Juego colaborativo para el aprendizaje de las matemáticas

Collaborative game for learning mathematics

José Isaac Hernández Quintero
ISAATC
Escuela Técnica Superior de Ingeniería Informática
Trabajo de Fin de Grado

La Laguna, 09 de julio de 2014

D. ª Carina Soledad González González, con N.I.F. 54.064.251-Z profesora Titular de Universidad adscrita al Departamento ISAATC del Departamento de la Universidad de La Laguna

C E R T I F I C A

Que la presente memoria titulada:

"Juego colaborativo para el aprendizaje de las matemáticas."

ha sido realizada bajo su dirección por D. José Isaac Hernández Quintero, con N.I.F. 78.718.908-M.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 9 de julio de 2014

Agradecimientos

Quiero agradecer a toda mi familia, en especial a mis padres, por apoyarme y animarme durante el transcurso de toda la carrera.

También agradecer a los compañeros que me han acompañado en este largo camino, ya amigos, porque siempre me han ayudado a encontrar las fuerzas para seguir adelante aunque ya casi no me quedaran.

Agradecer a todos los profesores que me han instruido para ayudarme a convertirme en un ingeniero, por sus enseñanzas y sus consejos. Y sobre todo agradecer a mi directora de proyecto por saber guiarme a lo largo de estos meses, dándome confianza para tomar decisiones propias y estando ahí siempre que la necesitaba.

Y por último, agradecer a la directiva por el gran trabajo realizado en el proceso de cambio del antiguo plan al actual.

Resumen

El objetivo de este trabajo ha sido el desarrollo de varios juegos colaborativos dentro del proyecto EMATIC, el cual ya contiene una serie de actividades individuales para que los jugadores desarrollen sus habilidades y capacidades matemáticas. Se pretende por tanto estudiar los patrones que deben seguir los juegos colaborativos, y desarrollar los mismos, de modo que demos una nueva dimensión al proyecto permitiendo que los jugadores desarrollen sus habilidades colaborando entre sí.

Como resultado del trabajo realizado se han desarrollado dos juegos colaborativos, uno de resolución de Sudokus, y otro de memorización. Además, se han realizado mejoras en las interfaces gráficas de la aplicación multidispositivo de EMATIC.

Palabras Clave

EMATIC, Aprendizaje colaborativo, Mecánicas de juego, Patrones de juegos colaborativos, Aplicación multidispositivo.

Abstract

The objective of this work has been the development of several collaborative games within the Ematic project, which already contains a number of individual activities for players to develop their math skills. It is therefore intended to study the patterns to be followed by collaborative games, and develop the same, so we give a new dimension to the project allowing players to develop their skills by working together.

As a result of work performed has developed two collaborative games, one solving Sudokus, and other of memorization. In addition, there have been improvements in the graphical interfaces of multi-device aplicaction Ematic.

Keywords

Ematic, Collaborative Learning, Game Mechanics, Patterns of collaborative games, multi-device Aplicaction.

Índice

Índice de figuras

Índice de tablas

Capítulo 1. EMATIC

Introducción

En este capítulo se explicarán los objetivos que se deberán alcanzar al finalizar el trabajo de fin de grado. Por ello, primero veremos en qué consiste el proyecto EMATIC, y a continuación detallaremos los objetivos que se deben alcanzar.

Proyecto EMATIC

El proyecto EMATIC (Enseñanza de Matemáticas por medio de las TIC) permite a los niños y niñas aprender matemáticas de una forma más entretenida, a base de actividades por ordenadores y *tablets*, las cuales desarrollan a su vez la habilidad para pensar de manera lógica y aprender diferentes aspectos de las matemáticas básicas. También permite a los estudiantes realizar un mismo ejercicio sin que este llegue a ser repetitivo, convirtiendo el proceso de aprendizaje en algo tan dinámico y entretenido como si se tratara de un juego, en lugar de una asignatura tediosa para ellos.

EMATIC está formado por un equipo multidisciplinar, en el cual profesionales de tecnologías de la información colaboran con educadores y maestros tomando su experiencia educativa en la evaluación del conocimiento. Además estos expertos prueban la herramienta como usuarios, obteniendo una valiosa retroalimentación para la parte técnica con el fin de mejorar la usabilidad de la plataforma.

El uso de programas informáticos para las distintas dificultades en los niños es un recurso imprescindible para conseguir mejores resultados en el proceso de enseñanza-aprendizaje. Normalmente, el alumnado de necesidades educativas especiales (en adelante NEE) y necesidades educativas específicas de apoyo educativo (en adelante NEAE), presentan problemas en el aprendizaje y el uso de las nuevas tecnologías puede resultar de ayuda.

En este sentido, en EMATIC el aprendizaje se realiza de forma amena y con un recurso atractivo. Los colores, el movimiento y el sonido estimulan la atención y se consiguen mejores resultados para el desarrollo del alumnado.

Equipo de trabajo

Este trabajo de fin de grado se realiza en paralelo a otros trabajos finales de grado, que también están relacionados con el proyecto EMATIC. Se forma un grupo de trabajo formado por una estudiante del Grado en Diseño, y tres estudiantes de Grado en Ingeniería Informática. De esta forma, la diseñadora gráfica se encarga de la realización de los diseños gráficos necesarios para las interfaces de la aplicación multidispositivo EMATIC, mientras que uno de los informáticos se encarga del desarrollo del agente pedagógico animado, otro de los informáticos se encarga de realizar las animaciones necesarias para la aplicación y finalmente, en este trabajo final de grado, se realizan los juegos colaborativos y la integración final de los diferentes módulos de EMATIC.

Objetivos

El objetivo que se pretende lograr es desarrollar juegos colaborativos en la aplicación multidispositivo EMATIC para que los usuarios, algunos con NEE, puedan mejorar sus habilidades matemáticas. Para ello se deberá estudiar en primer lugar los patrones que deben seguir los mismos para que sean colaborativos, a continuación estudiar las mecánicas de juego que se incluirán en cada juego, y por último proceder a su desarrollo.

Además, para el correcto funcionamiento de los juegos y su integración en la aplicación EMATIC, otro objetivo prioritario de este proyecto será la actualización de las interfaces gráficas y la arquitectura de la información de la aplicación multidispositivo EMATIC.

Plan de trabajo

Para cumplir los objetivos se programa el siguiente plan de trabajo:

- Semana 1 - 2: Análisis de requisitos de los juegos.
- Semana 3 - 4: Estudio de los patrones de juego colaborativo y aplicación de los mismos.
- Semana 4 - 6: Estudio de las mecánicas de juego y aplicación de las mismas.
- Semana 7 - 10: Desarrollo de primer juego.
- Semana 11 - 14: Desarrollo del segundo juego.
- Semana 15 - 16: Actualización de Interfaces e Integración.

Capítulo 2. Tecnologías utilizadas

Introducción

En este capítulo se presentará la tecnología con la que funciona el sistema EMATIC y las tecnologías que se utilizarán para la actualización de la aplicación multidispositivo.

Tecnologías de la aplicación

La aplicación EMATIC está desarrollada en Django 1.3.1 liberado el 9 de septiembre de 2011 (Django Project, 2011), y hace uso de los *framework*s *JQueryMobile* 1.1.1 liberado el 12 de julio de 2012 (JQueryMobile, 2012) y *KineticJS* 3.9.6 (KineticJS, s.f.). El trabajo de actualización consiste en sustituir *JQueryMobile* 1.1.1, por el *framework Bootstrap* 3.1.1 liberado el 13 de febrero de 2014 (Bootstrap, 2014). A continuación vamos a realizar una breve explicación sobre Django para entender cómo funciona la aplicación.

Django

Django es un *framework* MTV, es decir, Modelo *Template* Vista. Para entender este modelo vamos a hacer una analogía con el conocido MVC (Modelo Vista Controlador) (Pavón, 2009):

- El Modelo en Django sigue siendo Modelo
- La Vista en Django se llama Plantilla (*Template*)
- El Controlador en Django se llama Vista

Una imagen nos hará entender mejor esta relación:

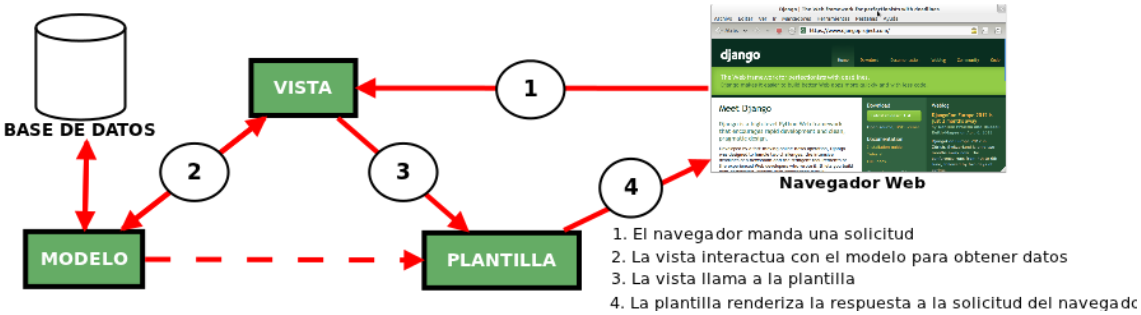


Figura 1 - Modelo MTV Django

Veamos que hace cada uno de ellos con un poco más de detalle y algunos conceptos adicionales.

El modelo

El modelo define los datos almacenados, se encuentra en forma de clases de *Python*, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

La vista

La vista se presenta en forma de funciones en *Python*, su propósito es determinar qué datos serán visualizados, entre otras cosas. El ORM, Mapeo Objeto-Relacional, (MarianoGuerra, 2012) de Django permite escribir código *Python* en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. Lo más importante a entender con respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla.

La plantilla

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django.

La configuración de las rutas

Django posee un mapeo de *URLs* que permite controlar el despliegue de las vistas, esta configuración es conocida como *URLConf*. El trabajo del *URLConf* es leer la *URL* que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo. El *URLConf* está construido con expresiones regulares en *Python* y sigue la filosofía de *Python*: explícito es mejor que implícito. Este *URLConf* permite que las rutas que maneje Django sean agradables y entendibles para el usuario.

Si consideramos al *URLConf* en el esquema anterior tendríamos este resultado más completo. (Montero, s.f.)

