

3D HOME

Una Interfaz para el Control de un Hogar Inteligente

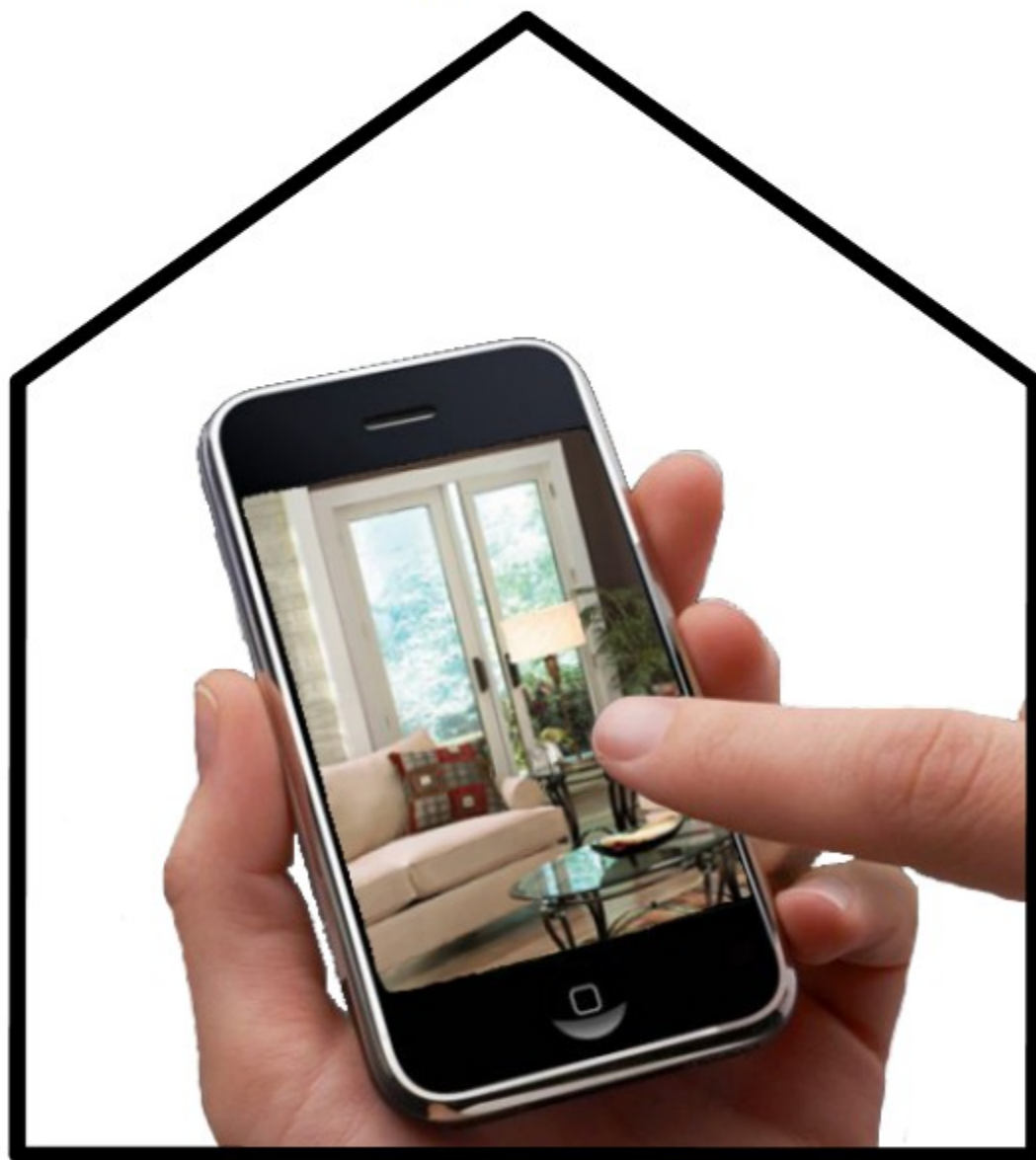
Autor:

Antonio Muñoz Rueda

Directores:

**Vicente Pelechano Ferragud
Pablo Muñoz Julve**

29 de Septiembre de 2011



**UNIVERSIDAD
POLITECNICA
DE VALENCIA**

DSIC
DEPARTAMENT DE SISTEMES
INFORMÀTICS I COMPUTACIÓ

Resumen

Actualmente, los dispositivos móviles en los que se pueden instalar aplicaciones son muy populares. Estas aplicaciones, que reciben el nombre de aplicaciones móviles, tienen una gran aceptación entre los usuarios y cubren una gran diversidad de actividades. Una de estas actividades es el control de los hogares inteligentes donde el usuario controla o automatiza varios aspectos de su hogar como los electrodomésticos o la luz. Para controlar este tipo de viviendas, el usuario interactúa con su hogar a través de una interfaz que puede ser ejecutada desde un dispositivo móvil. Ya existen algunas de estas interfaces pero obligan al usuario a memorizar órdenes o a buscar la opción deseada a través de menús.

En este trabajo se propone una aplicación móvil que ofrece una interfaz 3D con una alta calidad gráfica que trata de evitar estos inconvenientes. En la aplicación propuesta, el usuario controla a un personaje en primera persona dentro de una representación virtual fiel del hogar a controlar. Las habitaciones, muebles y electrodomésticos que hay en la aplicación son representaciones virtuales de los que hay en la vivienda real. Además, estos elementos de la aplicación están distribuidos de la misma forma que en el hogar real. Así, si el usuario desea realizar alguna acción sobre la vivienda controlada, solo tiene que hacer en la aplicación lo que haría en la vida real como ir al salón a encender una luz. Con esto se persigue un aprendizaje rápido y una interacción natural.

Para crear la aplicación se ha usado Unreal Development Kit Mobile, que es un motor de escenarios 3D interactivos gratuito que permite crear aplicaciones con una alta calidad gráfica para los dispositivos móviles iPod Touch, iPhone y iPad.

Aunque el objetivo de la aplicación desarrollada en este trabajo es controlar un hogar inteligente, esta aplicación se conecta a una maqueta que representa una vivienda inteligente. Esta maqueta tiene implementados algunos dispositivos que están presentes en los hogares inteligentes como luces conmutables y graduales, persianas, alarmas y ventiladores. Con estas pruebas se asegura el correcto funcionamiento de la aplicación en un hogar inteligente.

Índice de Contenido

<u>CAPÍTULO 1: INTRODUCCIÓN.....</u>	<u>11</u>
<u>1.1. Motivación.....</u>	<u>12</u>
<u>1.2. Solución Propuesta.....</u>	<u>12</u>
<u>1.3. Organización del Resto del Documento.....</u>	<u>14</u>
<u>CAPÍTULO 2: CONTEXTO TECNOLÓGICO.....</u>	<u>17</u>
<u>2.1. Domótica.....</u>	<u>17</u>
<u>2.1.1. Componentes de un Sistema Domótico.....</u>	<u>18</u>
<u>2.1.2. Arquitectura de un Sistema Domótico.....</u>	<u>19</u>
<u>2.2. Motores de Escenarios 3D Interactivos.....</u>	<u>21</u>
<u>2.2.1. Unreal Engine 3.....</u>	<u>22</u>
<u>2.2.2. Torque.....</u>	<u>24</u>
<u>2.2.3. Gamebryo.....</u>	<u>26</u>
<u>2.2.4. Unity.....</u>	<u>27</u>
<u>2.2.5. Discusión.....</u>	<u>28</u>
<u>2.3. Conclusiones.....</u>	<u>30</u>
<u>CAPÍTULO 3: ESTADO DEL ARTE.....</u>	<u>32</u>
<u>3.1. Interfaces WIMP.....</u>	<u>32</u>
<u>3.1.1. Ejemplo de uso.....</u>	<u>32</u>
<u>3.2. Reconocimiento de Voz.....</u>	<u>34</u>
<u>3.2.1. Ejemplos de Uso</u>	<u>36</u>
<u>3.3. Reconocimiento de Gestos.....</u>	<u>37</u>
<u>3.3.1. Ejemplo de uso.....</u>	<u>39</u>
<u>3.4. Interacción Sensible al Contexto.....</u>	<u>40</u>
<u>3.4.1. Ejemplo de Uso.....</u>	<u>41</u>
<u>3.5. Interacción Multimodal.....</u>	<u>42</u>
<u>3.5.1. Ejemplo de Uso.....</u>	<u>42</u>
<u>3.6. Conclusiones.....</u>	<u>44</u>
<u>CAPÍTULO 4: PROCESO DE DESARROLLO DE 3D HOME.....</u>	<u>47</u>
<u>4.1. Visión General del Proceso de Desarrollo en UDK Mobile.....</u>	<u>47</u>
<u>4.1.1. Fase de Análisis.....</u>	<u>48</u>
<u>4.1.2. Fase de Implementación.....</u>	<u>49</u>
<u>4.1.3. Fase de Despliegue.....</u>	<u>50</u>
<u>4.2. Etapas del Desarrollo de 3D Home.....</u>	<u>50</u>
<u>4.2.1. Análisis.....</u>	<u>50</u>

4.2.1.1. Definición del Plano.....	50
4.2.1.2. Definición de los elementos de interacción.....	51
4.2.2. Implementación.....	52
4.2.2.1. Creación de la Estructura.....	53
4.2.2.2. Aplicación de Materiales.....	53
4.2.2.3. Colocación de Elementos.....	54
4.2.2.4. Implementación del Comportamiento.....	56
4.2.2.5. Desarrollo de otras funcionalidades.....	59
4.2.3. Despliegue.....	62
4.3. Evolución de 3D Home.....	63
4.4. Flexibilidad de 3D Home.....	65
4.5. Conclusiones.....	69
CAPÍTULO 5: CASO DE ESTUDIO.....	71
5.1. Visión Global.....	71
5.2. Visión del Usuario.....	72
CAPÍTULO 6: CONCLUSIONES.....	91
6.1. Problemas Encontrados.....	91
6.2. Trabajos Futuros.....	92
ANEXO A: UNREAL DEVELOPMENT KIT MOBILE.....	94
ANEXO B: UNREALED.....	96
ANEXO C: UNREAL KISMET.....	99
ANEXO D: UNREAL MATINEE.....	115
ANEXO E: UNREALSCRIPT.....	118

Listado de Figuras

Figura 1. Esquema conceptual de un hogar inteligente.....	11
Figura 2. Salón de la aplicación propuesta.....	13
Figura 3. Maqueta de un hogar inteligente.....	14
Figura 4. Esquema de funcionamiento de un sistema domótico.....	19
Figura 5. Esquema de arquitectura centralizada.....	20
Figura 6. Esquema de arquitectura descentralizada.....	20
Figura 7. Esquema de arquitectura distribuida.....	20
Figura 8. Esquema de arquitectura híbrida/mixta.....	21
Figura 9. Ventajas más importantes de los motores de escenarios 3D interactivos.....	22
Figura 10. Desventajas más importantes de los motores de escenarios 3D interactivos.....	22
Figura 11. Gears of War.....	23
Figura 12. UnrealEd.....	24
Figura 13. Penny Arcade Adventures.....	24
Figura 14. World Editor.....	25
Figura 15. Fallout 3.....	26
Figura 16. World Builder (izq) y previsualizador de tiempo real (der).....	27
Figura 17. Tiger Woods PGA Tour Online.....	27
Figura 18. Unity Editor.....	28
Figura 19. Listado de viviendas inteligentes.....	33
Figura 20. Menú de una vivienda.....	33
Figura 21. Interfaces para cambiar el estado de los dispositivos.....	34
Figura 22. Efecto del tamaño del vocabulario de SNR en el reconocimiento de voz.....	35
Figura 23. Ejemplos de dispositivos hápticos.....	37
Figura 24. Esquema del sistema de interpretación de gestos basado en la visión.....	38
Figura 25. Entorno de un hogar inteligente.....	40
Figura 26. Gestos para el control de cortinas y luces.....	40
Figura 27. Vista de SmartHOME Research Lab y de los prototipos de interfaces gráficas de hogar inteligente en dispositivos móviles.....	43
Figura 28. Software de interfaz gráfica 3D.....	44
Figura 29. Proceso de creación de una vivienda en UDK Mobile.....	48
Figura 30. Plano de una casa.....	48
Figura 31. Comportamiento de la luz del salón.....	49
Figura 32. Interfaz de la luz del salón.....	49
Figura 33. Fase de implementación.....	49
Figura 34. Plano de la vivienda.....	51
Figura 35. Geometría del pasillo (izq.) y del salón (der.).....	53
Figura 36. Salón con las texturas aplicadas.....	54
Figura 37. Salón con los objetos añadidos.....	55
Figura 38. Modelo Kismet de la luz del salón.....	56
Figura 39. Objeto de secuencia Matinee detallado de Livingroom Toggleable Light.....	58
Figura 40. Objeto de secuencia Matinee de Livingroom Toggleable Light.....	58
Figura 41. Diagrama de clases de la comunicación con la maqueta.....	59
Figura 42. Ejemplo de comunicación con la maqueta.....	60
Figura 43. Clase Estado3DH.....	60
Figura 44. Clase ThreeDHome.....	61
Figura 45. Clase KismetGrabar.....	61
Figura 46. Categoría de Save Game.....	61
Figura 47. Objeto de secuencia Save Game.....	61

Figura 48. Clase KismetCargar.....	61
Figura 49. Fase de despliegue.....	62
Figura 50. Unreal Frontend.....	62
Figura 51. Salón cubierto con InterpActors.....	64
Figura 52. Luz gradual.....	65
Figura 53. Transformador de modelos.....	66
Figura 54. Ejemplo de modelo Autocad.....	67
Figura 55. Proceso de transformación de modelos.....	68
Figura 56. Proceso de reconstrucción del escenario.....	69
Figura 57. Diagrama de flujo del control de la maqueta.....	72
Figura 58. Diagrama de estados general de 3D Home.....	72
Figura 59. Pantalla de título.....	73
Figura 60. Estado por defecto de 3D Home.....	73
Figura 61. Baño.....	74
Figura 62. Dormitorio grande.....	75
Figura 63. Dormitorio pequeño de una cama.....	75
Figura 64. Dormitorio pequeño de dos camas.....	76
Figura 65. Aseo.....	76
Figura 66. Cocina.....	77
Figura 67. Salón.....	77
Figura 68. Recibidor de la vivienda.....	78
Figura 69. Posición de los elementos de interacción.....	79
Figura 70. Interfaz de la luz conmutable del salón.....	80
Figura 71. Diagrama de estados de la luz del salón.....	80
Figura 72. Ejemplo de interacción con la luz del salón.....	81
Figura 73. Interfaz del aire acondicionado.....	81
Figura 74. Diagrama de estados del aire acondicionado.....	82
Figura 75. Ejemplo de encendido del aire acondicionado.....	82
Figura 76. Ejemplo de apagado del aire acondicionado.....	83
Figura 77. Interfaz de la persiana.....	83
Figura 78. Diagrama de estados de la persiana.....	84
Figura 79. Ejemplo de interacción con la persiana.....	84
Figura 80. Interfaz para activar la alarma.....	85
Figura 81. Interfaz para apagar la alarma.....	86
Figura 82. Diagrama de estados de la alarma.....	86
Figura 83. Ejemplo de interacción con la alarma.....	87
Figura 84. Habitación donde está la luz gradual.....	87
Figura 85. Interfaz de selección de intensidad.....	88
Figura 86. Explicación gráfica del nuevo tipo de transición.....	88
Figura 87. Diagrama de estados de la luz gradual.....	88
Figura 88. Ejemplo de interacción con la luz gradual.....	89
Figura 89. Ejemplo de material con textura difusa (izq.) y con textura difusa y mapa normal (der.).....	95
Figura 90. Interfaz de Unreal Level Editor.....	96
Figura 91. Pinceles BSP y vista del pincel antes (izq.) y después (der.) de añadir la geometría al escenario.....	97
Figura 92. Luz (izq.) y posición inicial del jugador (der.) añadidas al escenario.....	97
Figura 93. Materiales aplicados (izq.) y malla estática añadida (der.) al escenario.....	98
Figura 94. Escenario compilado.....	98
Figura 95. Ejemplo de propiedades editables de un objeto de secuencia.....	100
Figura 96. Aspecto de Actor Factory por defecto (izq.) y con las propiedades no usadas escondidas	

(der.).....	101
Figura 97. Ejemplo de objeto comentado.....	101
Figura 98. Ejemplo de uso de los enlaces in y out.....	102
Figura 99. Ejemplo de Actor Factory.....	102
Figura 100. Ejemplo de Add Input Zone.....	103
Figura 101. Ejemplo de Add Int.....	104
Figura 102. Ejemplo de Delay.....	104
Figura 103. Ejemplo de Draw Text.....	105
Figura 104. Ejemplo de Play Sound.....	106
Figura 105. Ejemplo de Remove Input Zone.....	106
Figura 106. Ejemplo de Set Int	107
Figura 107. Ejemplo de Switch.....	107
Figura 108. Ejemplo de una secuencia Matinee.....	108
Figura 109. Ejemplo de una variable Int.....	109
Figura 110. Ejemplo de una variable Matinee Data	109
Figura 111. Ejemplo de una Named Variable sin emparejar (izq.) y emparejada (der.).....	109
Figura 112. Ejemplo de una variable Object.....	109
Figura 113. Ejemplo de una variable Player.....	110
Figura 114. Ejemplo de Level Loaded.....	110
Figura 115. Ejemplo de Mobile Button Access.....	110
Figura 116. Ejemplo de Mobile Object Picker.....	112
Figura 117. Ejemplo de Comment.....	112
Figura 118. Ejemplo de Load Game.....	113
Figura 119. Ejemplo de Save Game.....	113
Figura 120. Interfaz de Unreal Matinee.....	114
Figura 121. Interfaz del editor de curvas.....	115
Figura 122. Ejemplo de pista.....	115
Figura 123. Ejemplo de gráfico.....	115
Figura 124. Panel de línea temporal.....	116
Figura 125. Ejemplo de línea temporal.....	116
Figura 126. Arquitectura de UnrealScript.....	118

Listado de Tablas

Tabla 1. Plataformas objetivo.....	29
Tabla 2. Detalles del uso del lenguaje de programación.....	29
Tabla 3. Disponibilidad de versión gratuita y uso en campos ajenos a los juegos.....	29
Tabla 4. Elementos de interacción de 3D Home.....	52
Tabla 5. Botones de la tubería de trabajo.....	63
Tabla 6. Tipos de objetos de secuencia.....	99
Tabla 7. Colores de variables y su tipo asociado.....	100
Tabla 8. Ejemplos de operadores de UnrealScript.....	121

Capítulo 1

Introducción

Los hogares tienen sistemas y equipos que mejoran la calidad de vida y aumentan el confort de sus residentes. Algunos de ellos, como el agua y la electricidad son básicos y son habituales en las viviendas. Otros, pueden no ser tan comunes pero también ofrecen otros servicios. Ejemplos de estos últimos sistemas son la televisión que proporciona entretenimiento, el frigorífico que facilita la preservación de los alimentos y el aire acondicionado que regula la temperatura del hogar. Con el objetivo de aumentar el confort de los residentes del hogar, surgen los hogares inteligentes que integran todos estos equipos y sistemas para facilitar la gestión y mantenimiento del hogar.

Los productos y sistemas relacionados con el hogar inteligente pueden ser agrupados en las áreas mostradas en la figura 1. Estas áreas son [6]:

- Domótica: es la automatización y control de los sistemas del hogar de forma local y remota.
- Seguridad: son sistemas y funciones para dispositivos de seguridad como alarmas de intrusión, cámaras de vigilancia y alarmas de incendio.
- Multimedia: son los contenidos de información y entretenimiento que están relacionados con la captura, tratamiento y distribución de imágenes y sonido dentro y fuera de la vivienda.
- Comunicación o datos: es la distribución de ficheros de texto, imágenes y sonidos, el acceso a Internet, la red de telefonía, etc.

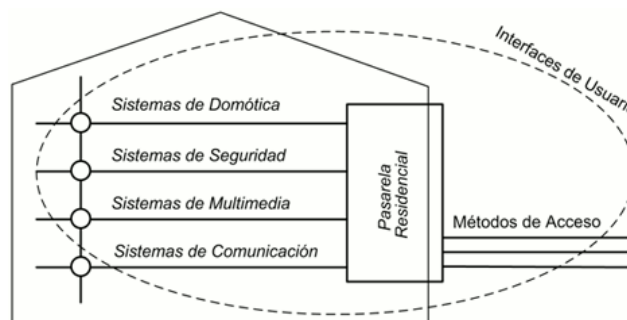


Figura 1. Esquema conceptual de un hogar inteligente

Un sistema domótico controla los dispositivos del hogar que están conectados a él. Por lo que cuando se habla de control, los términos "sistema domótico" y "hogar inteligente" se usan indistintamente. Sin embargo, no se deben confundir estos conceptos. Como se ve en la figura 1, un sistema domótico no es un hogar inteligente sino que forma parte de él.

Los hogares inteligentes deben dar la posibilidad de ser controlados por el usuario de alguna forma. Por esta razón, un sistema domótico debe proporcionar un mecanismo de control. Normalmente, estos mecanismos son interfaces gráficas donde, por ejemplo, se muestra la temperatura actual de la vivienda y el usuario puede, si lo desea, encender el aire acondicionado usando la misma interfaz.

Los teléfonos móviles que permiten la instalación de aplicaciones, también llamados *smartphones*, son usados por una gran cantidad de usuarios. En el año 2010 se vendieron 297 millones de smartphones en todo el mundo y se estima que en el 2011 se vendan 468 millones, aumentando las ventas un 57.7% respecto al 2010 [19]. Esta infraestructura de dispositivos móviles está atrayendo la atención de los desarrolladores de software para vender sus productos.

Dentro del ámbito de la aplicación que se presenta en este trabajo, existen algunas interfaces que permiten manejar el hogar inteligente de forma remota desde dispositivos móviles. Esta tesina propone una aplicación para controlar, desde un dispositivo móvil, una maqueta que representa un hogar inteligente. En esta aplicación, el usuario controla un hogar inteligente en primera persona al verse ubicado dentro de una representación virtual 3D de su hogar.

El resto de este capítulo está organizado de la siguiente forma: la sección 1.1 describe la motivación de esta tesina y los problemas que pretende resolver. La sección 1.2 describe brevemente la solución propuesta para resolver dichos problemas. Por último, en la sección 1.3 se estructura el contenido del resto del documento.

1.1. Motivación

Las capacidades de comunicación de los hogares y la posibilidad de acceder a sistemas de información desde cualquier localización son las bases que motivan la realización de este trabajo.

Por una parte, muchas personas usan *smartphones* en su vida diaria para distintas tareas como llamar por teléfono, escuchar música y leer documentos. Los *smartphones* permiten instalar aplicaciones que son desarrolladas por el fabricante del dispositivo, por el operador y por terceros. Esto ofrece un mercado donde los desarrolladores crean aplicaciones para todo tipo de actividades como música, juegos, redes sociales y educación. Esto provoca que los smartphones puedan integrarse con casi cualquier tipo de tarea que realicen las personas.

Por otra parte, las viviendas poseen medios de comunicación como Internet que pueden ser usados para comunicar el sistema domótico con una aplicación móvil. Así, un hogar inteligente puede ser controlado usando un dispositivo móvil. Esto aumenta la seguridad y el confort del usuario porque, desde cualquier lugar, puede ver el estado de la casa y controlar, si lo desea, su vivienda. Por ejemplo, podría comprobar desde su trabajo si se ha dejado alguna luz encendida de su casa y, de ser así, apagarla.

Se han desarrollado aplicaciones móviles cuyo fin es controlar un hogar inteligente con una interfaz gráfica o con la voz. Con estas aplicaciones, el usuario puede monitorizar y controlar desde cualquier lugar donde se tenga acceso a Internet un hogar inteligente. Sin embargo, si el usuario desea realizar alguna acción en su hogar, debe buscar la opción deseada en una serie de menús o acordarse de la orden de voz adecuada. Además, estas aplicaciones muestran una representación virtual simplificada del hogar inteligente. En esta tesina se pretende que, para controlar una vivienda inteligente, el usuario no tenga que memorizar órdenes de voz ni buscar entre varios listados de opciones cuál es la acción deseada. Además, trata de mostrar de forma fidedigna el hogar inteligente controlado.

1.2. Solución Propuesta

El objetivo de la tesina consiste en proporcionar una aplicación móvil para controlar de forma remota diversos servicios domóticos. El usuario controla estos servicios a través de un personaje virtual que está en una representación virtual de su casa. Las características de esta aplicación hace

que sea similar a un videojuego. Esta similitud causa que se crea conveniente usar las aproximaciones que se usan al desarrollar videojuegos para crear la aplicación. En la figura 2 se muestra una imagen de la aplicación propuesta.



Figura 2. Salón de la aplicación propuesta

Para ofrecer una aplicación que permita la representación 3D de un hogar, es necesario utilizar un motor gráfico o un motor de escenarios 3D interactivos. Actualmente, en el desarrollo de videojuegos, los motores de escenarios 3D interactivos son muy usados porque abstraen detalles de implementación. Estos motores evitan que el desarrollador se preocupe de cómo implementar, por ejemplo, el sistema de iluminación, el motor de físicas o la inteligencia artificial porque ya viene implementado. El desarrollador solo debe usar lo que ya está implementado, permitiéndole así centrarse en el diseño del videojuego.

En particular, el desarrollo de la aplicación propuesta se ha realizado con Unreal Development Kit Mobile (UDK Mobile) por varias razones. La primera de ellas es que su motor gráfico, Unreal Engine 3, es muy usado en la industria de videojuegos y la calidad gráfica que se puede alcanzar con él es muy alta. La segunda razón es que es gratuito y no limita su uso de ninguna forma. Y la última razón es que nos permite crear clientes para dispositivos móviles. Es importante que la aplicación pueda ser controlada desde cualquier sitio, ya que esto aporta comodidades para el usuario.

UDK Mobile proporciona las siguientes tecnologías para la realización de la tesina:

- Unreal Editor: editor visual que permite la creación de niveles donde un nivel es una representación virtual de un escenario. Se usa para crear la representación virtual de una casa.
- Unreal Engine 3: motor gráfico que permite conseguir una calidad gráfica fotorrealista. Se usa para que la casa tenga una alta calidad visual.
- Unreal Kismet: editor visual que permite definir la jugabilidad. Se usa para definir la interacción del usuario con la casa y el comportamiento del escenario ante cada interacción.
- Unreal Script: lenguaje de scripting que permite añadir elementos o modificar elementos existentes de UDK Mobile. Se usa, entre otras cosas, para crear los elementos necesarios para la comunicación remota con un hogar inteligente.

Con estas tecnologías se puede crear una interfaz 3D visualmente atractiva y una alta

inmersión porque en la aplicación, el usuario está en la representación virtual de una casa. La representación virtual hace que el usuario pueda identificar mejor los elementos que con otras aproximaciones. La razón de esto es que con otras aproximaciones, el usuario debe aprender dónde están las opciones o aprender las órdenes de voz que le permiten realizar las acciones deseadas. Sin embargo, con la aproximación propuesta, el usuario solo debe hacer lo que haría en la vida real para hacer la acción deseada. Por ejemplo, si el usuario quisiera encender la luz del salón, en vez de navegar por menús para encontrar la opción buscada, es suficiente con ir al salón y usar el interruptor de la luz o la propia lámpara. Este tipo de interacción es más natural para el usuario que la navegación de menús o la memorización de órdenes.

Para hacer las pruebas para el control de servicios domóticos, se ha usado una maqueta real que está ubicada en el Centro de Investigación en Métodos de Producción de Software, ProS. Esta maqueta representa un hogar inteligente y tiene diversos dispositivos reales que se activan y desactivan en función de lo que haga el usuario con la aplicación. Algunos de estos dispositivos son una luz conmutable, una luz gradual, un ventilador y un rodillo de persiana.



Figura 3. Maqueta de un hogar inteligente

1.3. Organización del Resto del Documento

El resto del documento se estructura de la siguiente forma:

- El capítulo 2 describe el contexto tecnológico en el que se enmarca la tesina.
- El capítulo 3 muestra distintas aproximaciones para el control de un hogar inteligente.
- El capítulo 4 describe la aplicación 3D Home y los pasos que se han seguido para su desarrollo.
- El capítulo 5 explica la funcionalidad de 3D Home desde el punto de vista del usuario.
- El capítulo 6 describe qué se ha conseguido con el proyecto y las mejoras que se pueden realizar sobre 3D Home.
- El anexo A explica los componentes de los que está formado UDK Mobile, los tipos de objetos que usa y las herramientas que ofrece.
- El anexo B explica en detalle la herramienta UnrealEd.

- El anexo C explica en detalle la herramienta Unreal Kismet.
- El anexo D explica en detalle la herramienta Unreal Matinee.
- El anexo E explica en detalle el lenguaje de programación UnrealScript.

Capítulo 2

Contexto Tecnológico

Para mejorar la comprensión de esta tesina, este capítulo introduce el contexto tecnológico en el que se enmarca esta tesina. 3D Home es una aplicación que tiene el objetivo de controlar un hogar inteligente de una forma intuitiva para el usuario. En la aplicación, el usuario realiza sus acciones en primera persona interactuando con dispositivos ubicados en el escenario, de forma que si quiere controlar su hogar, únicamente debe realizar las acciones que haría en la vida real. Esto da lugar a la necesidad de introducir dos áreas: la domótica y los motores de escenarios 3D interactivos.

El resto de este capítulo está organizado de la siguiente forma: la sección 2.1 explica qué es la domótica y la clasificación de los sistemas domóticos según su arquitectura. Luego, la sección 2.2 describe el concepto de motor de escenarios 3D interactivos, razona su uso para el desarrollo de 3D Home y compara distintos motores con el fin de averiguar cuál es el más adecuado para la aplicación a desarrollar. Por último, la sección 2.3 concluye el capítulo.

2.1. Domótica

En esta sección se introduce el concepto de domótica y se describen los componentes de un sistema domótico. Asimismo, se describen los tipos de sistemas domóticos según su arquitectura.

La primera vez que aparece el término domótica es en Francia, donde se acuñó la palabra "Domotique" [3]. Esta palabra es la contracción de las palabras "domo" e "informatique". Se podría definir la domótica como "el concepto de vivienda que integra todos los automatismos en materia de seguridad, gestión de la energía y comunicaciones". Pero una definición más técnica de la domótica sería la siguiente: "La domótica es el conjunto de tecnologías aplicadas al control y automatización inteligente de la vivienda" [6].

La domótica aporta varios beneficios a la vivienda:

- Facilita el ahorro energético: por ejemplo, puede reducir el consumo eléctrico subiendo las persianas de día. También puede ofrecer al usuario la información recogida mediante la monitorización de consumo para que el usuario sepa las causas del gasto energético y pueda cambiar sus hábitos para reducir su consumo.
- Fomenta la accesibilidad: por ejemplo, si un discapacitado físico quiere llamar por teléfono para comunicarse con alguien o solicitar teleasistencia, éste podría usar órdenes o comandos de voz para dicho fin.
- Aporta seguridad a las personas, animales y bienes: por ejemplo, si se detecta un incendio o una fuga de gas, el sistema podría llamar a los bomberos y avisar a todos los residentes de la casa de dicho problema, o bien mediante señales acústicas si están dentro de la vivienda o bien mediante un mensaje al dispositivo móvil del residente si éste está fuera.
- Convierte la vivienda en un hogar más confortable: por ejemplo, cuando el usuario sale del trabajo podría encender el aire acondicionado y ajustar la temperatura de éste para que

su vivienda se enfríe o caliente mientras llega a casa.

- Permite la comunicación con el hogar de forma remota: por ejemplo, si el usuario se ha ido de vacaciones podría revisar desde fuera de casa si está todo apagado, y en caso contrario apagarlo desde su dispositivo móvil.

Un sistema domótico puede ser tan sencillo como controlar solo la luz del salón de una vivienda. Pero también puede ser tan complejo como controlar toda una vivienda [20]. Un sistema domótico puede incluso saber en tiempo real la posición aproximada de cada residente de la vivienda [22]. Así, por ejemplo, se puede saber no solo si un residente está en el salón sino también si está junto a la televisión o en un sofá.

Un ejemplo de uso diario de un sistema domótico se describe en [3]: "Son las 7 de la mañana, suena el despertador, se levantan las persianas y se enciende la luz. Puntualmente como cada mañana el procesador le despierta, con la tranquilidad de saber que ha estado toda la noche cuidando su vivienda. Si hubiera habido algún escape de agua lo habría cortado y tendría un aviso. El jardín ha estado toda la noche protegido por un sistema de detección perimetral que conecta automáticamente los focos y el riego. Cuando baja a desayunar, el café ya está caliente, al igual que la cocina, que se ha encendido cuando él entraba. No se va a molestar en apagarla, ni tampoco las luces del pasillo porque lo hará el procesador. Al pasar por el cuarto de los niños, nota que acaba de encender la calefacción, les quedan quince minutos para levantarse. Ayer estuvieron jugando en el cuarto ¡menos mal que los enchufes fueron desactivados por el procesador!. Cuando se va de casa, toca suavemente la pantalla táctil de la entrada, le comunica que no hay ninguna ventana ni puerta abierta. Al salir con el coche por el jardín, se da cuenta que los primeros rayos del sol han apagado la luz exterior y han abierto las persianas del salón. Cuando llegue a la oficina, conectará el ordenador, introducirá su código personal y durante toda la mañana sabrá todo lo que pasa en su vivienda. Si de camino en el coche se ha olvidado de conectar algo, llamará con su teléfono móvil y le dirá al procesador que lo haga por él. Lo mismo hará cuando vaya de viaje a su apartamento que tiene en la sierra una hora antes de llegar, dará la orden para que el procesador conecte la calefacción y el apartamento se vaya caldeando."

En 2010, se han realizado 0.44 millones instalaciones de sistemas domóticos en todo el mundo. Se espera que este número vaya creciendo durante los próximos cuatro años a una tasa de un 65% anual llegando a las 5.38 millones de instalaciones en 2015 [7]. Este crecimiento se debe a que los consumidores quieren controlar sus hogares desde fuera de sus viviendas.

Existen empresas que ofrecen sus servicios como instaladores de sistemas domóticos. En España hay muchas empresas que se dedican a este tipo de servicios como por ejemplo Acive en Madrid, Domonova en Sevilla y BHT Ingenieros en Alicante [5].

2.1.1. Componentes de un Sistema Domótico

Un sistema domótico forma una red de control sobre los elementos que se desea controlar. Esta red de control se integra con la red de energía eléctrica y se coordina con el resto de redes con las que tenga relación como la telefonía, televisión y tecnologías de la información. Los distintos dispositivos que se pueden hallar en un sistema domótico se pueden agrupar en controladores, actuadores, sensores, buses de información e interfaces [20].

La pieza central de un sistema domótico es el **controlador** que es un aparato que gestiona el sistema domótico según la programación e información que recibe de los sensores, del usuario o de otros sistemas interconectados. En el sistema puede haber uno o más controladores. La información del sistema es mostrada en una **interfaz** que el usuario también puede usar configurar el controlador. La información que envían los **sensores**, el usuario y otros sistemas es transmitida al controlador mediante un **bus** que puede ser un cableado propio, redes de otros sistemas (red

eléctrica, red telefónica, red de datos) o de forma inalámbrica. El controlador recoge toda la información y la procesa. Una vez procesada, si hay que hacer alguna acción sobre algún electrodoméstico de la vivienda, el controlador envía señales a un aparato pequeño que hace de intermediario entre el controlador y el electrodoméstico a controlar. Estos aparatos se llaman **actuadores** y son capaces de ejecutar y/o recibir una orden del controlador y realizar una acción sobre un aparato o sistema (encendido/apagado, subida/bajada, apertura/cierre, etc.). En la figura 4 se muestra un esquema de este proceso.

Un sistema domótico puede acceder a redes exteriores de comunicación o información a través de las redes de la vivienda como la red telefónica o Internet.

