $visitado = array();
61      $camino = array();
62
63      for ($i=0; $i < $this->cantidad_nodos ; $i++) {
64        $visitado[$i] = false;
65        $distancia[$i] = '9999999' ;//$this->valor_maximo;
66        $camino[$i] = -1;
67      }
68      $distancia[ $global_id_inicio -1 ] = 0;
69
70      while (!$this->noEstenTodosVisitados($visitado)){
71        $nodo_actual = $this->getMenorDesconocido($visitado,$distancia);
72        $visitado[$nodo_actual] = true;
73        foreach ($this->lista_de_adyacencias[$nodo_actual] as $vecinos){
74          if (($nodo_actual+1) == $vecinos['punto_navegacion_1_id'])
            $vecino_id = $vecinos['punto_navegacion_2_id'];
76          else
            $vecino_id = $vecinos['punto_navegacion_1_id'];
78
79          $vecino_id --;
```

Figura 72. Captura de código: Función caminoMinimoDesdeHasta() Parte 1.

```
80          //Actualizo la distancia si es menor a la del nodo actual mas la que tengo
81          if ((!$visitado[$vecino_id]) &&
82              ($distancia[$vecino_id] > $distancia[$nodo_actual] + $vecinos['distancia'])){
83            $distancia[$vecino_id] = $distancia[$nodo_actual] + $vecinos['distancia'];
84            $camino[$vecino_id] = $nodo_actual;
85          }
86        }//End Foreach
87      }//End While
88
89      $this->camino = $camino;
90      $this->distancia = $distancia;
91      $this->camino_a_recorrer =$this->imprimirCamino($id_destino, $camino, $distancia);
92    }
```

Figura 73. Captura de código: Función caminoMinimoDesdeHasta() Parte 2.

La función caminoMinimoDesdeHasta() es la encargada de calcular el camino mínimo. Recibe como parámetros el punto de navegación inicial y el de destino. Comienza inicializando las variables y poniendo los nodos como no visitados, con distancias grandes y el arreglo de camino en valores nulos. Luego crea un bucle que se repite mientras no sean visitados cada uno de los nodos, en el cual toma como nodo actual al nodo no visitado con menor distancia, lo marca como visitado y busca en la lista de adyacencia los nodos que tienen conexión con el nodo actual, y si la distancia entre nodos resultan menor a las que contiene el arreglo de distancias, procede a actualizar la distancia. Este bucle se repite hasta recorrer todos los nodos del grafo, y finalmente, se obtiene en el arreglo de distancias, las distancias mínimas entre nodos, y en el de camino, los ids de cada uno de los nodos a recorrer para llegar de un punto a otro.

Anexo C: Implementación del Controlador

Con la clase Dijkstra y aplicación de administración implementadas, más los datos cargados (se reutilizaron de la base de datos anterior, cargando únicamente las ***paredes dibujables*** utilizando los ***puntos*** existentes) necesarios para representar el edificio de la facultad y navegarlo se continuó con el controlador, el cual actúa como nexo entre la capa de presentación y el modelo.

La aplicación frontend tenía un módulo llamado ***main***, en el cual está implementada toda la lógica de la aplicación

A continuación se muestran las modificaciones realizadas en el controlador ***actions.class.php,*** del módulo ***main***:

Función Pública - executeOrigen()

```
54   * Función que dibuja el template Origen
55   * @param sfWebRequest $request
56   */
57  public function executeOrigen(sfWebRequest $request){
58
59
60  }
```

Figura 74. Captura de código: Función Pública - executeOrigen()

Esta acción se utiliza solamente para crear la pantalla inicial, es una acción vacía que luego llamará al template correspondiente para recibir el ingreso de un código QR con la posición física actual.

Función Pública - executeBuscar()

```
/**
 * Función que dibuja el template Busqueda
 * @param sfWebRequest $request
 */
public function executeBuscar(sfWebRequest $request){

  $puntoNavegacionOrigen = Doctrine::getTable('PuntoNavegacion')->find($request->getParameter('id_punto_navegacion_origen'));
  $this->getUser()->setAttribute('origen_id',$puntoNavegacionOrigen->getEstructuraId());
  $this->getUser()->setAttribute('actual_id',$puntoNavegacionOrigen->getId());
  $this->getUser()->setAttribute('escala',sfConfig::get('app_escala'));
  $this->estructuras = EstructuraTable::getNavegables($puntoNavegacionOrigen->getEstructuraId());

  //si tiene un destino de una busqueda anterior lo guarda
  $this->destino = $this->getUser()->getAttribute('fin_id');
  $this->estructura = Doctrine::getTable('Estructura')->find($this->destino);
}
```

Figura 75. Captura de código: Función Pública - executeBuscar()

Esta acción toma el punto de navegación de origen enviado como parámetro y setea en la sesión del usuario el id de estructura del punto actual (si tiene una estructura asociada, si no el valor es null), el id del punto de navegación ingresado por código QR y la escala para el plano en dos dimensiones. En la variable estructuras guarda todas las estructuras navegables que posteriormente van a ser mostradas en el menú mostrado en el template de esta acción. Por último si existe un destino anteriormente seteado (de alguna búsqueda anterior), se almacena en una variable su estructura. Esto permite mostrar un atajo en la vista de búsqueda al último destino seleccionado, posibilitando al usuario en caso de desviarse del camino pautado, recordar fácilmente el destino elegido originalmente.

Función Pública - executeNavegar()

```
13    /**
14     * Función que permite navegar desde un punto de la facultad (situado por una aplicación externa) a otro seleccionado por el usuario
15     * @param sfWebRequest $request
16     */
17    public function executeNavegar(sfWebRequest $request){
18      //Se setea en el usuario el id de la estructura a donde el usuario desea navegar
19      $this->getUser()->setAttribute('fin_id',$request->getParameter('est_id'));
20
21      if ($this->getUser()->getAttribute('fin_id') == $this->getUser()->getAttribute('origen_id')){
22          $this->getUser()->setFlash('notice','Ud. se encuentra parado en el mismo lugar al que desea llegar. Por favor seleccione un nuevo
destino');
23          $this->getUser()->setAttribute('fin_id', null);
24          $this->redirect('main/buscar?id_punto_navegacion_origen='.$this->getUser()->getAttribute('actual_id'));
25      }
26
27      $this->estructuras = Doctrine::getTable('Estructura')->findAll();
28      $this->puntos_todas_paredes = Doctrine::getTable('Puntos')->findAll();
29      $this->paredes_dibujables = Doctrine::getTable('ParedDibujable')->findAll();
30      $d = new Dijkstra();
31      $d->cargarListaDeAdyacencia();
32
33      $estructura = Doctrine::getTable('Estructura')->find($request->getParameter('est_id'));
34
35      if ((!$estructura) || (!$estructura->getEsNavegable())){
36        $this->getUser()->setFlash('error','El lugar a donde intenta navegar no está disponible.');
37        $this->redirect('@homepage');
38      }
39
40      //Relative URL Ej: http://localhost/indoor-navigation
41      $this->basepath = ('http'.($request->isSecure() ? 's' : '').'://').
42                        $request->getHost().$request->getRelativeUrlRoot();
43      //Si es Producción index.php sino frontend_dev.php
44      $this->enviroment = (sfConfig::get('sf_environment') == 'dev' ? 'frontend_dev.php' : 'index.php');
45
46      $punto_navegacion = PuntoNavegacionTable::getPuntoNavegacionParaEstructuraId($estructura->getId());
47      $this->destino = $punto_navegacion->getId();
48      $d->caminoMinimoDesdeHasta($this->getUser()->getAttribute('actual_id'), $this->destino);
49      $this->puntos_navegacion = $d->getCamino();
50
51    }
```

Figura 76. Captura de código: Función Pública - executeNavegar()

La acción navegar primero setea en la sesión del usuario el id de estructura del destino seleccionado. Luego compara el destino con el origen para ver si son distintos, si son iguales le avisa al usuario que ya se encuentra en el destino y sino continua la ejecución del código.

Asigna a diferentes variables para utilizar luego en la vista todas las estructuras, todos los puntos de mapa y las paredes dibujables. Crea una instancia de la clase Dijkstra para el cálculo de caminos e inicializa la lista de adyacencias. Valida si la estructura del punto de navegación seleccionado por el usuario se encuentra navegable. Utiliza la función caminoMinimoDesdeHasta() de la clase Dijkstra con los parámetros seleccionados por el usuario, y retorna a la capa de presentación el arreglo con el camino de puntos de navegación a dibujar.

Función Pública - executeDetalleEstructura()

