Universidad Zaragoza
1542

Proyecto Fin de Carrera

Desarrollo de una herramienta para el diseño y ejecución de actividades enfocadas a ancianos con el tabletop NIKVision

Autora

Clara Bonillo Fernández

Directores

Dra. Eva Cerezo Bagdasari
Dr. Javier Marco Rubio

Departamento de Informática e Ingeniería de Sistemas (DIIS)
Escuela de Ingeniería y Arquitectura

Septiembre 2014

Desarrollo de una herramienta para el diseño y ejecución de actividades enfocadas a ancianos con el tabletop NIKVision
RESUMEN

En este proyecto se ha implementado una herramienta que permite diseñar y ejecutar actividades enfocadas a personas mayores para el tabletop NIKVision. El objetivo principal de este proyecto es que cualquier persona que no tenga conocimientos previos de programación y que desee desarrollar una actividad de este tipo sea capaz de hacerlo utilizando la herramienta sin excesivas dificultades.

El objetivo a largo plazo será conseguir implantar la herramienta en ámbitos sanitarios tales como residencias de ancianos, de modo que éstos puedan aprovechar las posibilidades que ofrecen los tabletops, proporcionarles una aproximación intuitiva a las tecnologías y permitirles mejorar sus capacidades cognitivas.

En cuanto a las tareas realizadas en este proyecto, estas han sido las siguientes: primero se realizó un estado del arte sobre tabletops ya existentes que utilizan la interacción tangible, sobre las áreas cognitivas que más sufren deterioro con la edad y sobre las actividades que se suelen desarrollar para mejorar la capacidad cognitiva de los ancianos.

Tras la realización del estado del arte, se procedió a seleccionar un conjunto de actividades que permitieran aprovechar al máximo el uso de objetos sobre tabletop y se hizo una propuesta de actividades a realizar. Tomando como base dichas actividades, se realizó el análisis de la herramienta extrayendo los requisitos que esta habría de cumplir.

Tras el análisis, se realizó el diseño e implementación de la herramienta, la cual permite que el diseño de actividades se realice sin necesidad de programar, sino que simplemente hay que escribir la información relativa a la actividad en un fichero que posteriormente será tratado por la herramienta. En paralelo a la implementación de la herramienta se fueron desarrollando las actividades seleccionadas, desarrollándose un total de 7 actividades distintas.

La herramienta también permite la ejecución de las actividades, ofreciendo dos opciones: la primera de ellas es ejecutar una actividad concreta indicando la actividad a ejecutar en un fichero de arranque, pudiendo elegir el usuario resetear la actividad cuando desee pulsando una tecla o situando un objeto concreto sobre el tabletop, pausar la actividad situando otro objeto concreto sobre el tabletop y pudiendo además modificar en ejecución el fichero de arranque para cambiar la actividad ejecutada.

La otra opción es mostrar en el propio tabletop un menú en el cuál se muestran todas las actividades desarrolladas, de modo que el usuario puede seleccionar la actividad que desee ejecutar en ese momento de forma táctil sobre el tabletop. Una vez dentro de la actividad, el usuario también tiene las opciones de vuelta al menú y de pausa mencionadas antes además de poder modificar en tiempo de ejecución las actividades.

La herramienta también guarda información sobre las actividades realizadas en un fichero de logs en el que se guarda la fecha y hora de realización de la actividad, el tiempo empleado en su resolución, las acciones correctas e incorrectas realizadas y si la actividad se ha completado hasta el final o no.

Por último, se realizó una evaluación de usabilidad de la herramienta cuando se tuvo un primer prototipo de la misma obteniendo buenos resultados y nuevas ideas que podrán utilizarse para trabajos futuros.

Agradecimientos

En primer lugar quiero dar las gracias a Eva y a Javier por la labor de dirección y supervisión realizada: gracias a Eva por las veces que ha tenido que corregir las múltiples versiones por las que ha ido pasando esta memoria y gracias a Javier por toda la ayuda proporcionada durante todo el desarrollo del proyecto, sin su ayuda este proyecto no sería el que es. Gracias también a Sandra por permitirme hacer un proyecto más completo con la colaboración de los estudiantes de Diseño Centrado en el Usuario.

En segundo lugar quiero dar las gracias a mi novio Pablo, por su apoyo constante durante todos los meses que ha durado este proyecto y sobre todo por su paciencia en los últimos meses del mismo, cuando los nervios eran cada vez más frecuentes.

Finalmente dar las gracias a mi hermana Claudia y a mis padres por haber estado ahí siempre y haberme tenido que aguantar cuando alguna cosa no iba todo lo bien que debiera ir durante el desarrollo del proyecto.

Gracias a todos.

Clara Bonillo Fernández

Índice

1. Introducción y objetivos

En los siguientes apartados se hará una introducción para ponerse en el contexto de realización de este Proyecto Fin de Carrera y se explicarán los objetivos del mismo.

1.1. Introducción: tecnología y tercera edad

El envejecimiento de la población a nivel mundial en estos últimos años ha supuesto una revolución demográfica: la OMS (Organización Mundial de la Salud) estima que el número de personas mayores de 60 años está alrededor de los 600 millones, y se prevé que esta cifra irá creciendo en años posteriores.

Aunque este hecho puede considerarse un éxito de las políticas de salud pública y del desarrollo socio-económico, también supone un esfuerzo por parte de la sociedad, ya que el declive de las capacidades tanto físicas como cognitivas que se produce en las personas mayores ha de ser considerado por el resto de la sociedad si queremos que estas personas se sientan integradas. Además, también hay que tener en cuenta que los ancianos han de adaptarse a la nueva realidad que les rodea, que consiste en un mundo cada vez más tecnificado.

Una manera de ayudar a esta adaptación es el desarrollo de actividades que motiven a las personas mayores a usar las nuevas tecnologías y que además permitan la detección de problemas o la mejora de capacidades. Sin embargo, el uso de dispositivos tecnológicos tales como el ratón o el teclado puede suponer una considerable dificultad a las personas mayores que empiecen a usar estas tecnologías y pueden causar el efecto contrario de que estas personas mayores sean reacias a usar las nuevas tecnologías en lugar de estar motivadas a usarlas.

Por esta razón, se está explorando el uso de nuevos dispositivos que vayan más allá de los periféricos estándar y de los estilos de interacción tradicionales, acercándose cada vez más al uso de mesas interactivas o, en inglés, tabletops.

A grandes rasgos, un tabletop es un dispositivo con un aspecto más o menos cercano al de una mesa convencional, cuya superficie está aumentada mediante la proyección de imagen y sonido procedente de una aplicación informática y en el que la interacción con dicha aplicación se lleva a cabo mediante movimientos de los dedos en contacto con la superficie de la mesa (multitáctil), permitiendo una interacción más intuitiva con la tecnología y con otros usuarios.

Esta forma de interacción presenta muchas ventajas de cara a los ancianos, ya que la amplia superficie del tabletop supone un espacio amplio para sus capacidades visuales y motrices, la estimulación audiovisual les resulta motivante, les ofrece un mayor rango de actividades que pueden abarcar uno o más aspectos de estimulación cognitiva y les permite trabajar en grupo, activando con ello su actividad e integración social.

Debido a todo ello, la presencia de estas mesas ha ido aumentado tanto a nivel de investigación como comercial ([PIX12], [SMA14] y [TOU13] son algunos ejemplos).

Sin embargo, también presentan una importante desventaja, y es que la complejidad y precisión de muchos gestos táctiles puede ser dificultosa y frustrante en usuarios con temblores, artritis, o reducción de motricidad final. Por ello, otros modelos de Interacción Natural, como la Interacción Tangible, pueden ofrecer una alternativa más adaptada a las características de los usuarios mayores [KSG06].

La interacción tangible plantea que la interacción entre el usuario y la aplicación informática sea a través de objetos físicos de uso cotidiano. Un entorno especialmente apropiado para la aplicación de Interacción Tangible son los dispositivos tabletop, ya que la superficie de una mesa es en sí mismo un entorno natural para la manipulación de objetos. Técnicamente, un dispositivo tabletop Tangible es capaz de identificar distintos objetos colocados en su superficie, así como seguir las distintas manipulaciones que los usuarios realizan con ellos, mostrando información proyectada sobre la misma superficie donde se manipulan los objetos.

El Grupo Giga Affective Lab de la Universidad de Zaragoza [GIG14] cuenta con NIKVision [MCB08], un tabletop tangible en el que la interacción se lleva acabo mediante objetos colocados sobre la superficie de una mesa [NIK14] (figura 1). A cada uno de esos objetos se le asigna un identificador específico (fiducial) para que pueda ser detectado por la aplicación informática que trabaja con la mesa (figura 2).

Figura 1: Arriba a la izquierda: siluetas mostradas en la mesa. Abajo a la izquierda: Materiales didácticos manipulables. Derecha: Niño emparejando los materiales con su silueta.

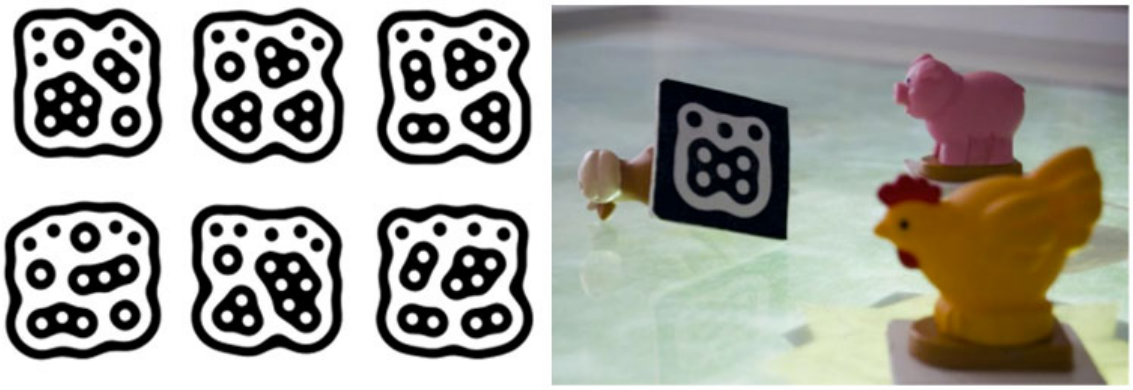


Figura 2: A la izquierda: ejemplos de fiduciales que se colocan en los objetos. A la derecha: juguetes de animales con los fiduciales ya colocados.

Gracias a esa mezcla de uso de tecnología digital e interacción natural, se consigue una aproximación más intuitiva a la tecnología, facilitando así que las personas mayores puedan beneficiarse de todo lo que ésta puede aportarles en cuanto a mejora de sus capacidades y de su calidad de vida.

Una vez puestos en contexto, en el siguiente apartado se explicarán los objetivos de este Proyecto Fin de Carrera.

1.2. Objetivos del proyecto

El principal objetivo de este Proyecto Fin de Carrera ha sido desarrollar una herramienta que permita diseñar y ejecutar actividades para personas mayores que aprovechen las ventajas que ofrecen los tabletops, utilizando para ello NIKVision [MCB08].

Lo que se ha intentado conseguir es que cualquier persona que necesite crear una actividad para tabletop pueda hacerlo de una manera sencilla con la herramienta realizada, sin necesidad de que esa persona necesite tener conocimientos de programación.

Como objetivo secundario, se han desarrollado también un conjunto de actividades como demostración de la validez de la herramienta junto con una evaluación de la misma.

En cuanto a la memoria, ésta consta de las siguientes secciones:

- Análisis: en esta sección primero se realizará un estado del arte en el que se estudiarán otro tipo de tabletop para ver cómo aprovechan la interacción tangible; segundo, se estudiarán las áreas cognitivas que más suelen trabajarse en actividades enfocadas a ancianos para proponer una selección que aproveche al máximo las ventajas de trabajar sobre un tabletop; tercero, se realizará dicha selección de actividades; cuarto, se propondrán actividades representativas para cada tipo de la selección anterior; y quinto, se extraerán los requisitos de las mismas.
- Diseño e implementación: en esta sección se realizará el diseño (a través de un diagrama de clases) de la herramienta en base a los requisitos anteriores extraídos de las actividades, se explicarán las clases más significativas de la herramienta y se explicará la forma de implementar las actividades.
- Evaluación de la herramienta: en esta sección se explicará el procedimiento que se siguió para su evaluación y los resultados que se obtuvieron.
- Desarrollo de actividades: en esta sección se explicará en detalle las actividades propuestas desarrolladas.
- Conclusiones y trabajo futuro: en esta sección se realizarán unas conclusiones relativas al proyecto realizado y se mencionaran algunas líneas de trabajo basadas en este proyecto que podrían seguirse en un futuro.
- Anexo I: en este anexo se profundizará en los detalles de implementación de la herramienta.
- Anexo II: en este anexo se profundizará en los detalles de la evaluación de la herramienta.
- Anexo III: en este anexo se mostrará el desarrollo temporal del proyecto.

2. Análisis

Antes de realizar el análisis, primero se va a realizar un pequeño estudio de las actividades que suelen utilizarse para mejorar las capacidades cognitivas y físicas de los ancianos, después se hará una clasificación en función a ese estudio, se escogerán algunas actividades de cada tipo y por último se extraerán los requisitos necesarios.

2.1. Estado del arte: tabletops y tercera edad

A continuación se van a estudiar ejemplos de tecnologías usadas por personas mayores, con especial énfasis en aquellas que hacen uso de tabletops, para ver cuáles de sus características podemos reutilizar para este proyecto y qué se podría aportar como nuevo.

En [JPM04] se describe una aplicación basada en el solitario para diferenciar ancianos con buena capacidad cognitiva y ancianos con algún desorden cognitivo. Aquí se tiene un ejemplo de uso de una aplicación que aparte de motivar a las personas mayores a usar las tecnologías a través de un juego, permite deducir cuál es su condición cognitiva mediante el análisis de los resultados que se obtuvieron después de que un grupo de personas mayores jugara a dicho juego durante tres semanas.

En [KKP13] se mencionan tres sistemas (CoCoMo, CoCoTa y E-CoRe) para mejorar la capacidad cognitiva:

- CoCoMo (*Computerized Cognitive Assessment in Middle Aged and Older Adults*): herramienta que permite llevar acabo 13 tareas relacionadas con actividades que suelen hacerse a lo largo de la vida diaria (contar dinero, recordar palabras o localizaciones...) repartidas entre 7 áreas cognitivas: velocidad motora, atención, lenguaje, cálculo, orientación espacio-temporal, memoria y funciones ejecutivas (figura 3).
- CoCoTa (*Computerized Cognitive Training Apparatus*): programa de entrenamiento cognitivo para mejorar y mantener las funciones cognitivas en buenas condiciones manteniendo un historial de los entrenamientos realizados (figura 3).
- E-CoRe (*Embodied COgnitive REhabilitation*): tabletop que utiliza una aplicación llamada "**Making Cookies**", en la que se utilizan distintos objetos para simular hacer galletas.

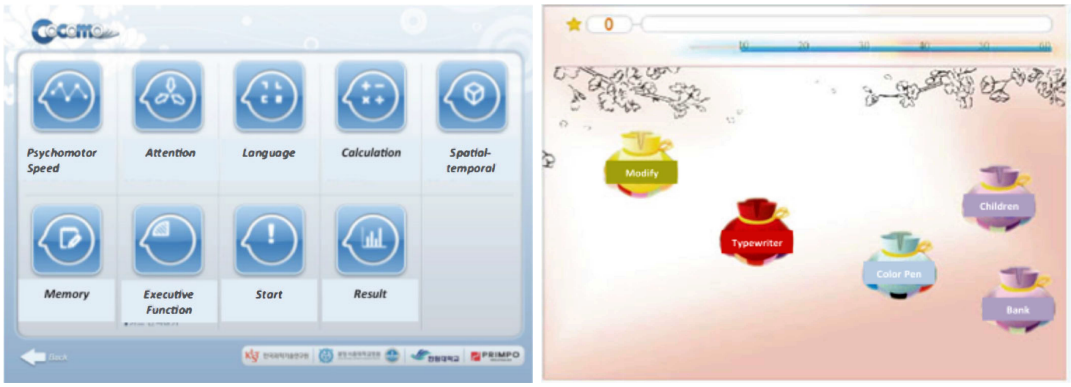


Figura 3: Captura de las aplicaciones CoCoMo (izquierda) y CoCoTa (derecha).

En [GMS09] se ha desarrollado un tabletop con una aplicación similar al juego de encontrar parejas interactuando con unos lápices especiales en lugar de con el dedo y diversos minijuegos, en los que se busca mejorar funciones cognitivas como la memoria, el razonamiento, la atención selectiva y dividida, y la clasificación (figura 4).

Figura 4: A la izquierda: prototipo de tabletop de Eldergames. A la derecha: objetos/lápices que utiliza.

Por último, en [AAG09] se ha desarrollado un tabletop (AIR Touch) junto con un conjunto de actividades para la rehabilitación física, tales como un juego de estallar globos que van apareciendo por pantalla, un juego de seguir el ritmo de una canción golpeando unos tambores en las esquinas de la pantalla y diversas actividades de dibujar utilizando el dedo como pincel (figura 5).

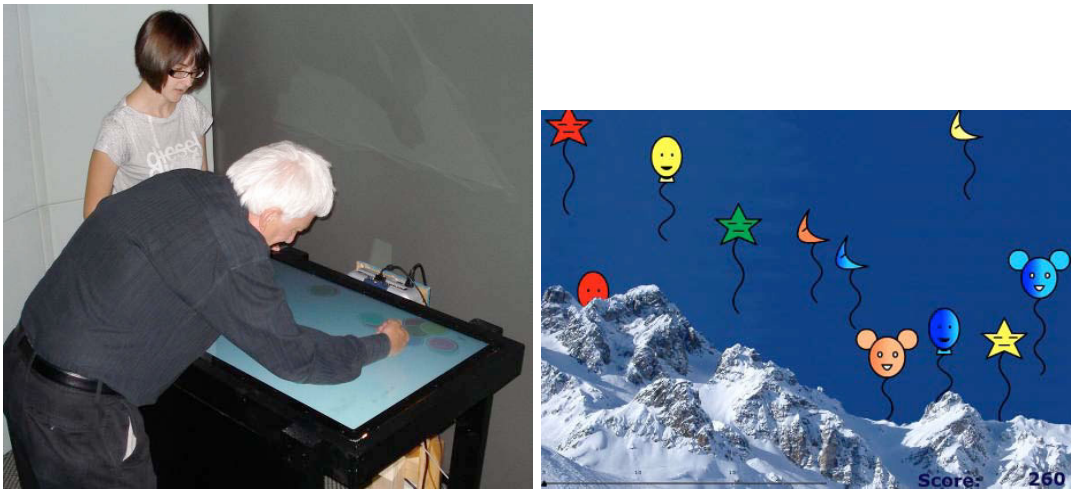


Figura 5: A la izquierda: Tabletop AIR touch. A la derecha: captura de la actividad “Pop those balloons”.

Del estudio de los ejemplos anteriores se puede concluir que aunque tanto el tabletop como las actividades desarrolladas para éste acercan a los ancianos a las nuevas tecnologías y les permiten seguir trabajando sus capacidades cognitivas para evitar (o por lo menos disminuir) su deterioro, no se aprovecha todo lo que se podría las ventajas de tener un tabletop, puesto que casi todas las actividades mencionadas antes se realizan usando la mano o algún dispositivo que podría ser fácilmente sustituido por la mano, como es el caso del lápiz de [GMS09].

Una manera de aprovechar más las ventajas del tabletop se ve en [LTK07], donde se plantea trabajar con cubos con distintos patrones dibujados sobre ellos, siendo el objetivo de la tarea alinearlos para que coincidan con el objeto que se muestra en la mesa (figura 6). Otro ejemplo de este mejor aprovechamiento se ha comentado ya en [KKP13] con el E-CoRe (figura 7), que utilizaba una simulación de hacer galletas.

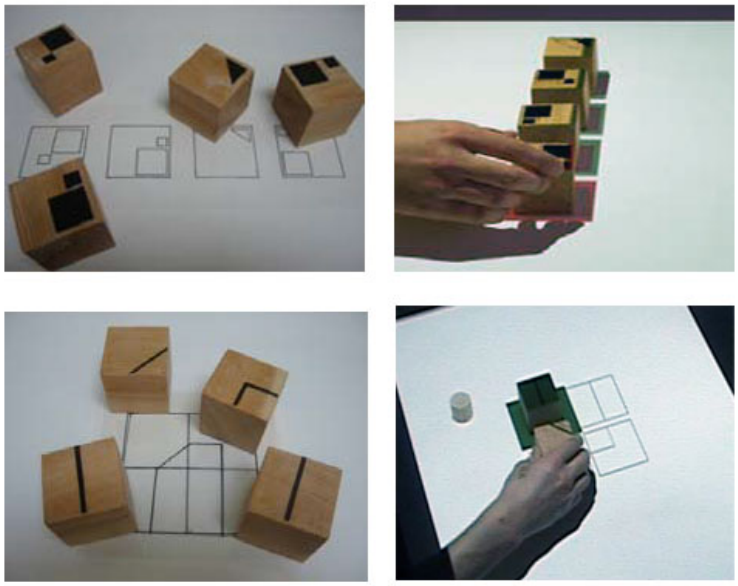


Figura 6: Prototipo de actividades mediante el uso de cubos

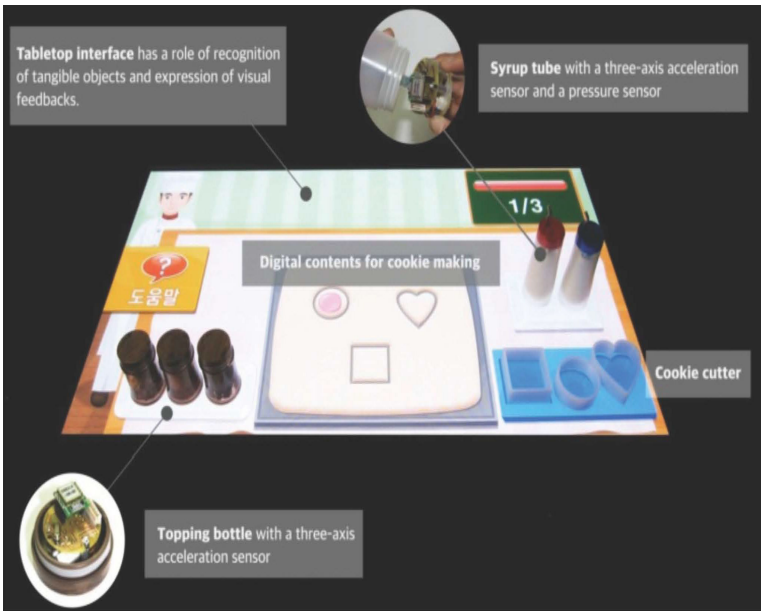


Figura 7: Actividad "Making Cookies" del tabletop E-CoRe

La ventaja de NIKVision es la variedad de objetos que puede utilizar, lo cual es posible gracias a ToyVision [MCB12], una herramienta que permite un prototipado fácil de juegos de tabletop, de modo que prácticamente cualquier juguete puede utilizarse sin necesidad de ser objetos concretos como los 4 elementos de cocina de [KKP13], los lápices de [GMS09] y los cubos de [LTK07].

Una vez realizado el estudio de algunos de los tabletop usados durante los últimos años con sus correspondientes actividades, en el siguiente apartado realizaremos el análisis de las actividades a desarrollar junto con los requisitos que tendrá que tener nuestra herramienta para permitir su diseño y ejecución.

2.2. Áreas cognitivas y actividades

Como se ha comentado, el envejecimiento trae consigo diversos problemas tanto físicos como cognitivos: además del deterioro de la vista y el oído y de la aparición de complicaciones motoras, las personas mayores sufren principalmente pérdidas de memoria, déficit cognitivo y de atención y dificultades lingüísticas; por lo tanto, el objetivo es centrarse en buscar actividades que ayuden a potenciar las áreas cognitivas tales como la memoria, la atención, la capacidad de cálculo, el lenguaje, la orientación espacio-temporal y el razonamiento para ayudar a prevenir, en la medida de lo posible, su deterioro.

Nuestro objetivo es encontrar una selección de actividades que cubra el máximo número de áreas cognitivas previamente mencionadas y que además aproveche al máximo el uso del tabletop, lo cual se traduce en nuestro caso como que las actividades elegidas han de necesitar el uso de objetos para su completitud.

Por lo tanto, antes de realizar la selección se explicará en qué consiste cada una de las áreas cognitivas mencionadas anteriormente junto con ejemplos de actividades que se suelen desarrollar para potenciar dichas áreas:

- Memoria: es una función del cerebro y, a la vez, un fenómeno de la mente que permite al organismo codificar, almacenar y recuperar la información del pasado. Como se ha dicho antes, las pérdidas de memoria aparecen con cada vez más frecuencia conforme una persona se hace mayor, así que este área cognitiva siempre está presente en las actividades enfocadas a ancianos.