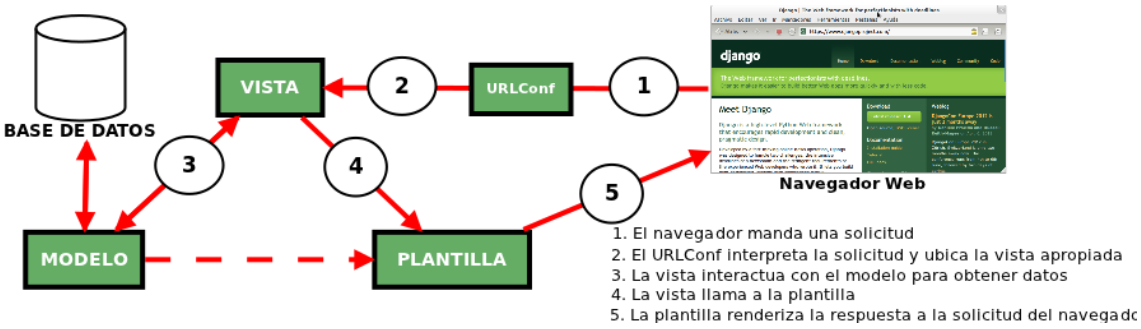
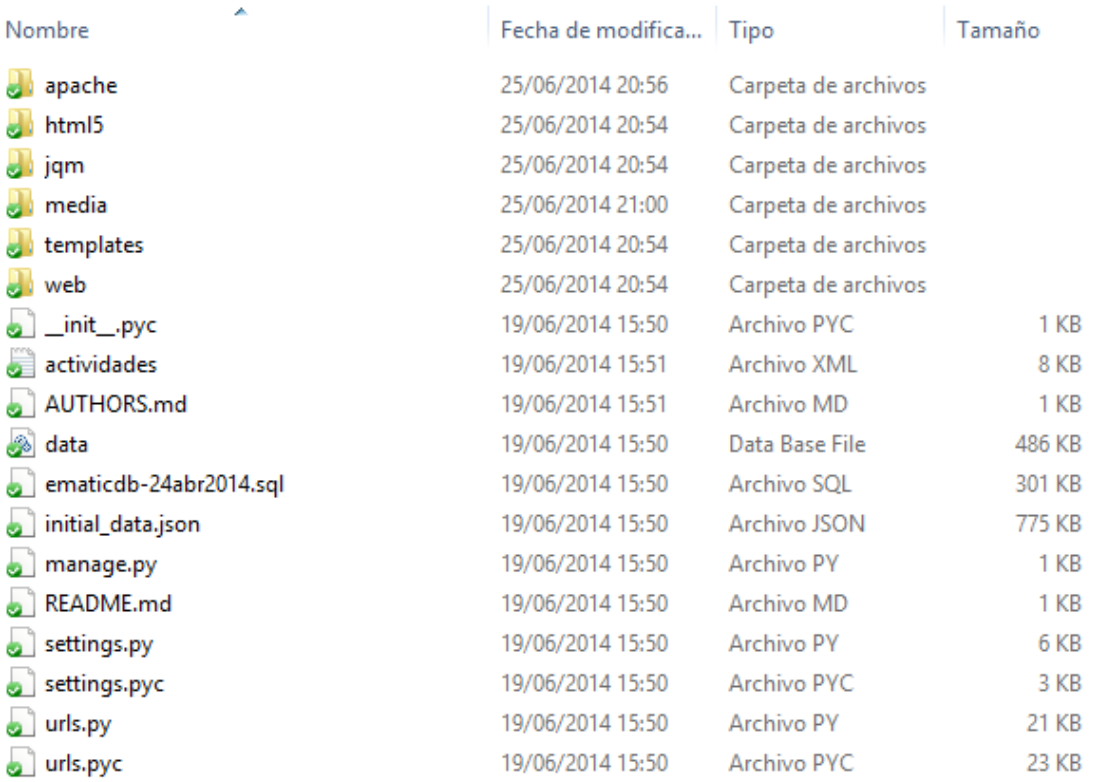


Figura 2 - Modelo MTV con *URLConf* de Django

Estructura de la aplicación EMATIC

El código fuente de la aplicación EMATIC está estructurado del siguiente modo:



Nombre	Fecha de modifica...	Tipo	Tamaño
apache	25/06/2014 20:56	Carpeta de archivos	
html5	25/06/2014 20:54	Carpeta de archivos	
jqm	25/06/2014 20:54	Carpeta de archivos	
media	25/06/2014 21:00	Carpeta de archivos	
templates	25/06/2014 20:54	Carpeta de archivos	
web	25/06/2014 20:54	Carpeta de archivos	
__init__.pyc	19/06/2014 15:50	Archivo PYC	1 KB
actividades	19/06/2014 15:51	Archivo XML	8 KB
AUTHORS.md	19/06/2014 15:51	Archivo MD	1 KB
data	19/06/2014 15:50	Data Base File	486 KB
ematicdb-24abr2014.sql	19/06/2014 15:50	Archivo SQL	301 KB
initial_data.json	19/06/2014 15:50	Archivo JSON	775 KB
manage.py	19/06/2014 15:50	Archivo PY	1 KB
README.md	19/06/2014 15:50	Archivo MD	1 KB
settings.py	19/06/2014 15:50	Archivo PY	6 KB
settings.pyc	19/06/2014 15:50	Archivo PYC	3 KB
urls.py	19/06/2014 15:50	Archivo PY	21 KB
urls.pyc	19/06/2014 15:50	Archivo PYC	23 KB

Figura 3 - Estructura código fuente de la aplicación EMATIC

El modelo de datos se encuentra ubicado en la carpeta web con el nombre de *"models.py"*. Aquí podemos encontrar los tipos de datos que son necesarios para el funcionamiento de la aplicación.

Las plantillas o *template*s se encuentran en la carpeta *template*s. Existe una plantilla para cada pantalla que se muestra en la aplicación.

La vista se encuentra ubicada en la carpeta web con el nombre de *"views.py"*. Allí encontramos todas las funciones que se comunican con el modelo de datos, manipulan los mismos y renderizan las plantillas.

En la carpeta media encontramos el código fuente de los *framework*s, así como imágenes y logos que son necesarios para el funcionamiento de la aplicación.

En el archivo *"urls.py"* se encuentran todas las rutas de la aplicación.

Bootstrap

A continuación se realiza una pequeña introducción a *Bootstrap*, el *framework* que utilizamos para desarrollar la nueva interfaz gráfica de la aplicación. Se ha elegido *Bootstrap* porque es un *framework* usado en sitios web muy importantes como Twitter o la NASA, su interfaz es conocida por la mayoría de los usuarios de las nuevas tecnologías, nos permite realizar un diseño adaptable, y está bien integrado con JQuery.

Twitter *Bootstrap* es un *framework* o conjunto de herramientas de software libre para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de *Javascript* opcionales adicionales.

Es el proyecto más popular en GitHub y es usado por la NASA y la MSNBC junto a demás organizaciones. (Hosted Appliance, 2014)

Podemos decir que los principios en los que se basa la versión 3 son:

- Responsive Design: Consiste en que la página trata de "hacer lo correcto" al ser visualizada independientemente del dispositivo y tamaño de la pantalla. El término fue acuñado por Ethan Marcotte en 2010.
- Mobile first: Al contrario que en la versión 2, en la 3, el diseño responsivo es la opción por defecto al trabajar con *Bootstrap*.
- Cross Browser: Trata de ser compatible con la mayoría de navegadores.
- Integración con jQuery: Está muy integrado con jquery para el que define nuevos plugins
- Buenas prácticas: Trata de emplear algunas de las prácticas más extendidas en cuanto a usabilidad, uso de css3/html5, organización del código… (Conocimiento Abierto, 2014)

Capítulo 3. Proyecto EMATIC

Introducción

En este capítulo explicaremos el funcionamiento de la aplicación multidispositivo EMATIC, a continuación mostraremos las interfaces gráficas de la aplicación, el diseño que se propone, y finalmente mostraremos los resultados obtenidos.

Estado de las interfaces gráficas

Como hemos mencionado anteriormente, la interfaz gráfica de la aplicación era generada apoyándose en el *framework JQueryMobile* 1.1.1, y presentaba el aspecto que se muestra a continuación.

La pantalla para hacer *login* era la siguiente (Figura 4):

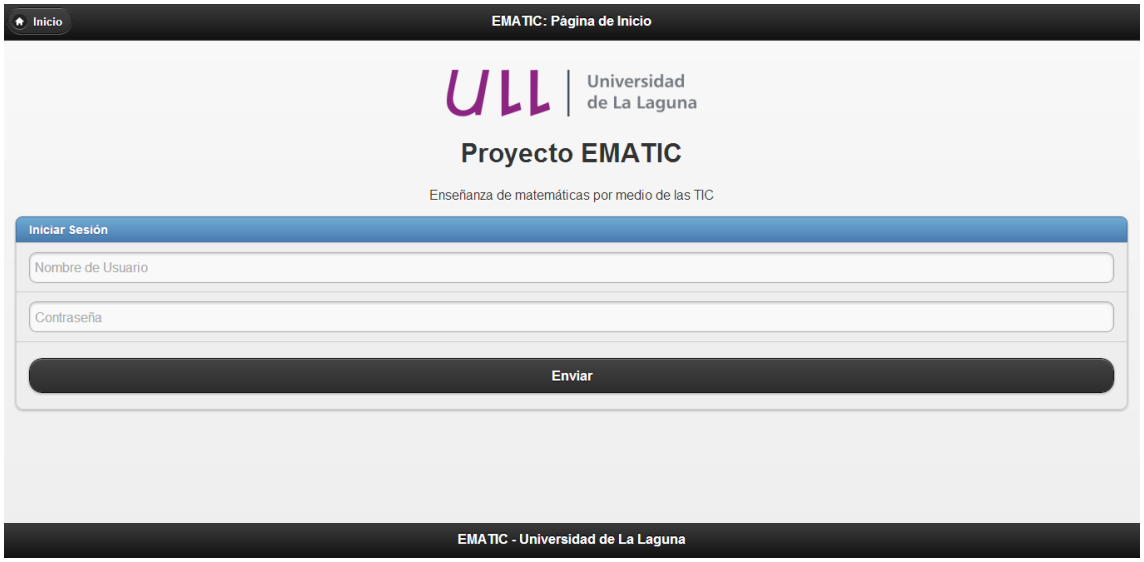


Figura 4 – Pantalla de *Login* (diseño antiguo)

Una vez realizábamos el *login* accedíamos al home, que presentaba el siguiente aspecto (Figura 5):

Figura 5 – Pantalla de Home (diseño antiguo)

Al pulsar "Iniciar Actividades" accedíamos a la pantalla que se muestra en la Figura 6:

Figura 6 – Pantalla de selección de actividades (diseño antiguo)

En esta pantalla podíamos seleccionar las actividades clasificadas en 10 categorías, las cuales eran:

- Clasificación
- Relaciones de orden
- Correspondencia
- Cuantificadores
- Contar
- Reconocimiento de números
- Cardinalidad
- Orden
- Ordinalidad
- Problemas

Una vez seleccionábamos una categoría accedíamos a una lista de ejercicios como esta (Figura 7):

Figura 7 – Pantalla de Lista de ejercicios (diseño antiguo)

Una vez seleccionábamos comenzar una actividad nos aparecía una nueva pantalla con el siguiente *popup* (Figura 8):

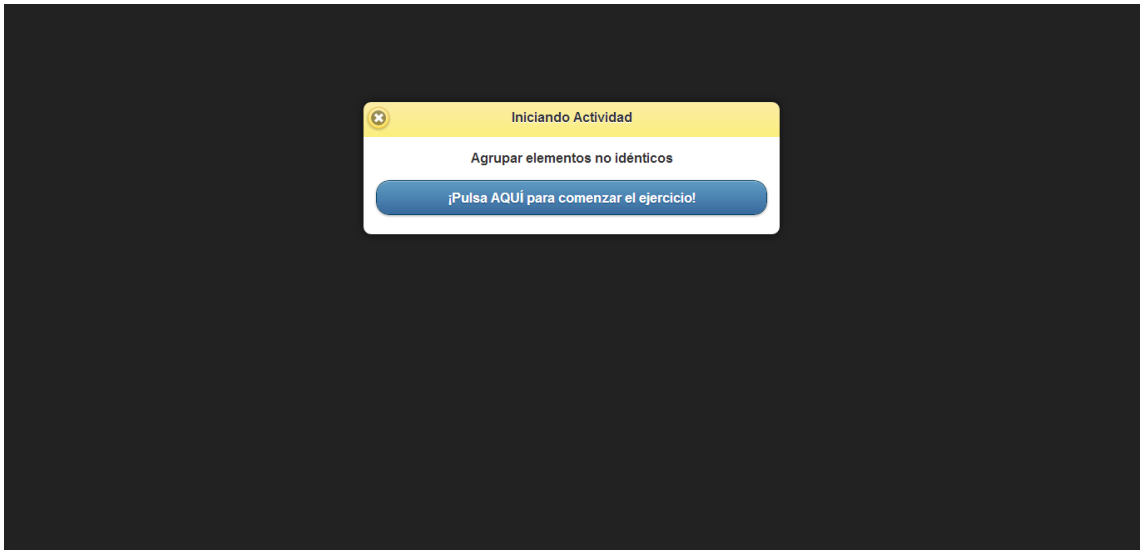


Figura 8 - Pantalla de actividad (diseño antiguo)