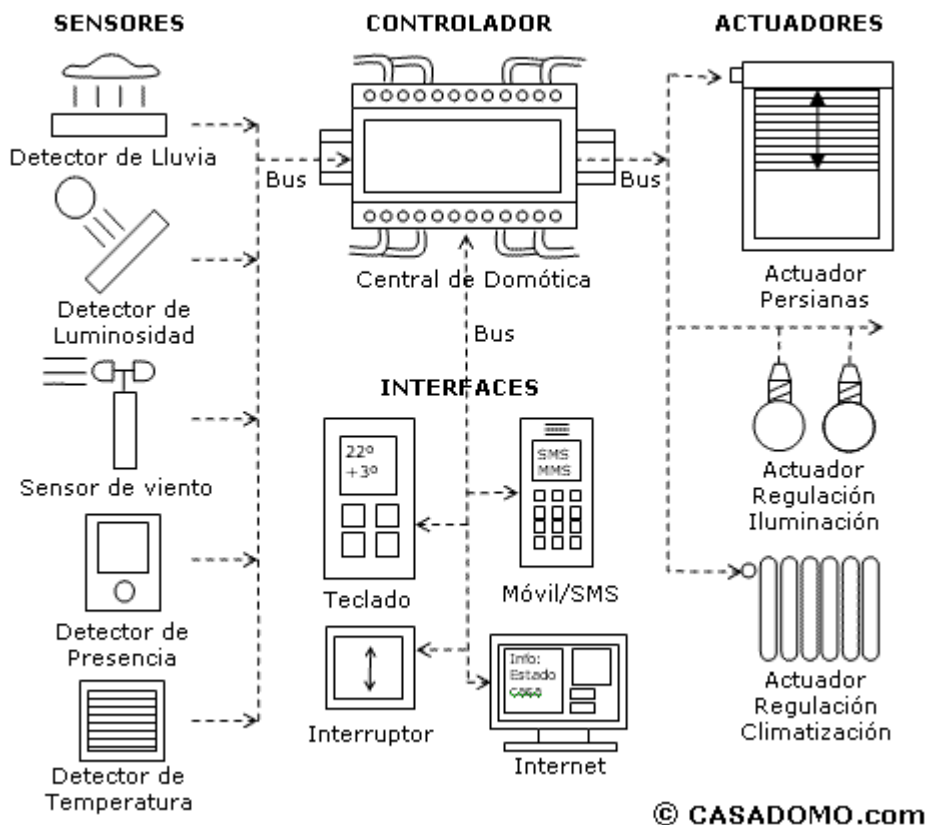
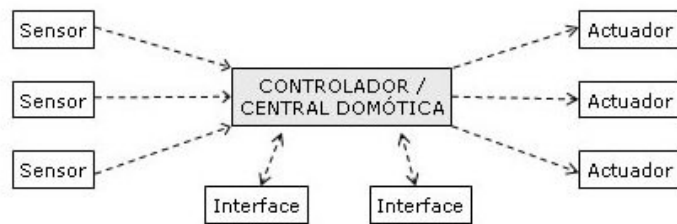


Figura 4. Esquema de funcionamiento de un sistema domótico

2.1.2. Arquitectura de un Sistema Domótico

Al instalar un sistema domótico, se debe decidir el número de controladores que éste va a tener y cómo van a estar distribuidos en la vivienda. El número de controladores y su localización definen la arquitectura del sistema domótico. Las distintas arquitecturas de los sistemas domóticos son:

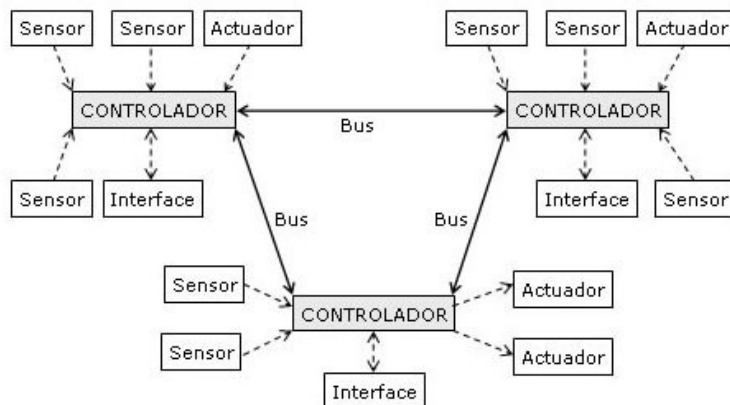
- **Arquitectura centralizada:** Un único controlador está conectado a los sensores, actuadores e interfaces. Este controlador es el corazón de la vivienda inteligente y si deja de funcionar, todo deja de funcionar.



© CASADOMO.com

Figura 5. Esquema de arquitectura centralizada

- Arquitectura descentralizada: Cada actuador, sensor e interfaz está conectado a un único controlador. Varios controladores, interconectados por un bus, se encargan del sistema domótico. Esta arquitectura evita que todos los sensores y actuadores del hogar dependan de un único controlador, así si un controlador deja de funcionar, solo quedarían afectados los que estuvieran comunicados con ese controlador.



© CASADOMO.com

Figura 6. Esquema de arquitectura descentralizada

- Arquitectura distribuida: Cada sensor y actuador es también un controlador. En esta arquitectura, cada controlador se ocupa solo del dispositivo en el que está instalado. Las ventajas de este tipo de sistemas son que el mal funcionamiento de un elemento no impide el correcto funcionamiento de los demás elementos, la facilidad en la instalación y la facilidad de ampliación por el escaso cableado.



© CASADOMO.com

Figura 7. Esquema de arquitectura distribuida

- Arquitectura híbrida/mixta: Se combinan las arquitecturas centralizadas, descentralizadas y distribuidas. A la vez que puede disponer de un controlador central o varios controladores descentralizados, los dispositivos de interfaces, sensores y actuadores pueden también ser controladores y procesar la información sin que necesariamente pase por otro controlador.

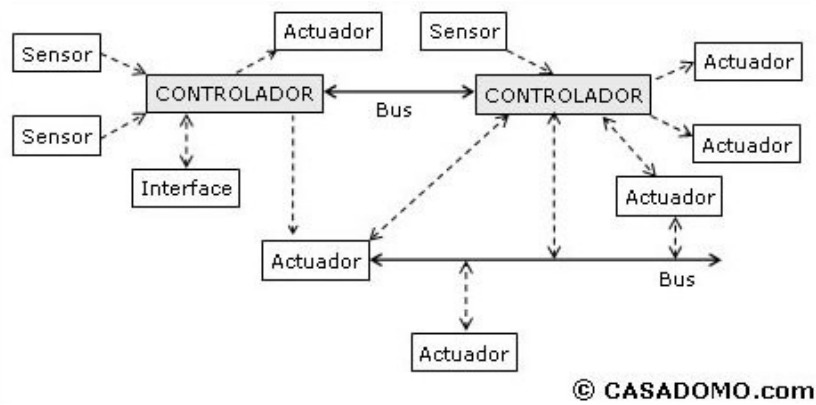


Figura 8. Esquema de arquitectura híbrida/mixta

2.2. Motores de Escenarios 3D Interactivos

Una vez explicados los sistemas domóticos, esta sección explica qué son los motores de escenarios 3D interactivos y qué ventajas y desventajas ofrecen a los desarrolladores de aplicaciones. Por último se describen y comparan varios de estos motores con el objetivo de elegir el más adecuado para el desarrollo de 3D Home.

Un motor de escenarios 3D interactivos abstrae los detalles de las tareas comunes de creación de escenarios 3D interactivos, como el renderizado y la física, para que los desarrolladores puedan centrarse en los detalles que diferencien sus escenarios de los demás [35].

Los motores ofrecen componentes reutilizables que pueden ser manipulados para crear escenarios interactivos. La animación de modelos, la detección de colisiones entre objetos, la física y las interfaces gráficas de usuario pueden ser componentes de un motor. Sin embargo, el contenido del escenario, los modelos y texturas específicos, el significado detrás de las colisiones de objetos y la forma en que los objetos interactúan con el mundo son los componentes que hacen al escenario único.

Las interfaces de programación de aplicaciones (APIs, Application Programming Interfaces) son las interfaces software que los sistemas operativos, bibliotecas y servicios proporcionan para que se puedan usar. Un kit de desarrollo de software (SDK, Software Development Kit) es una colección de librerías, APIs y herramientas que están disponibles para programar esos mismos sistemas operativos y servicios. Muchos motores ofrecen APIs en sus SDKs.

Durante mucho tiempo, las compañías hacían sus propios motores y mantenían esta tecnología en la empresa. Sin embargo, en los últimos años, el coste de hacer los motores se ha incrementado significativamente, y cada vez más compañías han empezado a especializarse en hacer motores completos o componentes de motores para venderlos a otras compañías. La forma de vender los motores es mediante licencias que permiten su uso y/o modificación. Este tipo de compañías se llaman proveedores de *middleware*. Como resultado, casi todos los componentes de un motor son comprados a una amplia variedad de precios, o descargados de proyectos de código abierto.

En 2009, Mark DeLoura realizó una encuesta a un gran grupo de ejecutivos de la industria del videojuego donde se puede ver qué opina la industria sobre los motores de escenarios 3D interactivos [8].

El 55% de los encuestados afirmaron estar usando un motor en su proyecto actual. De los que usan motores, el 39% usan Unreal, y el 22% usan Gamebryo. A pesar del gran uso de los motores comprados, solo el 9.3% dijeron que preferirían usarlos incluso si pudieran hacerlo fácilmente de

otra forma. El uso de los motores es tan extendido por las razones que se muestran en la siguiente gráfica.

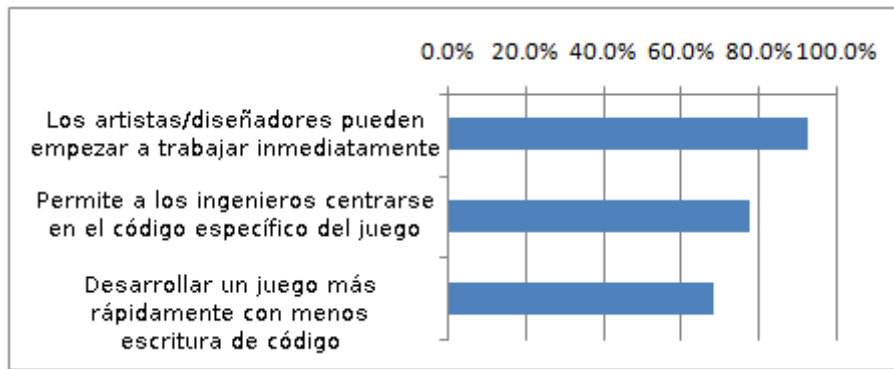


Figura 9. Ventajas más importantes de los motores de escenarios 3D interactivos

Los motores proporcionan a los creadores de contenidos herramientas de prototipado para conseguir una interactividad única o más refinada. Permiten a los programadores centrarse en la creación de tecnologías que distingan al escenario de otros de temática similar. Comprar un motor en vez de crear uno proporciona una ventaja adicional que es que facilita la contratación de gente familiarizada con el motor. Esto es porque el motor comprado puede haber sido usado por otros desarrolladores para crear sus aplicaciones.

Por otro lado, lo que más les preocupa a los encuestados se muestra en la siguiente gráfica.

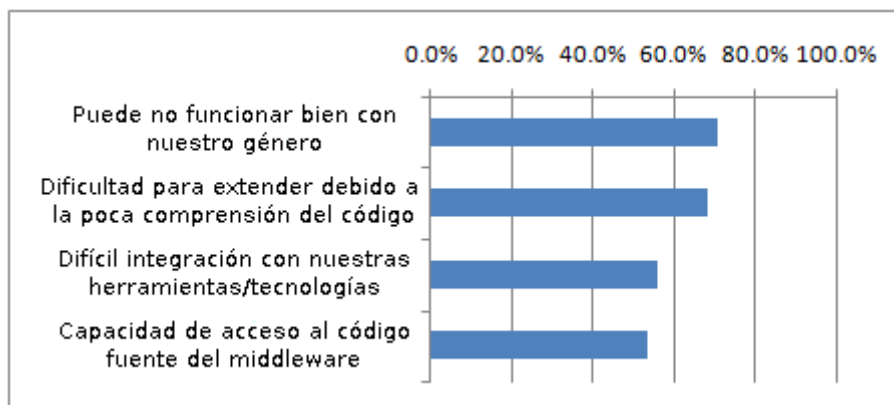


Figura 10. Desventajas más importantes de los motores de escenarios 3D interactivos

Las desventajas que afectarían al desarrollo de 3D Home son que el motor sea difícil de usar y de extender, que sea complicada la integración del motor con la maqueta y que no se pueda tener acceso al código fuente del motor. Esto último es una desventaja porque su acceso ayudaría tanto a entender el funcionamiento del motor como a evaluar la integración del motor con la maqueta.

Aunque hay una gran variedad de motores de escenarios 3D interactivos en el mercado, solo se van a explicar los tres más usados por la industria y Unity. Se va a explicar Unity porque éste es especialmente útil para desarrollar escenarios interactivos integrados en un navegador web.

2.2.1. Unreal Engine 3

El primero de ellos es Unreal Engine 3 [16] que fue usado para crear juegos como, por ejemplo, Gears of War. También ha sido usado en otros campos ajenos a los videojuegos como en arquitectura, cine y educación [23].

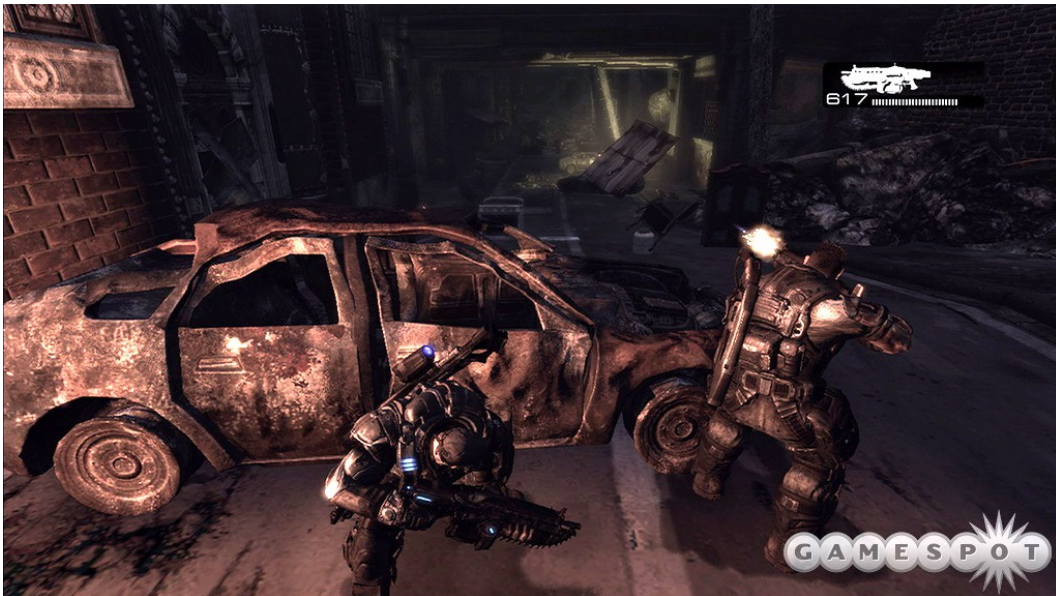


Figura 11. Gears of War

Unreal Engine 3 soporta OpenGL y DirectX 9 y 10. Además, permite crear juegos para PC, Xbox360, PS3 e iOS.

Algunas de las herramientas que ofrece Unreal Engine 3 son luces dinámicas, sistema de red, motor de físicas, shaders y sombreado. Shader es una tecnología [25] que utiliza un lenguaje específico de alto nivel que permite la independencia del hardware. El objetivo de los shaders es proporcionar al programador una interacción con la Unidad de Procesamiento Gráfico. Los shaders son utilizados para realizar transformaciones y crear efectos especiales, como iluminación, fuego o niebla. Unreal Engine 3 también:

- Permite el uso de música dinámica. Ésta es cuando los eventos del escenario pueden cambiar la música de fondo.
- Crea animaciones con AnimTrees, que son árboles de nodos de animación. Este árbol permite controlar cada detalle de la animación, como un músculo o hueso.
- Define la jugabilidad de forma visual, con Unreal Kismet.
- Permite crear scripts para modificar o añadir elementos a Unreal Engine 3. Para ello se usa el lenguaje de scripting Unreal Script, con éste se podría, por ejemplo, añadir nuevos tipos de luces o modificar el comportamiento de algún tipo de luz ya existente.

Unreal Engine 3 usa un editor visual llamado UnrealEd para crear niveles o escenarios.

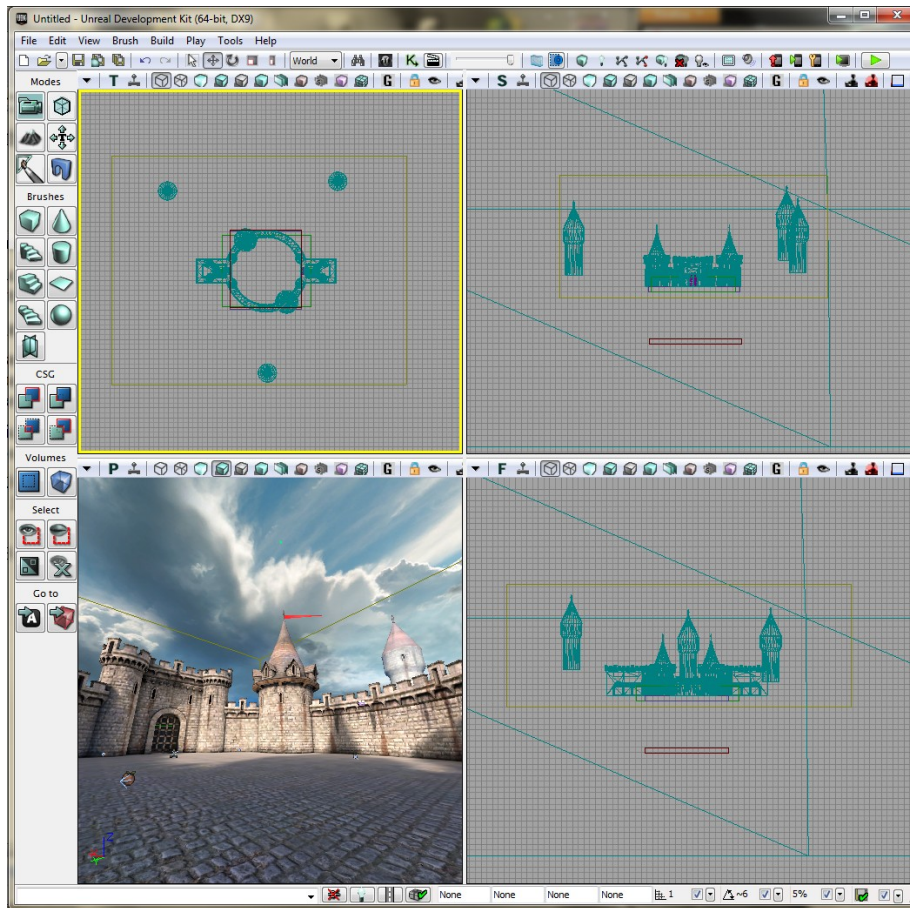


Figura 12. UnrealEd

2.2.2. Torque

El segundo de ellos es Torque [18] que fue usado para crear juegos como, por ejemplo, Penny Arcade Adventures.



Figura 13. Penny Arcade Adventures

Torque soporta tanto DirectX 9 como OpenGL. Además, permite crear juegos para PC, Mac, Web, Wii, Xbox 360 e iOS.

Algunas de las herramientas que ofrece Torque son luces dinámicas, sistemas de red, motor de físicas, shaders y sombreado. Además:

- Desde su versión más barata, 100\$ USD, se tiene acceso al código fuente del motor de escenarios 3D interactivos tanto para leerlo como para modificarlo.
- Facilita la publicación de aplicaciones basadas en navegadores web.

Torque usa un editor visual llamado World Editor para crear niveles. Este editor permite ver en vivo los cambios hechos en los materiales. Es decir, si se desea cambiar una textura del escenario, un artista puede cargarla en un programa de edición de imágenes como Photoshop o Gimp para hacer los cambios necesarios. La textura aparece en el motor con los nuevos cambios en tiempo real.

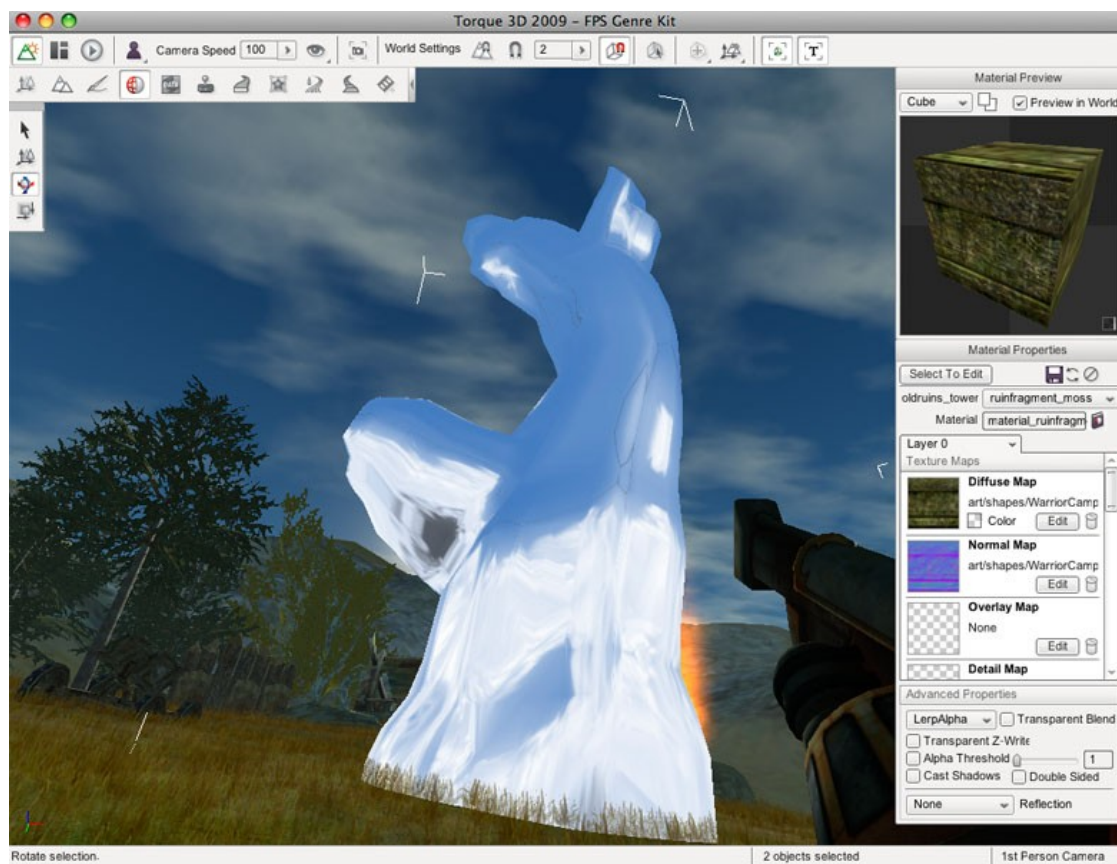


Figura 14. World Editor

2.2.3. Gamebryo

El tercero de ellos es Gamebryo [17] que fue usado para crear juegos como, por ejemplo, Fallout 3.



Figura 15. Fallout 3

Gamebryo soporta DirectX 9 y 10. Además, permite crear juegos para PC, Xbox360, PS3 y Wii.

Algunas de las herramientas que ofrece Gamebryo son luces dinámicas, sistema de red, motor de físicas, shaders y sombreado. Además:

- Permite el uso de música dinámica.
- Ofrece una solución de servidor para juegos online, lo que lo hace interesante para desarrollar juegos multijugador masivo online (MMOG, Massive Multiplayer Online Game).
- Ofrece una solución de transmisión multinúcleo llamada Floodgate con un enfoque particular para los desarrolladores de PlayStation 3 y Xbox 360.
- Usa SWIG (Simplified Wrapper and Interface Generator) que permite usar multitud de lenguajes de alto nivel, como Python, Ruby, Java, Lua y GameMonkey, para definir o modificar el comportamiento de entidades.

Gamebryo usa un editor visual llamado World Builder para crear niveles o escenarios. Además, proporciona también un previsualizador de tiempo real que se puede activar o pausar para que se pueda ver cómo quedaría realmente el escenario.

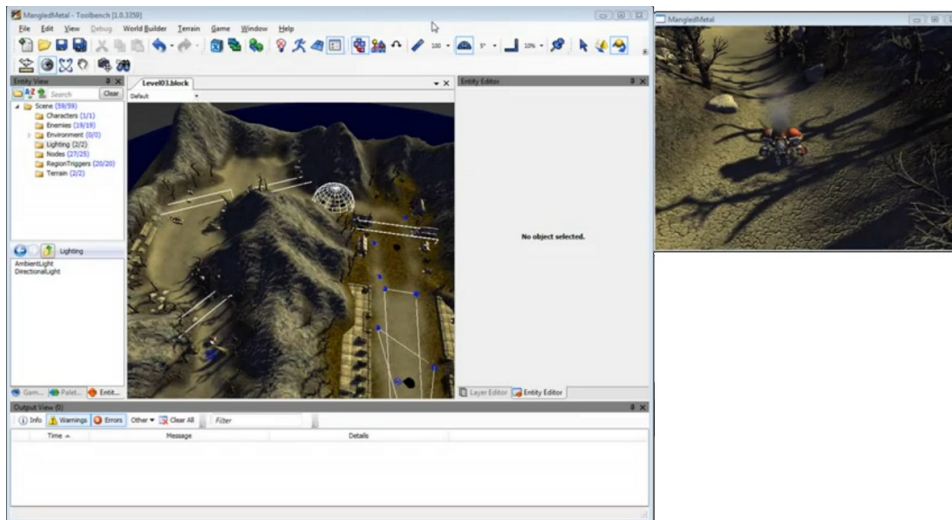


Figura 16. World Builder (izq) y previsualizador de tiempo real (der)

2.2.4. Unity

El último de los motores del que se va a hablar es Unity [34] que fue usado para crear juegos como, por ejemplo, Tiger Woods PGA Tour Online. Además, destaca cuando se quieren crear juegos para la web [9].



Figura 17. Tiger Woods PGA Tour Online

Unity 3 soporta tanto DirectX 9 y 10 como OpenGL. Además, permite crear juegos para PC, Mac, Web, Xbox 360, PlayStation 3, Wii, iOS y Android.

Algunas de las herramientas que ofrece Unity son luces dinámicas, sistema de red, motor de físicas, shaders y sombreado. Además:

- Es un motor visual de escenarios 3D interactivos, es decir, un motor de alto nivel de abstracción. Como tal, la escritura de código es mínima y se pueden hacer escenarios 3D interactivos de la forma más rápida. Sin embargo, no tiene tanta potencia como otros motores de menor nivel de abstracción como Unreal Engine 3.
- Facilita la comunicación entre una página web y el escenario. Para ello, usa Unity Web Player.

- Para acceder a páginas y servicios web, proporciona una interfaz fácil de usar. Permite tanto el modo síncrono como asíncrono.
- Permite integrar el escenario en un navegador web como un plugin.
- Usa los lenguajes de scripting JavaScript, C# y un dialecto de Python llamado Boo. Todos ellos son interoperables y pueden usar librerías .NET.
- Proporciona un entorno de desarrollo integrado para crear los scripts, éste es MonoDevelop.

Unity usa un editor visual llamado Unity Editor para crear niveles. El editor también muestra todas las variables públicas de los scripts para que sean fácilmente configurables y se vea de forma inmediata su efecto.

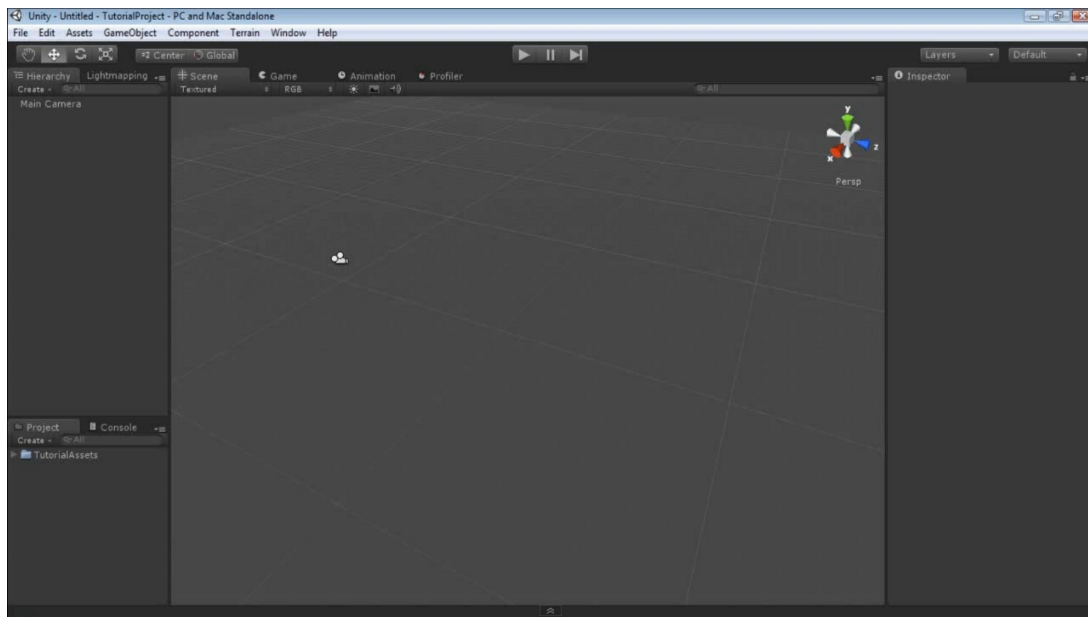


Figura 18. Unity Editor

2.2.5. Discusión

Después de haber descrito los motores de escenarios 3D interactivos, se va a comparar entre ellos para ver cuál es el más conveniente para el desarrollo de 3D Home.

Todos los motores anteriormente descritos proporcionan, entre otras herramientas, luces dinámicas, sistema de red, motor de físicas, shaders y sombreado. Todos ellos, con la licencia adecuada, también permiten el acceso y modificación del código fuente. Además, usan un editor visual para la creación de niveles. Por último soportan DirectX 9 y 10, a excepción de Torque que no soporta DirectX 10.

En la tabla 1 se puede observar para qué plataformas se pueden crear escenarios 3D interactivos con estos motores. Como se puede ver, el único que no permite crear escenarios en iOS es Gamebryo.

	PC	Mac	Web	Xbox 360	PlayStation 3	Wii	iOS	Android
Unreal Engine 3	Sí	No	No	Sí	Sí	No	Sí	No
Torque	Sí	Sí	Sí	Sí	No	Sí	Sí	No
Gamebryo	Sí	No	No	Sí	Sí	Sí	No	No
Unity	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí

Tabla 1. Plataformas objetivo

En la tabla 2 se muestran detalles del uso del lenguaje de programación. Un nivel medio de abstracción significa que se requiere un poco de programación para hacer escenarios completos. Un nivel alto de abstracción significa que permite crear escenarios sin escribir líneas de código, aunque sigue siendo posible recurrir a la programación para personalizar el escenario. Un motor con un nivel alto de abstracción permite crear escenarios de forma más rápida pero no ofrece tanta libertad como un motor con un nivel de abstracción medio.

	Nivel de Abstracción	Lenguajes para Scripting	IDE para Scripting
Unreal Engine 3	Medio	UnrealScript	No
Torque	Medio	TorqueScript	No
Gamebryo	Medio	Muchos usando SWIG	No
Unity	Alto	JavaScript, C# y Boo	Sí, MonoDevelop

Tabla 2. Detalles del uso del lenguaje de programación

Como se puede ver en la tabla 2, con Unity se pueden hacer escenarios más rápidamente porque tiene un nivel de abstracción más alto que los demás. Además es el único que proporciona un entorno de desarrollo integrado para crear los scripts. Por otro lado, Gamebryo es el que ofrece mayor libertad en la elección de lenguaje para crear scripts.

En la tabla 3 se expone si hay versiones gratuitas de los motores, si ha sido usado en otros campos ajenos a los videojuegos y otra información destacable.

	Versión Gratuita	Otros Campos	Otras Notas
Unreal Engine 3	Sí	Sí	Destaca en la calidad gráfica y es el más usado.
Torque	No	No	Destaca en el bajo coste de sus licencias, desde 100\$ USD.
Gamebryo	No	No	Destaca en la creación de MMOGS.
Unity	Sí, con funcionalidad limitada	No	Destaca en la creación de juegos web.

Tabla 3. Disponibilidad de versión gratuita y uso en campos ajenos a los juegos

En la tabla 3 se puede ver que Unreal Engine 3 y Unity ofrecen versiones gratuitas de sus motores, aunque Unity limita sus funcionalidades con esa versión. Aunque todos ofrecen una alta calidad gráfica, el que mejor calidad visual ofrece es Unreal Engine 3, llegando al fotorrealismo. Por último, Unreal Engine 3, además de ser el más usado, ha sido usado también en otros campos como arquitectura, educación y cine.

Entre los motores descritos, los que nos permiten realizar nuestro proyecto son Unreal Engine 3 y Unity porque se pueden hacer aplicaciones para dispositivos móviles. Además, tienen versiones gratuitas, aspecto que es preferible porque se adecua al carácter académico de este trabajo.

Este trabajo se ha realizado con Unreal Engine 3 porque no limita su funcionalidad y por ofrecer una mayor calidad visual que Unity. Además, la comunidad que apoya a Unreal Engine 3 es mucho mayor que Unity.

2.3. Conclusiones

En este capítulo se ha visto que la domótica es un conjunto de tecnologías cuyo objetivo es el control y automatización inteligente de una vivienda. Además, el número de instalaciones de sistemas domóticos está en crecimiento. Luego, se han detallado los componentes que forman un sistema domótico. Uno de estos componentes, el controlador, es el que determina la arquitectura del sistema domótico mediante la cantidad de controladores que haya y su localización.

A continuación se ha explicado que un motor de escenarios 3D interactivos facilita la creación de escenarios 3D interactivos porque aspectos que se repiten en todos los escenarios interactivos, como el renderizado y la física, vienen ya implementados. De esta forma los desarrolladores pueden centrarse en hacer su escenario interactivo único.

Después se ha explicado lo que son las compañías *middleware*, que son compañías que se han especializado en crear y vender sus propios motores o componentes de motores. La mayoría de desarrolladores, en vez de hacer su propio motor, compran licencias de motores.

Luego se han descrito los motores Unreal Engine 3, Torque, Gamebryo y Unity. Finalmente se ha elegido usar Unreal Engine 3 porque permite hacer aplicaciones para dispositivos móviles, ofrece una versión gratuita que no limita su funcionalidad, proporciona una alta calidad gráfica y es el motor más usado.

Este trabajo propone una aplicación para controlar un hogar inteligente. Esta aplicación muestra una representación virtual 3D de la vivienda que el usuario puede explorar en primera persona desde dentro de la casa. Además, el usuario también puede interactuar con distintos elementos de la casa para controlar el sistema domótico de su vivienda real. Para crear esta aplicación, lo más indicado es usar un motor de escenarios 3D interactivos porque éstos ofrecen herramientas para crear representaciones 3D de escenarios y para definir interacciones del usuario con los elementos de la aplicación.

Capítulo 3

Estado del Arte

El estado del arte es el conjunto de trabajos existentes que tratan el mismo problema que este trabajo. Este trabajo propone una aplicación para el control de un hogar inteligente, así que el estado del arte son todas aquellas aplicaciones cuyo objetivo también es dicho control. Estas aplicaciones, se pueden clasificar dependiendo del tipo de interacción que usan. Sin embargo, para entenderlas bien, antes se debe entender cómo funciona cada tipo de interacción en el ámbito de los hogares inteligentes. Una vez explicado un tipo de interacción, se explican uno o dos ejemplos de aplicaciones que usan este tipo de interacción.

En resumen, este capítulo introduce diversas aproximaciones para el control de una casa inteligente.

El resto de este capítulo está organizado de la siguiente forma: la sección 3.1 explica la interacción basada en interfaces WIMP, la sección 3.2 describe la interacción basada en el reconocimiento de voz, la sección 3.3 explica la interacción basada en el reconocimiento de gestos, la sección 3.4 describe la interacción basada en el contexto y la sección 3.5 explica la interacción multimodal. Todas estas secciones exponen ejemplos de uso en hogares inteligentes. Por último, la sección 3.6 concluye el capítulo.

3.1. Interfaces WIMP

El primer tipo de interacción a explicar es el más sencillo de todos.

Las interfaces WIMP son interfaces gráficas cuya interacción con el usuario está basada únicamente en ventanas, iconos, menús y dispositivos de apuntado como ratones y dedos.

3.1.1. Ejemplo de uso

Carlos Serrano, un alumno de la Universidad Politécnica de Valencia, desarrolló una interfaz de este tipo para controlar, mediante un iPhone, un hogar inteligente [33].

La aplicación permite controlar de forma remota varios hogares inteligentes, seleccionando cuál controlar mediante un listado como se muestra en la figura 19.



Figura 19. Listado de viviendas inteligentes

Una vez elegida la vivienda, el usuario elige la ubicación del dispositivo que quiere controlar. También puede optar por elegir directamente el dispositivo. Estas opciones se muestran en la figura 20.



Figura 20. Menú de una vivienda

Por último, cuando ya se ha elegido el dispositivo, puede cambiar el estado del dispositivo con los distintos elementos que ofrece la aplicación como botones o barras de desplazamiento como se muestra en la figura 21.