  Algunos ejemplos de actividades de memoria se pueden ver en [SOC12], un proyecto europeo que introduce una nueva aproximación para el entrenamiento cognitivo para personas mayores:

  - **Find the Pairs**: se muestran un conjunto de imágenes situadas en forma de matriz 3x6 y pasados unos segundos se ocultan. El objetivo es emparejar las imágenes que eran iguales.
  - **Do you remember your order**: se muestra un menú por pantalla con diversos primeros platos, segundos platos y postres junto con una elección de menú. Pasado un tiempo se oculta dicha elección de menú y el objetivo es recomponer el menú que se había pedido.
  - **Remember the Design**: se muestra un diseño compuesto por líneas. Pasados unos segundos se oculta y el objetivo es reproducir el diseño.

  Se puede ver que la mecánica de las actividades de memoria suele ser siempre la misma: al usuario se le muestra durante un tiempo determinado una información y pasado ese tiempo, esa información se le oculta y el usuario tiene que recordarla. Además, todas las actividades de memoria conllevan algo de atención.

- Atención: dentro de la atención, podemos distinguir a su vez entre tres subtipos [Dra07]:

  - Selectiva: aquella que se basa en reaccionar ante estímulos concretos de entre todos los que se presentan.
  - Dividida: aquella en la que hay que centrarse en varios estímulos a la vez, perteneciendo generalmente estos estímulos a sentidos diferentes (por ejemplo, vista y oído).
  - Sostenida: aquella que requiere concentración continua.

  Algunos ejemplos de actividades de atención se pueden ver en [SOC12], donde aparecen actividades como:

  - **Guess who**: se muestran al usuario un conjunto de imágenes de personas y se le van dando pistas para que adivine la persona de la que se está hablando. El usuario ha de ir descartando personas hasta quedarse con la persona correcta.

- **Lost in the city**: al usuario se le muestran 5 monigotes, 4 de los cuáles tienen el brazo enfocado a una dirección (arriba-derecha, derecha, abajo-derecha, abajo, abajo-izquierda, izquierda, arriba-izquieda o arriba) y el monigote que queda lo tiene enfocado a otra dirección. El usuario ha de modificar la posición del brazo del monigote que es distinto a los demás para que todos tengan la misma posición.

Otro ejemplo de trabajo con la atención puede verse en [AAG09] con su actividad "**Pop those Ballons**", en la que el usuario ha de estar atento para ver qué globos tiene que explotar.

- Cálculo: este área cognitiva se refiere a la capacidad de una persona de realizar cálculos matemáticos sin instrumentos adicionales. Ejemplos de actividades de este tipo se pueden encontrar en [EJE14], donde se mencionan ejercicios de cálculo tales como:

  - **Parchís**: se usa el cálculo cuando hay que ir contando los puntos por comerte una ficha, por llegar a una casilla final...
  - **Dominó**: al ser el objetivo del juego alcanzar una puntuación prefijada, el jugador tiene que estar constantemente calculando los puntos que lleva.
  - **Emparejamiento de fichas**: se dispone de un conjunto de fichas con números y operaciones matemáticas y hay que emparejar las fichas correctos; por ejemplo, si se tiene una ficha con un "75:3" hay que emparejarla con una ficha con un "25".

Otro ejemplo de cálculo se mencionaba en [KKP13], donde se tenía una actividad para contar monedas.

- Lenguaje: capacidad de los seres humanos para comunicarse por medio de signos. Actividades que potencian este área se pueden ver en [SOC12], donde hay actividades de emparejar sinónimos y antónimos. En [TAL14] se pueden ver diversos ejercicios de lenguaje tales como ordenar frases desordenadas o elegir la palabra que corresponde a la definición de entre varias opciones propuestas.

- Orientación espacio-temporal: es la toma de conciencia de los movimientos en el espacio y el tiempo de forma coordinada. Se relaciona con el conocimiento que tiene cada persona del tiempo en el que está (fecha y hora) y del lugar en el que está. En [EST14] se pueden ver ejemplos de ejercicios de este tipo, en los que se distingue entre:

  - Orientación temporal: donde se trabajan aspectos más recientes como el día, mes, año y estación, y aspectos más inmediatos, con preguntas del tipo: ¿qué hora es?, ¿en qué momento del día estamos?...
  - Orientación espacial: donde se trabajan aspectos como el lugar donde se encuentra y aspectos más recientes como ciudad, provincia, país... utilizándose preguntas del tipo: ¿en qué calle estamos?, ¿en qué planta estamos?...

- Razonamiento: facultad que permite resolver problemas, extraer conclusiones y aprender de manera consciente de los hechos, estableciendo conexiones causales y lógicas necesarias entre ellos. En [SOC12] se pueden ver ejemplos de actividades como:

  - **Analogies**: al usuario se le muestran dos imágenes relacionadas de alguna manera (por ejemplo, lluvia y un paraguas) y otra imagen (un sol) a la que tiene que asociar otra imagen guardando relación con las dos previamente mostradas. En este caso, la respuesta correcta podría sería elegir una imagen de una sombrilla.
  - **Differences**: se muestran dos imágenes y el usuario ha de elegir la característica que más las diferencia. Por ejemplo, se le muestra una rosa en una maceta y un girasol, y el usuario tiene que elegir de entre las opciones: "tiene una maceta", "son flores" y "son amarillas". Aquí la principal diferencia sería que el girasol es amarillo y la rosa no.
  - **Similarities**: lo mismo que "Differences" pero a la inversa, se trata de elegir la característica que más las asemeja.

Una vez explicadas las principales áreas cognitivas en las que enfocarse y ejemplos de actividades de las mismas, en el próximo apartado realizaremos la selección de aquellas que pueden beneficiarse de su implementación en un tabletop tangible como NikVision.

2.3. Selección de actividades

Gracias a la información analizada en el apartado anterior sobre los tipos de juegos que suelen utilizarse para ejercitar las capacidades cognitivas, se ha podido llegar a la conclusión de que actividades enfocadas a funciones cognitivas como el lenguaje, el cálculo y la orientación espacio-temporal no entrarán dentro de nuestra selección.

Los motivos de esta conclusión son los siguientes:

- Las actividades de lenguaje mencionadas tales como emparejar sinónimos, antónimos, ordenar frases y elegir la definición correcta pueden completarse en cualquier dispositivo táctil únicamente utilizando el dedo para seleccionar la respuesta correcta, y por tanto el uso de objetos para la resolución de actividades no aporta mucho.

  Se podría adaptar algunas de las actividades tales como la de elegir definiciones de modo que en vez de representar las opciones a elegir de forma escrita, estuvieran representadas como objetos de modo que el usuario tuviera que elegir de entre los objetos cuál se corresponde.

  Sin embargo, dado que este tipo de actividades sólo están pensadas para potenciar el lenguaje y no la atención (de usar objetos, el usuario tendría que usar la atención selectiva para ver qué objeto se corresponde con la definición), sería mejor no distraer al usuario del objetivo principal, que es simplemente seleccionar la palabra correcta.
- Las actividades de cálculo mencionadas, excluyendo la de emparejar fichas, se refieren a juegos de mesa tales como el parchís o el dominó; por lo tanto, el uso de estos juegos ya cubriría las necesidades de la actividad y no haría falta volver a implementarlas para tabletop, lo cual sería bastante costoso cuando el hecho de jugar al dominó o al parchís sobre un tabletop en lugar de sobre una mesa normal y corriente no aporta nada nuevo.
- Las actividades de orientación espacio-temporal se basan en hacer preguntas concretas y por tanto no necesitan del uso de objetos para su resolución.

Por lo tanto, la selección ha sido la siguiente:

- Actividades de memoria: puesto que además de la memoria, todas requieren algún tipo de atención (ya sea selectiva o dividida) y por tanto el tener que trabajar con objetos requiere algo de atención adicional.
- Actividades de atención: divididas a su vez en los tres subtipos de atención que hay [Dra07] (selectiva, dividida y sostenida).
- Actividades de razonamiento: puesto que el razonamiento suele ir acompañado de la atención sostenida y por tanto el uso de objetos puede potenciar su desarrollo.

Esto no quiere decir que actividades correspondientes al lenguaje, al cálculo y a la orientación espacio-temporal no se puedan diseñar con la herramienta en caso necesario, sino que en comparación con el resto de actividades el uso de objetos sobre tabletop no les aporta demasiado, por tanto se ha decidido no incluirlas.

En el siguiente apartado, se explicarán las actividades que se han desarrollado para cada tipo de la clasificación.

2.4. Propuesta de actividades a realizar

A continuación se van a comentar las actividades que se proponen como ejemplo para cada tipo de la clasificación establecida en el apartado anterior.

2.4.1. Actividades de memoria

- **Lista de la compra**: en pantalla se muestra una imagen con una lista de la compra con alimentos durante 5 segundos. Pasado ese tiempo, se quita la imagen de lista poniéndose una imagen con una bolsa de la compra y el usuario ha de seleccionar los alimentos correctos y situarlos en la bolsa.

  Los objetos que se utilizan en esta actividad son 12 juguetes de comidas y bebidas: agua, cebolla, guisantes, huevo, leche, manzana, naranja, pimiento, tomate, zanahoria, zumo de naranja y zumo de uva.

  La actividad puede estar compuesta de diferentes tareas que combine los alimentos de manera distinta, de modo que primero puede mostrarse una lista con los alimentos: agua, huevo, manzana y tomate (completándose la actividad cuando esos 4 alimentos estén sobre la bolsa) y después que salga una lista con una anotación que diga: "Mete en la bolsa dos bebidas" (completándose la tarea cuando estén sobre la bolsa las bebidas leche-zumo de uva, leche-agua, zumo de naranja-agua...).

- **Viajes**: al usuario se le muestra un mapa de Europa y se reproduce una grabación con un itinerario a seguir con distintos medios de transporte, por ejemplo: "Iré primero a Grecia en barco, después a Rumanía en tren y por último a Alemania en avión", de modo que el usuario ha de seleccionar el medio de transporte adecuado y situarlo en el país correspondiente. En el ejemplo, la tarea se completaría cuando el juguete de barco estuviera en Grecia, el de tren en Rumanía y el de avión en Alemania.

  Los objetos que se utilizan en esta actividad son 3 juguetes de medios de transporte: tren, avión y barco.

2.4.2. Actividades de atención

- **Tangram** (atención sostenida): este juego [TAM14] tiene dos modos: el fácil, en el que se ve en qué posición está cada una las piezas que conforman la figura, y el difícil, en el que solo se muestra el contorno de la figura. En la modalidad difícil también se trabaja el razonamiento, al tener que deducir dónde va cada pieza.

  Los objetos con los que trabajamos en esta actividad son las 7 piezas del Tangram.

- **Marca los símbolos** (atención selectiva): se presenta un mapa con símbolos de gasolineras, restaurantes, hoteles, farmacias...y se pide al usuario que marque todos los que sean de un tipo concreto.

  Los objetos que se utilizan en esta actividad son fichas que el usuario tendrá que colocar en los símbolos correspondientes.

- **¿Cuántos hay?** (atención dividida): Al usuario se le muestra una imagen con números entre el 0 y el 9. Nada más empezar la actividad, suena una grabación que le dice al usuario qué número ha de buscar; por ejemplo, la grabación dirá algo del estilo: "Marca todos los treses que aparecen en pantalla".

  El usuario tendrá que ir situando fichas sobre todos los números que encuentre de ese tipo mientras que a la vez se estará reproduciendo una grabación con una secuencia de golpes: por ejemplo, sonarán 4 golpes, habrá una pausa, sonarán 8 golpes, habrá una pausa... de modo que el usuario, a la vez que marca los números, durante la pausa tendrá que decir cuántos golpes han sonado.

  Los objetos que se utilizan en esta actividad son fichas que el usuario tendrá que colocar en los números correspondientes.

2.4.3. Actividades de razonamiento

- **Completa la secuencia**: usando fichas de dominó, se muestra una secuencia en la que faltan algunas piezas y el usuario ha de seleccionar la pieza que falta para completar la secuencia.

  Los objetos que se utilizan en esta actividad son las fichas de dominó.

- **Analogías**: se dispone de un conjunto de fichas con dos imágenes cada una. A cada imagen le corresponde una pareja, que estará en otra ficha diferente. El objetivo de la actividad es que todas las fichas formen una cadena de modo que todos los extremos de las fichas estén con su correspondiente pareja.

  Para este juego podría ser interesante que en lugar de haber áreas de interacción fijas, éstas se "creen" asociadas a las fichas, de modo que el usuario no tenga que preocuparse de dónde poner la ficha sino sólo de hacerla coincidir con la correspondiente pareja.

  Los objetos que se utilizan en esta actividad son las diferentes fichas.

2.5. Requisitos a cumplir

Si se analizan las actividades propuestas en el apartado anterior, se puede observar que presentan una característica común: para completar la actividad hay que colocar determinados objetos en determinadas áreas interactivas. En particular todas comparten:

- Una imagen de fondo de la actividad, que podrá cambiar o no.
- Areas interactivas.
- Objetos correctos e incorrectos asociados a dichas áreas interactivas.

Por tanto, se pueden deducir los requisitos funcionales que una herramienta destinada a su desarrollo habrá de cumplir, requisitos que se muestran en la Tabla 1.

Requisitos	Descripción	Actividad
R1	En la superficie de la mesa se han de tener áreas interactivas en las que poder poner uno o más objetos	Todas
R2	En algunas actividades, se ha de tener en cuenta la orientación de los objetos colocados sobre el área interactiva	*Atención*
R3	Las áreas tienen una lista de objetos correctos y una lista de objetos incorrectos	Todas
R4	La imagen que se muestra en pantalla puede cambiar pasado un determinado tiempo que será configurable	*Memoria*
R5	Las áreas interactivas pueden estar fijas en una posición en la pantalla o ir asociadas a objetos	Todas
R6	Se tiene que poder reproducir sonidos cuando se coloquen los objetos en las áreas dependiendo de si están bien o mal	Todas
R7	Se tiene que poder definir secuencias de actividades, de modo que cuando se complete una actividad, se pase a la siguiente	Todas
R8	Las actividades pueden requerir objetos concretos diferenciados o fichas del mismo tipo	Todas

Tabla 1: Requisitos funcionales

2.5.1. Requisitos funcionales adicionales

Además de los requisitos funcionales de la tabla 1 que se extraen de las necesidades de las actividades del apartado anterior, durante la realización del estado del arte se pudieron observar dos cosas:

- Se suele utilizar algún tipo de retroalimentación o feedback en las actividades. Por ejemplo, en la actividad "**Pop those balloons**" de [AAG09], cada vez que el usuario estalla un globo con éxito, éste desaparece y su puntuación aumenta, o en las actividades desarrolladas por [GMS09], se proporciona un feedback visual.

  Por lo tanto, es necesario que las actividades propuestas tengan algún tipo de feedback, ya sea visual o auditivo, de modo que el usuario pueda saber si la actividad la va resolviendo bien o no.

- Se suele utilizar algún tipo de registro de resultados (o logs) durante las actividades. Por ejemplo, en el juego del solitario de [JPM04] se guardaron los resultados de los usuarios durante las tres semanas que duró el estudio y en [GMS09], aparte del software de actividades el tabletop disponía de un módulo que permitía al usuario registrarse, de modo que cada vez que el usuario realizaba alguna actividad, esa información se guardaba para poder ser analizada después.

  Por lo tanto, podría ser útil que las actividades propuestas guardaran algún tipo de información por si posteriormente en algún trabajo futuro fuera interesante hacer un estudio sobre la evolución de un usuario que utilizara la herramienta.

  De este análisis se extraen, por tanto, dos nuevos requisitos funcionales que se presentan en la Tabla 2.

Requisitos	Descripción	Actividad
R9	Se tiene que poder mostrar imágenes cuando se coloquen los objetos en las áreas dependiendo de si están bien o mal	Todas
R10	Se tiene que poder guardar información relativa al proceso de resolución de cada actividad	Todas

Tabla 2: Requisitos funcionales adicionales

Por último, los requisitos no funcionales de la herramienta serían los siguientes:

Requisitos	Descripción
RN1	La herramienta se desarrollará con AS3. Por lo tanto se ha de tener instalado Adobe Flash CS5.5 en el ordenador para poder trabajar con la herramienta
RN2	Para el diseño de actividades es recomendable disponer de algún programa de tratamiento de gráficos como Photoshop o Gimp

Tabla 3: Requisitos no funcionales

La razón de elegir AS3 para programar fue debido a que se iban a necesitar algunas librerías y funciones ya implementadas que estaban programadas en AS3.

3. Diseño e implementación de la herramienta

En este apartado se va hablar tanto del diseño como de la implementación de la herramienta que permita el desarrollo de las actividades descritas en el apartado anterior.

3.1. Diagrama de clases

Una vez seleccionadas las actividades a realizar, queda hacer el diseño de la herramienta que permita a su vez diseñarlas y ejecutarlas. En la figura 8 se puede ver el diagrama de clases de la herramienta:

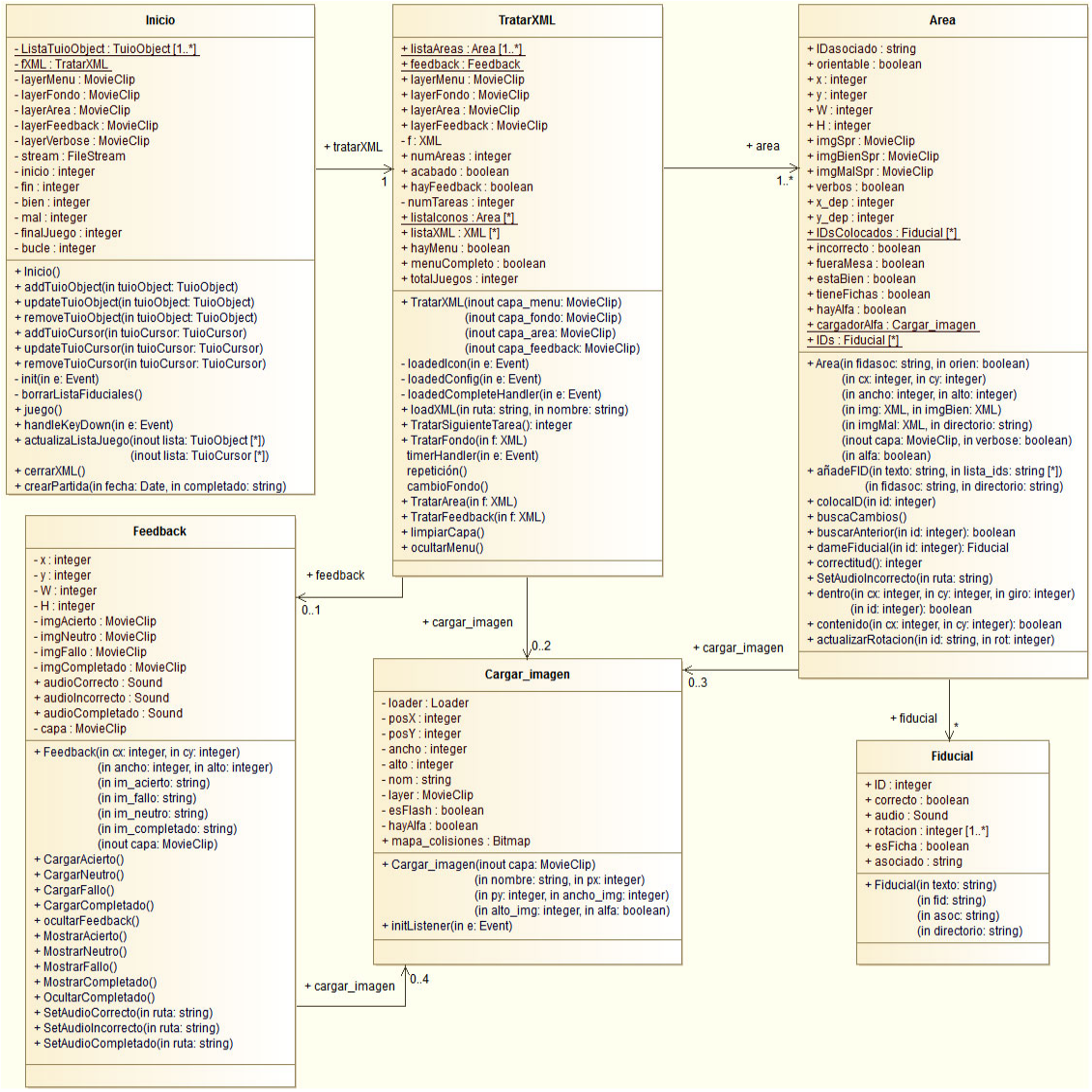


Figura 8: Diagrama de clases de la herramienta

En los siguientes apartados van a explicarse las clases de forma resumida con sus atributos y funciones más relevantes junto con algunos detalles de implementación.

Como se ha comentado anteriormente, todas las actividades a diseñar necesitan una imagen de fondo, áreas interactivas, objetos asociados a dichas áreas y feedback; por lo tanto, como mínimo la herramienta tendrá que implementar dichas clases.

Antes de seguir hay que decir que aparte de las clases mostradas en el diagrama, se utilizó una librería (tuio) que no va a ser explicada en detalle puesto que fue únicamente reutilizada de proyectos anteriores y no se realizó sobre ella ninguna modificación; únicamente decir que esta librería es la que permite establecer la conexión de la herramienta con el tabletop NIKVision mediante un socket UDP y es la que se encarga de gestionar el reconocimiento de los objetos que se ponen sobre la mesa.

Aparte de *tuio*, también se utilizó otra librería que se encargaba del dibujo de las áreas que tampoco se explicará puesto que también fue reutilizada.

3.2. Clase *Cargar_imagen*

Como su nombre indica, esta clase es la encargada de cargar una imagen en una capa (objeto MovieClip) en una posición concreta con unas dimensiones determinadas. Sus atributos son:

- **loader**: contiene el archivo de imagen (png o jpg).
- **posX**: posición X de la imagen en la actividad.
- **posY**: posición Y de la imagen en la actividad.
- **ancho**: ancho de la imagen.
- **alto**: alto de la imagen.
- **nom**: path absoluto de la imagen.
- **layer**: capa sobre la que se dibujará la imagen.
- **esFlash**: este atributo indica si la imagen pasada como parámetro es jpg/png o flash (.swf). Esto puede ser útil si cuando se completa un área se quiere que en lugar de una imagen, aparezca una animación indicando la completitud del área.
- **hayAlfa**: este atributo indica si hay que guardarse el bitmap de la imagen (se presupone que la imagen con la que se trabaja cuando *hayAlfa* es *true* es png).
- **mapa_colisiones**: variable en la que se guarda el bitmap de la imagen si el atributo *hayAlfa* es *true*; en caso contrario, vale *null*.

Las funciones de la clase *Cargar_imagen* son las siguientes:

- **Cargar_imagen**: constructor de la clase.
- **initListener**: función encargada de cargar la imagen <nombre> en la capa <capa> en la posición (<px>,<py>) de la pantalla y con las dimensiones <alto_img> y <ancho_img>.

Ejemplos de código de dichas funciones se pueden ver en el anexo 1.

3.3. Clase *Area*

La clase *Area* es la encargada del tratamiento de áreas. Sus atributos son los siguientes:

- **x**: posición X del área en la actividad.
- **y**: posición Y del área en la actividad.
- **W**: ancho del área.
- **H**: alto del área.
- **IDs**: lista de ids de los objetos o fichas asociados al área, tanto correctos como incorrectos (puede ser una lista vacía y que el área no tenga objetos asociados).
- **IDsColocados**: lista de ids de los objetos o fichas colocados sobre el área.

- **incorrecto**: indica si el área tiene lista de objetos incorrectos.
- **tieneFichas**: indica si hay fichas en la lista de objetos asociados.
- **verbos**: si es *true*, indica que el contorno del área será posteriormente dibujado cuando se ejecute la actividad; si es *false*, no se dibujará el contorno.
- **estaBien**: indica el estado de la última acción realizada sobre el área. Puede tener tres valores: el valor 0, que indica que no ha habido ningún cambio, el valor 1, que indica que ha habido un cambio en el área y ha sido correcto, y el valor -1, que indica que ha habido un cambio en el área y ha sido incorrecto. El uso de este atributo se verá posteriormente en la clase *Inicio*, en la que se combina con el feedback para saber cuándo cambiar de imagen neutra a correcta o incorrecta.
- **hayAlfa**: si es *true*, indica que se va a tener en cuenta el canal alfa de su imagen neutra; si es *false*, no se tendrá en cuenta el canal alfa.
- **cargadorAlfa**: objeto *Cargar_imagen* asociado a la imagen neutra del área. Se utiliza para poder acceder al atributo *mapa_colisiones* y sólo es distinto de *null* cuando *hayAlfa* vale *true*.