```
87    public function executeDetalleEstructura(sfWebRequest $request){
88      $this->estructura =
89        Doctrine::getTable('Estructura')->find($request->getParameter('idEstructura'));
90      $this->destino =
91          EstructuraTable::getPuntoNavegacion($request->getParameter('idEstructura'));
92      $this->multimedia =
93        MultimediaTable::getMultimediaPara($request->getParameter('idEstructura'));
94      $this->getUser()->setAttribute('fin_id', null);
95    }
```

Figura 77. Captura de código: Función Pública - executeDetalleEstructura ()

La función detalleEstructura() toma del request los parámetros de selección del usuario el Id de la estructura que desea ver en detalle y el Id de Destino actual. Busca en la base de datos toda la información referente, y devuelve a la capa de presentación las variables necesarias para que el usuario pueda observar su selección.

Las funciones zoomIn() y zoomOut() realizaban la modificación del valor de la escala en el que se está encuentra dibujado el edificio y sus partes. Este valor era modificado cuando el usuario de la aplicación disparaba la acción desde la vista. En la nueva implementación, el plano de dos dimensiones tiene fijada la escala en un valor que no se modifica.

Las funciones públicas executeZoomIn() y executeZoomOut() fueron quitadas del sistema dado que esta funcionalidad se le quitó al mismo y en caso de que fuera necesario implementarla para las tres dimensiones, por la arquitectura planteada en el nuevo diseño, debería ir en la vista y no en el controlador.

Anexo D: Implementación del Frontend

En Symfony, las vistas se gestionan de la siguiente manera: por cada acción existe un template que representa a la vista de la misma, si se tiene por ejemplo la acción ***buscar***, representada por el método ***executeBuscar()***, existe un template llamado ***buscarSuccess.php*** que representa a la vista de la misma. Dentro del template se puede ejecutar código PHP e invocar algunas de las variables definidas en la acción. Por ejemplo, la instrucción ***$this->destino = $punto_navegacion->getId()***, guarda en la variable ***$destino*** un id de un punto de navegación el cual puede ser accedido desde la vista como una variable global.

Habiendo explicado cómo interactúa el controlador con el modelo, se procederá a mostrar la capa de presentación.

Template - layout.php

Symfony utiliza el patrón de diseño ***decorator*** para representar la vista de sus aplicaciones, logrando evitar duplicar código (ya sea código PHP o etiquetas HTML, por ejemplo para representar un menú).

El patrón *decorator* resuelve el problema mostrando el contenido con una plantilla (template de cada acción) que después se decora con una plantilla global que en Symfony se llama **layout**.

El layout por defecto de todas las aplicaciones es un archivo llamado layout.php que se encuentra en el directorio apps/frontend/templates/. En este directorio se guardan todas las plantillas globales de una aplicación. (Figura 78)

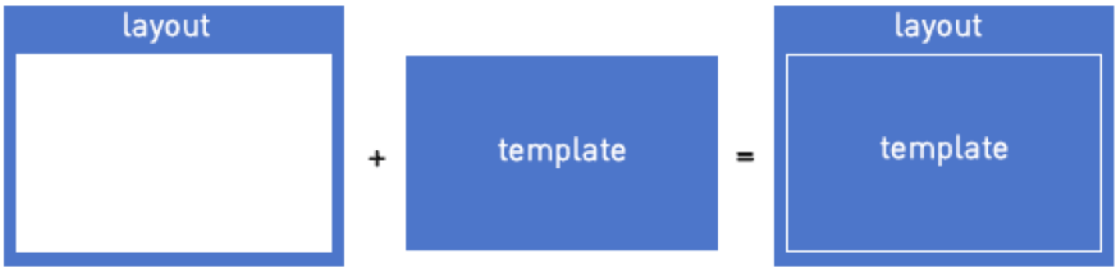


Figura 78. Patrón Decorator.

El layout.php de la aplicación quedó definido como se muestra en la figura 79.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html>
3    <head>
4        <title>Indoor Navigation</title>
5        <meta charset="utf-8" />
6        <?php include_stylesheets();?>
7        <?php include_javascripts();?>
8
9        <link rel="shortcut icon" href="<?php echo public_path('/favicon.ico') ?>" />
10   </head>
11   <body>
12     <div data-role="page" style="width:100%">
13       <div data-role="header">
14         <h1 style="font-size: 22px">Navegaci&oacute;n Indoor</h1>
15       </div><!-- /header -->
16
17       <div data-role="navbar">
18           <ul>
19             <li><?php echo link_to('Buscar', 'main/origen', array("rel" => "external", "style" => "font-size: 20px"));?></li>
20           </ul>
21       </div><!-- /navbar -->
22
23       <div data-role="content">
24
25         <?php echo $sf_content; ?>
26       </div><!-- /content -->
27
28       <div data-role="footer">
29         <h4 style="font-size: 22px"> &copy; 2014 | Navegaci&oacute;n Indoor 3D | Versi&oacute;n <?php echo sfConfig::get('app_version');?>
</h4>
30       </div><!-- /footer -->
31     </div><!-- /div data-role -->
32   </body>
33 </html>
```

Este script PHP es el encargado de estructurar la página principal de la aplicación frontend. Cada uno de los <divs> posee un atributo (data-role) que permite clasificar el tag HTML para darle ciertas propiedades (a través del framework jQueryMobile). A continuación, se detallan las propiedades que estructuran la página:

: Por defecto este atributo nos define la estructura de la página.
: Bloque cabecera de la página.

Figura 80. Captura del Bloque cabecera de la página.

