Universidad Zaragoza

Proyecto Fin de Carrera

Desarrollo de un editor gráfico basado en un lenguaje de modelado visual para juegos híbridos.

Autor

Guillermo Lafuente Gállego

Directores

Dra. Sandra Baldassarri
Dr. Javier Marco Rubio

Escuela de Ingeniería y Arquitectura

2015

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

Datos técnicos

Título

Desarrollo de un editor gráfico basado en un lenguaje de modelado visual para juegos híbridos.

Autor

Guillermo Lafuente Gállego

DNI

77131796-P

Titulación

Ingeniería Informática

Directores

Dra. Sandra Baldassarri

Dr. Javier Marco Rubio

Departamento

Informática e Ingeniería de Sistemas

Centro

Escuela de Ingeniería y Arquitectura

Universidad

Universidad de Zaragoza

Fecha

Septiembre 2015

Desarrollo de un editor gráfico basado en un lenguaje de modelado visual para juegos híbridos.

Resumen

Durante los últimos años se ha desarrollado en el GIGA Affective Lab, NIKVision, un prototipo de tabletop o superficie activa de interacción tangible. Conjuntamente se ha desarrollado ToyVision, un framework que facilita la implementación de juegos híbridos aislando al desarrollador del hardware de NIKVision.

ToyVision se basa en el Lenguaje de Modelado de Interfaces Tangibles de Usuario (TUIML) el cual permite, mediante un lenguaje basado en marcas (XML), definir las diferentes piezas u objetos que van a formar parte en el juego, así como las distintas manipulaciones que se puedan realizar sobre ellas. En concreto, ToyVision soporta tanto manipulaciones pasivas realizadas por el usuario, como manipulaciones activas, realizadas por el sistema mediante actuadores electrónicos utilizando un microcontrolador Arduino.

El objetivo de este Proyecto Fin de Carrera ha sido la creación de un editor gráfico para ToyVision. Este editor da soporte a diseñadores de juegos híbridos para modificar piezas de juego convencionales para ser usadas en juegos para NIKVision, permitiendo definir visualmente el TUIML del juego. Concretamente, el editor gráfico creado en este PFC permite:

- Definir visualmente y de forma sencilla los diferentes objetos que se utilizan en un juego.
- Definir las manipulaciones realizadas por el usuario asociadas a dichos objetos.
- Definir las manipulaciones que realiza el sistema sobre los objetos, dando soporte para conectar los actuadores electrónicos al microcontrolador Arduino.
- Probar el funcionamiento de las piezas y sus manipulaciones en el tabletop NIKVision de forma inmediata una vez definidas en el editor gráfico.
- A partir de las piezas y manipulaciones especificadas, generar el fichero toys.xml que contenga el TUIML del juego.

Por último, se han realizado una serie de pruebas, tanto funcionales como de usabilidad, para comprobar el correcto funcionamiento del editor y conocer la valoración de éste por parte de usuarios finales.

Agradecimientos

En primer lugar quiero dar las gracias a mis directores, Sandra y Javier, sin cuya inestimable ayuda no habría sido posible la realización de este PFC.

En segundo lugar me gustaría agradecer a mi familia todo el apoyo recibido. En especial quiero agradecer a mis padres y a mi hermana por haberme dado ánimos en los momentos de bloqueo.

Índice

1. Introducción

En este primer capítulo se expone el contexto en el que se ha desarrollado este Proyecto Fin de Carrera y se enumeran los objetivos del mismo. También se presenta la estructura de esta memoria.

1.1. Contexto

Durante los últimos años, en el GIGA Affective Lab [GIGAWeb] se ha desarrollado NIKVision [MCB08], un prototipo de tabletop o superficie horizontal activa. NIKVision está diseñado específicamente para ser utilizado por niños, y permite la interacción mediante la manipulación de juguetes sobre la superficie de la mesa (figura 1).

Figura 1. Niños jugando con NIKVision.

Al colocar un objeto sobre la mesa, ésta lo detecta a través de una cámara infrarroja. Para facilitar el reconocimiento de los objetos, a éstos se les añaden marcadores denominados fiduciales. De esta forma se puede distinguir un objeto de otro y conocer su posición y su orientación. NIKVision también cuenta con un proyector que permite el visionado de imágenes sobre la superficie de la mesa.

Conjuntamente a NIKVision, se ha desarrollado ToyVision [PFCU13], un *toolkit* para el desarrollo de juegos híbridos en dicho tabletop, aportando una capa de comunicación entre NIKVision y la aplicación *host*, quien ejecuta el código del juego. ToyVision se basa en el *framework* de visión por computador ReacTIVision [KB07], al que se le ha añadido una nueva capa de abstracción, implementado un nuevo protocolo de comunicación basado en mensajes

XML y dado soporte para sensores y actuadores embebido en las piezas de juego, conectados a un microcontrolador Arduino [ArduinoWeb]. En la figura 2 se puede observar un esquema de la arquitectura de ToyVision.

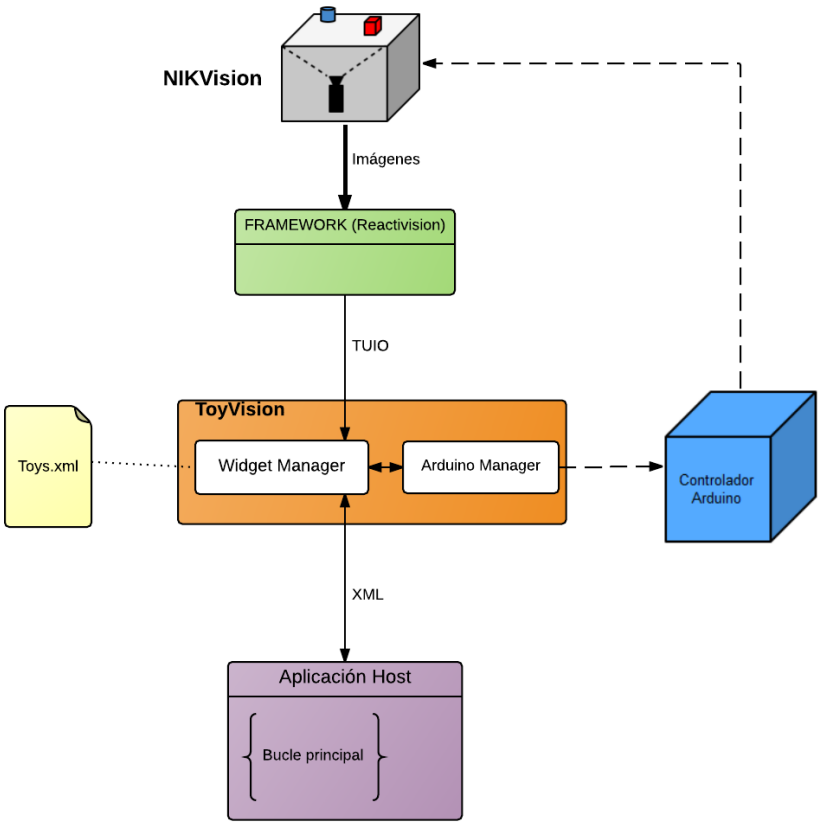


Figura 2. Esquema de ToyVision.

ReacTIVision se encarga de la comunicación con el hardware de visión infrarroja del dispositivo tabletop. Para ello aplica un algoritmo de visión por computador a las imágenes que le envía la cámara de NIKVision y reconoce los fiduciales asociados a los objetos. Finalmente envía la información de los objetos a la aplicación host utilizando el protocolo TUIO [KBBC05].Este protocolo se basa en eventos *add, update y remove*, enviando un mensaje cada vez que un objeto es colocado, movido o quitado de la mesa, respectivamente. Este protocolo distingue además el tipo de objeto, diferenciando objetos de tipo *Cursor* (objetos simples como fichas de juego), *Object* (con un fiducial asociado) y *Blob* (objetos deformables).

Los eventos TUIO son recogidos e interpretados por ToyVision en el módulo "*Widget Manager*", el cual busca relaciones entre distintos eventos para relacionarlos directamente con manipulaciones de las piezas de juego sobre el tabletop, y descartando aquellas que no tienen ningún significado en el juego concreto que se está ejecutando en la aplicación *host.* Esta traducción de eventos de bajo nivel a eventos de alto nivel de abstracción se realiza a

través del lenguaje de modelado TUIML (*Tangible User Interface Modelling Language*) [MCB14].

Mediante TUIML se puede especificar las relaciones entre las manipulaciones de objetos, y la información digital de la aplicación. Dicha relación se especifica mediante relaciones **Token and Constraint** (**TAC**) (ver sección 2.1. para más detalles). ToyVision utiliza una sintaxis en lenguaje de marcas XML para especificar la colección de relaciones **TAC** que tiene significado para un juego o aplicación concreta en el tabletop NIKVision. Dichas relaciones se guardan en un fichero toys.xml, al cual recurre el *"Widget Manager"* para obtener su base de conocimiento sobre las manipulaciones con significado durante la ejecución del juego.

ToyVision también incluye un *"Arduino Manager"* que se encarga de la comunicación con el microcontrolador Arduino para recoger la información de sensores y mandar órdenes a actuadores electrónicos embebidos en las piezas de juego. Esto permite tener piezas de juego activas, capaces de responder autónomamente durante la ejecución del juego.

Finalmente, la comunicación entre ToyVision y la aplicación *host* encargada de ejecutar el juego se realiza mediante un *socket* TCP/IP, a través del cual se envían mensajes codificados en XML. Dichos mensajes siguen la sintaxis del TUIML para indicar el estado actualizado de las piezas de juego sobre el tablero. Dicha comunicación es bidireccional: ToyVision envía mensajes indicando las manipulaciones de los usuarios, y la aplicación *host* envía mensajes a ToyVision con comandos para ejecutar acciones sobre las piezas de juego a través de los actuadores embebido y conectados al microcontrolador Arduino.

1.2. Objetivos del proyecto

Para definir un juego en NIKVision es necesario especificar su TUIML en el correspondiente fichero toys.xml. Actualmente, esto implica escribirlo directamente en formato XML, hecho que complica el proceso de creación. Por otro lado, es necesaria una herramienta que permita probar los dispositivos electrónicos conectados al juego mediante una placa arduino.

El objetivo de este Proyecto Fin de Carrera es la creación de un editor gráfico para el *framework* ToyVision que de soporte a diseñadores para modificar juguetes convencionales para luego ser usados en juegos híbridos para NIKVision, permitiendo definir visualmente la estructura del juego. Este editor facilitará en gran medida la especificación del lenguaje TUIML del juego y la generación del fichero toys.xml asociado

Concretamente, este editor gráfico debe permitir:

- Definir visualmente y de forma sencilla los diferentes objetos que se utilizan en un juego.
- Definir las manipulaciones realizadas por el usuario asociadas a dichos objetos.
- Definir las manipulaciones que realiza el sistema sobre los objetos, dando soporte para conectar los actuadores electrónicos al microcontrolador Arduino.

- Probar el funcionamiento de las piezas y sus manipulaciones en el tabletop NIKVision de forma inmediata una vez definidas en el editor gráfico.
- A partir de las piezas y manipulaciones especificadas, generar el fichero toys.xml que contenga el TUIML del juego.

1.3. Estructura de la memoria

A continuación se presentan los seis capítulos que conforman la memoria del trabajo:

- **Introducción.** En este primer capítulo se presenta el contexto en el que se encuadra este proyecto, así como los objetivos del mismo y la estructura de esta memoria.
- **Análisis.** El segundo capítulo muestra el análisis del problema a resolver, incluyendo el estado del arte y los requisitos del sistema obtenidos de dicho análisis.
- **Diseño e implementación.** En el tercer capítulo de la memoria se presenta el diseño elegido para el editor gráfico, así como las principales decisiones tanto de diseño como de implementación tomadas durante el desarrollo del mismo.
- **Resultados obtenidos.** En este apartado se muestra la versión final del editor, detallando los aspectos principales del mismo.
- **Evaluación.** En este capítulo se presentan las diferentes pruebas realizadas al editor y los resultados obtenidos de las mismas.
- **Conclusiones y trabajo futuro.** En el último capítulo de esta memoria se exponen las conclusiones obtenidas del desarrollo de este proyecto, así como algunas líneas de trabajo a seguir desarrollando en un futuro.

La memoria de este proyecto se completa con una serie de anexos que incluyen los documentos generados durante la realización del proyecto y la gestión y distribución del tiempo de desarrollo del mismo. Los anexos son los siguientes:

- **ANEXO A. Documentación de Software.** En este primer anexo se exponen los documentos realizados durante las fases de análisis y diseño.
- **ANEXO B. Ficheros Toys.xml.** En este anexo se incluyen los ficheros toys.xml de los juegos realizados con el editor para evaluar su funcionalidad.
- **ANEXO C. Gestión del Proyecto.** Este anexo incluye la planificación y gestión temporal del proyecto.

2. Análisis

Este capítulo se compone de dos partes diferenciadas. En el primer apartado se expone una introducción a la interacción tangible. En la segunda parte de este capítulo se muestran los requisitos del sistema obtenidos tras haber analizado el problema.

2.1. Estado del Arte

En 1997, el Tangible Media Group del MIT (Massachusetts Institute of Technology) [TMGWeb] desarrolló metaDESK [UI97] (figura 3), un sistema con el que se interactuaba mediante la manipulación de objetos físicos sobre una superficie horizontal. Con metaDESK se introdujo el concepto de Interfaz Tangible de Usuario (TUI).

Figura 3. Sistema metaDESK (izq.) y objeto para manipularlo o *Phicon* (Dcha.).

El sistema metaDESK se componía de una superficie horizontal sobre la que se proyectaba un mapa, el cual se manipulaba mediante unas figuras denominadas *phicons (Phisical Icons)* (figura 3 dcha.). Al colocar los *phicons* sobre el mapa, éste se modificaba para hacerlo coincidir con los mismos. También contenía una lupa que al ponerla sobre el mapa mostraba en el área dentro de ella una capa diferente de información del mapa. Otro de los elementos que incluía metaDESK era una escala para modificar el zoom del mapa. Por último, mediante una pantalla adicional se podía observar una simulación tridimensional del mapa.

Para especificar la estructura y la funcionalidad de las TUIs se propuso el paradigma **TAC** (**Token and Constraint**) [SLCJ04]. Este paradigma permite definir los objetos físicos que intervienen en una TUI y las restricciones físicas que se aplican a dichos objetos.

Basándose en el paradigma **TAC** se desarrolló un Lenguaje de Modelado de Interfaces Tangibles de Usuario (TUIML) [MCB14], cuyo esquema se puede ver en la figura 4. En este paradigma se aprecian tres tipos de elementos diferentes:

- **Token:** objeto físico que interviene en el juego.
- **Constraint:** restricción física impuesta por un **token** a otro (**subtoken**).
- **TAC:** relación de un **Constraint** con un **token**, indicando las manipulaciones del **token** sobre el área del **constraint**.

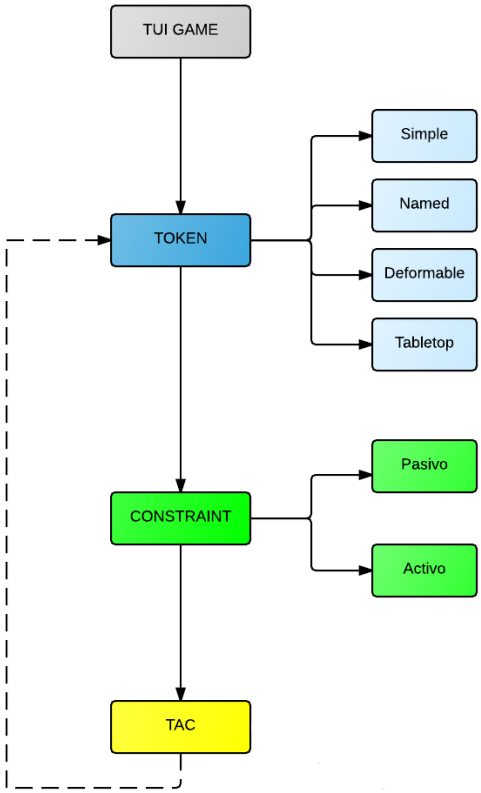


Figura 4. Esquema del TUIML

A continuación se explican con detalle los diferentes elementos del TUIML.

- **Token.** Cada elemento físico que interviene en el juego. Los **tokens** se clasifican en los siguientes tipos:
    - **Token simple**. Puede ser una pieza pequeña o un dedo. De un **token simple** se puede conocer su posición.
    - **Named token**. Piezas que llevan asociado un fiducial de tal forma que pueden ser identificados y conocer de ellos su posición y orientación.
    - **Token deformable.** Esta clase de **token** tiene forma variable, como por ejemplo los objetos hechos con plastilina o telas.
    - **Tabletop.** El propio tabletop también se considera como un **token** debido a que pueden especificarse restricciones.

- **Constraint.** Representan las restricciones físicas de un **token** respecto a la manipulación de otro **token**. La clasificación de los **constraints** es la siguiente:

    - **Constraint pasivo**. Es el propio usuario el que actúa sobre él.

    - **Constraint activo.** El sistema provoca un cambio de estado en el objeto físico dentro del área asociada al **constraint**.