Además de estos atributos comunes a todas las áreas, en el análisis de requisitos que se realizó se determinó que había actividades (como la de *Analogías*) que necesitaban áreas que dependieran de los objetos. Para manejar esa funcionalidad, se necesitan además de los anteriores, los atributos:

- **IDasociado**: id del objeto al que va asociada el área.
- **orientable**: indica si el área ha de girar con respecto al objeto asociado.
- **x_dep**: posición X absoluta del área.
- **y_dep**: posición Y absoluta del área.
- **fueraMesa**: indica si el objeto asociado está fuera de la mesa.

Por último, para facilitar la realización de algunas actividades, puede ser interesante asociar imágenes a áreas, de modo que depende del estado del área aparezca una imagen u otra. Para eso, se utilizan los atributos:

- **imgSpr**: imagen que aparece asociada al área cuando es neutra.
- **imgBienSpr**: imagen que aparece asociada al área cuando está bien.
- **imgMalSpr**: imagen que aparece asociada al área cuando está mal.

En el diagrama se refleja la relación entre la clase *Area* y la clase *Cargar_imagen* indicando que un área puede tener asociadas de 0 a 3 imágenes.

Las funciones de la clase *Area* son las siguientes:

- **Area**: constructor de la clase.
- **añadeFID**: añade los fiduciales asociados al área.
- **colocaID**: mete el id de fiducial <id> en la lista <IDsColocados>.
- **buscaCambios**: busca las diferencias entre el estado anterior del área y el actual.
- **buscarAnterior**: devuelve true si <id> no estaba colocado antes
- **dameFiducial**: devuelve el objeto fiducial a partir de su <id> o null si no estaba.
- **correctitud**: devuelve el estado del área (-1 incorrecta, 0 neutra, 1 correcta, 2 algo mal).
- **SetAudioIncorrecto**: carga el audio incorrecto.

- **dentro**: comprueba si (<Cx>,<Cy>) están dentro del área con la orientación <giro> correcta.
- **contenido**: comprueba si (<Cx>,<Cy>) están dentro del área.
- **actualizarRotacion**: actualiza la rotacion de los fiduciales dependientes de <id> con respecto a la rotación <rot> de este.

En el anexo 1 se puede ver el detalle del código de algunas de sus funciones más relevantes.

3.4. Clase *Fiducial*

La clase *Fiducial* es la encargada del tratamiento de los objetos asociados áreas. Los atributos de *Fiducial* son los siguientes:

- **ID**: id del fiducial asociado al objeto.
- **correcto**: indica si el objeto se considera correcto o no en el área a la que pertenece.
- **audio**: audio asociado a la colocación del fiducial en el área a la que pertenece.
- **esFicha**: indica si es un fiducial con id único o una ficha.

Aparte de los atributos anteriores, si el fiducial pertenece a un área que además depende de un fiducial, también hay que guardarse dicho fiducial asociado en el atributo ***asociado***.

Por último, si el fiducial ha de tener una orientación u orientaciones determinadas para que se considere correcto, en el atributo ***rotacion*** se guardan dichas orientaciones.

En cuanto a las funciones de la clase *Fiducial*, ésta sólo posee en el constructor, el cuál se explicará con algo más de detalle en el anexo 1.

3.5. Clase *Feedback*

La clase *Feedback* es la encargada del tratamiento del feedback de la actividad. Los atributos de *Feedback* son los siguientes:

- **x**: posición X del feedback en la actividad.
- **y**: posición Y del feedback en la actividad.
- **W**: ancho del feedback.
- **H**: alto del feedback.
- **imgAcierto**: imagen que aparece cuando se coloca un objeto bien en un área.
- **imgNeutro**: imagen que aparece mientras no se coloca ningún objeto.
- **imgFallo**: imagen que aparece cuando se coloca un objeto mal en un área.
- **imgCompletado**: imagen que aparece cuando finaliza con éxito la tarea.
- **audioCorrecto**: sonido que se reproduce cuando se coloca un objeto bien en un área.
- **audioIncorrecto**: sonido que se reproduce cuando se coloca un objeto mal en un área.
- **audioCompletado**: sonido que se reproduce cuando finaliza con éxito la tarea.
- **capa**: capa sobre la que se dibujará la imagen.

De los atributos anteriores, los únicos obligatorios para todo objeto *Feedback* son los 4 primeros (x, y, W y H) y *capa*; los demás son todos opcionales, de modo que el objeto *Feedback* puede estar compuesto de diversas combinaciones entre imágenes y audio.

Las funciones de la clase *Feedback* son las siguientes:

- **Feedback**: constructor de la clase.
- **CargarAcierto**: carga la imagen de acierto.
- **CargarFallo**: carga la imagen de fallo.
- **CargarNeutro**: carga la imagen neutra.
- **CargarCompletado**: carga la imagen de completitud de la tarea.
- **ocultarFeedback**: oculta el feedback de la capa <capa>.
- **MostrarAcierto**: muestra la imagen de acierto cuando un objeto se coloca bien en un área.
- **MostrarFallo**: muestra la imagen de fallo cuando un objeto se coloca mal en un área.
- **MostrarNeutro**: muestra la imagen de neutro mientras no se coloque ningún objeto.
- **MostrarCompletado**: muestra la imagen de completado cuando la tarea se completa con éxito.
- **OcultarCompletado**: oculta la imagen de completitud de la tarea.
- **SetAudioCorrecto**: carga el audio correcto.
- **SetAudioIncorrecto**: carga el audio incorrecto.
- **SetAudioCompletado**: carga el audio de completitud de la tarea.

En el diagrama se refleja la relación entre la clase *Feedback* y la clase *Cargar_imagen* indicando que un feedback puede tener asociadas de 0 a 4 imágenes.

La explicación de las funciones previamente mencionadas se realizará con algo más de detalle en el anexo 1.

3.6. Clase *TratarXML*

El elegir el nombre "*TratarXML*"para esta clase se debe a lo siguiente: para representar la información de las actividades (áreas, fiduciales, feedback...) de modo que pueda ser interpretada con facilidad por la herramienta, se optó por poner la información característica de cada actividad en forma de fichero XML. La razón de hacerlo así fue porque ya se había realizado un proyecto anterior en el que también se manejaban áreas similares y la forma de representar la información era con un fichero XML, así que se reutilizó la idea.

Al principio, la clase *TratarXML* se encargaba únicamente del tratamiento de información del fichero XML de la actividad junto con la inicialización de la misma a través de una ventana de selección en la que había que navegar por las carpetas para elegir el juego a ejecutar.

Sin embargo, a la hora de hacer pruebas y de ejecutar actividades se vio que sería más cómodo que una actividad elegida pudiera ejecutarse directamente al iniciar la herramienta sin necesidad de navegar por las carpetas, y que también sería interesante aprovechar el uso del tabletop de forma que al iniciar la herramienta, apareciera un menú por pantalla con iconos de las actividades desarrolladas en ese momento de modo que de forma táctil se pudiera elegir la actividad a ejecutar.

Por lo tanto, la herramienta proporciona dos opciones de ejecución de actividades:

- mediante un fichero XML de configuración (config.xml) con único elemento (arranque) en el que se indica la actividad a ejecutar.
- mediante el menú mostrado en la figura 9, en el que se muestran las actividades disponibles con iconos que a su vez son áreas interactivas de modo que cuando se pone un objeto o un dedo encima de ellas, se ejecuta la actividad seleccionada. Si hay más de 11 actividades, poniendo el dedo sobre la flecha verde de la parte inferior derecha de la pantalla, se mostrarían el resto de actividades.

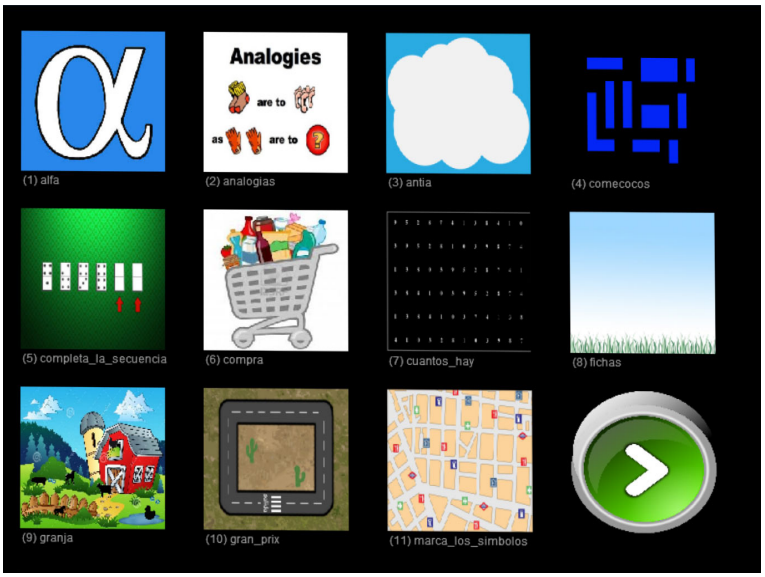


Figura 9: Menú de la herramienta

Sus atributos son los siguientes:

- **listaAreas**: objetos *Area* de la actividad.
- **feedback**: objeto *Feedback* de la actividad.
- **layerFondo**: capa que contiene el fondo de la actividad.
- **layerArea**: capa que contiene las áreas de la actividad.
- **layerFeedback**: capa que contiene el feedback de la actividad.
- **f**: fichero XML que contiene la actividad.
- **numAreas**: total de áreas de la actividad.
- **acabado**: indica si el XML de la actividad ha acabado de cargarse.
- **hayFeedback**: indica si la actividad tiene feedback.
- **numTareas**: total de tareas de la actividad.
- **layerMenu**: capa en la que se dibujará el menú.
- **listaIconos**: lista de áreas que representan los iconos de menú. Por pantalla caben 11 iconos.
- **listaXML**: lista de ficheros XML que aparecerán en el menú.
- **hayMenu**: si vale *true*, indica que se ha elegido la opción de menú; si vale *false*, significa que se ha elegido la opción del fichero de arranque "config.xml".
- **menuCompleto**: si vale *true*, indica que las imágenes del menú han terminado de cargarse.
- **totalJuegos**: número de juegos que aparecerán en el menú.

Las funciones de la clase *TratarXML* son las siguientes:

- **TratarXML**: es la función constructora, y como parámetros sólo recibe las capas (menú, fondo, área y feedback) que serán usadas en la clase *Inicio*.
- **loadedIcon**: función encargada de la creación de áreas de los iconos y de su colocación en el menú.

- **loadXML**: función que se encarga de cargar el XML que se le pasa como parámetro. Si el fichero que se le pasa es el de arranque (config.xml) llama a la función *loadedConfig*; si se le pasa cualquier otro fichero XML, llama a la función *loadedCompleteHandler*.
- **loadedCompleteHandler**: esta función es la que realiza el tratamiento completo del fichero XML. Primero se guarda el valor del verbose del XML (para saber si habrá que dibujar las áreas o no), después cuenta el número de tareas contenidas en el XML y por último para cada tarea, llama a la función *TratarSiguienteTarea*.
- **loadedConfig**: función que lee del fichero de arranque el juego a ejecutar y vuelve a llamar a la función *loadXML* para cargar directamente el juego a través de la función *loadedCompleteHandler*.
- **TratarSiguienteTarea**: esta función realiza el tratamiento de cada una de las partes del XML, llamando a las funciones *TratarFondo*, *TratarArea* y *TratarFeedback*.
- **TratarFondo**: trata el fondo de la actividad.
- **TratarArea**: trata las áreas de la actividad.
- **TratarFeedback**: trata el feedback de la actividad.
- **limpiarCapa**: elimina todos los elementos de las capas.
- **ocultarMenu**: función que se encarga de eliminar todos los elementos que se encuentran en la capa de menú.

En el diagrama se refleja la relación entre la clase *TratarXML* y la clase *Area* indicando que un objeto TratarXML puede tener asociadas 0 o más áreas, su relación con la clase *Feedback* indicando que puede tener asociado un feedback o no y su relación con la clase *Cargar_imagen*, con la cuál se relaciona al tener que tratar el fondo de la actividad.

Aparte de las funciones anteriores que son usadas por todo objeto *TratarXML*, hay otras funciones que sólo se usan cuando la actividad requiere un sonido de fondo (por ejemplo la actividad *Marca los símbolos* o la actividad *Viajes*) que puede repetirse periódicamente o no. Para ello, se usan las funciones:

- **timeHandler**: reproduce el sonido al inicio de la actividad.
- **repetición**: reproduce el sonido cada cierto tiempo durante la actividad.

Por último, si una actividad requiere un cambio de fondo (por ejemplo la actividad *Lista de la compra*), se usará la función ***cambioFondo***.

Ejemplos de código de las funciones más relevantes se pueden ver en el anexo 1.

3.7. Clase *Inicio*

En un comienzo, la clase *Inicio* era la encargada de realizar la ejecución de la herramienta apoyándose principalmente en la clase *TratarXML* puesto que es la que tiene toda la información relativa al menú y a las actividades. Sin embargo, debido a la conclusión que se llegó en el análisis de que podría ser interesante guardar información sobre las actividades, también es la encargada de generar el fichero de resultados (o de logs) de las actividades con información relativa a estas. El formato del fichero se muestra en la figura 10.

El fichero XML de logs tendrá el mismo nombre que la actividad y en él se irán guardando las partidas que se vayan jugando con la información de la fecha, la hora, el tiempo jugado, las acciones correctas e incorrectas realizadas y la indicación de si el juego se ha completado del todo o no.

```
<partida>
<fecha dia="dd/mm/aaaa" hora="hh:mm:ss"/>
<tiempo_jugado segundos="[num]"/>
<completitud acciones_correctas="[num]" acciones_incorrectas="[num]" acabado="si/no"/>
</partida>
<partida>
...
</partida>
```

Figura 10: Formato del fichero XML de resultados de una actividad

Los atributos de la clase *Inicio* son los siguientes:

- **ListaTuioObject**: objetos puestos sobre la mesa.
- **fXML**: objeto *TratarXML* con toda la información de la actividad.
- **layerFondo**: capa que contiene el fondo de la actividad.
- **layerArea**: capa que contiene las áreas de la actividad.
- **layerFeedback**: capa que contiene el feedback de la actividad.
- **layerMenu**: capa en la que se dibujará el menú.
- **layerVerbose**: capa en la que se dibujarán las áreas si el atributo *verbose* del fichero *TratarXML* asociado a la clase *Inicio* es *true*.
- **stream**: fichero de resultados en el que se actualizará la información de las partidas jugadas.
- **inicio**: tiempo de inicio de la actividad.
- **fin**: tiempo de fin de la actividad.
- **bien**: total de acciones realizadas correctamente durante la actividad.
- **mal**: total de acciones realizadas incorrectamente durante la actividad.
- **finalJuego**: variable que controla si ha acabado el juego o no.
- **bucle**: bucle del juego.

Las funciones de la clase *Inicio* son las siguientes.

- **Inicio**: constructor de la clase.
- **addTuioObject**: función que añade un objeto a la lista <ListaTuioObject>.
- **updateTuioObject**: función que actualiza un objeto de la lista <ListaTuioObject>.
- **removeTuioObject**: función que borra un objeto de la lista <ListaTuioObject>.
- **addTuioCursor**: función que añade un cursor a la lista <ListaTuioObject>.
- **updateTuioCursor**: función que actualiza un cursor de la lista <ListaTuioObject>.
- **removeTuioCursor**: función que borra un cursor de la lista <ListaTuioObject>.
- **init**: clase constructora de la herramienta. Inicializa las capas de menú, fondo, área, feedback y verbose y realiza la conexión con el tabletop a través de *tuio*. Por último, inicia el bucle del juego.
- **juego**: función que está continuamente ejecutándose y que comprueba el estado de juego en cada momento.
- **actualizaListaJuego**: función encargada de la comprobación de las áreas de la tarea que se está ejecutando.

- **borrarListaFiduciales**: función usada por la función *juego* para eliminar los objetos que hay sobre la mesa.
- **handleKeyDown**: función de tratamiento de teclas usada principalmente para facilitar el desarrollo de actividades, de modo que las teclas del 1 al 9 permiten la carga directa de los 9 primeros juegos que aparezcan en el menú sin necesidad de seleccionarlos manualmente y la tecla 0 permite resetear la actividad en cualquier momento.
- **cerrarXML**: función que se encarga de crear/actualizar el fichero XML de logs de la actividad que se esté ejecutando en ese momento.
- **crearPartida**: función que guarda la información de la última partida ejecutada en un XML interno que posteriormente la función *cerrarXML* volcará en el fichero de logs.

Por último, comentar que como funcionalidad adicional se implementó la posibilidad de pausar el juego y de volver al menú durante la ejecución de la actividad mediante la colocación de dos objetos especial sobre la mesa.

Ejemplos de código de las funciones más relevantes se pueden ver en el anexo 1.

3.8. Fichero de definición de actividades

Como se ha comentado en el apartado de diseño e implementación, cada actividad necesita 4 elementos básicos: imagen de fondo, áreas, objetos asociados y algún tipo de feedback, cuya información se ha de guardar en un fichero XML para facilitar su tratamiento a la herramienta.

El formato del fichero XML se muestra en la figura 11:

```
<juego>
    <verbose valor="si/no"/>
    <tarea>
        <fondo>
            <color rgb="rrggbb"/>
            <icono path="carpeta/nombrefichero"/>
            <imagen path="carpeta/nombrefichero" cambiar="si/no" tiempo="num_segundos"/>
            <imagen2 path="carpeta/nombrefichero"/>
            <sonido path="carpeta/nombrefichero" repetir="si/no" tiempo="num_segundos"/>
        </fondo>
        <area>
            <posicion x="[num]" y="[num]" ancho="[num]" alto="[num]" alfa="si/no"/>
            <imagen path="carpeta/nombrefichero" x="[num]" y="[num]" ancho="[num]" alto="[num]"/>
            <imagenBien img="carpeta/nombrefichero" x="[num]" y="[num]" ancho="[num]" alto="[num]"/>
            <imagenMal img="carpeta/nombrefichero"  x="[num]" y="[num]" ancho="[num]" alto="[num]"/>
            <fidasoc id="[num]" orientable="si/no"/>
            <fid id="[num1,…numN/*],[num]fichas" orientable="ang1,ang2,…" correcto="si/no" sonido="carpeta/fichero.mp3"/>
            <fid id="[num1,…numN/*],[num]fichas" orientable="ang1,ang2,…" correcto="si/no" sonido="carpeta/fichero.mp3"/>
            ...
        </area>
        ...
        <feedback>
            <pos x="[num]" y="[num]" ancho="[num]" alto="[num]"/>
            <imagen_acierto path="carpeta/nombrefichero"/>
            <imagen_fallo path="carpeta/nombrefichero"/>
            <imagen_neutro path="carpeta/nombrefichero"/>
            <imagen_completado path="carpeta/nombrefichero"/>
            <sonido_acierto path="carpeta/nombrefichero"/>
            <sonido_completado path="carpeta/nombrefichero"/>
        </feedback>
    </tarea>
    ...
</juego>
```

Figura 11: Formato del fichero XML de una actividad

La explicación detallada del fichero se puede consultar en el anexo 1.

4. Evaluación de la herramienta

Una vez que se tuvo un prototipo implementado de la herramienta, alumnos de la asignatura Diseño Centrado en el Usuario de Grado de Ingeniería Informática de la Escuela de Ingeniería y Arquitectura de Zaragoza hicieron, como práctica opcional de la asignatura, el diseño de una actividad para el tabletop NIKVision. Participaron 16 alumnos en total y se dividieron en 4 grupos con 4 personas por grupo.

El objetivo principal que se buscaba era ver si la herramienta permitía realizar el diseño de actividades de manera sencilla para aquellas personas que no estuvieran familiarizadas ni con el tabletop NIKVision ni con la herramienta mediante una evaluación de su usabilidad.

4.1. Metodología

En el siguiente apartado se explicará el procedimiento que se siguió para realizar la evaluación del prototipo de la herramienta.

4.1.1. Antes de la sesión

Una semana antes de la primera sesión de evaluación, los alumnos que iban a participar se dividieron en grupos de 4 personas, desempeñando cada miembro del grupo un rol diferente. Dichos roles eran los siguientes:

- Diseñador: encargado de crear el concepto de juego y de coordinar al resto de los miembros del equipo para que el juego resultante cumpla con su diseño.
- Juguetero: encargado de conseguir o de construir los juguetes necesarios y de su adaptación para que puedan ser usados en el tabletop NIKVision.
- Grafista: encargado de conseguir o crear los recursos multimedia necesarios para el juego, tales como gráficos, animaciones y sonidos.
- Desarrollador: encargado de implementar el juego en el XML de la herramienta para que sea ejecutado con la herramienta en el tabletop NIKVision.

También se le proporcionó a cada uno documentación sobre lo que es NIKVision [NIK14], ejemplos de juegos desarrollados para NIKVision y documentación personalizada para cada rol. Una vez decididos los roles, cada grupo pensaba el juego que quería hacer y mandaba un correo para confirmar si el juego cumplía los requisitos necesarios para ser desarrollado sobre NIKVision, los cuáles se resumen en:

- Que estuviera dentro de las posibilidades técnicas que ofrece la mesa; es decir, que la actividad esté basada en la manipulación de objetos sobre la mesa.
- Que previsiblemente diera tiempo a realizarla en el tiempo que duraba la sesión (2-3 horas).

Como comentario, ninguna de las ideas propuestas tuvo que ser rechazada, sino como mucho simplificada ligeramente o dividida en diversas fases para asegurar que se podría realizar por lo menos un mínimo de la actividad durante la sesión.

Los alumnos que participaron en total en la práctica fueron 16 estudiantes, por tanto se hicieron 4 sesiones de 2-3 horas cada una. Los alumnos tenían edades comprendidas entre los 21 y 29 años, siendo todos estudiantes de grado en informática como se ha comentado antes.

4.1.2. Durante la sesión

Las sesiones tuvieron lugar en el laboratorio 2.07 del edificio Ada Byron de la Escuela de Ingeniería y Arquitectura de Zaragoza y la metodología de cada una de las sesiones fue siempre la misma: durante la sesión había dos evaluadores, de modo que conforme iban viniendo los alumnos, uno de ellos apuntaba sus nombres, sus edades, los roles que desempeñaban y el juego que iban a realizar. Seguidamente, el otro evaluador explicaba de forma práctica el funcionamiento de NIKVision y ya a continuación cada miembro del grupo se ponía a desempeñar su papel.

La forma de organizarse era la siguiente:

- Al desarrollador se le dejaba el ordenador conectado al tabletop para que pudiera estar haciendo pruebas sobre la mesa a la vez que programaba.
- Al juguetero se le proporcionaban materiales tales como papel, tijeras, pegamento, corcho, cuter...para que pudiera ir haciendo los juguetes en el caso de crearlos de cero o se le proporcionaba directamente los juguetes que luego él adaptaba para que fueran reconocidos en la mesa. También se le facilitaban los fiduciales ya impresos.
- Al grafista se le dejaba un ordenador del laboratorio para que pudiera buscar los recursos multimedia necesarios para su actividad. El ordenador en cuestión también disponía de programas de trabajo gráfico como photoshop o gimp para poder realizar los gráficos o animaciones.
- Al diseñador se le proporcionaba otro ordenador por si necesitaba ayudar al grafista en algún aspecto del diseño del juego, aunque la mayor parte del tiempo se dedicaba a coordinar el trabajo entre el desarrollador y el grafista, asegurándose de que el primero tuviera todo lo necesario del segundo para continuar.

Durante la sesión, uno de los evaluadores se dedicaba a resolver las dudas que les iban surgiendo proporcionando soporte técnico (dudas relacionadas con la mesa, la detección de los objetos...) mientras el otro se encargaba de tomar notas de observación cuando ocurrían incidencias durante la sesión (dudas y malfuncionamientos) así como posibles mejoras a realizar que se derivaban de dichas incidencias.

4.1.3. Después de la sesión

Al finalizar la sesión, todos los alumnos completaron un cuestionario IMI (Intrinsic Motivation Inventory) [IMI85] y aquellos que habían desempeñado el rol de desarrollador, hicieron además un cuestionario SUS (System Usability Scale) [SUS96]. También se les pidió que dieran su opinión sobre la sesión con dos preguntas de libre redacción (ver tabla 4).

El cuestionario SUS permite obtener una visión global de la usabilidad de la herramienta desarrollada. Está compuesto por 10 preguntas evaluadas según una escala de Likert de 1 a 7 puntos (significando el 1 *totalmente en descuerdo* y el 7 *totalmente de acuerdo*) que responden a declaraciones del estilo: "la herramienta ha sido fácil de usar" (ver tabla 5). Este cuestionario sólo lo rellena el desarrollador.