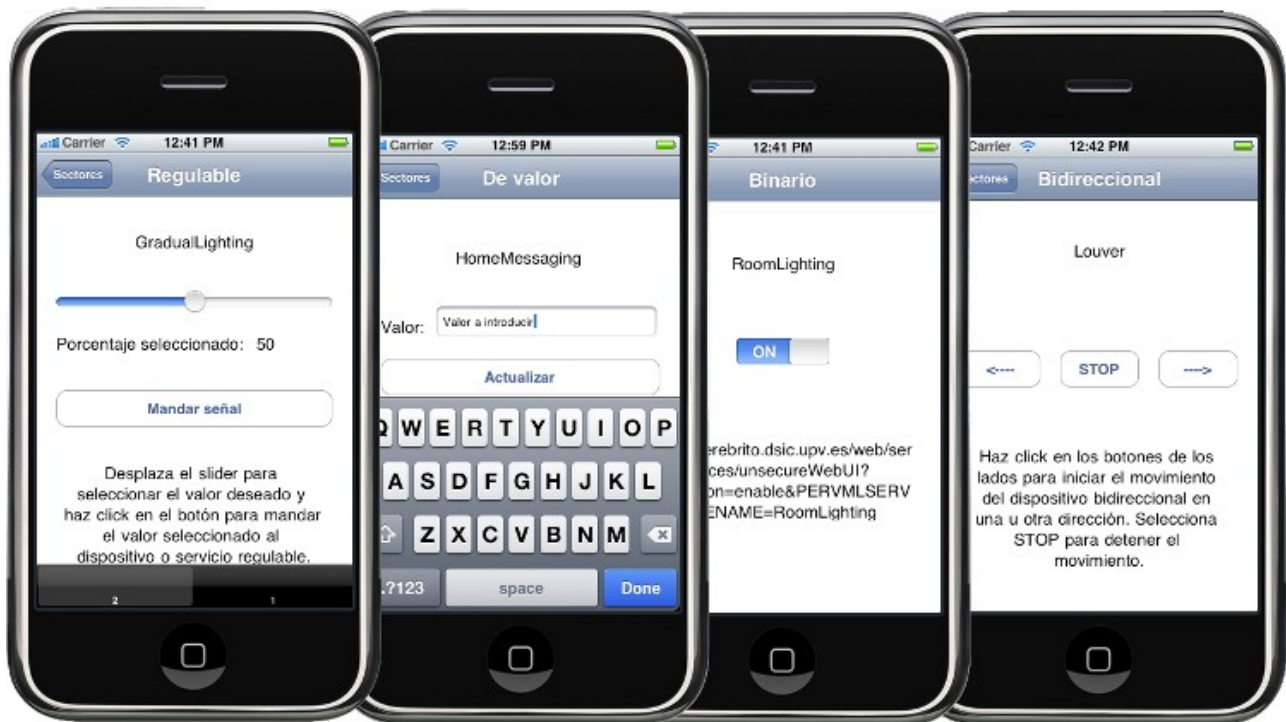


Figura 21. Interfaces para cambiar el estado de los dispositivos

3.2. Reconocimiento de Voz

El segundo tipo de interacción a explicar es el reconocimiento de voz restringido al ámbito de los hogares inteligentes.

Cuando se considera la interacción natural con los usuarios, uno de los métodos más amigables es la interacción vocal [26]. La interacción vocal es adecuada para el entorno físico de los hogares inteligentes porque puede ser usada, por ejemplo, mientras el usuario se está duchando, subiendo escaleras o está en la oscuridad.

Las aplicaciones de reconocimiento de voz pueden clasificarse en:

- Sistemas de reconocimiento de palabras aisladas. En estos sistemas cada palabra es pronunciada con pausas antes y después. Por ejemplo, podría tratar frases como "... Calentar ... horno ... a ... 220 ... grados ..."
- Aplicaciones de órdenes y control con un vocabulario pequeño. En estos sistemas, la gramática está bien definida y las órdenes a reconocer deben ajustarse completamente a esta gramática.. Por ejemplo, si un sistema solo reconoce los infinitivos de algunos verbos podría tratar frases como "Calentar el horno a 220 grados" pero no frases como "Calienta el horno a 220 grados".
- Sistemas de habla continua con un vocabulario grande. Estos sistemas son capaces de eliminar información no significativa para el sistema y reconocer la orden emitida por el usuario. Por ejemplo, podría tratar frases como "Calentar el horno a 220 grados" y "Quiero que calientes el... el horno a... no sé... a 220 grados"

Un sistema de hogar inteligente sería un sistema de habla continua cuya flexibilidad lingüística debería ser limitada mediante el diseño de una gramática adecuada.

La habilidad de reconocer con precisión habla capturada que ha sido limitada de la forma descrita arriba, depende principalmente del tamaño del vocabulario y del ruido ambiental como se muestra en la figura 22. Así se puede mejorar el reconocimiento restringiendo el tamaño del vocabulario, y después reduciendo el ruido en la señal recibida por el sistema de interacción vocal.

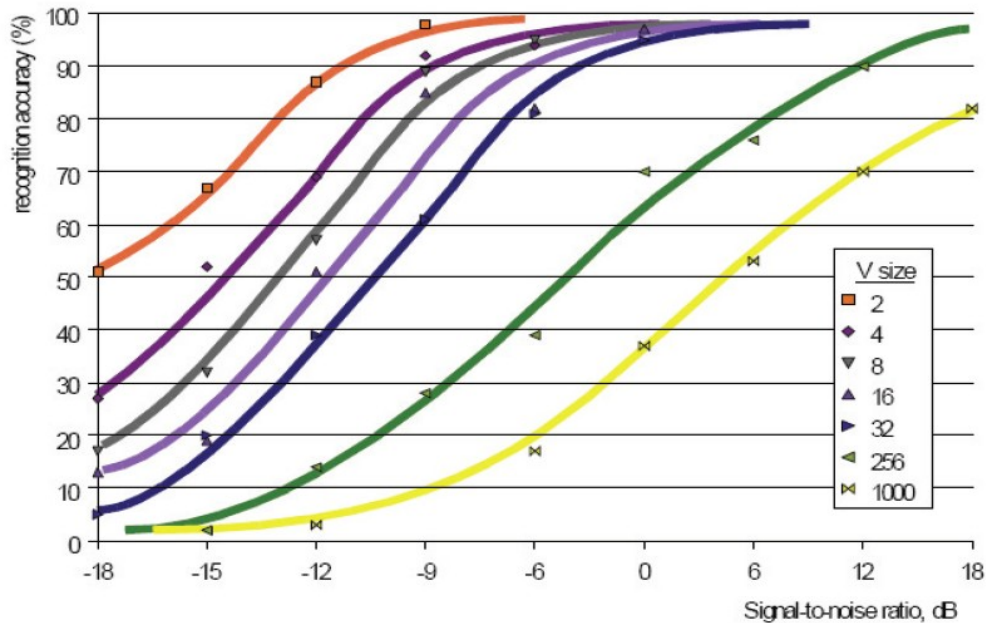


Figura 22. Efecto del tamaño del vocabulario de SNR en el reconocimiento de voz

Muchas aplicaciones de órdenes y control tienen un vocabulario pequeño (hasta 50 palabras). Por ejemplo, una frase hablada configurable que identifique al usuario para iniciar una sesión de interacción vocal, las órdenes para controlar las luces podrían incluir "enciende", "apaga", la localización, y algunas pocas palabras más. Además, el dispositivo a ser controlado debería ser identificado. Por ejemplo, "Por favor, enciende la luz del baño" está compuesto de una frase identificadora "Por favor", una operación "enciende", una localización "baño", y un dispositivo "luz".

En los hogares inteligentes, el reconocimiento de habla puede estar controlado de forma distribuida, con dispositivos empotrados como una televisión que sea controlable con la voz. Actualmente, estos dispositivos tienen pocos recursos de sistema, así que es necesario emplear un motor de reconocimiento de habla ligero y rápido en dichos contextos.

También es posible realizar el reconocimiento de voz utilizando un servidor central que se encarga del procesamiento, conectado a un conjunto de micrófonos y altavoces repartidos por toda la casa: esto requiere un ancho de banda de comunicaciones mayor que en un sistema distribuido e introduce retrasos en las comunicaciones. Sin embargo, permite al motor de reconocimiento automático de habla operar en un ordenador más rápido con menos restricciones de memoria. Con la continua mejora de los sistemas empotrados, esta razón está siendo cada vez más débil.

La interacción vocal para hogares inteligentes proporciona un buen objetivo de implementación para el reconocimiento automático de habla: el conjunto de usuarios es pequeño y pueden ser predeterminados, las localizaciones físicas están bien definidas, la gramática y el conjunto de órdenes puede ser restringido, y muchas fuentes de ruido ya están controladas por el sistema de control del hogar. Una alternativa razonable es el uso de un dispositivo móvil llevado por un usuario, con el que hablar. Ejemplos serían que el usuario llevara un micrófono colgado del cuello de la camisa o incluso que se comunicara mediante el teléfono móvil. Esto simplifica significativamente el procesamiento de audio requerido porque el dispositivo móvil siempre es el receptor del mensaje hablado, sin embargo, el usuario estaría obligado a llevar dicho dispositivo.

Las asunciones de un hogar inteligente de este tipo incluye que el usuario ha realizado el entrenamiento, conoce la sintaxis operacional de la interacción vocal y ha tenido tiempo para familiarizarse con el sistema. Si no es así, el usuario no podrá comunicarse con el hogar inteligente de forma eficiente.

Uno de los criterios principales de rendimiento en el reconocimiento de habla es el porcentaje de tareas que son completadas sin la intervención de otro usuario.

El objetivo final de la interacción vocal en los hogares inteligentes es que sea capaz de interpretar, entender y realizar órdenes complejas como "Por favor, resérvame un vuelo nocturno a Tokyo el siguiente jueves y reserva una suite adecuada en el hotel Grand Hyatt". Probablemente, todavía queden varios años para que este tipo de órdenes sean manejadas con éxito por los sistemas automáticos.

3.2.1. Ejemplos de Uso

INSPIRE

INSPIRE es un asistente del hogar que puede controlar un hogar inteligente mediante el habla [27]. INSPIRE está compuesto de algunos componentes como preprocesamiento acústico y cancelación adaptativa de ruido, reconocimiento de habla y hablante, gestión de diálogo, y salida de habla. El sistema puede ser dirigido en el entorno del hogar mediante una matriz de micrófonos, un micrófono portátil, o de forma remota mediante la red telefónica. INSPIRE está disponible en dos idiomas, alemán y griego.

Como mecanismos de salida de información, este sistema de interacción vocal ofrece tres metáforas distintas:

- "Cabeza parlante" que consiste en un avatar que habla en una pantalla montada en una pared.
- "Fantasma" que consiste en un asistente invisible que proporciona información usando los altavoces de toda la casa.
- "Dispositivos inteligentes" que consiste en un asistente invisible que proporciona información usando solo un único altavoz.

Sphinx

Sphinx es un reconocedor automático de habla, es de código abierto y es un ejemplo excelente de un sistema moderno y flexible de reconocimiento de habla [26]. Sphinx proporciona e integra algunas capacidades que le permiten ser adaptado a un amplio rango de aplicaciones diferentes de reconocimiento de habla.

Puede ser usado para el reconocimiento de una única palabra, o expandido para tener vocabularios grandes que contengan decenas de miles de palabras. Puede ejecutarse desde en un sistema empujado (PocketSphinx) hasta en un servidor grande y potente. Sphinx es altamente configurable y muy flexible.

Sphinx2, el motor de decodificación para Sphinx II, puede ser una buena opción para los servicios de un hogar inteligente porque usa algunos archivos de modelo y bases de datos apropiadas. Éstos son clasificados en tres categorías:

- El diccionario/lexicón de pronunciación que define palabras de interés, y la pronunciación fonética de cada una.

- Modelos acústicos basados en Modelos Ocultos de Markov (HMM, Hidden Markov Models) para fonos base y trifonos. Sphinx2 usa modelos acústicos de densidad semicontinua y continua que son generados típicamente por el entrenador de modelos acústicos de Sphinx.
- Un modelo predeterminado de lenguaje que acepta dos tipos de lenguajes: grafo de estados finito y modelos N-grama.

Sphinx tienen un vocabulario variable de forma continua. Esto permite distintas complejidades de diálogo para ajustarse a las necesidades cambiantes de la interacción vocal humano-ordenador.

3.3. Reconocimiento de Gestos

El tercer tipo de interacción a explicar es el reconocimiento de gestos restringido al ámbito de los hogares inteligentes.

El proceso de comunicación con gestos manuales puede ser descrito como sigue. Los gestos se originan como un concepto mental de la persona que hace el gesto. Luego, son expresados mediante el movimiento de los brazos y manos. Finalmente, las personas receptoras perciben los gestos como transmisiones de imágenes visuales que interpretan usando el conocimiento que poseen sobre estos gestos.

El reconocimiento de gestos requiere de que configuraciones estáticas y/o dinámicas de la mano, brazo e incluso de otras partes del cuerpo humano sean medidas por una máquina [28]. Hay dos formas de conseguirlo:

- Usar dispositivos hápticos que miden directamente la posición espacial y los ángulos de las articulaciones.
- Usar técnicas de interacción sin contacto basadas en vídeo. Esta aproximación usa un conjunto de videocámaras y técnicas de visión por computador para interpretar los gestos.

Usar dispositivos hápticos fue la primera aproximación que se usó para el reconocimiento de gestos. Sin embargo, estos dispositivos son demasiado incómodos para el público en general como se puede observar en la figura 23.

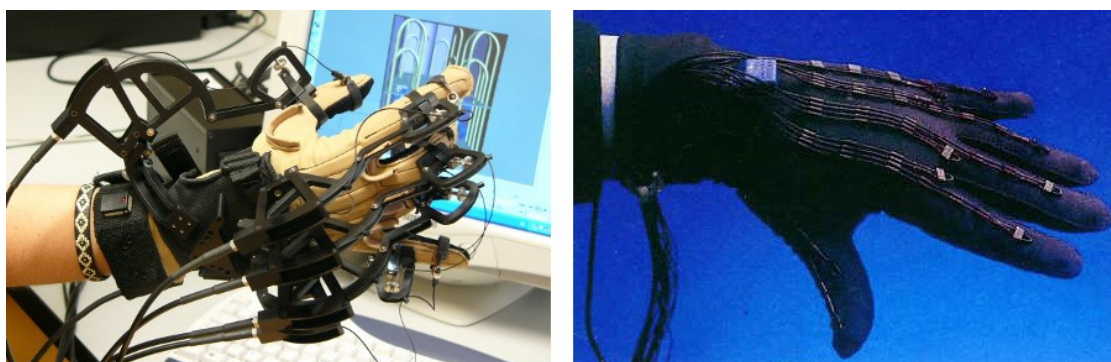


Figura 23. Ejemplos de dispositivos hápticos

La forma de resolver este inconveniente fue seguir otra aproximación: usar técnicas de interacción sin contacto basadas en vídeo. Esta última aproximación es la que se va a explicar a continuación.

La figura 24 muestra un esquema del funcionamiento de un sistema de interpretación de gestos basado en la visión. En resumen, primero se elige un modelo de gestos que usará el sistema

para el reconocimiento de gestos. Luego, de las imágenes capturadas mediante sistemas de captura de vídeo, como por ejemplo las cámaras de vídeo, se capturan las imágenes y se identifica a la persona en la imagen. Después, de la persona identificada se extraen las características significativas para el modelo elegido anteriormente y se estiman los parámetros a partir de las características extraídas. Por último, los parámetros son clasificados en la fase de reconocimiento y se predice cuál es el gesto realizado. A continuación, se explica con más detalle el proceso.

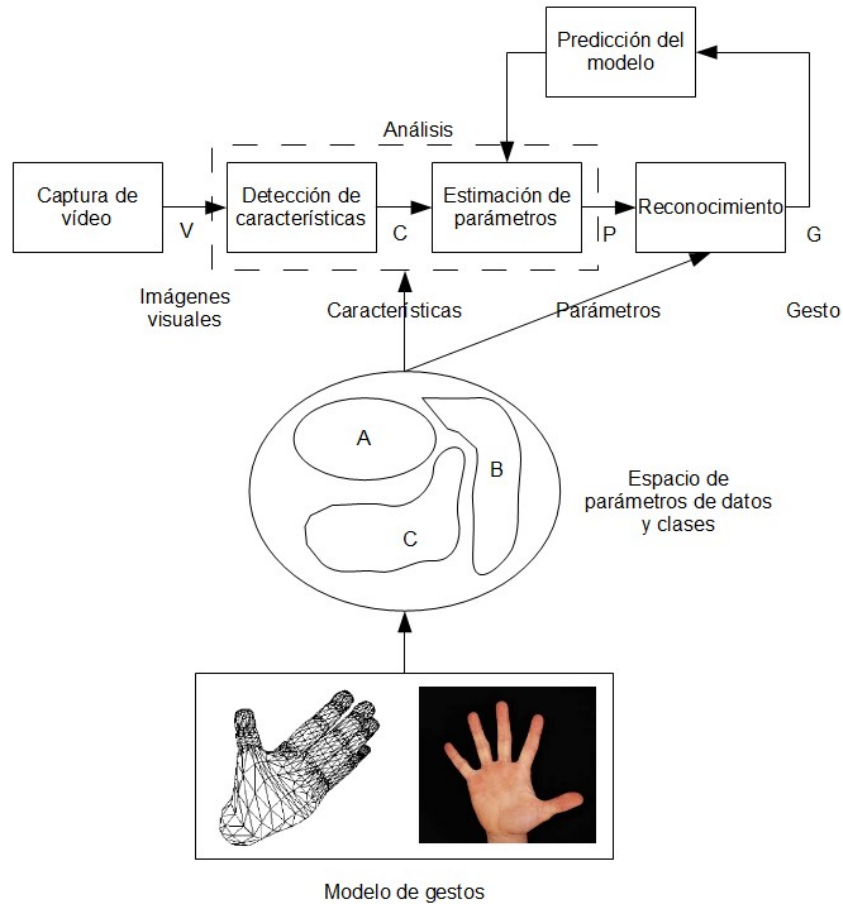


Figura 24. Esquema del sistema de interpretación de gestos basado en la visión

Los gestos son observados como movimientos de las manos y brazos en un espacio 3D. Así, la descripción de los gestos implica la caracterización de sus propiedades espaciales y temporales. Primero, se elige un modelo del gesto. Esta descripción es lo que forma el modelo espacial y temporal de un gesto en concreto. El modelo espacial puede ser tan simple como una foto de una mano o puede ser tan complejo como un modelo 3D de una mano. El modelo temporal lo forma una sucesión de fotos donde se puede observar el cambio de las manos y brazos a lo largo del tiempo.

El primer paso en la interpretación de gestos basado en la visión es elegir un modelo del gesto. Para el modelado espacial de gestos hay dos aproximaciones principales:

- Modelos basados en un modelo 3D de la mano: estos modelos usan modelos articulados de la mano y brazo humanos para estimar los parámetros de movimiento de la mano y el brazo. Más adelante, estos movimientos son reconocidos como gestos. Los parámetros de esta aproximación son los ángulos de articulaciones y la posición de la palma de la mano.
- Modelos basados en la apariencia: estos modelos enlazan directamente la apariencia de los movimientos de la mano y brazo con imágenes visuales para especificar los gestos. Los parámetros de esta aproximación son imágenes, parámetros de geometría de la imagen,

parámetros de movimiento de la imagen, movimiento y la posición de las puntas de los dedos.

A continuación, se analizan las imágenes visuales de una o varias transmisiones de entrada de vídeo para extraer los parámetros del modelo. Esta fase se divide en dos fases que siguen el orden correspondiente: detección de características y estimación de parámetros.

En el proceso de detección, la persona que realiza los gestos es extraída del resto de la imagen. Normalmente se usan dos tipos de señales para este proceso:

- Señales de color: son aplicables porque el color de la piel humana es normalmente más diferenciable y menos sensible a los cambios de iluminación en el espacio de saturación de color que en el espacio de color estándar RGB. La principal desventaja es la variabilidad del color de piel en las diferentes condiciones de iluminación, lo que puede hacer indetectable al usuario.
- Señales de movimiento: estas señales asumen que una única persona es la que hace los gestos y que está estacionaria con respecto al fondo (también estacionario). Con estas asunciones, es fácil detectar el movimiento de la mano y brazo y, de esta forma, localizar a la persona. Sin embargo, estas mismas asunciones son sus desventajas.

Luego, se extraen características significativas de la persona para el modelo elegido anteriormente. Algunas de las características que pueden ser útiles son las puntas de los dedos y las siluetas o contornos de manos y brazos. En el caso de que el modelo sea del tipo basado en la apariencia también podrían ser características interesantes las propias imágenes que muestran las manos y brazos o a la propia persona.

En el proceso de estimación de parámetros, se estiman los parámetros del modelo a partir de las características extraídas anteriormente. Este proceso será distinto dependiendo de modelado espacial que se haya decidido usar.

Finalmente, después de la fase de análisis, se recogen los parámetros extraídos y se introducen como entrada en la fase de reconocimiento de gestos. En esta fase, los parámetros son clasificados e interpretados según el modelo aceptado. El reconocimiento también puede influenciar en la fase de análisis prediciendo el modelo gestual en la siguiente instancia de tiempo. La evaluación de una aproximación de reconocimiento de gestos en particular se hace con la precisión, robustez, velocidad y el número de gestos que cubre.

3.3.1. Ejemplo de uso

Daehwan Kim y Daijin Kim proponen un esquema de detección hacia adelante que realiza la segmentación y el reconocimiento de gestos a la vez [24]. Explican que hay dos tipos de gestos dependiendo de su intención que son los gestos naturales, que no son significativos, y los gestos artificiales, que pueden expresar significados más detallado y variados usando movimientos predefinidos.

En este trabajo, se aplica el método de detección al control de cortinas e iluminación de un hogar inteligente usando gestos de la parte superior del tronco humano. Para ello, se extraen los datos de la forma 2D de los gestos del cuerpo humano y los datos 3D de las articulaciones de los marcadores. A continuación, construyen un mapeado asociativo desde los datos de la forma 2D hasta los datos 3D de las articulaciones.

El hogar inteligente tiene tres cámaras que están sujetas al techo en ángulos de 0° y $\pm 45^\circ$. En la figura 25 se muestra el entorno de hogar inteligente que ha sido usado.



Figura 25. Entorno de un hogar inteligente

Se definieron ocho gestos para controlar la apertura y cierre de las cortinas, y el encendido y apagado de las luces. En la figura 26 se muestran los ocho gestos.



Figura 26. Gestos para el control de cortinas y luces

Se hicieron pruebas para calcular la precisión del sistema de reconocimiento de gestos propuesto. Esta precisión venía determinada por las secuencias de gestos reconocidos correctamente sobre el número total de secuencias de gestos probadas. La precisión fue de un 95.42%.

3.4. Interacción Sensible al Contexto

El cuarto tipo de interacción a explicar es la interacción sensible al contexto restringida al ámbito de los hogares inteligentes.

La sensibilidad al contexto se refiere a la idea de que los ordenadores pueden sentir y reaccionar a los cambios de su entorno [29][32]. Por ejemplo, un teléfono móvil sensible al contexto puede saber que actualmente está en una sala de reuniones, y que el usuario se ha sentado. De esta forma, el teléfono puede concluir que el usuario está actualmente en una reunión y rechazar

llamadas que no sean importantes.

Los sistemas sensibles al contexto se adaptan a la localización del usuario, grupo de gente cercana, servidores, dispositivos accesibles, al igual que a los cambios sufridos por dichas cosas durante el tiempo. Un sistema con estas capacidades puede examinar el entorno de computación y reaccionar a los cambios. El contexto incluye la localización del usuario, iluminación, nivel de ruido, conectividad de red, costes de comunicación, ancho de banda de comunicación e incluso la situación social.

Los sistemas sensibles al contexto deben preocuparse de la adquisición del contexto, la abstracción y comprensión del mismo, y aplicar el comportamiento adecuado al contexto reconocido.

El nivel de interactividad de las aplicaciones sensibles al contexto varía mucho, desde dejar que el usuario defina manualmente los parámetros de cómo una aplicación debería comportarse hasta proporcionar automáticamente al usuario servicios e información que el desarrollador crea relevantes [2]. Hay tres tipos de interactividad:

- Personalización: la aplicación permite al usuario especificar sus propios ajustes para indicar cómo debe comportarse la aplicación en una situación determinada. Es una característica común en las aplicaciones de ordenador. Un ejemplo es cambiar el fondo de escritorio y los iconos que se quieren mostrar en el escritorio del ordenador.
- Sensibilidad al contexto pasiva: la aplicación presenta el contexto actualizado al usuario pero deja que éste decida cómo cambiar el comportamiento de la aplicación.
- Sensibilidad al contexto activa: la aplicación cambia de forma autónoma su comportamiento conforme a la información recogida por los sensores.

3.4.1. Ejemplo de Uso

UbiHome (Ubiquitous Home) es un proyecto consistente en proporcionar un entorno para realizar pruebas de tecnologías de computación empotrada para hogares inteligentes [22].

UbiHome proporciona entornos de computación distribuidos donde los contextos son directamente intercambiados entre los ubiSensores y ubiServicios sin ningún servidor centralizado que gestione el contexto. El conflicto de usuarios es una situación en la que algunos usuarios acceden al mismo servicio a la vez. Cuando ocurre el conflicto, el ubiServicio reaccionará al usuario que tenga la prioridad más alta. UbiHome proporciona un servicio personalizado que de forma autónoma toma la acción apropiada usando la situación del individuo en vez de condiciones generales aplicables a todos los usuarios. Esto es porque los usuarios con una GUI son capaces de especificar su contexto condicional para activar un servicio individual.

UbiUCAM (Unified Context-aware Application Model) es un modelo donde la información está centrada en el usuario en términos de 5W1H. Esto indica "un usuario en concreto (Who) está", "en una localización determinada (Where)", "en un instante de tiempo (When)", "prestando atención a un cierto objeto/servicio (What)", "representando una expresión con señales físicas (How)", o "a causa de una intención o emoción en concreto (Why)".

Los ubiSensores y ubiServicios están basados en ubiUCAM. El ubiSensor es el responsable de detectar el cambio en la situación de los usuarios o entornos y luego proporciona contextos preliminares en la forma de 5WH1 a todos los ubiServicios. A continuación se explican dos ejemplos de ubiSensores:

- UbiKey: usa memoria portátil para generar la identidad del usuario (Who) y el tiempo de entrada/salida (When). Cada usuario lleva la ubiKey que contiene el perfil del usuario

como identificador y lista de sus servicios favoritos.

- UbiFloor: sensores on/off son agregados en cada espacio de 2cm*5cm, determina la posición del usuario (Where) sintiendo los sensores presionados. Detecta la identidad del usuario analizando su patrón de andadura (Who), y también genera el tiempo (When).

En ubiHome, los ubiServicios proporcionan a los residentes servicios personalizados acordes con el contexto del usuario. Un ejemplo de ubiServicio es un reproductor multimedia basado en contexto (c-MP, context-based Media Player). C-MP es un reproductor multimedia que proporciona servicios centrados en el usuario usando el contexto del identificador de dicho usuario (Who), la localización del usuario (Where), tiempo (When), gestos (How), objetos para la reproducción de vídeos (What) y la intención del usuario para controlar la reproducción de vídeo (Why). Por ejemplo, después de que un residente entre en el salón con la ubiKey, se sienta en el sofá enfrente de la televisión. Luego, un menú de ubiServicio aparece automáticamente en el monitor. Si el residente selecciona reproducción de vídeo en el menú, el c-MP muestra una lista de títulos de películas con el historial del usuario. Cuando el residente se levanta del sofá, c-MP pausa automáticamente la película. Si vuelve y se sienta antes de que pasen 30 segundos, c-MP continúa la película. Si no vuelve en 30 segundos o sale de ubiHome, c-MP guarda el estado de pausado y tiempo de la película y la para automáticamente.

3.5. Interacción Multimodal

El último tipo de interacción a explicar es la interacción multimodal restringida al ámbito de los hogares inteligentes.

Las interfaces gráficas permiten la presentación paralela de datos al usuario y un acceso rápido a información relevante usando interfaces que usan punteros como el ratón o el dedo [30]. Por otra parte, en los terminales móviles, el reconocimiento de habla ayuda al usuario a superar la limitación de otras modalidades de entrada. Además, el habla puede ser usado también en situaciones donde los ojos y/o las manos están ocupadas, como por ejemplo mientras se conduce un coche.

El habla y el GUI pueden ejecutarse en paralelo o permitir operaciones suplementarias.

La aproximación más simple permite solo una modalidad de interacción a la vez, y no requiere sincronización entre las distintas modalidades. La otra aproximación permite distintas modalidades a la vez, pero necesita una sincronización semántica por parte del usuario. Un ejemplo de sincronización semántica es seleccionar un objeto en la pantalla y pedir, usando reconocimiento de habla, "Dame más información sobre este objeto".

3.5.1. Ejemplo de Uso

Ruser, Borodulkin y Leisner proponen una interfaz multimodal que permite interactuar con el hogar inteligente mediante la voz o mediante representaciones gráficas 2D y 3D de la casa [31]. Estas interfaces están implementadas en diferentes dispositivos, tanto estacionarios como móviles. Para probar la interfaz multimodal, han usado SmartHOME Research Lab, que es un hogar familiar del campus universitario. Tanto SmartHOME Research Lab como algunas vistas de las interfaces se muestran en la figura 27.



Figura 27. Vista de SmarthHOME Research Lab y de los prototipos de interfaces gráficas de hogar inteligente en dispositivos móviles

La interfaz gráfica 2D sigue la aproximación WIMP y requiere poca potencia de computación, con lo que puede ser usada en dispositivos móviles como PDAs.

Con el fin de aumentar la facilidad de uso, se diseñó un entorno gráfico 3D virtual del hogar inteligente. El modelo visual de la casa y sus habitaciones con los sensores y actuadores asignados sustituye la aproximación WIMP. Para usar esta interfaz, el usuario puede clickar con un dedo o puntero en la pantalla táctil.

Para el control hablado, se persigue la detección de voz, el procesamiento del habla y la generación del habla. Con el fin de garantizar la mejor interacción con el usuario, se usa una base de datos con sintaxis y órdenes limitadas.

Después de cargar el software de visualización del servidor Web en el terminal del hogar, es creada una interfaz gráfica como se muestra en la figura 28. Un menú emergente (área 1) muestra la estructura jerárquica de todas las habitaciones para facilitar la selección de habitaciones. En el área 2 se puede establecer el ángulo de inclinación y el nivel de transparencia. Los iconos del área 3 pueden ser activados para mostrar otros menús, como las últimas llamadas telefónicas realizadas. El área 4 muestra diferentes datos ambientales como la humedad, temperatura y el número de personas en la habitación. El área 5 muestra información contextual adicional como recomendaciones y advertencias. El área 6 es un menú de controles para controlar dispositivos como luces y persianas.

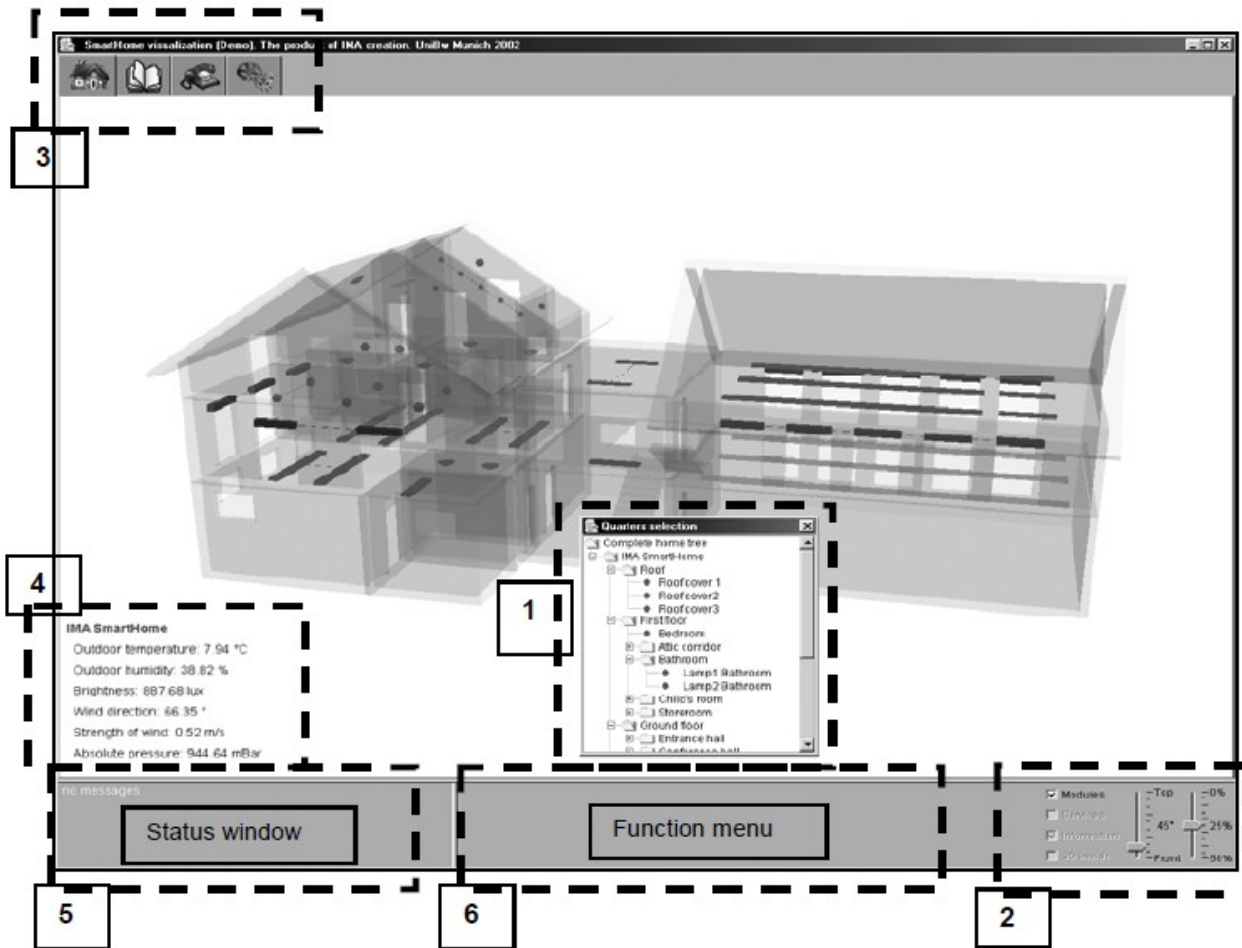


Figura 28. Software de interfaz gráfica 3D

El usuario también tiene la posibilidad de usar órdenes de voz y recibir mensajes de voz. En muchos casos, la interacción hablada puede sustituir tanto la manipulación con la pantalla táctil como la interfaz WIMP. Para la interacción hablada, el usuario nombra un objeto de la habitación, especifica el dispositivo y la acción a realizar (ej: "por favor, enciende la luz" o "dime la temperatura exterior"). Como resultado, la acción es realizada y la información deseada es transmitida por el generador de habla del sistema. Al mismo tiempo, el sistema actualiza la visualización gráfica acorde con la orden del usuario.

Esta aproximación ofrece distintas formas de interactuar con la vivienda, entre ellas la visualización 3D. Incluso se puede interactuar con ella desde los dispositivos móviles. Sin embargo, la interfaz de la representación 3D puede ser demasiado grande para un dispositivo móvil, siendo difícil, por ejemplo, seleccionar una habitación.

3.6. Conclusiones

Una vez descritas las aproximaciones existentes para el control de hogares inteligentes, se va a explicar la opción presentada por esta tesina.

3D Home es una aplicación móvil que permite controlar un hogar inteligente mediante una representación virtual de una vivienda, haciendo intuitivo su uso. El control se realiza interactuando con los diversos elementos del hogar virtual. Además, los cambios que se realizan en la vivienda inteligente, quedan reflejados de forma visual en la aplicación.

En 3D Home se muestra una representación 3D virtual de una vivienda que incluye elementos con los que el usuario puede interactuar. Por esta razón se ha decidido usar un motor de escenarios 3D interactivos para el desarrollo de 3D Home. El motor usado es Unreal Development Kit Mobile que permite crear aplicaciones para dispositivos móviles que soporten iOS como iPhone, iPod Touch! e iPad.

Capítulo 4

Proceso de Desarrollo de 3D Home

Para desarrollar 3D Home se usa un motor de escenarios 3D interactivos porque permite la representación de una vivienda en 3D. El motor utilizado es Unreal Engine 3, que forma parte del entorno de desarrollo Unreal Development Kit Mobile (UDK Mobile). UDK Mobile es una aplicación gratuita que, ofrece diferentes herramientas que han sido utilizadas para el desarrollo de 3D Home. Estas herramientas son editores gráficos para crear la vivienda y el comportamiento de los elementos de su interior. Además, también ofrece un lenguaje de programación que se puede usar para añadir o modificar elementos de UDK Mobile.