  Por otro lado, un **constraint** también puede tener asociado un sensor (**constraint pasivo**) o un actuador (**constraint activo**) conectado mediante una placa de Arduino.

- **TACs.** Representan las relaciones entre los diferentes **tokens** a través de sus **constraints**. Un **TAC** define la asociación de un **constraint** con un **token**, denominado **subtoken**, así como las manipulaciones del **subtoken** dentro del área definida por el **constraint**. Estas manipulaciones en la tabla 1.

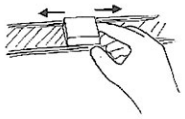
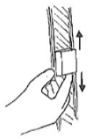
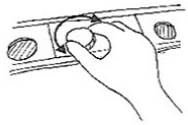
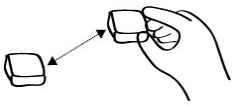
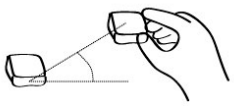
Icono	Nombre	Descripción
	Asociativo	Asociación del **subtoken** al **constraint**.
	Manipulativo X	Posición relativa horizontal del **token** respecto al **constraint**.
	Manipulativo Y	Posición relativa vertical del **token** respecto al **constraint**.
	Manipulativo Rotación	Orientación del **subtoken** respecto al **constraint**.
	Manipulativo Distancia	Distancia relativa del **subtoken** dentro del **constraint**.
	Manipulativo Orientación	Ángulo relativo del **subtoken** respecto del **constraint**.

Tabla 1. Manipulaciones posibles definidas por un TAC.

Durante la fase de análisis se estudiaron algunos editores gráficos que sirven como interfaz para la especificación de juegos en TUIs.

Papier-Mâché

Papier-Mâché [KLLL04] es un editor gráfico para TUIs que permite la introducción de objetos a un modelo generando una serie de eventos para cada objeto. A través de una cámara este editor obtiene imágenes a las que aplica un algoritmo de reconocimiento identificando los objetos que se ven en ésta. Para cada uno de los objetos reconocidos, a los que denomina *phobs*, se generan eventos de *add, update* y *remove*. *Papier-Mâché* utiliza códigos de barras para crear identificadores únicos asociados a los objetos. La interfaz de *Papier-Mâché* se puede observar en la figura 5

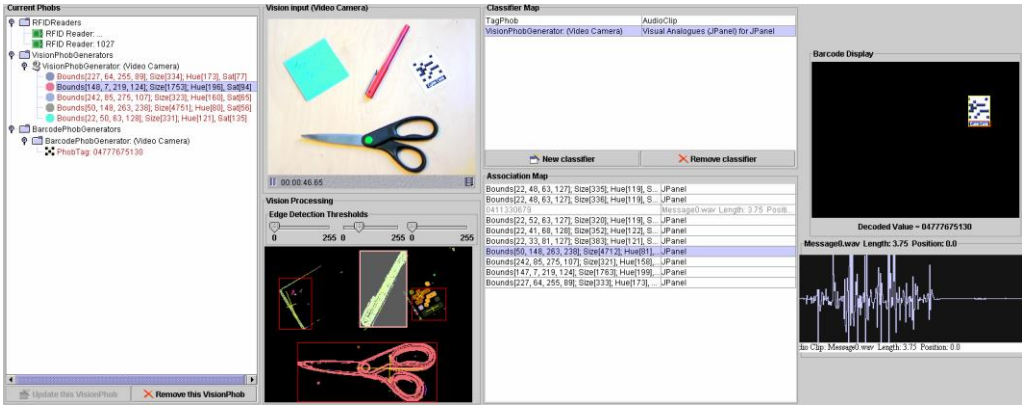


Figura 5. Interfaz Gráfica de *Papier-Mâché*.

Prism

Basado en la arquitectura TUIMS (*Tangible User Interface Management System*), *Prism* [LSJ04] permite crear representaciones gráficas de objetos físicos. Para ello cuenta con un entorno de desarrollo y simulación basado en Java3D. Para representar la estructura de las aplicaciones tangibles creadas, este editor utiliza el paradigma **TAC**.

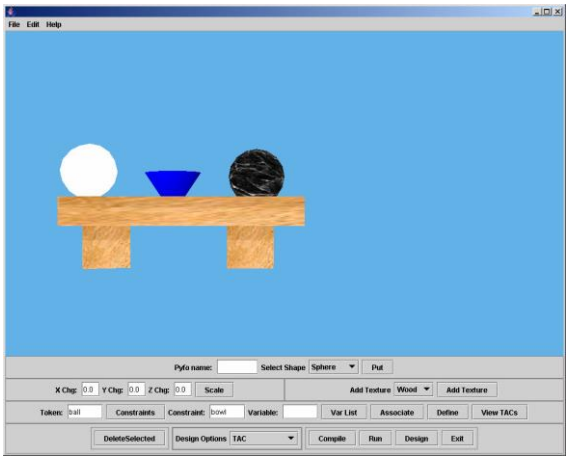


Figura 6. Interfaz gráfica de *Prism*.

Los editores anteriormente mencionados permiten la creación de aplicaciones para TUIs. Sin embargo, ambos poseen limitaciones que dificultan la especificación de aplicaciones tangibles para ToyVision. *Papier-Mâché* se basa en eventos *add, update* y *remove* (similar al protocolo TUIO), distinto al TUIML de ToyVision. Por otro lado, Prism se basa en el paradigma **TAC** y genera un fichero con la especificación del TUIML del juego, pero éste se basa en simulaciones, y en el caso concreto de NIKVision no se comunicaría con el tabletop.

2.2. Análisis de Requisitos

En este apartado se muestran los requisitos del editor obtenidos tras realizar el análisis del problema. Los documentos generados en el proyecto durante la fase de análisis se pueden consultar en el anexo A de esta memoria.

Requisitos funcionales

Para la elaboración de los requisitos funcionales del sistema, a éstos se les han clasificado en tres grandes grupos:

- **Requisitos generales.** Aquellos relacionados con la labor del editor sobre los juegos, en concreto los relacionados con la creación, carga y guardado de los mismos.

- **Requisitos de edición de elementos.** Aquí se engloban todos los requisitos relacionados con añadir, editar y eliminar los elementos de los que se compone el juego.

- **Requisitos de visualización.** Los requisitos clasificados en este grupo están relacionados con la información que el editor deberá mostrar al usuario.

Los requisitos funcionales obtenidos se muestran clasificados en la siguiente tabla:

Código	Descripción
Requisitos generales	
RF-1	El sistema permitirá crear un juego nuevo
RF-2	El sistema permitirá la carga de juegos
RF-3	El sistema permitirá el guardado de juegos.
Requisitos de edición de elementos	
RF-4	El sistema permitirá añadir, editar y eliminar **tokens** al juego.
RF-5	El sistema permitirá añadir, editar y eliminar restricciones (**constraints**) sobre los **tokens**.
RF-6	El sistema permitirá añadir, editar y eliminar **TACs** sobre los **constraints**.
RF-7	El sistema permitirá la definición de áreas poligonales sobre la superficie del tabletop.

Requisitos de visualización	
RF-8	El sistema mostrará el estado actual de la mesa.
RF-9	El sistema mostrará el TUIML del juego.
RF-10	El sistema permitirá probar los sensores y actuadores asociados a las manipulaciones activas.
RF-11	El sistema mostrará información sobre los elementos del juego añadidos.

Requisitos no funcionales

Los requisitos no funcionales son aquellos relacionados con los aspectos técnicos del editor. Éstos especifican, entre otros, la estructura de los datos que se utilizarán en la aplicación o las tecnologías en las que se realizará.

Código	Descripción
RNF-1	El sistema se implementará utilizando el lenguaje Processing.
RNF-2	El sistema interactuará con el hardware NIKVision
RNF-3	El sistema se ejecutará en tiempo real.
RNF-4	El sistema deberá permitir la interacción con la plataforma Arduino.

3. Diseño e implementación

Para el proceso de creación del editor se ha seguido un desarrollo iterativo incremental. En cada iteración realizada se obtenía un prototipo funcional del editor, al que se le iban incorporando nuevas funcionalidades en las posteriores iteraciones. De esta forma se consigue un *feedback* por parte de un experto en etapas tempranas del proceso de desarrollo.

3.1. Diseño

Para realizar el diseño del prototipado de la interfaz del editor, primero se elaboró un *wireframe* para definir la distribución de los elementos en la pantalla de la aplicación, para posteriormente elaborar un *mockup* de la misma, para visionar de forma más detallada el aspecto de la aplicación.

Versión 1

La pantalla principal de este primer diseño (figura 7) se divide en 6 zonas. En la zona superior izquierda se encuentra el botón de dibujar área, el cual se utiliza para borrar los puntos del área existente en esquema del estado actual de la mesa y empezar de nuevo a dibujar un área. En la zona superior derecha se localizan los botones para añadir un nuevo elemento al modelo, habiendo un botón para cada tipo de elemento. En la zona central izquierda se sitúa el esquema con el estado actual de la mesa, en el que se muestran los **tokens** colocados sobre la mesa y los **constraints** asociados a esos **tokens**. A la derecha de éste se encuentra representado el TUIML del juego. En la parte inferior izquierda de la pantalla se sitúa una zona de edición en la que se pueda modificar los datos de los elementos del juego. Finalmente, hay un botón para generar el XML del juego en la zona inferior derecha.

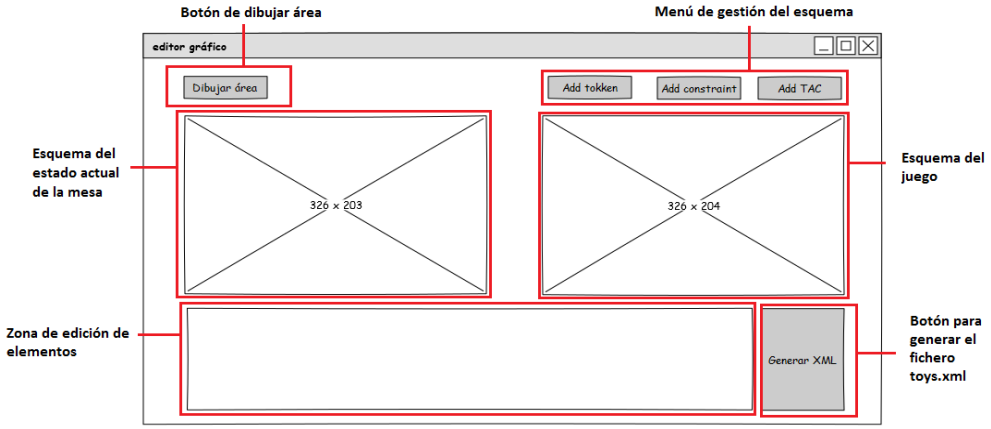


Figura 7. Wireframe del primer prototipo de la pantalla principal del editor.

Esta primera versión no permitía la carga de juegos ni la creación de juegos nuevos, por lo que se desarrolló una segunda versión del prototipo que añadía estas funcionalidades.

Versión 2

En el segundo prototipo en *wireframe* (figura 8), se añadieron los botones para crear un nuevo juego y para cargar un juego ya creado, que junto con el botón para generar el XML del juego se sitúan en la zona superior izquierda. De esta forma, los tres botones conforman el menú para la gestión de los juegos. El botón de dibujar área se eliminó de la pantalla principal del editor y así mostrarlo en la ventana para añadir un constraint, momento en el que utiliza. Por último, al haber recolocado el botón "generar XML" se ha aprovechado ese espacio para ampliar el área de edición de elementos.

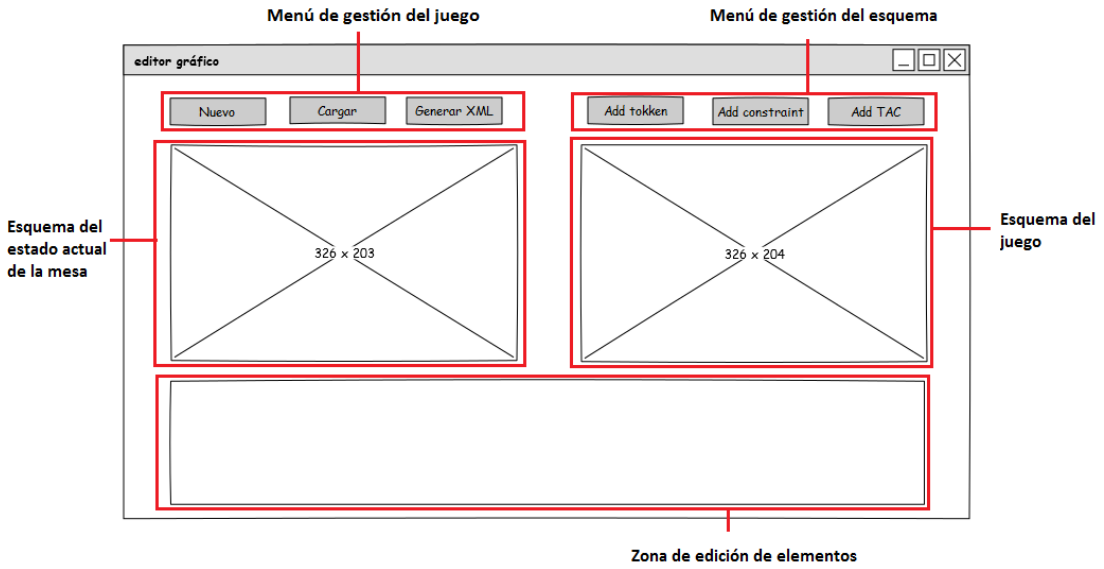


Figura 8. Segundo prototipo en wireframe.

Tras realizar el segundo *wireframe*, se realizó un tercer prototipo, esta vez en forma de *mockup* (figura 9) para visualizar el aspecto final del editor. Los **tokens** en la mesa se representarían como círculos blancos en la mesa, y las áreas de **constraint** se mostrarían en color rojo sobre el fondo negro de la mesa. En el esquema del TUIML del juego los **tokens** se mostrarían en color azul, los **constraints** en verde y los **TACs** en color amarillo, mostrando con flechas las relaciones entre los diferentes elementos. El elemento seleccionado se mostraría con el borde más grueso que el resto. Por otro lado, en la zona de edición de elementos se situarían botones para guardar los cambios realizados en el elemento (botón "actualizar") y para eliminar el elemento del modelo (botón "borrar"). En este prototipo se añadieron las ventanas para añadir elementos al modelo.

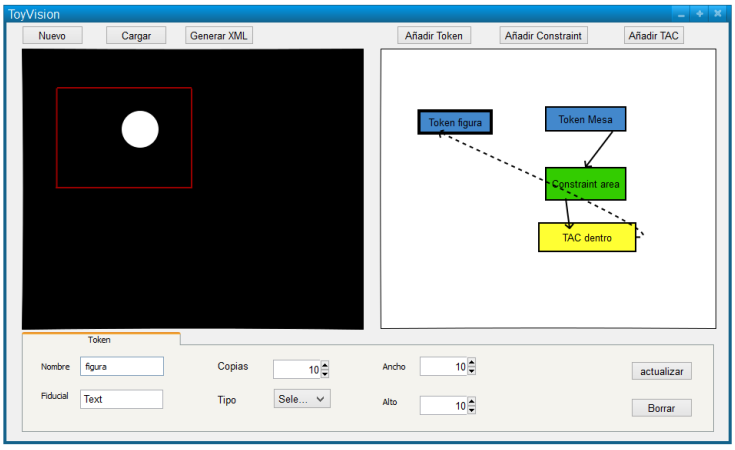


Figura 9. Tercer prototipo del editor.

En el prototipo de la ventana para añadir un **token** (figura 10), aparecen los campos con la información que deberá completar el usuario para introducirlo en el modelo. En la parte inferior aparecen los botones para aceptar o cancelar la introducción del **token**. En concreto, los campos de esta ventana son los siguientes:

- **Nombre:** campo de texto para introducir el nombre del **token**.
- **Tipo:** lista de los tipos de **token** entre los que el usuario deberá elegir el que corresponda:
  - **Simple**
  - **Named**
  - **Deformable**
- **Fiducial:** campo de texto donde se introducirá el código numérico que identificará la marca del **token**. Este campo sólo se activará si el tipo de **token** seleccionado es **named**.
- **Copias:** *spin box* (campo de texto numérico con botones en un lateral para incrementar y decrementar el valor) para introducir el número elementos del juego que coincidan con ese **token**.
- **Ancho:** *spin box* para introducir el ancho del **token** en píxeles.
- **Alto:** *spin box* para introducir el alto del **token** en píxeles.

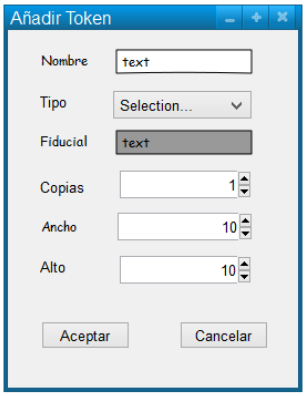


Figura 10. Primer prototipo de ventana para añadir un token.