El cuestionario IMI pretende obtener información sobre la experiencia subjetiva relativa a la actividad llevada a cabo por los alumnos en la sesión. El cuestionario IMI utilizado está formado por 19 preguntas también evaluadas mediante la escala de Likert anterior y además cada pregunta permite medir una de las siguientes sub-escalas: interés/diversión, competencia percibida y presión/tensión percibida (ver tabla 6).

Si bien el cuestionario IMI permite evaluar subescalas adicionales, se decidió centrarse únicamente en esas tres porque lo que se quería saber era si los roles se desempeñaban satisfactoriamente sin resultar ninguno demasiado tedioso (interés/diversión), pudiendo cumplir cada uno con los requisitos propios de cada rol (competencia percibida) y si además el desempeño de dichos roles se podía realizar de forma fluida sin demasiado estrés (presión/tensión). Este cuestionario lo rellenaron todos los alumnos.

En cuanto a las observaciones realizadas durante la sesión, se rellenó una hoja en la que se iba anotando, para cada sesión, la hora de la incidencia, el rol de la persona que la detectó, el tipo de incidencia (consulta o problema técnico) que surgió y la descripción de la incidencia (ver tabla 7).

Pregunta
1. Explica brevemente en qué aspectos la herramienta no ha sido capaz de soportar tus ideas
2. Cualquier otro comentario general sería bienvenido

Tabla 4: Preguntas de libre redacción

Pregunta
1. Creo que me gustaría usar la herramienta con frecuencia
2. He encontrado la herramienta excesivamente compleja
3. Pienso que la herramienta ha sido fácil de usar
4. Creo que necesitaría soporte de una persona técnica para usar la herramienta
5. He visto que las distintas funcionalidades de la herramienta están bien integradas
6. Creo que hay mucha inconsistencia en la herramienta
7. Imagino que la mayoría de la gente aprenderá a usar la herramienta muy rápidamente
8. Encuentro extraña la forma de usar la herramienta
9. Mientras usaba la herramienta, me he sentido confiad@
10. He tenido que aprender muchas cosas para poder empezar a usar la herramienta

Tabla 5: Cuestionario SUS

Pregunta	Sub-escala
1. Durante la actividad pensaba en lo mucho que estaba disfrutando	Interés/diversión
2. No me he sentido nervios@ para nada en la actividad	Presión/tensión
3. La actividad no ha sido capaz de mantener mi atención	Interés/diversión
4. Pienso que lo he hecho realmente bien durante la actividad	Interés/diversión
5. La actividad me ha parecido muy interesante	Interés/diversión
6. Me he sentido muy tens@ durante la actividad	Presión/tensión
7. La actividad ha sido muy divertida	Interés/diversión
8. Creo que me he defendido mejor en la actividad que el resto de mis compañer@s	Interés/diversión
9. Durante la actividad, en todo momento sentía que tenía el control de lo que hacía	Competencia percibida
10. No he podido hacer algunas de las ideas que tenía para la actividad	Competencia percibida
11. Durante la actividad, estaba pensando en otras cosas	Interés/diversión
12. Me he distraído frecuentemente con otras cosas ajenas a la actividad	Interés/diversión
13. La actividad me ha abstraído por completo	Interés/diversión
14. La actividad ha sido intrínsicamente interesante	Interés/diversión
15. La actividad ha despertado mi curiosidad	Interés/diversión
16. Me he sentido tan inmers@ en la actividad que he perdido la noción del tiempo	Interés/diversión
17. Salgo con una opinión muy positiva de la actividad	Interés/diversión
18. La actividad ha estimulado mi imaginación	Interés/diversión
19. Estoy deseando participar en otra actividad similar	Interés/diversión

Tabla 6: Cuestionario IMI

Hora	Rol	Consulta, problema técnico	Breve descripción
...	...	...	...

Tabla 7: Formato de la hoja de observaciones

4.2. Sesiones de evaluación

En este apartado, se van a explicar las actividades que se realizaron en las 4 sesiones de prácticas.

4.2.1. Comecocos

La actividad que se desarrolló en la primera sesión fue una adaptación del videojuego Pac-Man [PAC14]. El concepto de juego es el siguiente: se tiene un laberinto lleno de obstáculos y de cocos (figura 12) por el cuál tiene que moverse el comecocos sin chocar con las paredes mientras va "comiendo" los cocos que ve a su paso. A su vez, hay un fantasma que lo va persiguiendo por el laberinto. La actividad terminaría cuando el comecocos se ha comido todos los cocos o cuando el fantasma pilla al comecocos.

En un inicio también, se quería hacer que el comecocos pudiera convertirse en comefantasmas al pasar por áreas especiales, pero debido a que ese comportamiento no es soportado por la herramienta (ya que ésta no permite programación) ese requisito de juego no se pudo desarrollar.

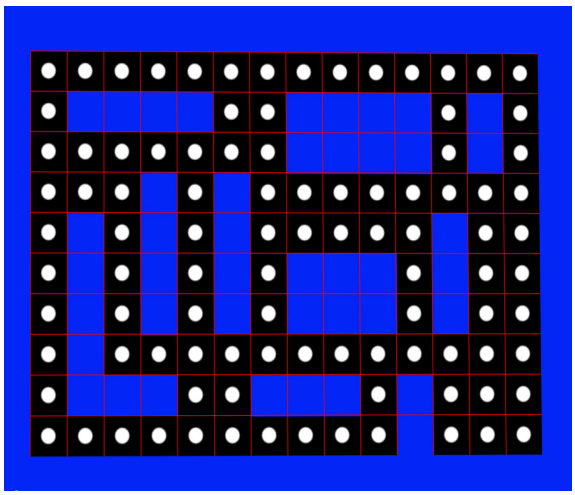


Figura 12: Imagen de fondo de la actividad *Comecocos* con sus áreas de interacción.

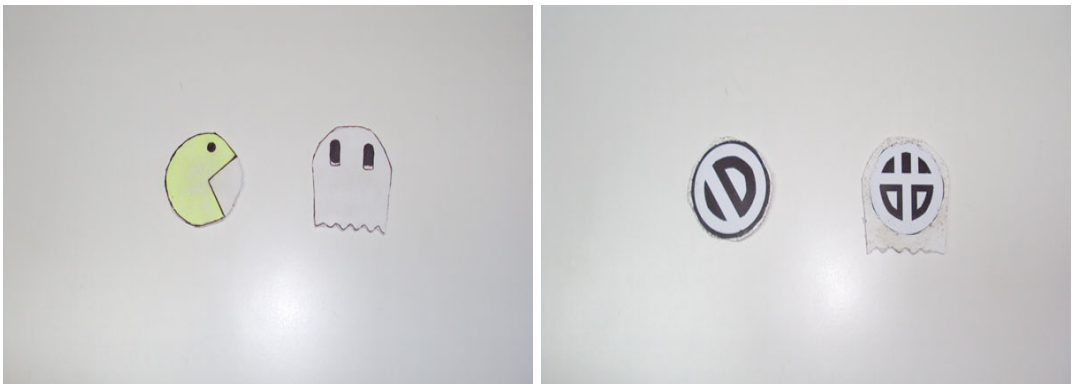


Figura 13: Juguetes de la actividad *Comecocos*

Por lo tanto, en esta actividad se manejan los dos juguetes mostrados en la figura 13: el comecocos y el fantasma. En la figura 12 se puede ver que hay áreas de interacción en todo el escenario, para que así en todo momento se pueda detectar cuándo el comecocos se come un coco y cuándo el comecocos o el fantasma chocan con la pared, y así además poder ejecutar un sonido u otro. Además, cada vez que el comecocos come un coco, ese área pasa considerarse correcta y por lo tanto el coco "desaparece".

4.2.2. Granja

La actividad que se desarrolló en la segunda sesión temía como temática una granja.

El concepto de juego es el siguiente: en la granja se pueden ver siluetas de 6 animales (figura 14): un caballo, un cerdo, una gallina, una oveja, un pato y una vaca. El objetivo de la actividad es que el usuario sitúe los 6 juguetes de animales en su silueta correspondiente. Para ayudar al usuario, en esta actividad se hace uso del feedback, de modo que:

- Cada vez que se coloca un animal bien, aparece una cara sonriente en la esquina superior izquierda de la pantalla durante 3 segundos.
- Cada vez que se coloca un animal mal, aparece una cara triste en la esquina superior izquierda de la pantalla durante 3 segundos y suena el sonido correspondiente a la silueta sobre la que se ha colocado el animal para dar una pista de a qué animal corresponde dicha silueta.

La actividad concluye cuando se han colocado los 6 animales en sus siluetas correspondientes y cuando se ha colocado un juguete especial que no tiene silueta (el perro) en cualquier lugar de la pantalla.

En esta actividad se manejan los 7 juguetes mostrados en la figura 15: el cerdo, el caballo, la gallina, la oveja, el pato, el perro y la vaca. En la figura 14 se puede ver que cada silueta tiene su área de interacción excepto la del perro, que como se ha dicho, ocupa toda la pantalla.

Figura 14: Imagen de fondo de la actividad *Granja* con sus áreas de interacción.

Figura 15: Juguetes de la actividad *Granja*.

4.2.3. Series

La actividad que se desarrolló en la tercera sesión fue sencilla en concepto pero fue de las que más trabajo llevó. Como se puede ver en la figura 16, el escenario de la actividad consiste en una serie de figuras que siguen algún tipo de patrón. El objetivo de la actividad es que el usuario sitúe la figura o figuras correspondientes para completar la serie. Para ayudar al usuario, en esta actividad se hace uso de las imágenes bien y mal asociadas a áreas, de modo que:

- Cada vez que se coloca una pieza bien, aparece una cara sonriente encima del área en la que se ha colocado la pieza, manteniéndose ya desde entonces ahí.
- Cada vez que se coloca una pieza mal, aparece una cara triste encima del área en la que se ha colocado la pieza, y se mantiene ahí hasta que la pieza se quita.

La actividad concluye cuando se han completado las 8 series de las que consta la actividad.

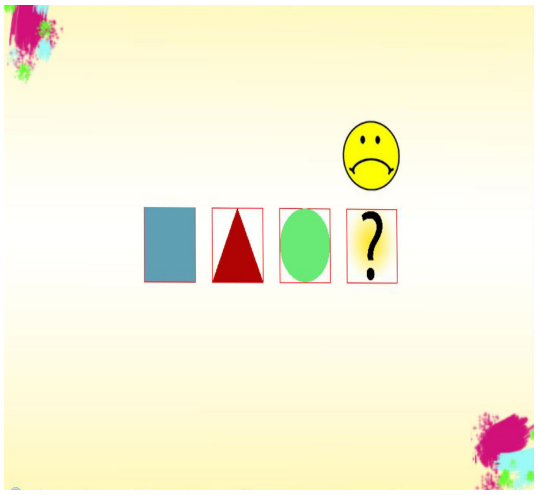


Figura 16: Imagen de fondo de la actividad *Series* con sus áreas de interacción.

En esta actividad se manejan los 12 juguetes mostrados en la figura 17. En la figura 16 se puede ver que las áreas de interacción se encuentran rodeando las figuras y los interrogantes que indican dónde colocar la pieza.

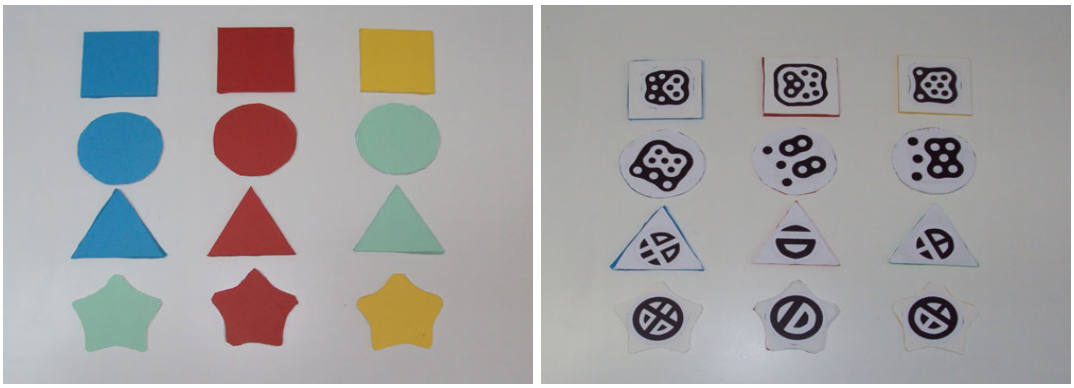


Figura 17: Juguetes de la actividad *Series*

4.2.4. Gran Prix

La actividad que se desarrolló en la cuarta y última sesión fue más parecida en complejidad a la del *Comecocos* de la primera sesión. En la figura 18 se puede ver el circuito que tenía pensado el grupo.

Como se puede ver en la figura 19, el circuito tuvo que ser modificado por motivos que posteriormente se explicarán y al final el escenario de la actividad consistió en un circuito rectangular rodeado de hierba y de desierto, por el cuál van a correr un coche y una moto.

El objetivo de la actividad es hacer una carrera con los dos transportes siguiendo el circuito sin salirse de los bordes. Cada vez que un transporte pasa por un tramo del circuito, suena el ruido de motor correspondiente y cada vez que un transporte se sale del circuito ya sea por el exterior o por el interior, suena un sonido de derrape.

La actividad empieza situando los dos transportes en la casilla la salida. De ese modo, el semáforo que indica el comienzo de la carrera se activa junto con un sonido de cuenta atrás. Cuando el semáforo se pone en verde empieza la carrera. La actividad termina cuando uno de los dos transportes llega a la meta, resultando el ganador.

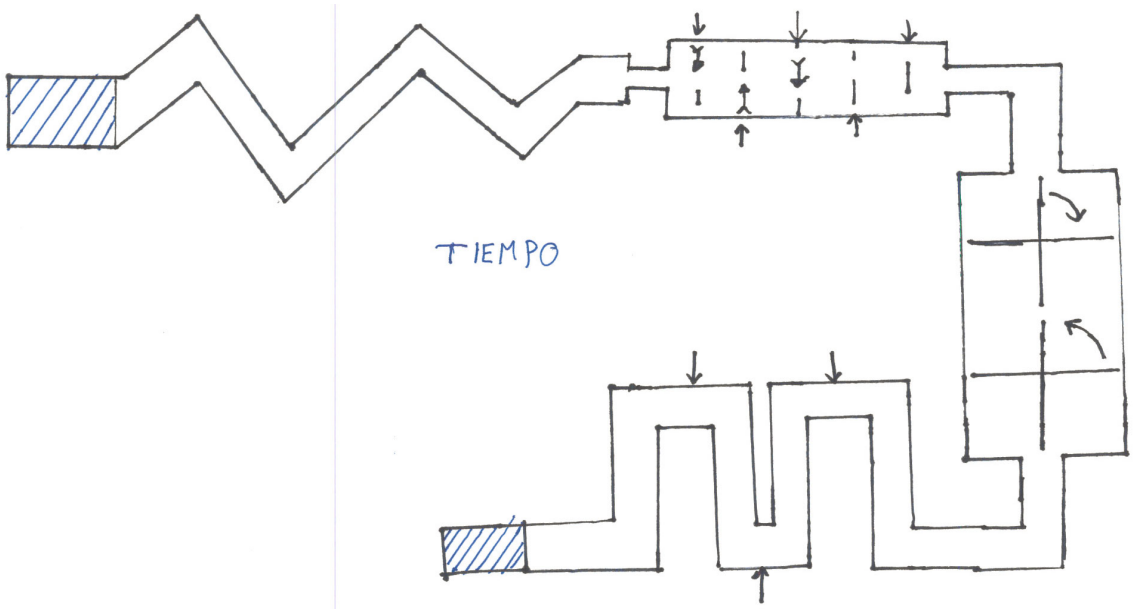


Figura 18: Circuito inicial de la actividad *Gran Prix*

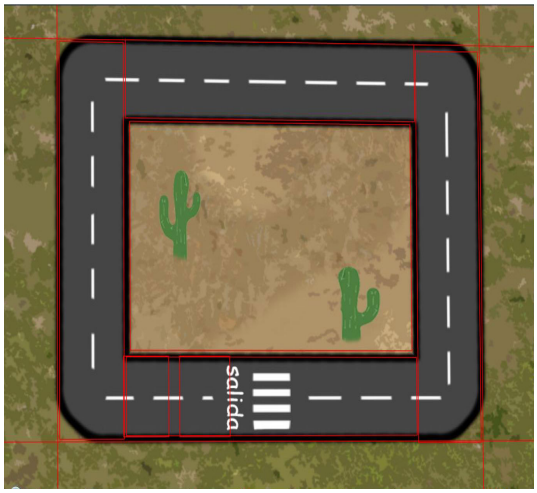


Figura 19: Imagen de fondo de la actividad *Gran Prix* con sus áreas de interacción.

En esta actividad se manejan los dos juguetes mostrados en la figura 20: el coche y la moto. En la figura 19 se puede ver que las áreas de interacción son todo el escenario, ya que hay que detectar

el circuito, el exterior e interior del circuito, la salida y la meta.

Figura 20: Juguetes de la actividad *Gran Prix*

4.3. Resultados

A continuación se muestran los resultados generales obtenidos en la evaluación de la herramienta. Las respuestas de los alumnos a las preguntas de libre redacción, las observaciones anotadas durante las sesiones y los cuestionarios IMI separados por actividad se puede consultar en el anexo 2.

4.3.1. Resultados cuestionario SUS

En total, el resultado promedio del SUS es de 74,57 sobre 100 con una desviación típica de 15,33. Los resultados individuales del SUS se muestran a continuación en la figura 21.

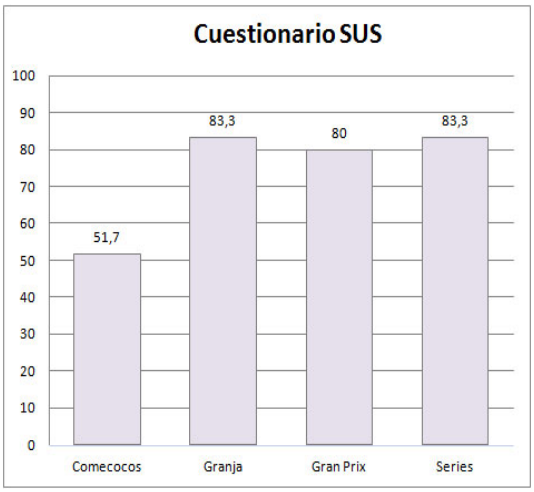


Figura 21: Cuestionarios SUS realizados por los desarrolladores

Como se puede observar, los resultados son bastante parecidos entre sí exceptuando el primer caso, siendo los resultados de *Granja*, *Gran Prix* y *Series* superiores al tercer cuartil, con lo cuál la herramienta tiene altas calificaciones de usabilidad y satisfacción.

Analizando los resultados, una posible razón para esa diferencia de resultados de la actividad *Comecocos* con respecto a las demás puede ser que de las cuatro actividades desarrolladas, fue la más complicada y tediosa de implementar debido a la gran cantidad de áreas que hacían falta (140 áreas) y por no tener la herramienta capacidad de programar. Además el *Comecocos* fue la actividad que más hubo que adaptar a la mesa no pudiendo realizar muchas de las ideas que tenían en mente.

4.3.2. Resultados cuestionario IMI

En cuanto a los cuestionarios IMI, se van a hacer dos tipos de análisis: el primero se centrará en los resultados obtenidos en función de los roles y el segundo en los resultados obtenidos en función de las actividades. En el anexo 2 se pueden consultar los resultados obtenidos en cada actividad por separado.

A continuación, se van a mostrar los resultados obtenidos agrupados por actividad (figura 22):

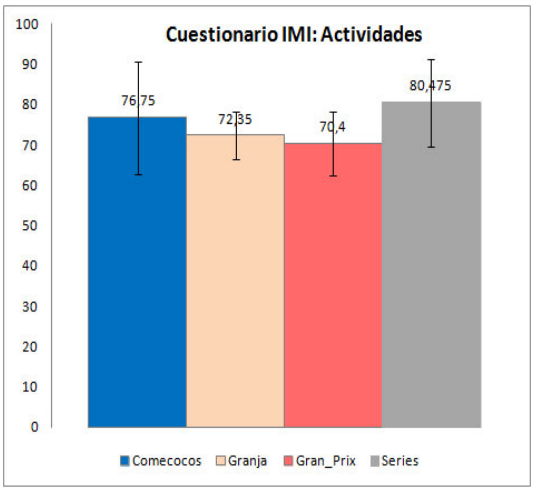


Figura 22: Cuestionarios IMI agrupados por actividad

Los resultados conjuntos de las 4 actividades superan en media el 70 % de satisfacción, siendo el más elevado el resultado de la actividad *Series* y el más bajo el de la actividad *Gran Prix*. Además, tanto el resultado de *Series* como el de *Comecocos* superan el tercer cuartil y la media de las 4 actividades es del 75 %, con lo cual se puede concluir que el nivel global de satisfacción es bastante elevado.

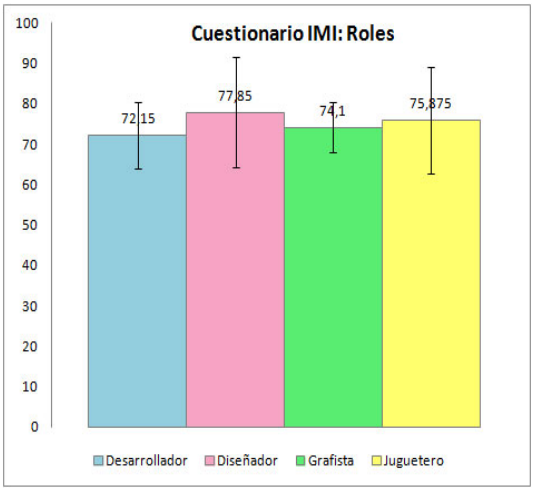


Figura 23: Cuestionarios IMI agrupados por rol

En cuanto a los resultados agrupados por roles (figura 23) se observa que éstos superan también en media el 70 % de satisfacción, siendo el más elevado el resultado del rol diseñador y el más bajo el de desarrollador, pero la diferencia entre estos valores extremos es bastante pequeña. Además, tanto el resultado del rol diseñador como el del rol juguetero superan el tercer cuartil y la media de los 4 roles es también del 75 %, con lo cual el nivel global de satisfacción es bastante elevado.

4.3.3. Resultados preguntas libre redacción y observaciones

En cuanto a los resultados globales de las preguntas de libre redacción, vamos a mostrar a continuación en una tabla algunas de las opiniones más interesantes que dieron los alumnos al finalizar la sesión, entendiendo por "interesante" aquella opinión que ha permitido extraer un nuevo requisito para la herramienta.

En la tabla 8 se muestra la actividad o actividades en las que se hizo el comentario, el rol del alumno que lo hizo y el comentario en sí.

Actividades	Roles	Comentario
Comecocos	Desarrollador y Grafista	El diseño de la actividad no permite programación
Series	Desarrollador	Estaría bien permitir volver a la tarea anterior mediante algún área interactiva
Gran Prix	Grafista y Juguetero	Dificultad en el diseño de escenarios por sólo permitir áreas rectangulares

Tabla 8: Resumen de opiniones de las preguntas de libre redacción

En la actividad ***Comecocos***, el desarrollador y el grafista echaron de menos la capacidad de programar, puesto que muchas cosas de las que tenían en mente (fantasmas controlados por el ordenador, las píldoras para matar fantasmas, puntos, controlar número de vidas...) no pudieron hacerse.

En la actividad ***Series***, se planteó que sería interesante volver a una tarea anterior en vez de sólo dar la opción de resetear siempre desde cero el juego, ya que como la actividad era un conjunto de muchas series, podría ser interesante volver a una serie anterior concreta en vez de tener que estar obligado a repetir siempre la actividad desde el principio, ya que las series van subiendo de dificultad y a lo mejor sólo te interesa hacer repetir las últimas series.

En la actividad ***Gran Prix***, la principal limitación fue el hecho de tener que usar obligatoriamente áreas rectangulares, de modo que hubo que modificar el circuito que inicialmente tenía zonas curvas que las áreas no iban a poder cubrir de forma suficientemente ajustada.

Como comentario general, se puede ver que los alumnos que desempeñaron los roles de desarrollador, grafista y juguetero son los que aportaron más opiniones, mientras que todos los diseñadores opinaron que la herramienta les permitía hacer todo lo necesario.

Por lo tanto, como conclusión global a las respuestas dadas por los alumnos, se pueden extraer los siguientes nuevos requisitos:

- Incluir la posibilidad de programar comportamientos específicos para las actividades.
- Permitir volver a la tarea anterior en vez de al principio de la actividad.
- Poder definir áreas de distintas formas, no sólo rectangulares.

De dichos requisitos se implementó el tercero, puesto que entre las actividades que se habían elegido para demostrar la validez de la herramienta se encontraba el Tangram y este tiene piezas compuestas por triángulos cuya definición podría facilitarse en caso de trabajar con áreas no rectangulares.