Tras pulsar el botón para comenzar el ejercicio, un agente nos hablaba explicándonos que teníamos que hacer en el ejercicio. Al finalizar su discurso los elementos que aparecían en la pantalla se volvían manipulables. La pantalla tenía el siguiente aspecto (Figura 9):

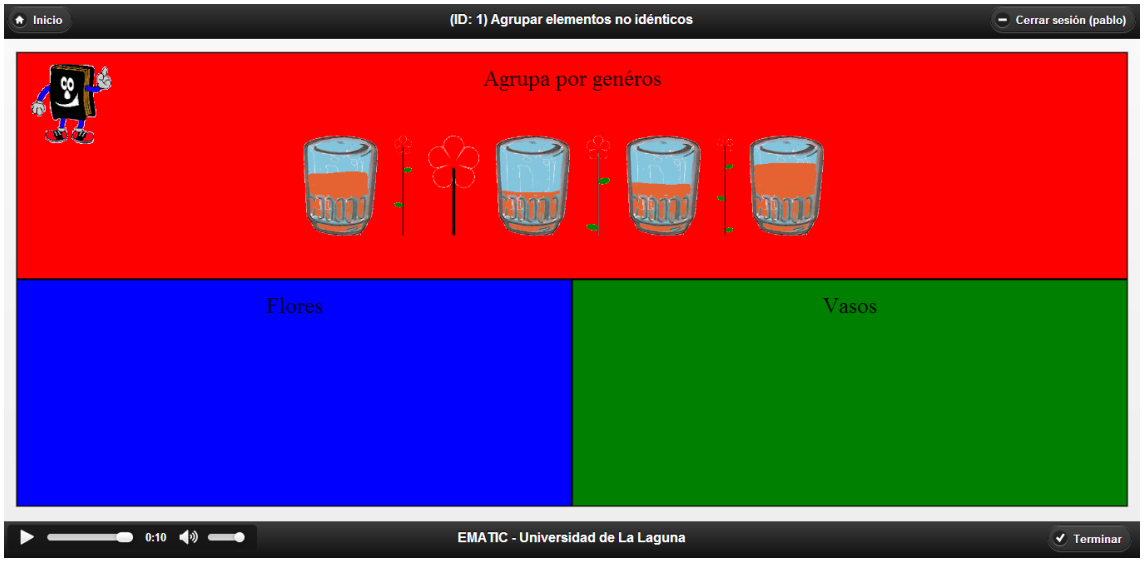


Figura 9 - Pantalla de actividad (diseño antiguo)

A continuación el usuario resolvía el ejercicio, y tras pulsar el botón "Terminar" le aparecía la siguiente pantalla (Figura 10):

Figura 10 - Pantalla de actividad (diseño antiguo)

El usuario debía pulsar "Ver resultado", y le aparecía la siguiente pantalla, en la que se señalan que objetos estaban colocados correctamente (Figura 11):

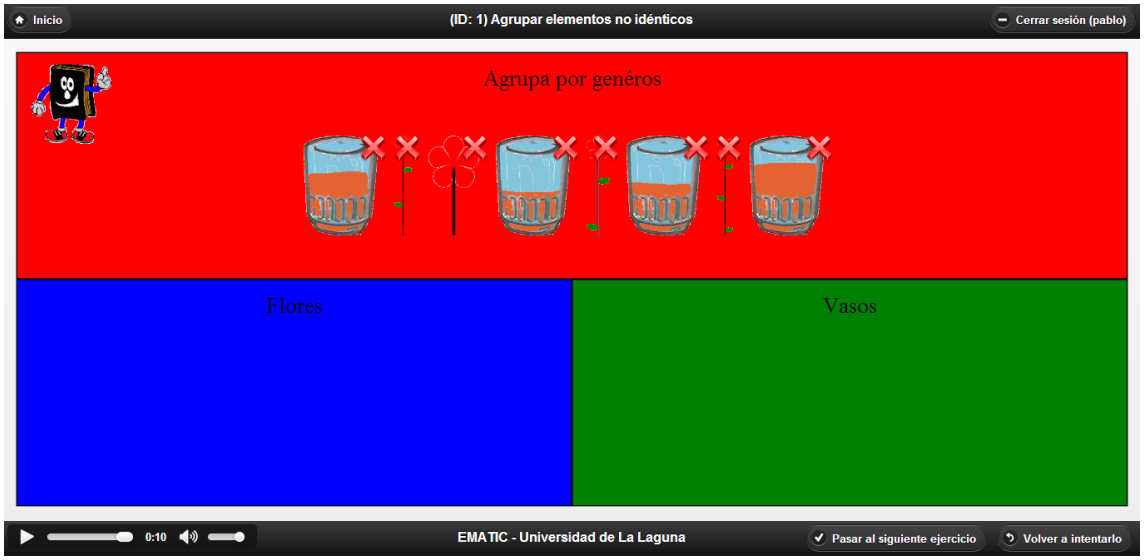


Figura 11 – Pantalla de actividad (diseño antiguo)

Actualización de las interfaces gráficas

La actualización de las interfaces gráficas se realiza según una nueva propuesta de mejora sobre la arquitectura de la información y la estética y estilos de la aplicación EMATIC.

De esta forma, se eliminan las 10 categorías de ejercicios que existían, y se simplifican en 4:

- Reconocimiento de números: En esta nueva categoría se incluyen las antiguas categorías de reconocimiento de números, contar y cardinalidad.
- Relaciones de orden: En esta nueva categoría se incluyen las antiguas categorías de relaciones de orden, orden y ordinalidad.
- Clasificación: En esta nueva categoría se incluyen las antiguas categorías de clasificación, correspondencia y cuantificadores.
- Problemas: Esta nueva categoría se corresponde con la antigua de problemas.

De esta forma, al pulsar en cada categoría vamos a escenarios distintos. En cada escenario existen una serie de objetos gráficos animados, con los que el usuario puede interactuar para acceder a distintas actividades de EMATIC.

Para conseguir adaptar los diseños propuestos a la aplicación multidispositivo se modifican los *template*s existentes. En primer lugar se procede a modificar el archivo *"base.html"*, ubicado en la carpeta *"templates"*, ya que este es el archivo base desde el que se extienden el resto de *template*s de la aplicación. En este archivo colocamos las etiquetas necesarias para cargar las hojas de estilo y archivos *javascript* de *Bootstrap* y JQuery y creamos la estructura de la aplicación.

Para crear la estructura de la aplicación, en primer lugar creamos la barra superior. Indicamos que los botones de ajustes, perfil y cerrar sesión solo deben mostrarse si el usuario se ha identificado, pues esta barra se muestra en la pantalla de iniciar sesión al igual que en el resto de pantallas de la aplicación. Además utilizamos la etiqueta de django "{% block *identificador* %}" y "{% endblock %}" para permitir añadir nuevo contenido en ese bloque desde el resto de *template*s

que sean extensiones suyas. En este caso, llamamos al bloque "*header*". El código de la barra superior queda del siguiente modo (Figura 12):

```
<!-- Fixed navbar -->
<div class="navbar navbar-default navbar-fixed-top" role="navigation">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" c
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="{% url home %}">
                  <img id="icono-home" src="/media/iconos/EmaticIconoHor
            </a>
    </div>
    <div class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                    {% if user.is_authenticated %}
                          <img id="icono-ajustes" src="/media/iconos/Ema
                    {% endif %}
            </ul>
      <ul class="nav navbar-nav navbar-right">
                  {% if user.is_authenticated %}
                          <li> <a href="{% url perfil_jugador %}"> <img
                          <li id="li-cerrar-sesion">
                                  <a href="{% url logout %}">
                                          <span id="span-cerrar-sesion"
```

Figura 12 - Barra de navegación superior

A continuación creamos el contenedor principal donde gracias a las etiquetas "{% block *identificador* %}" y "{% endblock %}" antes mencionadas, podremos incluir el código correspondiente al agente que nos ayuda en las actividades en el bloque "agente", y el código necesario para que se cargen las distintas pantallas en el bloque "*content*".

Para acabar con el archivo *"base.html"* declaramos al final un bloque para añadir el *footer* dependiendo de la pantalla en la que nos encontremos, que llamaremos "*footer*", y un bloque para añadir código *javascript*, que llamamos "*javascript*".

El siguiente objetivo es crear el inicio de sesión y el home. No existe un diseño propuesto para el inicio de sesión, por lo que la intención es que siga la línea marcada por el diseño del home. Ambas pantallas se cargan desde el mismo

template, que se llama *"home.html"* y se encuentra en la carpeta *"templates/web"*, por lo que procedemos a la modificación del mismo.

En primer lugar creamos el inicio de sesión, y para ello declaramos que este *template* es una extensión del *"base.html"* utilizando el código "{% extends 'base.html' %}". A continuación utilizamos las sentencias "block" antes mencionadas, y desarrollamos código en ellas. De este modo al renderizar la página, se mostrará el código desarrollado en *"base.html"*, con los bloques que hemos declarado en este archivo sustituidos por el código que desarrollemos en *"home.html"*.

En el bloque "*content*" desarrollamos el contenido de la página. En este bloque utilizamos la sentencia "{% if user.is_authenticated %}" para saber si el usuario ha iniciado sesión. En el caso de que no haya iniciado sesión cargamos la página de inicio de sesión creando el siguiente formulario (Figura 13):

```
<form action="{% url home %}" method="post" class="form-horizontal" role="form" id="form-login">
  {% csrf_token %}
  <div class="form-group">
        <label for="username" class="col-md-offset-2 col-md-2 control-label">{% trans 'Nombre de Usuario'
        <div class="col-md-5">
          <input type="text" class="form-control" name="username" id="username" value="" placeholder="{% t
        </div>
  </div>
  <div class="form-group">
        <label for="password" class="col-md-offset-2 col-md-2 control-label">{% trans 'Contraseña' %}</lab
        <div class="col-md-5">
          <input type="password" class="form-control" name="password" id="password" value="" placeholder="
        </div>
  </div>
  <div class="form-group">
        <div class="col-md-12 text-center">
          <button type="submit" class="btn btn-default">{% trans 'Iniciar sesión' %}</button>
        </div>
  </div>
</form>
```

Figura 13 - Formulario inicio de sesión

Añadimos el bloque del "*footer*", y de este modo, la pantalla de inicio de sesión queda así (Figura 14):

Figura 14 – Pantalla de Inicio (diseño actual)

A continuación, desarrollamos en el mismo archivo el home de la aplicación, que se renderizará cuando el usuario haya iniciado sesión. En este caso, simplemente creamos varias filas e insertamos las imágenes necesarias en el bloque "*content*", quedando el home con el siguiente aspecto (Figura 15):

Figura 15 – Pantalla de Home (diseño actual)

El siguiente paso es enlazar cada una de las 4 imágenes que simbolizan las distintas categorías con su escenario correspondiente. Para ello tenemos que crear un enlace en cada una de las imágenes, y crear 4 nuevos *templates*, *"feria.html"*, *"oficina.html"*, *"habitación.html"*, y *"supermercado.html"*. Además de crear estos 4 ficheros, hay que añadir las rutas de estos ficheros al archivo *"urls.py"* como se detalla a continuación (Figura 16):

```
url(r'^supermercado/$', 'web.views.supermercado', name='supermercado'),
url(r'^feria/$', 'web.views.feria', name='feria'),
url(r'^habitacion/$', 'web.views.habitacion', name='habitacion'),
url(r'^oficina/$', 'web.views.oficina', name='oficina'),
```

Figura 16 - Urls escenarios

Como observamos en primer lugar indicamos el patrón que corresponde con la *URL* que colocamos en el navegador. La primera "r" indica que a continuación vamos a escribir una cadena de caracteres permitiendo que no tengamos que colocar secuencias de escape para caracteres propios de expresiones regulares. El carácter "^" indica el comienzo de nuestra expresión, y el "$" indica el final. Lo que se encuentra entre ellos es la cadena a buscar. A continuación indicamos donde debemos dirigirnos si encontramos la cadena, y por último asignamos un alias a la *URL*.