La figura 11 muestra el prototipo de la ventana para añadir un **constraint**. El aspecto de esta ventana cambia en función de si es un **constraint** asociado a un área o un **constraint** asociado a un sensor o un actuador mediante la plataforma arduino. Los campos comunes a ambas ventanas son los siguientes:

- **Token:** lista para seleccionar el **token** al que está asociado el **constraint**.
- **Nombre:** campo de texto para introducir el nombre del **constraint**.
- **Tipo:** lista para seleccionar el tipo de **constraint** (pasivo o activo).
- **Arduino:** casilla de verificación para indicar si el **constraint** está asociado a un controlador arduino o no.

En el caso de que se trate de un **constraint** asociado a un área (figura 11 izq.)además de los campos comunes se muestran los siguientes:

- **Orientable:** casilla de verificación para indicar si el **constraint** es orientable o no.
- **Dibujar área:** botón para limpiar el área del **constraint** y volver a dibujarlo.

Si por el contrario se trata de un **constraint** asociado a un controlador arduino (figura 11 dcha.), los campos que se muestran, a parte de los comunes, son los siguientes:

- **Num. Arduino:** campo de texto para indicar el número de la placa de arduino asociada al **constraint**.
- **Actuador:** lista para seleccionar el tipo de actuador al que está asociado el **constraint**.
- **Puerto:** campo de texto para indicar el puerto de la placa al que está conectado el controlador.

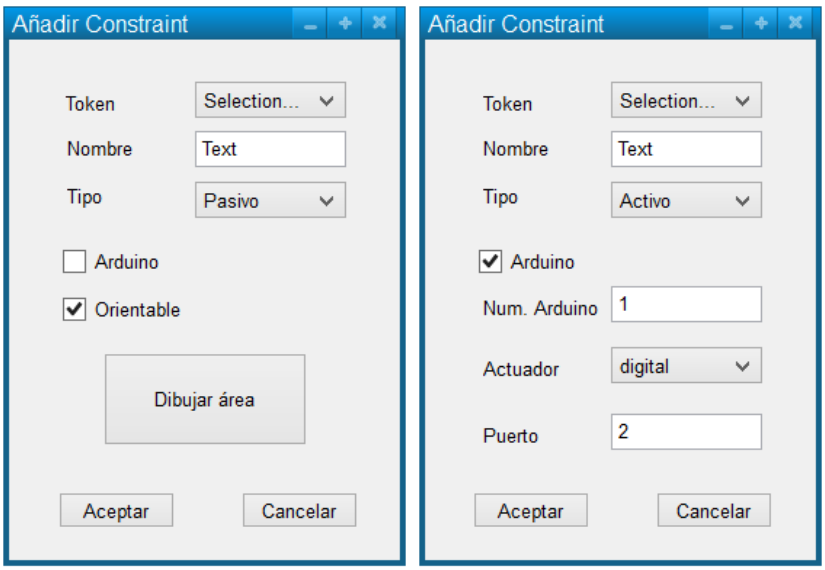


Figura 11. Prototipo de ventana para añadir constraint con área asociada (izq.) y un constraint con un controlador arduino asociado (dcha.).

El prototipo de la ventana para añadir un **TAC** (figura 12) cuenta con dos listas, una para seleccionar el **constraint** al que está asociado y otra para seleccionar el **subtoken** conectado al **TAC**. Esta ventana también posee casillas de verificación para seleccionar las manipulaciones del **subtoken** sobre el **constraint** que define el **TAC**.

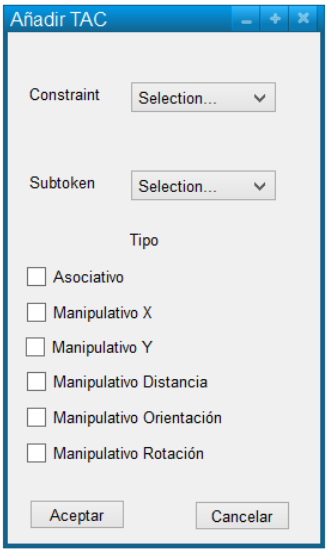


Figura 12. Prototipo de ventana para añadir un TAC.

Versión final

En la versión final del prototipo (figura 13), se cambió la proporción del esquema del estado actual de la mesa, pasando el ratio de aspecto a 16:9, el mismo que tiene la mesa. Este cambio se realizó debido a que las áreas de **constraint** se deformaban a causa de que las proporciones de la mesa cambiaban. El símbolo para marcar los **named tokens** ha pasado a ser un cuadrado blanco con una marca para identificar su orientación y el número de su fiducial.

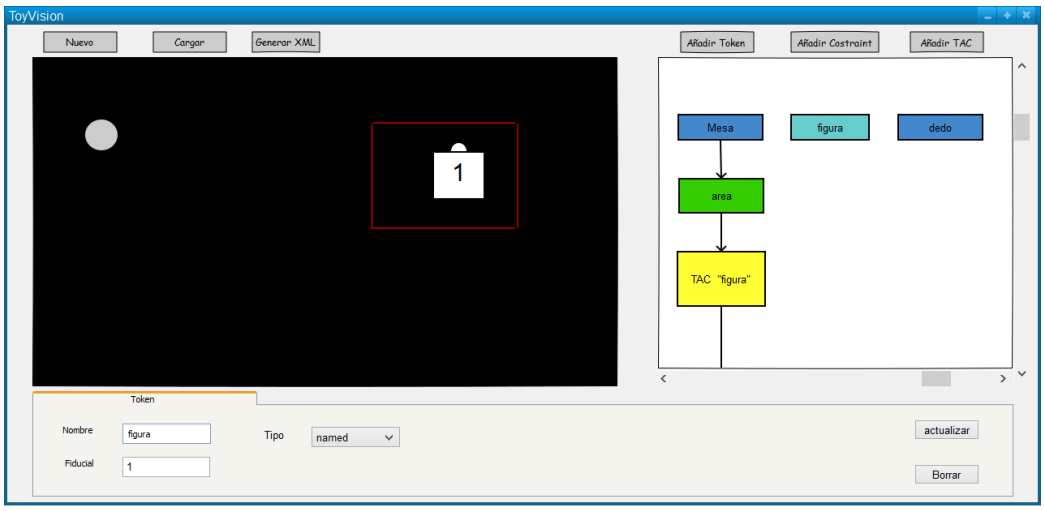


Figura 13. Prototipo final de la pantalla principal.

En el área con el esquema del TUIML del juego, se han eliminado las conexiones entre el **TAC** y el **subtoken**, añadiendo el nombre del mismo en el rectángulo del **TAC**. También se han añadido barras en el lateral derecho y en la parte inferior para aumentar el tamaño disponible en el esquema. Finalmente, se ha cambiado la forma de mostrar el elemento seleccionado, cambiando el color por una tonalidad más clara en este caso.

En la versión final del prototipo de la ventana para añadir un **token** (figura 14) se han eliminado los siguientes campos: ancho, alto y copias debido a que no eran necesarios y con el objetivo de simplificar la ventana.

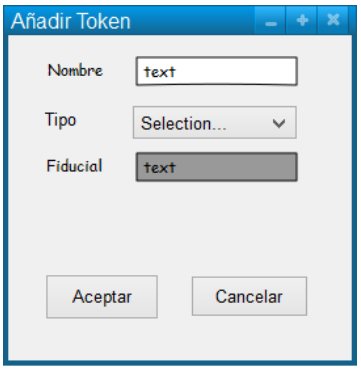


Figura 14. Prototipo final de la ventana para añadir un token.

En cuanto al prototipo de la ventana para añadir un **constraint**, en su versión final (figura 15) se ha cambiado la forma de selección del tipo, pasando a ser dos casillas de opción, una para seleccionar el tipo pasivo y otra para activo.

En la ventana para añadir un **constraint** asociado a un área (figura 15 izq.) se ha añadido la casilla de verificación "visible", para indicar si el **constraint** es visible o no, ya que es uno de los atributos que definidos en el TUIML.

En el caso de la ventana para añadir un **constraint** asociado a un controlador arduino (figura 15 dcha.) se han realizado varios cambios para facilitar la introducción de datos. El campo "num. Arduino" ha pasado de ser un campo de texto a una lista con las placas conectadas al sistema. El campo "actuador" se ha renombrado, pasando a llamarse "componente". El campo "puerto" ha cambiado de nombre por "pin" y la introducción del dato ha pasado a realizarse escogiendo el pin de una serie de casillas de opción. Finalmente se ha añadido un área para probar el controlador asociado al **constraint**.

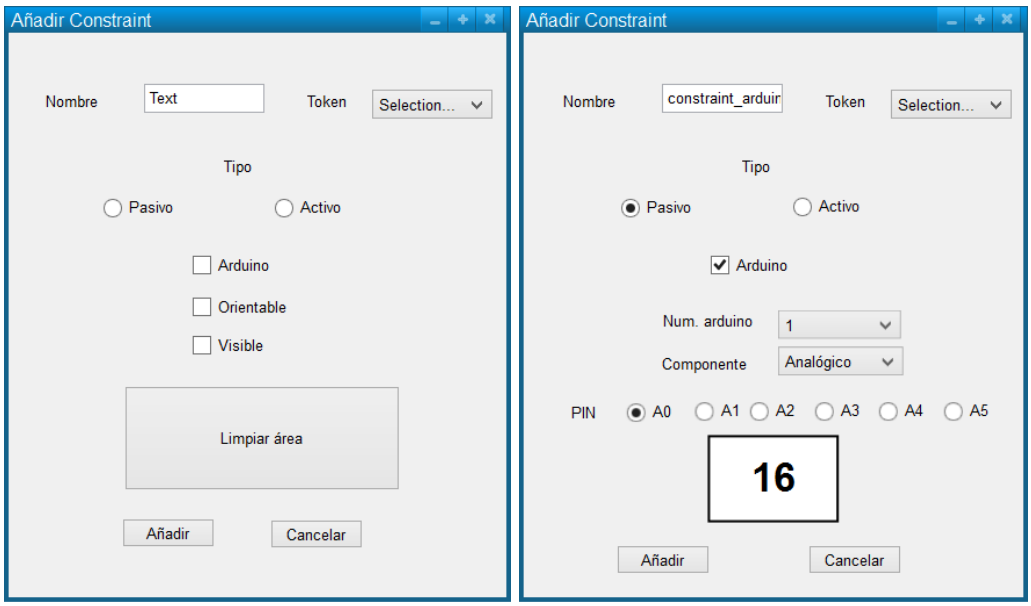


Figura 15. Prototipo final de la ventana para añadir un constraint asociado a un área (izq.) y uno asociado a un controlador Arduino (dcha.).

En el prototipo final de la ventana para añadir un **TAC** (figura 16), para facilitar la interpretación de las manipulaciones, se han sustituido los nombres por los iconos que las representan.

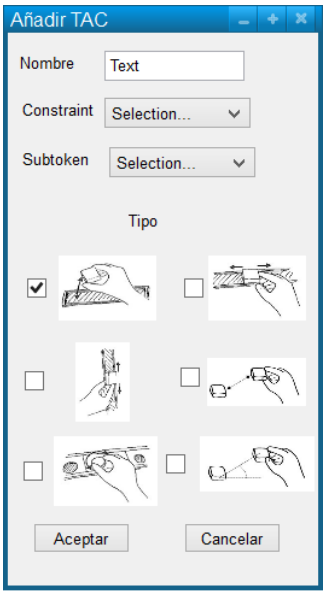


Figura 16. Prototipo final de la ventana para añadir un TAC.

3.2. Implementación

En este apartado se exponen las principales decisiones tomadas a la hora de implementar el editor. Entre ellas, se encuentran tanto el lenguaje y las librerías utilizadas como las clases que la componen.

La implementación del editor se realizó utilizando el lenguaje Processing [ProcessingWeb]. Este lenguaje de programación está basado en java, aunque con algunos cambios que, entre otras cosas, facilitan el dibujo de gráficos por pantalla. Processing también dispone de un gran conjunto de librerías que aportan diversas funcionalidades. A continuación se comentan brevemente las librerías utilizadas en el editor.

- **Arduino Processing [ArduinoProcessingWeb].** Esta librería permite la comunicación de la aplicación con el controlador arduino. También permite su configuración. Se utiliza para probar los sensores y actuadores de los **constraints** que los llevan asociados.

- **G4P [G4PWeb].** La librería G4P proporciona controles para la interfaz gráfica de usuario y funciones personalizables de gestión de eventos. Esta librería se ha utilizado para crear y gestionar las ventanas para añadir elementos, las zonas de edición y los botones de los dos menús de la pantalla principal del editor. Se eligió utilizar esta librería en vez de otras también consultadas como Guido [GuidoWeb], o ControlP5 [ControlP5Web], por la variedad de los controladores disponibles.

- **TUIO [TUIOWeb].** Esta librería implementa un cliente para el protocolo TUIO. En el editor se utiliza para la comunicación con NIKVision a través de ReacTIVision. Esta

comunicación permite conocer en todo momento los objetos que se encuentran colocados sobre la mesa, y su posición y orientación (en caso de que lleven un fiducial).

En el diagrama de clases (figura 17), la clase principal del editor es la clase *editor_grafico*, que contiene el bucle principal de la aplicación. Cada elemento del modelo, ya sea **token**, **constraint** o **TAC**, tiene una instancia de su clase correspondiente, organizándose de manera jerárquica. La parte de la interfaz gráfica tiene su clase para la pantalla principal (clase *gui*) y una para cada tipo de ventana y cada tipo de zona de edición. También hay una clase para las placas arduino, creándose una instancia por cada placa que haya conectada. Por último, se ha implementado una clase *point* para representar los vértices de las áreas de los **constraints**.

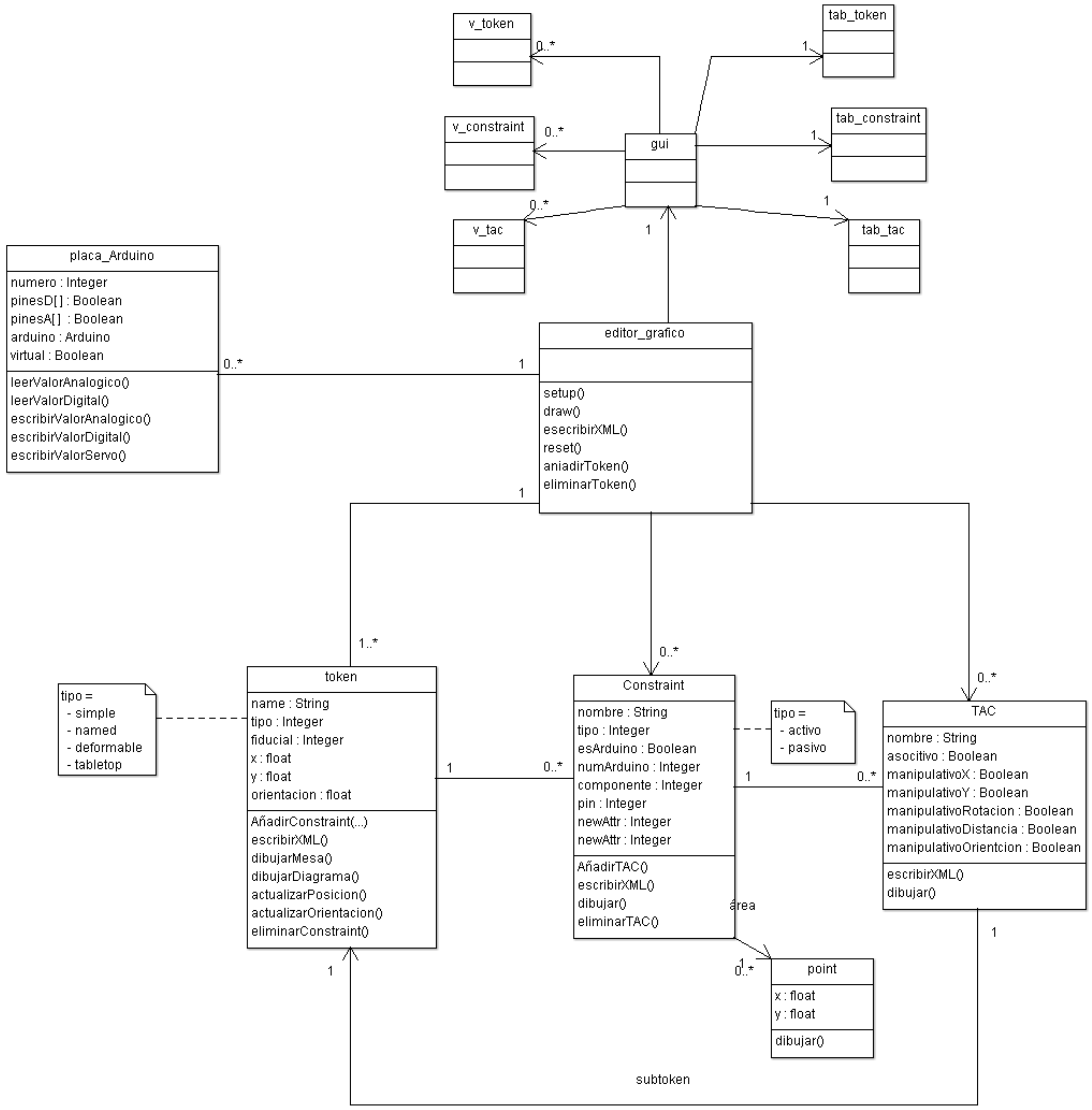


Figura 17. Diagrama de clases.