Como conclusión general a las observaciones realizadas en las 4 actividades, se puede extrae un nuevo requisito a implementar en la herramienta y una mejora a realizar:

- **Nuevo requisito**: para facilitar el uso de pruebas mientras se desarrolla la actividad, la herramienta ha de volver al menú/resetear la actividad actual cuando finalice.
- **Mejora**: reescribir algunas secciones de la documentación.

4.4. Conclusiones de la evaluación

Como conclusión global de la evaluación se puede decir que el prototipo inicial de la herramienta tenía una buena usabilidad al obtener un resultado de casi un 75 % en el cuestionario SUS realizado por los desarrolladores.

Además, los resultados de los cuestionarios IMI realizados para medir el nivel de satisfacción a la hora de utilizar la herramienta están siempre entre el 70 y el 75 %, con lo que se puede decir que el manejo de la herramienta es bastante sencillo.

Por último, comentar que gracias a las observaciones tomadas durante las sesiones y a los comentarios realizados por los alumnos, se ha podido realizar una mejora de la herramienta (permitir una vuelta al menú desde las actividades) y se han proporcionado ideas para posibles trabajos futuros, tales como incluir la posibilidad de programación.

5. Desarrollo de actividades

A continuación se explicará en detalle el desarrollo de las actividades que se propusieron en el apartado *Propuesta de actividades a realizar*.

5.1. Actividad *Lista de la compra*

La primera imagen de la actividad es la que se muestra en la parte izquierda de la figura 24, y pasados 5 segundos se mostraría la imagen que aparece en la parte derecha la figura 24.

Figura 24: A la izquierda: imagen de inicio de la actividad *Lista de la compra*. A la derecha: área de interacción de la actividad *Lista de la compra*.

En cuanto al feedback, en este caso utilizaremos las tres imágenes de la figura 25, de forma que en la parte superior de la pantalla, aparecerá la imagen con expresión neutra que se mantendrá mientras no se ponga ningún objeto en el área interactiva de la bolsa. Cuando se ponga un objeto:

- Si el objeto es uno de los de la lista, la imagen de cara neutra pasará a ser la imagen de cara feliz durante 3 segundos, indicando que el objeto se ha colocado bien, y además se reproducirá un sonido de acierto. Pasados esos 3 segundos, la cara volverá a ser neutra.
- Si el objeto no es uno de los de la lista, la imagen de cara neutra pasará a ser la imagen de cara triste durante 3 segundos, indicando que el objeto se ha colocado mal, y además se reproducirá un sonido de fallo. Pasados esos 3 segundos, la cara volverá a ser neutra.

Figura 25: Imagen de cara neutra, cara feliz y cara triste para el feedback de las actividades

Para esta actividad, los objetos correctos serán los juguetes de alimentos correspondientes a la lista mostrada en la figura 26 (pollo, guisantes, cebolla, tomate y leche) y los incorrectos, el resto de juguetes.

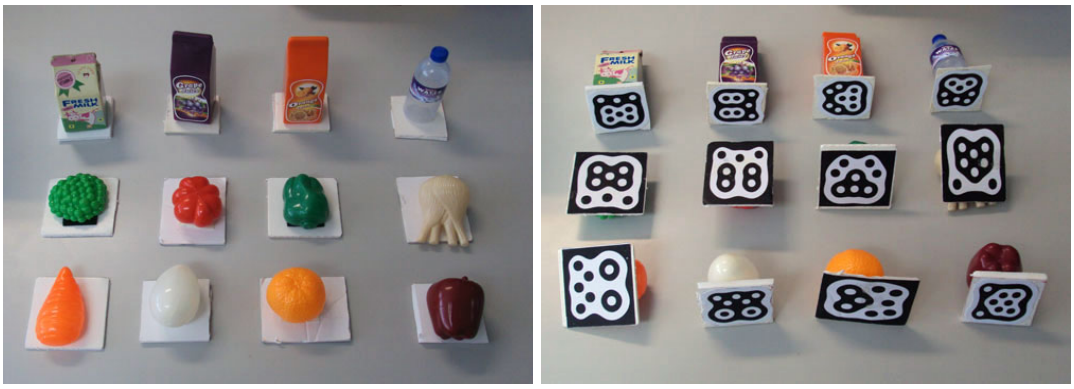


Figura 26: Juguetes de la actividad *Lista de la compra*

5.2. Actividad *Viajes*

En este caso sólo hay una imagen: el mapa de Europa mostrado en la figura 27, y las áreas de interacción son algunos de los países, como se muestra en la figura.

Para esta actividad, los objetos correctos serán los juguetes de medios de transporte mencionados en la grabación inicial de la actividad situados en los países mencionados. Por ejemplo, si la grabación de inicio de la la actividad dice: *"Iré primero a Grecia en barco, después a Rumanía en tren y por último a Alemania en avión"*, la actividad será correcta cuando el juguete del barco esté en el área de Grecia, el juguete del tren en el área de Rumanía y el juguete del avión en el área de Alemania.

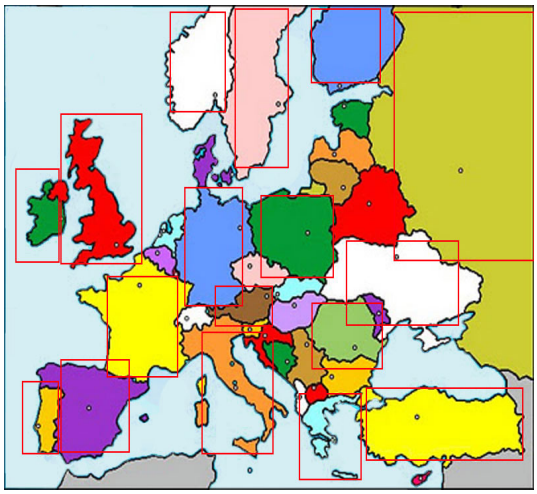


Figura 27: Imagen de fondo y áreas de interacción de la actividad *Viajes*.

En cuanto al feedback, se usarán las mismas imágenes utilizadas en la actividad *Lista de la compra*, sólo que ahora la forma de activarse será la siguiente:

- Si se coloca el barco en Grecia, el tren en Rumanía o el avión en Alemania, la imagen de cara neutra pasará a ser la imagen de cara feliz durante 3 segundos, indicando que el objeto se ha colocado bien, y además se reproducirá un sonido de acierto. Pasados esos 3 segundos, la cara volverá a ser neutra.
- Si se coloca el barco en cualquier país que no sea Grecia, el tren en cualquier país que no sea Rumanía o el avión en cualquier país que no sea Alemania, la imagen de cara neutra pasará a ser la imagen de cara triste durante 3 segundos, indicando que el objeto se ha colocado mal, y además se reproducirá un sonido de fallo. Pasados esos 3 segundos, la cara volverá a ser neutra.

En cuanto a los juguetes utilizados, se pueden ver en la figura 28.

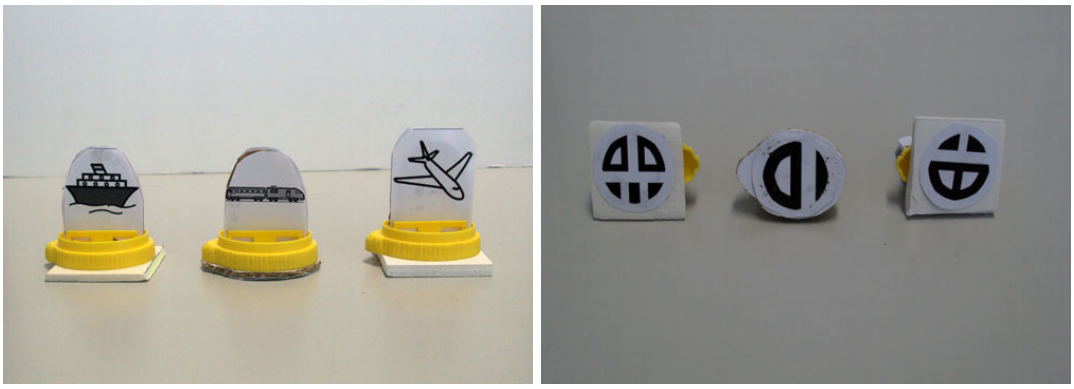


Figura 28: Juguetes de la actividad *Viajes*

5.3. Actividad *Tangram*

En la figura 29 se muestra un ejemplo de figura en el modo fácil y en el modo difícil, junto con las áreas de interacción en el modo fácil.

Para esta actividad, los objetos correctos son las 7 figuras del Tangram situadas en el sitio indicado con la orientación indicada.

En cuanto al feedback, se usarán las mismas imágenes utilizadas en las actividades *Lista de la compra* y *Viajes*, sólo que ahora la forma de activarse será la siguiente:

- Cuando se coloque una figura en su sitio correcto y con la orientación adecuada, la imagen de cara neutra pasará a ser la imagen de cara feliz durante 3 segundos, indicando que el objeto se ha colocado bien, y además se reproducirá un sonido de acierto. Pasados esos 3 segundos, la cara volverá a ser neutra.
- Cada vez que se coloque una figura donde no se corresponde, la imagen de cara neutra pasará a ser la imagen de cara triste durante 3 segundos, indicando que el objeto se ha colocado mal, y además se reproducirá un sonido de fallo. Pasados esos 3 segundos, la cara volverá a ser neutra.

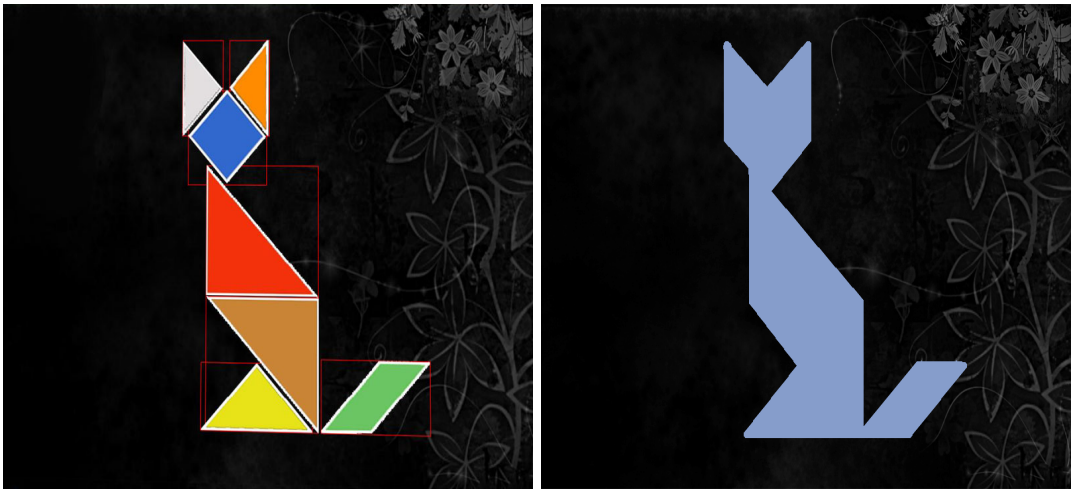


Figura 29: A la izquierda: imagen de fondo de una figura de la actividad *Tangram* en modo fácil. A la derecha: imagen de fondo de la misma figura en modo difícil.

En cuanto a los juguetes utilizados, se pueden ver en la figura 30.

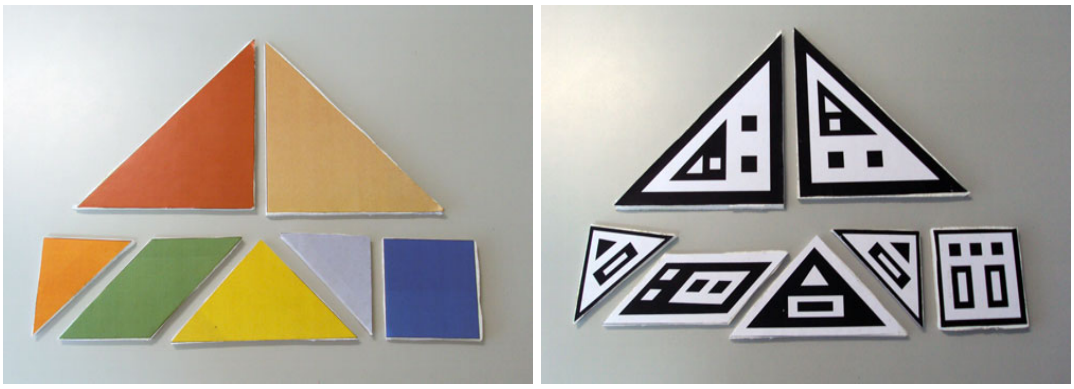


Figura 30: Juguetes de la actividad *Tangram*

5.4. Actividad *Marca los símbolos*

En este caso sólo hay una imagen, el mapa, mostrado en la figura 31, y las áreas de interacción son los símbolos que aparecen en el mapa, como se muestra en la figura.

Para esta actividad, los objetos correctos son las fichas colocadas en los símbolos que se indiquen en la grabación inicial de la actividad. Por ejemplo, si la grabación de inicio de la la actividad dice: *"Sitúe fichas en todos los símbolos de gasolineras"*, la actividad se completará cuando todos los símbolos de gasolinera tengan una ficha encima.

En cuanto al feedback, en este caso además de las imágenes mostradas en la figura 32 también se usará sonido sonido, de modo que:

- Cuando se coloque una ficha en el símbolo indicado, se reproducirá un sonido de acierto indicando que se ha colocado bien y aparecerá la imagen de acierto sobre el símbolo.
- Cada vez que se coloque una ficha en un símbolo no indicado, se reproducirá un sonido de fallo indicando que se ha colocado mal y aparecerá la imagen de fallo sobre el símbolo.

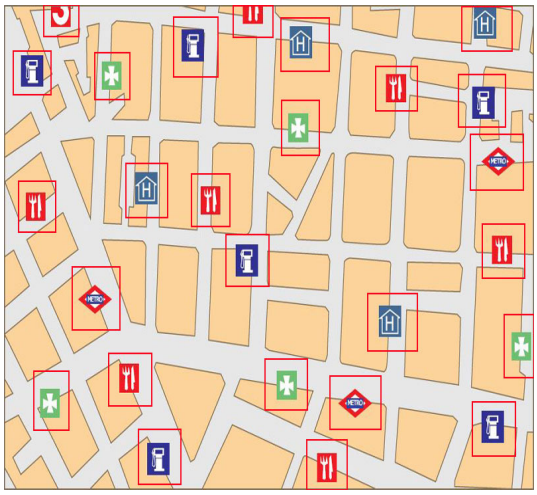


Figura 31: Imagen de fondo y áreas de interacción de la actividad *Marca los símbolos*.

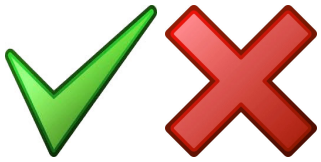


Figura 32: Imagen de acierto y de fallo para la actividad *Marca los símbolos*

En cuanto a los juguetes utilizados, en este caso se usaron las fichas mostradas en la figura 33.

Figura 33: Fichas que se usan en lugar de juguetes con fiduciales

5.5. Actividad *¿Cuántos hay?*

En este caso sólo hay una imagen, el conjunto de números, mostrado en la figura 34, y las áreas de interacción son cada uno de los números que aparecen, como se muestra en la figura.

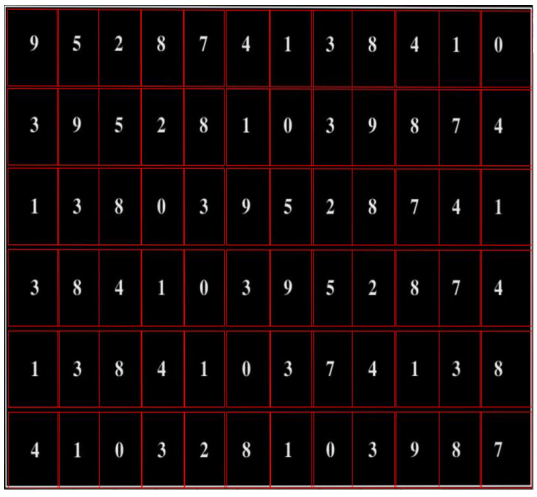


Figura 34: Imagen de fondo y áreas de interacción de la actividad *Marca los símbolos*.

En cuanto al feedback, en este caso se reutilizan las imágenes mostradas en la figura 32, de modo que cuando se sitúa una ficha sobre el número correcto, aparece el tick de acierto sobre el número, y cuando se coloca una ficha sobre un número incorrecto, aparece la X de error sobre el número.

En este caso no utilizamos sonido en el feedback ya que el objetivo de esta actividad era que además de que el usuario localizara los números correctos, estuviera pendiente de la secuencia de golpes que iba sonando mientras completaba la actividad, con lo cual si poníamos un sonido de acierto o de fallo, interfería con la secuencia dificultando la tarea.

En cuanto a los juguetes utilizados, en este caso se usaron las fichas mostradas anteriormente en la figura 33.

5.6. Actividad *Completa la secuencia*

En este caso sólo hay una imagen, la secuencia, mostrado en la figura 35, y el áreas de interacción es la ficha que el usuario ha de poner para completar la secuencia, como se muestra en la figura.

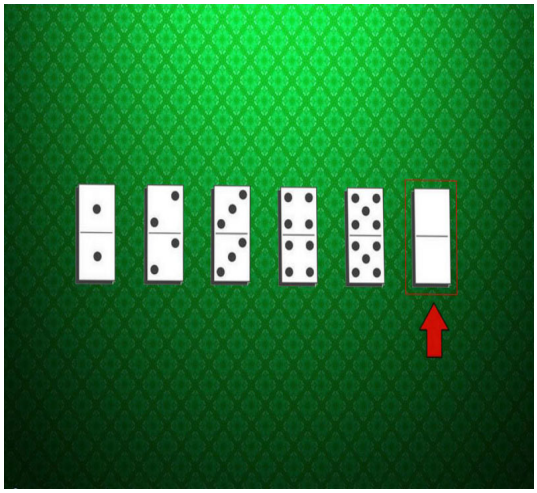


Figura 35: Imagen de fondo y área de interacción de la actividad *Completa la secuencia*.

Para esta actividad, el objeto correcto es la ficha de dominó que completa la secuencia (en el caso de la secuencia mostrada en la figura 35, la ficha que falta es la 6/6).

En cuanto al feedback, se usarán las mismas imágenes utilizadas en las actividades *Lista de la compra*, *Viajes* y *Tangram* sólo que ahora la forma de activarse será la siguiente:

- Cuando se coloque una ficha de dominó correcta, la imagen de cara neutra pasará a ser la imagen de cara feliz durante 3 segundos, indicando que el objeto se ha colocado bien, y además se reproducirá un sonido de acierto. Pasados esos 3 segundos, la cara volverá a ser neutra.
- Cada vez que se coloque una ficha de dominó incorrecta, la imagen de cara neutra pasará a ser la imagen de cara triste durante 3 segundos, indicando que el objeto se ha colocado mal, y además se reproducirá un sonido de fallo. Pasados esos 3 segundos, la cara volverá a ser neutra.

En cuanto a los juguetes utilizados, se pueden ver en la figura 36.

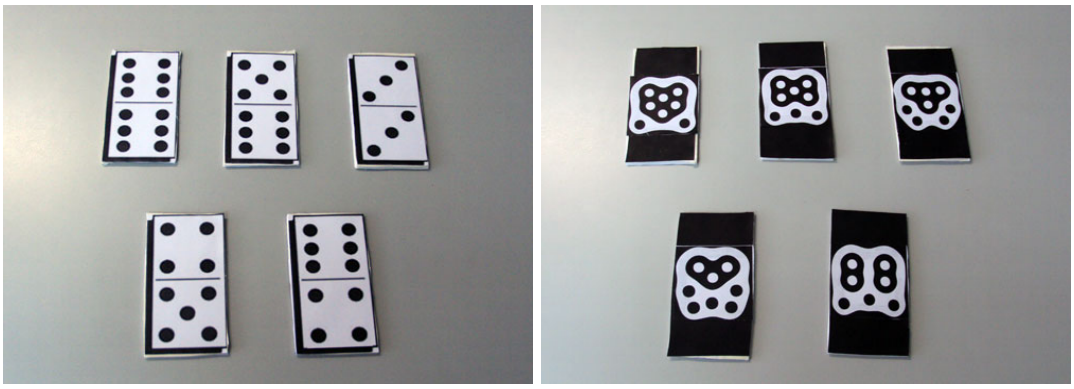


Figura 36: Juguetes de la actividad *Completa la secuencia*

5.7. Actividad *Analogías*

En este caso no hay ninguna imagen de fondo, debido a que las áreas de interacción van asociadas a cada una de las fichas de modo que el usuario pueda situar las fichas en cualquier parte del tabletop y así sólo preocuparse de hacer coincidir los extremos de las fichas con las correspondientes parejas.

En la figura 37 se puede ver una captura del juego junto con los 10 juguetes utilizados.

Figura 37: Juguetes de la actividad *Analogías*

6. Conclusiones y trabajo futuro

La meta de este proyecto era implementar una herramienta que permitiera desarrollar actividades enfocadas a personas de la tercera edad para el tabletop NIKVision, así como desarrollar un conjunto de actividades que demostraran la validez de la herramienta.

La implementación de la herramienta se ha llevado a cabo con éxito, permitiendo un diseño cómodo de actividades al permitir al desarrollador de la actividad la posibilidad de hacer cambios sobre la misma en tiempo de ejecución, lo cuál facilita bastante la realización de pruebas mientras se está desarrollando.

La herramienta además permite la ejecución de las mismas sin necesidad de ningún dispositivo externo como teclado o ratón, que era uno de nuestros objetivos principales al querer evitar dificultades a las personas mayores menos acostumbradas al uso de estas tecnologías.

Además, se ha podido realizar una evaluación de la herramienta que ha permitido extraer las siguientes conclusiones: primero, la herramienta tiene una buena usabilidad, como muestran los resultados de la evaluación en los que se obtiene que la usabilidad de la herramienta es superior al 74 sobre 100, y segundo, gracias a las actividades realizadas en las sesiones de evaluación, aunque la herramienta desarrollada es para soporte de personas mayores, por la evaluación se ve que puede servir para realizar actividades para cualquier ámbito, por ejemplo actividades meramente lúdicas.

En cuanto a trabajo futuro, uno de los objetivos sería implantar la herramienta en un ámbito sanitario para que pueda ser probada por personas mayores y así ver si su uso les permite mejorar sus capacidades cognitivas y evitar su deterioro, así como comprobar la utilidad de los ficheros de logs generados durante la realización de las actividades para analizar si la información almacenada en ellos es suficiente o si habría que anotar algún otro tipo de datos. Esto también permitiría confirmar que la herramienta tiene una buena usabilidad para cualquier persona que no tenga conocimientos previos de programación.

Por último, sería interesante incluir la posibilidad de programación en el diseño de actividades, lo cuál complicaría más el desarrollo de las mismas pero también permitiría a su vez realizar juegos más complejos. En este proyecto no fue necesario incluir dicha posibilidad porque las actividades que iban a desarrollarse no requerían tal grado de complejidad, pero podría ser interesante dar esa opción si la herramienta fuera a utilizarse en nuevos ámbitos y no sólo como soporte para personas de la tercera edad.

Anexo 1. Detalles de implementación

En este anexo se profundizará en la implementación de las clases *Cargar_imagen*, *Area*, *Fiducial*, *Feedback*, *TratarXML* e *Inicio*, las cuáles se comentaron brevemente en el apartado de Diseño e Implementación.

A1.1 Clase *Cargar_imagen*

El uso de la variable *mapa_colisiones* se verá en el apartado de implementación de la clase *Area*. En las figuras 38 y 39 se pueden ver fragmentos de código de las funciones de *Cargar_imagen*.

```
// Create the loader
loader = new Loader( );

if (nom.substr(nom.length-3, nom.length)=="swf") esFlash=true;

loader.contentLoaderInfo.addEventListener(Event.INIT, initListener);

// Load the bitmap
loader.load(new URLRequest(nombre));
```

Figura 38: Fragmento de código donde se distingue si la imagen es o no flash y se llama a la función de carga

```
// ------------------------------------------------------------------------
private function initListener (e:Event):void {
// ------------------------------------------------------------------------
//  Pre: El bitmap ha sido cargado e inicializado.
//  Post: Carga la imagen en pantalla con las dimensiones indicadas.
// ------------------------------------------------------------------------

    // Add the loaded bitmap to display list
    var dibujo:MovieClip=new MovieClip();

    // Se coge el ancho alto del XML, si no lo incluye, se saca de la imagen
    if (ancho<0) ancho=loader.content.width;
    if (alto<0) alto=loader.content.height;

    // Distinguimos si la animación es flash o no
    if (esFlash) dibujo = MovieClip(loader.contentLoaderInfo.content);
    else dibujo.addChild(loader.contentLoaderInfo.content);

    loader.content.width=ancho;
    loader.content.height=alto;

    layer.addChild(dibujo);
    dibujo.x=posX-ancho/2;
    dibujo.y=posY-alto/2;

    if (hayAlfa) mapa_colisiones = Bitmap(loader.content);
}
```

Figura 39: Función de *Cargar_imagen* que realiza la carga de la imagen sobre la capa

A1.2 Clase *Area*

En el apartado de diseño e implementación se comentó que un Área podía tener en cuenta el canal alfa de su imagen asociada.