El resto del capítulo está organizado como sigue: primero, la sección 4.1 describe de forma general el proceso de desarrollo de una casa en UDK Mobile. A continuación, la sección 4.2 describe de forma detallada el proceso de desarrollo de 3D Home. Luego, la sección 4.3 explica los cambios que ha sufrido la aplicación desde su versión inicial. Después, la sección 4.4 describe el problema de flexibilidad que tiene 3D Home y una propuesta para tratar este problema. Por último, la sección 4.5 concluye el capítulo.

4.1. Visión General del Proceso de Desarrollo en UDK Mobile

Para crear una vivienda en UDK Mobile se sigue un proceso parecido al que se sigue cuando se construye una vivienda real. Primero se define cómo va a ser la vivienda, después se construye y por último se entrega la vivienda al usuario. En la figura 29 se muestra este proceso.

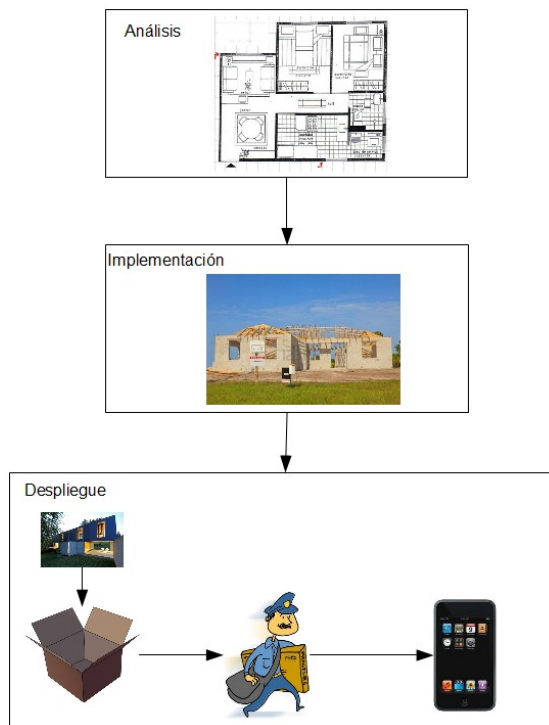


Figura 29. Proceso de creación de una vivienda en UDK Mobile

El resto de la sección está organizado como sigue: primero, la sección 5.1.1 explica la fase de análisis. Luego, la sección 5.1.2 describe la fase de implementación. Por último, la sección 5.1.3 explica la fase de despliegue.

4.1.1. Fase de Análisis

La primera fase del proceso es la fase de análisis. En esta fase se define la forma de la casa y la distribución de las habitaciones y los muebles en, por ejemplo, un plano como se muestra en la figura 30.

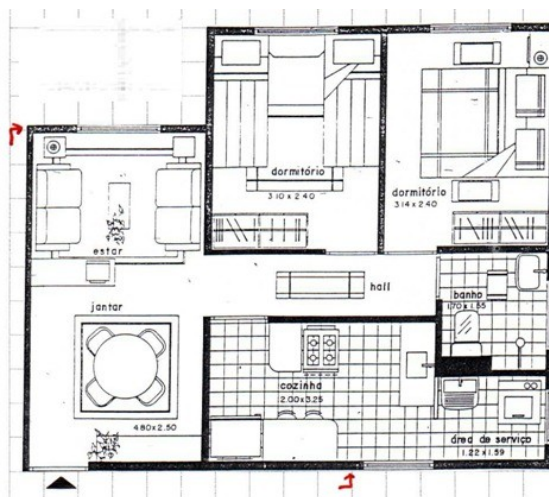


Figura 30. Plano de una casa

Además, se pueden definir los dispositivos de interacción y su comportamiento. En la figura 31 se muestra un ejemplo de definición del comportamiento.

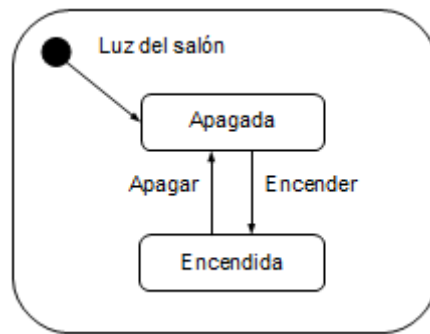


Figura 31. Comportamiento de la luz del salón

En la fase de análisis, también se pueden describir las interfaces con las que el usuario controla los elementos de interacción. En la figura 32 se muestra un ejemplo de definición de interfaz.

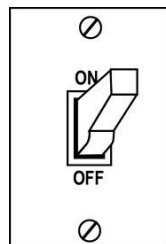


Figura 32. Interfaz de la luz del salón

4.1.2. Fase de Implementación

La segunda fase del proceso es la fase de implementación. En esta fase se usan las distintas herramientas de UDK Mobile para crear una vivienda. Para ello se siguen distintos pasos como se muestra en la figura 33.

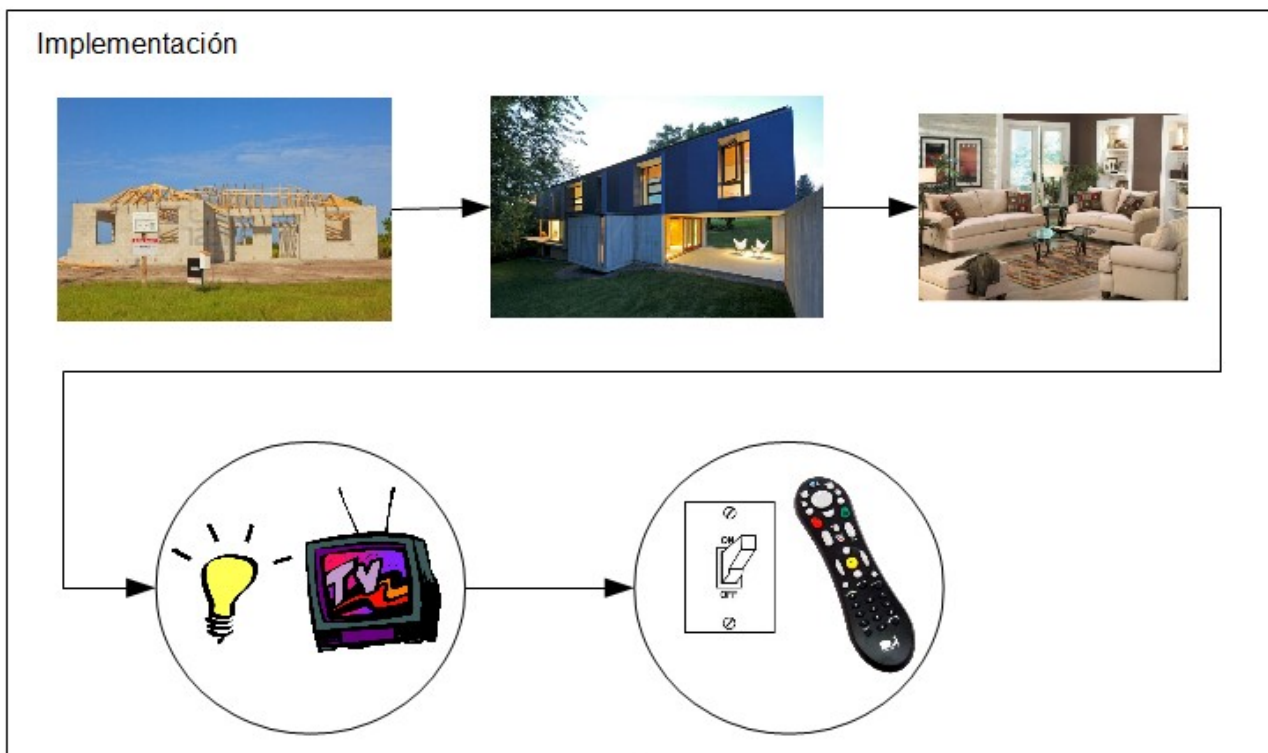


Figura 33. Fase de implementación

El orden mostrado en la figura 33 solo es una propuesta. La flexibilidad de UDK Mobile

permite seguir cualquier orden e incluso combinar pasos. Sin embargo, el orden ilustrado en la figura 33 facilita la comprensión del proceso de implementación de una vivienda. Los pasos seguidos en la figura 33 son los siguientes:

- El primer paso es crear la estructura de la vivienda. En este paso se define la forma de la vivienda, cuántas habitaciones tiene y cómo están distribuidas.
- El segundo paso es pintar la vivienda. En este paso se define el material y color que tendrán las paredes, techo y suelo de las distintas habitaciones.
- El tercer paso es amueblar la vivienda. En este paso se colocan los muebles y dispositivos como lámparas y televisiones de las distintas habitaciones. En este paso, los elementos puestos son estáticos, esto quiere decir que no sufren ningún tipo de cambio como moverse o cambiar de color. Así, por ejemplo, las lámparas colocadas no se encienden.
- El cuarto paso es definir el comportamiento de los elementos. Por ejemplo, que una lámpara ilumine una habitación o que una persiana suba y baje.
- El quinto y último paso es crear las interfaces que permitirán al usuario controlar los dispositivos de la casa. Por ejemplo, un interruptor para apagar y encender la luz de una habitación.

4.1.3. Fase de Despliegue

En esta última fase, se prepara la aplicación para ser enviada e instalada en un dispositivo móvil. Sin embargo, para poder realizar esta fase se necesita tener una licencia de desarrollador de iOS.

4.2. Etapas del Desarrollo de 3D Home

En esta sección se describen los pasos realizados para desarrollar 3D Home.

El resto de la sección está organizado como sigue: primero, la sección 4.2.1 describe la fase de análisis de 3D Home. Luego, la sección 4.2.2 describe la fase de implementación de 3D Home. Por último, la sección 4.2.3 explica la fase de despliegue de 3D Home.

4.2.1. Análisis

La primera fase es la fase de análisis, aquí se definió la distribución de las habitaciones de la vivienda, qué dispositivos iban a ser de interacción y el comportamiento de dichos dispositivos. En esta sección no se va a explicar ni el comportamiento de los dispositivos ni las interfaces porque se explican en el capítulo 5.

El resto de la sección está organizado como sigue: en la sección 4.2.1.1 se explica la distribución de habitaciones y en la sección 4.2.1.2 se describen los dispositivos de interacción.

4.2.1.1. Definición del Plano

Lo primero que hay que hacer para crear un escenario es definir el escenario que se pretende diseñar. En el caso de esta tesina, se pretende diseñar una vivienda. Así que se dibujó un plano de la vivienda que se pretendía modelar.

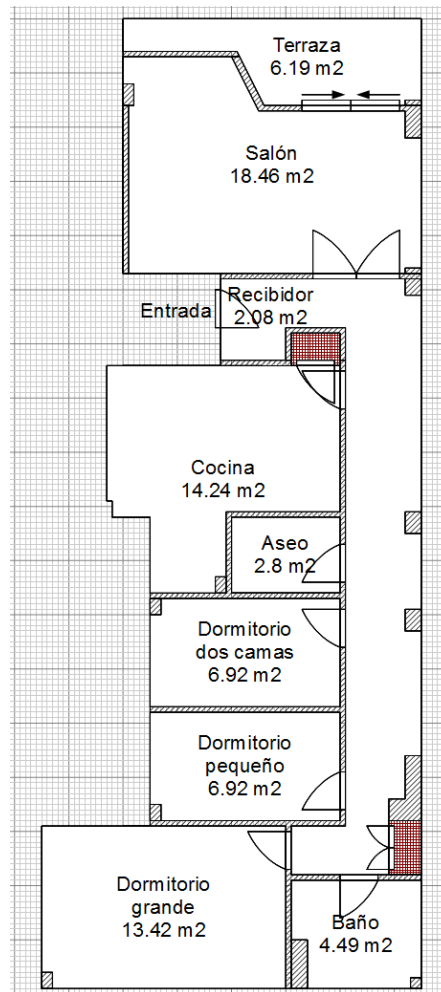


Figura 34. Plano de la vivienda

El plano mostrado en la figura 34 es la idea original de lo que se quiere crear. Durante la creación del escenario se han hecho cambios menores en la vivienda con el objetivo de mejorar la usabilidad. Por ejemplo, se vio que los huecos de las puertas eran muy estrechos y hacían que a veces hubiera problemas para entrar o salir de una habitación, así que se hicieron más anchos.

4.2.1.2. Definición de los elementos de interacción

Algunos de los elementos de interacción que hay en 3D Home se corresponden con los dispositivos que se quieren controlar de la maqueta. No todos los dispositivos de la maqueta son controlables porque hay algunos que tienen el mismo tipo de control. Por ejemplo, hay más de una luz conmutable en la maqueta y solo se controla una de ellas.

Hay un dispositivo de interacción de 3D Home que no controla el dispositivo correspondiente en la maqueta. Éste es la alarma, y la razón principal para hacer este dispositivo era explorar otras opciones como la reproducción de sonido.

En la tabla 4 se muestran todos los elementos de interacción que hay en 3D Home, en qué habitación están y qué dispositivos son los que responden ante la interacción. Esta tabla además muestra qué dispositivo de la maqueta controla cada elemento de interacción.

Habitación	Elemento de interacción	Dispositivo objetivo	Maqueta
Aseo	Lámpara	Lámpara de esta habitación	-
Aseo	Interruptor	Lámpara de esta habitación	-
Baño	Lámpara	Lámpara de esta habitación	-
Baño	Interruptor	Lámpara de esta habitación	-
Cocina	Lámpara	Lámpara de esta habitación	-
Cocina	Interruptor	Lámpara de esta habitación	-
Dormitorio dos camas	Lámpara gradual	Lámpara de esta habitación	Luz gradual
Dormitorio dos camas	Interruptor	Lámpara de esta habitación	Luz gradual
Dormitorio grande	Lámpara	Lámpara de esta habitación	-
Dormitorio grande	Interruptor	Lámpara de esta habitación	-
Dormitorio pequeño	Lámpara	Lámpara de esta habitación	-
Dormitorio pequeño	Interruptor	Lámpara de esta habitación	-
Pasillo	Lámpara 1	Todas las lámparas del pasillo	-
Pasillo	Lámpara 2	Todas las lámparas del pasillo	-
Pasillo	Lámpara 3	Todas las lámparas del pasillo	-
Pasillo	Lámpara 4	Todas las lámparas del pasillo	-
Pasillo	Interruptor 1	Todas las lámparas del pasillo	-
Pasillo	Interruptor 2	Todas las lámparas del pasillo	-
Pasillo	Interruptor 3	Todas las lámparas del pasillo	-
Pasillo	Interruptor 4	Todas las lámparas del pasillo	-
Recibidor	Alarma	Alarma	-
Salón	Lámpara	Lámpara de esta habitación	Luz conmutable
Salón	Interruptor cerca de la entrada	Lámpara de esta habitación	Luz conmutable
Salón	Aire acondicionado	Aire acondicionado	Ventilador
Salón	Interruptor cerca del aire acondicionado	Aire acondicionado	Ventilador
Salón	Ventana que enmarca la persiana	Persiana	Persiana

Tabla 4. Elementos de interacción de 3D Home

4.2.2. Implementación

En esta sección se describen los pasos realizados para implementar 3D Home. Además, también describe las herramientas usadas en estos pasos. Estas herramientas son ofrecidas por UDK Mobile.

El resto de la sección está organizado como sigue: primero, la sección 5.2.2.1 describe la creación del esqueleto de la vivienda. Luego, la sección 5.2.2.2 explica la aplicación de materiales sobre el esqueleto previamente hecho. A continuación, la sección 5.2.2.3 describe la colocación de elementos en la vivienda. Después, la sección 5.2.2.4 explica la implementación de las interfaces y del comportamiento de los dispositivos de 3D Home. Finalmente, la sección 5.2.2.5 explica el desarrollo de otras funcionalidades cuyo requisito ha sido escribir líneas de código.

4.2.2.1. Creación de la Estructura

Una vez se tiene una idea clara de lo que se quiere hacer ya se puede empezar con el primer paso de la fase de implementación: crear la estructura de la vivienda. Para ello se usa la herramienta UnrealEd que permite crear visualmente distintos entornos como, en nuestro caso, una vivienda.

Lo primero que se ha hecho es crear la geometría del escenario usando los pinceles BSP y herramientas que ofrece UnrealEd para personalizar estos pinceles. BSP es el esqueleto del nivel y forma la mayoría de las piezas grandes geométricas del escenario. Los pinceles BSP son los bloques de construcción más básicos de Unreal.

A continuación se muestran imágenes sacadas en tiempo de diseño, como el resto de imágenes de esta sección, de 3D Home solo con la geometría creada.

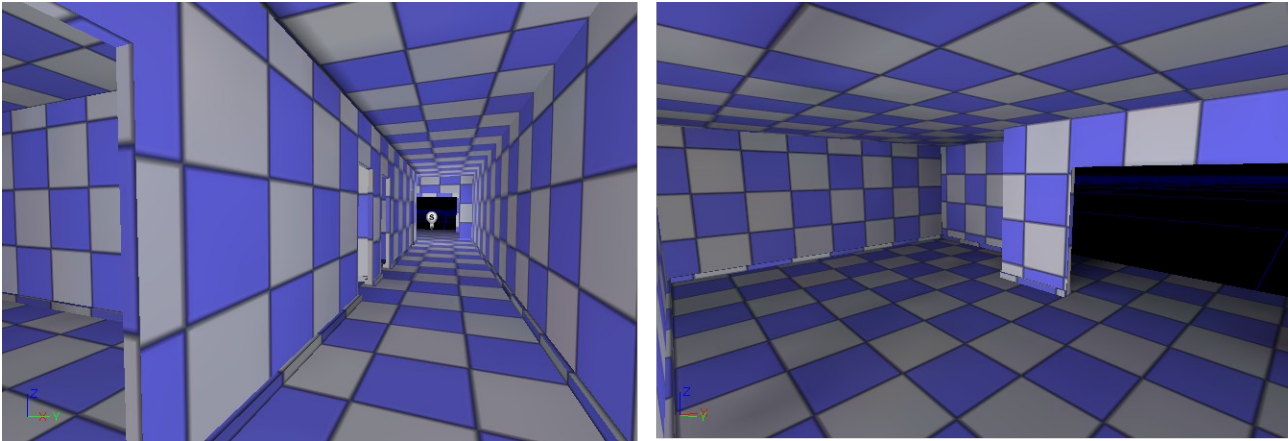


Figura 35. Geometría del pasillo (izq.) y del salón (der.)

En la figura 35 se pueden ver las paredes, el techo y el suelo. También están los huecos de las puertas y, en el suelo, se puede ver la geometría de los rodapiés. Por último, en el fondo del pasillo se puede ver la posición de una fuente de luz, puesta para poder ver el escenario en tiempo de ejecución mientras se hacen las pruebas.

4.2.2.2. Aplicación de Materiales

Una vez se ha creado la estructura de la vivienda, el siguiente paso es definir los materiales y colores que se aplicarán a la estructura del escenario. Para este paso también se usa la herramienta UnrealEd.

Un material es una pintura que se aplica a las superficies de los objetos del escenario. Toda superficie visible que se crea tiene un material aplicado, y los materiales contienen al menos una textura. Las texturas son imágenes 2D, creadas con herramientas externas de edición de imágenes digitales como Gimp o Photoshop. Ejemplos de texturas son la textura difusa, que pone el color al objeto, y el mapa normal, que hace que el objeto parezca mucho más complejo de lo que realmente es.

En la figura 36 se muestra una imagen del salón donde se puede ver que se han aplicado dos texturas diferente. Una para el suelo y rodapiés que simula un parquet, y otra para las paredes y el techo.



Figura 36. Salón con las texturas aplicadas

4.2.2.3. Colocación de Elementos

Después de la aplicación de los materiales, se colocan los elementos deseados en el escenario. Estos elementos pueden ser de varios tipos como objetos 3D, luces y bloques de colisión. Para este paso también se usa la herramienta UnrealEd.

Los objetos 3D son construidos con mallas poligonales (colección de vértices, aristas y caras que definen la forma de los objetos). Las mallas estáticas son un tipo de mallas poligonales que consumen poco tiempo de procesador y pueden ser usadas para crear formas muy complejas, sin embargo no se pueden animar. Esta animación también incluye el cambio de su material y los cambios de iluminación recibidos en tiempo de ejecución. Además, las mallas estáticas pueden ser instanciadas múltiples veces sin requerir tiempo de CPU porque las dibuja la tarjeta de vídeo.

Se han usado mallas estáticas para los objetos de todo el escenario con excepción de los del salón que son InterpActors que son objetos que toman como base a las mallas estáticas y pueden ser animados. Los objetos del salón son InterpActors porque reciben la iluminación de una luz gradual que es una luz que cambia su intensidad en el tiempo de ejecución. Los modelos usados para las mallas estáticas e InterpActors no se han creado específicamente para este trabajo, se han cogido de [21].

En la figura 37 se puede ver todos los elementos añadidos en el salón. Hay objetos 3D como los sofás, el televisor y la lámpara. También hay una luz que está situada justo detrás de la lámpara.



Figura 37. Sal3n con los objetos a3nadiridos

Las 3reas de colisi3n de los objetos, en UDK Mobile, normalmente se pueden generar de forma autom3tica resultando en muchos de ellos en un cubo que rodea al objeto. Para crear el 3rea de colisi3n de los sof3s, se ha preferido usar bloques de colisi3n porque no se quer3a un cubo que impidiera al usuario subirse al sof3. Estos bloques de colisi3n est3n representados por cubos con los bordes rosas. Con estos bloques de colisi3n, se puede definir un 3rea de colisi3n compleja de forma sencilla.

4.2.2.4. Implementación del Comportamiento

El siguiente paso a realizar es definir las interfaces y el comportamiento de los elementos de interacción. Es decir, se ha juntado los pasos 4 y 5 de la fase de implementación mostrada en la figura 33.

Para implementar las interfaces y el comportamiento se ha usado Unreal Kismet que es una herramienta de UDK Mobile que permite definir cómo el usuario puede interactuar con un objeto y cómo debe comportarse el objeto ante dicha interacción. El comportamiento también incluye cambios visuales del entorno, estos cambios son llamados animaciones. Para la creación de animaciones, se usa Unreal Matinee que también es una herramienta de UDK Mobile. Estas animaciones no son solo rotación y traslación de objetos, también incluyen los cambios de iluminación y materiales.

Unreal Kismet

Unreal Kismet permite definir el comportamiento de forma visual mediante modelos Kismet.

Un modelo Kismet se puede ver como una red de módulos, conocidos como objetos de secuencia, conectados por cables. Para entender mejor qué es Unreal Kismet, se va a explicar el modelo mostrado en la figura 38. Este modelo es una versión simplificada del modelo de la luz del salón.

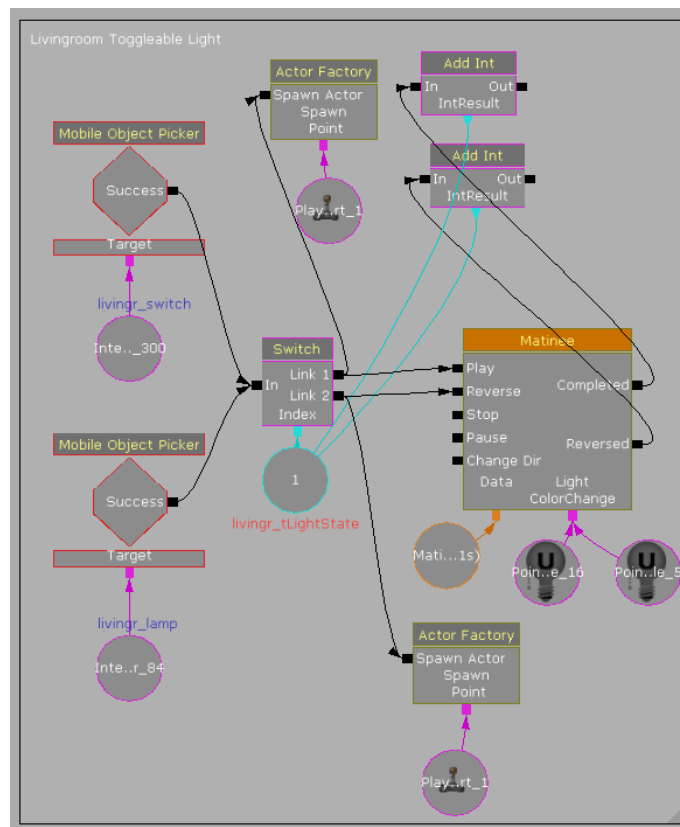


Figura 38. Modelo Kismet de la luz del salón

En la figura 38 se pueden observar todos los tipos de objetos de secuencia que tiene Unreal Kismet:

- Eventos: son objetos que crean una "entrada" en el modelo, probablemente desde el usuario. Son representados por un diamante rojo. Un ejemplo de evento es Mobile Object Picker.

- Acciones: son objetos que realizan alguna acción en el escenario. Son representados por una caja con entradas a la izquierda, salidas en la derecha y conexiones de variables en la parte inferior. Ejemplos de acciones son Actor Factory y Add Int.
- Condiciones: son objetos que afectan al control del flujo del modelo. Un ejemplo de condición es Switch.
- Variables: son objetos que almacenan información de un determinado tipo para usar en un evento, acción o condición. Son representados por un círculo coloreado donde el color depende del tipo de variable. Todos los círculos que hay en la figura 38 son variables donde los morados son de tipo referencias a objetos, el azul es de tipo número entero y el naranja es de tipo objeto Matinee.
- La secuencia Matinee: crea una nueva secuencia Matinee.

A continuación se explica el modelo: hay varios elementos del salón que son susceptibles de ser tocados (un evento Mobile Object Picker por cada elemento). Estos elementos son el interruptor cerca de la entrada (referencia a livingr_switch) y la lámpara (referencia a livingr_lamp). Cuando se toca cualquiera de estos elementos, se entra en el switch y dependiendo del estado actual de la luz (variable livingr_tLightState) hará las siguientes acciones:

- Si el estado actual de la luz es apagada, codificado como 1, hace lo que está conectado en "Link 1", es decir:
 - Por un lado reproduce hacia delante la animación (secuencia Matinee), en términos prácticos enciende la luz. Una vez haya terminado la animación (evento Completed), indica el nuevo estado de la luz en la variable livingr_tLightState (2 para luz encendida).
 - Por otro lado envía el mensaje "tl_on" al servidor (acción Actor Factory) para encender la luz conmutable de la maqueta.
- Si el estado actual es encendida, codificado como 2, hace lo que está conectado en "Link 2", es decir:
 - Por un lado reproduce hacia atrás la animación (secuencia Matinee), en términos prácticos apaga la luz. Una vez haya terminado la animación (evento Reversed), indica el nuevo estado de la luz en la variable livingr_lightState (1 para luz apagada) y después graba la partida.
 - Por otro lado envía el mensaje "tl_off" al servidor (acción Actor Factory) para apagar la luz conmutable de la maqueta.

Unreal Matinee

Unreal Matinee permite definir animaciones de actores que son objetos del escenario que pueden moverse, interactuar con otros actores y afectar el entorno como una persiana o una luz. Las animaciones se crean de forma visual mediante un editor de curvas. En este editor de curvas, se usan fotogramas clave para cambiar las propiedades de los actores a lo largo del tiempo.

Para entender mejor qué es Unreal Matinee, se va a explicar la animación creada para la luz del salón.

Para definir una animación, primero se debe crear un grupo en el que se pondrá la animación. Para esta animación se ha creado una animación de tipo Light Color en el grupo LightColorChange como se muestra en la mitad inferior de la figura 39.

Esta animación cambia la propiedad LightColor de las luces del salón con dos fotogramas clave como se expone en la parte superior de la figura 39.

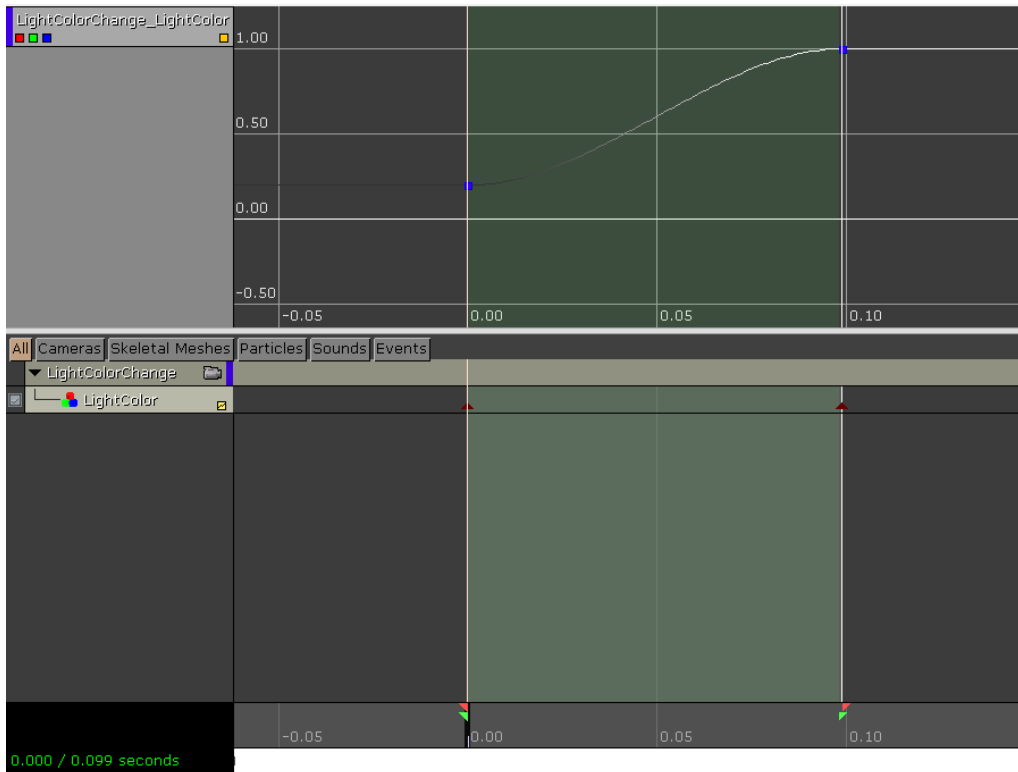


Figura 39. Objeto de secuencia Matinee detallado de Livingroom Toggleable Light

Cada fotograma clave de esta animación almacena tres valores y su codificación es rojo, verde y azul para el primer, segundo y tercer valor. Así, la animación empieza con el color RGB (0.20, 0.20, 0.20) que se corresponde con un gris oscuro y termina con el color RGB (1, 1, 1) que se corresponde con el blanco. Ésta dura 0.099 segundos.

La secuencia Matinee crea un enlace de variable para el grupo definido como se ve en la figura 40 y aplica los cambios especificados en el grupo LightColorChange a las dos luces que hay en el salón.

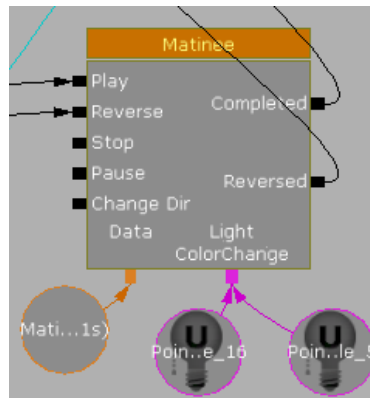


Figura 40. Objeto de secuencia Matinee de Livingroom Toggleable Light

4.2.2.5. Desarrollo de otras funcionalidades

UDK Mobile ofrece la posibilidad de crear nuevos elementos o modificar los ya existentes usando el lenguaje de programación UnrealScript. En este trabajo, se usa este lenguaje de programación para desarrollar funcionalidades que no se pueden hacer con las otras herramientas ofrecidas por UDK Mobile.

Primero se va a explicar lo suficiente de los diagramas de clases para entender los usados en esta sección. El diagrama de clases es uno de los modelos propuestos por UML para describir de forma gráfica las clases del sistema y las relaciones que existen entre ellas. Una clase es una entidad básica que contiene la información de un objeto, junto con sus atributos y operaciones. Los atributos son propiedades de los objetos y las operaciones son servicios que pueden ser requeridos a cualquier objeto de la clase para que muestre un comportamiento. En algunas clases se ha preferido omitir las operaciones para facilitar la explicación.

El resto de la sección está organizado como sigue: primero, la sección 4.2.6.1 describe las clases creadas para la comunicación con la maqueta. Y después, la sección 4.2.6.2 explica las clases creadas para guardar y cargar el estado de la aplicación.

Comunicación con la Maqueta

La primera funcionalidad desarrollada consiste en permitir que 3D Home pueda comunicarse con la maqueta.

La comunicación se realiza a través de un servidor que gestiona la interacción con la maqueta a través de peticiones HTTP recibidas. Una petición HTTP es un mensaje que sigue el protocolo de transferencia de hipertexto. Este protocolo es, en resumen, una serie de reglas que debe cumplir el mensaje para que pueda enviarse por Internet.

En 3D Home, dos clases son las que se encargan de esta comunicación: Factoría y Mensaje. En la figura 41 se describe la relación que hay entre ellas y la información que contienen.



Figura 41. Diagrama de clases de la comunicación con la maqueta

En 3D Home hay tantas instancias de Factoría como acciones distintas se quieran hacer, y cada una de estas acciones tiene asociado un mensaje distinto. Las instancias de Factoría están conectadas a las acciones de los dispositivos con los que se puede controlar la maqueta. Por ejemplo, aunque se usa la misma interfaz para encender y apagar la luz del salón, son dos acciones distintas que dependen del estado actual de la luz. Para la acción de encender la luz hay una factoría asociada y para la acción de apagar la luz hay otra factoría asociada como se muestra en la figura 38.

Cada vez que, en tiempo de ejecución de la aplicación, se activa una instancia de Factoría, ésta crea una instancia de Mensaje. La clase Mensaje se encarga de recoger el mensaje de Factoría, crear una petición HTTP que entienda el servidor de la maqueta, que dependerá del mensaje recibido, y enviar la petición a la dirección del servidor de la maqueta. La dirección está representada por anfitrión y puerto y está definida en el archivo MobileThreeDHome.ini. Este proceso se muestra con un ejemplo en la figura 42.

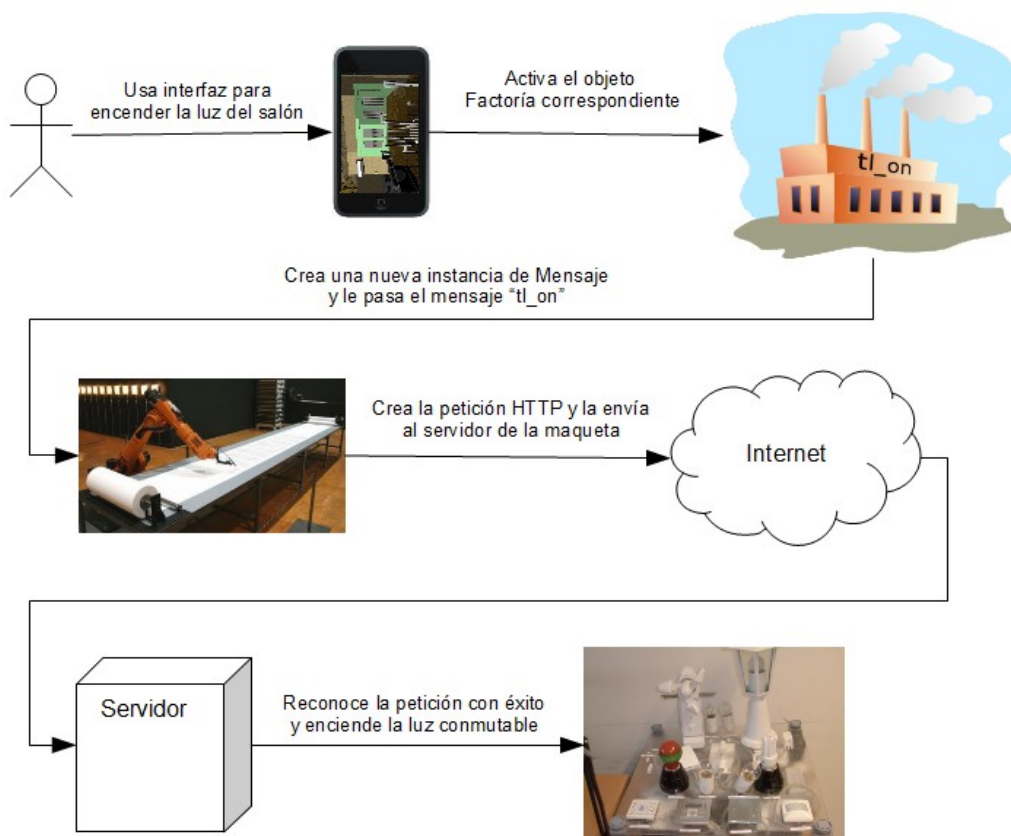


Figura 42. Ejemplo de comunicación con la maqueta

Guardar y Cargar el Estado de 3D Home

El objetivo de esta aplicación es controlar un hogar inteligente, así que a priori no tiene sentido que se guarde el estado de 3D Home porque cuando se inicie la aplicación debería cargar el estado de la vivienda. Sin embargo, dado que la maqueta que se está usando no guarda su estado, se ha decidido crear un sistema de guardado y cargado del estado de 3D Home para comprobar la viabilidad de cargar un estado guardado.