El siguiente paso es crear 4 nuevas funciones en el archivo *"views.py"*, una para cada escenario. Desde cada función que hemos creado renderizaremos el *template* que se mostrará en el navegador. A continuación creamos los 4 *templates*. Todos siguen la misma estructura, utilizamos el mismo "*footer*" que en el home, y añadimos en el contenido la imagen del escenario correspondiente en el bloque "*content*". La correspondencia entre escenarios y categorías de problemas se ha detallado en el apartado anterior, por lo que vamos a ver como ha quedado finalmente cada escenario.

Primero veremos el escenario de la feria (Figura 17):

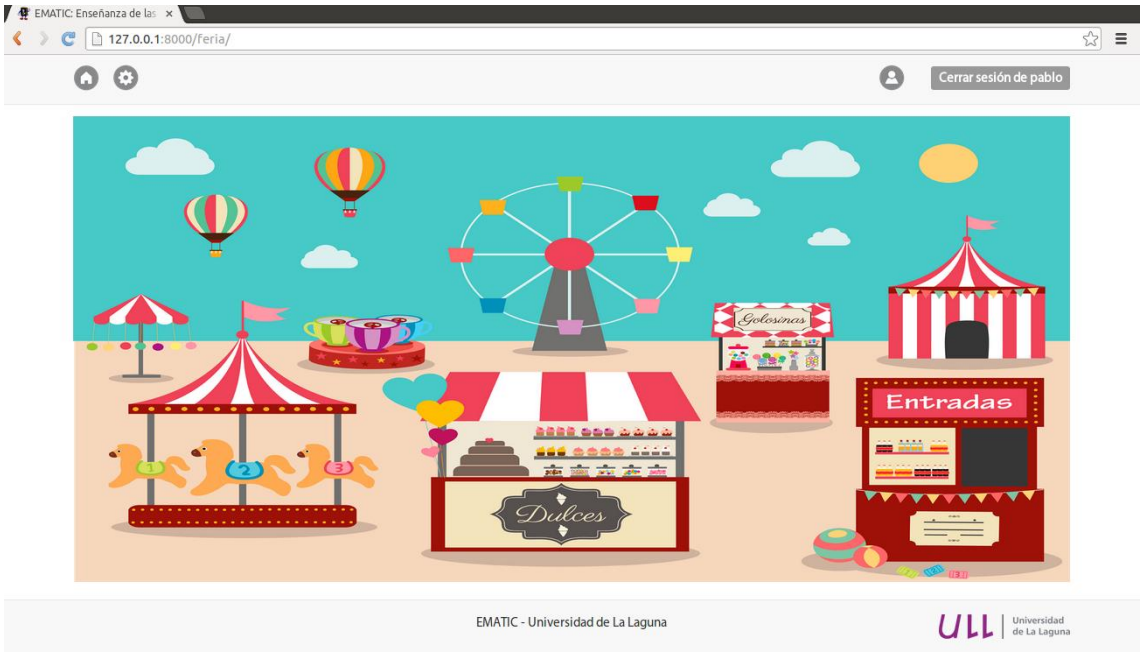


Figura 17 - Escenario de feria (diseño actual)

El escenario de la habitación queda del siguiente modo (Figura 18):

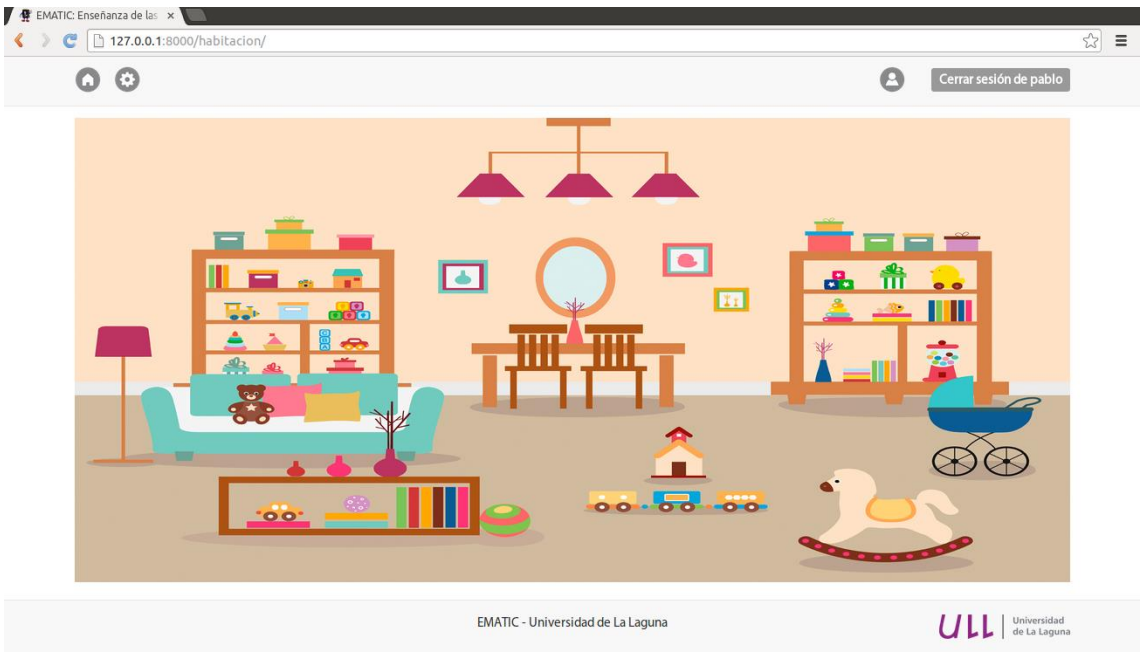


Figura 18 - Escenario habitación (diseño actual)

El escenario del supermercado queda del siguiente modo (Figura 19):

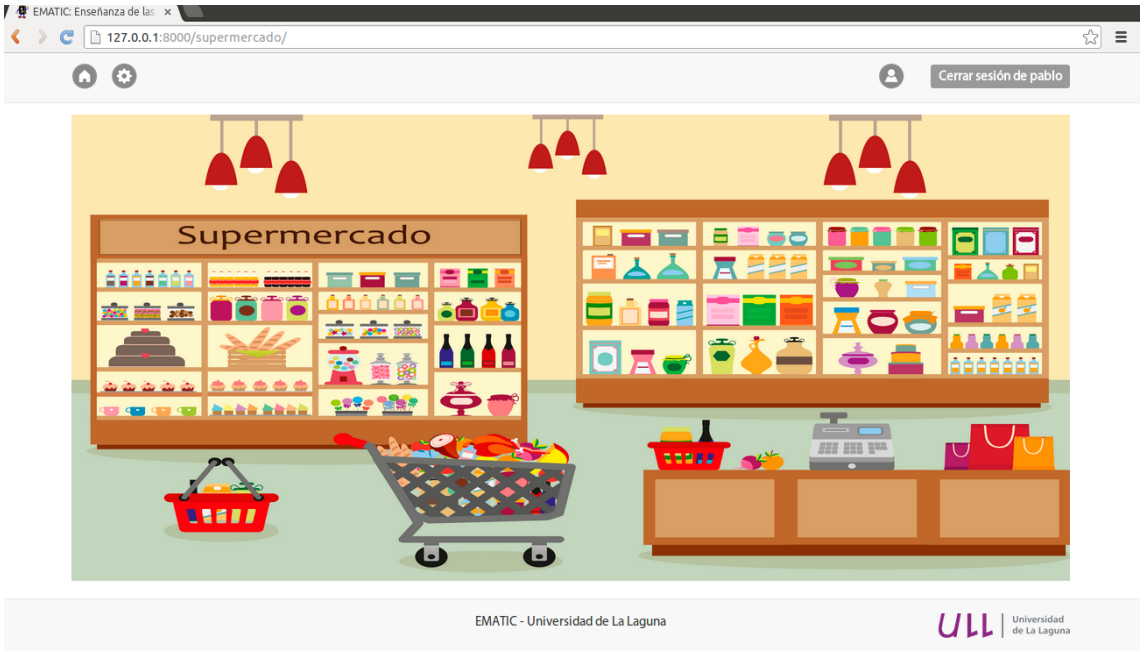


Figura 19 - Escenario supermercado (diseño actual)

El escenario de la oficina queda del siguiente modo (Figura 20):

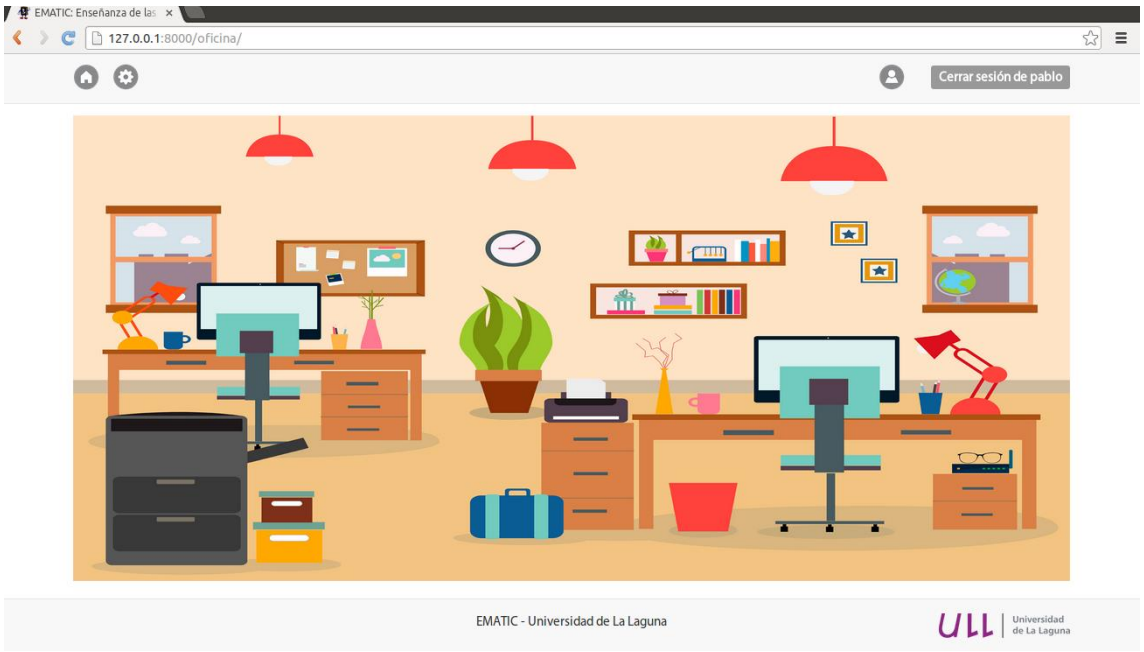


Figura 20 - Escenario oficina (diseño actual)

Finalmente desarrollamos el diseño de las actividades. No existe un diseño propuesto para las actividades, por lo que la intención es que siga la línea marcada por el diseño del resto de la aplicación. Para cada actividad tenemos que modificar dos *templates*, uno es el *template* que se carga al inicio y muestra la actividad, el otro es un *template* que se carga mediante uso de Ajax una vez el usuario da por terminada la actividad, y muestra los resultados obtenidos.