Clase *editor_grafico*

Se trata de la clase principal del editor. Contiene las constantes necesarias para dibujar el editor y especificar los tipos de algunos elementos. Las principales variables que contiene esta clase son las siguientes:

- **puntos:** *ArrayList* de objetos de la clase *point*. Al seleccionar en el área de la mesa en el editor para definir un área de **constraint**, el vértice correspondiente a ese punto se añade a esta lista.
- **tokens:** *ArrayList* de objetos de la clase **token**. Contiene una lista con todos los **tokens** del TUIML del juego.
- **allConstraints:** *ArrayList* de objetos de la clase **constraint**. Contiene una lista con todos los **constraints** del TUIML del juego.
- **allTACs:** *ArrayList* de objetos de la clase *TAC*. Contiene una lista con todos los **TACs** del TUIML del juego.
- **arduinos:** *ArrayList* de objetos de la clase placa_arduino. Contiene una lista con todas las placas de arduino conectadas al editor.
- **xml:** Objeto de la clase XML. Esta variable se utiliza para cargar y guardar un juego.

Los métodos más importantes implementados en esta clase son los siguientes:

- **setup():** Este método se ejecuta al iniciar el editor. Realiza la configuración inicial.
- **draw():** Contiene el bucle principal del programa.
- **aniadirToken(*token* tk):** Método para añadir un **token** al modelo.
- **eliminarToken(*int* idx):** Método para eliminar un **token** al modelo.
- **escribirXML():** Método para generar el fichero toys.xml.
- **reset():** Método para resetear el editor y crear un juego nuevo.
- **mousePressed():** Método para capturar eventos del ratón sobre la pantalla principal. Se utiliza para dibujar áreas sobre la mesa y para seleccionar elementos del TUIML en el diagrama.

También incluye los métodos relacionados con el protocolo TUIO que gestionan los eventos al añadir o mover un **token** sobre la mesa.

Clase *placa_arduino*

La clase *placa_arduino* se encarga de los controladores arduino. Existe una instancia de esta clase por cada placa conectada al host. Esta clase contiene las siguientes variables:

- **num**: Variable de tipo entero. Contiene el número de la placa para poder identificarla.

- **pinesD**: *Array* de tipo *boolean* que representa los pines digitales de la placa. Si el pin ya está utilizado por un **constraint** el booleano correspondiente valdrá *true*. En caso contrario valdrá *false.*

- **pinesA:** *Array* de tipo *boolean* que representa los pines analógicos de la placa. Si el pin ya está utilizado por un **constraint** el booleano correspondiente valdrá *true*. En caso contrario valdrá *false.*

- **a:** Variable de la clase *arduino.* Se utiliza para la comunicación con la placa arduino.

- **virtual:** Variable de tipo *boolean* que indica si la placa está conectada o no. Se utiliza para simular placas virtuales a la hora de cargar juegos.

Los métodos incluidos en esta clase, a parte de los métodos *get* y *set* de las variables anteriormente mencionadas, son los siguientes:

- **leerValorAnalogico(*int* pin):** Lee un valor analógico del pin que se pasa como parámetro.

- **leerValorDigital(*int* pin):** Lee un valor digital del pin pasado como parámetro.

- **escribirValorAnalogico(*int* pin, *int* valor):** Escribe en el valor analógico indicado en el pin correspondiente.

- **escribirValorDigital(*int* pin, *int* valor):** Escribe en el valor digital indicado en el pin correspondiente.

- **escribirValorServo(*int* pin, *int* valor):** Escribe en el valor indicado en el servo conectado al pin correspondiente.

- **limpiarPlaca():** Pone a *false* todos los pines de la placa. Se utiliza al crear un juego nuevo.

- **esVirtual():** Devuelve cierto si la placa es virtual(variable *virtual= true).*

Clase *token*

Esta clase contiene toda la información y los métodos asociados a un **token**. Existe una instancia por cada **token** definido en el juego. Las variables de esta clase son las siguientes:

- **indice:** *Int* que contiene la posición del **token** en la lista de **tokens** de la clase principal.

- **nombre:** *String* que contiene el nombre del **token**.

- **fid:** Variable de tipo *int* que contiene el número que identifica el fiducial del token. Sólo se usa en **named tokens.**
- **tipo**: *Int* que indica el tipo de token.
- **x**: *Float* que indica la posición en el eje x del **token** en la mesa.
- **y**: *Float* que indica la posición en el eje y del **token** en la mesa.
- **orientacion:** *Float* que indica la orientación del **token** en la mesa.
- **restricciones**: *ArrayList* que contiene todos los **constraints** asociados a ese **token**.

Los métodos más importantes incluidos en esta clase son los siguientes.

- **aniadirConstraint(*Constraint* c):** Método para añadir un constraint asociado a ese **token**.
- **eliminarConstraint(*Constraint* c):** Método para eliminar un constraint asociado a ese **token**.
- **dibujarMesa():** Método para dibujar el **token** en el área de la mesa.
- **dibujarDiagrama():** Método para dibujar el **token** en el TUIML del juego.
- **escribirXML():** Método para escribir el código xml asociado a ese **token**.

Clase *Constraint*

La clase **Constraint** incluye los atributos y las funciones relativas a un **Constraint**. Al igual que con la clase **token**, existe una instancia por **Constraint** en el modelo. Los atributos de esta clase son los siguientes:

- **índice:** *Int* que contiene la posición del **constraint** en la lista de **constraints** de la clase principal.
- **padre**: Variable de la clase ***token*** que contiene el toque al que está asociado el **constraint**.
- **nombre**: S*tring* con el nombre del **constraint***.*
- **tipo**: Variable de tipo *boolean* que indica el tipo del **constraint** (activo o pasivo).
- **es_arduino**: *Boolean* que indica si el **constraint** está asociado a un controlador arduino o no.
- **Visible**: Este *boolean* indica si el **constraint** es visible o no.
- **TACs**: *ArrayList* de objetos de la clase ***TAC*** que contiene todos los **TACs** asociados a ese **constraint**.

- **orientable**: *Boolean* que indica si el **constraint** es orientable o no.
- **area**: *ArrayList* de objetos de la clase *point* que contiene la lista de vértices que componen el área asociada a ese constraint.
- **num_arduino**: Variable entera que almacena el número de la placa arduino al que está conectado el **constraint**.
- **Puerto**: Variable de tipo *int* que indica el pin de la placa arduino al que está asociado el **constraint.**
- **Componente**: Variable entera que guarda el tipo de componente al que este asociado el **constraint.**

En esta clase se incluyen los siguientes métodos:

- **aniadirTAC(TAC e):** Método para añadir un **TAC** asociado a ese **constraint**.
- **borrarTAC(TAC t):** Método para eliminar un **TAC** asociado a ese **constraint**.
- **dibujarArea():** Método para dibujar el área del **constraint** en la zona de la mesa.
- **dibujarDiagrama():**Método para dibujar el **constraint** en el TUIML del juego.
- **escribirXML():** Método para generar el código XML asociado a ese **constraint**.

Clase TAC

Esta clase contiene toda la información y los métodos asociados a un **TAC**. Existe una instancia por cada **TAC** definido en el juego. Los atributos de esta clase son:

- **padre**: Variable de tipo ***constraint*** que contiene el **constraint** al que está asociado el **TAC**.
- **índice:** *Int* que contiene la posición del **TAC** en la lista de **TACs** de la clase principal.
- **Asociativo**: *Boolean* que indica si el **TAC** es asociativo. Esta variable es siempre *true*.
- **manipulativoX:** *Boolean* que indica si el **TAC** es manipulativo en el eje X.
- **manipulativoY**: *Boolean* que indica si el **TAC** es manipulativo en el eje Y.
- **manipulativoRot**: *Boolean* que indica si el **TAC** es manipulativo en rotación.
- **manipulativoDist**: *Boolean* que indica si el **TAC** es manipulativo en distancia.
- **manipulativoOrient**: *Boolean* que indica si el **TAC** es manipulativo en orientación.
- **subtoken**: Variable de tipo ***token*** que indica el subtoken del **TAC**.

Los métodos que contiene la clase TAC, a parte de los métodos get y set para las variables son los siguientes.

- **dibujar():** Dibuja el TAC en el diagrama del TUIML.
- **escribirXML():** Genera el código XML asociado al **TAC**.

Clase point

Esta clase representa un punto y es la más simple del editor. Contiene las coordenadas que componen un punto y los métodos asociados al mismo. Las variables de esta clase son:

- **x:** Variable de tipo *float* que contiene la coordenada x del punto.
- **y**: Variable de tipo *float* que contiene la coordenada y del punto.

A parte de los métodos *get* y *set* de las coordenadas y el constructor, esta clase contiene un único método:

- **dibujar():** dibuja el punto en las coordenadas x e y de la mesa.

Clase *gui*

Esta clase gestiona parte de la interfaz gráfica de la pantalla principal. Contiene todas las variables relativas a los controles que aparecen en ésta, así como las rutinas de gestión de los eventos generados por los mismos.

Clase *v_token*

Esta clase gestiona la ventana para añadir un nuevo **token**, incluyendo todos los controles de la misma y las rutinas de gestión de los eventos asociados a éstos.

Clase *v_constraint*

La clase *v_constraint* gestiona la ventana para añadir un nuevo **constraint**, incluyendo todos los controles de la ventana y las rutinas de gestión de los eventos asociados a éstos.

Clase *v_tac*

Esta clase se encarga de gestionar la ventana para añadir un nuevo **TAC**, incluyendo todos los controles de la ventana y las rutinas de gestión de los eventos asociados a éstos.

Clase *tab_token*

Esta clase contiene todas las variables y métodos para gestionar la zona de edición de un **token**.

Clase tab_constraint

La clase *tab_constraint* se encarga de gestionar la zona de edición de un **constraint**. Contiene todos los controles y los métodos de gestión de eventos para para modificar la información asociada a un **constraint**.

Clase tab_tac

Esta clase se encarga de gestionar la zona de edición de un **TAC**, incluyendo todos los controles de la ventana y las rutinas de gestión de los eventos asociados a éstos.

.

4. Resultados obtenidos

En este apartado se muestra el editor final, exponiendo las decisiones tomadas para cada una de sus partes, y se explica el diagrama de navegación del editor.

La pantalla principal, que se puede ver en la figura 18, se divide en 5 áreas. En la parte superior se sitúan los botones relativos al proyecto y los de añadir nuevos elementos al proyecto.

- **Menú de gestión del juego.** Estos botones sirven para crear un nuevo juego, cargar uno existente o guardar el juego actual.

- **Menú de gestión del esquema.** En este menú se encuentran los botones para añadir nuevos elementos al modelo del TUIML.

- **Estado actual de la mesa.** En esta zona se muestra el estado actual de las mesa.

- **TUIML del juego.** En la zona del esquema del juego se muestra el TUIML del juego especificado hasta el momento.

- **Zona de edición de elementos.** En esta zona se permite editar la información asociada a un elemento.

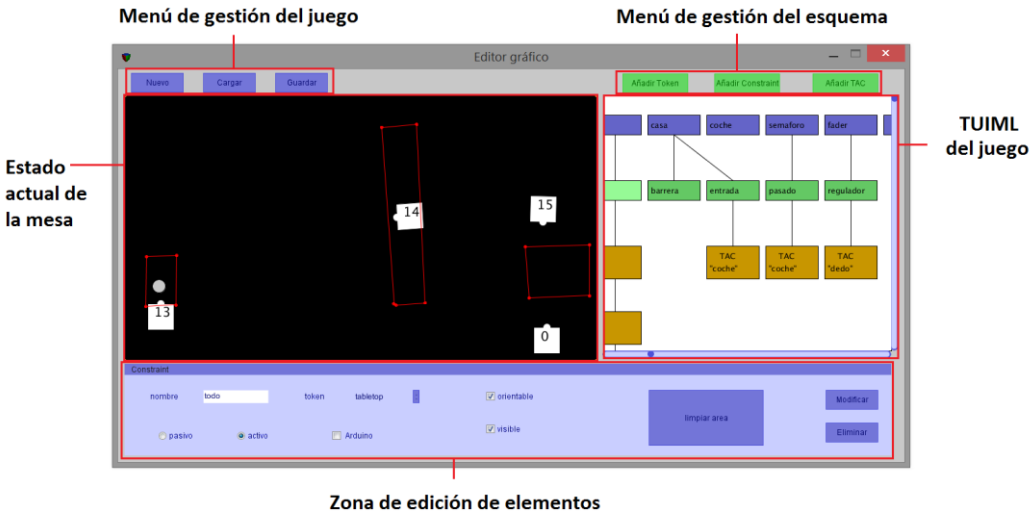


Figura 18. Pantalla principal de la versión final del editor.

En la zona superior izquierda se sitúan los botones del menú de gestión del juego. Los botones que conforman este menú son los siguientes:

- **Nuevo**. Este botón sirve para crear un juego nuevo.
- **Cargar.** Con este botón se puede cargar un juego ya existente. Al hacer clic sobre él se abre un diálogo donde se puede seleccionar la ruta done se encuentra el fichero
- **Guardar.** Este botón permite guardar el juego actual. Al pulsar sobre él se abre un cuadro de diálogo para especificar el nombre la ruta del fichero. Una vez seleccionado, el juego se guarda en formato XML.

En la parte superior izquierda se encuentra el menú de gestión del esquema. Éste permite añadir nuevos elementos al TUIML del juego. Los botones de los que se compone este menú son los siguientes:

- **Añadir Token.** Este botón permite añadir un token al TUIML del juego.
- **Añadir constraint**. Con este botón se puede añadir un constraint al TUIML.
- **Añadir TAC.** Este botón sirve para añadir un TAC al TUIML.

En la parte central a la izquierda aparece el estado actual de la mesa (detalle en la figura 19). Al colocar un objeto sobre el tabletop, éste se mostrará en el área en la misma posición que en la mesa, así como las áreas de los **constraints** asociados al mismo. La representación de cada token depende del tipo del mismo:

- **Named token.** Los objetos con un fiducial asociado se muestran en blanco con forma cuadrada con un semicírculo en la parte superior. El número del fiducial asociado al objeto se muestra dentro del cuadrado que lo representa. Dado que los **named token** poseen orientación, la representación de los mismos se encuentra rotada en función de la orientación del **token**.
- **Token simple.** se representa con un círculo de color gris.

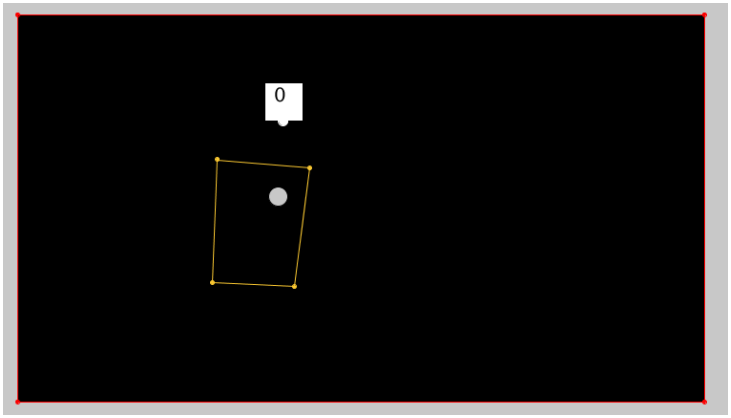


Figura 19. Captura del área de la mesa.

Las áreas de **constraint** asociadas a los **tokens** que se encuentran en la mesa se muestran en color rojo. Estas áreas se posicionan en función de las coordenadas de **token** al que están asociadas y la orientación, si se trata de un **constraint** orientable. El dibujo de una nueva área de **constraint** se realiza seleccionando sobre la zona de la mesa para definir los vértices que la componen. Durante la definición del área ésta se muestra en color amarillo, volviéndose blanco cuando el vértice a definir cierra el área. Un área también se representa de este color cuando se selecciona el **constraint** al que está asociada.

En la zona central a la derecha se sitúa la zona con el esquema del juego (figura 20). En él aparece el TUIML definido por el usuario para el juego actual. El usuario puede seleccionar cualquier elemento del esquema haciendo clic sobre él. Los **tokens** y los **constraints** definidos aparecen en el esquema ordenados de forma horizontal en el mismo orden en el que han sido definidos. Sin embargo, los **TACs** se ordenan de forma vertical debajo de los **constraints** a los que están asociados, ordenándose los **TACs** asociados a un mismo **constraint** en el orden en el que han sido definidos. Cada **TAC** en el esquema se muestra con el nombre del **subtoken** al que están asociados.

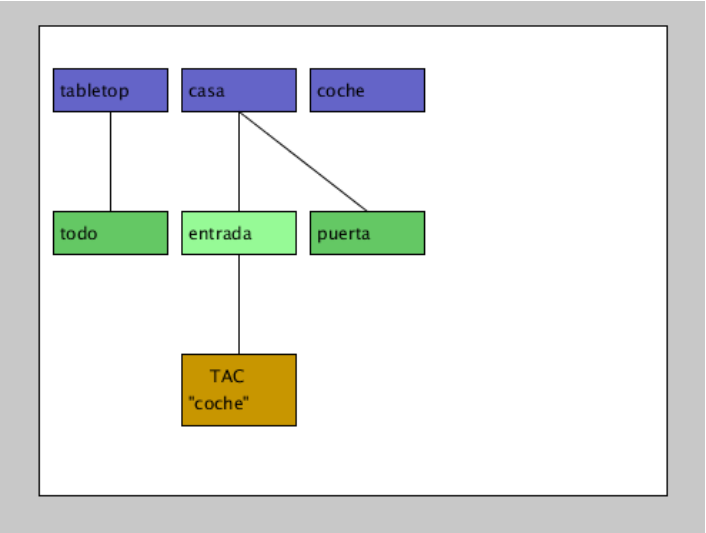


Figura 20. Área del esquema TUIML del juego.