Las clases creadas para el sistema de guardar y cargar el estado de 3D Home son las siguientes: ThreeDHome, Estado3DH, KismetGrabar y KismetCargar.

Estado3DH es la clase que define el estado de la aplicación. El estado de la aplicación está definido por el aire acondicionado, la alarma, la persiana, la luz gradual y las luces conmutables del salón, de todas las habitaciones, del baño, del aseo, de la cocina y del pasillo. En la figura 43 se muestra esta clase.

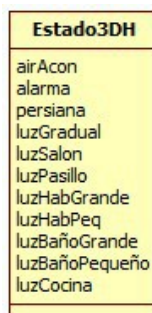


Figura 43. Clase Estado3DH

Todos los atributos de Estado3DH excepto luzGradual tienen dos estados: las luces y el aire

acondicionado tienen encendido y apagado, la alarma tiene armada y disparada y la persiana tiene bajada y subida. El atributo luzGradual tiene cinco estados, uno por cada porcentaje de intensidad permitido en 3D Home que son 0%, 25%, 50%, 75% y 100%.

En la clase ThreeDHome se define la función grabar() que es la que se encarga de guardar el estado de la aplicación en el archivo "SaveGame.bin". Esta clase se muestra en la figura 44.

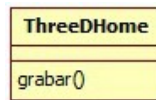


Figura 44. Clase ThreeDHome

KismetGrabar es la clase que define el objeto de secuencia "Save Game" de Unreal Kismet. Cuando este objeto de secuencia se activa, llama a la función grabar() de threeDHome para guardar el estado actual de la aplicación. La clase KismetGrabar se muestra en la figura 45.

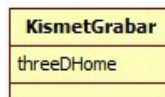


Figura 45. Clase KismetGrabar

El nombre del objeto de secuencia es "Save Game" y está en la categoría "3DH_Game". En la figura 46 se muestra dónde vería el diseñador este objeto de secuencia en Unreal Kismet.

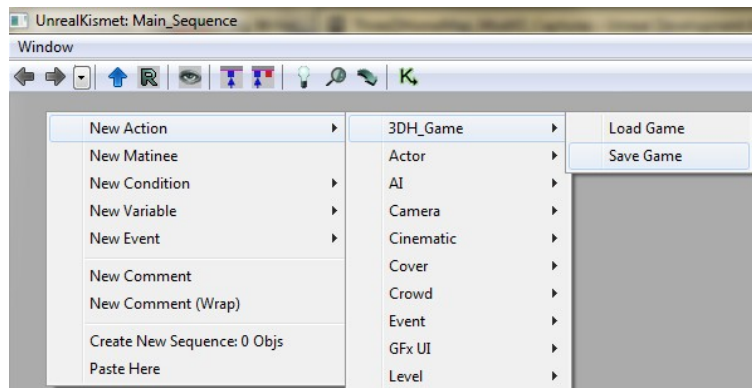


Figura 46. Categoría de Save Game

El único enlace de variables que tiene se llama "variables" y se pueden enlazar variables definidas en Kismet de tipo entero. Además, se pueden realizar operaciones de lectura sobre las variables enlazadas pero no de escritura. En la figura 47 se muestra cómo vería el diseñador este objeto de secuencia en Unreal Kismet.

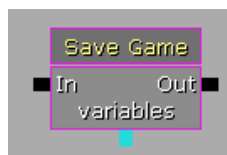


Figura 47. Objeto de secuencia Save Game

KismetCargar es la clase que define el objeto de secuencia "Load Game" de Unreal Kismet. Cuando este objeto de secuencia se activa, abre el archivo donde está guardado el estado de la aplicación y carga los valores del estado guardado en el estado actual de la aplicación. La clase KismetCargar se muestra en la figura 48.



Figura 48. Clase KismetCargar

El único enlace de variables que tiene se llama "variables" y se pueden enlazar variables definidas en Kismet de tipo entero. Además, se pueden realizar operaciones de lectura y escritura sobre las variables enlazadas. El aspecto de este objeto de secuencia es análogo a "Save Game" a excepción del nombre del objeto que pone "Load Game".

4.2.3. Despliegue

La fase de despliegue, mostrada en la figura 49, es la última del proceso de desarrollo de UDK Mobile. Para realizar esta fase, se necesita una licencia de desarrollador de iOS. Para 3D Home no se ha podido hacer este paso porque no se tiene dicha licencia.

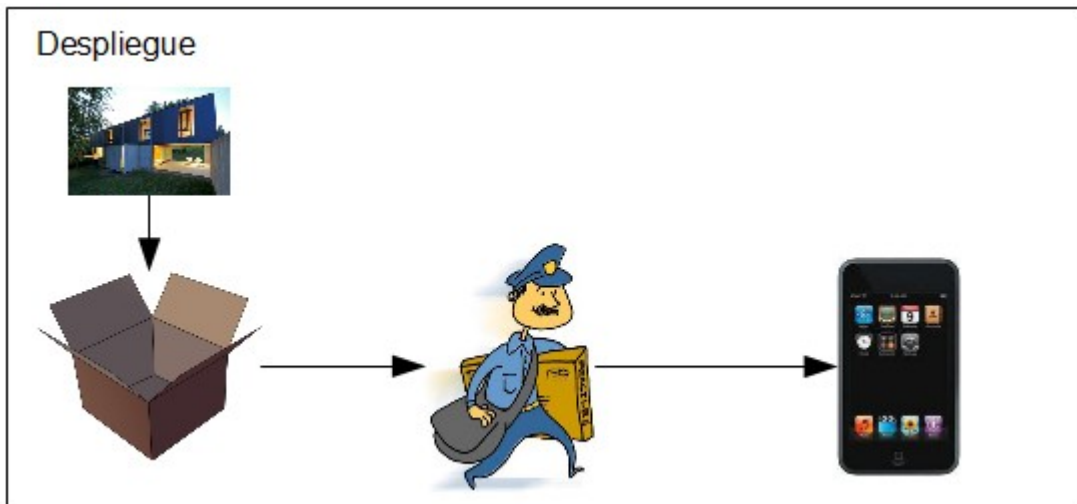


Figura 49. Fase de despliegue

En esta fase, la aplicación es empaquetada, para luego ser enviada e instalada a un dispositivo móvil que tenga un sistema operativo iOS como iPhone. Para ello UDK Mobile ofrece una herramienta llamada Unreal Frontend. Esta herramienta ofrece una interfaz gráfica que está dividida en cuatro áreas como se muestra en la figura 50.

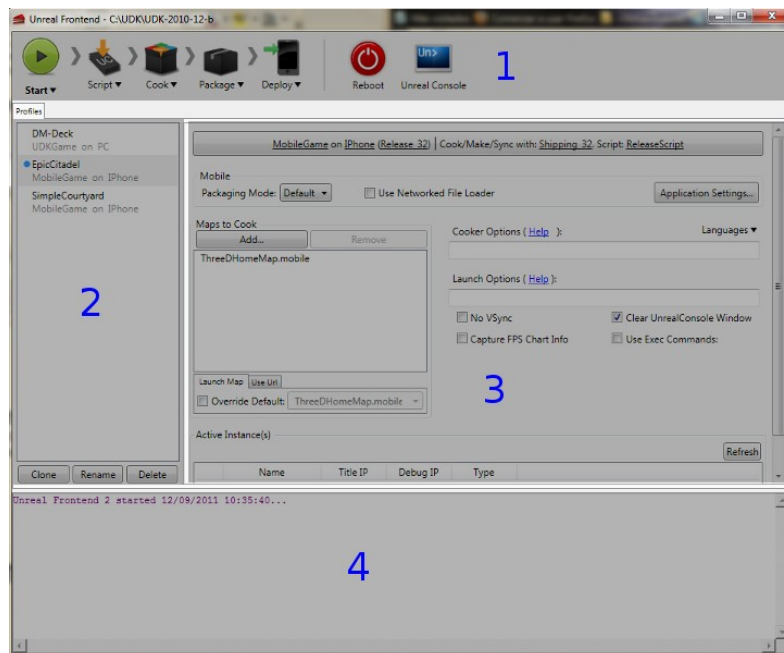


Figura 50. Unreal Frontend

El área 1 es la barra de herramientas. En esta área, además de otras opciones, está la tubería de trabajo. Una tubería es un proceso que contiene varias fases secuenciales donde la entrada de cada

una es la salida de la anterior. Por ejemplo, la entrada de la fase Cook, de la figura 50, es la salida de la fase Script. La tubería se puede personalizar con los botones mostrados. Todos los pasos que representan los botones excepto Start se pueden desactivar si se desea. En la tabla 5 se explica de forma general cada uno de estos botones.






Botón	Descripción
 Start ▼	Empieza la ejecución de la tubería de trabajo actual
 Script ▼	Compila todo el código de UnrealScript que use la aplicación
 Cook ▼	Integra el mapa y todos los paquetes de contenidos como materiales, objetos 3D y sonidos de la aplicación
 Package ▼	Empaqueta la aplicación para dispositivos móviles que tengan como sistema operativo iOS
 Deploy ▼	Entrega la aplicación empaquetada a los dispositivos iOS conectados a la máquina donde se ejecuta UDK Mobile

Tabla 5. Botones de la tubería de trabajo

El área 2 muestra todos los perfiles existentes de configuración de opciones. Un perfil de configuración es una colección individual de todas las opciones configuradas al igual que la configuración de la tubería de trabajo. Se pueden crear y eliminar perfiles.

El área 3 es el panel donde se configuran varias opciones para las fases de la tubería de trabajo. En este panel se indica también cuál es el mapa que se quiere empaquetar, en este caso es ThreeDHomeMap.mobile.

El área 4 muestra el progreso de las acciones realizadas por Unreal Frontend. Incluye información general, advertencias y errores.

4.3. Evolución de 3D Home

El diseño de la aplicación no ha sido estático. Mientras se ha ido avanzando en su desarrollo, el diseño también ha ido evolucionando. Una vez se ha creado el escenario inicial, se hacen las pruebas para detectar fallos de la aplicación o de usabilidad. A continuación se explican los cambios realizados en 3D Home para resolver los fallos detectados. Las imágenes mostradas en esta sección son de la aplicación en tiempo de ejecución.

En 3D Home se tienen varias luces dinámicas, éstas son las luces conmutables, la luz gradual y la luz parpadeante de la alarma. Por esto, el mayor y principal problema para nuestra tesina que se ha encontrado en UDK en su versión Mobile es que las luces dinámicas no funcionan bien. En las notas adicionales de UDK Mobile ya notifican que las luces dinámicas no funcionan como deberían y no garantizan el buen funcionamiento de las mismas pero no indican qué es lo que falla exactamente.

El mal funcionamiento de las luces dinámicas derivan en dos problemas. El primero de ellos es que las luces dinámicas no iluminan las texturas de la geometría del escenario. Para resolver el primer problema, se ha decidido cubrir las paredes, techo, suelo y rodapiés con InterpActors que simulen dichos objetos.



Figura 51. Salón cubierto con InterpActors

El segundo de ellos ya se puede ver en la figura 51, y es que no ilumina correctamente los objetos. Por ejemplo, la pared que rodea la persiana está compuesto por más de un objeto con el mismo material, y el mal funcionamiento de la iluminación hace que se distingan los objetos usados.

Como se puede observar en la figura 51, ya no está la luz gradual. Este cambio se ha realizado para no saturar el salón de elementos de interacción. Ahora, la luz gradual está en la habitación donde hay dos camas como se muestra en la figura 52.



Figura 52. Luz gradual

Otro cambio realizado es que todas las luces de la casa sean dinámicas, al principio todas las luces eran estáticas excepto una luz conmutable y una gradual. Ahora, todas las luces son conmutables excepto la luz gradual. Esto acentúa más el problema con las luces dinámicas, pero aumenta la interactividad con el escenario.

Cuando se probó la aplicación se detectó que el hueco de las puertas podría ser otro problema. Al principio eran más estrechas y a veces provocaban dificultades para entrar o salir de una habitación, así que se decidió aumentar el ancho de los huecos un 50%.

También se cambiaron el tamaño de las interfaces, al principio los botones eran más pequeños y se decidió aumentar su tamaño considerablemente para facilitar su uso.

Para la activación y desactivación del aire acondicionado, se decidió añadir un texto que avisara cuando el aire acondicionado se encendía y se apagaba. Este cambio vino motivado porque el feedback visual que recibía el usuario, la animación del aire acondicionado, se consideró que podía pasar desapercibido.

Por último, para la activación de la alarma, se pensó que debía hacerse más notoria su activación. Así que se añadió un sonido que se reprodujera de forma cíclica para avisar de que se había disparado la alarma. El sonido se ha cogido de [1].

4.4. Flexibilidad de 3D Home

Cuando se empezó con la tesina, se propuso crear una vivienda de forma dinámica. Esto quiere decir que la aplicación fuera capaz de crear una vivienda a partir de un fichero que contuviera la especificación de la vivienda. Esta especificación podría incluir aspectos como el número de habitaciones, su distribución, y los dispositivos domóticos y su posición.

Sin embargo, no se ha podido hacer debido a la tecnología usada. Se han encontrado, ya sea por no conocer completamente UDK Mobile o porque éste no lo permita, varios inconvenientes para crear una vivienda dinámica en la aplicación. Estos problemas son:

- No se puede modificar la estructura de la vivienda, esto incluye mover, crear y eliminar habitaciones. Las ideas que se barajaron para resolver el problema fueron:
 - Hacer una vivienda con puertas que estuvieran abiertas o cerradas dependiendo de la vivienda que se quisiera representar. Esto inmediatamente se vio que no servía porque la distribución de las habitaciones no cambiaba, simplemente se limitaba el acceso a algunas habitaciones.
 - Crear distintas viviendas y que el usuario pudiera elegir qué vivienda usar. Esto solo serviría para un conjunto de viviendas, así que se deshechó la idea.
- No se pueden introducir elementos o dispositivos nuevos. Las ideas que se barajaron para resolver el problema fueron:
 - Poner tanto los dispositivos necesarios como los dispositivos que pudiera tener más adelante la vivienda. Estos últimos elementos estarían ocultos se mostrarían cuando el usuario adquiriese dicho dispositivo en su vivienda real. Se desestimó la idea porque solo resolvería la situación si se supiera de antemano qué dispositivos puede tener el usuario, y éste no es el caso.

Aunque no se haya podido crear una casa de forma dinámica, el desarrollo de esta aplicación ha servido para comprobar que se puede usar UDK Mobile para controlar hogares inteligentes reales.

Se ha visto que, para 3D Home, primero se ha especificado la casa con un plano y luego se ha construido la vivienda desde cero con UDK Mobile. Lo que se propone a continuación es usar la especificación de la vivienda para generar de forma automática objetos que puedan ser usados en UDK Mobile.

Existen programas que permiten especificar viviendas, uno de los más usados es AutoCAD. Este programa permite, entre otras cosas, crear representaciones bidimensionales y tridimensionales de edificios.

Se podría crear una herramienta que fuera capaz de, a partir de una especificación que podría ser, por ejemplo, un modelo hecho con AutoCAD, transformar dicho modelo en uno o más modelos listos para ser usados en UDK Mobile.

Este transformador de modelos tendría como entrada un modelo hecho con alguna de las herramientas usadas por arquitectos como, por ejemplo, Autocad. Y tendría como salida uno o más modelos guardados en un formato que entienda UDK Mobile como por ejemplo .ase. Esto se puede ver en la figura 53.

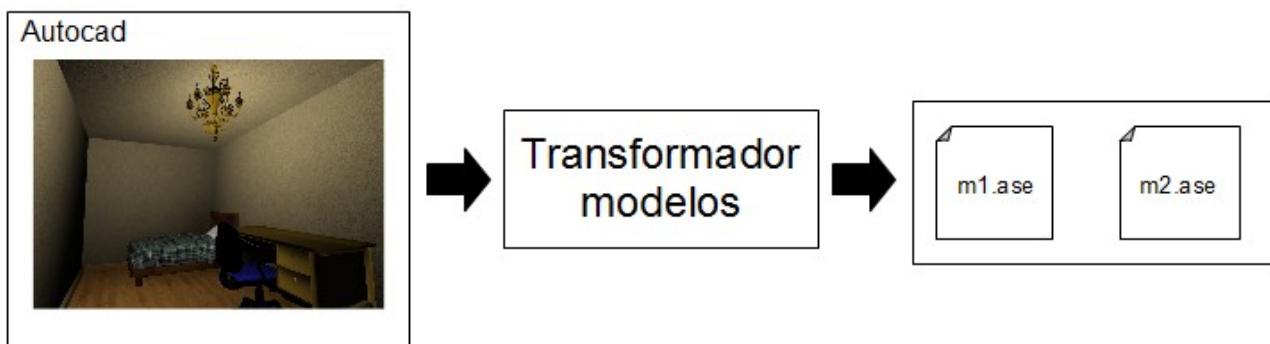


Figura 53. Transformador de modelos

Cada modelo importado se comporta como un objeto en su conjunto. Por ejemplo, se tiene un modelo Autocad donde hay una habitación totalmente amueblada, incluyendo una lámpara. Si se quiere hacer una habitación donde la luz se encienda al tocar la lámpara de la habitación entonces

sería suficiente con separar el modelo autocad en dos modelos: uno para la lámpara y otro para el resto. Esto es porque queremos dos comportamientos distintos. Entonces, el transformador de modelos debería tener como salida, como mínimo, tantos modelos como objetos con comportamientos distintos se quieran del modelo Autocad.

El proceso de transformación se va a explicar con el siguiente ejemplo. Se tiene un modelo Autocad que representa una habitación amueblada que incluye una lámpara como se muestra en la figura 54. Lo que se quiere hacer es, con UDK Mobile, una habitación que al tocar la lámpara, se encienda o apague una luz de la habitación.



Figura 54. Ejemplo de modelo Autocad

Para ello, hay que transformar el modelo Autocad a un formato que entienda UDK Mobile. Los pasos a seguir para realizar dicha transformación se muestran en la figura 55.

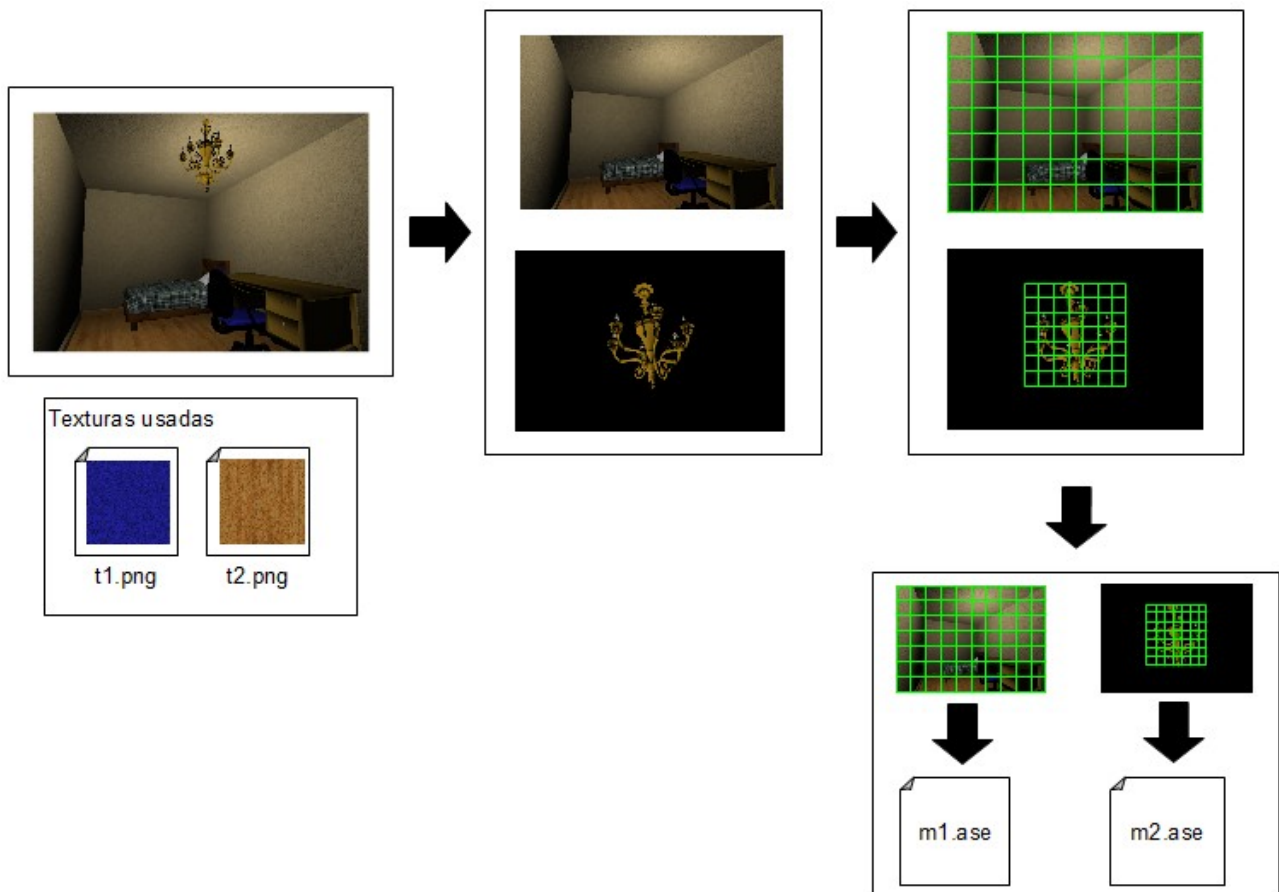


Figura 55. Proceso de transformación de modelos

Para empezar, se tiene el modelo Autocad y los archivos de las texturas usadas en dicho modelo. En la transformación solo se usará el modelo porque las texturas serán necesarias después de la transformación. Los pasos a realizar son los siguientes:

- Primero, separar el modelo Autocad en tantos modelos como objetos con comportamientos distintos se quiera. En este caso hay dos que son la lámpara que se podrá usar y el resto con el que no se podrá interactuar.
- Cuando UDK Mobile importa modelos, no importa los materiales de los mismos. Por ello, el segundo paso es añadir coordenadas a los modelos. Estas coordenadas sirven para poder añadir materiales a los objetos. Un ejemplo de técnica para añadir coordenadas es UVW mapping.
- El tercer paso es guardar los modelos en un formato que entienda UDK Mobile como .ase.

Una vez se tienen ya los modelos en un formato que entienda UDK Mobile, se pueden importar los modelos a UDK Mobile y reconstruir el escenario como estaba en Autocad. Este proceso se muestra en la figura 56.

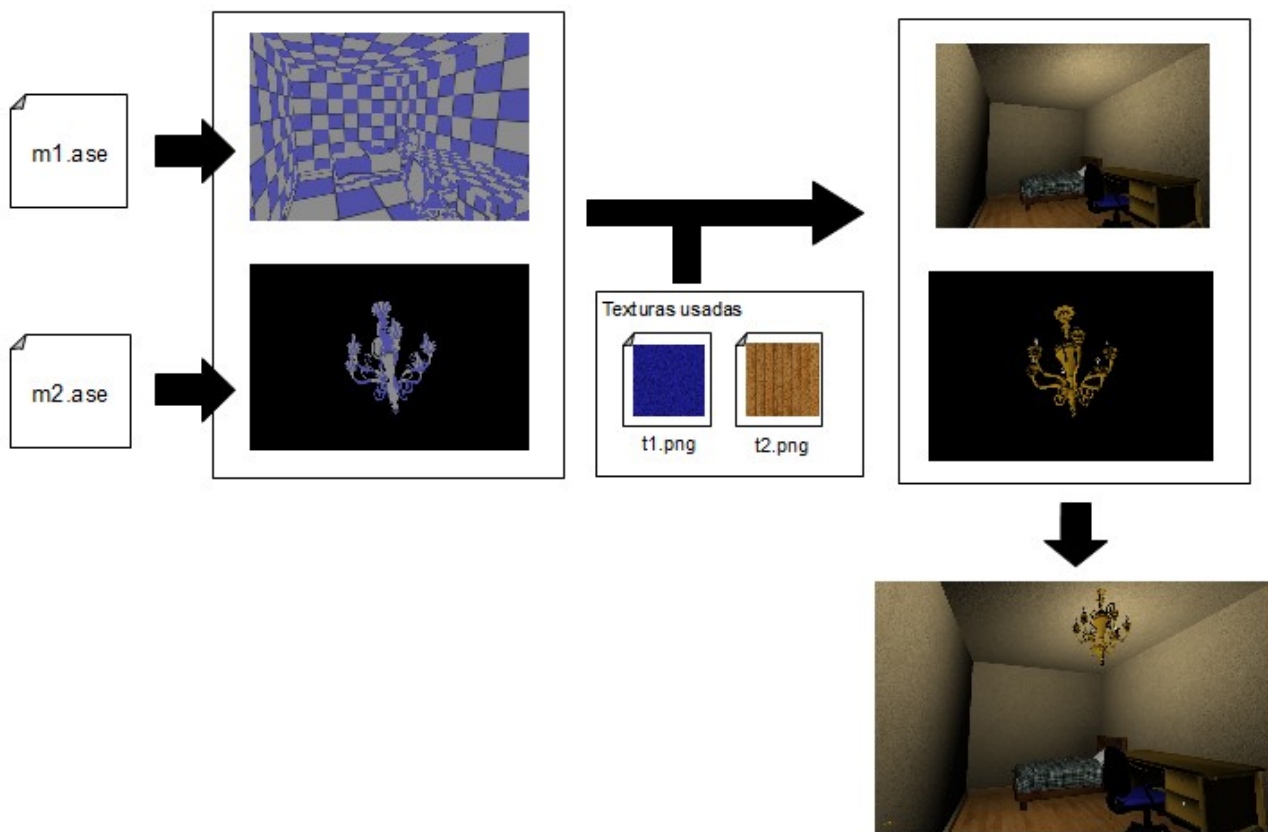


Figura 56. Proceso de reconstrucción del escenario

Todos los pasos que se describen a continuación se hacen usando las herramientas proporcionadas por UDK Mobile:

- El primer paso es importar los modelos. Una vez importados se puede ver que UDK Mobile no importa los materiales usados en los modelos.
- El segundo paso es crear y aplicar los materiales que habían en el modelo Autocad. Para crear los materiales se utilizan las texturas usadas en el modelo Autocad. Luego, se aplican los materiales creados a los modelos importados.
- El último paso es colocar la lámpara como estaba en el modelo Autocad.

Ahora ya se tiene el escenario como estaba en Autocad, y se puede implementar el comportamiento deseado a la lámpara.

4.5. Conclusiones

Con 3D Home se ha demostrado que se puede controlar un hogar inteligente con UDK Mobile. Sin embargo, el problema que presenta es que solo sirve para las viviendas que compartan la estructura de la casa. Si se quiere controlar viviendas con otra estructura se debe crear otra vivienda con UDK.

En respuesta a este problema se ha propuesto el desarrollo de una herramienta que facilite la creación de viviendas con UDK Mobile. Esta herramienta tendría como entrada la especificación de una vivienda y como salida uno o varios objetos que podrían ser usados en UDK para la creación de la vivienda. También se ha explicado el proceso que se seguiría tanto en la herramienta para crear los objetos como en UDK para crear la vivienda con los objetos creados.

Capítulo 5

Caso de Estudio

3D Home es una aplicación donde el usuario puede interactuar con la maqueta a través de la representación virtual de una casa. Esta representación virtual se ha creado con Unreal Development Kit Mobile (a partir de ahora UDK Mobile) que ofrece herramientas y facilidades para dicho fin.

En este capítulo se van a mostrar diversos diagramas de estado para explicar el comportamiento de los objetos de interacción de 3D Home. Por ello, a continuación se explica lo suficiente de los diagramas de estados para entender los usados en este documento.

El diagrama de estados es uno de los modelos propuestos por UML (Lenguaje Unificado de Modelado) para describir de forma gráfica el comportamiento de un objeto. Un diagrama de estados especifica las secuencias de estados por las que pasa un objeto durante su vida, en respuesta a eventos, junto con sus respuestas. Un estado es un período de tiempo durante el cual espera algún evento y gráficamente está representado por un rectángulo redondeado. Una transición es una relación entre dos estados que indica que cuando el objeto esté en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado. Gráficamente, las transiciones son representadas como flechas.

Además de los elementos descritos anteriormente, un diagrama de estados puede tener dos estados especiales que son el estado inicial y el estado final. El estado inicial indica el punto de comienzo por defecto del objeto y está representado por un círculo negro. El estado final indica que la ejecución del objeto ha finalizado y está representado por un círculo negro dentro de un círculo blanco.

El resto del capítulo está organizado como sigue: primero, la sección 5.1 ofrece una visión global de la aplicación. La sección 5.2 explica la funcionalidad de la aplicación desde el punto de vista del usuario.

5.1. Visión Global

3D Home es la aplicación cliente de un sistema que tiene una arquitectura cliente-servidor. En un sistema de este tipo, una aplicación llamada cliente puede requerir algún tipo de servicio a otra aplicación llamada servidor mediante una red como una LAN (Red de Área Local) o Internet. Aunque servidores y clientes suelen estar en máquinas separadas, no es necesario que sea así.

3D Home es una aplicación móvil que permite al usuario controlar una maqueta de dispositivos reales que representa un hogar inteligente. En la figura 57 se muestra que, para este control, la aplicación usa los servicios ofrecidos por un servidor web que se encarga de gestionar los servicios domóticos de una maqueta de dispositivos reales.

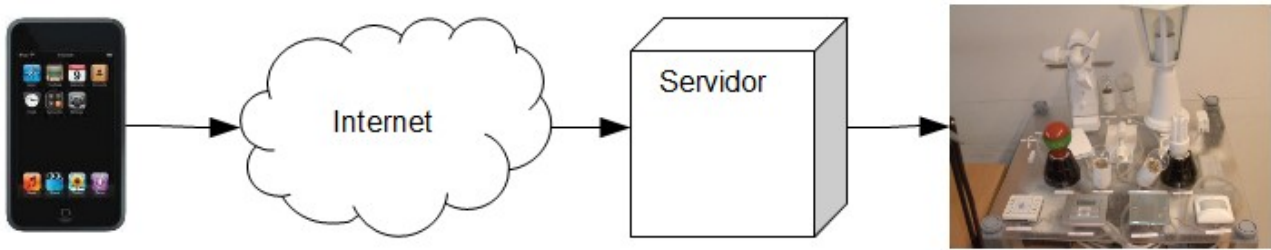


Figura 57. Diagrama de flujo del control de la maqueta

El funcionamiento ideal de la aplicación es que, al iniciarse, modificara su estado para que coincida con el estado de la maqueta. Sin embargo, la maqueta no guarda su estado. Así que se ha recurrido a un sistema de guardado de la aplicación. En este sistema, siempre que el usuario realiza cualquier cambio en la maqueta, 3D Home guarda el estado de la aplicación. Y cuando el usuario inicia 3D Home, se carga el estado que tenía la aplicación la última vez que se cerró.

5.2. Visión del Usuario

El usuario usa 3D Home para controlar la maqueta, a continuación se explica de forma detallada las interfaces que usa. También se describe el efecto que tiene el uso de dichas interfaces tanto en la aplicación como en la maqueta.

3D Home tiene como plataforma objetivo iOS. Esta plataforma la usan los dispositivos iPhone, iPod Touch! y iPad. 3D Home se centra en iPhone y iPod Touch! por su similitud en el tamaño de pantalla y porque no se tenía acceso a un iPad.

En iPhone y iPod Touch!, el usuario interactúa con las aplicaciones de forma totalmente táctil. Esto, junto con el pequeño tamaño de la pantalla, condiciona el diseño de las interfaces porque hay que hacer los elementos lo suficientemente grandes y lo suficientemente separados entre ellos para que se puedan usar sin dificultad. Esto también afecta a la información mostrada al usuario porque si no es bastante grande, el usuario tendrá problemas para leerla.

Antes de describir las interfaces, se va a explicar qué es "usar un elemento" en 3D Home. En las aplicaciones táctiles hay distintas aproximaciones para usar elementos como simplemente tocar la pantalla en el elemento que se quiere usar o tocar y dejar de tocar la pantalla. En 3D Home se ha optado por la segunda aproximación.

En la figura 58 se muestra el comportamiento general de la aplicación.

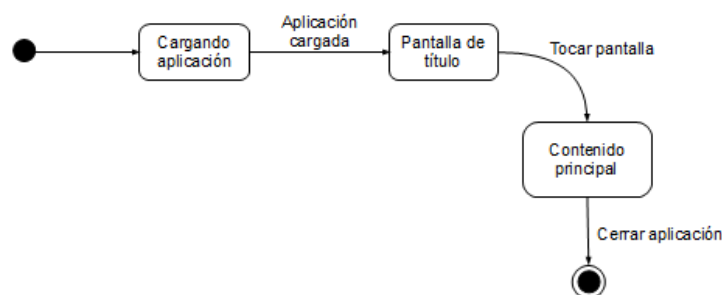


Figura 58. Diagrama de estados general de 3D Home

Cuando se inicia la aplicación, primero se muestra una pantalla de carga. Cuando termina de cargar, se muestra la pantalla de título donde se espera que el usuario toque la pantalla para empezar a usar la aplicación. Esta pantalla de título se muestra en la figura 59.



Figura 59. Pantalla de título

En la figura 60 se muestra una vista en primera persona dentro de la representación virtual de la casa con el estado inicial de la casa. Este estado inicial puede ser el que viene por defecto o, si ya se ha interactuado antes con la casa, el estado en el que se dejó la casa anteriormente. El estado por defecto es todas las luces apagadas, alarma armada, persiana bajada y aire acondicionado apagado.

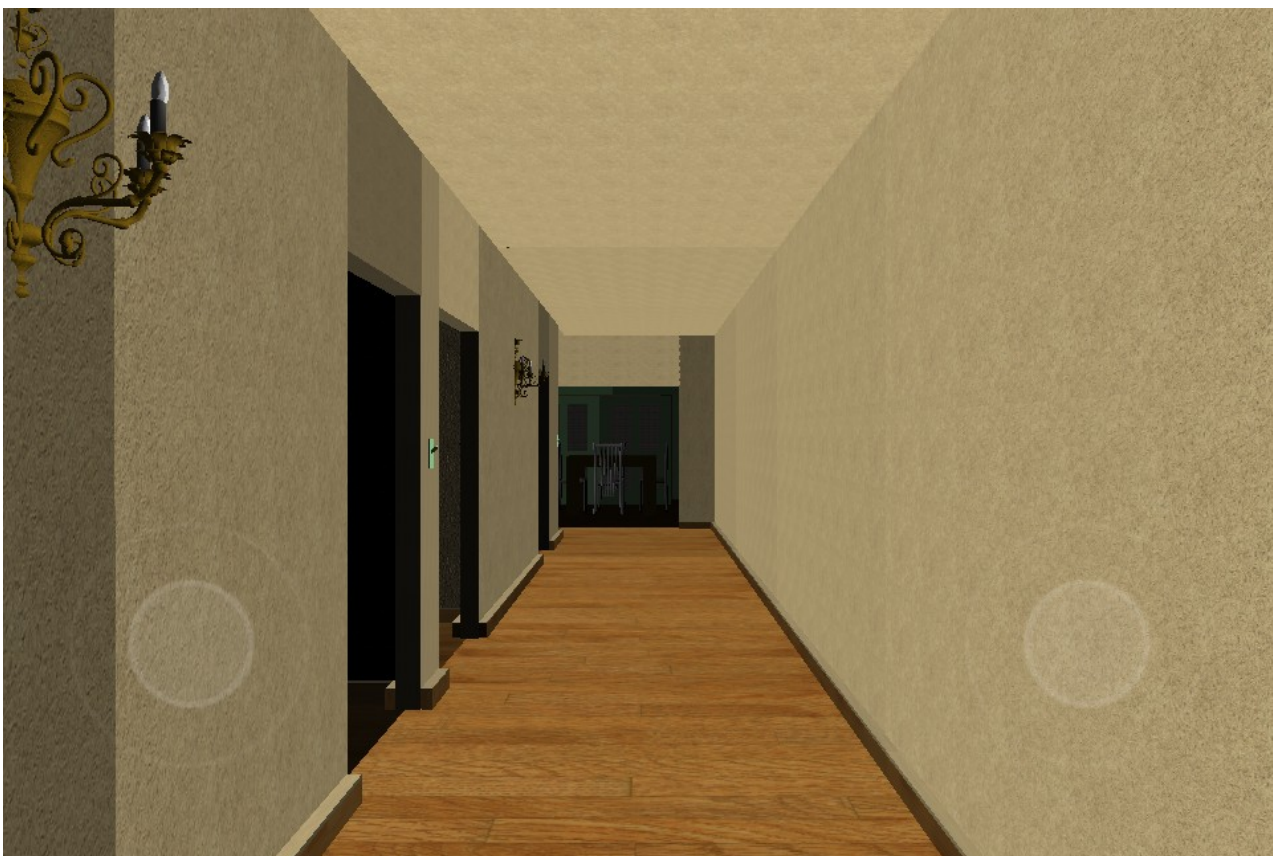


Figura 60. Estado por defecto de 3D Home

Como se puede ver en la figura 60, el usuario usa dos joysticks virtuales situados en la parte inferior de la pantalla para explorar la casa. El joystick izquierdo se usa para moverse y el derecho para mirar alrededor.