Primero modificamos el *template* inicial. En primer lugar modificamos los *popup* que se mostraban, pues funcionan de modo distinto en *JQueryMobile*, que es el *framework* con el que estaban desarrollados que con *Bootstrap*. Creamos un " " con ID "modal-final-place" que será donde se cargará mediante Ajax (w3schools, s.f.) el *popup* donde se muestran los resultados del ejercicio. Gracias a la tecnología Ajax, podemos realizar una llamada asíncrona al servidor, y de este modo recargar solamente la parte de la página que nos interese, en este caso, el *popup*. Y creamos el *popup* inicial siguiendo la siguiente estructura (Figura 21):

```
<div id="modal-final-place"></div>

<div id="modal-inicio" class="modal fade">
  <div class="modal-dialog">
        <div class="modal-content">
          <div class="modal-header">
                <button id="cerrar-modal" type="button" class="close" data-dismiss="modal" aria-hidden="true">
                <h4 class="modal-title text-center">Iniciando Actividad</h4>
          </div>
          <div class="modal-body">
                <p>{{ actividad }}</p>
          </div>
          <div class="modal-footer">
                <button id="confirmInit" type="button" class="btn btn-primary" data-dismiss="modal">¡Pulsa par
          </div>
        </div>
  </div>
</div>
```

Figura 21 - *Popup* inicio de actividad

La lógica de las actividades, que estaba desarrollada con el *framework KineticJS* no la tocamos, simplemente modificamos el lugar donde se cargan los resultados de la actividad para que se carguen en el " " antes mencionado.

Para arrancar las actividades, se ejecuta un código *javascript* que se encuentra en el archivo "launcher.js", ubicado en la carpeta *"templates"*. Por tanto hemos de modificar este archivo para mostrar correctamente el *popup* inicial, y vincular correctamente las instrucciones que se han de ejecutar una vez el usuario quiera comenzar la actividad.

A continuación modificamos el *footer*, para adaptarlo a *Bootstrap* quedando del siguiente modo (Figura 22):

```
<div class="row">
        <div class="col-md-4">
            <audio id="audioagente" controls><source src="{% url play_audio tts actividad.id user.id %}
        </div>
        <div class="col-md-4">
                <p class="text-center">EMATIC - Universidad de La Laguna</p>
        </div>
        <div id="terminar-volver" class="col-md-4 text-right">
                <button id="send"  class="btn btn-success">Terminar</button>
        </div>
</div>
```

Figura 22 - *Footer* de actividades

Una vez modificado el *template* inicial hemos de modificar el *template* que se envía como respuesta a la petición Ajax de finalizar el juego con los datos de la partida corregida. Este *template* se compone de un *popup* en el que se muestran los resultados, y una lógica en *javascript* con la solución de la actividad y una actualización del *footer*. Nosotros solo modificaremos el *popup*, y la actualización del *footer*. El *popup* sigue la estructura del inicial antes mostrado, y en el *footer* se elimina el botón de terminar juego y se añaden dos nuevos, uno para repetir el ejercicio y otro para pasar al siguiente ejercicio.

De este modo, la pantalla inicial de las actividades queda así (Figura 23):

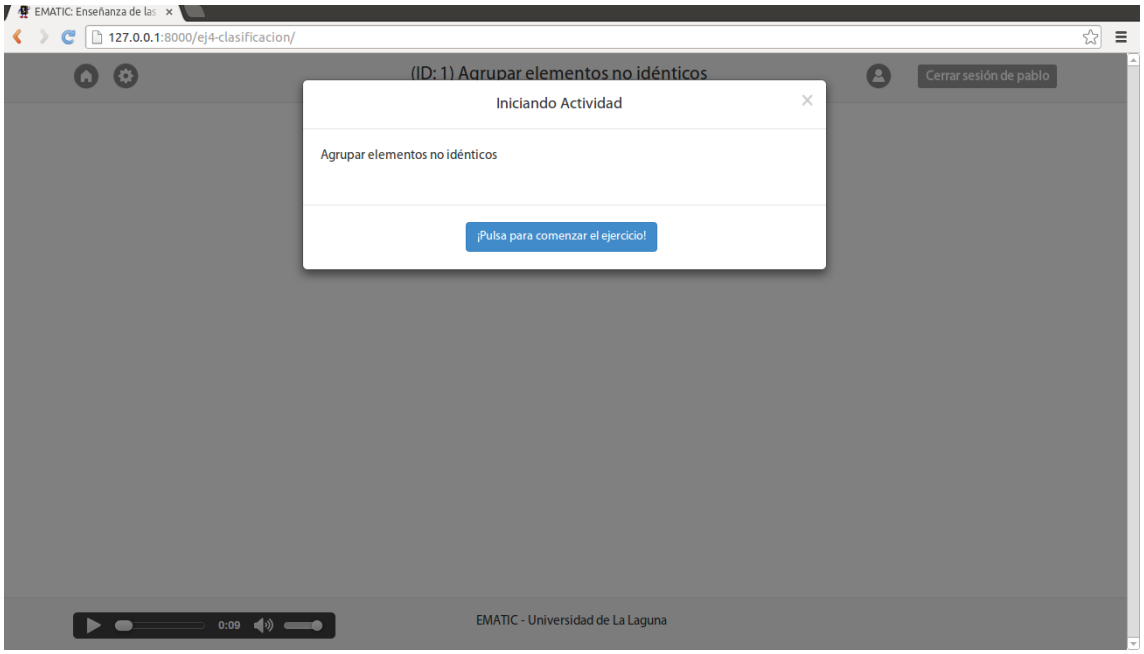


Figura 23 - Pantalla actividad 1 (diseño actual)

Una vez pulsamos el botón de comenzar ejercicio, nos aparece la actividad del siguiente modo (Figura 24):

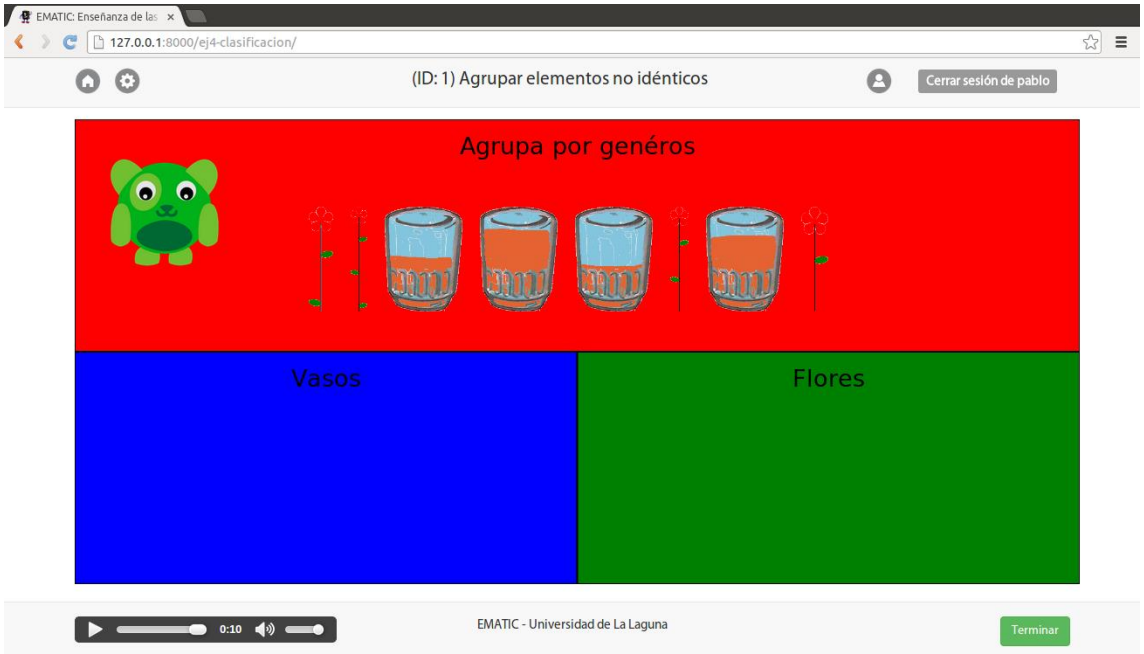


Figura 24 - Pantalla actividad 2 (diseño actual)

Una vez realizamos la actividad y pulsamos el botón de terminar, nos aparece la siguiente pantalla (Figura 25):

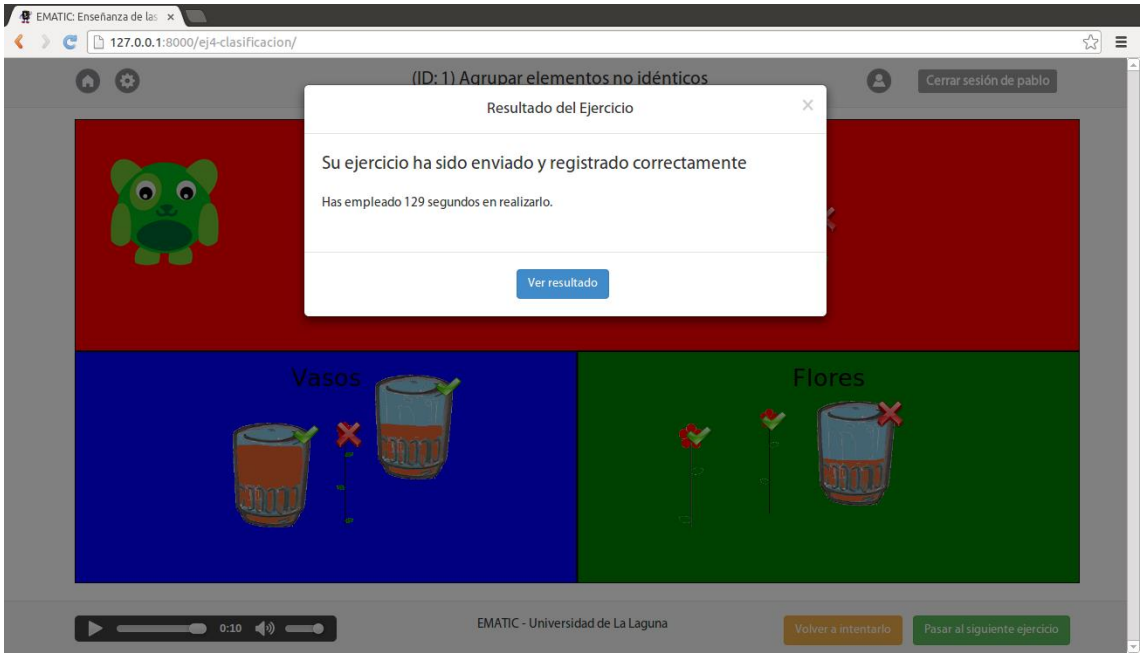


Figura 25 - Pantalla actividad 3 (diseño actual)

Si pulsamos el botón "Ver resultado", se muestra la siguiente pantalla (Figura 26):

Figura 26 - Pantalla actividad 4 (diseño actual)

De este modo, finalizamos la actualización visual de la aplicación.