El hecho de dar la opción de usar el canal alfa de una imagen png surgió en el proceso de evaluación de la herramienta (explicado en el segundo anexo), y surgió porque hay actividades como el *Tangram* en las que debido a las formas de los objetos utilizados, sería interesante poder

trabajar con áreas no rectangulares. Con el canal alfa se puede hacer eso puesto que nos podemos quedar con la parte opaca de la imagen y considerarla el área de interacción dejando fuera la parte no opaca. El uso de esto se verá más claramente en la clase *Inicio*.

A continuación, vamos a comentar algunas de las funciones más relevantes de la clase *Area*:

- **Area**: es la clase constructora. Además de inicializar sus atributos, carga (si las hay) las imágenes cuya información se le pasa como parámetro mediante un fichero XML tal y como se ve en el diagrama de clases de la herramienta.

  En la figura 40 se puede ver el fragmento de código en el que se realiza la inicialización de algunos de los atributos y la carga de la imagen neutra.

```
if (img!=null) p=directorio+img.@path;
if (imgBien!=null) pBien=directorio+imgBien.@path;
if (imgMal!=null) pMal=directorio+imgMal.@path;

IDasociado=fidasoc;

imgAncho=-1;
imgAlto=-1;
verbos=verbose;

orientable=orien;
hayAlfa=alfa;

// Imagen neutra
if (img!=null) {
    if (img.@ancho>0) imgAncho=int(img.@ancho);
    if (img.@alto>0) imgAlto=int(img.@alto);
    imX=cX;
    imY=cY;
    imX=imX+int(img.@x);
    imY=imY+int(img.@y);

    imgSpr=new MovieClip();
    capa.addChild(imgSpr);

    cargadorAlfa= new Cargar_imagen(imgSpr,p,imX,imY,imgAncho,imgAlto,hayAlfa);
    if (fidasoc!=null) {
        imgSpr.x=1000;
        imgSpr.y=1000;
    }
}
```

  Figura 40: Fragmento de código de la función constructora de la clase *Area*

  Se puede ver que en un inicio el ancho y el alto de la imagen se inicializan a -1 para después actualizarse en el caso de que el fichero XML de la imagen neutra tenga ancho y alto (*img.@ancho* y *img.@alto*). Esto se hace porque si el *Cargar_imagen* ve que el ancho o el alto es -1, interpreta que hay que tomar el ancho y alto que tiene la imagen real sin necesidad de escalarla.

  También se ve que las coordenadas (x,y) de la imagen se calculan sumando las coordenadas del área a las de la imagen, puesto que como se comentó ya, las coordenadas de las imágenes dependen son siempre respectivas al centro del área.

- **añadeFID**: añade una lista de fiduciales a la lista *IDs*. Ya comentamos en la implementación de la herramienta que un área tiene la opción de poder completarse de más de una manera; es decir, un área puede completarse cuando se colocan sobre ellas diferentes combinaciones de fiduciales.

Por ejemplo, en el caso del *Tangram* en la que hay objetos distintos pero iguales en forma y tamaño, un área puede considerarse correcta cuando se coloca sobre ella uno u otro, o en la actividad *Lista de la compra* si la lista dice: "Mete en la bolsa dos bebidas", el área se consideraría correcta si se pusiera sobre ella las combinaciones vino-zumo, vino-leche, zumo-leche.

Por lo tanto, esto se traduce en implementación en que necesitamos guardarnos en el atributo *IDs* una lista de lista de fiduciales. En la figura 41 se puede ver la función completa.

```
//-------------------------------------------------------------------------------------------
public function añadeFID(texto:XML,lista_ids:Array,fid_asoc:String, directorio:String):void {
//-------------------------------------------------------------------------------------------
    var lista_fid:Array = new Array();
    var indice:int=0;

    for (var i:int=0; i<lista_ids.length; i++) {
        // Si hay fichas, añadimos tantas como se indique (todas con ID=999)
        if (lista_ids[i].indexOf("ficha")!=-1) {
            var fichas:Array = lista_ids[i].split("ficha");
            var numFichas=fichas[0];
            for (var j:int=0; j<numFichas; j++) {
                lista_fid[indice]=new Fiducial(texto,"999",fid_asoc,directorio);
                indice++;
            }
        }
        else {
            lista_fid[indice]=new Fiducial(texto,lista_ids[i],fid_asoc,directorio);
            indice++;
        }
    }

    IDs.push(lista_fid);
}
```

Figura 41: Función *añadeFID* de la clase *Area*

El hecho de distinguir entre fichas y no fichas se hace porque en el caso de las fichas, es necesario tener en cuenta cuántas fichas se necesitan para que el área se considere correcta, mientras que si se trata de un fiducial, al tener un id único nunca va a repetirse un id en un área. El hecho de elegir 999 como id de las fichas se hizo porque se analizó que en ninguna actividad se van a tener que usar más 998 objetos.

- **buscaCambios**: busca las diferencias entre el estado anterior del área y el actual. De esta función depende el funcionamiento del feedback, puesto que es la que comprueba qué nuevos fiduciales se han puesto sobre el área y si estos son correctos o no, para así además reproducir los sonidos correctos o incorrectos correspondientes. En la figura 42 se puede ver un fragmento de la función.

Lo primero que se hace es mirar si se ha producido algún cambio (la función *buscarAnterior* devolvería *true* en ese caso) y si ese es el caso, se inicializa el clip de la imagen por si se trata de un flash para ya a continuación tratar los cambios.

A continuación se mira si el fiducial que se ha cambiado (función *dameFiducial* que busca por la lista *IDs* para devolver el fiducial correspondiente al id que se la pasa como parámetro) es distinto de *null*, lo cuál significa que es uno de la lista de *IDs*, correcto o incorrecto.

La variable *yaBien* sólo vale *true* cuando el área ya ha sido correctamente completada y la variable *sonido* empieza inicializada a *true*, de modo que si el área está bien y *sonido* de momento no vale *false*, significa que se acaba de colocar el último objeto correcto en el área y por tanto, si lo hay, hay que reproducir el sonido del fiducial.

En el caso de que el área todavía no se haya completado, se mira si el fiducial es correcto o incorrecto, se actualiza el atributo *estaBien* para posteriormente cambiar el feedback y se reproduce el sonido si es que hay.

En el caso de que el fiducial cambiado fuera *null*, significa que se ha colocado sobre el área un fiducial que no está en la lista y por lo tanto es un fiducial incorrecto, por lo tanto se actualiza la variable *estaBien* a -1 y se reproduce el sonido si lo hay. La primera comprobación de si se ha colocado un "ficha" se hace por si accidentalmente se ha colocado un dedo sobre el área (los dedos cuentan como fichas porque su tamaño es muy similar) para que el feedback no indique que ha habido fallo.

```
if (buscarAnterior(IDsColocados[i])) {
    var fiducialCambiado:Fiducial;
    fiducialCambiado=dameFiducial(IDsColocados[i]);
    if (fiducialCambiado!=null) {
        if (imgBienSpr!=null && imgBienSpr.visible) {
            var miObjeto:DisplayObject = imgBienSpr.getChildAt(0);
            var miClip:MovieClip = miObjeto as MovieClip;
            if (miClip.currentFrame<miClip.totalFrames) miClip.play();
        }
        // El sonido correcto sonará sólo cuando el área esté del todo bien
        if (yaBien) {
            if (sonido) {
                estaBien=1;
                if (fiducialCambiado.audio!=null) fiducialCambiado.audio.play();
                sonido=false;
            }
        }
        // El sonido incorrecto sonará siempre
        else {
            if (fiducialCambiado.correcto) estaBien=1;
            else {
                estaBien=-1;
                if (fiducialCambiado.audio!=null) fiducialCambiado.audio.play();
            }
        }
    } // fiducialCambiado != null
    else {
        if (IDsColocados[i]==999 && !tieneFichas)  {}
        else {
            if (audioIncorrecto!=null) {
                estaBien=-1;
                audioIncorrecto.play();
            }
        }
    }
```

Figura 42: Fragmento de la función *buscaCambios* de la clase *Area*

- **correctitud**: devuelve el estado del área (-1 incorrecta, 0 neutra, 1 correcta, 2 hay algo mal). Para ello, se hacen las siguientes comprobaciones:
  - Comprobar que todos los fiduciales que han de estar colocados lo están. Para ello va recorriendo todas las listas de fiduciales y en cuanto una es correcta, se guarda esa información es una variable. También se comprueba que en el caso de que el área tenga fichas en alguna de sus sublistas de fiduciales, el número de fichas colocado sea correcto.
  - Comprobar que no haya ningún fiducial puesto que sea incorrecto. En el caso de que el área no tenga fichas en ninguna de sus sublistas de fiduciales, no se tiene en cuenta para que si se coloca un dedo sobre el área, no salte error.
  - Comprobar el estado de las comprobaciones anteriores y en función de eso decir si el área es:
    - incorrecta (-1): faltan objetos correctos por poner y alguno de los puestos es incorrecto.

- neutra (0): falta algún objeto correcto por poner pero no hay ninguno incorrecto puesto.
- correcta (1): están todos los objetos correctos puestos y no hay ninguno incorrecto puesto.
- con algo mal (2): están todos los objetos correctos puestos pero hay alguno incorrecto puesto.

Según el valor del estado se ponen visibles (si las hay) las imagen que corresponde entre la imagen neutra, bien y mal asociadas al área si el área no depende de ningún fiducial o en caso de que dependa si dicho fiducial se encuentra sobre la mesa, para que en caso de quitar el fiducial la imagen no se quede en pantalla. Por último, se devuelve el valor del estado.

En las figuras 43 y 44 se puede ver el detalle del código de las comprobaciones.

```
// Primero comprobamos si esta todo correcto
for each (var lista_fid:Array in IDs) {
    completada=true; algunCorrecto=false; numF=0;
    for each (var fidRequerido:Fiducial in lista_fid) {
        // Comprobamos los fiduciales que no son fichas
        if (!fidRequerido.esFicha) {
            if (fidRequerido.correcto==true) {
                algunCorrecto=true;
                //Comprobamos si esta puesto
                estaPuesto=false;
                for each (var fidPuesto:int in IDsColocados) {
                    if (fidPuesto==fidRequerido.ID) estaPuesto=true;
                }
                if (!estaPuesto) completada=false;
            }
            else numIncorrectos++;
        }
        // Pasamos a contar las fichas necesarias
        else {
            if (fidRequerido.correcto==true) {
                algunCorrecto=true;
                numF++;
            }
        }
    } // segundo for
    /* Si hay fichas, comprobamos que el numero de fichas colocadas
    es correcto */
    if (tieneFichas) {
        if (numF!=numC) completada=false;
    }
    /* En cuanto una de las sublistas es correcta, el area
    se considera correcta */
    if (!algunCorrecto) completada=false;
    completado_OR = completado_OR || completada;
}
```

Figura 43: Fragmento de la función *correctitud* de la clase *Area* en la que se comprueba si todos los fiduciales puestos sobre el área son correctos

```
// Despues comprobamos si alguno de los puestos es incorrecto
var correcto_OR:Boolean=false;
var correcta,esta:Boolean;
if (incorrecto) {
    for each (var lista_fid:Array in IDs) {
        correcta=true;
        for each (var fidPuesto:int in IDsColocados) {
            esta=false;
            for each (var fidRequerido:Fiducial in lista_fid) {
                if ((fidPuesto==fidRequerido.ID) && (fidRequerido.correcto)) esta=true;
                // Si el area no tiene en cuenta fichas, las obviamos si estan puestas
                if (!tieneFichas && fidPuesto==999) esta=true;
            }
            correcta=correcta && esta;
        }
        /* En cuanto una de las sublistas es correcta, el area
        se considera correcta */
        correcto_OR = correcto_OR || correcta;
    }
}
else { correcto_OR=true; }

if (IDs.length==0) {
    if (IDsColocados.length==0) { correcto_OR=true; completado_OR=true; }
}

// Comprobamos estado del area

if (!correcto_OR && !completado_OR) estado=-1;
else if (completado_OR && correcto_OR) { estado=1; yaBien=true; }
else if (completado_OR && !correcto_OR) estado=2;
```

Figura 44: Fragmento de la función *correctitud* de la clase *Area* en la que se comprueba si no hay ningún fiducial incorrecto colocado sobre el área

- **dentro**: comprueba si un fiducial está dentro del área con la orientación correcta. Como un fiducial puede tener varias orientaciones, en cuanto una de ellas se cumple se considera que el fiducial está orientado correctamente. Una vez que está comprobado el giro, se comprueba que sus coordenadas están dentro del área y si lo están, la función devuelve *true*; en caso contrario devuelve *false*.

  En la figura 45 se puede ver el fragmento de código en el que se comprueba si la orientación es correcta.

```
for each (var lista_fid:Array in IDs) {
    for each (var fid:Fiducial in lista_fid) {
        // Localizamos el fiducial con el que hay que comparar la rotacion
        if (fid.ID==id) {
            if (fid.rotacion!=null) {
                // Comprobamos todas las posibles orientaciones
                no_girado=false;
                for(var i:int=0; i<fid.rotacion.length; i++) {
                    difGiro=Math.abs(fid.rotacion[i]-giro);
                    if (difGiro>180) difGiro=360-difGiro;
                    girado = girado || (difGiro<30);
                }
            }
        }
    } // sublistas de fid
} // lista de fid
```

Figura 45: Fragmento de la función *dentro* de la clase *Area* en la que se comprueban las orientaciones

- **actualizarRotacion**: actualiza la rotación de los fiduciales dependientes de otro fiducial con respecto a la rotación de este. Esta función se utiliza únicamente cuando estamos en un área orientable que depende de otro fiducial, ya que cuándo éste gire, el área girará con él y por tanto las rotaciones de los fiduciales tendrán que ir actualizándose.

A1.3 Clase *Fiducial*

La clase *Fiducial* se encarga de la creación de los fiduciales de cada área. Solo tiene una función constructora a la que se le pasa como parámetros:

- un fichero XML con la información relativa al fiducial.
- el id del fiducial.
- el fiducial asociado del área a la que pertenece (si el área no depende de ningún fiducial vale *null*).
- el directorio donde se encuentra, si lo hay, el archivo de sonido asociado al fiducial.

La función constructora se puede ver en la figura 46:

```
//--------------------------------------------------------------------------------
public function Fiducial(texto:XML,fid:String, asoc:String,directorio:String) {
//--------------------------------------------------------------------------------
// Codigo constructor
//--------------------------------------------------------------------------------

     ID=int(fid);

     if (ID==999) esFicha=true;

     if (texto.@correcto=="si") correcto=true;
     else correcto=false;

     if (String(texto.@sonido).length>0) {
        audio=new Sound(new URLRequest(directorio+texto.@sonido));
     }

     if (String(texto.@orientable).length>0) {
         var lista_rot:Array = texto.@orientable.split(",");
         rotacion = new Array();
         rotacion_fija = new Array();
         for(var i:int=0; i<lista_rot.length; i++) {
             rotacion[i]=(int)(lista_rot[i]);
             rotacion_fija[i]=(int)(lista_rot[i]);
         }
     }

     if (asoc!=null) asociado=asoc;
}
```

Figura 46: Función constructora de la clase *Fiducial*

Nos guardamos dos listas de rotaciones porque una es la lista fija con el giro siempre respecto al ángulo 0 (variable *rotacion_fija*) y la otra lista (variable *rotacion*) es la que se irá actualizando con respecto al fiducial asociado si es que éste existe.

A1.4 Clase *Feedback*

La clase *Feedback* se encarga de la creación de los Feedback de cada área. A parte de la función constructora en la que solo se realiza la inicialización de sus atributos, el resto de funciones se encargan de hacer la carga de las imágenes y de sonidos si es que los hay y de mostrarlas y reproducirlos respectivamente.

En las figuras 47, 48 y 49 se pueden ejemplos de las funciones características de esta clase.

```
//----------------------------------------------------------------------
public function CargarAcierto():void {
//----------------------------------------------------------------------
// Carga la imagen de feedback de acierto.
//----------------------------------------------------------------------

    if (acierto!=null) {
        imgAcierto=new MovieClip();
        var cargador= new Cargar_imagen(imgAcierto,acierto,x,y,W,H,false);
        C.addChild(imgAcierto);
    }
}
```

Figura 47: Ejemplo de función de carga de imagen de la clase *Feedback*

```
    //-------------------------------------------------------
    public function MostrarAcierto():void {
    //-------------------------------------------------------
    // Muestra la imagen de feedback de acierto.
    //-------------------------------------------------------

        ocultarFeedback();
        if (imgAcierto!=null) imgAcierto.visible=true;

    }
```

Figura 48: Ejemplo de función de mostrar imagen de la clase *Feedback*

```
//----------------------------------------------------------------------
public function SetAudioCorrecto(ruta:String):void {
//----------------------------------------------------------------------
//  Pone el audio asociado al feedback con el sonido de acierto.
//----------------------------------------------------------------------

  if (ruta.length>0) {
    audioCorrecto=new Sound(new URLRequest(ruta));
  }

}
```

Figura 49: Ejemplo de función de cargar sonido de la clase *Feedback*

A1.5 Clase *TratarXML*

La clase *TratarXML* se encarga del tratamiento de información de la actividad junto con la creación del menú y la inicialización del fichero de arranque.

A continuación, vamos a comentar algunas de sus funciones más relevantes:

- **TratarXML**: es la función constructora, y como parámetros sólo recibe las capas (menú, fondo, área y feedback) que serán usadas en la clase *Inicio*. Además de la inicialización de sus 4 atributos, se encarga de la creación del menú, de modo que lo primero que hace es comprobar si el fichero de arranque existe: si existe, procede a cargar el fichero XML indicado en él; si no existe, procede a la creación del menú.

  Se presupone que el fichero de configuración está en el mismo directorio desde el cuál se ejecuta la herramienta y que los ficheros XML de los juegos se encuentran en una carpeta llamada "juegos" en el escritorio.

En las figuras 50 y 51 se pueden ver los fragmentos de código en los que se trata el fichero de arranque y la creación del menú.

```
// La carpeta de juegos siempre estará en el escritorio
directorio= File.desktopDirectory.nativePath + "\\juegos";

// Compruebo si existe el fichero de configuracion
var config:String = File.applicationDirectory.nativePath + "\\config.xml";
var fileConf:File = new File(config);
if (fileConf.exists) {
    // Si existe, se lee el juego que se va a usar
    loadXML(config,fileConf.name);
}
```

Figura 50: Fragmento de código de la función constructora de la clase *TratarXML* donde se carga el fichero de arranque

```
else {
    // Si no existe, mostramos el menú
    hayMenu=true;
    var carpetaJuegos:File = new File(directorio);
    var lista:Array = carpetaJuegos.getDirectoryListing();

    listaXML = new Array();
    var nom:String;
    // Filtramos la carpeta juegos para quedarnos solo con los XML
    lista.forEach(
        function (fichero:*, index:int, array:Array){
            nom = fichero.name;
            if (!fichero.isDirectory && nom.substr(nom.length-3, nom.length)=="xml"){
                listaXML.push(fichero);
            }
        }
    )

    // Calculamos el ancho que tendran los iconos
    totalJuegos=listaXML.length;
    if (totalJuegos<4) {
        anchoIc=(800-((totalJuegos-1)*40))/totalJuegos;
        xIc=anchoIc/2;
    }

    listaIconos=new Array(listaXML.length);

    // Cargamos los iconos y los ponemos por pantalla
    listaXML.forEach(
        function (fichero:*, index:int, array:Array){
            var loader= new URLLoader(new URLRequest(fichero.nativePath));
            loader.addEventListener(Event.COMPLETE, loadedIcon);
        }
    ) // forEach
}
```

Figura 51: Fragmento de código de la función constructora de la clase *TratarXML* donde se crea el menú

- **loadedIcon**: función encargada de la creación de áreas de los iconos y de su colocación en el menú. Para seleccionar el icono que representará la actividad, hay dos opciones:
  - Usar el atributo <icono> especificado en el XML de la actividad.
  - Si el anterior atributo no existe, usar el fondo de la actividad como icono.

Los iconos se colocan en forma de una matriz de 3x4 indicándose debajo de cada icono el nombre del juego. En la figura 52 se puede ver el fragmento de código de la función en la que se realiza la creación del menú.

```
// Ajustamos tamaño del icono y creamos el área
iconoG.@ancho=anchoIc; iconoG.@alto=150;
listaIconos[numIc]=new Area(null,null,xIc, yIc, anchoIc, 150, iconoG, null, null, directorio, layerMenu, true,false);
if (numIc>=11) listaIconos[numIc].imgSpr.visible=false;

// Añadimos el texto del icono
var tf:TextField = new TextField();
tf.autoSize = TextFieldAutoSize.LEFT;
tf.selectable = false;
tf.defaultTextFormat = new TextFormat("Arial", null, 0x999999);
tf.appendText("(" + (numIc+1) + ") " + listaXML[numIc].name.substring(0,listaXML[numIc].name.length-4));
tf.x = xIc-anchoIc/2;
tf.y = yIc+75;

layerMenu.addChild(tf);
listaIconos[numIc].textoIcono=tf;
if (numIc>=11)  listaIconos[numIc].textoIcono.visible=false;

// Actualizamos nuevas posiciones de los iconos
if (cont>=4) { yIc+=190; cont=0; xIc=100; }
else xIc+=(anchoIc+40);
numIc++; cont++;

// Cargamos los siguientes iconos
if ((numIc%11)==0 && totalJuegos>11) {
    path=<imagen/>;
    path.@path="siguiente.png";
    path.@ancho=anchoIc;
    path.@alto=150;
    // Area del botón de navegación de menú
    siguiente=new Area(null,null,xIc, yIc, anchoIc, 150, path, null, null, directorio, layerMenu, true,false);
    yIc=100; xIc=100; cont=1;
}

// Indicamos si están cargados todos los iconos
if (numIc==totalJuegos) menuCompleto=true
```

Figura 52: Fragmento de código de la función *loadedIcon* de la clase *TratarXML* donde se crean los iconos y se dibuja el menú

- La clase *TratarXML* utiliza las funciones *TratarFondo*, *TratarArea* y *TratarFeedback* para realizar el tratamiento de la información de la actividad.

  Como el comportamiento de estas funciones es muy similar (pues todas se encargan de ir leyendo el fichero y de ir creando los objetos correspondientes), en la figura 53 se muestra un fragmento de la función *TratarArea* para que sirva de ejemplo para el resto de funciones.

  Se puede ver que a la función *Area* se le pasan los parámetros previamente extraídos del fichero para posteriormente realizar el tratamiento de fiduciales del área, añadiéndose a esta con la función *añadeFID*. También se actualizan algunos de los atributos de área como *incorrecto* y *tieneFichas*.

```
for each (var img:XML in f.elements(name="imagen")) imagen=img;
for each (var img:XML in f.elements(name="imagenBien")) imagenBien=img;
for each (var img:XML in f.elements(name="imagenMal"))  imagenMal=img;

for each (var fid:XML in f.elements(name="fidasoc")) {
    fid_asoc=fid.@id;
    if (fid.@orientable=="si")  orientable=true;
}

listaAreas[numAreas]=new Area(fid_asoc,orientable,area_x, area_y, area_ancho, area_alto, imagen, ...

// Tratamos los fiduciales
var hayF:Boolean=false;

for each (var fid:XML in f.elements(name="fid")) {
    if (fid.@id!="*") { //es un fiducial concreto
        // Sacamos la lista de fiduciales
        var lista_fid:Array = fid.@id.split(",");
        for (var i:int=0; i<lista_fid.length; i++) {
            if (lista_fid[i].indexOf("ficha")!=-1) hayF=true;
        }
        listaAreas[numAreas].añadeFID(fid,lista_fid,fid_asoc,directorio);
    } else { //es el resto de fiduciales
        if (String(fid.@sonido).length>0) {
            listaAreas[numAreas].SetAudioIncorrecto(directorio+fid.@sonido);
        }
    }
    // Indicamos si algun fiducial es incorrecto
    if (String(fid.@correcto).length>0) {
        if (fid.@correcto=="no") listaAreas[numAreas].incorrecto=true;
    }
}

listaAreas[numAreas].tieneFichas=hayF;
```

Figura 53: Fragmento de código de la función *TratarArea* de la clase *TratarXML* donde se realiza la creación de áreas y fiduciales

A1.6 Clase *Inicio*

La clase *Inicio* es la encargada de realizar la ejecución de la herramienta y de generar el fichero de resultados (o de logs) de las actividades con información relativa a estas.