En la parte inferior de la pantalla principal se encuentra la zona de edición. Ésta permite modificar y eliminar los elementos definidos en el TUIML. Al seleccionar un elemento, se muestra en esta zona la información de ese elemento para poder ser editada por el usuario.

Al seleccionar un **token**, en la zona de edición (figura 21) se muestra su nombre, su tipo y, si es un **named token**, el fiducial asociado a éste. En la zona derecha se muestran los botones para modificar o eliminar el **token** seleccionado.

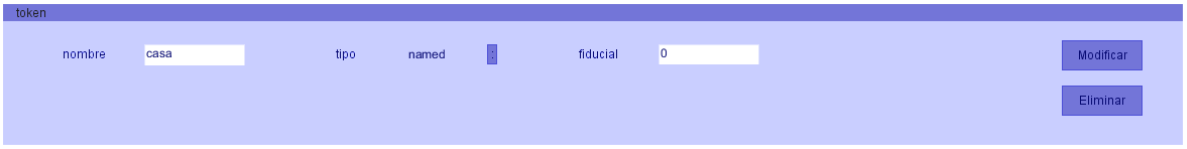


Figura 21. Zona de edición para un token

Si el elemento seleccionado es un **constraint**, la zona de edición mostrada varía según si el **constraint** tiene un área asociada (figura 22 arriba) o un pin de un controlador arduino (figura 22 abajo). Los componentes que aparecen en ambos casos son los siguientes:

- Botón para modificar el **constraint.**
- Botón para eliminar el **constraint.**
- Cuadro de texto para modificar el nombre del **constraint.**
- Lista con los tokens para seleccionar aquel al que esté asociado el **constraint.**
- Selección del tipo de **constraint**.
- Selección de si está asociado a un controlador arduino o no.

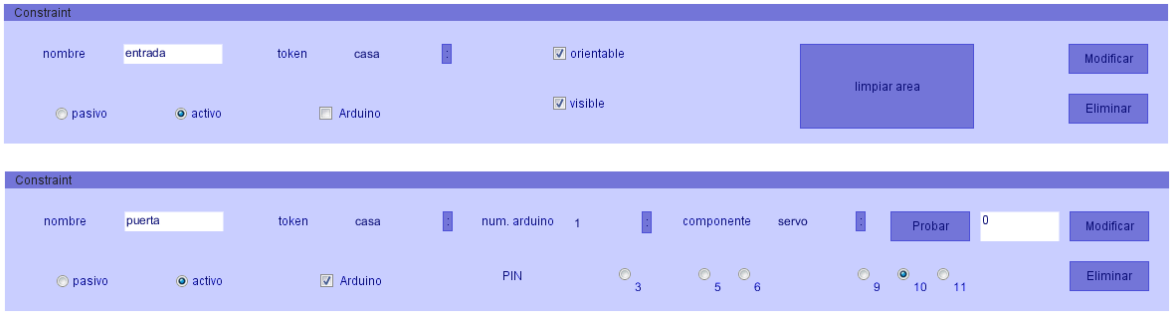


Figura 22. Zona de edición para un Constraint con área (arriba) y uno asociado a un controlador Arduino (abajo).

En el caso de que se trate de un **constraint** con un área asociada aparecen en pantalla los siguientes controles:

- Selección de si se trata de un **constraint** visible o no.
- Selección de si el **constraint** es orientable o no.
- Botón “limpiar área” para borrar el área asociada y poder redibujarla.

Cuando el **constraint** está asociado a un componente electrónico mediante arduino, de éste se muestran los siguientes controles:

- Selección del número de la placa arduino al que está conectado.
- Tipo de componente que lleva asociado.
- Pin de arduino al que está conectado.

Los pines de arduino que se muestran varían en función del tipo de **constraint** y componente.

- **Constraint pasivo**.
    - Si el componente es analógico se muestran los pines analógicos A0-A5.
    - Si el componente es digital se muestran los pines digitales 2-13.
- **Constraint activo**.
    - Si el componente es analógico o servo, se muestran el subconjunto de los pines digitales 3, 5, 6, 9, 10 y 11.
    - Si el componente es digital se muestran los pines digitales 2-13.

Los pines ocupados por otros elementos se muestran en color rojo y no pueden ser seleccionados.

Por último, también se puede comprobar el funcionamiento del componente conectado:

- **Sensor** (**constraint pasivo**): se muestra el valor recibido en el pin especificado por el usuario.
- **Actuador** (**constraint activo**): los botones mostrados para probarlo varían en función del tipo de componente.
    - **Componente digital**: se muestra un interruptor para probarlo.
    - **Componente analógico o servo:** se muestra un campo de texto para indicar el valor a enviar al actuador y un botón para enviar ese valor.

Al seleccionar un **TAC**, en la zona de edición (figura 23) se permite cambiar el **constraint** del que es hijo, así como en **subtoken** al que está asociado. También se pueden seleccionar la asociación del **subtoken** al área de **constraint** y las manipulaciones permitidas sobre el **subtoken**. Estas manipulaciones son las siguientes:

- Asociación del **subtoken** al **constraint**.
- Desplazamiento en el eje X
- Desplazamiento en el eje Y
- Rotación del **subtoken**
- Distancia del **subtoken** al centro del área de **constraint**
- Orientación del **subtoken** dentro del **constraint**.

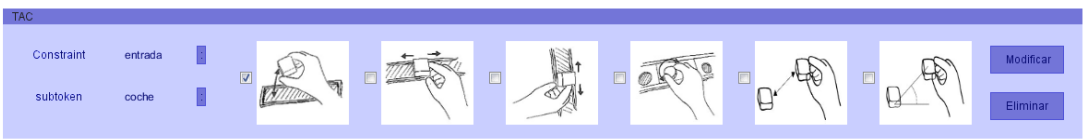


Figura 23. Zona de edición para un TAC.

Ventana añadir token

Esta ventana (figura 24) se abre al pulsar en el botón de añadir **token.** En ella se encuentra un formulario que permite la introducción de los datos del **token**.

Figura 24. Ventana para añadir un token.

Los elementos de esta ventana son los siguientes:

- **Nombre**. Campo de texto para introducir el nombre del **token**.
- **Fiducial.** Campo de texto para introducir el fiducial. Este campo solo es visible si el tipo del **token** es **named**.
- **Tipo.** Lista desplegable para seleccionar el tipo del **token**.
- **Añadir**. Botón para aceptar la adición del **token** al TUIML del juego.
- **Cancelar.** Botón para cancelar la adición del **token** al TUIML del juego.

Ventana añadir Constraint

Los componentes que aparecen en esta ventana varían en función de si el constraint tiene un área asociada o un controlador arduino. En ambos casos aparecen los siguientes elementos:

- **Nombre.** Campo de texto para introducir el nombre del **constraint**.
- **Token.** Lista desplegable para seleccionar el token al que está asociado.
- **Tipo.** Selección del tipo de constraint (pasivo o activo).
- **Arduino.** Selección que permite indicar si este trata de un constraint con área asociada o está asociado a un controlador electrónico.

Si se trata de un **constraint** con un área asociada (figura 25), se muestran las siguientes opciones:

- **Orientable.** Permite seleccionar si el área del **constraint** es orientable o no. En caso de que se seleccione esta opción los vértices que definen el área rotarán en función de la orientación del **token**.
- **Visible.** Permite indicar si el área del **constraint** es visible o no.
- **Limpiar Área.** Este botón permite borrar el área definida para poder volver a dibujarla.

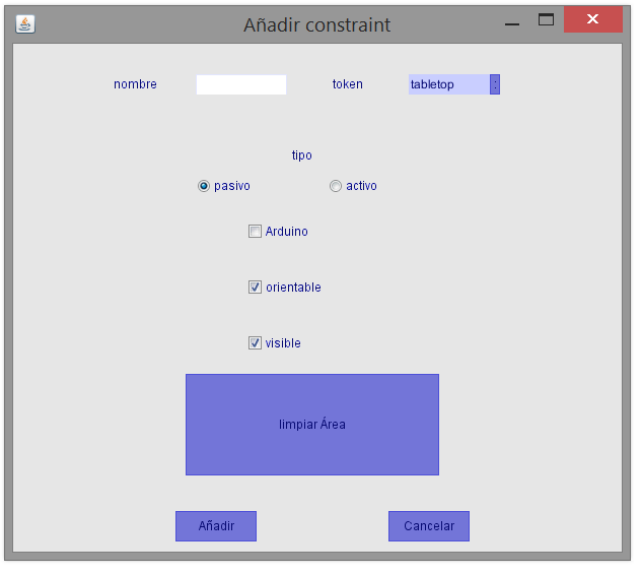


Figura 25. Ventana para añadir un constraint con área.

En el caso de que el **constraint** esté asociado a un sensor o un actuador por medio de una placa arduino, se mostrarán las opciones para definir un **constraint** de este tipo (figura 26). Los componentes que se muestran en este caso son los siguientes:

- **Num. Arduino**. Debido a que el editor tiene soporte para varias placas el usuario debe elegir el número de placa arduino a la que estará asociado mediante esta lista desplegable.
- **Componente**. El tipo del componente electrónico se selecciona mediante este desplegable. El contenido de la lista dependerá en función de si se trata de un sensor (**constraint pasivo**) o un actuador (**constraint activo**):
  - **Constraint pasivo**. El usuario elige entre un sensor analógico o uno digital.
  - **Constraint activo.** Las opciones para este tipo de constraint son: actuador analógico, actuador digital o servo.
- **Pin.** El usuario debe elegir el pin al que estará conectado el componente arduino. Los pines ya ocupados por otros **constraints** se muestran en color rojo para señalar

que no están disponibles. Los pines que se muestran para seleccionar varían en función del tipo de **constraint** y de componente:

- **Sensor analógico.** se muestran los pines A0-A5.
- **Sensor y actuador digital.** Se muestran los pines 2-13.
- **Actuador analógico y servo.** Se muestran un subconjunto de los pines digitales. En concreto, se muestran los siguientes: 3, 5, 6, 9, 10 y 11.

Debajo de los pines, se encuentra una zona donde se puede probar el componente electrónico.

- En el caso de un sensor, se muestra el valor recibido por el arduino en ese pin.
- Si el componente es un actuador digital, aparece un interruptor que al accionar encenderá o apagará el actuador.
- Si por el contrario es un actuador analógico o un servo aparecerá un campo de texto para introducir el valor a enviar y un botón para enviar dicho valor.

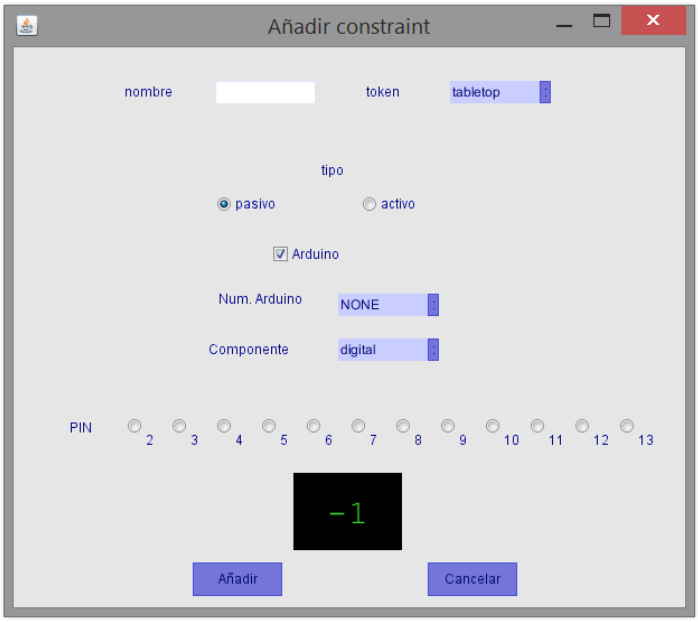


Figura 26. Ventana para añadir un Constraint asociado a un controlador arduino.

Ventana añadir TAC

La ventana para añadir un nuevo **TAC** al modelo (figura 27) se abre cuando el usuario pulsa el botón "añadir **TAC**". En ella aparecen los siguientes componentes:

- **Constraint.** Se selecciona el **constraint** al que está asociado.

- **subtoken**. El usuario debe seleccionar el **token** que se manipulará dentro del **constraint.**
- **Manipulativo**. Se seleccionan las manipulaciones permitidas sobre el **subtoken**.

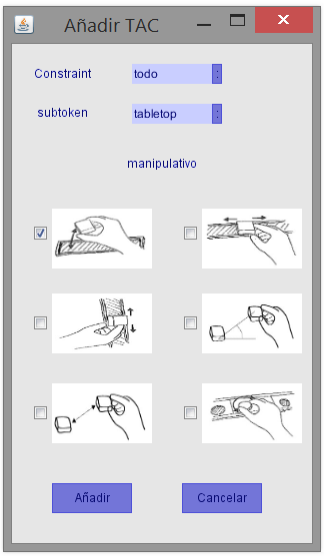


Figura 27. Ventana para añadir un TAC.

5. Evaluación

En este capítulo se muestran las pruebas realizadas al editor y los resultados obtenidos en las mismas. En el primer apartado se presentan las pruebas funcionales realizadas para comprobar el correcto funcionamiento de la aplicación. En el segundo apartado se exponen las pruebas de usabilidad del editor realizadas con usuarios.

5.1. Pruebas funcionales

Como pruebas funcionales, se implementaron varios juegos simples utilizando el editor para comprobar el correcto funcionamiento de las diferentes partes del mismo y detectar errores para poder solucionarlos. En este apartado se muestra el proceso de creación de un juego, en concreto del juego **Dragon's Cave**, a modo de ejemplo. Este juego incluye **tokens** de todos los tipos soportados (NIKVision aún no puede detectar **tokens deformables**), y **constraints** con área asociada y con un controlador arduino.

Para concluir las pruebas y verificar la funcionalidad del editor, un experto desarrolló una versión de la aplicación metaDESK desde cero, definiendo el TUIML de éste, utilizando el editor gráfico.

Los ficheros toys.xml generados para estos juegos se pueden ver en el anexo B.

Dragon's Cave

Dragon's Cave es el juego con el TUIML más completo que se elaborado utilizando el editor gráfico, el cual pude verse en la figura 28.

Figura 28. Juego Dragon's Cave.

Este juego es para uno o dos jugadores (ver piezas en la figura 27), que manejan la figura de un rey (figura 27.12) y un caballero (figura 27.13). El objetivo del juego es matar al dragón (figura 27.14), para lo que se necesita una espada que está guardada en un cofre (figura 27.1). El jugador debe conseguir esta espada para poder matar al dragón con ella. Mientras tanto el dragón, controlado por el sistema, busca al rey y al caballero girando sobre sí mismo por medio de un *servo*, y en cuanto está orientado con alguno de ellos, le lanza una llamarada. El jugador que recibe la llamarada, muere, aunque puede protegerse de las llamas ocultándose tras una columna (figura 27.15). Por otro lado el jugador también puede llamar la atención del dragón distrayéndole de su compañero y así permitir a éste matar al dragón antes de que éste lo descubra.