: Bloque a utilizar para la barra de navegación

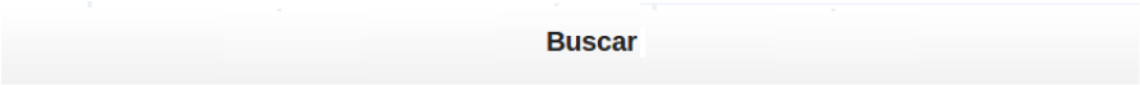


Figura 81. Captura del Bloque a utilizar para la barra de navegación

: Bloque que se utiliza para mostrar el contenido dinámico de la página. En este bloque veremos el resultado de disparar las acciones buscar(), navegar(), detalleEstructura(), etc.

El contenido se invoca mediante la instrucción ***<?php echo $sf_content; ?>***

: Bloque pie de la página.

© 2014 | Navegación Indoor 3D | Versión 2.0

Figura 82. Captura del Bloque de pie de página.

Template - barra_de_navegacion.php

```
<div style="position:absolute;" data-role="controlgroup" data-type="horizontal" data-mini="false">

  <a type="button" id="link_retroceder" href="javascript:getPuntoAnterior()"
     data-icon="arrow-l" data-theme="a" rel = "external">Retroceder </a>

  <a type="button" id="link_avanzar" href="javascript:getPuntoSiguiente()"
     data-icon="arrow-r" data-theme="a" rel = "external">Avanzar </a>

  <a id="link_360_izquierda" href="javascript:rotar360(2*Math.PI,camara)"
     data-role="button" data-icon="arrow-l" data-theme="a" rel = "external">360º Izquierda
  </a>
  <a id="link_360_derecha" href="javascript:rotar360(-2*Math.PI,camara)"
     data-role="button" data-icon="arrow-r" data-theme="a" rel = "external">360º Derecha
  </a>
</div>
```

Figura 83. Captura de código: Template - barra_de_navegacion.php

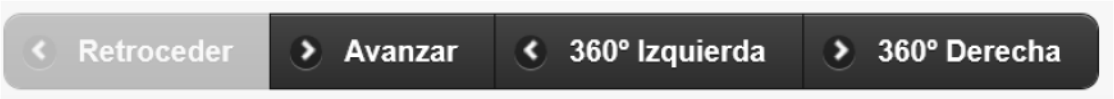


Figura 84. Captura de barra de navegación 3D

La barra de navegación le permite al usuario realizar diferentes acciones sobre el plano en tres dimensiones que se dibuja en el canvas.

Template - barra_qr.php

```
<div style="position:absolute; left: 10px; bottom:10px" data-role="controlgroup" data-type="horizontal" data-mini="false">

  <a id="link_encender_lector" href="javascript:encenderLector()"
     data-role="button" data-icon="refresh" data-theme="a" rel = "external">Activar QR
  </a>

  <a id="link_apagar_lector" href="javascript:apagarLector()"
     data-role="button" data-icon="refresh" data-theme="a" rel = "external">Desactivar QR
  </a>

  <a id="link_recalcular_camino" href="javascript:recalcularCamino()"
     data-role="button" data-icon="refresh" data-theme="a" rel = "external">Recalcular Camino
  </a>

</div>
```

Figura 85. Captura de código:Template - barra_qr.php

Figura 86. Captura de barra de códigos QR

La barra QR le permite al usuario activar y desactivar el lector de códigos QR que funciona por medio de la cámara del dispositivo, a su vez, le permite recalcular el camino en caso de que la posición física del usuario sea diferente a alguno de los puntos del camino pautado para llegar al destino.

Template - mensajes_usuario.php

```
<?php use_helper('I18N');?>

<?php if ($sf_user->hasFlash('error')): ?>
  <div class="mensaje_error"><?php echo __($sf_user->getFlash('error'), array(), 'sf_admin') ?></div>
<?php endif; ?>

<?php if ($sf_user->hasFlash('notice')): ?>
  <div style="margin: 4px 0;
    padding: 4px 4px 4px 30px;
    background: url(../images/notice.png) no-repeat 10px 4px;
    border-top: 1px solid #ddd;
    border-bottom: 1px solid #ddd;
    background-color: #E1F49D;
    color: black;
    font-weight: bold;"><?php echo sfOutputEscaper::unescape(__($sf_user->getFlash('notice'), array(), 'sf_admin'))
?></div>
<?php endif; ?>
```

Figura 87. Captura de código: Template - mensajes_usuario.php

Este template muestra en pantalla los mensajes de usuario, ya sean errores o simplemente avisos acerca de algún suceso.

Template - origenSuccess.php

```
<script>
    if (!window.WebGLRenderingContext) {
        // the browser doesn't even know what WebGL is

        window.location = "frontend2d.php?punto_navegacion_origen_id=1";

    }
    _BASEPATH = "<?php echo $basepath;?>";
</script>


<?php use_helper("I18N");?>
<h3 style="font-size: 20px"><?php echo __('Lea un código QR para determinar su ubicación
actual.');?></h3>
<div id="camara" class="origen">
    <video  id="camsource" autoplay width="320" height="240">Put your fallback message here.</video>
    <canvas id="qr-canvas" width="320" height="240" style="display:none"></canvas>
</div>

<script>
    activarCamara(true);
</script>
```

Figura 88. Captura de código: Template - origenSuccess.php

El template ***origen*** (Figura 89) le muestra al usuario el lector de códigos QR permitiéndole leer uno para determinar su posición física actual. En el caso de que el navegador no soporte WebGL, se redirecciona a la aplicación en dos dimensiones.

Figura 89. Captura de lector de códigos QR de la página principal.

Template - buscarSuccess.php

```
<?php use_helper("I18N");?>
<?php if ($sf_user->hasFlash('notice')): ?>
  <div class="success" id="success2">
    <?php echo $sf_user->getFlash('notice') ?>
  </div>
<?php endif; ?>
<?php
    if ($destino != null){
    echo link_to('Continuar navegando hacia '. $estructura->getNombre(),'main/navegar?est_id='.$destino,array(
                "rel" => "external", "style" => "font-size: 18px"));
    }
?>
<h3 style="font-size: 20px"><?php echo __('Seleccione el destino.');?></h3>
<ul data-role="listview" data-inset="true" data-filter="true" data-filter-placeholder="Escriba parte del nombre para
filtrar...">
  <?php foreach ($estructuras as $estructura):?>
  <li> <?php echo link_to($estructura->getNombre(),'main/navegar?est_id='.$estructura->getId(),array(
                "rel" => "external", "style" => "font-size: 18px"));?>
  </li>
  <?php endforeach?>
</ul>
```

Figura 90. Captura de código: Template - buscarSuccess.php

En el template ***buscar*** (Figura 91), se recorren todas las estructuras y para cada una dibuja se dibuja un tag <a> con el nombre de la estructura. Como se observa, el tag <ul> tiene ciertos atributos propios del framework jQueryMobile. Permitiendo mostrar los ítems en forma de lista con un buscador. Al hacer clic sobre el nombre de una estructura, será disparada la acción navegar() enviando como parámetro el ID de la estructura seleccionada.