A continuación, vamos a comentar algunas de sus funciones más relevantes:

- **juego**: función que está continuamente ejecutándose y que comprueba el estado de juego en cada momento, haciendo la comprobación de si se ha seleccionado alguna actividad en el caso de que haya menú y llamando a su vez a la función *actualizaListaJuego* que es la encargada de comprobar el estado de las áreas de la actividad que se esté ejecutando en ese momento.

  En la figuras 54 y 55 se pueden ver fragmentos de código de la función.

  Se puede ver en la función que si no se ha seleccionado nada (si *noSeleccion* vale *true*) y es necesario cargar un juego (si *cargarJuego* vale *true*), se procede a limpiar la pantalla, a indicar que no es necesario ya cargar un juego (poner *cargarJuego* a *false*) y a crear el objeto *TratarXML* para inmediatamente después llamar a la función *actualizaListaJuego*.

  Una vez que la tarea de la actividad termina, se comprueba si hay más tareas que tratar y en caso de no haberlas, indicamos que se va a volver al menú o a resetear la actividad ejecutada en el caso de que no haya menú y se escribe en el fichero de logs los resultados. La actualización de las variables *bien* y *mal* utilizadas en el código se realiza en la función *actualizaListaJuego*.

```
if (finalJuego==0) {
    // Miramos si hay que crear de nuevo el Menu
    if (noSeleccion && cargarJuego) {
        if (fXML!=null) fXML.limpiarCapa();
        limpiarPantalla();
        borrarListaFiduciales();
        fXML= new TratarXML(layerMenu,layerFondo, layerArea, layerFeedback);
        cargarJuego=false;
        if(!fXML.hayMenu) noSeleccion=false;
        primera=true;
    }
    // Actualiza la lista de juego
    actualizaListaJuego(ListaTuioObject);
}
```

Figura 54: Fragmento de código de la función *juego* de la clase *Inicio* donde se comprueba si se ha seleccionado alguna actividad

```
// Siguiente tarea
if (finalJuego>0 && sonido_acabado) {

    // En primer lugar borramos de pantalla
    if (fXML.hayFeedback) fXML.listaFeedback.OcultarCompletado();

    fXML.limpiarCapa();
    fXML.ocultarMenu();

    if (fXML.TratarSiguienteTarea()>0) {
        finalJuego=0;
        sonido_acabado=false;
    }
    else {
        // El juego terminó
        if (fXML.hayFeedback) fXML.listaFeedback.MostrarCompletado();
        // finalJuego==0 para volver al inicio
        finalJuego=0;
        sonido_acabado=false;
        noSeleccion=true;

        // Apuntamos resultados en el fichero de logs
        fin=getTimer();
        crearPartida(fecha,"si");
        cerrarXML();

        borrarListaFiduciales();
    }
}
```

Figura 55: Fragmento de código de la función *juego* de la clase *Inicio* donde se trata el final de la tarea y se escribe en el fichero de logs

- **actualizaListaJuego**: función encargada de la comprobación de las áreas de la tarea que se está ejecutando. Lo primero que hace esta función es ver si todavía se está en el menú o si ya se ha seleccionado un juego (figura 56).

  Una vez cargado el juego, se crea el fichero de logs en caso de que no exista guardándonos la información sobre la fecha y la hora de inicio de la partida. A continuación, se procede a dibujar las áreas (en el caso de que la variable verbose esté a *true*) y a recorrer, para cada área, toda la lista de objetos situados sobre la mesa (variable *ListaTuioObject* que se le pasa como parámetro a la función). Para cada objeto se comprueba:

  - si el área actual tiene un fiducial asociado, de modo que si el área es orientable se actualice, además de la posición, la rotación del área y la del objeto en cuestión en función del fiducial asociado.

```
if (fXML.menuCompleto) {
    //Dibujamos los iconos
    for (var i:int=0; i<fXML.listaIconos.length; i++) {
        ListaTuioObject.forEach(
            function (item:*, index:int, array:Array) {
                if (fXML.listaIconos[i].contenido(item.x,item.y) && item.ID!=18) {
                    // Indicamos que ya hemos seleccionado, ocultamos el menu y
                    // cargamos el juego seleccionado
                    noSeleccion=false;
                    fXML.ocultarMenu();
                    fXML.loadXML(fXML.listaXML[i].nativePath,fXML.listaXML[i].name);
                    nomJuego=fXML.listaXML[i].name;
                }
            }
        ) // forEach
    }
} // menuCompleto
```

Figura 56: Fragmento de código de la función *actualizaListaJuego* de la clase *Inicio* donde se comprueba selección de actividad desde el menú

- si el objeto en cuestión está dentro del área con la rotación adecuada (función *dentro* de *Area*) teniendo en cuenta además si el área tiene en cuenta el alfa de la imagen o no, para comprobar en caso de que tenga en cuenta el alfa si el objeto está en una zona opaca de la imagen. Si se cumplen las condiciones, se añade el objeto a la lista de ids del área (función *ColocaID* de *Area*) (figura 57).

```
// Comprobamos si el juguete esta dentro
if (!fXML.listaAreas[i].hayAlfa) {
    if (fXML.listaAreas[i].dentro(item.x,item.y,item.rotation,item.id)) {
        // Añadimos el juguete a la lista de colocados del area i-esima.
        fXML.listaAreas[i].ColocaID(item.ID);
    }
}
else {
    if (fXML.listaAreas[i].cargadorAlfa.mapa_colisiones.bitmapData!=null) {
        var alpha:String = (fXML.listaAreas[i].cargadorAlfa.mapa_colisiones.bitmapData.getPixel32(item.x,item.y) >> 24 & 0xFF)
        if (alpha!="0" && fXML.listaAreas[i].dentro(item.x,item.y,item.rotation,item.id)) {
            fXML.listaAreas[i].ColocaID(item.ID);
        }
    }
}
```

Figura 57: Fragmento de código de la función *actualizaListaJuego* de la clase *Inicio* donde se comprueba si el objeto está dentro del área

Tras comprobar cada objeto colocado, se mira el estado del área (función *correctitud* de *Area*), y depende de lo que devuelva, actualizamos el feedback si es que lo hay y el estado de juego (figura 58).

Por último, si tarea se ha completado, mostramos la imagen de completado de feedback si es que la hay e indicamos que la tarea se ha completado para que la función *juego* cargue la siguiente tarea o vuelva al menú.

```
// Calculamos el estado del area
var estadoArea:int; // -1: incorrecto . 0: neutro . 1: correcto
estadoArea=fXML.listaAreas[i].correctitud();

// Buscar cambios con la lista anterior
fXML.listaAreas[i].buscaCambios();
// Comprobamos el estado del fiducial que se ha colocado
if (fXML.listaAreas[i].estaBien==1) {
    if (fXML.hayFeedback==true) {
        fXML.listaFeedback.MostrarAcierto();
        bucle = setTimeout(volverANeutro, tiempo*1000);
        if (fXML.listaFeedback.audioCorrecto!=null) fXML.listaFeedback.audioCorrecto.play();
        bien++;
    }
}
else if (fXML.listaAreas[i].estaBien==-1) {
    if (fXML.hayFeedback==true) {
        fXML.listaFeedback.MostrarFallo();
        bucle = setTimeout(volverANeutro, tiempo*1000);
        if (fXML.listaFeedback.audioIncorrecto!=null) fXML.listaFeedback.audioIncorrecto.play();
        mal++;
    }
}
```

Figura 58: Fragmento de código de la función *actualizaListaJuego* de la clase *Inicio* donde se comprueba el estado de área y se actualiza el feedback

- **crearPartida**: función que guarda la información de la última partida ejecutada en un XML interno que posteriormente la función *cerrarXML* volcará en el fichero de logs (figura 59).

```
// Variables XML que componen la partida
var partida_FECHA:XML;
var partida_TIEMPO:XML;
var partida_COMPLETITUD:XML;

var stringDia = fecha.date + "/" + (fecha.month+1) + "/" + fecha.fullYear;
var stringHora:String;
if (fecha.hours<10) stringHora = "0" + fecha.hours + ":";
else  stringHora = fecha.hours + ":";
if (fecha.minutes<10) stringHora = stringHora + "0" + fecha.minutes + ":";
else  stringHora = stringHora + fecha.minutes + ":";
if (fecha.seconds<10) stringHora = stringHora + "0" + fecha.seconds;
else  stringHora = stringHora + fecha.seconds;

// Añadimos la linea <FECHA ... />
partida_FECHA=<fecha/>;
partida_FECHA.@dia=stringDia;
partida_FECHA.@hora=stringHora
partida.newElement=partida_FECHA;

var total:int = (fin-inicio)/1000;

// Añadimos la linea <TIEMPO_JUGADO ... />
partida_TIEMPO=<tiempo_jugado/>;
partida_TIEMPO.@segundos=total;
partida.newElement=partida_TIEMPO;

// Añadimos la linea <COMPLETITUD ... />
partida_COMPLETITUD=<completitud/>;
partida_COMPLETITUD.@acciones_correctas=bien;
partida_COMPLETITUD.@acciones_incorrectas=mal;
partida_COMPLETITUD.@acabado=completado;
partida.newElement=partida_COMPLETITUD;
```

Figura 59: Fragmento de código de la función *crearPartida* de la clase *Inicio* donde se crea el XML interno que se volcará en el fichero de logs

A1.7 Fichero de definición de actividades

El elemento raíz del XML es <*juego*>, el cuál tiene 2 tipos de elementos:

- <**verbose**>: cuyo valor puede ser "sí" (lo cuál indica que las áreas se dibujan en la pantalla como rectángulos rojos) o "no" (las áreas no aparecen dibujadas). La opción de dibujarlas o no se hizo para facilitar el desarrollo de actividades.
- <**tarea**>: un juego puede tener una o varias tareas, y cada una de ellas tiene sus propios elementos. De este modo, se pueden hacer diversas variantes de un mismo juego sin tener que hacer ficheros XML diferentes.

Cada elemento <*tarea*> tiene a su vez 3 tipos de elementos:

- <**fondo**>: contiene la información necesaria para establecer los elementos de fondo de la actividad, tanto imágenes como sonidos. Un área sólo puede tener un elemento *fondo*. Los elementos de este son:
  - <**color**>: indica el color de fondo de la actividad en el atributo *rgb* en caso de que no haya imagen de fondo.
  - <**icono**>: indica el icono representativo de la actividad con el atributo *path*. Su uso se explicará en el apartado de implementación de la herramienta.
  - <**imagen**>: indica la primera o única imagen de fondo de la actividad con el atributo *path*. Además, contiene los atributos:
    - **cambiar**: cuyo valor puede ser "sí", lo cuál indica que *imagen* cambia a *imagen2* a los *tiempo* segundos de haber empezado la actividad, o "no", la imagen de fondo de la actividad no cambia.
    - **tiempo**: indica los segundos que han de pasar desde el comienzo de la actividad para cambiar de *imagen* a *imagen2*.
  - <**imagen2**>: indica la segunda imagen de fondo de la actividad con el atributo *path*.
  - <**sonido**>: indica el sonido de fondo de la actividad con el atributo *path*. A su vez, contiene los atributos:
    - **repetir**: cuyo valor puede ser "sí", indicando que el fichero de sonido *path* se reproduce cada *tiempo* segundos, o "no", indicando que el fichero de sonido *path* sólo se reproduce una vez al comienzo de la actividad.
    - **tiempo**: indica cada cuántos segundos se reproduce el fichero de sonido *path*.
- <**area**>: contiene la información necesaria para la creación de un área de la actividad. Sus elementos son:
  - **posicion**: indica la posición del área en pantalla, indicando que el centro del área está en el punto (x, y) de la pantalla y sus dimensiones son los atributos *ancho* y *alto*. El atributo *alfa* indica si se ha de tener en cuenta el canal alfa de la imagen. Su uso se explicará en el apartado de implementación de la herramienta.
  - **imagen**: indica la imagen neutra asociada al área con el atributo *path*, indicando sus coordenadas y dimensiones de la misma manera que en *posicion*. Hay que tener en cuenta a la hora de trabajar con x e y que estos son relativos al centro del área, de modo que la posición real de la imagen es aquella cuyo centro está en ($posicion.x+imagen.x$, $posicion.y+imagen.y$).
  - **imagenBien**: indica la imagen asociada al área con el atributo *path* que aparece cuando ésta es completada correctamente. Sus atributos son exactamente los mismos que los de *imagen*.
  - **imagenMal**: indica la imagen asociada al área con el atributo *path* que aparece cuando sobre esta se coloca un objeto incorrecto. Sus atributos son exactamente los mismos que los de *imagen*.
  - **fidasoc**: objeto del que depende el área. Sus atributos son:

- **id**: indica el identificador del fiducial del que depende el área.
- **orientable**: cuyo valor puede ser "sí", indicando que en el caso de que el fiducial asociado gire sobre la mesa, el área ha de girar con él, o "no", indicando que el área no gira cuando gire el fiducial asociado, sino que sólo actualiza su posición cuando este se mueva.

- **fid**: contiene la información necesaria sobre los objetos asociados al área. Un mismo área puede tener 0 o más elementos *fid*. Sus atributos son:
  - **id**: indica los ids de los fiduciales asociados al área. El formato a seguir son los números de los ids entre comas en caso de tratar con objetos o el número de fichas que hay que colocar sobre el área, con el formato "[num]fichas", donde "[num]" es un número mayor que 0. Si se coloca un "*" en id significa "cualquier fiducial".
  - **orientable**: indica las orientaciones que ha de tener los fiduciales para ser considerados correctos o incorrectos. El formato a seguir son los diversos grados de orientación entre comas. Es un atributo opcional, en caso de no aparecer, significa que el objeto se considera correcto o incorrecto independientemente de la orientación que tenga.
  - **correcto**: cuyo valor puede ser "sí", indicando que los fiduciales se consideran correctos cuando se colocan sobre el área, o "no", indicando que los fiduciales se consideran incorrectos cuando se colocan sobre el área.
  - **sonido**: en el caso de que *correcto* valga "sí", indica el fichero de sonido que se reproducirá cuando todos los fiduciales de la lista *id* estén colocados sobre el área; en el caso de que *correcto* valga "no", indica el fichero de sonido que se reproducirá cada vez que se coloque un objeto de la lista *id* sobre ella. Es un atributo opcional.

Si aparece más de un elemento *fid* en un área, significa que el área puede ser completada correctamente de más de una manera: por ejemplo, si un área tiene n elementos *fid*, el área se considera correcta con tal de que se cumplan las condiciones de uno de los n elementos *fid*.

Hay que tener en cuenta también que de salvo el elemento *posicion*, el resto de elementos de *area* son opcionales, de modo que las áreas no tienen por qué tener un fiducial del que depender o imágenes asociadas.

- <**feedback**>: contiene la información necesaria para establecer los elementos de feedback de la actividad, tanto imágenes como sonidos. Un *juego* puede no tener el elemento *feedback*, pero si lo tiene ha de ser único. Los elementos de *feedback* son:

  - **pos**: indica la posición del feedback en pantalla, indicando que el centro del feedback está en el punto (x, y) de la pantalla y sus dimensiones son los atributos *ancho* y *alto*.
  - **imagen_acierto**: su atributo *path* indica la imagen que aparece cuando se coloca un objeto bien en un área.
  - **imagen_fallo**: su atributo *path* indica la imagen que aparece cuando se coloca un objeto mal en un área.
  - **imagen_neutro**: su atributo *path* indica la imagen que aparece por defecto mientras no se coloque ningún objeto en ningún área.
  - **imagen_completado**: su atributo *path* indica la imagen que aparece cuando se completa con éxito la tarea.
  - **sonido_acierto**: su atributo *path* indica el sonido que se reproduce cuando se coloca un objeto bien en un área.
  - **sonido_completado**: su atributo *path* indica el sonido que se reproduce cuando se coloca un objeto mal en un área.

Hay que tener en cuenta que salvo *pos*, todos los elementos de *feedback* son opcionales, de modo que pueden hacerse diversas combinaciones entre imágenes y sonidos para dar variedad a las actividades.

Una cosa a tener en cuenta también es que a la hora de definir imágenes en *area*, si lo que se busca es que la imagen se sitúe centrada en el área, no es necesario definir sus atributos x e y, puesto que la herramienta hará coincidir automáticamente los centros de la imagen y del área a la que pertenece.

Tampoco son necesarios los atributos *ancho* y *alto* si se quiere que la imagen mantenga sus dimensiones, estos solo son necesarios cuando se desea reescalar la imagen. De la misma manera, si no se especifica *ancho* y *alto* en el elemento *posicion* de *area*, esta toma por defecto el *ancho* y *alto* indicado en *imagen*.

Anexo 2. Detalles de los resultados de evaluación

En este segundo anexo se explicará en detalle los resultados obtenidos en las observaciones, en las preguntas de libre redacción y en los cuestionarios IMI.

Tras la realización de los 3 tipos de encuestas por los alumnos (preguntas de libre redacción, cuestionario SUS, cuestionario IMI) junto con las observaciones anotadas en las sesiones, se han podido extraer diversos resultados. Para las preguntas de libre redacción, en este apartado sólo se mostrarán lo que contestó cada alumno para posteriormente agrupar los comentarios comunes y hacer su análisis en el siguiente apartado.

A2.1 Resultados observaciones *Comecocos*

En la tabla 9 se muestran las observaciones anotadas durante la primera sesión.

Tipo	Rol	Descripción
Malfun.	Juguetero	Juguete del comecocos tiene la base muy grande para las paredes
Malfun.	Desarrollador	Incomodidad para hacer pruebas, ya que si se completa el juego hay que reiniciar la herramienta
Malfun.	Desarrollador	Sonido incorrecto no funciona
Duda	Desarrollador	Tamaño de feedback del fiducial lleva a confusión al ser siempre del mismo tamaño independientemente del tamaño del objeto

Tabla 9: Dudas y malfuncionamientos de la actividad *Comecocos*

El primer fallo que se tuvo de hacer la base del comecocos demasiado grande fue por un fallo de coordinación entre el desarrollador y el juguetero, ya que antes de hacer la base se hubiera tenido que hacer la prueba de que el fiducial asociado al objeto cabía bien entre las paredes del laberinto.

El malfuncionamiento de tener que reiniciar la herramienta dio la idea de que podría ser interesante que cuando se completara la actividad, se volviera al menú en lugar de salir. El fallo de que no sonaran los sonidos incorrectos se solucionó para la sesión 3.

Por último, la confusión del tamaño del feedback del fiducial se produjo al hacer las pruebas con el juguete del comecocos, ya que en la visualización sobre la mesa, el feedback asociado al juguete no cabía entre las paredes del laberinto aunque el fiducial asociado al juguete, que es el que hay que tener en cuenta, sí lo hiciera.

Como comentario adicional, para la actividad *Comecocos*, pudimos comprobar la utilidad de la opción de vuelta al menú junto con la recarga de los juegos, ya que esta actividad requería programar muchas áreas y era necesario estar comprobando continuamente que se iban definiendo bien.

Por lo tanto, si se analizan los resultados de la actividad *Comecocos*, a parte de la corrección del fallo del sonido se pudieron extraer dos ideas:

- Podría ser útil modificar la herramienta de modo que al acabar una actividad, se volviera al menú en lugar de tener que resetearse.
- Es necesario explicar mejor el funcionamiento del feedback de los objetos para que éste no cree confusión.

A2.2 Resultados observaciones *Granja*

En la tabla 10 se muestran las observaciones anotadas durante la segunda sesión.

Tipo	Rol	Descripción
Duda	Desarrollador	Duda sobre la relación de los anchos y posiciones con respecto al área
Duda	Desarrollador	Duda sobre si el feedback necesita obligatoriamente imagen neutra, imagen bien e imagen mal
Malfun.	Desarrollador	El feedback funciona raro, a veces muestra la cara de correcto cuando algo está mal y viceversa

Tabla 10: Dudas y malfuncionamientos de la actividad *Granja*

La primera duda surgió por el hecho de que las posiciones de las imágenes asociadas a áreas son relativas al centro del área, de modo que se entendió que los anchos y altos se especificaban también a partir del centro del área.

La segunda duda surgió porque en este juego, se necesitaba imagen bien e imagen mal pero no imagen neutra.

En cuanto al malfuncionamiento del feedback, fue un error que se pudo arreglar para la sesión 3.

Por lo tanto, si se analizan los resultados de la actividad *Granja* se puede concluir que no se echó en falta ninguna funcionalidad adicional en la herramienta pero que era necesario explicar mejor en el documento de definición de la herramienta:

- La relación de las imágenes asociadas a las áreas y sus atributos: posición x, posición y, ancho y alto.
- Aclarar el uso de imagen bien, imagen mal e imagen neutra, recalcando cuáles de esos tags son opcionales u obligatorios.

A2.3 Resultados observaciones *Series*

En la tabla 11 se muestran las observaciones anotadas durante la tercera sesión.

Tipo	Rol	Descripción
Duda	Desarrollador	Duda sobre si todas las áreas son interactivas. Faltaba indicar en el documento que puede haber áreas sin el tag <fid>
Duda	Desarrollador y Diseñador	Dudas sobre usar el feedback o usar los tag <imagen>, <imagenBien> e <imagenMal>
Malfun.	Juguetero	Problemas con la adaptación de los fiduciales a los juguetes
Malfun.	Desarrollador	No se muestra nunca la imagen de estado incorrecto

Tabla 11: Dudas y malfuncionamientos de la actividad *Series*

Para la actividad *Series*, la duda sobre si todas las áreas necesitaban un tag <fid> (es decir, una lista de objetos correctos o incorrectos) surgió porque para la actividad se necesitaban áreas que tuvieran asociadas imágenes aunque no se necesitara colocar ningún objeto sobre dichas áreas.

La duda sobre si usar el feedback o la combinación de imagen neutra, bien y mal surgió por no saber cuál era la mejor manera de indicar al usuario cuándo estaba poniendo una figura bien o cuándo la estaba poniendo mal, pues como se ha mencionado en apartados anteriores, hay varias formas de hacerlo.

El problema de los fiduciales surgió principalmente por las formas geométricas triangulares y estrelladas, ya que hubo que adaptar bastante los fiduciales iniciales proporcionados al juguetero para que se adaptaran bien a las figuras y pudieran detectarse correctamente sobre la mesa.

Por último, el fallo mencionado de no mostrarse la imagen adecuada cuando se coloca un objeto mal se arregló para la siguiente sesión de prácticas.

Por lo tanto, si se analizan los resultados de la actividad *Series*, tampoco se echó en falta ninguna funcionalidad adicional en la herramienta, pero sí que se pudo concluir que era necesario explicar mejor en el documento de definición de la herramienta:

- Los atributos de las áreas, recalcando más cuáles son opcionales y cuáles obligatorios.
- La diferencia entre utilizar el feedback y utilizar la imagen bien, imagen mal e imagen neutra.

A2.4 Resultados observaciones *Gran Prix*

En la tabla 12 se muestran las observaciones anotadas durante la cuarta sesión.

Tipo	Rol	Descripción
Malfun.	Grafista	Al llevar hecho un circuito con curvas, hubo necesidad de redibujarlo puesto que las áreas han de ser rectangulares
Malfun.	Desarrollador	El objeto de vuelta al menú a veces cuelga la herramienta
Malfun.	Desarrollador	Se produce un fallo si no se define una imagen de fondo
Malfun.	Desarrollador	Si defines todas las áreas sin el tag <fid>, la actividad finaliza
Malfun.	Desarrollador	Sonido correcto sigue sonando aunque haya acabado la actividad
Duda	Desarrollador	Duda sobre el uso de los OR y los AND en los fiduciales
Malfun.	Desarrollador	En áreas con múltiples fiduciales, el sonido bien ha de sonar cuando estén todos bien puestos, no cada vez que se ponga uno

Tabla 12: Dudas y malfuncionamientos de la actividad *Gran Prix*

El primer problema de esta actividad fue la necesidad de rediseñar el circuito que inicialmente se planteó puesto que este tenía zonas curvas, y las áreas que pueden definirse son solo rectangulares.

El error de vuelta al menú se producía porque si el juguete se colocaba en un área del menú en la que hubiera un juego, la herramienta entraba directamente al juego sin darle tiempo a cargar bien el menú y se colgaba.

El error del fondo se producía porque para crear el menú, se utiliza el tag <icono> y en caso de que este no exista, el tag <fondo>, por tanto si no se define ninguno de ellos, la herramienta da un error al intentar acceder a este último.

El error de que la actividad finalice si defines todas las áreas sin el tag <fid> no es un error como tal, ya que las áreas que no tienen una lista objetos correctos se consideran siempre correctas. El único problema de esto es que cuando se hacían pruebas, si definías solo áreas sin el tag <fid> la actividad se consideraba correcta y finalizaba, lo cuál obligaba a añadir como mínimo en un área un tag <fid> cualquiera para que no finalizara. Este problema se derivaba del hecho de que cuando una actividad se completaba había que resetear de nuevo la herramienta. Posteriormente se implementó que la herramienta no finalizase sino que volviera al menú al completarse la actividad.