Capítulo 4. Patrones de juegos colaborativos

Introducción

En este capítulo explicaremos el modelo que usaremos para promover la colaboración en los juegos que se desarrollarán a continuación, descubriremos las condiciones iniciales que deben especificarse, cómo se estructura la colaboración, y cómo podemos mantener la misma.

Modelo de aprendizaje colaborativo

Para asegurar que diseñamos un ambiente en el que se promueva la colaboración en cada uno de los juegos que se diseñarán en los siguientes capítulos decidimos utilizar el modelo descrito en el artículo "Designing Collaborative Learning Environments Using Digital Games" (Collazos, Guerrero, Pino, Ochoa, & Stahl, 2007).

El modelo propuesto involucra tres actividades interrelacionadas, las cuales proporcionan un *feedback* que permite realizar el mejor diseño para crear el mejor ambiente de aprendizaje colaborativo. El modelo intenta ayudar en la colaboración en dos direcciones: establecer la situación en la que se desarrollará la actividad colaborativa, y estructurar la colaboración (Collazos, Guerrero, Pino, Ochoa, & Stahl, 2007) (Figura 27).

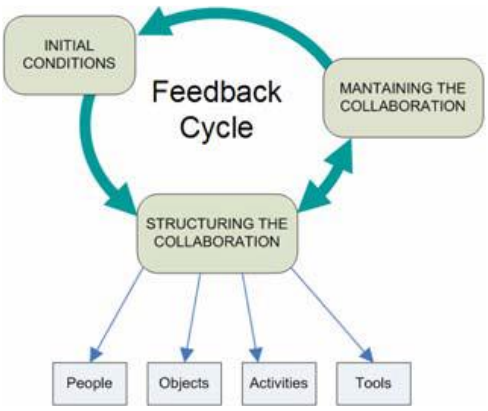


Figura 27 - Modelo de aprendizaje colaborativo

Condiciones iniciales

Como podemos observar, en primer lugar hemos de definir las condiciones iniciales en las que se desarrollará la colaboración con el objetivo de que la misma sea satisfactoria. Estas condiciones iniciales están formadas por un conjunto de elementos que vamos a explicar a continuación (Collazos, González, Gutiérrez, & Guerrero, 2014).

- Tipo de actividad: Debemos de definir claramente el tipo de actividad que será realizada por los miembros del grupo para resolver el problema. Por ejemplo, resolver un puzzle.
- Naturaleza de los colaboradores: Debemos definir el tipo de interacción que se va a realizar. Podemos encontrar tres tipos de interacciones: *peer-to-peer*, estudiante-profesor, o estudiante-ordenador.
- Heterogeneidad del grupo: Debemos definir el tamaño del grupo, el sexo de los miembros del grupo, y las diferencias entre los componentes del grupo. Habitualmente en los grupos más pequeños hay menor riesgo de exclusión de algunos de los participantes. Además, los grupos más pequeños requieren menos habilidades para la gestión de grupos y normalmente deciden más rápido.
- Interdependencias positivas: Este es uno de los elementos claves en el éxito del grupo. Basándonos en el trabajo de Collazos, Guerrero, Pino & Ochoa (2007), podemos identificar las siguientes interdependencias positivas:
    - Interdependencia positiva de meta: Los jugadores deben comprender que solo podrán terminar con éxito su aprendizaje solo si los demás miembros del grupo terminan con éxito sus metas. El grupo está motivado para conseguir una meta común, por lo tanto deben estar preocupados por cuanto aprenden los demás miembros, deben entender que caerán o tendrán éxito, pero siempre juntos.
    - Interdependencia positiva de recompensa: Podemos dar una recompensa común si se realiza un trabajo en grupo satisfactorio, de manera que los miembros se esfuercen por

conseguirla. Cuando el grupo va consiguiendo alcanzar metas, cada miembro recibe la misma recompensa.

- Interdependencia positiva de recursos: Cada miembro tiene una parte de la información, recursos, o materiales necesarios para completar una tarea, de modo que los miembros tienen que combinar estos recursos para que el grupo pueda alcanzar sus objetivos. Mientras los miembros no combinen sus recursos, el grupo no podrá completar la tarea con éxito.
- Interdependencia positiva de rol: Se asigna roles interconectados a cada miembro del grupo, de modo que sea necesaria la cooperación entre los distintos roles para completar la tarea con éxito.
- Interdependencia positiva de identidad: Los miembros del grupo tienen que encontrar y acordar una identidad, que puede ser un nombre, un lema, un lema, una bandera o una canción.
- Interdependencia ambiental: Los miembros del grupo están relacionados por el medio físico en el que trabajan. Si no hay un entorno físico común donde los miembros del grupo pueden jugar, debe existir un entorno virtual compartido en el que pueden resolver las tareas de forma colaborativa.
- Interdependencia positiva de fantasía: Se les da a los miembros del grupo una tarea imaginaria. Los estudiantes tienen que encontrar soluciones para situaciones extremas, tales como salvar la vida o la manipulación de una tecnología muy poderosa.
- Interdependencia positiva de tarea: El trabajo tiene que ser organizado de forma secuencial. Los miembros del grupo tienen que dividir el trabajo y estar vinculados entre sí. Tan pronto como un equipo lleva a cabo su parte de la tarea, el siguiente equipo puede continuar con su responsabilidad, y así sucesivamente.
- Interdependencia positiva de enemigo exterior: Se deben poner los grupos en competencia unos con otros. De esta manera, los miembros del grupo se sienten interdependientes

y hacen lo mejor para ganar la competición y estar por encima de los demás grupos.

- Ajuste de la colaboración: Corresponde al lugar donde se va a realizar la colaboración. Puede ser en un aula de clase, en un lugar de trabajo, en casa o en un espacio virtual.
- Condiciones de la colaboración: Especifica el tipo de mediación, como puede ser mediada por ordenador.
- Periodo de colaboración: Especifica el intervalo de tiempo en el que se realizará la actividad colaborativa. Se puede especificar en minutos, horas, días, semanas o meses.

Estructurando la colaboración

Además de definir las condiciones iniciales, se debe de especificar un proceso de colaboración, el cual puede incluir varias actividades. En cada actividad, el grupo tiene que conseguir un resultado, y los miembros del grupo tienen que jugar un rol. Los elementos propuestos para diseñar el proceso de colaboración son los siguientes (Collazos, González, Gutiérrez, & Guerrero, 2014):

- Actividades: Este elemento representa las tareas que deben ser realizadas por los miembros del grupo durante el proceso de colaboración. Incluye el flujo de trabajo que componen las actividades, incluyendo metas y reglas de cada actividad.
- Roles: Este elemento determina los roles que deben estar presentes en el proceso e colaboración. Cada miembro del grupo tiene un rol que jugar en cada actividad. Cada rol asigna responsabilidades y recompensas a los usuarios. Los roles deben rotar.
- Herramientas: Este elemento representa las herramientas a través de las cuales las personas pueden realizar las actividades colaborativas. Estas herramientas deben permitir que los colaboradores puedan comunicarse, coordinar y participar en el proceso. Los miembros del grupo deben comunicarse y coordinarse entre ellos con el fin de realizar las tareas que son independientes, que no están completamente descritas o que requieren negociación.
- Objetos: Los objetos representan el conocimiento que es compartido por los miembros del grupo durante una actividad. Este conocimiento puede incluir varios recursos, tales como objetos digitales, una parte de la interfaz de usuario, las estrategias de coordinación, decisiones, metas y mecanismos de sensibilización. Por ejemplo, la discusión de las estrategias para resolver un problema ayuda a los miembros del grupo a la construcción de una visión compartida (objeto compartido) de sus objetivos y tareas que se deben ejecutar.

Manteniendo la colaboración

El último aspecto a considerar está relacionado con la estrategia que se puede utilizar para mantener la colaboración entre los miembros del grupo. Tal estrategia podría ser realizada por un facilitador o por los miembros del equipo.

No hay garantías de que las interacciones entre los miembros del equipo ocurran en realidad. Por lo tanto, Se necesita una cierta regulación externa para que estas interacciones ocurran. Una manera de proporcionar ese tipo de regulación es a través del mediador cognitivo. El papel del mediador no será el de intervenir a nivel de tareas, pero sí garantizar que todos los miembros del grupo participan, y pedir con frecuencia preguntas como las siguientes: ¿Qué pasó? ¿Qué significa? El papel del mediador cognitivo es mantener el foco de la discusión, guiar a los estudiantes a través del proceso de construcción del conocimiento (Collazos, González, Gutiérrez, & Guerrero, 2014).

Capítulo 5. Mecánicas en juegos colaborativos

Introducción

En este capítulo veremos algunas mecánicas de juego que pueden ser incluidas en los mismos y explicaremos las mecánicas que hemos decidido incluir en nuestros juegos. Finalmente desarrollaremos una implementación del perfil del jugador donde se podrán ver los puntos acumulados que posee así como las medallas que ha conseguido.

¿Por qué es necesario jugar?

Los videojuegos estimulan este tipo de aprendizajes, y habitualmente proporcionan a quien juega muchas funcionalidades para motivar su autoaprendizaje: retos de dificultad creciente, posibilidad de pausar el juego o guardar la partida, todos los retos tienen solución y son justos, etc. Eso hace que podamos progresar siempre que lo intentemos, sin estar a merced de eventualidades externas (Garaizar, 2012).

Por tanto, ya que el objetivo de los juegos colaborativos que se desarrollarán es que los usuarios aprendan y mejoren sus habilidades, se hace necesario introducir mecánicas de juego que hagan atractiva la experiencia del usuario al jugar.

Mecánicas de juego

Según Gabe Zichermann y Christopher Cunningham (Zichermann & Cunningham, 2011) podemos incluir mecánicas en nuestros juegos como las siguientes:

- Puntos: Los puntos son importantes, independientemente de si su acumulación se comparte entre los jugadores, o incluso entre el diseñador y el jugador. Se puede decir que los puntos son un requisito indispensable en un sistema gamificado.
- Niveles: Con este sistema se premia la implicación del usuario en la actividad otorgándole un nivel o descripción con el que distinguirse del resto, y que anima a los usuarios nuevos a igualarles. Así se hace, por ejemplo, en los foros de discusión de Internet en los que en función de la participación del usuario se le asigna un nivel como 'Veterano', 'Novato', etc.
- Clasificaciones: Someten a los usuarios a un sistema de clasificación que tiene en cuenta su implicación en la actividad. De esta manera se explota el espíritu competitivo de los usuarios.
- Medallas: La gente desea medallas por todo tipo de razones. Para muchas personas, la recolección de medallas es una motivación poderosa. Para los diseñadores de juegos, las medallas son una excelente manera de fomentar la promoción social de sus productos y servicios. Además las medallas también marcan la culminación de los objetivos y el progreso constante del juego dentro del sistema.

Mecánicas incluidas en los juegos

Las mecánicas que se incluyen en nuestros juegos colaborativos son las siguientes:

- Se crea un sistema de puntuaciones, de modo que cada jugador recibe puntos al finalizar un juego, los cuales se van acumulando en su perfil. Para hacer posible que la puntuación sea guardada correctamente se añade la clase PuntuacionJugador al modelo de datos (Figura 28).

```
class PuntuacionJugador(models.Model):
    user = models.ForeignKey(User)
    puntuacion = models.PositiveIntegerField()
    memo = models.PositiveIntegerField()
    sudoku = models.PositiveIntegerField()

    def __unicode__(self):
        return unicode(self.puntuacion)
```

Figura 28 - Clase PuntuacionJugador

Como se puede observar, en esta clase almacenamos la puntuación acumulada del jugador, y el nivel máximo que ha alcanzado en el juego de Sudoku y de memoria.