Figura 91. Captura de listado de estructuras navegables de la pantalla de búsqueda.

Template - navegarSuccess.php

Este template es el encargado de dibujar los planos completo de la facultad, tanto en dos, como en tres dimensiones, las paredes, el camino hacia la estructura seleccionada por el usuario, etc. A continuación (Figura 92) se detalla en profundidad el código del template.

Show detail

```
1  <?php use_helper("JavascriptBase");?>
2
3  <!-- Interfaz de navegación-->
4  <div id="ThreeJS" style="position: relative; left:0; border-style: solid; left:0px; top:0px; height: 500px; width: window.innerWidth">
5      <?php include_partial('global/barra_navegacion');?><!-- Acá se dibuja la barra de control de navegacion-->
6      <div id="camara" class="navegar" style="position:absolute; left: 10px; bottom:70px">
7          <video  id="camsource" autoplay width="320" height="240">Put your fallback message here.</video>
8          <canvas id="qr-canvas" width="320" height="240" style="display:none"></canvas>
9      </div><!-- Acá se dibuja la cámara-->
10     <?php include_partial('global/barra_qr');?> <!-- Acá se dibuja la barra de control de la camara-->
11     <div id="container" style="position:absolute; right: 10px; bottom:10px"> </div><!-- Acá se dibuja el canvas 2D-->
12 </div><!-- Acá se dibuja el canvas 3D-->
13
14 <!-- Mensaje de llegada a destino -->
15 <div data-role="popup" id="popupDialog" data-overlay-theme="a" data-theme="a" style="max-width:600px;" class="ui-corner-all" aria-disabled="false" data-shadow="true"
   data-corners="true" data-transition="pop">
16   <div data-role="header" data-theme="a" class="ui-corner-top ui-header ui-bar-d" role="banner">
17     <h1 class="ui-title" role="heading" aria-level="1">Felicitaciones</h1>
18   </div>
19   <div data-role="content" data-theme="a" style="padding:15px;" class="ui-corner-bottom ui-content ui-body-d" role="main">
20     <p>Usted a llegado a su destino</p>
21     <?php echo link_to('Navegar a otro sitio' ,'main/origen',array("rel" => "external", " data-role" => "button", "data-icon" => "search" , "data-theme"=>"a", "data-
   corners" => "true"));?>
22     <?php echo link_to('Ver informacón' ,'main/detalleEstructura?idEstructura='.$sf_user->getAttribute('fin_id'),array("rel" => "external", " data-role" => "button",
   "data-icon" => "search" , "data-theme"=>"a", "data-corners" => "true"));?>
23     <?php echo link_to_function('Cerrar' ,'cerrarDialogo()',array("data-rel"=>"back" ,"data-role" => "button", "data-icon" => "back" ,  "data-theme"=>"a", "data-
   corners" => "true"));?>
24   </div>
25 </div>
```

Figura 92. Captura de código:Template - navegarSuccess.php

Como se observa en los comentarios del código, la primera parte define la interfaz de navegación, es decir, el plano en tres dimensiones, y dentro del mismo la barra de navegación, el lector de códigos QR, la barra QR, y el plano en dos dimensiones. (Figura 93)

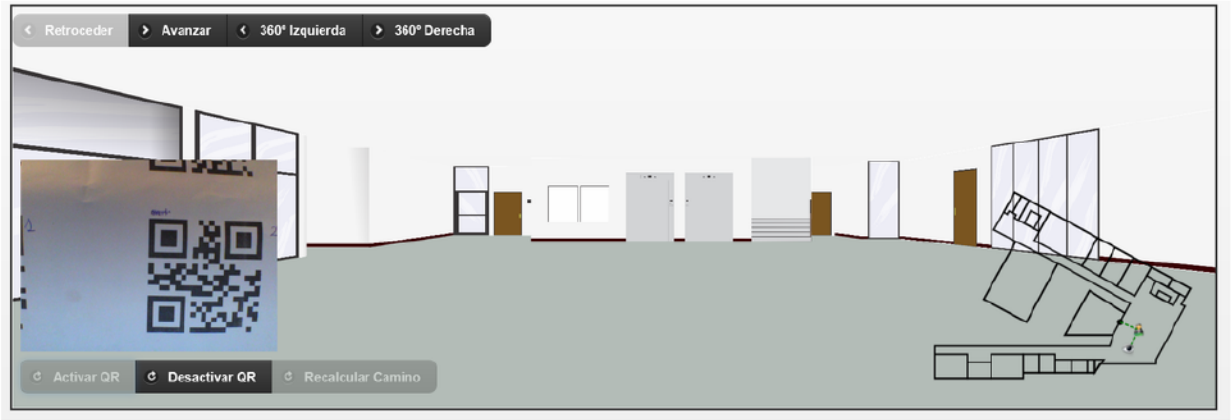


Figura 93. Captura de la pantalla de navegación.

La segunda parte define el formato del mensaje de llegada a destino (Figura 94) que va a ser mostrado en pantalla cuando el usuario lea la posición física del destino. Como se puede observar contiene también tags propios de la librería jQueryMobile.

Figura 94. Captura de popup con mensaje de llegada a destino

Luego se definen los posibles mensajes de usuario de la siguiente manera, finalizando las declaraciones de HTML5.

```
27 <!-- Mensajes de usuario -->
28 <?php include_partial('global/mensajes_usuario');?>
```

Figura 95. Captura de código: Mensajes de usuario

El resto del template contiene código javascript con algo de PHP (utilizado para obtener las variables del controlador) que gestiona toda la lógica del cliente, ya sea para crear el mapa como para gestionar diferentes eventos.

El código que se muestra en la Figura 96 se encarga de definir algunas variables que van a ser utilizadas por el plano 2D y a su vez la instanciación del Stage (escenario) y el Layer (capa), donde van representarse los diferentes elementos.

```
31 <script>
32     //declaraciones 2D
33     _BASEPATH = "<?php echo $basepath;?>";
34     _ENVIROMENT = "<?php echo $enviroment;?>";
35
36     app_maximo_y = <?php echo sfConfig::get('app_maximo_y')?>;
37     escala =  <?php echo $sf_user->getAttribute('escala')?>;
38
39     var stage = new Kinetic.Stage({
40               container: "container",
41               width: <?php echo sfConfig::get('app_canvas_width');?>,
42               height: <?php echo sfConfig::get('app_canvas_height');?>
43         });
44
45     var layer = new Kinetic.Layer();
46
```