En 3D Home, para interactuar con elementos como las lámparas y el aire acondicionado se ha seguido una filosofía que se aproxima a la realidad. Por ejemplo, si se quiere encender una lámpara entonces el usuario deberá buscar el interruptor correspondiente y usarlo. Además, para comodidad del usuario, éste puede interactuar directamente con el elemento en vez de con el interruptor. Así, si el usuario quiere encender la lámpara del salón, puede entrar en el salón y usar la lámpara del techo sin necesidad de buscar el interruptor.

En 3D Home, hay elementos que tienen un tono verdoso como los interruptores. Estos elementos son elementos de interacción que permiten controlar las luces, aire acondicionado, persiana y alarma de la vivienda.

La vivienda representada en 3D Home, además del pasillo visto en la figura 60, tiene varias habitaciones. A continuación se muestran imágenes de todas estas habitaciones, estas imágenes muestran solo una parte de la habitación y puede no mostrar todos los elementos que tiene.

En la figura 61 se puede ver el baño que consta de un lavabo, un báter, una lámpara y un interruptor. Los elementos de interacción del baño son la lámpara y el interruptor.



Figura 61. Baño

En la figura 62 se puede ver el dormitorio grande que consta de una cama de matrimonio, cuatro armarios, dos mesitas de noche, una cajonera, un televisor, una lámpara y un interruptor. Los elementos de interacción de este dormitorio son la lámpara y el interruptor.



Figura 62. Dormitorio grande

En la figura 63 se puede ver un dormitorio que consta de una cama, un armario, una mesita de noche, dos escritorios, una silla giratoria, una lámpara y un interruptor. Los elementos de interacción de este dormitorio son la lámpara y el interruptor.



Figura 63. Dormitorio pequeño de una cama

En la figura 64 se puede ver otro dormitorio pequeño que consta de dos camas, un armario, una mesita de noche, una lámpara gradual y un interruptor. Los elementos de interacción de este dormitorio son la lámpara gradual y el interruptor.



Figura 64. Dormitorio pequeño de dos camas

En la figura 65 se puede ver el aseo que consta de un báter, un lavabo, una lámpara y un interruptor. Los elementos de interacción del aseo son la lámpara y el interruptor.

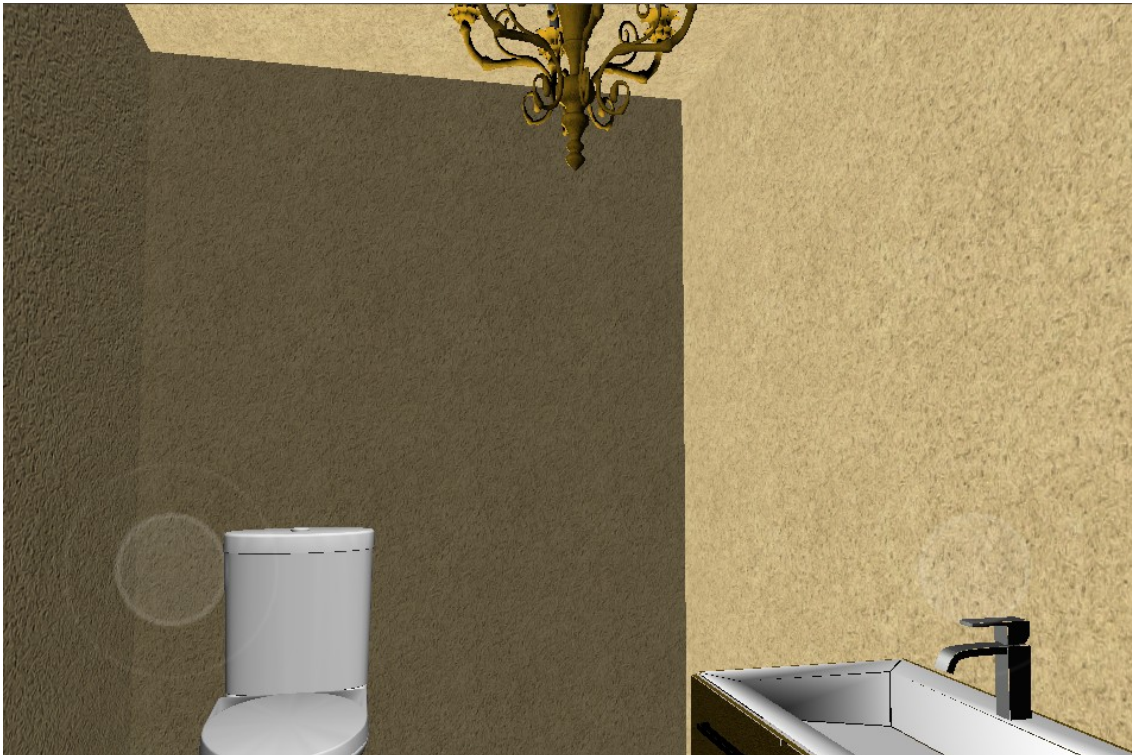


Figura 65. Aseo

En la figura 66 se puede ver la cocina que consta de varios armarios y cajoneras, un horno, un frigorífico, una mesa, cuatro sillas, una lámpara y un interruptor. Los elementos de interacción de la cocina son la lámpara y el interruptor.



Figura 66. Cocina

En la figura 67 se puede ver el salón que consta de tres sofás, un televisor, varios armarios, dos jarrones, una mesa, cuatro sillas, una ventana, una persiana, un aire acondicionado, una lámpara y dos interruptores. Los elementos de interacción de esta habitación son la persiana, el aire acondicionado, la lámpara y los dos interruptores.



Figura 67. Salón

En la figura 68 se puede ver el recibidor de la vivienda que consta de un armario, una puerta y una alarma. El elemento de interacción del recibidor es la alarma.



Figura 68. Recibidor de la vivienda

Para tener una visión global de la posición de todos los elementos de interacción, en la figura 69 se muestra la posición de todos los elementos de interacción en la vivienda.

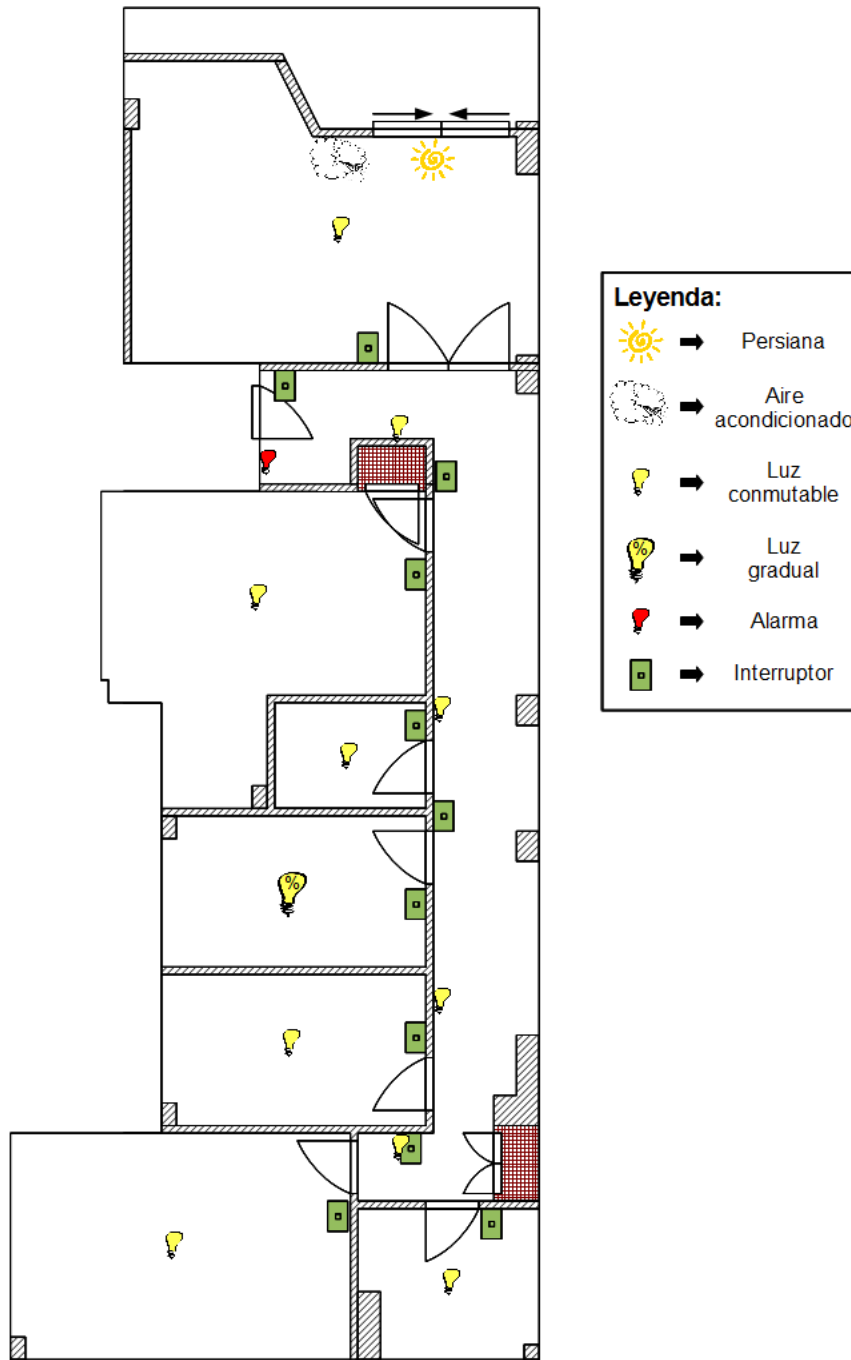


Figura 69. Posición de los elementos de interacción

Una vez se ha mostrado todos los elementos de la vivienda, se explica en detalle las distintas interfaces de los elementos de interacción.

Luces conmutables

El primer tipo de interfaz que se ha creado es para los elementos que solo tienen dos estados como las luces conmutables. Los estados de estas luces son encendidas y apagadas. Para usarlas, se pueden usar tanto los interruptores, a excepción del interruptor del aire acondicionado, como las propias lámparas. Cuando se interactúa con una lámpara o interruptor solo se encenderá o apagará la luz de la habitación donde está el interruptor o la lámpara usada.

Para explicar mejor las luces conmutables, se toma como ejemplo la luz conmutable del salón porque es la única luz conmutable de 3D Home que está conectada a la maqueta. Los elementos de interacción para controlar la luz del salón están señalizados en la figura 70.



Figura 70. Interfaz de la luz conmutable del salón

En la figura 71 se muestra el comportamiento de una luz conmutable.

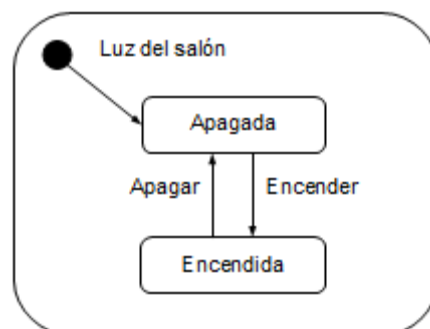


Figura 71. Diagrama de estados de la luz del salón

Cuando se usa la interfaz de la luz del salón, 3D Home realiza distintas acciones dependiendo del estado actual de éste:

- Si está apagada:

- Enciende la luz.
- Envía un mensaje a la maqueta para encender la luz conmutable.
- Si está encendida:
 - Apaga la luz.
 - Envía un mensaje a la maqueta para apagar la luz conmutable.

En la figura 72 se muestra el aspecto de la aplicación cuando el usuario apaga la luz.



Figura 72. Ejemplo de interacción con la luz del salón

Aire acondicionado

Otro elemento con el que se puede interactuar es el aire acondicionado. Para interactuar con él se puede usar el interruptor que hay cerca del aire acondicionado o el propio aire acondicionado. Estos elementos de interacción están señalizados en la figura 73.

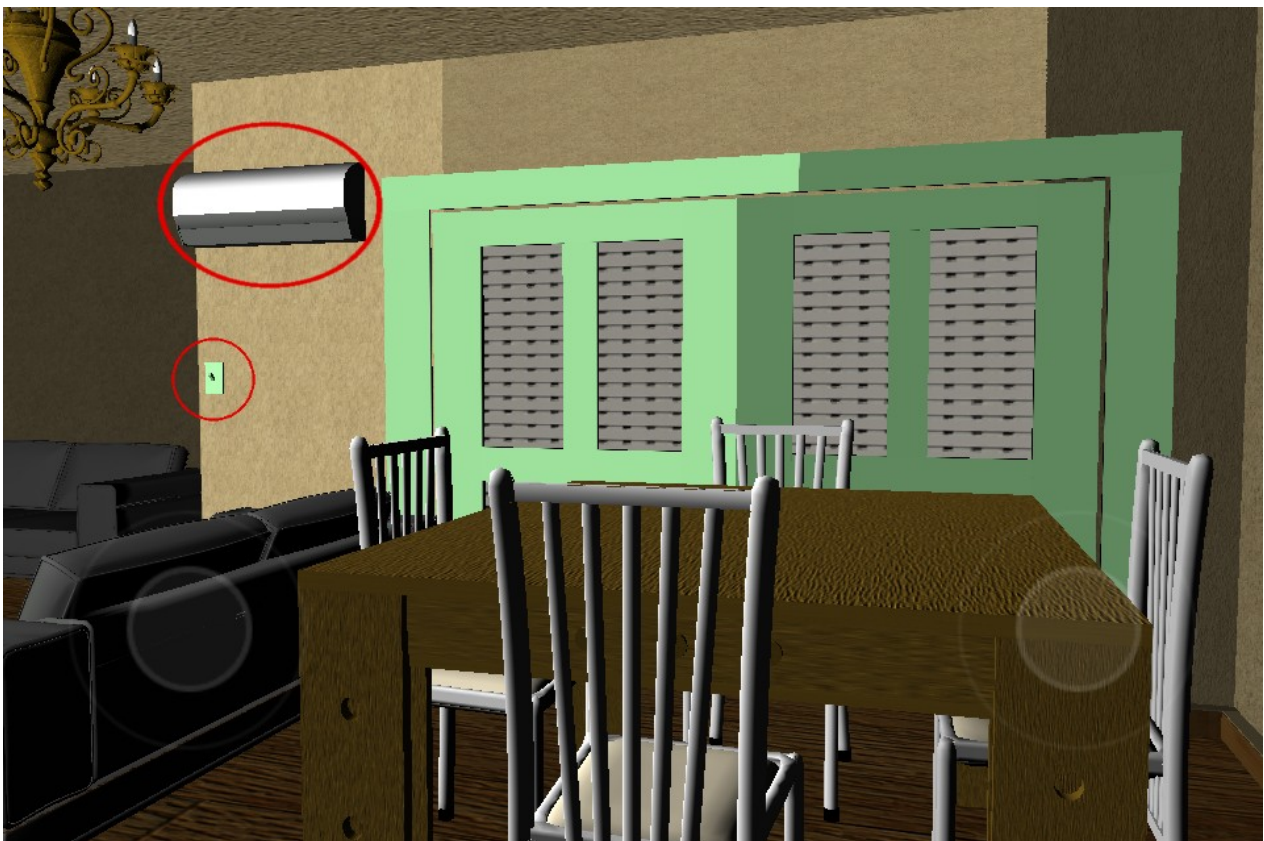


Figura 73. Interfaz del aire acondicionado

En la figura 74 se muestra el comportamiento del aire acondicionado.

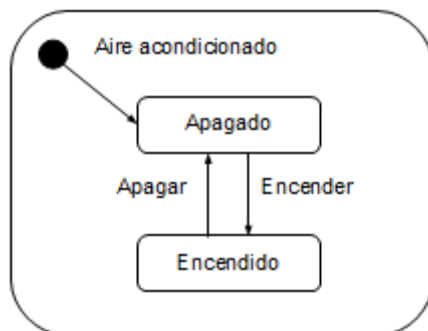


Figura 74. Diagrama de estados del aire acondicionado

Cuando se usa la interfaz del aire acondicionado, 3D Home realiza distintas acciones dependiendo del estado actual de éste:

- Si está apagado:
 - La parte inferior del aire acondicionado se abre para dejar pasar el aire.
 - Como la parte inferior del aire acondicionado es pequeña, es posible que la animación pase inadvertida. Por esto también muestra por pantalla el mensaje "Air Conditioning: Enabled" durante 3 segundos.
 - Además envía un mensaje a la maqueta para encender el ventilador.
- Si está encendido:
 - La parte inferior del aire acondicionado se cierra.
 - Como la parte inferior del aire acondicionado es pequeña, es posible que la animación pase inadvertida. Por esto también muestra por pantalla el mensaje "Air Conditioning: Disabled" durante 3 segundos.
 - Además envía un mensaje a la maqueta para apagar el ventilador.

En la figura 75 se muestra el aspecto de la aplicación cuando el usuario enciende el aire acondicionado. Las dos imágenes son el estado inicial y final de la animación.

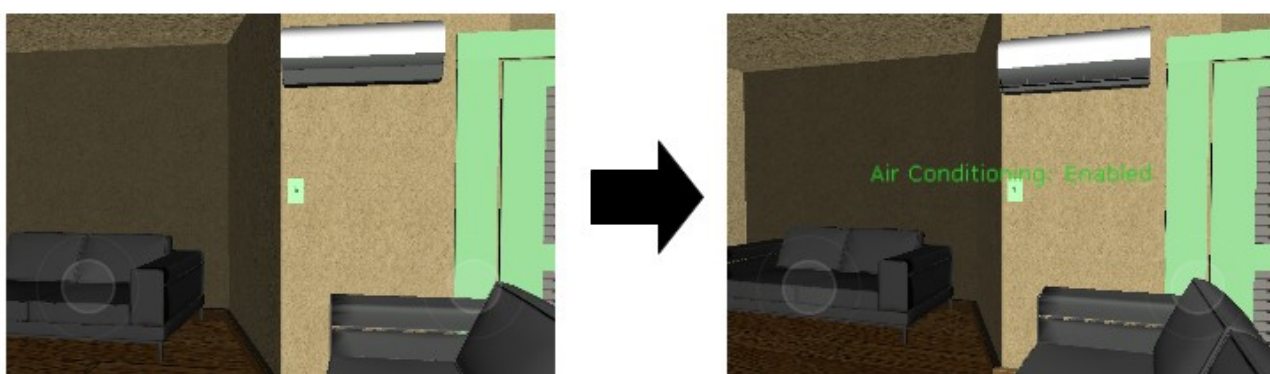


Figura 75. Ejemplo de encendido del aire acondicionado

En la figura 76 se muestra el aspecto de la aplicación cuando el usuario apaga el aire acondicionado. Las dos imágenes son el estado inicial y final de la animación.

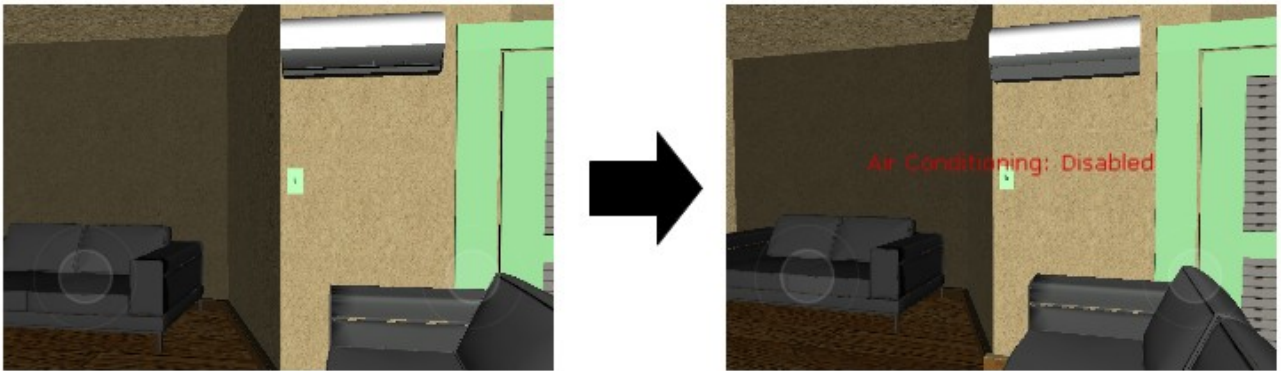


Figura 76. Ejemplo de apagado del aire acondicionado

Persiana

Para interactuar con la persiana solo se puede usar el área que cubre la ventana que enmarca la persiana. El objeto de interacción está señalizado en la figura 77.

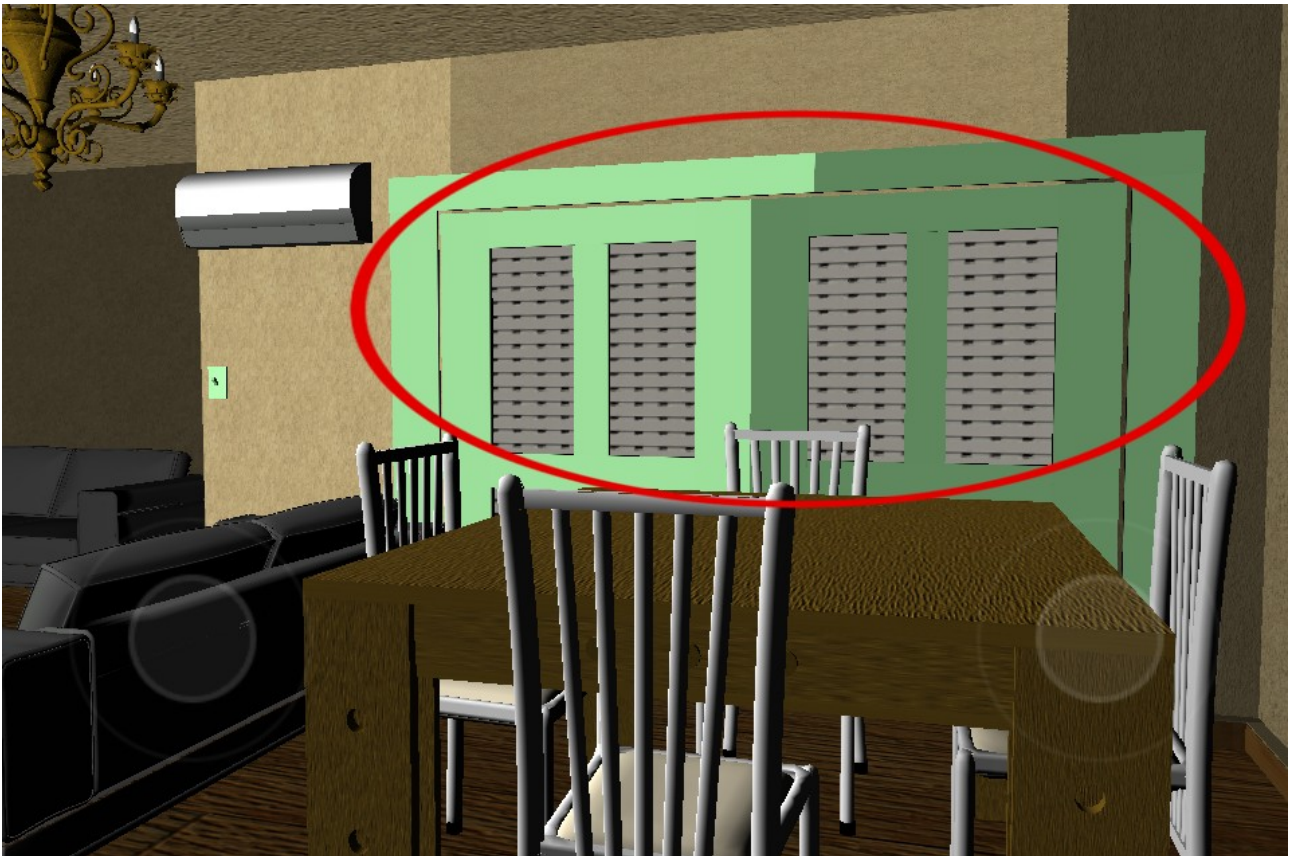


Figura 77. Interfaz de la persiana

En la figura 78 se muestra el comportamiento de la persiana.

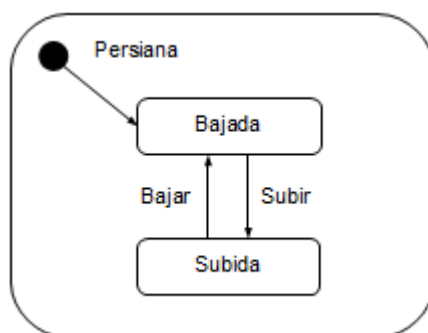


Figura 78. Diagrama de estados de la persiana

Cuando se usa la interfaz de la persiana, 3D Home realiza distintas acciones dependiendo del estado actual de ésta:

- Si está bajada:
 - Sube la persiana.
 - Envía un mensaje a la maqueta para que el rodillo gire en sentido horario. Cuando, en la aplicación, la persiana termina de subir entonces 3D Home también envía un mensaje a la maqueta para que el rodillo se detenga.
- Si está subida:
 - Baja la persiana.
 - Envía un mensaje a la maqueta para que el rodillo gire en sentido antihorario. Cuando, en la aplicación, la persiana termina de bajar entonces 3D Home también envía un mensaje a la maqueta para que el rodillo se detenga.

En la figura 79 se muestra el aspecto de la aplicación cuando el usuario sube la persiana. Las dos imágenes son el estado inicial y final de la animación.

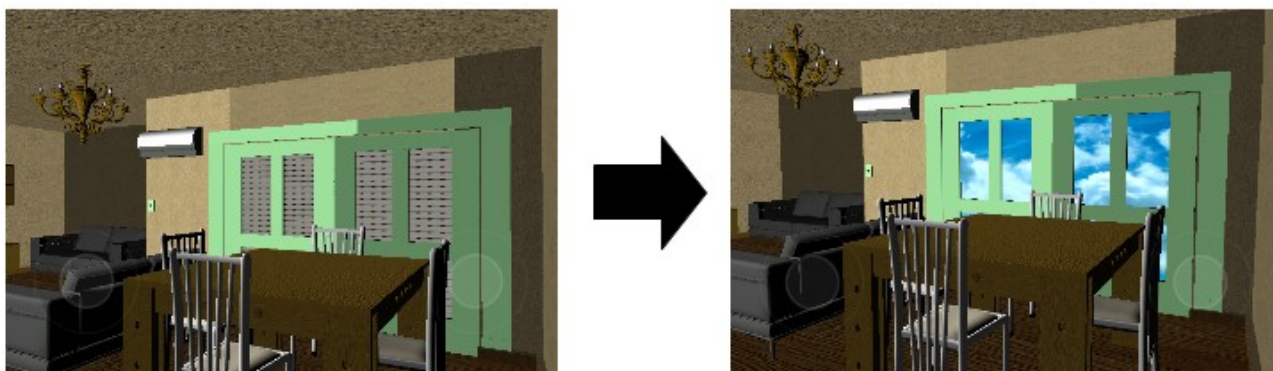


Figura 79. Ejemplo de interacción con la persiana

Alarma

La alarma representa un detector de presencia que, cuando está armada, muestra una luz parpadeante roja como se muestra en la figura 80. Para realizar las pruebas de la alarma, se ha simplificado su comportamiento para que se dispare cuando se use.



Figura 80. Interfaz para activar la alarma

Una vez se ha disparado la alarma, la luz parpadeante pasa a ser verde, se muestra por pantalla un mensaje indicando que la alarma ha sido activada y un botón para desactivarla. Se puede apagar la alarma, armándola así, con el botón mostrado o con la propia alarma. Estos elementos de interacción están señalizados en la figura 81.



Figura 81. Interfaz para apagar la alarma

En la figura 82 se muestra el comportamiento de la alarma.

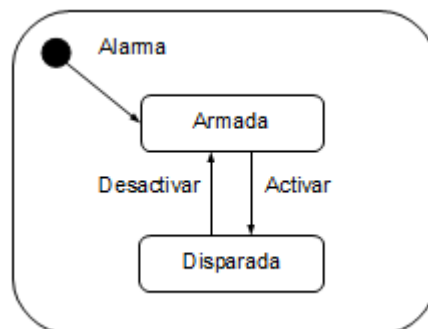


Figura 82. Diagrama de estados de la alarma

Como se ha visto anteriormente, la alarma tiene dos interfaces distintas dependiendo de su estado:

- Si se usa la interfaz de la alarma armada:
 - Muestra una luz intermitente verde.
 - También muestra por pantalla el mensaje "Alarm: Activated" de duración indefinida.
 - Además, muestra un botón para apagar la alarma.
 - Por último, reproduce un sonido de forma cíclica de duración indefinida.

- Si se usa la interfaz de la alarma disparada:
 - Muestra una luz intermitente roja.

- También oculta el mensaje "Alarm: Activated" y el botón de apagado de alarma.
- Por último, detiene la reproducción del sonido.

En la figura 83 se muestra el aspecto de la aplicación cuando el usuario activa la alarma.

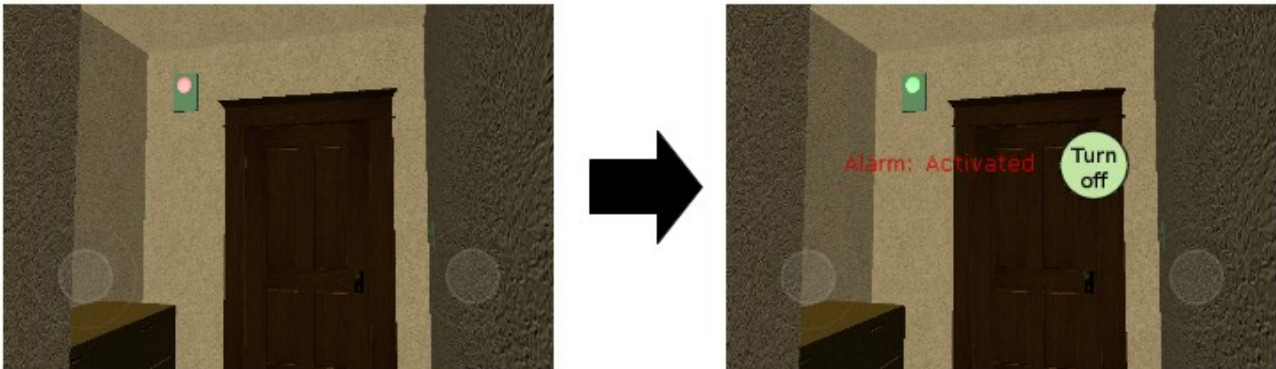


Figura 83. Ejemplo de interacción con la alarma

Luz gradual

Por último, en la vivienda también hay una luz gradual. Ésta se usa como todas las luces, usando la propia lámpara que está señalizada en la figura 84 o el interruptor que hay en la habitación.



Figura 84. Habitación donde está la luz gradual

Cuando se usa la luz gradual, aparece una interfaz para seleccionar la intensidad de la luz. Esta interfaz se muestra en la figura 85. Una vez queda seleccionada, la interfaz desaparece requiriendo usar otra vez la luz gradual si se quiere cambiar nuevamente la intensidad.

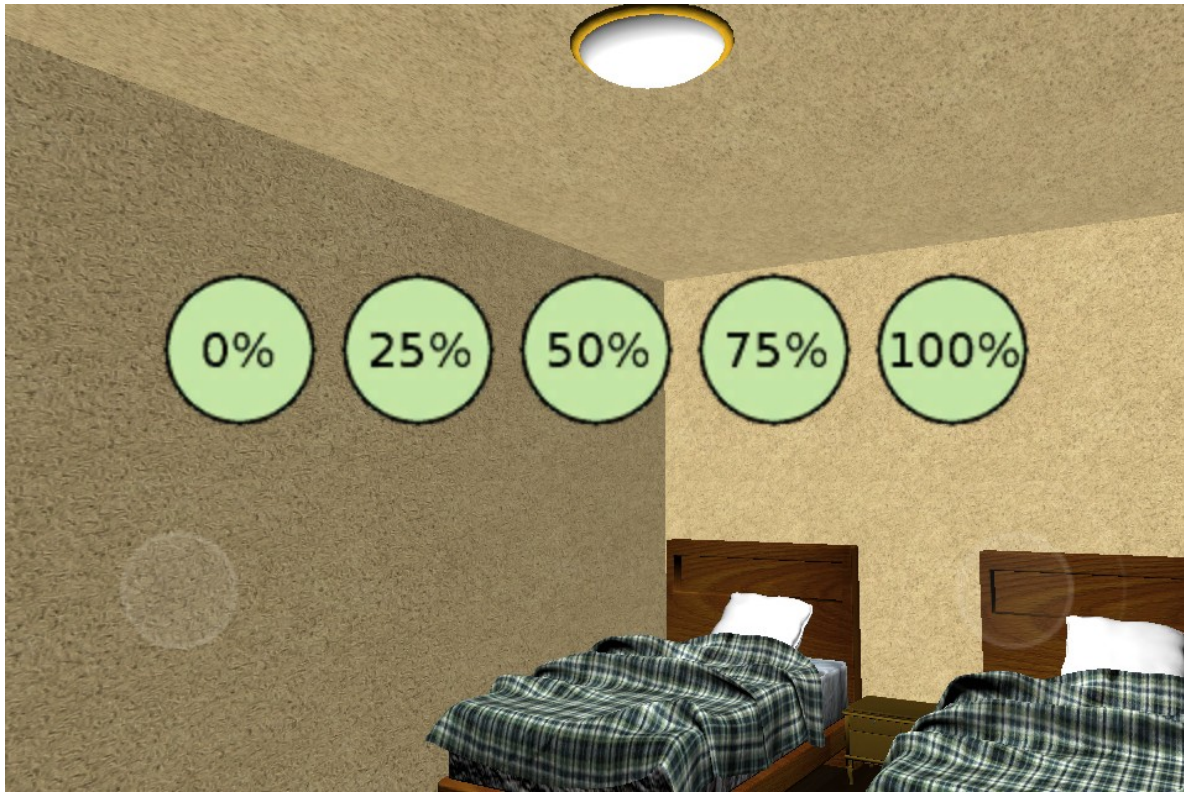


Figura 85. Interfaz de selección de intensidad

En el diagrama de estados de la luz gradual, se usa un nuevo tipo de transición que es una flecha bidireccional coloreada. Una flecha bidireccional equivale a dos flechas donde el sentido de cada flecha viene determinado por su color y tiene el evento del mismo color. Esta explicación está ilustrada en la figura 86. Este tipo de anotación no pertenece a UML y se ha usado para facilitar la lectura del diagrama.

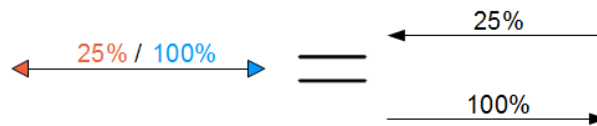


Figura 86. Explicación gráfica del nuevo tipo de transición

En la figura 87 se muestra el comportamiento de la luz gradual.

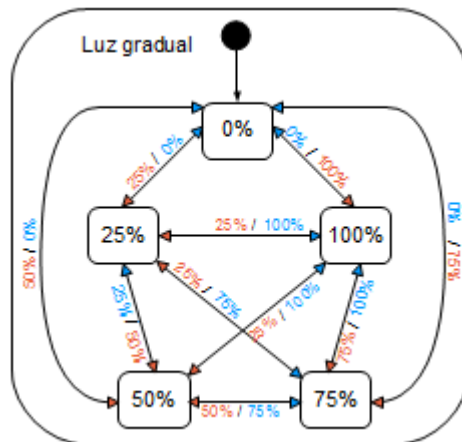


Figura 87. Diagrama de estados de la luz gradual

Los estados son las intensidades permitidas para la luz gradual, y las transiciones indican que se deben realizar acciones cuando se pulsa el botón indicado en la transición. Este botón es uno de

los botones de la interfaz mostrada en la figura 85. Cuando se usa cualquiera de estos botones:

- La intensidad de la luz disminuye o aumenta hasta llegar al porcentaje de intensidad indicado.
- Envía un mensaje a la maqueta para que la luz gradual se ilumine al porcentaje de intensidad indicado.
- Oculta la interfaz de selección de intensidad.