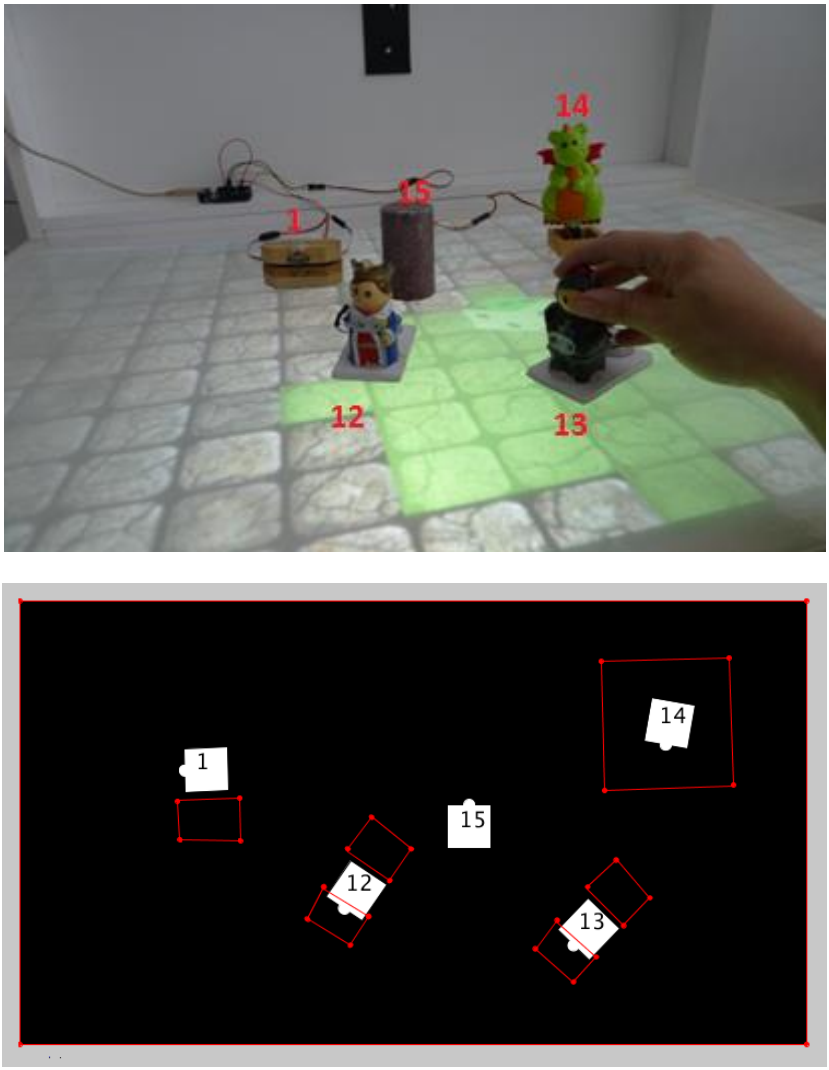


Figura 29. Piezas del juego Dragon's Cave con sus fiduciale (arriba) y su representación en la zona de la mesa del editor (debajo)

El esquema con el TUIML del juego puede verse en la figura 30. En este caso hay dos **constraints** asociados a un controlador arduino, concretamente el **constraint** "tapa" que

controla la apertura de la tapa del cofre (figura 31) y el **constraint** “motor”, que se encarga de hacer girar al dragón. El fichero tos.xml generado para este TUIML se puede ver en el anexo B.1.

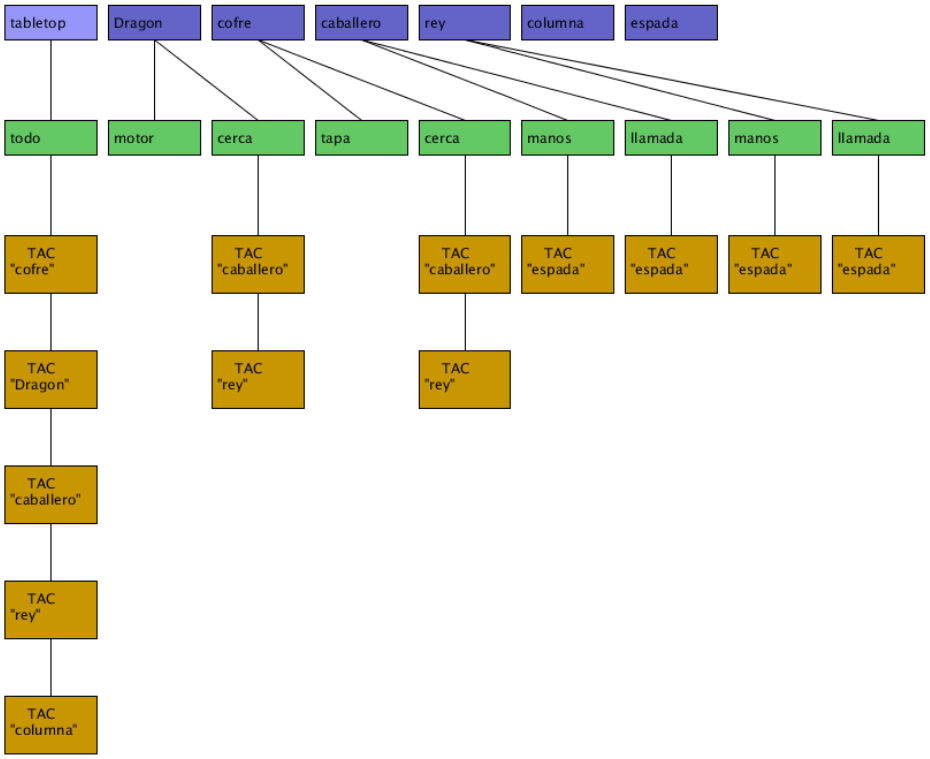


Figura 30. TUIML del juego Dragon’s Cave.

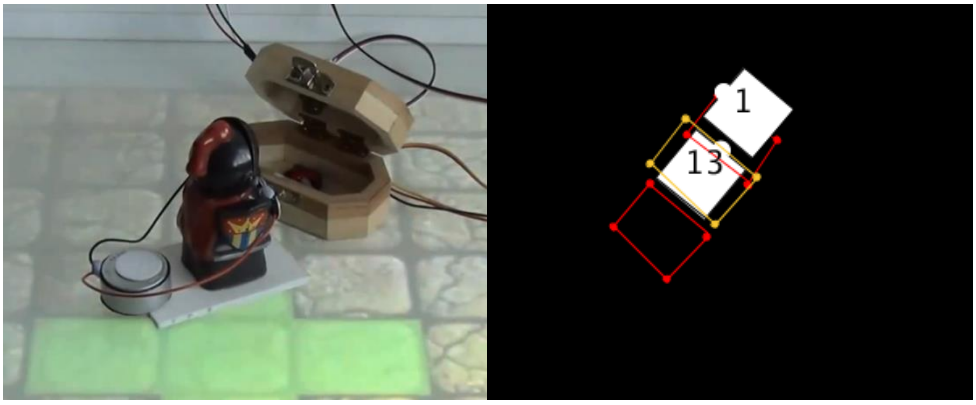


Figura 31. Cofre de Dragon’s Cave abriéndose al estar el guerrero delante (izq.) y la representación en la zona de la mesa del editor (dcha.)

MetaDESK

Como prueba funcional final un evaluador externo desarrolló una versión de la aplicación metaDESK (figura 32) desde cero utilizando el editor gráfico. Esta aplicación permite la manipulación de un mapa mediante la colocación de objetos sobre la superficie del tabletop (ver detalles en el apartado 2.1).

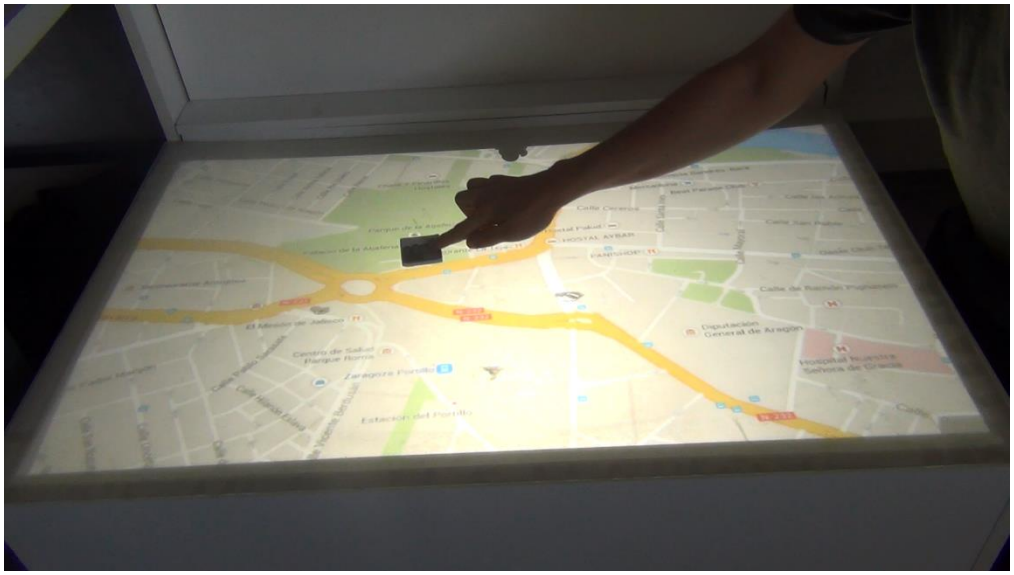


Figura 32. metaDESK.

En metaDESK la superficie del tabletop representa el mapa de Zaragoza. Tal y como se muestra en la figura 33, la aplicación se utilizan objetos que representan monumentos de Zaragoza, en este caso el objeto simboliza la basílica del Pilar (figura 33.1). También hay un objeto que representa el palacio de la Aljafería. Al colocar uno de estos sobre la mesa el mapa se orienta hasta hacerlo coincidir con su localización en el mapa.

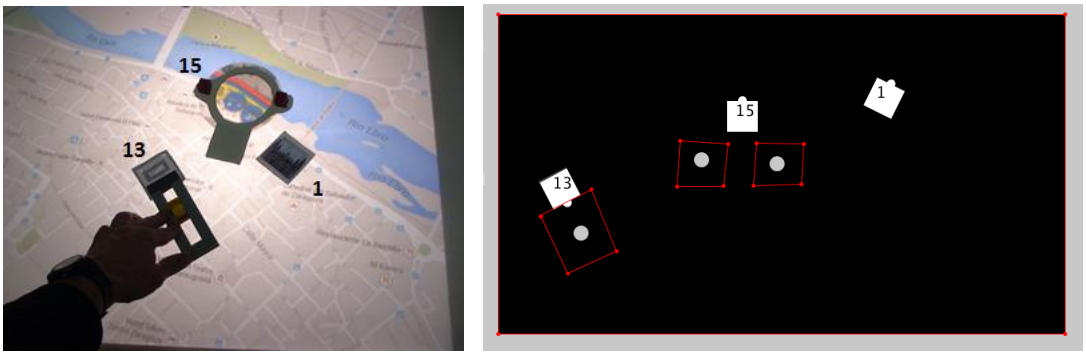


Figura 33. Objetos de metaDESK con sus fiduciales (dcha.) y su representación en el editor.

En la figura se muestra una lupa que contiene dos pulsadores (figura 33.15), uno a cada lado. Al presionarlos se pueden observar a través de la lupa diferentes capas de información del mapa. En concreto, al presionar el pulsador derecho se observa el mapa con la capa de satélite, y al hacerlo en el izquierdo se muestra una capa con las líneas de autobús de la ciudad.

La figura también muestra un *fader* (figura 33.13) que incluye el juego y que permite ampliar y reducir el tamaño del mapa al desplazar verticalmente una pieza sobre él.

En la figura 34 se muestra el esquema con el TUIML de metaDESK, y el área del estado actual de la mesa en el editor, en el que se puede ver cómo se han definido las áreas de constraint. Los **tokens** que aparecen en el esquema son la mesa, las figuras de los monumentos, la lupa, el navegador (*fader*) y el **token simple** "dedo". La mesa tiene asociado el **Constraint** "todo", que contiene los **TACs** con los **subtokens** "pilar", "aljafería", "lupa" y "navegador", estos **TACs** simbolizan qué **tokens** tiene sentido colocar sobre la mesa y que, por tanto, enviará un mensaje al colocarlos. La lupa tiene asociados dos **constraints**, uno para cada pulsador. Cada uno de estos **tokens** tiene un **TAC** con el **token** "dedo" como **subtoken**. El **token** navegador tiene un **Constraint**, zoom, que sirve para cambiar el nivel de zoom del mapa manipulando el **token** "dedo" sobre él. El fichero toys.xml asociado al TUIML de la figura 34 se puede ver en el anexo B.2.

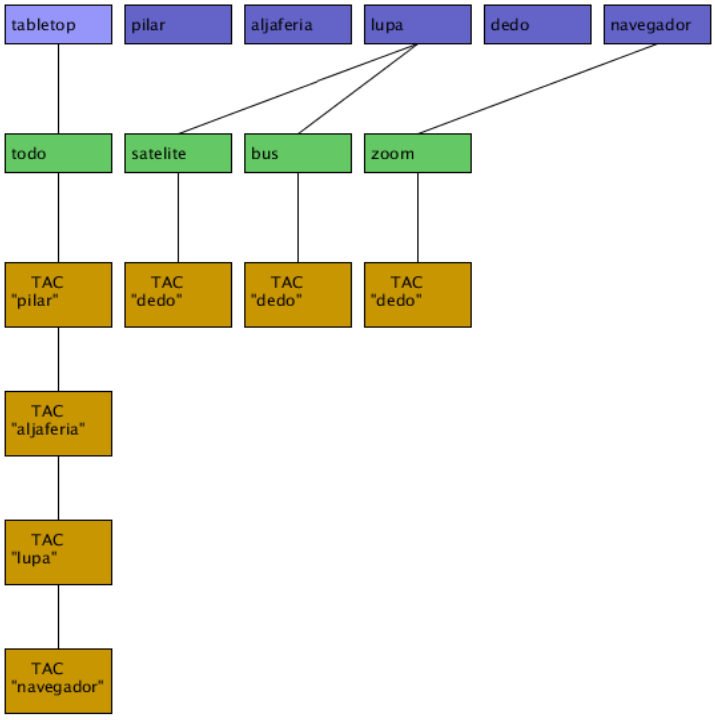


Figura 34. Esquema TUIML de metaDESK.

Tras haber utilizado el editor para elaborar los juegos anteriormente mencionados, además de otros juegos, se comprueba que el editor permite la especificación del TUIML de un juego, además de la realización de pruebas sobre los diferentes sensores y actuadores que participan en el juego, cumpliendo así las funcionalidades establecidas en los requisitos en el apartado 2.2.

5.2. Pruebas de usabilidad

Para evaluar la usabilidad del editor, se realizó una sesión con seis alumnos del máster de ingeniería informática de la Universidad de Zaragoza, divididos en parejas. Ésta consistió en la especificación y elaboración de un juego para NIKVision utilizando el editor. En esta sesión el editor se evaluó conjuntamente con el TUIML y la última versión del toolkit ToyVision.

El juego: Coche y casa con barrera

El juego que los usuarios debían realizar durante la sesión era "coche y casa con barrera" (figura 35). Éste contiene los siguientes elementos:

- Un coche de juguete.
- Un semáforo.
- Una casa con una barrera en la entrada.
- Un *fader.*

El jugador maneja el coche de juguete a través de un par de calles. En el cruce entre ambas calles se encuentra el semáforo que alterna de color entre rojo y verde cada cierto tiempo. El coche debe entrar en la casa para lo que deberá colocarse delante de ella para que la barrera de ésta se abra. La barrera también se puede abrir o cerrar mediante la manipulación del *fader* a modo de mando.

Figura 35. Niños jugando a coche y casa con barrera.

Un ejemplo de TUIML elaborado utilizando el editor que especifica este juego se muestra en la figura 36.

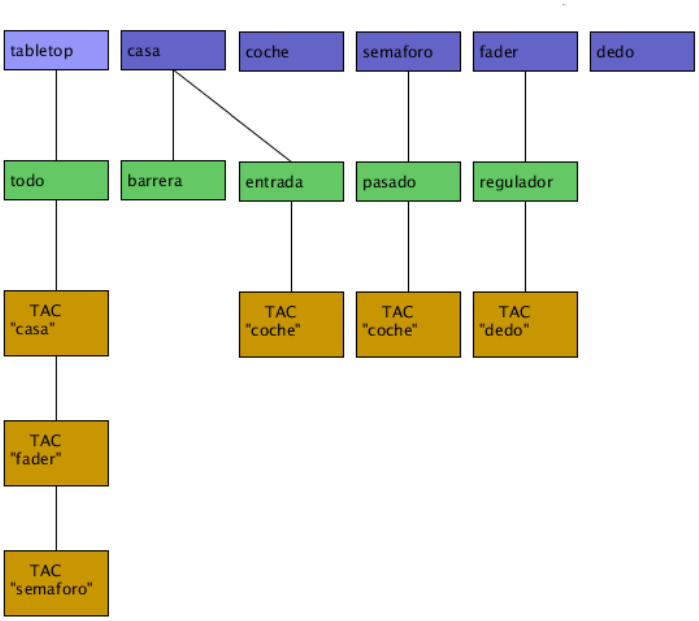


Figura 36. TUIML del juego coche y casa con barrera.

Metodología de la evaluación

La sesión consistió en la elaboración por parejas de un juego para ToyVision. El juego que realizaron era "coche y casa con barrera", el cual se explica a continuación. La sesión se dividió en las siguientes tareas:

- **Tarea 1**. En la primera tarea, y tras serles explicada la dinámica del juego, cada una de las parejas debía elaborar en papel el esquema TUIML del juego.

- **Tarea 2**. La segunda tarea consistía en la elaboración el esquema realizado en la fase anterior usando el editor gráfico conectado a NIKVision y a una placa arduino para obtener el fichero toys.xml del juego. Es en esta fase cuando se evaluó el editor gráfico realizado en este PFC.

- **Tarea 3**. En la tercera tarea las parejas de usuarios debían programar el bucle principal del juego para, finalmente, ejecutarlo y comprobar su correcto funcionamiento.

- **Evaluación.** Tras realizar la sesión, los usuarios rellenaron de manera individual un cuestionario SUS (System Usability Scale) [B86] para obtener la opinión de los usuarios sobre la facilidad de uso del editor y la satisfacción de los mismos.

Durante la sesión

Durante el desarrollo de la sesión se fueron apuntando las observaciones que se iban realizando sobre el trabajo de los usuarios. Concretamente, en la tarea 2, en la que se evaluaba el editor, se anotaron los diferentes fallos que se iban detectando, los cuales se han dividido en tres grupos: fallos técnicos del editor, fallos de usabilidad y comentarios de usuarios.