Figura 96. Captura de código: Declaraciones plano 2D

Luego (Figura 97) se declaran los diferentes controles para el plano en 3D y se toma el valor de las variables pasadas desde el controlador.

```
48     //declaraciones 3D
49     var container, scene, camara, renderer;// controls
50     var keyboard = new THREEx.KeyboardState();
51
52     posActualLogica = <?php echo count($puntos_navegacion)-1?>;
53     indiceInicioCamino = <?php echo count($puntos_navegacion)-1?>;
54     cantPuntos = <?php echo count($puntos_navegacion)-1?>;
55     puntosNavegacion = [];
56
57     <?php foreach($puntos_navegacion as $punto_navegacion):?>
58         var punto = new Object();
59         punto.id = <?php echo $punto_navegacion->getId();?>;
60         punto.x = <?php echo $punto_navegacion->getPuntoOrigenX();?>;
61         punto.y = <?php echo $punto_navegacion->getPuntoOrigenY();?>;
62         puntosNavegacion.push(punto);
63     <?php endforeach;?>
64
65     puntoNavegacionFisica = new Object();
66     puntoNavegacionFisica.id = puntosNavegacion[indiceInicioCamino].id;
67     puntoNavegacionFisica.x = puntosNavegacion[indiceInicioCamino].x;
68     puntoNavegacionFisica.y = puntosNavegacion[indiceInicioCamino].y;
69
```

Figura 97. Captura de código: Declaraciones plano 3D

La siguiente instrucción (Figura 99) ejecuta la función ***inicio()*** luego de cargar toda la página. dicha función se encarga de dibujar el plano de la facultad en dos dimensiones y de instanciar el framework de Three.js (función ***init()***))y generar la vista inicial de la cámara (función ***animate()***).

```
70      $(document).ready(inicio);
```

Figura 98. Captura de código: invocación a la función inicio()

```
73 function inicio(){
74     //inicio 2D
75     dibujarFacu(stage);
76     dibujarTodosLosPuntos(stage,layer);
77     stage.draw();
78
79     //inicio 3D
80     init();
81     animate();
82 }
```

Figura 99. Captura de código: función inicio()

La función dibujarFacu() (Figura 100) interactúa con variables provenientes del controlador y se encarga de construir el plano en dos dimensiones respetando la escala establecida en la aplicación.

```
85  //dibujo de facultad en 2D
86  function dibujarFacu(stage){
87     <?php foreach ($estructuras as $estructura):?>
88       <?php $puntos = PuntosTable::getPuntosBy($estructura->getId());?>
89       <?php if (count($puntos) > 0):?>
90         <?php $arreglo_de_puntos = array()?>
91         <?php for($i = 0; $i < count($puntos)-1; $i++):?>
92           <?php $arreglo_de_puntos[] = $puntos[$i]->getPuntoOrigenX()/($sf_user->getAttribute('escala'))+10;?>
93           <?php $arreglo_de_puntos[] = (sfConfig::get('app_maximo_y') - $puntos[$i]->getPuntoOrigenY())
94                                                 / $sf_user->getAttribute('escala');?>
95         <?php endfor;?>
96           <?php $arreglo_de_puntos[] = $puntos[count($puntos)-1]->getPuntoOrigenX()/$sf_user->getAttribute('escala')+10;?>
97           <?php $arreglo_de_puntos[] = (sfConfig::get('app_maximo_y') - $puntos[count($puntos)-1]->getPuntoOrigenY())
98                                                 / $sf_user->getAttribute('escala');?>
99             dibujarPoligono(stage
100                 , "<?php echo $estructura->getTipo();?>"
101                 , "<?php echo $estructura->getId();?>"
102                 , "<?php echo $destino;?>"
103                 , new Array(<?php echo implode(',',$arreglo_de_puntos);?>));
104       <?php endif;?>
105    <?php endforeach;?>}
```

Figura 100. Captura de código: función dibujarFacu()

Luego viene la función init() (Figura 101), la más compleja y larga, la misma se explicará en partes para simplificar su comprensión.

La primera parte de la función init() crea una escena y una cámara y le asigna a la cámara la posición actual del usuario.

```
108 function init()
109 {
110         // SCENE
111         scene = new THREE.Scene();
112         alturaCamara = 60;
113         // CAMERA
114         var SCREEN_WIDTH = window.innerWidth-36, SCREEN_HEIGHT = 500;
115         var VIEW_ANGLE = 35, ASPECT = SCREEN_WIDTH / SCREEN_HEIGHT, NEAR = 0.1, FAR = 20000;
116         camara = new THREE.PerspectiveCamera( VIEW_ANGLE, ASPECT, NEAR, FAR);
117         camara.position.set(puntosNavegacion[posActualLogica].x,alturaCamara,-puntosNavegacion[posActualLogica].y);
```

Figura 101. Captura de código: Encabezado función init()

La segunda parte, calcula el ángulo de rotación de la cámara y se la asigna al eje Y de la cámara. El ángulo de rotación de la cámara depende de la posición del punto siguiente del camino.

```
119         //calcula pendiente y angulo de rotacion para mirar al siguiente punto
120         var pendiente = (puntosNavegacion[posActualLogica-1].x - camara.position.x) / (-puntosNavegacion[posActualLogica-1].y - camara.position.z);
121         var angulo_rotacion = Math.atan(pendiente);
122
123         // si el punto de destino de z es mayor al actual, da media vuelta mas
124         if (-puntosNavegacion[posActualLogica-1].y>camara.position.z){
125             angulo_rotacion += Math.PI;
126         }
127
128         camara.rotation.y = angulo_rotacion;
129         scene.add(camara);
```

Figura 102. Captura de código: cálculo de pendiente y ángulo de rotación de pared.

Luego se declara el Renderer (Figura 103) mencionado en la sección de herramientas utilizadas.

```
131         // RENDERER
132         if ( ThreeJsDetector.webgl )
133                 renderer = new THREE.WebGLRenderer( {antialias:true} );
134         else
135                 renderer = new THREE.CanvasRenderer();
136         renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
137         renderer.setClearColorHex(0xfefcff, 1);
138         container = document.getElementById( 'ThreeJS' );
139         container.appendChild( renderer.domElement );
140
```

Figura 103. Captura de código: declaración del renderer

Se declara el piso del escenario y se agrega a la escena el mismo.

```
141         // FLOOR
142         var floorTexture = new THREE.ImageUtils.loadTexture('../../../../images/cemento_alisado.jpg');
143         floorTexture.wrapS = floorTexture.wrapT = THREE.RepeatWrapping;
144         floorTexture.repeat.set( 10, 10 );
145         var floorMaterial = new THREE.MeshBasicMaterial( { map: floorTexture, side: THREE.DoubleSide } );
146         var floorGeometry = new THREE.PlaneGeometry(10000, 10000, 10, 10);
147         var floor = new THREE.Mesh(floorGeometry, floorMaterial);
148         floor.position.y = -0.5;
149         floor.rotation.x = Math.PI / 2;
150         scene.add(floor);
151
```