El error del sonido fue un error que se detectó en esta sesión porque hasta el momento los archivos de música que se usaban en las actividades eran muy cortos (un par de segundos) pero en esta actividad se usaron archivos de más de medio minuto, con lo cuál cuando la actividad finalizaba, al volver al menú el sonido seguía sonando.

La confusión sobre el uso de los AND y los OR en fiduciales surgió porque en los juegos de ejemplo que se les mostraron a los alumnos, para que un área se considerara correcta se necesitaba que en ese área hubiera un determinado número de objetos. Sin embargo, en *Gran Prix* las áreas se consideraban correctas si sobre ellas se colocaba la moto o el coche, no los dos a la vez obligatoriamente.

El último error se detectó porque la carrera empezaba cuando los dos coches se situaban en el área de salida, el semáforo aparecía y se oía el sonido de salida. El problema era que tal como estaba implementada la herramienta, en vez de reproducirse el sonido cuando los dos transportes estuvieran posicionados sobre el área, sonaba cada vez que se posicionaba uno. El error fue posteriormente corregido.

Por lo tanto, si se analizan los resultados de la actividad *Gran Prix*, a parte de que esta sesión fue la que permitió descubrir más errores de implementación, también permitió extraer dos ideas:

- Podría ser útil modificar la herramienta de modo que pudieran definirse áreas no rectangulares (de ese modo se podría haber hecho el circuito inicial propuesto que tenía curvas).
- Es necesario explicar mejor la forma de definir los fiduciales de las áreas.

A2.5 Resultados libre redacción *Comecocos*

En la tabla 13 se muestran las preguntas de libre redacción contestadas por los alumnos durante la primera sesión.

Pregunta 1: La herramienta no ha sido capaz de soportar tus ideas porque...	
Rol	Opinión
Desarrollador	No soporta programación
Diseñador	Se han podido realizar todas las ideas
Grafista	No poder programar hace que el desarrollo del juego sea limitado
Juguetero	La actividad se ha realizado de manera rápida y con poca dificultad
Pregunta 2: Cualquier otro comentario será bien recibido	
Rol	Opinión
Desarrollador	Se trata de un proyecto interesante con fines educativos o lúdicos
Diseñador	–
Grafista	–
Juguetero	–

Tabla 13: Preguntas libre redacción *Comecocos*

A2.6 Resultados libre redacción *Granja*

En la tabla 14 se muestran las preguntas de libre redacción contestadas por los alumnos durante la segunda sesión.

Pregunta 1: La herramienta no ha sido capaz de soportar tus ideas porque...	
Rol	Opinión
Desarrollador	–
Diseñador	Ha soportado todo
Grafista	Ha soportado todo aunque hubiera algún error con los bordes de la pantalla
Juguetero	Ha soportado todo
Pregunta 2: Cualquier otro comentario será bien recibido	
Rol	Opinión
Desarrollador	–
Diseñador	De haber sabido mejor el funcionamiento de la herramienta se podría haber hecho algo más complicado
Grafista	–
Juguetero	–

Tabla 14: Preguntas libre redacción *Granja*

A2.7 Resultados libre redacción *Series*

En la tabla 15 se muestran las preguntas de libre redacción contestadas por los alumnos durante la tercera sesión.

Pregunta 1: La herramienta no ha sido capaz de soportar tus ideas porque...	
Rol	Opinión
Desarrollador	Soporta todas las ideas a pesar de algún problema técnico de mostrar imágenes o reproducir sonidos
Diseñador	Ha soportado todo
Grafista	Ha soportado todo
Juguetero	Ha soportado todo
Pregunta 2: Cualquier otro comentario será bien recibido	
Rol	Opinión
Desarrollador	La aplicación es muy sencilla pero podría permitir además el poder volver a una tarea previa mediante algún área interactiva
Diseñador	–
Grafista	Actividad interesante y positiva
Juguetero	Actividad interesante, se podría hacer alguna práctica más larga

Tabla 15: Preguntas libre redacción *Series*

A2.8 Resultados libre redacción *Gran Prix*

En la tabla 16 se muestran las preguntas de libre redacción contestadas por los alumnos durante la cuarta sesión.

Pregunta 1: La herramienta no ha sido capaz de soportar tus ideas porque...	
Rol	Opinión
Desarrollador	–
Diseñador	Animaciones para hacer los obstáculos de la carrera
Grafista	Áreas difíciles de programar
Juguetero	Dificultad en el diseño de escenarios de geometría compleja por sólo permitir áreas rectangulares
Pregunta 2: Cualquier otro comentario será bien recibido	
Rol	Opinión
Desarrollador	Es la primera vez que se ve una herramienta así para el desarrollo de videojuegos
Diseñador	–
Grafista	Trabajar con áreas no rectangulares abriría muchas puertas
Juguetero	Actividad relajada y entretenida que resulta positiva para desconectar

Tabla 16: Preguntas libre redacción *Gran Prix*

A2.9 Resultados cuestionarios IMI

A continuación, se mostrarán los resultados cuestionarios IMI separados por actividad.

En cuanto a los resultados de la actividad *Comecocos* (figura 60) se observa que todos los resultados superan el 60 % de satisfacción en la realización de la actividad, siendo los resultados más bajos los del desarrollador y los del grafista pero superando el tercer cuartil los del diseñador y el juguetero.

Analizando los resultados, el hecho de que los resultados del desarrollador y el grafista sean algo más bajos comparados con los del diseñador y el juguetero puede ser porque tanto el desarrollador y el grafista tuvieron que hacer un trabajo bastante tedioso, el primero con la definición de áreas en el XML y el segundo con el cálculo de los anchos y los altos para pasárselos al desarrollador.

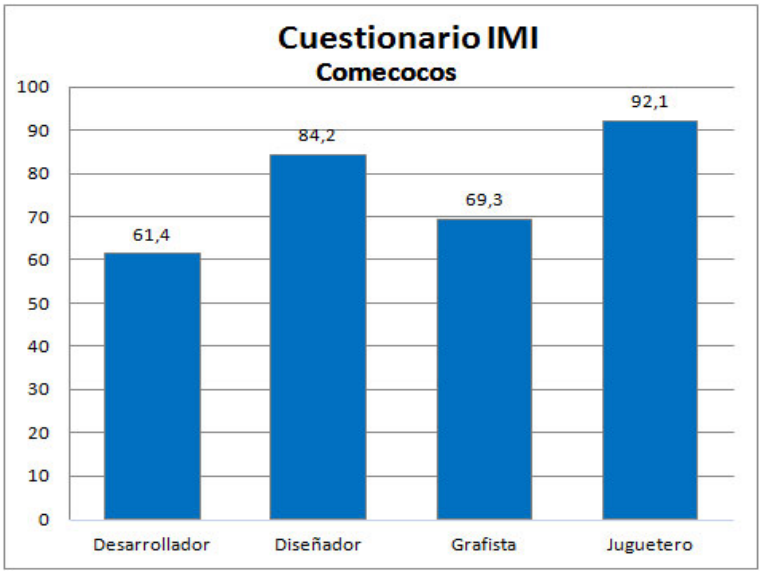


Figura 60: Cuestionarios IMI realizados por los miembros de la actividad *Comecocos*

En cuanto a los resultados de la actividad *Granja* (figura 61) se observa que los resultados del desarrollador, diseñador y grafista son muy parecidos (superando el desarrollador y el grafista el tercer cuartil y el diseñador quedándose muy cerca de este) pero el del juguetero es algo más bajo.

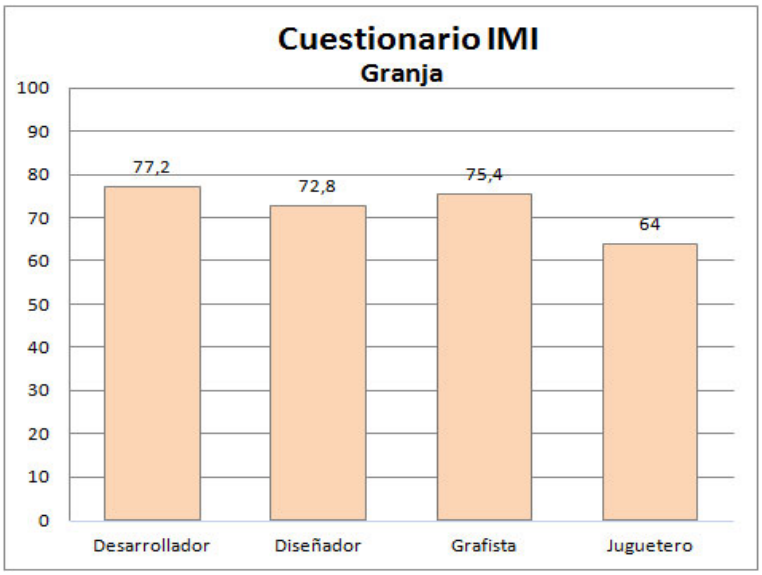


Figura 61: Cuestionarios IMI realizados por los miembros de la actividad *Granja*

Analizando los resultados, el hecho de que los resultados del juguetero sean bajos en comparación con los demás puede deberse a que para esta actividad, la implementación requerida era bastante básica pero se necesitaban bastantes juguetes (un caballo, un cerdo, una gallina, una oveja, un perro, un pato y una vaca), y aunque de los 7 jueguetes, le proporcionamos 5 (el caballo, el cerdo, la gallina, la oveja y la vaca), la realización del perro y del pato fue algo pesada por la forma de los animales escogidos.

En cuanto a los resultados de la actividad *Series* (figura 62) se observa que los resultados del desarrollador, diseñador y juguetero superan el tercer cuartil, notándose bastante la diferencia entre la puntuación más alta (diseñador con un 93 % de satisfacción) con respecto a la del juguetero.

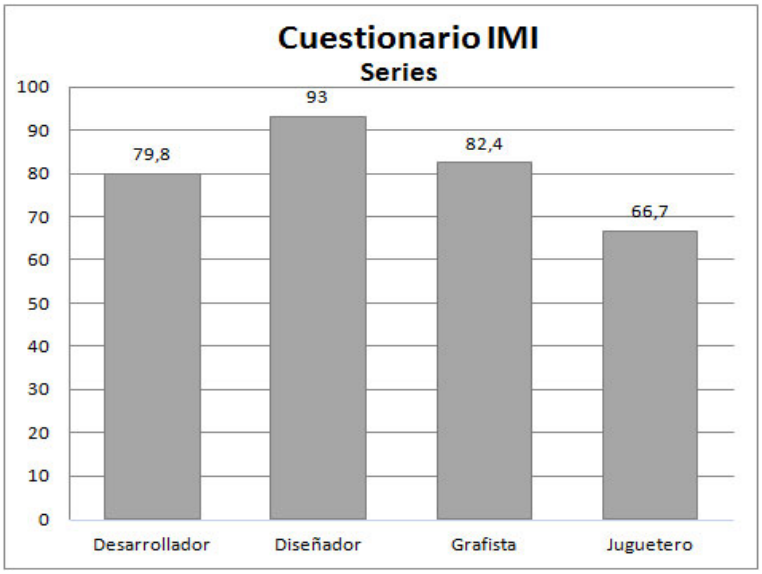


Figura 62: Cuestionarios IMI realizados por los miembros de la actividad *Series*

Analizando los resultados, las puntuaciones en general son bastante altas exceptuando la del juguetero debido a que la actividad en sí era bastante sencilla pero en cambio requería muchos juguetes, con lo cuál el juguetero tuvo que desempeñar un trabajo bastante repetitivo.

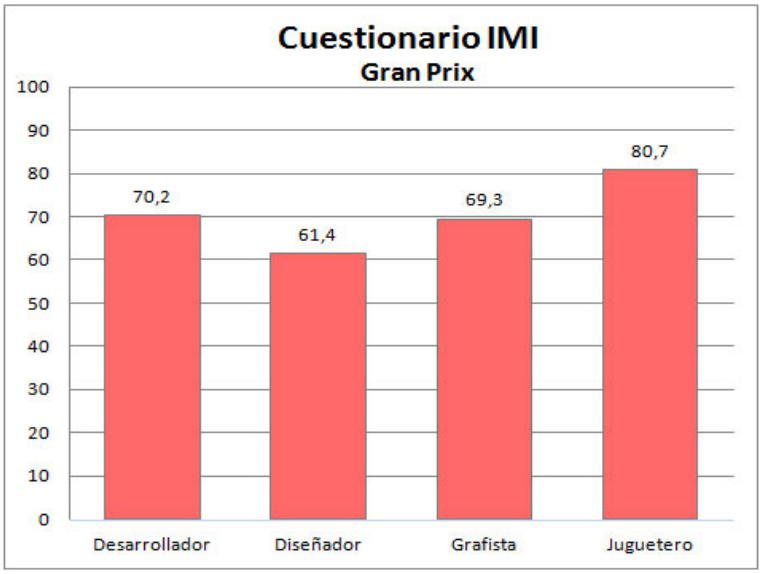


Figura 63: Cuestionarios IMI realizados por los miembros de la actividad *Gran Prix*

En cuanto a los resultados de la actividad *Gran Prix* (figura 63) se observa que todos los resultados superan el 60 % de satisfacción superando los del juguetero el tercer cuartil, teniendo este la puntuación más alta y el diseñador la más baja.

Analizando los resultados, el hecho de que los resultados del desarrollador, diseñador, y grafista sean más bajos que los del juguetero puede deberse a que en esa sesión, los juguetes a adaptar por el juguetero no requerían excesiva dificultad mientras que el diseñador tuvo que estar ayudando bastante al grafista debido a que fue necesario modificar el circuito que tenían ideado en un principio por no adaptarse correctamente a las áreas.

Anexo 3. Desarrollo temporal

En este apartado se muestran las tareas realizadas a lo largo del desarrollo del proyecto en las figuras 64 y 65.

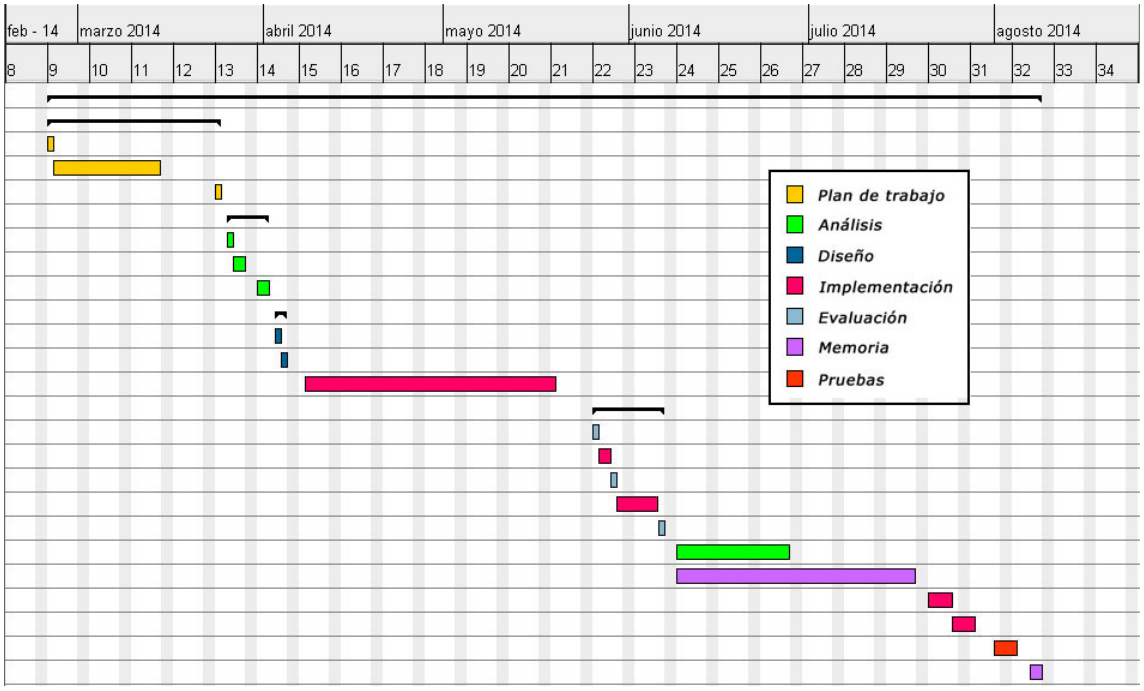
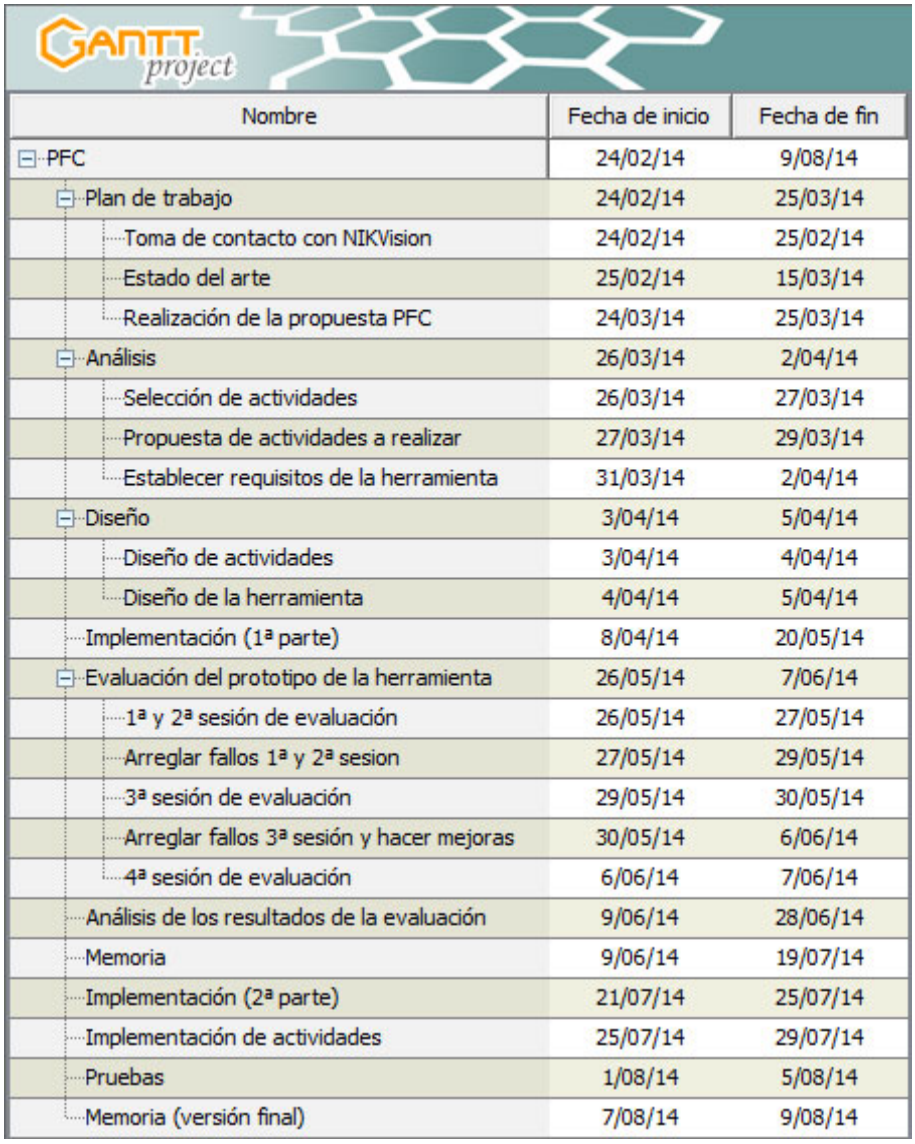


Figura 64: Diagrama de gantt del Proyecto Fin de Carrera.



Nombre	Fecha de inicio	Fecha de fin
PFC	24/02/14	9/08/14
Plan de trabajo	24/02/14	25/03/14
Toma de contacto con NIKVision	24/02/14	25/02/14
Estado del arte	25/02/14	15/03/14
Realización de la propuesta PFC	24/03/14	25/03/14
Análisis	26/03/14	2/04/14
Selección de actividades	26/03/14	27/03/14
Propuesta de actividades a realizar	27/03/14	29/03/14
Establecer requisitos de la herramienta	31/03/14	2/04/14
Diseño	3/04/14	5/04/14
Diseño de actividades	3/04/14	4/04/14
Diseño de la herramienta	4/04/14	5/04/14
Implementación (1ª parte)	8/04/14	20/05/14
Evaluación del prototipo de la herramienta	26/05/14	7/06/14
1ª y 2ª sesión de evaluación	26/05/14	27/05/14
Arreglar fallos 1ª y 2ª sesion	27/05/14	29/05/14
3ª sesión de evaluación	29/05/14	30/05/14
Arreglar fallos 3ª sesión y hacer mejoras	30/05/14	6/06/14
4ª sesión de evaluación	6/06/14	7/06/14
Análisis de los resultados de la evaluación	9/06/14	28/06/14
Memoria	9/06/14	19/07/14
Implementación (2ª parte)	21/07/14	25/07/14
Implementación de actividades	25/07/14	29/07/14
Pruebas	1/08/14	5/08/14
Memoria (versión final)	7/08/14	9/08/14

Figura 65: Tareas del Proyecto Fin de Carrera.

Bibliografía

[AAG09] P. Annett, M. Anderson, F. Goertzen, D. Halton, J. Ranson, Q. Bischof, and W. Boulanger. Using a multi-touch tabletop for upper extremity motor rehabilitation. In *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*, 2009

[Dra07] M. A. Drake. Evaluación de la atención. En *D. Burin, M. Drake, P.Harris. Evaluación neuropsicológica en adultos*. Buenos Aires: Paidós, 2007

[IMI85] E. L. Deci & R. M. Ryan, Intrinsic Motivation and Self-Determination in Human Behavior. Springer. 1985

[EJE14] Ejercicios personas mayores: https://obrasocial.lacaixa.es/deployedfiles/obrasocial/Estaticos/pdf/Gente_30/ejercicios_es.pdf

[EST14] Ejercicios de estimulación cognitiva: http://blog.infoelder.com/ejercicios-de-estimulacio-cognitiva-que-se-pueden-hacer-en-casa

[GMS09] L. Gamberini, F. Martino, B. Seraglia, A. Spagnolli, M. Fabregat, F. Ibanez, M. Alcaniz, and J. M. Andres. Eldergames project: An innovative mixed reality table-top solution to preserve cognitive functions. In *Human System Interactions, 2009. HSI '09. 2nd Conference on*, 2009

[GIG14] Giga Affective Lab: http://giga.cps.unizar.es/affectivelab/

[JPM04] H. Jimison, M. Pavel, J. McKanna, and J. Pavel. Unobtrusive monitoring of computer interactions to detect cognitive status in elders. In *IEEE Transactions on Information Technology in Biomedicine*, 2004

[KSG06] D. Kern, M. Stringer, G. Fitzpatrick, and A. Schmidt. Curball–A Prototype Tangible Game for Inter-Generational Play. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06*. 15th IEEE International Workshops on (pp. 412-418). IEEE, 2006

[KKP13] G.H. Kwon, L. Kim, and S. Park. Development of a cognitive assessment tool and training systems for elderly cognitive impairment: A case study of a successful development of a cognitive assessment tool and training systems for the elderly people in South Korea. In *proceedings of 7th International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, 2013

[LTK07] M. Leitner, M. Tomitsch, T. Költringer, K. Kappel, T. Greshenig. Designing tangible tabletop interfaces for patients in rehabilitation. In *Proceedings of Conference & Workshop on Assistive Technologies for People with Vision and Hearing Impairments: Assistive Technology for All Ages*. M. Hersh (Eds.), Spain. pp. (1-7), 2007

[MCB08] Javier Marco, Eva Cerezo, Sandra Baldasarri. NIKVISION: NATURAL INTERACTION FOR KIDS, 2008

[MCB12] Javier Marco, Eva Cerezo, Sandra Baldasarri. ToyVision: A Toolkit for Prototyping Tabletop Tangible Games. In *The fourth ACM SIGCHI EICS*, 2012.

[NIK14] NIKVision: http://webdiis.unizar.es/~jmarco/?page_id=10&lang=es

[PAC14] Pac-Man: http://es.wikipedia.org/wiki/Pac-Man

[PIX12] Pixelsense: http://www.microsoft.com/en-us/pixelsense

[SMA14] Smart Table: http://smarttech.com/table

[SOC12] Sociable: http://www.cognitivetraining.eu/?q=services-0

[SUS96] J. Brooke, SUS-A quick and dirty usability scale. Usability evaluation in industry, 189, 194. 1996

[TAL14] Ejercicios de estimulación cognitiva: http://tallerescognitiva.com/descargas/muestra.pdf

[TAM14] Tangram: http://es.wikipedia.org/wiki/Tangram

[TOU13] TouchTable: http://www.touchtable.com/

Índice de figuras