- Se crean tres categorías de jugadores. Cada jugador es asignado a una categoría según los puntos que tiene acumulados. Las categorías son las siguientes (Tabla 1):

Categoría	Puntos necesarios para alcanzar la categoría
Novato	0 puntos
Intermedio	9000 puntos
Experto	18000 puntos

Tabla 1 - Categorías de jugadores

- Se desarrollan tres niveles de dificultad para cada juego.

- Se dan las siguientes medallas por completar los siguientes objetivos (Tabla 2):

Medalla	Objetivo a completar
¡Eres un jugador novato!	Iniciar sesión
¡Eres un jugador intermedio!	Alcanzar 9000 puntos
¡Eres un jugador experto!	Alcanzar 18000 puntos
Novato en Sudoku	Superar nivel 1 del Sudoku
Intermedio en Sudoku	Superar nivel 2 del Sudoku
Experto en Sudoku	Superar nivel 3 del Sudoku
Novato en memoria	Superar nivel 1 de memoria
Intermedio en memoria	Superar nivel 2 de memoria
Experto en memoria	Superar nivel 3 de memoria

Tabla 2 - Sistema de medallas

Perfil del jugador

Con el objetivo de que el jugador pueda visualizar el número de puntos acumulados que posee, su categoría de jugador, y las medallas obtenidas, se crea la pantalla de perfil de usuario.

En primer lugar se crea el *template "puntuación_usuario.html"*, y se ubica en la carpeta *"templates/web"*. A continuación creamos una nueva función en el archivo *"views.py"* desde la que solicitaremos la información de puntuaciones al modelo de datos, y posteriormente renderizamos el *template "puntuación_usuario.html"* pasándole la información que hemos solicitado al modelo (Figura 29).

```
@login_required
@alumno_required
def perfil_jugador(request):

    p_usuario = PuntuacionJugador.objects.get(alumno=request.user.alumno_set.all()[0])

    return render_to_response("web/puntuacion_usuario.html",
                              {'p_usuario' : p_usuario },
                              context_instance=RequestContext(request))
```

Figura 29 - View de perfil de jugador

A continuación desarrollamos el código del *template "puntuación_usuario.html"*, mostrando la puntuación del usuario, su categoría y las medallas que ha logrado. Obtenemos como resultado final la siguiente pantalla (Figura 30):

Figura 30 - Perfil del jugador

Capítulo 6. Juego colaborativo de Sudoku

Introducción

En este capítulo describiremos en que consiste el juego del Sudoku. A continuación, haremos un análisis de los patrones de juego que incluiremos y finalmente, explicaremos como se ha implementado aportando varias capturas de pantalla.

¿Qué es un Sudoku?

El Sudoku es un rompecabezas matemático de colocación que se popularizó en Japón en 1986 y se dio a conocer en el ámbito internacional en 2005.

El objetivo es rellenar una cuadrícula de 9×9 celdas dividida en subcuadrículas de 3×3 con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas.

No se debe repetir ninguna cifra en una misma fila, columna o subcuadrícula. (Sudoku-online.org, 2014)

En nuestro caso, al tratarse de actividades dirigidas a niños pequeños con NEE y NEAE hemos decidido realizar una adaptación del juego utilizando cuadrículas de 2x2 celdas, y números del 1 al 4.

Patrones de juego

Siguiendo el modelo descrito en el Capítulo 3, se configuran las siguientes condiciones iniciales para el juego:

- Tipo de actividad
    - Resolver un Sudoku
- Naturaleza de los colaboradores
    - Interacción *peer-to-peer*
- Heterogeneidad del grupo
    - Tamaño del grupo: 2 personas
    - Género (Sexo): Mixto
    - Diferencias dentro del grupo: El juego se desarrolla entre niños de un rango de edad entre 9 y 14 años, en el que pueden existir diferencias en su nivel intelectual y a nivel cognitivo.
- Interdependencias positivas
    - Interdependencia positiva de meta: Esta interdependencia existe porque cada miembro del grupo necesita que el otro colabore para lograr completar el juego. El juego terminará con éxito solo si los dos miembros del grupo colaboran.
    - Interdependencia positiva de rol: Esta interdependencia existe porque los jugadores juegan dos roles, el primero es el de jugador realizando movimientos de números, y el segundo es el de evaluador, aprobando o descartando el movimiento que ha hecho su compañero.
    - Interdependencia positiva de recompensa: Esta interdependencia existe porque cada jugador recibe una puntuación individual, que al final se sumarán dando como resultado una puntuación global.
- Ajuste de la colaboración
    - Mediada por ordenador o en clase
- Condiciones de la colaboración
    - Mediada por ordenador
- Periodo de colaboración
    - 15-30 minutos

La forma de estructurar la colaboración será la siguiente:

- Actividades
    - Global: Resolver el sudoku
    - Parcial: Completar el mayor número de huecos correctamente
    - Reglas: Cada jugador puede manipular cualquier hueco vacío que esté disponible. Si el juego no se resuelve en un determinado número de turnos, el juego se acaba.

- Roles
    - Jugador: Realiza un movimiento de número
    - Evaluador: Decide si el movimiento realizado por el compañero se mantiene o se descarta
- Herramientas
    - Tabletas u ordenador
- Objetos
    - Tablero de sudoku
    - Set de números que faltan para completar el sudoku comunes a ambos jugadores
    - Colección de trofeos

Por último se acuerdan las siguientes pautas para mantener la colaboración:

- Existe un facilitador en forma de personaje del juego que explicará el juego y orientará a los jugadores
- Fase inicial: Al inicio del juego se sincroniza a los jugadores decidiendo quien comienza a jugar primero.
- Fase de desarrollo: Durante el desarrollo del juego, los movimientos que realiza un jugador deben ser aprobados por el compañero.
- Fase de conclusión: Ambos jugadores deben aprobar que están de acuerdo con el sudoku que ha quedado al final de la partida. Si no están de acuerdo, no puede acabar la partida correctamente.

Implementación del juego

En primer lugar definimos un modelo de datos para identificar lo que es un juego en la aplicación multidispositivo. Esto nos servirá para introducir los 3 niveles del Sudoku y los 3 niveles del juego de memoria. El modelo, desarrollado en el archivo *"models.py"*, es el siguiente (Figura 31):

```
class Juego(models.Model):
    titulo = models.CharField(max_length = 100)
    descripcion = models.CharField(max_length = 250) # El WS de TTS de Google sól
    texto_resumen = models.CharField(max_length = 200, null = True, blank = True)
    nombre_vista = models.CharField(max_length = 100)

    def __unicode__(self):
        return unicode(self.titulo)
```

Figura 31 - Modelo de datos de los juegos

En "titulo" guardamos el nombre del juego, por ejemplo, "Sudoku nivel 1". En "descripcion", el texto que queremos que el agente diga al inicio del juego, en "texto_resumen" una explicación de en qué consiste el juego, y en "nombre_vista" la url del *template* del juego.

Además, en este juego hemos necesitado crear dos clases más en el modelo de datos para almacenar los sudokus que tienen que cargarse al principio del juego, y las soluciones asociadas a estos sudokus. Estas clases son "InicialSudoku" y "SolucionSudoku", y en ellas tenemos un campo para identificar al sudoku y a su solución correspondiente, un campo donde está almacenado el nivel del Sudoku y de su solución, y 16 campos para almacenar los números que se deben cargar en cada posición. En estos 16 campos podemos encontrar números desde el 1 al 4, y en la clase "InicialSudoku" además podemos encontrar algún 0, que significa que esa posición está vacía al iniciar el juego. Además la clase "InicialSudoku" tiene un campo extra para almacenar la *url* del *template* que carga la solución una vez se termine el juego.

A continuación añadimos las rutas para cargar el inicio de los tres niveles y la carga de soluciones para cada nivel en el archivo *"urls.py"*.

El siguiente paso es crear una función para cada nivel en el archivo *"views.py"*. Estas funciones tienen como nombre "juego_colaborativo_sudoku_*nivel*". En cada una de estas funciones cargamos del modelo los datos del juego que vamos a jugar, la imagen de los números del 1 al 4, y la posición inicial de un Sudoku seleccionada aleatoriamente entre las disponibles del nivel en el que nos encontramos. Tras la carga de datos, renderizamos el *template* de inicio de juego con los datos que hemos solicitado al modelo.

A continuación creamos un *template* para cada nivel del juego al que llamaremos *"juego_colaborativo_sudoku_nivel.html"*. En cada uno de estos *template*s desarrollaremos el código para crear el tablero de juego, colocar los elementos en sus posiciones, y permitir que los jugadores jueguen. Para desarrollar la creación del tablero, y permitir que los jugadores puedan desplazar los números por el tablero se hace uso del *framework KineticJS* 5.1.0.

En primer lugar creamos el escenario en el que colocaremos el tablero de juego del siguiente modo (Figura 32):

```
var stage = new Kinetic.Stage({
  container: "container",
  width: window.innerWidth - 150,
  height: window.innerHeight - 250
});
```

Figura 32 - Crear escenario

En este escenario que hemos definido colocaremos todos los elementos que genera el *framework KineticJS*. El primer elemento que creamos es el tablero, compuesto de 17 rectángulos, el primero rectángulo será el lugar donde se coloquen los números que no tienen una posición asociada en las posiciones iniciales del sudoku, y los 16 rectángulos restantes representan cada uno una posición del tablero del Sudoku.

El siguiente paso es cargar en una variable de *javascript* la posición inicial del Sudoku que hemos enviado desde la función de *"views.py"*, y recorrer las 16 posiciones del tablero del Sudoku rellenándolas con números si para esa posición

hay un número asociado. Hemos de recordar que en algunas posiciones encontraremos que desde *"views.py"* se carga un número 0, lo que significará que hemos de dejar esa posición del Sudoku vacía. Los números que se cargan en este recorrido no podrán ser movidos por los usuarios, esto lo conseguimos colocando el atributo *"draggable"*, propio del *framework KineticJS*, como false al cargar el número.

Al realizar el recorrido por todas las casillas antes mencionado, llevamos la cuenta de la cantidad de números 1, 2, 3 y 4 que se han insertado, de modo que al acabar el recorrido sabemos cuántos número 1, 2, 3, y 4 faltan por colocar en el tablero, pues siempre tiene que haber 4 números 1, 4 números 2, 4 números 3 y 4 números 4. A continuación colocamos en el primer rectángulo del que habíamos hablado anteriormente todos estos números que faltan, colocándolos con la propiedad *"draggable"* en true, de modo que se podrán mover, y colocándolos dentro de un cuadrado rojo sombreado, diferenciándolos de este modo del resto de números, para que el usuario tenga claro que puede ser movido a lo largo del juego.

A continuación creamos los *popup* necesarios para que el juego funcione, que son, finalizar el turno de un jugador, el cual se activa automáticamente cuando un jugador realiza el movimiento de un número de un modo correcto, finalizar la partida, que se activa automáticamente cuando un jugador pulsa el botón de finalizar partida, y el de inicio del juego.

Como hemos dicho, el *popup* de finalizar turno se ejecuta cuando un jugador realiza un movimiento válido, y en él se le pregunta al jugador con el que colabora si está de acuerdo con el movimiento o no. El jugador con el que colabora seleccionará si está de acuerdo o no, y jugará su turno. Si el jugador con el que colabora selecciona que no está de acuerdo con el movimiento, se revertirá el movimiento realizado.