Figura 104. Captura de código: declaración del piso del escenario

Se habilita la botonera de navegación virtual y se procede a construir las paredes del plano tridimensional.

```
151
152         habilitarBotonera();
153         //construccion de paredes
154         <?php
155
156         foreach ($paredes_dibujables as $pared):
157             $punto1 = $pared->getPunto1();
158             $punto2 = $pared->getPunto2();
159
160             //distancia = raiz[(y2 - y1)^2 + (x2 - x1)^2]
161             $distancia = sqrt(pow($punto2->getPuntoOrigenY() - $punto1->getPuntoOrigenY(),2) + pow($punto2->getPuntoOrigenX() - $punto1->getPuntoOrigenX(),2));
162
163             //punto medio = (x1 + x2) / 2 ; (y1 + y2) / 2
164             $punto_medio_x = ($punto2->getPuntoOrigenX() + $punto1->getPuntoOrigenX())/2;
165             $punto_medio_y = ($punto2->getPuntoOrigenY() + $punto1->getPuntoOrigenY())/2;
166
167             //pendiente  = m = (y1-y2) / (x1-x2)
168             //m = TAN(α)
169             //α= ATAN(m) , ATAN=arcotangente
170
171             if (($punto1->getPuntoOrigenX() - $punto2->getPuntoOrigenX()) != 0){
172                 $pendiente = ($punto1->getPuntoOrigenY()   $punto2->getPuntoOrigenY()) / ($punto1->getPuntoOrigenX()   $punto2->getPuntoOrigenX());
173                 $angulo_rotacion = atan($pendiente);
174             }else{
175                 $angulo_rotacion = deg2rad(90);
176             }
177             $orientacion = $pared->getOrientacionPared()->getNombre();
178             $link = $pared->getLinkImagen();
179
180             ?>
181
182             dibujarPared(<?php echo  $distancia ?>,<?php echo $punto_medio_x ?>,<?php echo -$punto_medio_y ?>,<?php echo $angulo_rotacion?>,'<?php echo
$orientacion?>','<?php echo $link?>');
183             <?php
184         endforeach;
185         ?>
```

Figura 105. Captura de código: Habilitación de botonera y construcción de paredes.

Finalmente, se activa la cámara pero no el lector QR y se deshabilita el botón que se encarga de recalcular el camino en caso de que el usuario se encuentre fuera del mismo.

```
194         activarCamara(false);
195         deshabilitarBotonRecalcular();
196
```

Figura 106. Captura de código: Activación de cámara y deshabilitación de botón recalcular.

Con esto queda finalizada la función init().

Las funciones animate() y render() (Figura 107) fueron descriptas anteriormente, la primera se encarga de gestionar las animaciones en tres dimensiones y render sitúa a la cámara dentro del escenario. Es importante destacar que la función TWEEN.update(), se ejecuta dentro del bucle de animate, chequeando continuamente que existan animaciones gestionadas por el motor de animaciones Tweenjs.

```
199 function animate()
200 {
201     requestAnimationFrame( animate );
202     render();
203     TWEEN.update();
204
205 }
206
207 function render()
208 {
209      renderer.render( scene, camara );
210 }
```

Figura 107. Captura de código: funciones anímate() y render().

Como se puede ver no todas las funciones que se utilizan dentro de este template están definidas dentro del mismo, muchas otras se encuentran en diferentes archivos con código javascript. Se mostrarán a continuación las más importantes.

La función ***activarCamara()*** (Figura 108) localizada en el archivo ***activar-camara.js***, se encarga de instanciar la cámara del dispositivo cliente y de acuerdo al parámetro que recibe determina si se enciende el lector o no, informando en pantalla en caso de existir alguna incompatibilidad con el navegador.

```
1  function activarCamara(encenderLector){
2
3
4      navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserM
   navigator.msGetUserMedia;
5
6      var cam_video_id = "camsource"
7
8      // Assign the <video> element to a variable
9      var video = document.getElementById(cam_video_id);
10     var options = {
11         "audio": false,
12         "video": true
13     };
14     // Replace the source of the video element with the stream from the camara
15     if (navigator.getUserMedia) {
16         navigator.getUserMedia(options, function(stream) {
17             video.src = (window.URL && window.URL.createObjectURL(stream)) || stream;
18         }, function(error) {
19             console.log(error)
20         });
21         // Below is the latest syntax. Using the old syntax for the time being for backwards compatibility.
22         // navigator.getUserMedia({video: true}, successCallback, errorCallback);
23     } else {
24         alert('Sorry, native web camara streaming (getUserMedia) is not supported by this browser...');
25         return;
26     }
27
28     $(document).ready(function() {
29         if (!navigator.getUserMedia) return;
30         cam = camera(cam_video_id);
31         $('#link_apagar_lector').addClass('ui-disabled');
32                 if (encenderLector){
33                         cam.start();
34                 }
35     })
36
37
38
39 }
```

Figura 108. Captura de código: función activarCamara()

El archivo ***kinetic-functions.js***, provee muchas funciones que se utilizan durante la construcción del plano bidimensional. Se explicarán las más importantes.

La primera se llama ***dibujarPolígono()*** y es invocada por la anteriormente mencionada ***dibujarFacu()***. La misma crea polígonos a partir de una serie de puntos.

```
1  //Función que dibuja los poligonos de la facultad (Estructuras) y agrega un link a aquellos que son navegables.
2  function dibujarPoligono(stage, tipoEstructura, idEstructura, idDestino, pointsToDraw){
3    var poly = new Kinetic.Polygon( { points: pointsToDraw, fill: '#75D6B5', stroke: "black", strokeWidth: 2 });
4
5    if (tipoEstructura == 1){
6      poly.on("touchstart click", function() {
7        window.location = _BASEPATH + "/" +
8                          _ENVIROMENT + "/main/detalleEstructura?idEstructura="+idEstructura+"&idDestino="+idDestino;
9      });
10
11     poly.on("touchstart click", function() {
12       var complexText = layer.get("#myText")[0];
13       layer.remove(complexText);
14     })
15
16     poly.setFill('#75D6B5');
17   }
18   else if (tipoEstructura == 2){
19     poly.setFill('#FFCC99');
20   }
21   else{
22     poly.setFill('#BCA0A0');
23   }
24
25   //add the poly to the layer
26   layer.add(poly);
27   // add the layer to the stage
28   stage.add(layer);
29 }
```

Figura 109. Captura de código: función dibujarPoligono()

La siguiente se llama ***dibujarPuntoNavegacion()*** (Figura 110), y genera un punto de navegación dentro del plano en dos dimensiones o una imagen (de una persona o de un ojo), de acuerdo a los parámetros que recibe. la misma es utilizada por la función ***dibujarTodosLosPuntos()*** (Figura 111), que se encarga de dibujar el camino del recorrido.

```
31 //Función que dibuja los puntos de navegación.
32 //Dibuja circulos si primero es falso y una imagen si es verdadero.
33 function dibujarPuntoNavegacion(stage,layer,x,y,id, primero,dibujoOjito){
34   // Si primero, se carga una imagen en vez de dibujar un punto
35   if(primero){
36     var imageObj = new Image();
37     imageObj.onload = function() {
38       var imageUser = new Kinetic.Image({
39         x: x-8, y:
40         y-8, image: imageObj,
41         width: 16,
42         height: 16 });
43         layer.add(imageUser);
44         stage.add(layer);
45     }
46
47     if (dibujoOjito){
48                 imageObj.src = _BASEPATH + "/images/eye.png";
49         }else{
50                 imageObj.src = _BASEPATH + "/images/people1.png";
51         }
52    }//Si no es el punto inicial dibujamos un punto de navegacion simple
53    else{
54      var circle = new Kinetic.Circle({
55        x: x,
56        y: y,
57        radius: 2,
58        fill: "#1C777B",
59        stroke: "black",
60        strokeWidth: 4 });
61
62      layer.add(circle);
63      stage.add(layer);
64    }
65 }
```