Fallos técnicos del editor:

- Se observaron fallos a la hora de cargar un juego. Algunos **constraints** definidos como pasivos cambiaban a activos al cargar un fichero. Tras comprobar el código fuente se consiguió solucionar este fallo.
- A la hora de crear un nuevo proyecto, los *scrolls* del esquema TUIML no volvían al estado inicial.

Problemas de usabilidad:

- Algunos usuarios no asociaron el botón "generar XML" con la acción de guardar el proyecto, por lo que se decidió cambiar el texto del botón por "guardar" para que resultase más claro.
- El editor no permite conectar un arduino en mitad de su ejecución, por lo que para añadir una placa es necesario guardar el proyecto y reiniciar el editor. Esto se había decidido así debido a que la librería de arduino utilizada no permite identificar de forma única una placa al desconectarla y volverla a conectar. Este hecho podría dar problemas, asociando a los **constraints** una placa que no corresponde y que tiene conectados otros controladores.

Comentarios de usuarios:

- Un usuario sugirió la posibilidad de cargar una imagen como fondo de la mesa del editor. Esto facilitaría la definición de las áreas de los **constraints** debido a que se tendría una referencia para definir los vértices. Esta opción sería interesante implementarla en trabajos futuros.

Durante la observación del uso del editor se comprobó que todos los usuarios pudieron completar la especificación del juego utilizando el editor y se pudieron observar fallos en éste que posteriormente fueron solucionados.

Después de la sesión

Una vez finalizada la sesión, los usuarios rellenaron un cuestionario SUS (System Usability Scale). Este cuestionario se utiliza para medir de forma rápida y fiable la usabilidad de un sistema y consta de nueve afirmaciones generales sobre el sistema a evaluar, con la que el usuario debe marcar el grado de acuerdo con la afirmación de 1 a 7, siendo 1 estar en total

desacuerdo, y 7 estar totalmente de acuerdo. A continuación se muestran las preguntas incluidas en este cuestionario.

1. Creo que me gustaría utilizar el editor con frecuencia.
2. He encontrado el editor innecesariamente complejo.
3. Pienso que el editor ha sido fácil de usar.
4. Creo que necesitaría soporte técnico de una persona para usar el editor.
5. He visto que las distintas funcionalidades del editor están bien integradas.
6. Creo que la mayoría de la gente aprenderá a usar el editor muy rápido.
7. Encuentro extraña la forma de usar el editor.
8. Mientras usaba el editor, me he sentido confiado.
9. He tenido que aprender muchas cosas para empezar a usar el editor.

A la hora de calcular el resultado, las afirmaciones positivas sobre el editor se valoran con el valor marcado restándole 1, mientras que las afirmaciones negativas se valoran como 7 menos el valor marcado, siendo el valor en el que se está en total desacuerdo el más valorado. Finalmente se suman los valores de las respuestas de cada pregunta y se obtiene la nota total. Dado que el valor máximo que se puede sacar en este cuestionario es 54, se realiza una sencilla regla de 3 para obtener el valor sobre 100.

La figura 37 muestra los resultados obtenidos del cuestionario SUS, los cuales han sido positivos. La nota media obtenida de todos los usuarios ha sido de 77,47. Ésta es superior a 50, por lo que los usuarios han valorado positivamente el editor. Además, todas las notas individuales son superiores al percentil 60, siendo la puntuación dada por el usuario 2 la única por debajo del percentil 70. Analizando las respuestas dadas por este usuario, se observa que las puntuaciones más bajas se han dado para las preguntas 1 (“Creo que me gustaría utilizar el editor con frecuencia”) y 7 (“Encuentro extraña la forma de usar el editor”). La respuesta para ambas se ha situado por encima del punto medio de la escala de medición.

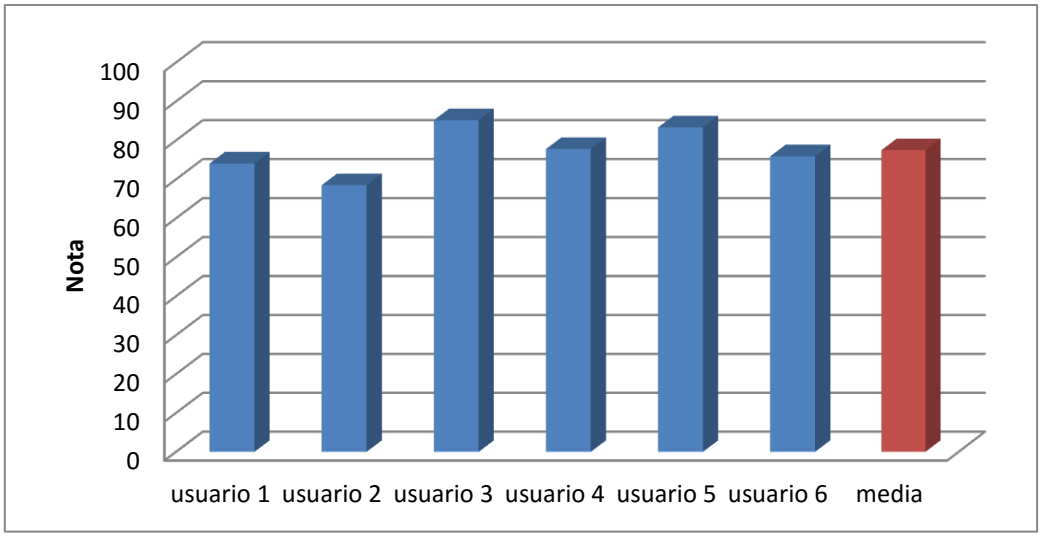


Figura 37. Notas obtenidas en el cuestionario SUS.

En la figura 38 se puede ver las notas medias obtenidas en cada pregunta. En la gráfica se puede comprobar que todas las preguntas han obtenido una puntuación mayor de 60.

En el diagrama se observa que las preguntas 3 (“pienso que el editor ha sido fácil de usar”) y 9 (“He tenido que aprender muchas cosas para empezar a usar el editor”) han sido las que mayor media de valoración han obtenido, siendo ambas superiores a 80. Éstas están relacionadas con la facilidad de uso del editor, por lo que se puede deducir que éste es fácil de usar.

Por otro lado, la nota media más baja ha sido la de la pregunta 5 (“He visto que las distintas funcionalidades del editor están bien integradas.”). El valor medio obtenido para esta pregunta es de 63,9, por encima del percentil 60, por lo que la valoración de esta pregunta ha sido positiva.

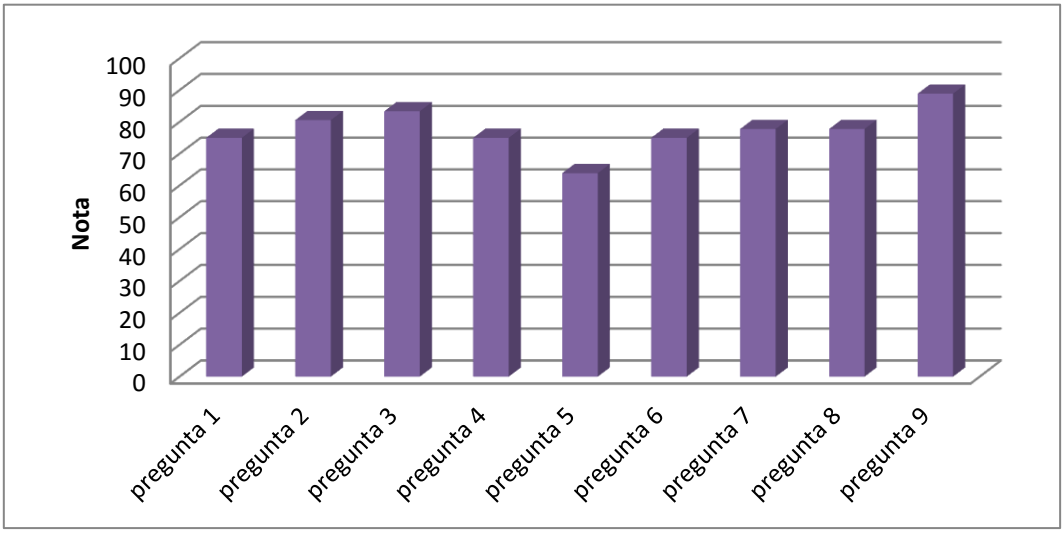


Figura 38. Nota media obtenida de cada pregunta.

Conclusiones

De lo anteriormente escrito, se puede deducir que los resultados de esta evaluación han sido satisfactorios. Durante la observación se detectaron algunos problemas que fueron solucionados posteriormente. Por otro lado, los resultados obtenidos del cuestionario SUS han demostrado una valoración positiva del editor por parte de los usuarios.

6. Conclusiones y trabajo futuro

En este último capítulo se presenta una valoración del trabajo realizado en este Proyecto Fin de Carrera. El primer apartado del capítulo se exponen las conclusiones obtenidas del desarrollo del PFC, incluyendo la consecución de objetivos. En la segunda parte del capítulo se realiza una valoración personal del trabajo realizado. En el tercer apartado se indican posibles trabajos futuros relacionados con el editor gráfico.

6.1. Conclusiones

El objetivo inicial de este Proyecto Fin de Carrera era, tal y como se indica en el apartado 1.2 de esta memoria la creación de un editor gráfico para el *framework* ToyVision que permitiera:

- Definir de forma gráfica y los diferentes elementos de los que se compone el juego.
- Probar los diferentes sensores y actuadores utilizados en él.
- Traducirlo a un lenguaje de marcas entendible por el *framework.*

El editor elaborado en este Proyecto cumple todos los objetivos propuestos al inicio, permitiendo definir de forma gráfica el TUIML del juego a desarrollar. También permite la interacción con el hardware NIKVision y con los diferentes controladores arduino que se conectados al *host*.

Finalmente, el editor genera el código XML del TUIML definido por el usuario, entendible por ToyVision. Además, y como se desprende de la evaluación de la usabilidad (apartado 4.2 de la memoria), la interacción con el editor se realiza de forma sencilla. En conclusión, la consecución de los objetivos iniciales se ha logrado de forma exitosa.

Personalmente, la realización de este proyecto ha supuesto una gran experiencia. Me permitido conocer una tecnología como es la interacción tangible, la cual desconocía. Por otro lado he podido poner en práctica los conocimientos adquiridos a lo largo de la carrera en este proyecto gestionando los recursos de los que disponía para llevar a cabo los objetivos (para más detalles, ver anexo C). También me ha permitido aprender sobre las tecnologías utilizadas en el proyecto, tales como la plataforma de hardware Arduino o el lenguaje de programación Processing.

6.2. Trabajo futuro

En este apartado se proponen posibles trabajos futuros relacionados con el editor elaborado en este Proyecto Fin de Carrera.

- **Soporte para Kinect.** En la versión más reciente de ToyVision se ha incorporado el controlador Kinect. Esto incluye nuevos tipos de elementos en los juegos que el editor debería incluir en futuras versiones. También sería interesante plantear cómo representar en el editor el espacio del juego, que pasaría de las dos dimensiones actuales que tiene la mesa a 3 dimensiones.

- **Integración en un Entorno de Desarrollo.** El editor gráfico desarrollado permite definir el esquema y los elementos del juego a crear, pero el bucle principal del juego debe programarse de forma separada. La opción de integrar este editor dentro de un entorno de desarrollo permitiendo así realizar en la misma aplicación la definición del TUIML del juego y la programación del bucle principal del mismo.

Anexo A. Documentación de Software

En este anexo se incluye todo lo relacionado con la documentación de Software, incluyendo la metodología utilizada y los diagramas realizados durante las fases de análisis y diseño.

A.1. Metodología utilizada

Durante el proceso de desarrollo del editor se ha seguido una metodología con un ciclo de vida incremental. En esta metodología en cada iteración se desarrolla una versión funcional del producto, a la que se le añaden nuevas funcionales en posteriores iteraciones. Para la especificación de los diagramas se ha utilizado el Lenguaje de Modelado Unificado (UML) [UMLWeb].

A.2. Diagrama de Casos de Uso

En el diagrama de casos de uso (figura 39) se observa que los principales casos de uso son los que tiene cualquier editor. Para los juegos los casos de uso obtenidos son:

- Crear un juego nuevo.
- Cargar un juego existente.
- guardar un juego generando el fichero XML asociado.

Para cada elemento del juego, los casos de uso son los siguientes:

- Añadir/editar/eliminar un **token** del TUIML.
- Añadir/editar/eliminar un **constraint** del TUIML.
- Añadir/editar/eliminar un **TAC** del TUIML.

Por último, también se han establecido casos de uso relacionados con la visualización:

- Visualizar el TUIML del juego.
- Probar un controlador asociado a una placa arduino.
- Visualizar el estado actual de la mesa.

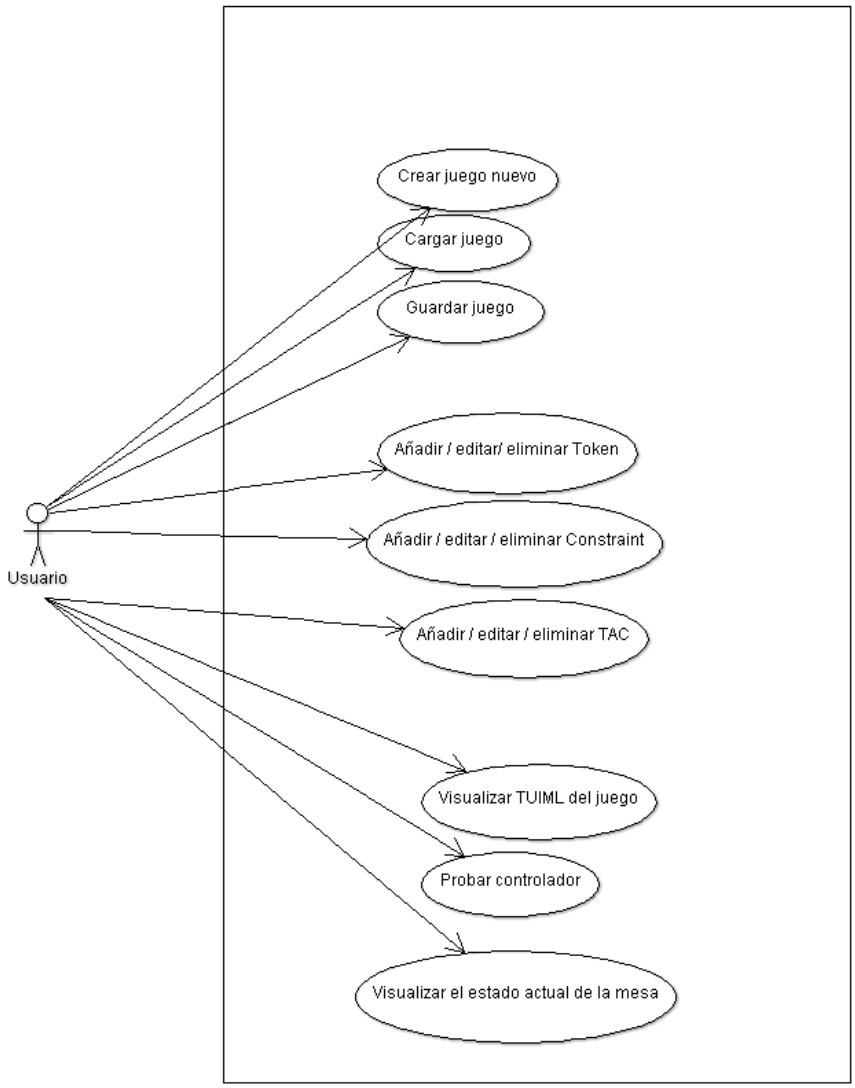


Figura 39. Diagrama de Casos de Uso.

A.3. Diagrama de secuencia

En el diagrama de secuencia (figura 40) se puede ver la secuencia de acciones para el caso de uso de crear, editar y eliminar un constraint. En el diagrama se puede observar la estructura jerárquica de las clases en el editor, ya que es la clase token la que crea y elimina una instancia de la clase constraint.