En la figura 88 se muestra el aspecto de la aplicación cuando el usuario cambia la intensidad de la luz del 50% al 100%.

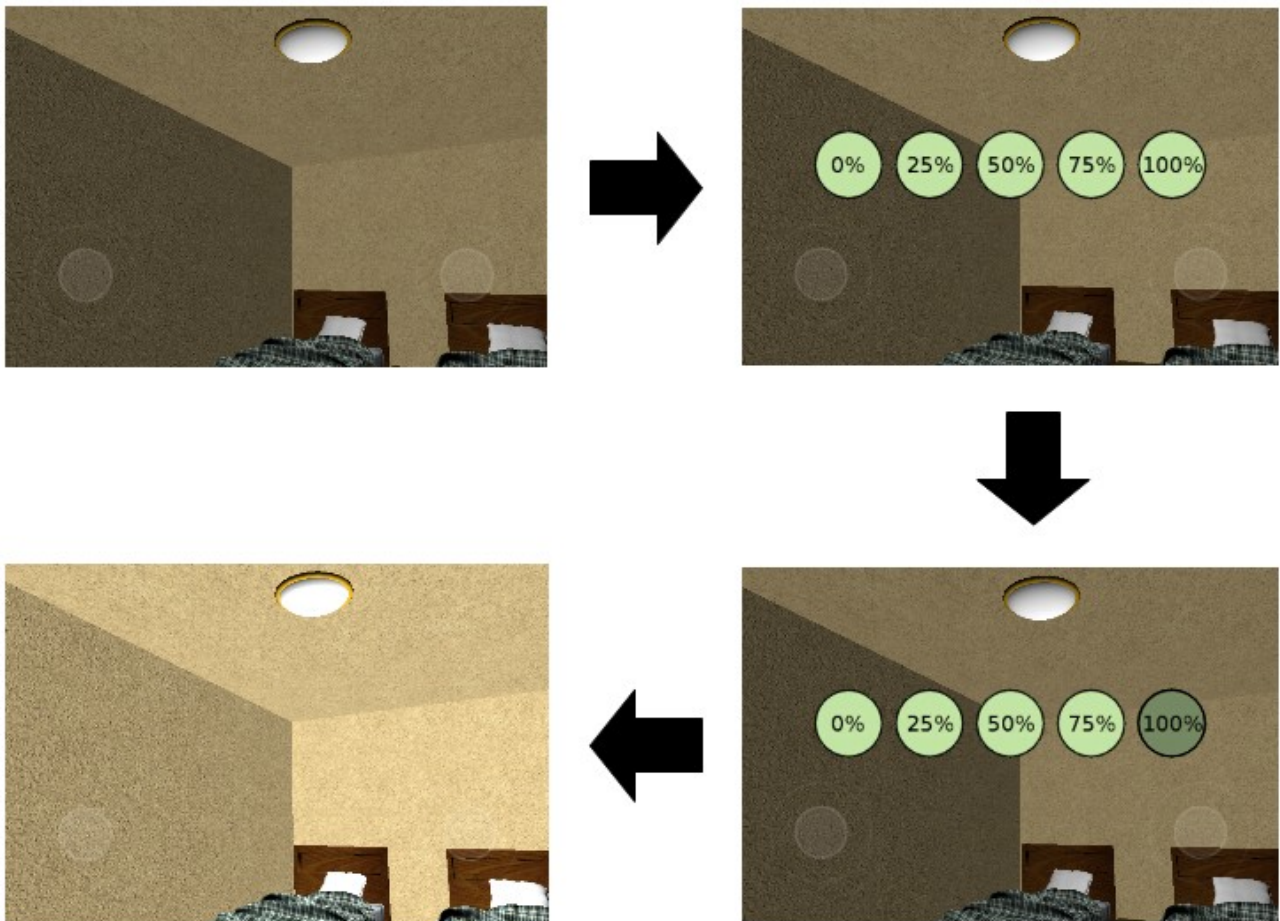


Figura 88. Ejemplo de interacción con la luz gradual

Capítulo 6

Conclusiones

3D Home es una aplicación móvil para el control de hogares inteligentes. Se ha realizado una aproximación distinta buscando un control fácil e intuitivo. Para ello, el usuario maneja una representación de sí mismo en primera persona dentro de una representación virtual de una casa. Esta aproximación hace que sea preferible usar un motor gráfico para crear dichas representaciones. Por esta razón, se ha usado un motor de escenarios 3D interactivos que, entre otras herramientas, ofrece un motor gráfico. Este motor ha permitido hacer gran parte de la aplicación de forma visual, aunque algunos aspectos como la comunicación con el hogar inteligente se ha necesitado un poco de programación. A lo largo del desarrollo, se han encontrado varios problemas, algunos de ellos se han podido resolver satisfactoriamente y otros no.

El resto del capítulo está organizado como sigue: la sección 6.1 explica los problemas encontrados en la realización de 3D Home y la sección 6.2 describe algunos posibles trabajos para la mejora de la aplicación.

6.1. Problemas Encontrados

Al realizar el trabajo propuesto por la tesina, presentado tres problemas principalmente.

El primero de ellos es que el desarrollo de 3D Home ha implicado conocer distintas herramientas que ofrece UDK Mobile. Una de estas herramientas, UnrealEd, es básica sin embargo hay otras que dependiendo de la aplicación a desarrollar se requerirán de su uso, lo que implica también aprenderla, o no. Ejemplos de las herramientas son Matinee que sirve para hacer animaciones y SpeedTree que sirve para simular la interacción de los árboles y vegetación con distintos elementos como el viento y las explosiones.

Además de UnrealEd que sirve para crear el escenario, ha sido necesario aprender Unreal Matinee, Unreal Kismet y UnrealScript. Unreal Matinee y Unreal Kismet son dos editores visuales para crear animaciones y definir el comportamiento de la aplicación, mientras que UnrealScript es un lenguaje de programación para modificar elementos existentes de UDK Mobile o crear nuevos elementos. Estas herramientas no son todas las que ofrece UDK Mobile, pero sí las necesarias para el desarrollo de 3D Home.

UDK Mobile ofrece la posibilidad de importar modelos para usarlos como mallas estáticas, pero no para crearlos. Así que, además de aprender las herramientas que ofrece UDK Mobile, ha sido necesario aprender herramientas externas para poder crear los modelos necesarios para 3D Home.

Los otros dos problemas, ausencia de carga del estado de la maqueta y el mal funcionamiento de las luces dinámicas, han surgido por las limitaciones de las herramientas utilizadas.

Respecto al primero de estos problemas, la maqueta usada para hacer las pruebas de 3D

Home no guarda estado. Esto significa que no guarda en ningún archivo cuál es el estado actual de cada dispositivo de la maqueta. Así, para poder realizar las pruebas de la aplicación, se ha programado un sistema de guardado/carga de estado de 3D Home con UnrealScript.

Respecto al segundo problema, en esta tesina se usan muchas luces dinámicas para la iluminación del escenario. Una luz dinámica es una luz que es capaz de cambiar sus propiedades, como el color, en tiempo de ejecución. Aunque UDK Mobile proporcione estas luces, actualmente no garantiza su buen funcionamiento. No se puede hacer nada ante este mal funcionamiento salvo esperar a que una de las actualizaciones de Epic solviente este problema.

6.2. Trabajos Futuros

El trabajo realizado en esta tesina no está cerrado y se pueden introducir varios futuros trabajos. A continuación se explican estos trabajos.

Un usuario se puede encontrar con que realiza la misma tarea todos los días. Por ejemplo, encender el aire acondicionado en el verano para enfriar la temperatura. **Una de las posibles mejoras de 3D Home sería ofrecer la posibilidad de automatizar estas tareas.** Las tareas automatizadas aumentan el confort del usuario y reducen el consumo de recursos naturales como la energía y el agua. Sin embargo, el usuario debe ser capaz de configurar qué tareas desea automatizar y cómo quiere que sean realizadas. Así, se evita que el usuario tenga que soportar comportamientos no deseados de los dispositivos automatizados.

Otro trabajo futuro sería la personalización del hogar por parte del usuario. Esta personalización permitiría al usuario modificar el color y textura de las paredes o el suelo. Una personalización que podría plantearse a partir de una fotografía de un material realizada por el usuario. Así en 3D Home, con esta nueva funcionalidad, el usuario entraría en la habitación a pintar, indicaría que desea pintar las paredes de la habitación y haría una foto al nuevo material. 3D Home buscaría un patrón de repetición en la fotografía y pintaría las paredes de la habitación con un mosaico del patrón.

Con el desarrollo de 3D Home se ha visto que 3D Home solo cubre una casa en particular. Si se quiere controlar una vivienda con otra distribución de habitaciones, se debe crear otra representación virtual con UDK Mobile. **Un posible trabajo futuro sería desarrollar una herramienta que ayudara a crear una vivienda a partir de una especificación.** Esta herramienta tendría como entrada la especificación de la vivienda a crear y como salida uno o más objetos que podrían ser usados por UDK Mobile. Estos objetos se usarían para construir, con UDK Mobile, la vivienda propuesta por la especificación.

Finalmente, **el último trabajo que se plantea es la realización de pruebas con el usuario final sobre el propio dispositivo móvil.** No se han podido hacer dichas pruebas porque para poder crear el instalador de 3D Home es necesario tener una licencia de desarrollador iOS. Estas pruebas permitirían conocer la usabilidad, saber si los usuarios identifican bien los elementos de interacción y el nivel de efectividad con el que esta aplicación permite controlar los elementos de interacción y, en consecuencia, los dispositivos reales de la maqueta.

Anexo A: Unreal Development Kit Mobile

Unreal Development Kit Mobile (a partir de ahora, UDK Mobile) [4] tiene los siguientes componentes:

- Motor gráfico: controla lo que el jugador ve mientras interactúa con un escenario. Controla aspectos como la apariencia física de los objetos, la transparencia u opacidad de los objetos, qué objetos tapan a qué otros y la iluminación.
- Motor de sonido: controla todo el sonido que el jugador recibe. Controla aspectos como el sonido de los pasos del personaje que lleva el jugador, de las balas al ser disparadas o al chocar contra una pared, y del viento. También es el responsable de cambiar la música cuando la acción cambia, por ejemplo no es la misma música cuando exploras una habitación de forma sigilosa que cuando te enfrentas a un grupo de enemigos.
- Motor de físicas: integra las leyes de la física en el escenario.
- Gestor de entradas: su función es reconocer la entrada del jugador, independientemente del dispositivo de entrada que use (mando de videoconsola, joystick, teclado, etc), para posteriormente convertirla en una orden que dice al escenario cómo reaccionar.
- Infraestructura de red: permite la comunicación cliente-servidor para, por ejemplo, las partidas multijugador.
- El intérprete UnrealScript: UnrealScript es un lenguaje de *script* que permite al usuario ajustar cualquier cosa del motor. El intérprete UnrealScript transforma los *scripts* creados en código que el motor pueda procesar.

Todos los activos de escenario de UDK Mobile, excepto los mapas, son almacenados en paquetes. Estos paquetes son usados por UDK Mobile como contenedores para cargar los activos de aplicación. Los activos de escenario pueden ser:

- Mapas: también llamados niveles o escenarios, son donde creas los entornos en Unreal Engine. Más en concreto, un mapa es una colección de varios activos de escenario que son mostrados al jugador.
- Texturas y materiales: las texturas son imágenes 2D, creadas con herramientas externas de edición de imágenes digitales como Gimp o Photoshop. Un material es una pintura que se aplica a las superficies de los objetos del escenario. Toda superficie visible que se crea tiene un material aplicado, y los materiales contienen al menos una textura. Ejemplos de texturas son la textura difusa, que pone el color al objeto, y el mapa normal, que hace que el objeto parezca mucho más complejo de lo que realmente es. En la figura 89 se muestra cómo cambia el aspecto de un objeto cuando se aplica un mapa normal.

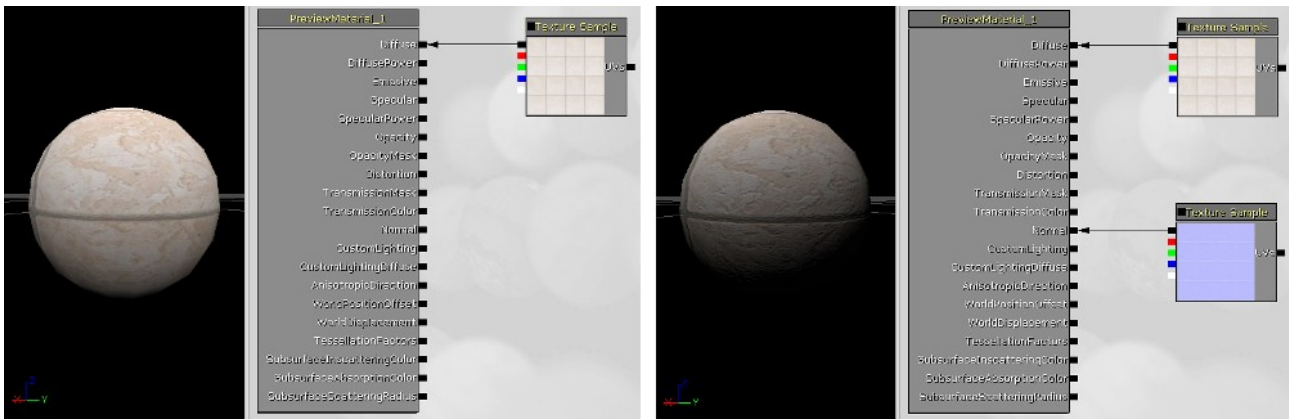


Figura 89. Ejemplo de material con textura difusa (izq.) y con textura difusa y mapa normal (der.)

- Sonidos: los sonidos son creados con herramientas externas de adquisición o edición de audio como Audacity o Cubase, para luego ser importados a UDK Mobile. Estos sonidos pueden luego ser mezclados o manipulados con SoundCue Editor y Kismet, ambas herramientas son de UDK Mobile.
- Mallas estáticas: los objetos 3D son construidos con mallas poligonales (colección de vértices, aristas y caras que definen la forma de los objetos). Las mallas estáticas son un tipo de mallas poligonales que renderizan muy rápido y pueden ser usadas para crear formas muy complejas, sin embargo no puedes animar sus vértices. Además, pueden ser instanciadas múltiples veces sin requerir tiempo de CPU porque las dibuja la tarjeta de vídeo.
- Animaciones y mallas esqueléticas: los personajes del escenario normalmente no son mallas estáticas, son mallas esqueléticas. Son mallas que se deforman basándose en un esqueleto digital. El proceso de introducir el esqueleto en la malla se hace con una herramienta externa de animación 3D como 3ds Max o Maya. Las mallas y sus animaciones están almacenadas de forma separada en paquetes, así cualquier personaje con una estructura esquelética similar puede usar la misma animación.

UDK Mobile nos ofrece varias herramientas para crear nuestros niveles, éstas son:

- UnrealEd: permite crear visualmente el escenario donde se moverá el personaje.
- Unreal Kismet: accesible desde UnrealEd, permite definir la jugabilidad del nivel.
- Unreal Matinee: accesible desde Unreal Kismet, permite definir las animaciones del escenario.
- Unreal Script: permite programar nuevos *scripts* para usarlos en el nivel.

Anexo B: UnrealEd

Visión General

UnrealEd es un conjunto de herramientas para trabajar con el contenido en Unreal Engine [15]. Es usado para el diseño de niveles, pero también tiene editores y exploradores para importar y manipular contenido.

Unreal Level Editor

Unreal Level Editor es la herramienta de edición principal central de UnrealEd. En este editor es donde se crean los niveles. La interfaz del editor de niveles, que se muestra en la figura 90, está dividido en:

1. Barra de menús.
2. Barra de herramientas.
3. Caja de herramientas.
4. Vistas: ofrece las vistas frontal, cenital, lateral y en perspectiva.
5. Barra de estado.

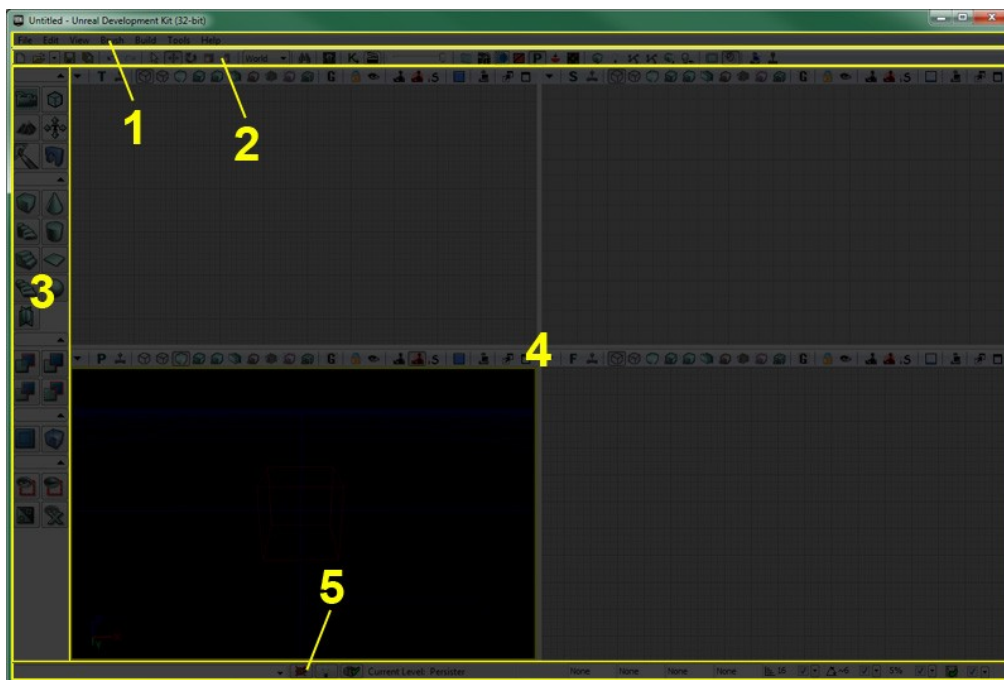


Figura 90. Interfaz de Unreal Level Editor

Creación de Niveles

UnrealEd proporciona dos estilos para hacer niveles, el estilo aditivo y el estilo sustractivo. En el primer estilo, el escenario está vacío y con los pinceles mostrados en la figura 91 se va añadiendo la geometría. En el segundo estilo el escenario está completamente lleno de geometría y con los pinceles se "excava" para crear el nivel deseado. Este último estilo es útil si se quiere representar escenarios subterráneos. Por defecto, se crean los niveles con un estilo Aditivo.

BSP es el casco del nivel. Forma la mayoría de las piezas grandes geométricas del nivel. Para añadir geometría BSP, se usa uno de los Builder Brush que nos proporciona UnrealEd. En las vistas aparece el pincel en rojo. Por defecto se usa el cubo. A continuación se muestra una imagen antes de añadir la geometría al nivel, y otra imagen donde se ha añadido la geometría al nivel donde además se ha apartado el pincel rojo para una mejor visualización.

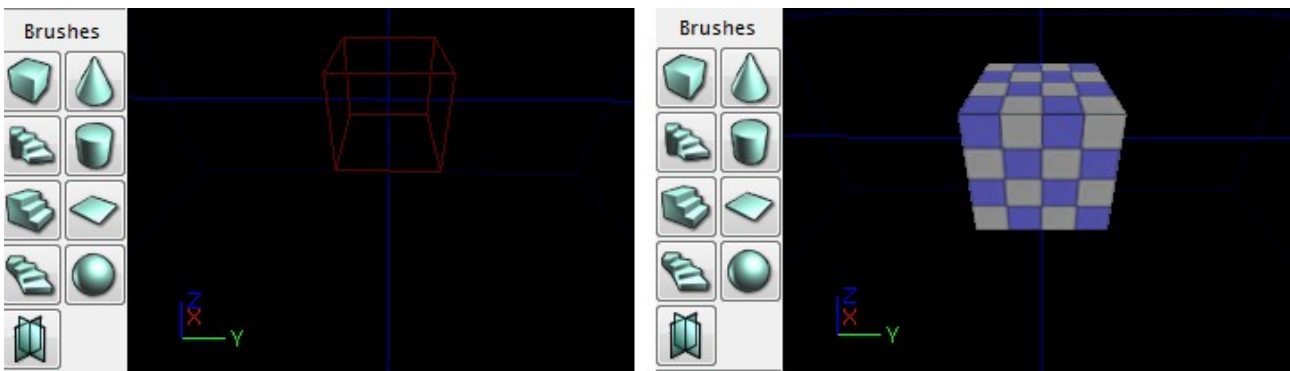


Figura 91. Pinceles BSP y vista del pincel antes (izq.) y después (der.) de añadir la geometría al escenario

Además es necesario añadir al menos una luz si se quiere visualizar algo mientras se juega, además de la posición y orientación iniciales del jugador como se muestra en la figura 92.

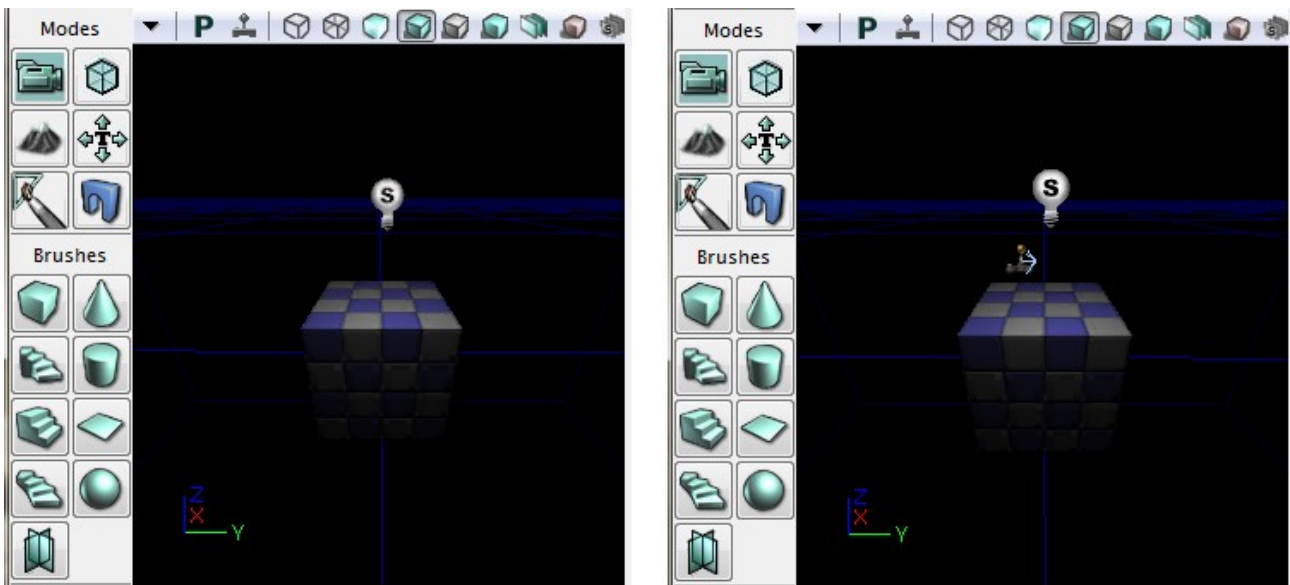


Figura 92. Luz (izq.) y posición inicial del jugador (der.) añadidas al escenario

Además, hay que aplicar los materiales deseados a la geometría BSP. Y por último, añadir el resto de elementos, como por ejemplo las mallas estáticas. Esto se muestra en la figura 93.

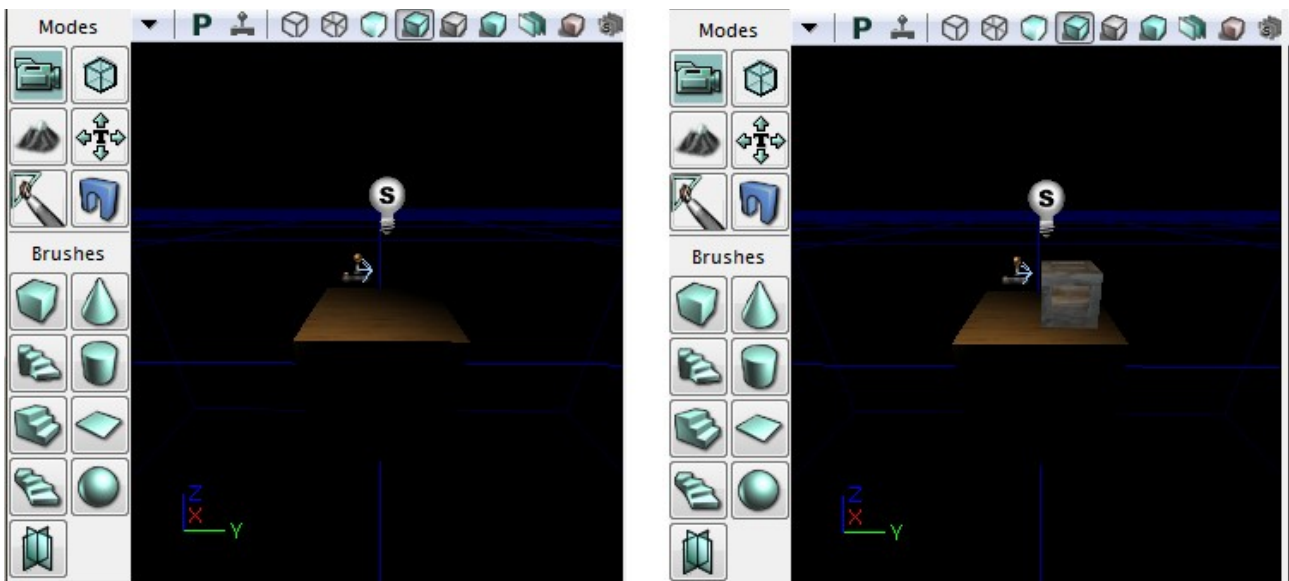


Figura 93. Materiales aplicados (iqz.) y malla estática añadida (der.) al escenario

Aún así, éste no será el aspecto que tendrá el nivel. Para ver el aspecto que tendrá, hay que compilar el nivel. En la figura 94 se muestra el aspecto del nivel después de la compilación.

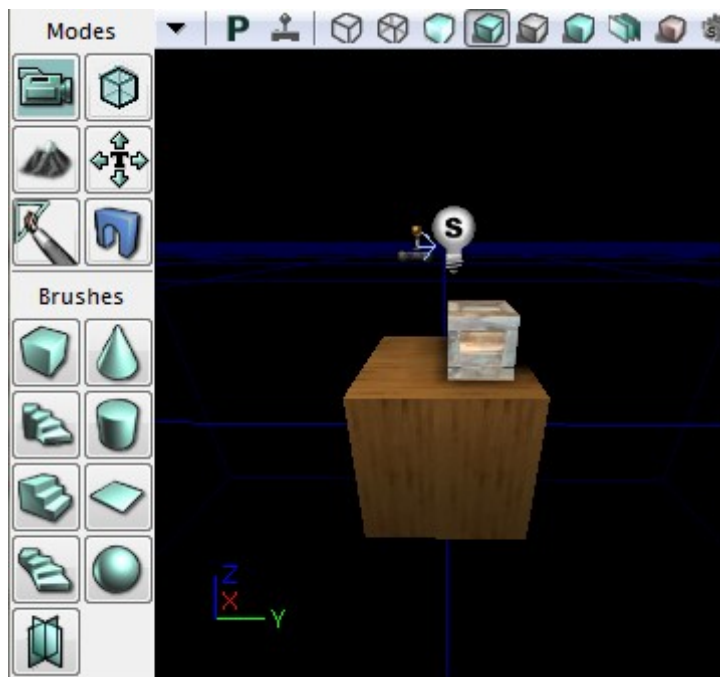


Figura 94. Escenario compilado

Anexo C: Unreal Kismet

Visión General

Unreal Kismet es un sistema visual de scripting que permite a los diseñadores describir el flujo complejo de jugabilidad del nivel [14]. Kismet se puede ver como una red de módulos, conocidos como objetos de secuencia, conectados por cables. Cada objeto de secuencia realiza una función específica, y los cables se usan para transmitir información de un objeto de secuencia a otro. En la tabla 6 se explican los distintos tipos de objetos de secuencia, y en la tabla 7 se muestran los distintos tipos de variables.

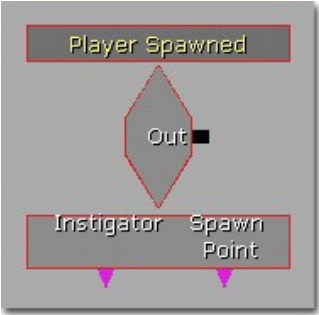

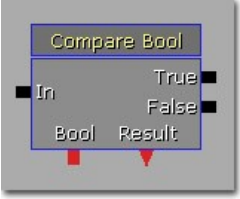

Evento		<p>Los eventos son objetos que crean una "entrada" en tu secuencia, probablemente desde un actor en el escenario. Son representados por un diamante rojo.</p>
Acción		<p>Las acciones son objetos que realizan alguna acción en los actores de tu nivel. Representados por una caja con entradas a la izquierda, salidas en la derecha y conexiones de variables en la parte inferior.</p>
Condición		<p>Las condiciones son objetos que afectan al control del flujo de tu secuencia.</p>
Variable		<p>Las variables son objetos que almacenan información de un determinado tipo para usar en un evento, acción o condición. Son representados como un círculo coloreado.</p>

Tabla 6. Tipos de objetos de secuencia

Rojo	Booleano (<i>true</i> o <i>false</i>)
Azul	Número real (Ej: 1.45)
Cian	Número entero (Ej: 7)
Verde	Cadena de caracteres (Ej: "Foo")
Oro	Vector (Ej: (0.3, 4.56, 8.3))
Púrpura	Referencia a un objeto (Ej: un actor en el nivel), lista de objetos, volumen de objeto o jugador
Naranja	Datos de Matinee
Negro	Variable externa o variable normal que está sin enlazar.

Tabla 7. Colores de variables y su tipo asociado

En Unreal Kismet, algunas propiedades de los objetos de secuencia solo pueden ser modificadas desde el editor gráfico, otras han de ser modificadas desde la ventana de propiedades, y otras desde cualquiera de las dos formas. En la ventana de propiedades, las propiedades que han sido cambiados sus valores por defecto aparecen en negrita como se muestra en la figura 95.

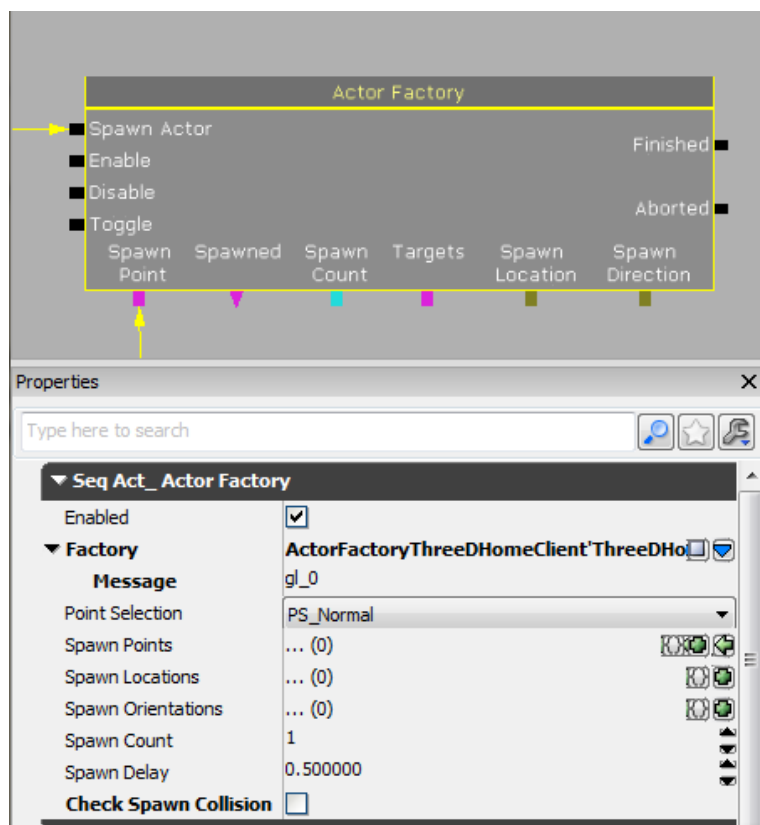


Figura 95. Ejemplo de propiedades editables de un objeto de secuencia

En el editor gráfico, las propiedades no usadas de los objetos de secuencia pueden ser escondidas para facilitar la lectura del modelo. También se pueden esconder o mostrar las propiedades que interesen como se muestra en la figura 96.



Figura 96. Aspecto de Actor Factory por defecto (izq.) y con las propiedades no usadas escondidas (der.)

Referencia

Unreal Kismet tiene muchos objetos de secuencia pero solo se van a explicar los objetos de secuencia, propiedades y enlaces (de entrada, de salida y de variables) que se han usado para el proyecto [13][12].

Propiedades comunes

Obj Comment

Todos los objetos de secuencia comparten algunas propiedades. Obj Comment es una de ellas, y sirve para comentar el objeto. El comentario aparece en la parte superior del objeto comentado.



Figura 97. Ejemplo de objeto comentado

Acciones

Enlaces comunes

Los siguientes enlaces son comunes a muchas acciones.

- **Enlace de entrada in:** activa la acción.
- **Enlace de salida out:** cuando la acción finalice, activará el *Sequence Object* que esté conectado a este enlace.

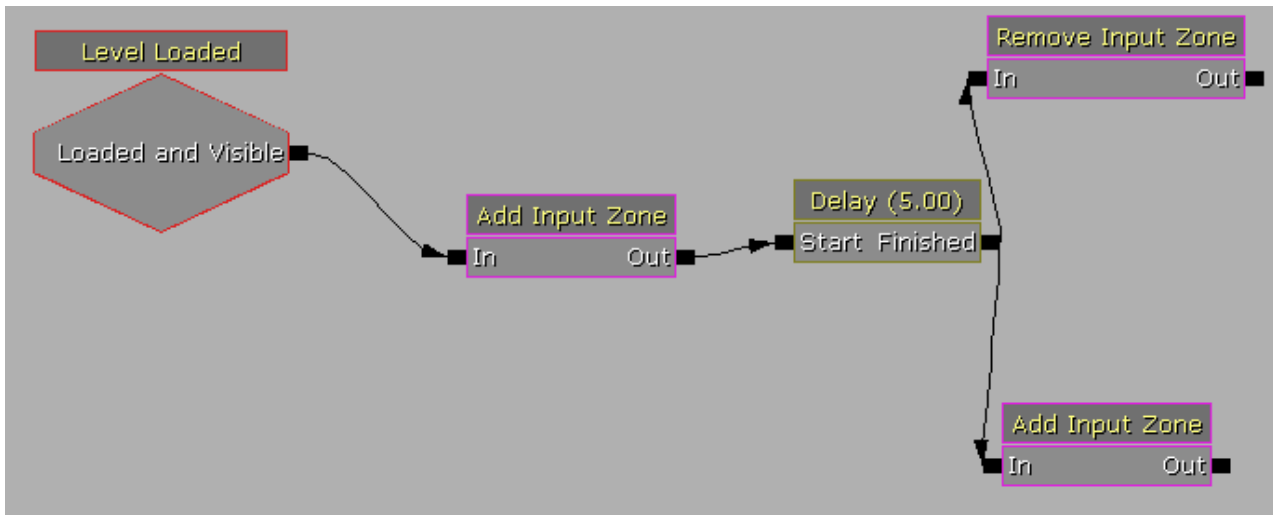


Figura 98. Ejemplo de uso de los enlaces *in* y *out*

Actor Factory

Esta acción se usa para crear objetos en el nivel durante el tiempo de ejecución. Algunos ejemplos serían efectos de partículas o personajes controlados por una Inteligencia Artificial. Esta acción instancia un tipo particular de un factoría de actores que se encarga de crear un cierto tipo de actor.

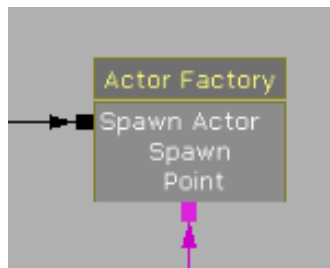


Figura 99. Ejemplo de Actor Factory

• Propiedades

- **Check Spawn Collision:** Si está activada, se comprueba si el actor que se crea colisiona con cualquier objeto del nivel. Si colisionara, entonces no se crearía.
- **Enabled:** Si está activada, la acción está activa y creará actores.
- **Factory:** La factoría de actores que se usa para la creación de actores de esta acción.
- **Spawn Count:** El número total de actores a crear. Coincide con el enlace de variable *Spawn Count*.
- **Spawn Points:** Un vector de actores para usar como localizaciones y orientaciones para crear nuevos actores. Corresponde al enlace de variable *Spawn Point*.