Figura 110. Captura de código: función dibujarPuntoNavegacion()

```
117 function dibujarTodosLosPuntos(stage,layer){
118
119     var aux = new Array();
120     for (var i=0;i<puntosNavegacion.length;i++)
121     {
122         if (i<=indiceInicioCamino){
123             var id = null;
124             var x = puntosNavegacion[indiceInicioCamino-i].x/escala + 10;
125             var y = (app_maximo_y - puntosNavegacion[indiceInicioCamino-i].y)/escala;
126             aux.push(x);
127             aux.push(y);
128             if (i == puntosNavegacion.length - posActualLogica - 1){
129                 dibujarPuntoNavegacion(stage,layer,x,y,id,true,true);
130             }else{
131                 dibujarPuntoNavegacion(stage,layer,x,y,id,false,true);
132             }
133         }
134     }
135     dibujarLinea(stage, layer, aux);
136     dibujarPuntoNavegacion(stage,layer,puntoNavegacionFisica.x/escala + 10,(app_maximo_y - puntoNavegacionFisica.y)/escala,puntoNavegacionFisica.id,
true,false);
137 }
138
```

dibujarTodosLosPuntos() a su vez utiliza a ***dibujarLinea()*** (Figura 112), la cual se encarga de dibujar las líneas punteadas entre los distintos puntos de navegación.

```
67 //Función que dibuja lineas entre los puntos de navegación.
68 function dibujarLinea(stage, layer, pointsToDraw){
69   var lineaCamino = new Kinetic.Line({
70           points: pointsToDraw,
71           stroke: "green",
72           strokeWidth: 2,
73           lineJoin: "round",
74           dashArray: [5, 2]
75         });
76   layer.add(lineaCamino);
77   stage.add(layer);
78 }
```

Figura 112. Captura de código: función dibujarLinea()

Por otra parte, el archivo ***threejs-functions.js***, provee otras funciones que se utilizan durante la construcción del plano tridimensional. Las más importantes se detallan en el siguiente párrafo.

dibujarPared() (Figura 113), crea una pared en el plano de tres dimensiones con los datos pasados por parámetro, distancia, punto medio, ángulo de rotación, orientación de la imagen que utiliza como textura y link de la imagen.

```
function dibujarPared(distancia,puntoMedioX,puntoMedioY,anguloRotacion,orientacion,link) {

    var border = 0;
    var elevation = 150;

    // se crea la pared
    var materialArray = [];
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));

    if (orientacion == 'Norte/Oeste'){
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('../../../../images/imagenes-navegador/'+link)
}));
    }else{
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('../../../../images/imagenes-navegador/'+link)
}));
        materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture('') }));
    }
    var MovingCubeMat = new THREE.MeshFaceMaterial(materialArray);
    var MovingCubeGeom = new THREE.CubeGeometry(distancia, elevation, border, 1, 1, 1, materialArray );

    pared = new THREE.Mesh( MovingCubeGeom, MovingCubeMat );
    pared.position.set(puntoMedioX, 75, puntoMedioY);
    pared.rotation.y = anguloRotacion;
    scene.add( pared );
}
```

Figura 113. Captura de código: función dibujarPared()

irAPunto() *(Figura 114), gestiona la animación para mover virtualmente la cámara de un punto hacia el otro. Es utilizada por* **getPuntoSiguiente()** *y* **getPuntoAnterior().**

```
function irAPunto(destinoX,destinoY,destinoZ,cam) {

    //calcula pendiente y angulo de rotacion
    var pendiente = (cam.position.x - destinoX) / (cam.position.z - destinoZ);
    var angulo_rotacion = Math.atan(pendiente);

    angulo_rotacion = normalizarAnguloRotacion(cam,destinoZ,angulo_rotacion);

    var tiempo = tiempoRotacion(cam,angulo_rotacion);

    //rotacion
    var animacionRotacion = new TWEEN.Tween(cam.rotation).to({
         x: 0,
         y: angulo_rotacion,
         z: 0,},tiempo);
     animacionRotacion.easing(TWEEN.Easing.Linear.None).onUpdate(function () {
         //mientras rota no hace nada
     });
     animacionRotacion.onComplete(function () {
         //termina de rotar y avanza


        tiempo = tiempoAvanceRetroceso(cam,destinoX,destinoZ);
        var animacionAvance = new TWEEN.Tween(cam.position).to({
              x: destinoX,
              y: destinoY,
              z: destinoZ
         },tiempo);
        animacionAvance.easing(TWEEN.Easing.Linear.None).onUpdate(function () {
           //mientras avanza no hace nada
         });
        animacionAvance.onComplete(function () {
           //termina de avanzar y habilita la botonera
           habilitarBotonera();
        });
        //comienza a avanzar
        animacionAvance.start();



     });
     //comienza a rotar
     animacionRotacion.start();
}
```

Figura 114. Captura de código: función irAPunto()

getPuntoSiguiente() y ***getPuntoAnterior()*** (Figura 115), gestionan las animaciones para mover la cámara de un punto al otro pero a su vez actualiza el plano bidimensional y gestiona la botonera de navegación.

```
74  function getPuntoSiguiente()
75  {
76      deshabilitarBotonera();
77      posActualLogica--;
78      irAPunto(puntosNavegacion[posActualLogica].x,alturaCamara,-puntosNavegacion[posActualLogica].y,camara);
79      borrarTodosLosPuntos(stage, layer);
80      dibujarTodosLosPuntos(stage,layer);
81  }
82
83  function getPuntoAnterior()
84  {
85      deshabilitarBotonera();
86      posActualLogica++;
87      irAPunto(puntosNavegacion[posActualLogica].x,alturaCamara,-puntosNavegacion[posActualLogica].y,camara);
88      borrarTodosLosPuntos(stage, layer);
89      dibujarTodosLosPuntos(stage,layer);
90   }
91
```

Figura 115. Captura de código: funciones getPuntoSiguiente() y getPuntoAnterior()

rotar360() (Figura 116), gestiona la animación para rotar virtualmente la cámara en un punto 360º para la derecha o la izquierda según el ángulo de rotación recibido como parámetro.

```
92  function rotar360(angulo_rotacion,cam){
93       //calcula pendiente y angulo de rotacion
94      deshabilitarBotonera();
95      //rotacion
96      var animacionRotacion = new TWEEN.Tween(cam.rotation).to({
97           x: 0,
98           y: cam.rotation.y + angulo_rotacion,
99           z: 0,},5000);
100      animacionRotacion.easing(TWEEN.Easing.Linear.None).onUpdate(function () {
101          //mientras rota no hace nada
102      });
103      animacionRotacion.onComplete(function () {
104          //termina de rotar y normaliza el angulo
105          cam.rotation.y -= angulo_rotacion;
106          habilitarBotonera();
107
108      });
109
110      //comienza a rotar
111      animacionRotacion.start();
112  }
```

Figura 116. Captura de código: función rotar360()