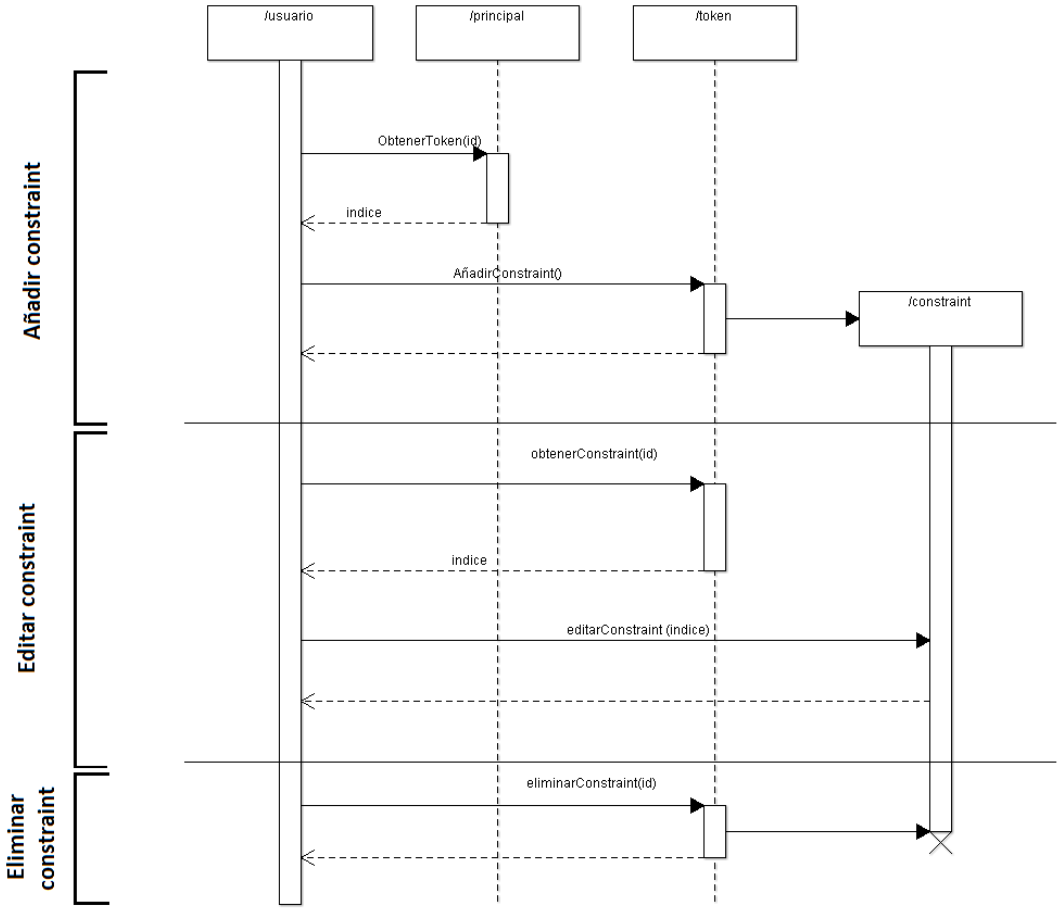


Figura 40. Diagrama de Secuencia.

Anexo B. Ficheros toys.xml.

En este anexo se muestran los ficheros toys.xml que definen el esquema TUIML del juego. Estos ficheros son utilizados por ToyVision durante la ejecución del juego.

B.1. Dragon's Cave

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
  <token height="600" name="tabletop" type="tabletop" width="800">
    <constraint list_vertex="0.0,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0,0.0" name="todo" orientable="true"
    type="active" visible="true">
      <tac subtoken="cofre" type="associative"/>
      <tac subtoken="Dragon" type="associative/manipulative X/manipulative Y/manipulative rot"/>
      <tac subtoken="caballero" type="associative/manipulative X/manipulative Y/manipulative rot"/>
      <tac subtoken="rey" type="associative/manipulative X/manipulative Y/manipulative rot"/>
      <tac subtoken="columna" type="associative/manipulative X/manipulative Y"/>
    </constraint>
  </token>
  <token id="14" name="Dragon" type="named">
    <constraint arduino="1" component="servo" name="motor" pin="10" type="active"/>
    <constraint list_vertex="0.08094916,0.14073102,-0.082828626,0.15145794,
        -0.087474585,-0.13979343,0.0749664,-0.14796197,0.08089802,0.13826357" name="cerca"
        orientable="false" type="active" visible="true">
      <tac subtoken="caballero" type="associative"/>
      <tac subtoken="rey" type="associative"/>
    </constraint>

  </token>
  <token id="1" name="cofre" type="named">
    <constraint arduino="1" component="servo" name="tapa" pin="11" type="active"/>
    <constraint list_vertex="-0.037255347,-0.06713351,-0.03739774,0.073607035,
        -0.091561794,0.07096467,-0.08725776,-0.06482427,-0.038644224,-0.06713795" name="cerca"
        orientable="true" type="active" visible="true">
      <tac subtoken="caballero" type="associative"/>
      <tac subtoken="rey" type="associative"/>
    </constraint>
  </token>
  <token id="13" name="caballero" type="named">
    <constraint list_vertex="0.035401616,-0.038642555,-0.031195397,-0.032690167,
        -0.03246119,-0.1092003,0.03135924,-0.11500751,0.036871556,-0.033778977" name="manos"
        orientable="true" type="active" visible="true">
      <tac subtoken="espada" type="associative"/>
    </constraint>
    <constraint list_vertex="0.03707573,0.06254827,-0.028255502,0.061023872,-0.02821466,0.14747885,
        0.035850763,0.15648006,0.038545668,0.06741179" name="llamada" orientable="true"
        type="active" visible="true">
      <tac subtoken="espada" type="associative"/>
    </constraint>
  </token>
  <token id="12" name="rey" type="named">
    <constraint list_vertex="0.035401616,-0.038642555,-0.031195397,-0.032690167,-0.03246119,-0.1092003,
    0.03135924,-0.11500751,0.036871556,-0.033778977" name="manos" orientable="true" type="active" visible="true">
      <tac subtoken="espada" type="associative"/>
    </constraint>
    <constraint list_vertex="0.03707573,0.06254827,-0.028255502,0.061023872,-0.02821466,0.14747885,0.035850763,
    0.15648006,0.038545668,0.06741179" name="llamada" orientable="true" type="active" visible="true">
      <tac subtoken="espada" type="associative"/>
    </constraint>
  </token>
  <token id="15" name="columna" type="named"/>
  <token name="espada" type="simple"/>
</xml>
```

B.2. Coche y casa con barrera

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
  <token height="600" name="tabletop" type="tabletop" width="800">
    <constraint list_vertex="0.0,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0,0.0"
    name="todo" orientable="true" type="active" visible="true">
      <tac subtoken="casa" type="associative"/>
      <tac subtoken="fader" type="associative"/>
      <tac subtoken="semaforo" type="associative/manipulative X/manipulative Y"/>
    </constraint>
  </token>
  <token id="0" name="casa" type="named">
    <constraint arduino="1" component="servo" name="barrera" pin="10" type="active"/>
    <constraint list_vertex="0.090738565,-0.17123337,-0.038296096,-0.15973893,
        -0.045594696,-0.35450956,0.09046088,-0.36142027,0.09077823,-0.1687652"
        name="entrada" orientable="true" type="passive" visible="true">
      <tac subtoken="coche" type="associative"/>
    </constraint>
  </token>
  <token id="15" name="coche" type="named"/>
  <token id="14" name="semaforo" type="named">
     <constraint list_vertex="-0.18270198,-0.080727845,0.192298,-0.08319709,0.1936869,
        0.05013621,-0.18547976,0.03532153,-0.18547976,-0.07578957,-0.18270198,-0.083196975"
        name="pasado" orientable="true" type="passive" visible="true">
        <tac subtoken="coche" type="associative"/>
     </constraint>
  </token>
  <token id="13" name="fader" type="named">
    <constraint list_vertex="0.031250004,-0.045384876,-0.031249998,-0.037977464,-0.029861104,
        -0.22563177,0.034027804,-0.23057002,0.03263888,-0.042915694" name="regulador"
        orientable="true" type="passive" visible="true">
      <tac subtoken="dedo" type="associative/manipulative Y"/>
    </constraint>
  </token>
  <token name="dedo" type="simple"/>
</xml>
```

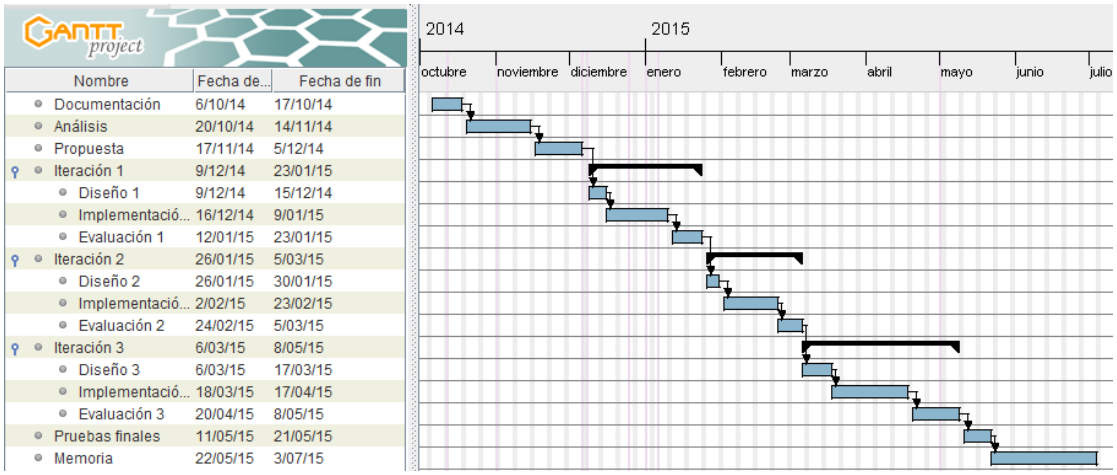
B.3. metaDESK

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
  <token height="600" name="tabletop" type="tabletop" width="800">
    <constraint list_vertex="0.0,0.0,1.0,0.0,1.0,1.0,0.0,1.0,0.0,0.0" name="todo"
        orientable="true" type="active" visible="true">
      <tac subtoken="pilar" type="associative/manipulative X/manipulative Y/manipulative rot"/>
      <tac subtoken="aljaferia" type="associative/manipulative X/manipulative Y/manipulative rot"/>
      <tac subtoken="lupa" type="associative/manipulative X/manipulative Y/manipulative rot"/>
      <tac subtoken="navegador" type="associative/manipulative X/manipulative Y/manipulative rot"/>
    </constraint>
  </token>
  <token id="1" name="pilar" type="named"/>
  <token id="0" name="aljaferia" type="named"/>
  <token id="15" name="lupa" type="named">
    <constraint list_vertex="0.104223825,0.21442154,0.019501608,0.21935979,0.02505719,0.086026445,
        0.1083905,0.088495634,0.104223825,0.21442154" name="satelite" orientable="true"
        type="passive" visible="true">
      <tac subtoken="dedo" type="associative"/>
    </constraint>
    <constraint list_vertex="-0.03327616,0.22182891,-0.11522063,0.22182886,-0.10966504,0.07861903,
        -0.024942825,0.0884956,-0.03327616,0.21935979" name="bus" orientable="true" type="passive"
        visible="true">
      <tac subtoken="dedo" type="associative"/>
    </constraint>
  </token>
  <token name="dedo" type="simple"/>
  <token id="13" name="navegador" type="named">
    <constraint list_vertex="0.048850734,-0.043844834,-0.04849281,-0.048635792,-0.036651503,-0.25675204,
        0.057997115,-0.24687979,0.052933928,-0.048998747" name="zoom" orientable="true" type="passive"
        visible="true">
      <tac subtoken="dedo" type="associative/manipulative Y"/>
    </constraint>
  </token>
</xml>
```

Anexo C. Gestión del Proyecto

En este anexo se muestra la gestión del tiempo a la hora de realizar el proyecto.

Tal y como muestra la figura 41, proyecto se empezó en octubre del 2014, y se ha desarrollado a lo largo de 8 meses, dedicando al principio 6 horas diarias. Inicialmente se planificó una duración menor del proyecto, que finalmente hubo que ampliar debido a la compaginación del mismo con unas prácticas en empresa durante 4 meses que redujeron a la mitad las horas diarias dedicadas al mismo.



Nombre	Fecha de...	Fecha de fin
Documentación	6/10/14	17/10/14
Análisis	20/10/14	14/11/14
Propuesta	17/11/14	5/12/14
Iteración 1	9/12/14	23/01/15
Diseño 1	9/12/14	15/12/14
Implementació...	16/12/14	9/01/15
Evaluación 1	12/01/15	23/01/15
Iteración 2	26/01/15	5/03/15
Diseño 2	26/01/15	30/01/15
Implementació...	2/02/15	23/02/15
Evaluación 2	24/02/15	5/03/15
Iteración 3	6/03/15	8/05/15
Diseño 3	6/03/15	17/03/15
Implementació...	18/03/15	17/04/15
Evaluación 3	20/04/15	8/05/15
Pruebas finales	11/05/15	21/05/15
Memoria	22/05/15	3/07/15

Figura 41. Diagrama de Gantt de la gestión del tiempo

En la figura 42 se muestra la distribución de tiempos de las fases del proyecto. La duración total del mismo ha sido de 760 horas, dedicando la mayor parte de éstas en la implementación del editor (31%) y la redacción de la memoria (24%).

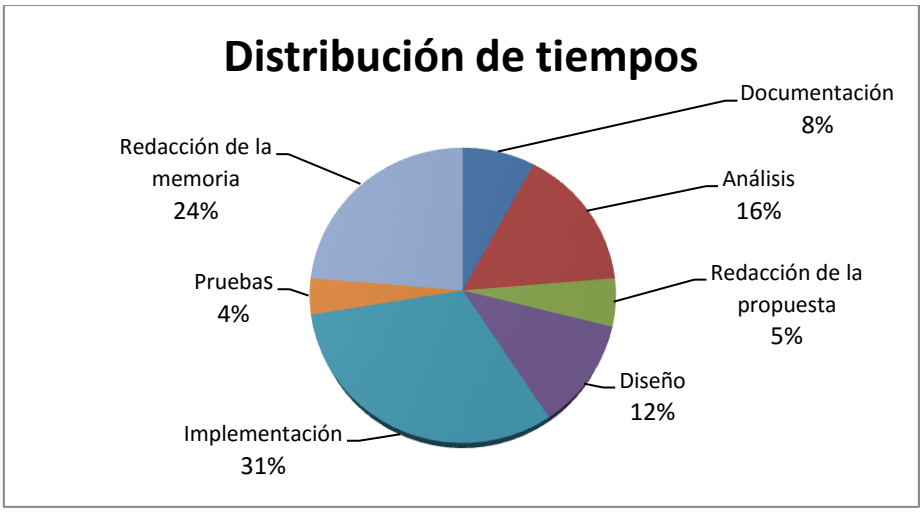


Figura 42. Distribución de tiempos de las fases de proyecto.

Índice de figuras

Bibliografía

- [ArduinoProcessingWeb] http://playground.arduino.cc/Interfacing/Processing (último acceso: Junio 2015).
- [ArduinoWeb] web de la plataforma Arduino: https://www.arduino.cc/ (último acceso: Junio 2015)
- [AsteroidsWiki] https://es.wikipedia.org/wiki/Asteroids (último acceso: Junio 2015)
- [B86] Brooke, J. (1996). SUS-A quick and dirty usability scale. *Usability evaluation in industry*, *189*(194), (pp. 4-7).
- [ControlP5Web] http://www.sojamo.de/libraries/controlP5/ (último acceso: Agosto 2015).
- [G4PWeb] http://www.lagers.org.uk/g4p/ (último acceso: Junio 2015).
- [GIGAWeb] Web del GIGA Affective Lab: http://giga.cps.unizar.es/affectivelab/ (último acceso: Junio 2015).
- [GuidoWeb] https://github.com/fjenett/Guido (último acceso: Agosto 2015).
- [KB07] Kaltenbrunner, M., & Bencina, R. (2007, February). reacTIVision: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction* (pp. 69-74). ACM.
- [KBBC05] Kaltenbrunner, M., Bovermann, T., Bencina, R., & Costanza, E. (2005, May). TUIO: A protocol for table-top tangible user interfaces. In *Proc. of the The 6th Int'l Workshop on Gesture in Human-Computer Interaction and Simulation* (pp. 1-5).
- [KLLL04] Klemmer, S. R., Li, J., Lin, J., & Landay, J. A. (2004, April). Papier-Mache: toolkit support for tangible input. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 399-406). ACM.
- [LSJ04]Leland, N., Shaer, N., & Jacob, R. (2004). TUIMS: Laying the Foundations for a tangible user interface Management System In. *Proc. of Pervasive'04*.
- [MCB08] Marco, J., Cerezo, E., & Baldassarri, S. (2008) NikVision: Desarrollo de Videojuegos Basados en Interfaces Naturales. In *18º Congreso Español de Informática Gráfica* (pp. 233-236).
- MCB14] Marco, J., Cerezo, E., & Baldassarri, S. (2014) Lenguaje de Modelado de Juegos de Tablero Híbridos. In*Proceedings of the XV Congreso Internacional Interacción Persona-Ordenador (Tenerife, Spain) Interacción 2014* (pp. 224-231).
- [PFCU13] (Proyecto Fin de Carrera) UBIDE, E. Gestión de juguetes tangibles activos para una mesa interactiva. Universidad de Zaragoza, 2013.

- [ProcessingWeb] https://www.processing.org/ (último acceso: Junio 2015).
- [SLCJ04] Shaer, O., Leland, N., Calvillo-Gamez, E. H., & Jacob, R. J. (2004). The TAC paradigm: specifying tangible user interfaces. *Personal and Ubiquitous Computing*, *8*(5), (pp. 359-369).
- [TMGWeb] web del Tangible Media Group: http://tangible.media.mit.edu/ (último acceso: Junio 2015)
- [TUIOWeb] Web del protocolo TUIO: http://www.tuio.org/ (último acceso: Junio 2015).
- [UI97] Ullmer, B., & Ishii, H. (1997, October). The metaDESK: models and prototypes for tangible user interfaces. In *Proceedings of the 10th annual ACM symposium on User interface software and technology* (pp. 223-232). ACM.
- [UMLWeb] www.uml.org/ (último acceso: Agosto 2015).