• Enlaces de entrada

- **Spawn Actor:** Activar esta entrada causa la creación de uno o más actores, determinado por *Spawn Count*.

• Enlaces de variable

- **Spawn Point:** Establece los Actores a usar para crear las localizaciones y orientaciones. Coincide con la propiedad *Spawn Points*.

Add Input Zone

Esta acción añade una zona de entrada única en la pantalla. Esta zona permite recoger información táctil.

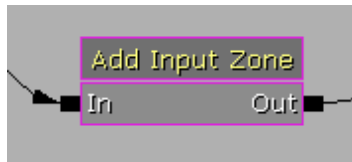


Figura 100. Ejemplo de Add Input Zone

• Propiedades

- **Zone Name:** Especifica el nombre de la nueva zona de entrada. Es importante para referenciarlo en los eventos Kismet de entrada.
- **New Zone:** Crea una nueva zona de entrada.
 - **Type:** Especifica el tipo de zona de entrada: *button*, *joystick*, *trackball*, *slider*.
 - **Size X:** Establece la anchura de la zona de entrada.
 - **Size Y:** Establece la altura de la zona de entrada.
 - **Relative X:** Si está seleccionado, la posición horizontal de la esquina superior izquierda de la zona se asumirá que está entre el rango [0.0, 1.0].
 - **Relative Y:** Si está seleccionado, la posición vertical de la esquina superior izquierda de la zona se asumirá que está entre el rango [0.0, 1.0].
 - **Relative Size X:** Si está seleccionado, la anchura de la zona de la zona se asumirá que está entre el rango [0.0, 1.0].
 - **Relative Size Y:** Si está seleccionado, la altura de la zona se asumirá que está entre el rango [0.0, 1.0].
 - **Override Texture 1:** Especifica la ruta completa a la textura que renderizará en la zona de entrada. En el caso de la zona *button* será lo que mostrará cuando el botón no esté presionado.
 - **Override Texture 2:** Especifica la ruta completa a la textura que renderizará en la zona de entrada. En el caso de la zona *button* será lo que mostrará cuando el botón esté presionado.

Add Int

Esta acción suma dos variables de tipo entero y devuelve el resultado.

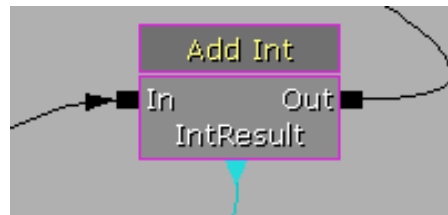


Figura 101. Ejemplo de Add Int

- **Propiedades**

- **Value A:** Establece el primer entero a sumar. Coincide con el enlace de variable A.
- **Value B:** Establece el segundo entero a sumar. Coincide con el enlace de variable B.

- **Enlaces de variable**

- **Int Result:** Saca el resultado de la suma como un valor entero.

Delay

Esta acción causa un retraso en una secuencia con una duración variable. Los retrasos pueden ser pausados, reiniciados y abortados.



Figura 102. Ejemplo de Delay

- **Propiedades**

- **Duration:** Establece el tiempo (en segundos) que la acción esperará antes de disparar el enlace de salida *Finished*. Coincide con el enlace de variable *Duration*.

- **Enlaces de entrada**

- **Start:** Empieza el temporizador para el retraso.

- **Enlaces de salida**

- **Finished:** Se dispara cuando el temporizador llega a su fin.

Draw Text

Esta acción muestra texto en pantalla en tiempo de ejecución.

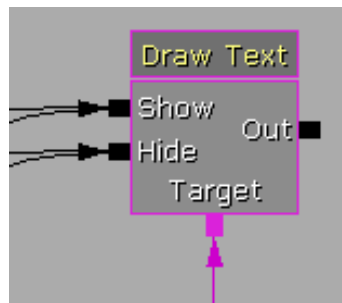


Figura 103. Ejemplo de Draw Text

- **Propiedades**

- **Display Time Seconds:** Establece la cantidad de tiempo que se enseña el mensaje. Si es negativo, el mensaje se enseña hasta que es escondido.
- **Draw Text Info**
 - **Message Color:** Establece el color a usar para dibujar el texto.
 - **Message Font:** Establece la fuente a usar para dibujar el texto.
 - **Message Font Scale:** Establece el tamaño a usar para dibujar el texto.
 - **Message Offset:** Establece el desplazamiento a usar cuando se dibuja el texto.
 - **Message Text:** Establece el texto a mostrar.

- **Enlaces de entrada**

- **Show:** Muestra el mensaje de texto.
- **Hide:** Para de mostrar el mensaje de texto.

- **Enlaces de variable**

- **Target:** Establece el actor a quien se va a mostrar el mensaje.

Play Sound

Esta acción reproduce un *SoundCue* especificado.



Figura 104. Ejemplo de Play Sound

- **Propiedades**

- **Play Sound:** Establece la *SoundCue* a reproducir.

- **Enlaces de entrada**

- **Play:** Reproduce el sonido especificado.
- **Stop:** Para la reproducción del sonido.

Remove Input Zone

Esta acción elimina una zona de entrada de la pantalla.

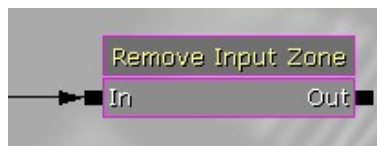


Figura 105. Ejemplo de Remove Input Zone

- **Propiedades**

- **Zone Name:** Especifica el nombre de la zona de entrada a eliminar.

Set Int

Esta acción establece el valor de una variable entera usando el valor de otra variable entera.

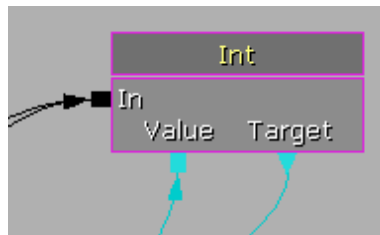


Figura 106. Ejemplo de Set Int

- **Propiedades**

- **Value:** Establece una lista de valores, la suma de la cual es usada para establecer el valor de la variable objetivo. Coincide con el enlace de variable *Value*.

- **Enlaces de variable**

- **Value:** Establece una lista de valores, la suma de la cual es usada para establecer el valor de la variable objetivo. Coincide con la propiedad *Value*.
- **Target:** Establece la variable objetivo donde se establecerá el valor.

Switch

Esta acción tiene un número definible de salidas, e incrementará a la siguiente salida cada vez que el *switch* sea disparado.

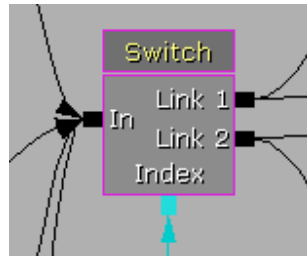


Figura 107. Ejemplo de Switch

- **Propiedades**

- **Increment Amount:** Establece la cantidad a incrementar de las variables enlazadas cuando se activa el *switch*.
- **Indices:** Guarda el índice del enlace de salida actualmente activo.
- **Link Count:** Establece el número de enlaces de salida que se quieren exponer.
- **Looping:** Si está seleccionado, el *switch* volverá a la primera salida cuando se haya alcanzado el máximo número de salidas.

- **Enlaces de entrada**

- **In:** Causa que el siguiente enlace de salida sea disparada.

- **Enlaces de variable**

- **Index:** Establece o saca el índice del enlace de salida actualmente activo.

Matinee

Crea una nueva secuencia Matinee. El editor de Matinee se explica con más detalle en el apartado 7.4.

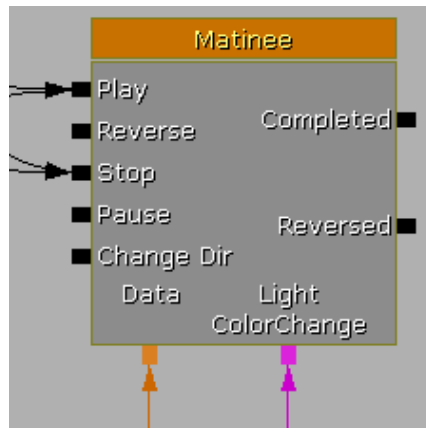


Figura 108. Ejemplo de una secuencia Matinee

- **Enlaces de entrada**

- **Play:** Causa que la animación de la secuencia Matinee se reproduzca hacia delante.
- **Reverse:** Causa que la animación de la secuencia Matinee se reproduzca hacia atrás.
- **Stop:** Causa que la animación de la secuencia Matinee se pare.
- **Pause:** Causa que la animación de la secuencia Matinee se pause. Disparando esta entrada una segunda vez hace que la animación continúe.
- **Change Dir:** Causa que la animación de la secuencia Matinee cambie la dirección actual (hacia delante o hacia atrás).

- **Enlaces de salida**

- **Completed:** Se dispara cuando la secuencia Matinee llega al final de la animación mientras se está reproduciendo hacia delante.
- **Reversed:** Se dispara cuando la secuencia Matinee llega al principio de la animación mientras se está reproduciendo hacia atrás.

- **Enlaces de variable**

- **Data:** Establece el objeto Matinee Data para este Matinee.
- **Group Variable Inputs:** Estas entradas de variables no existen inicialmente. Son creados mediante la generación de grupos dentro del objeto de secuencia de Matinee. Cada grupo es una animación, y los actores enlazados a ese grupo serán los que sean animados con la animación de dicho grupo.

Variables

Int

Esta variable guarda una variable entera.

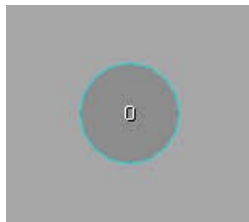


Figura 109. Ejemplo de una variable Int

Matinee Data

Esta variable es un tipo de variable especial diseñado para guardar datos Matinee. Esto permite que diferentes secuencias Matinee puedan compartir los mismos datos.



Figura 110. Ejemplo de una variable Matinee Data

Named Variable

Esta variable permite pasar el valor de otra variable a esta variable basada en emparejar el *FindVarName* con el valor *VarName* asignada a la variable original. En resumen, apunta a otra variable.



Figura 111. Ejemplo de una Named Variable sin emparejar (izq.) y emparejada (der.)

Object

Esta variable guarda una referencia a cualquier objeto en el nivel, incluidos los creados desde Kismet.



Figura 112. Ejemplo de una variable Object

Player

Esta variable guarda una referencia al jugador o jugadores de la partida.



Figura 113. Ejemplo de una variable Player

Eventos

Level Loaded

Este evento disparará el enlace de salida *Loaded and Visible* cuando el nivel actual haya sido cargado y visible.



Figura 114. Ejemplo de Level Loaded

- **Enlaces de salida**

- **Loaded and Visible:** Se dispara cuando el nivel es cargado y visible.

Mobile Button Access

Este evento engancha hasta una *Input Zone* de tipo *Button*.

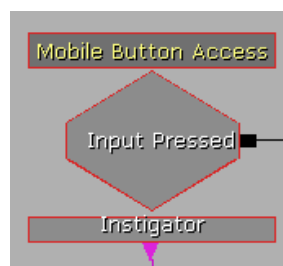


Figura 115. Ejemplo de Mobile Button Access

- **Propiedades**

- **SeqEvent_MobileButton**
 - **Send Press Only On Touch Down:** Si está seleccionado, la salida Input Pressed solo se activará cuando el botón sea presionado.
 - **Send Press Only On Touch Up:** Si está seleccionado, la salida Input Pressed solo se activará cuando el botón sea presionado y despresionado.

- **SeqEvent_MobileZoneBase**
 - **Target Zone Name:** Guarda el nombre de la zona de entrada que estará asociada a este evento.
- **Enlaces de salida**
 - **Input Pressed:** Se dispara cuando se considere que el botón está siendo presionado. Depende de qué evento esté activo: *Send Press Only On Touch Down* o *Send Press Only On Touch Up*.
- **Enlaces de variable**
 - **Instigator:** Devuelve como salida el jugador responsable de la entrada táctil.

Mobile Object Picker

Este evento permite indicar si se ha hecho un toque (entrada táctil) sobre un objeto específico del nivel.

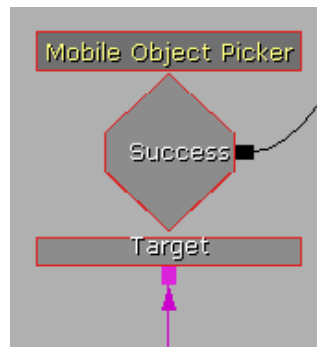


Figura 116. Ejemplo de Mobile Object Picker

- **Enlaces de salida**
 - **Success:** Se dispara cuando un toque ocurre sobre el objeto objetivo.
- **Enlaces de variable**
 - **Target:** Especifica el objeto a comprobar para los toques.

Comment

Comment (Wrap)

Permite crear una caja de comentario para proporcionar una descripción.

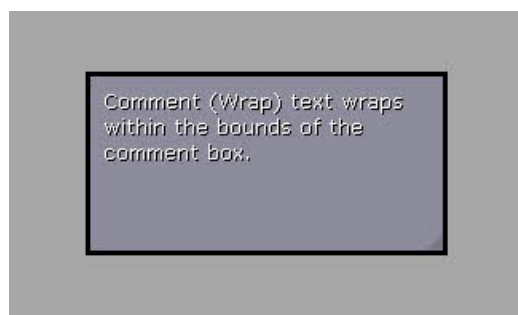


Figura 117. Ejemplo de Comment

- **Properties**
 - **Size X:** Establece la anchura de la caja de comentario.
 - **Size Y:** Establece la altura de la caja de comentario.

Nuevos objetos de secuencia

Para este proyecto se han creado nuevos objetos de secuencia. En el anexo E se explicarán con más detalle.

Load Game

Esta acción permite cargar el estado del nivel. Si no hay ningún estado guardado, empezará el nivel con el estado inicial.

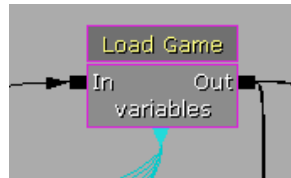


Figura 118. Ejemplo de Load Game

- **Enlaces de variable**

- **variables:** Se indica las variables de las cuales depende el estado del nivel.

Save Game

Esta acción permite guardar el estado del nivel.

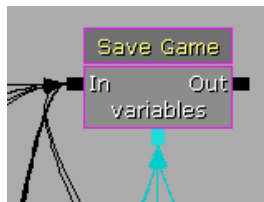


Figura 119. Ejemplo de Save Game

- **Enlaces de variable**

- **variables:** Se indica las variables de las cuales depende el estado del nivel.

Anexo D: Unreal Matinee

Visión General

Matinee es una herramienta para cambiar las propiedades de los actores del nivel a lo largo del tiempo [11]. Esto se hace mediante el uso de fotogramas clave. En otras palabras, sirve para animar a los actores.

La interfaz de Matinee, que se muestra en la figura 120, se compone de cinco partes:

1. Barra de menús.
2. Barra de herramientas.
3. Editor de curvas: permite, de forma gráfica, visualizar y editar las curvas de animaciones usadas por las pistas en la secuencia Matinee.
4. Panel de línea temporal: contiene una lista de todas las carpetas, grupos y pistas contenidas dentro de la secuencia Matinee y muestra la información de los fotogramas clave en la línea temporal donde pueden ser editados.
5. Panel de propiedades: muestra las propiedades de la carpeta, grupo o pista seleccionada en el panel de línea temporal.

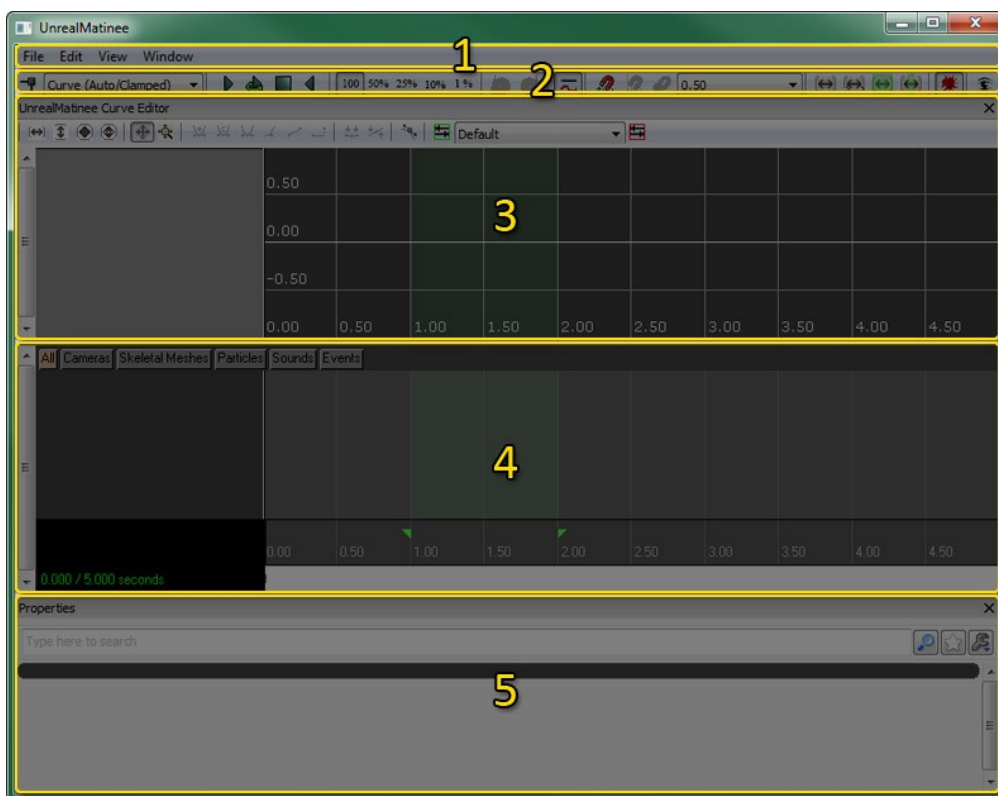


Figura 120. Interfaz de Unreal Matinee

Editor de Curvas

El editor de curvas se compone de tres partes:

1. Barra de herramientas.
2. Lista de pistas: muestra todas las pistas de curva.
3. Gráfico: es una representación gráfica de la curva con el tiempo.



Figura 121. Interfaz del editor de curvas

Lista de pistas

Cada pista muestra el nombre de la propiedad asociado con la pista. Además también muestra botones conmutables para cada curva individual en la pista, y un botón conmutable de visibilidad general. Los botones conmutables para las curvas individuales están codificadas mediante colores.



Figura 122. Ejemplo de pista

Gráfico

Los ejes de esta representación gráfica se corresponden con el tiempo para el eje horizontal, y con el valor de la propiedad para el eje vertical.

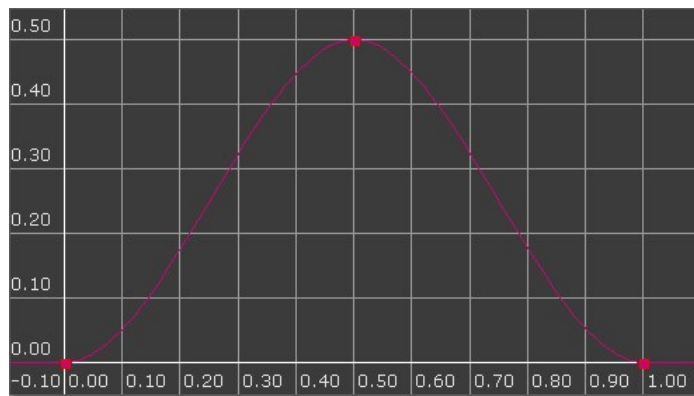


Figura 123. Ejemplo de gráfico

Las fotogramas clave que hay a lo largo de la curva, tres en este caso, son mostrados como puntos que pueden ser seleccionados y manipulados para modificar visualmente la curva.

Panel de Línea Temporal

El panel de línea temporal se compone de:

1. Pestañas de grupo: muestra todas las pestañas de grupo que existen en la secuencia actual de Matinee. Son útiles para organizar grupos y pistas basadas en su función.
2. Lista de grupos/pistas: muestra todos los grupos y pistas de la pestaña actualmente seleccionada.
3. Información de línea temporal.
4. Línea temporal: es una representación gráfica de todos los fotogramas clave de todas las pistas que hay en la secuencia.



Figura 124. Panel de línea temporal

Línea temporal

Muestra el tiempo en el fondo horizontalmente y contiene los marcadores de comienzo y final de la sección de bucle (en verde) y de la secuencia en sí misma (en rojo). Los marcadores que hay en la parte superior son fotogramas clave específicos de cada pista. En el ejemplo de abajo tenemos dos pistas, por eso hay dos filas de fotogramas clave.

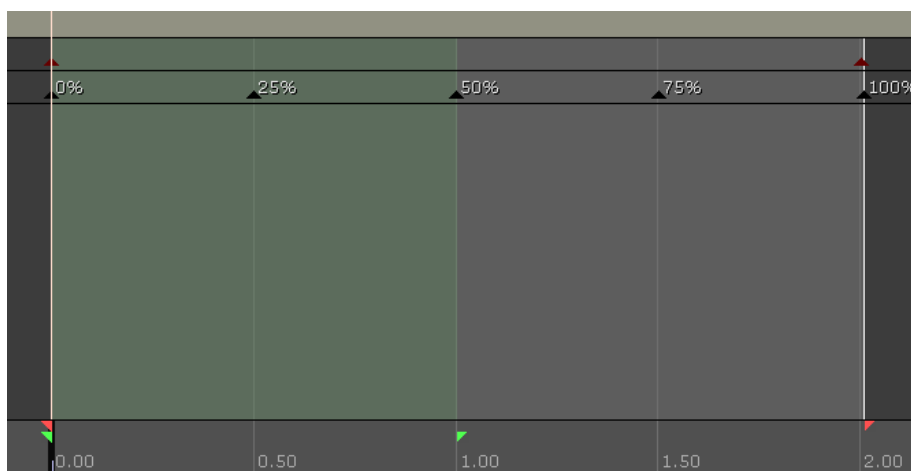


Figura 125. Ejemplo de línea temporal

Anexo E: UnrealScript

Visión General

UnrealScript es un lenguaje integrado de programación orientado a objetos [10]. Los principales aspectos en que destaca UnrealScript son los siguientes:

- Da soporte a los conceptos de tiempo, estado, propiedades y redes. Esto simplifica en gran medida el código UnrealScript.
- Proporciona la simplicidad de programación del estilo de Java, orientación de objetos, y comprobación de errores en tiempo de compilación. Al igual que Java lleva una plataforma de programación clara a los programadores Web, UnrealScript proporciona un lenguaje de programación igualmente claro, simple y robusto para los escenarios 3D interactivos. Los conceptos principales que UnrealScript deriva de Java son:
 - Un entorno sin punteros con la recolección automática de basura.
 - Un grafo de clases simple con herencia única.
 - Un comprobador fuerte de tipos en tiempo de compilación.
 - Una ejecución "*sandbox*" segura del lado cliente. La ejecución *sandbox* es un mecanismo de seguridad para ejecutar programas de forma separada. Normalmente es usada para ejecutar código no probado, o para ejecutar programas sin un origen de confianza.
 - La apariencia familiar del código C/C++/Java.
- Permite una programación rica y de alto nivel en términos de objetos de juego e interacciones en vez de bits y píxeles. Se sacrifica velocidad de ejecución en favor de simplicidad y potencia.

Arquitectura

Para entender el código UnrealScript, es importante entender las relaciones de alto nivel de los objetos en Unreal.

Unreal es puramente orientado a objetos, así que tiene un modelo de objetos bien definido con conceptos como el grafo de objetos, serialización, tiempo de vida de objetos, y polimorfismo. Nuevas funcionalidades y tipos de objeto pueden ser añadidos a Unreal en tiempo de ejecución, y esta extensión toma la forma de crear subclases.

Object es la clase padre de todos los objetos en Unreal. Todas las funciones de la clase *Object* son accesibles en cualquier parte porque todo deriva de *Object*. *Object* es una clase base abstracta. Toda la funcionalidad es proporcionada por las subclases, como *Texture* (un mapa de textura), *TextBuffer* (un trozo de texto), y *Class* (que describe la clase de otros objetos).

Actor (extiende a *Object*) es la clase padre de todos los objetos independientes de Unreal. La clase *Actor* contiene toda la funcionalidad necesaria para que un actor pueda moverse, interactuar con otros actores, afectar el entorno, y otras cosas útiles relacionadas con los juegos.

Pawn (extiende a *Actor*) es la clase padre de todas las criaturas y jugadores de Unreal que son capaces de albergar Inteligencia Artificial de alto nivel y controles de jugador.

Class (extiende a *Object*) es una clase especial de objeto que describe una clase de objeto. Una clase es un objeto, y una clase describe varios objetos.

La relación de las cuatro clases descritas anteriormente se muestra de forma gráfica en la figura 126.

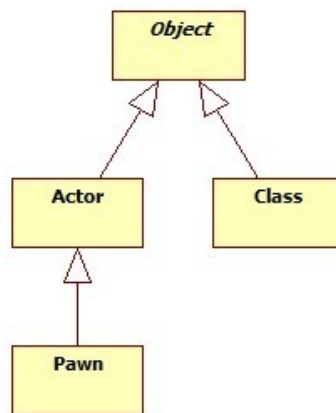


Figura 126. Arquitectura de UnrealScript

Con UnrealScript, el 99% del tiempo dedicado a escribir código, lo hará para una clase derivada de Actor. Esto es así porque la funcionalidad útil de UnrealScript está relacionado con el juego y tiene que ver con actores.

Sintaxis

Solo se explican los elementos de UnrealScript que se han usado en el proyecto.

Clases

Cada script se corresponde con una clase, y el script empieza declarando la clase, la clase padre, y cualquier información adicional que sea relevante para la clase.

```
class MyClass extends MyParentClass;
```

Aquí se está declarando una nueva clase llamada "MyClass", que hereda la funcionalidad de "MyParentClass".

La declaración de clase puede tener especificadores opcionales que afecten a la clase, un ejemplo es:

- **Config(IniName):** indica que esta clase puede almacenar datos en el .ini. Si hay variables configurables en la clase (declaradas con "config" o "globalconfig"), éstas variables son almacenadas en el archivo de configuración especificado.

Variables

Las variables pueden ser de instancia o locales. Las de instancia aparecen inmediatamente después de la declaración de clase y se aplica a todo el objeto, usan la palabra clave `var`. Las locales aparecen dentro de una función y solo se activan cuando se ejecuta la función, usan la palabra clave `local`.

Algunos de los tipos de variable soportados en UnrealScript son:

- **int**: un valor entero de 32 bits.
- **string**: una cadena de caracteres
- **referencia a un actor u objeto**: una variable que hace referencia a otro objeto o actor del nivel.
- **array<Type>**: un vector de variables de tipo `Type`.

Las variables pueden tener especificadores adicionales que describan la variable. Algunos de ellos son:

- **config**: esta variable será configurable. El valor actual puede ser almacenado en el archivo `ini` y será cargado cuando sea creada. No se le puede dar valor en `default properties`. Implica `const`.
- **const**: trata el contenido de la variable como constante, puedes leer el valor pero no escribirlo.
- **private**: solo es accesible por el script de la clase; ninguna otra clase puede acceder a la variable.

En UnrealScript, se puede hacer una variable "editable", para que los usuarios puedan editar el valor de la variable en UnrealEd. Se consigue añadiendo después de la palabra `var`, `(MyCategory)` donde `MyCategory` es la categoría donde quieres que esté la variable, si no se indica nada en `MyCategory` entonces estará en la categoría por defecto.

Expresiones

Para asignar un valor a una variable, se usa `"="`.

Para convertir de forma explícita un tipo de datos a otro, se usa el nombre del tipo seguido entre paréntesis de la variable a convertir.

Ejemplo:

```
function Test()
{
    local int i;
    local string s;

    s = string(i);
}
```

Funciones

En UnrealScript, se puede declarar nuevas funciones y escribir nuevas versiones de funciones existentes (sobreescribir funciones). Las funciones pueden tener uno o más parámetros, y pueden devolver un valor.

Ejemplo:

```
function int Suma (int a, int b)
{
    return a + b;
}
```

La palabra `function` siempre precede a la declaración de la función. Opcionalmente, es seguido del tipo devuelto por la función. Continúa con el nombre de la función, y luego la lista de parámetros de la función entre paréntesis.

Por defecto, cualquier variable local que se declare en una función es inicializada a cero.

Las funciones pueden ser llamadas de forma recursiva.

Algunas funciones UnrealScript son llamadas por el motor cuando suceden ciertos eventos. Por ejemplo, cuando un actor es tocado por otro actor, el motor realiza una llamada a la función *Touch* para decir quién le está tocando.

Cuando se llama a una función, los parámetros son pasados por valor. Si se quieren pasar por referencia, se debe usar el especificador de parámetro `out` justo antes del tipo de parámetro.

Las funciones también pueden tener especificadores. Algunos de ellos son:

- **simulated:** declara que una función puede ejecutarse en el lado cliente cuando un actor es o un proxy simulado o un proxy autónomo.
- **event:** la palabra clave `event` tiene el mismo significado para UnrealScript que `function`. La diferencia está en que con `event` se asegura que el código C++ generado por UnrealEd está sincronizado con las funciones UnrealScript, y elimina la posibilidad de pasar parámetros inválidos a una función UnrealScript.

Estructuras de control

UnrealScript soporta todos los flujos de control estándar de C/C++/Java. Uno de ellos es la estructura `if-then-else`. A continuación se muestra un ejemplo:

```
if( LightBrightness < 20)
    `Log( "My light is dim" );
else if( LightBrightness < 40)
    `Log( "My light is medium" );
else
    `Log( "My light is very bright" );
```


UnrealScript usa `foreach` para facilitar el manejo de grandes grupos de actores. "foreach" trabaja junto con un tipo especial de función llamada "iterator" cuyo propósito es iterar a través de la lista de actores. A continuación se muestra un ejemplo que muestra una lista de todas las luces en el nivel:

```
foreach AllActors(class'Actor', A)
{
    if(A.LightType != LT_None )
    {
        log( A );
    }
}
```

El primer parámetro del `foreach` es una clase que especifica el tipo de actor a buscar. El segundo parámetro es una variable donde se asignará un actor durante cada iteración del bucle `foreach`.

Hay varias funciones iterador de trabajan con "foreach". `AllActors` es una de ellas, e itera a través de todos los actores del nivel.

Funcionalidad del lenguaje

UnrealScript proporciona una amplia variedad de operadores. En la tabla 8 se muestran algunos de estos operadores.

<u>Operador</u>	<u>Tipos a los que se aplica</u>	<u>Significado</u>
\$	String	Concatenación de strings.
==	Todos	Comparar para igualdad
!=	Todos	Comparar para desigualdad

Tabla 8. Ejemplos de operadores de UnrealScript

UnrealScript también proporciona especificadores de llamadas de función. Uno de ellos es `Super`, que llama a la versión correspondiente de la función en la clase padre. Ejemplo:

```
function MyExample (actor Other)
{
    Super.Touch( Other );
}
```

Además, UnrealScript permite llamar a funciones estáticas de una clase específica. Ejemplo:

```
class'SkaarjTrooper'.static.SomeFunction();
```

Por último, para establecer los valores por defecto de las propiedades de un actor, se puede hacer en el bloque `defaultproperties`. Este bloque tiene las siguientes características:

- Las declaraciones no están permitidas.
- Los punto y comas pueden ser colocados al final de cada línea, pero no es necesario.
- Los valores por defecto son heredados por las clases hijas. Los valores especificados en el `defaultproperties` de las clases hijas sobrescriben los valores especificados por las clases

padres.

Referencias

- [1] Acoustica, Acoustica, 2011, <http://www.acoustica.com/mp3-audio-mixer/sounds.htm>
- [2] Barkhuus L., Dey A., Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined, Proceedings of UbiComp 2003, pages 150-156, Springer.
- [3] Barzanallana R., Domótica. Automatización de viviendas, 2007, <http://www.um.es/docencia/barzana/IATS/Iats09.html>
- [4] Busby J., Parrish Z., Wilson J., Mastering Unreal Technology, Volume I: Introduction to Level Design with Unreal Engine 3, Sams,2009
- [5] Cedom, Instaladores de domótica España, 2011, <http://www.cedom.es/instaladores.php>
- [6] Cedom, Qué es domótica, 2011, <http://www.cedom.es/que-es-domotica.php>
- [7] Clauser, G., Smart Homes Growing in Popularity, 2011, http://www.cepro.com/article/smart_homes_growing_in_popularity/
- [8] DeLoura M., The Engine Survey: Technology Results, 2009, http://www.gamasutra.com/blogs/MarkDeLoura/20090316/903/The_Engine_Survey_Technology_Results.php
- [9] DeLoura M., Game Engines at GDC 2010, 2010, http://www.gamasutra.com/blogs/MarkDeLoura/20100308/4616/Game_Engines_at_GDC_2010.php
- [10] Epic Games, UnrealScript Language Reference, 2011, <http://udn.epicgames.com/Three/UnrealScriptReference.html>
- [11] Epic Games, Unreal Matine User Guide, 2011, <http://udn.epicgames.com/Three/MatineeUserGuide.html>
- [12] Epic Games, Kismet Reference, 2011, <http://udn.epicgames.com/Three/KismetReference.html>
- [13] Epic Games, Mobile Kismet Reference, 2011, <http://udn.epicgames.com/Three/MobileKismetReference.html>
- [14] Epic Games, Kismet User Guide, 2011, <http://udn.epicgames.com/Three/KismetUserGuide.html>
- [15] Epic Games, Unreal Editor User Guide, 2011, <http://udn.epicgames.com/Three/UnrealEdUserGuide.html>
- [16] Epic Games, Features of Unreal Engine 3, 2011, <http://www.unrealengine.com/features>
- [17] Gamebryo, Gamebryo, 2011, <http://www.gamebryo.com/>
- [18] GarageGames, Torque Game Engine, 2011, <http://www.garagegames.com/products/torque-3d>
- [19] Gartner, Gartner says Android to command nearly half of worldwide smartphone operating system market by year-end 2012, 2011, <http://www.gartner.com/it/page.jsp?id=1622614>
- [20] Grupo Tecma Red, Domótica - Introducción, 2011, <http://www.casadomo.com/noticiasDetalle.aspx?c=14&idm=21>
- [21] InterArtCenter, Artist-3D.com , 2011, <http://artist-3d.com/>

- [22] Jang S., Shin C., Oh Y., Woo W., Introduction of 'ubiHome' Testbed, Korea · Japan Joint Workshop on Ubiquitous Computing & Networking Systems 2005 (UbiCNS 2005)
- [23] Jordan J., Engines of Creation: An Overview of Game Engines, 2008, http://www.gamasutra.com/view/feature/3832/engines_of_creation_an_overview_.php
- [24] Kim D., Kim D., An Intelligent Smart Home Control Using Body Gestures, 2006 International Conference on Hybrid Information Technology (ICHIT'06)
- [25] Lighthouse3d.com, GLSL 1.2 Tutorial, 2011, <http://www.lighthouse3d.com/tutorials/glsl-tutorial/?ogldir2>
- [26] McLoughlin I., Reza H., Speech Recognition for Smart Homes, inTech,2008
- [27] Möller S., Krebber J., Raake A., Smeele P., Rajman M., Melichar M., Pallotta V., Tsakou G., Kladis B., Vovos A., Hoonhout J., Schuchardt D., Fakotakis N., Ganchev T., Potamitis I., INSPIRE: Evaluation of a Smart-Home System for Infotainment Management and Device Control, 2004, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.3466&rep=rep1&type=pdf>
- [28] Pavlovic V., Sharma R., Huang T., Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, No. 7, July 1997
- [29] Robles R., Kim T., Review: Context Aware Tools for Smart Home Development, International Journal of Smart Home Vol.4, No.1, January, 2010
- [30] Rössler H., Siene J., Wajda W., Hoffmann J., Kostrzewa M., Multimodal Interaction for Mobile Environments, 2001, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.7689&rep=rep1&type=pdf>
- [31] Ruser H., Borodulkin L., Leisner D., Multi-modal 'Smart Home' user interface, 2003, <http://www.borodulkin.de/prof/files/Bor03a.pdf>
- [32] Schilit B., Adams N., Want R., Context-Aware Computing Applications, IEEE Workshop on Mobile Computing Systems and Applications, December 8-9 1994
- [33] Serrano G., Desarrollo de una aplicación iPhone para interactuar con una vivienda domótica, 2010
- [34] Unity Technologies, Unity: Game Development Tool, 2011, <http://unity3d.com/unity/>
- [35] Ward J., What is a game engine?, 2008, http://www.gamecareerguide.com/features/529/what_is_a_game_.php