Las siguientes funciones habilitan o deshabilitan los botones de la botonera (Figuras 117 a 120), son utilizados entre otras funciones por ***chequearBotonera()***, ***deshabilitarBotonera(), habilitarBotonera(),*** estos últimas utilizadas para bloquear o desbloquear la botonera durante la ejecución de la animación de la cámara.

```
114 function deshabilitarBotonRetroceder(){
115     $('#link_retroceder').addClass('ui-disabled');
116 }
117
118 function deshabilitarBotonAvanzar(){
119     $('#link_avanzar').addClass('ui-disabled');
120
121 }
```

Figura 117. Captura de código: funciones deshabilitarBotonRetrocerder() y deshabilitarBotonAvanzar()

```
123 function deshabilitarBotones360(){
124     $('#link_360_derecha').addClass('ui-disabled');
125     $('#link_360_izquierda').addClass('ui-disabled');
126 }
127
128 function habilitarBotones360(){
129     $('#link_360_derecha').removeClass('ui-disabled');
130     $('#link_360_izquierda').removeClass('ui-disabled');
131
132 }
```

Figura 118. Captura de código: funciones deshabilitarBotones360() y habilitarBotones360()

```
133
134 function habilitarBotonRetroceder(){
135     $('#link_retroceder').removeClass('ui-disabled');
136 }
137
138 function habilitarBotonAvanzar(){
139     $('#link_avanzar').removeClass('ui-disabled');
140 }
```

Figura 119. Captura de código: funciones habilitarBotonRetrocerder() y habilitarBotonAvanzar()

```
151 function chequearBotonera(){
152     habilitarBotonAvanzar();
153     habilitarBotonRetroceder();
154     if (posActualLogica == 0){
155         deshabilitarBotonAvanzar();
156     }else{
157         if (posActualLogica == cantPuntos){
158             deshabilitarBotonRetroceder();
159         }
160     }
161
162 }
163
164 function deshabilitarBotonera(){
165     deshabilitarBotonAvanzar();
166     deshabilitarBotonRetroceder();
167     deshabilitarBotones360();
168     deshabilitarBotonPosActual();
169 }
170 function habilitarBotonera(){
171     chequearBotonera();
172     habilitarBotones360();
173     habilitarBotonPosActual();
174 }
175
```

Figura 120. Captura de código: funciones chequearBotonera(), deshabilitarBotonera() y habilitarBotonera()

Las funciones mencionadas a continuación (Figura 121) calculan los tiempos que deben tardar las animaciones para que todas se muevan a la misma velocidad. ***tiempoRotacion()*** calcula el tiempo que debe tardar la cámara en girar hacia un ángulo pasado por parámetro con respecto a su ángulo actual. ***tiempoAvanceRetroceso(),*** determina el tiempo que debe tardar en moverse la cámara desde su posición actual hasta la posición destino.

```
201 function tiempoRotacion(cam,angulo_rotacion){
202
203     return Math.abs(cam.rotation.y - angulo_rotacion)*1000;
204 }
205
206 function tiempoAvanceRetroceso(cam,destinoX,destinoZ){
207     //distancia = raiz[(y2 - y1)^2 + (x2 - x1)^2]
208
209     return Math.sqrt(Math.pow(cam.position.x - destinoX,2) + Math.pow(cam.position.z - destinoZ,2))*3;
210
211 }
```

Figura 121. Captura de código: funciones tiempoRotacion() y tiempoAvanceRetroceso()

Las funciones ***encenderLector()*** y ***apagarLector()*** (Figura 122) encienden y apagan el lector de códigos QR.

```
213 function encenderLector(){
214     cam.start();
215     $('#link_encender_lector').addClass('ui-disabled');
216     $('#link_apagar_lector').removeClass('ui-disabled');
217 }
218
219 function apagarLector(){
220     cam.stop();
221     $('#link_apagar_lector').addClass('ui-disabled');
222     $('#link_encender_lector').removeClass('ui-disabled');
223 }
224
```

Figura 122. Captura de código: funciones encenderLector() y apagarLector()

actualizarPosicionFisica(), actualiza en el plano de dos dimensiones el punto con la posición física determinada por la lectura del código QR por parte del usuario. ***chequearNuevaUbicacionFisica(),*** chequea si el código leído está fuera del camino pautado, si es así se habilita el botón "recalcular camino", si está adentro y además es el último, se muestra el mensaje de llegada al destino. (Figura 123)

```
225 function actualizarPosicionFisica(codigoQR){
226
227     var arregloQR = codigoQR.split(",");
228     puntoNavegacionFisica.id = arregloQR[0];
229     puntoNavegacionFisica.x = arregloQR[1];
230     puntoNavegacionFisica.y = arregloQR[2];
231     borrarTodosLosPuntos(stage, layer);
232     dibujarTodosLosPuntos(stage, layer);
233 }
234
235 function chequearNuevaUbicacionFisica(codigoQr){
236
237     //chequea si el punto leido está en el camino
238     var arregloQr = codigoQr.split(",");
239     var estaEnElCamino = false;
240     var esElUltimo = false;
241     for (var i=0;i<puntosNavegacion.length;i++){
242         if (puntosNavegacion[i].id == arregloQr[0]){
243             estaEnElCamino = true;
244             if (i == 0){
245                 esElUltimo = true;
246             }
247         }
248     }
249
250     //el punto leído está en el camino
251     if (estaEnElCamino){
252         deshabilitarBotonRecalcular();
253         if (esElUltimo){
254           $("#popupDialog").popup("open",100,300);
255         }
256     //el punto leido no esta en el camino
257     }else{
258         habilitarBotonRecalcular();
259     }
260 }
```

Figura 123. Captura de código: funciones actualizarPosicionFisica() y chequearNuevaUbicacionFisica()

habilitarBotonRecalcular() y ***deshabilitarBotonRecalcular()***, habilitan y deshabilitan el botón de recalcular camino respectivamente. ***recalcularCamino()*** redirecciona la aplicación a la acción ***buscar*** pasando como parámetro la posición física actual como origen permitiendo seleccionar el mismo destino o uno nuevo y generar una nueva navegación. (Figura 124)

```
262 function habilitarBotonRecalcular(){
263     $('#link_recalcular_camino').removeClass('ui-disabled');
264 }
265
266 function deshabilitarBotonRecalcular(){
267     $('#link_recalcular_camino').addClass('ui-disabled');
268 }
269
270 function recalcularCamino(){
271
272     if (_BASEPATH.split(':').length>0){
273         location.href = _BASEPATH +'/index.php/main/buscar?id_punto_navegacion_origen=' + puntoNavegacionFisica.id;
274     }else{
275         location.href = '/3dindoornavigation/web/index.php/main/buscar?id_punto_navegacion_origen=' + puntoNavegacionFisica.id;
276     }
277 }
```

Figura 124. Captura de código: funciones habilitarBotoneraRecalcular(), deshabilitarBotoneraRecalcular() y recalcularCamino()

A lo largo de los anexos se ha detallado la implementación del modelo, el controlador y la vista, detallando los métodos y funciones más importantes que determinan la estructura y funcionamiento de la aplicación desarrollada.