El *popup* de finalizar la partida se ejecuta al pulsar el botón de finalizar partida, y en él se le pregunta al jugador con el que se colabora si está de acuerdo con que se acabe la partida o no. Si el jugador que colabora está de acuerdo se finaliza la partida, si no, se acaba el turno del jugador que solicitó que se finalizara la partida y comienza el del jugador con el que colabora.

La ventana emergente inicial nos da una instrucciones para jugar al Sudoku, y tras pulsar el botón de comenzar partida comenzamos el juego.

El siguiente paso es añadir un límite de turnos al juego. Para eso creamos una variable *javascript* que vaya contando los turnos que quedan por jugarse. Se inicializa en un valor distinto para cada uno de los 3 niveles, en el caso del nivel 1 es de 20 turnos, en el del nivel 2 de 18 turnos, y en el nivel 3 de 18 turnos. Cada vez que mostramos el *popup* de fin de turno o el de fin de partida restamos en una unidad esta variable, y si llega a 0 en algún momento, finalizamos el juego automáticamente.

A continuación incluimos las puntuaciones en el juego. Cada jugador posee una puntuación individual, y en el caso de ambos es de 100 puntos. Esta puntuación nunca puede verse incrementada, por lo que para todos los niveles la puntuación máxima que se puede conseguir es de 200 puntos en total, sumando los 100 de cada jugador. Sin embargo, esta puntuación si puede aminorar. Si al finalizar un turno el jugador 1, el jugador 2 que debe seleccionar si está de acuerdo con el movimiento realizado por el jugador 1 o no selecciona que no está de acuerdo, se restarán puntos a ambos jugadores. En el nivel 1, se le restarán 10 puntos al jugador 2 y 15 al jugador 1, en el nivel 2, se le restarán 10 puntos al jugador 2 y 15 al jugador 1 y en el nivel 3, se le restarán 15 puntos al jugador 2 y 20 al jugador 1. Además, si un jugador selecciona finalizar la partida, y el jugador con el que colabora selecciona que no está de acuerdo también se restarán puntos. En todos los niveles se restarán 10 puntos a ambos jugadores (Tabla 3).

Acción	**Puntuación Jugador 1**	**Puntuación Jugador 2**
Inicio del juego	100	100
Nivel 1: Jugador 1 rechaza movimiento del jugador 2	-10	-15
Nivel 2: Jugador 1 rechaza movimiento del jugador 2	-10	-15
Nivel 3: Jugador 1 rechaza movimiento del jugador 2	-15	-20

Un jugador rechaza la proposición de finalizar la partida del otro jugador	-10	-10

Tabla 3 - Puntuación de Sudoku

Ahora creamos la función Ajax que se ejecutará cuando ambos jugadores quieran finalizar la partida, o cuando se acaben los turnos disponibles. En ella primero creamos una estructura de datos para guardar el número que se encuentra en cada posición el Sudoku y la completamos. Además calculamos el total de números que se encuentran colocados en las 16 posiciones del tablero de Sudoku, cuantos números 1, 2, 3, y 4 han sido colocados en las 16 posiciones del Sudoku, la cantidad de turnos que quedan por jugarse, y las puntuaciones del usuario. Para acabar con la función Ajax llamamos a la función que crearemos en *"views.py"* enviándole los datos que acabamos de mencionar, y cargamos el resultado, que será un nuevo *template*, en una etiqueta vacía previamente creada.

Para finalizar con este *template "juego_colaborativo_sudoku_nivel.html"* creamos una estructura HTML con *Bootstrap* para mostrar correctamente al usuario el agente, las puntuaciones, los turnos restantes, el tablero, y el *footer*, tal y como se muestra a continuación (Figura 33):

```
{% block content %}
<div class="row">
        <div id="texto-puntuacion" class="col-md-5">
                        <p id="texto_puntuacion" class="cabeceras-sudoku"></p>
        </div>
        <div id="texto-sudoku" class="col-md-3">
                        <p id="texto_turno" class="cabeceras-sudoku">  </p>
        </div>
        <div id="texto-movimientos" class="col-md-4">
                        <p id="texto_movimientos" class="cabeceras-sudoku">  </p>
        </div>
    <div id="container" class="col-md-12" style="display: none; margin: auto;">
    </div>
</div>
    <div id="dim">
    </div>
{% endblock %}
```

Figura 33 - Código Sudoku

A continuación creamos en el archivo *"views.py"* las funciones que serán llamadas en las funciones Ajax desde cada uno de los *template*s que acabamos de ver. Cada una de estas tres funciones se llamará "ajax_results_sudoku_*nivel*". En cada una de ellas en primer lugar rescatamos los datos que se nos envían en el POST de la función Ajax, después creamos una variable que nos sirve para saber si el juego ha sido resuelto de forma correcta o no, y la inicializamos a verdadero. Solicitamos al modelo la solución del Sudoku, y vamos comparando con un bucle si la solución enviada por el usuario es correcta o no. Si detectamos un error, guardamos esa posición en una variable y colocamos como falsa la variable que guarda si se ha solucionado correctamente el juego.

El siguiente paso es calcular la puntuación conseguida. Si el juego se ha resuelto correctamente se suman las puntuaciones de los jugadores a la puntuación global del jugador que ha iniciado sesión, y se guarda como nuevo nivel alcanzado en el que se encuentre, siempre y cuando sea mayor al máximo nivel alcanzado que se encuentra guardado en la base de datos. Si no se resuelve correctamente la puntuación obtenida es de 0 puntos.

Finalmente en este archivo renderizamos el *template* que mostrará al usuario su puntuación en la pantalla, si ha ganado o no y sus errores.

El último paso es crear los *template*s que acabamos de mencionar, uno para cada nivel. Tendrán como nombre *"solucion-sudoku-nivel.html"*. En ellos simplemente cargaremos un *popup* en el que mostramos al usuario si ha ganado o perdido, su puntuación, y el tiempo empleado, modificamos los textos que se mostraban encima de la pantalla, modificamos el *footer* para que el usuario pueda volver a jugar, y colocamos unas cruces rojas en las posiciones del Sudoku que no se han completado correctamente.

Para finalizar mostraremos 5 capturas de pantalla de como ha quedado el juego tras realizar este trabajo.

En primer lugar mostramos la pantalla inicial del juego (Figura 34):

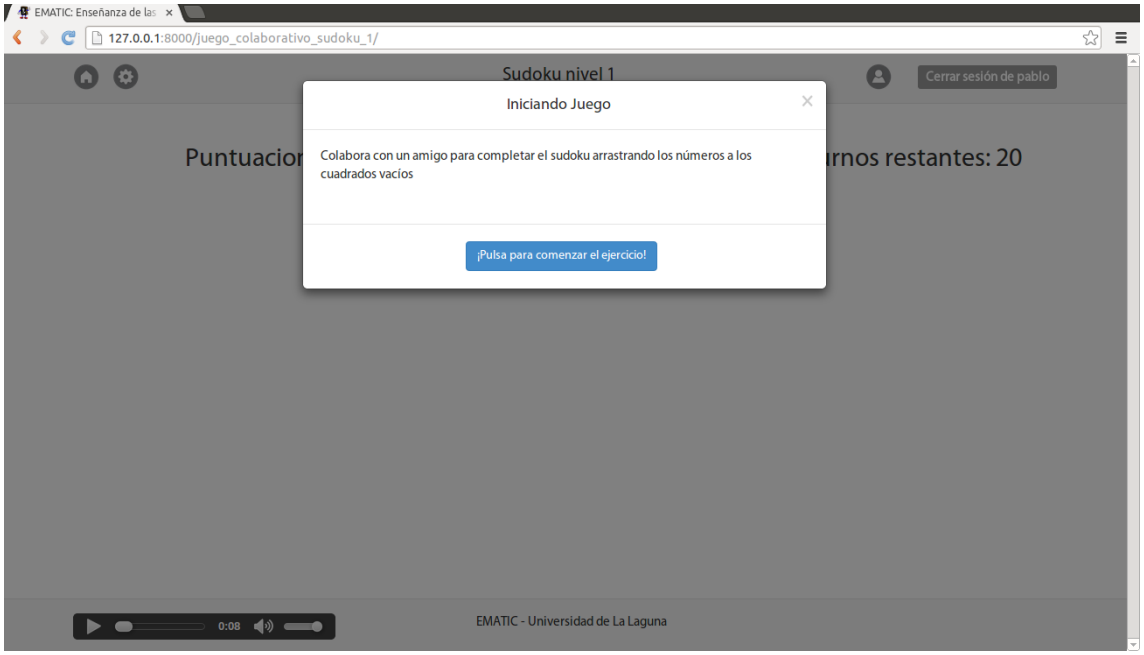


Figura 34 - Pantalla Sudoku 1

A continuación mostramos como queda la pantalla tras pulsar comenzar el ejercicio (Figura 35):

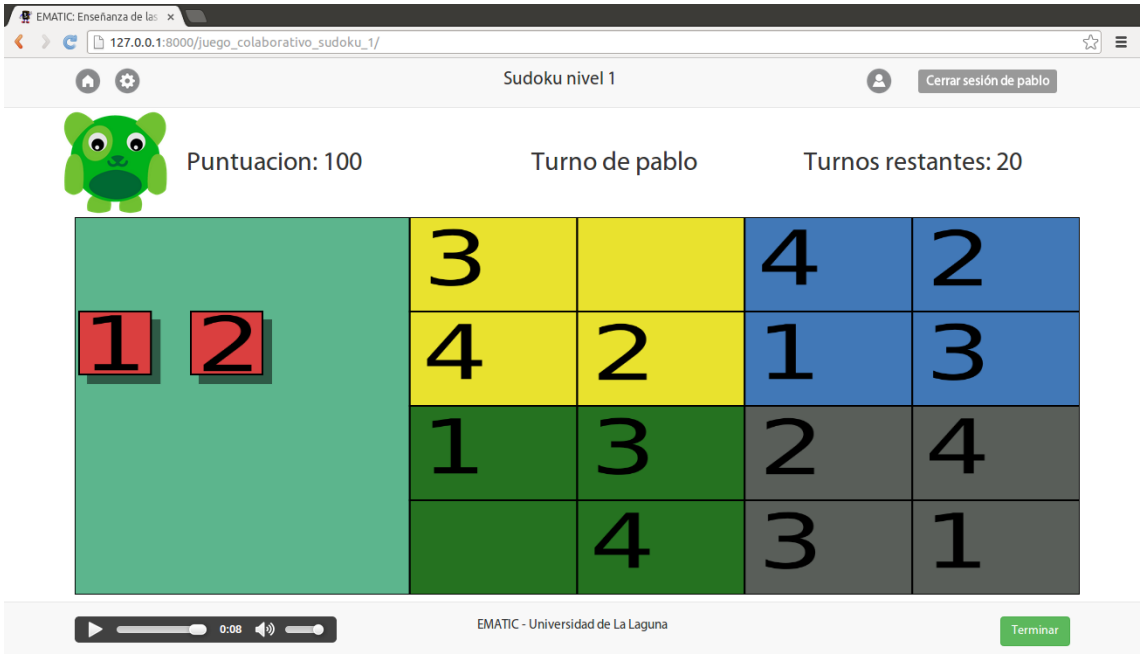


Figura 35 - Pantalla Sudoku 2

Mostramos ahora el *popup* que se muestra cuando un jugador realiza un movimiento correcto (Figura 36):

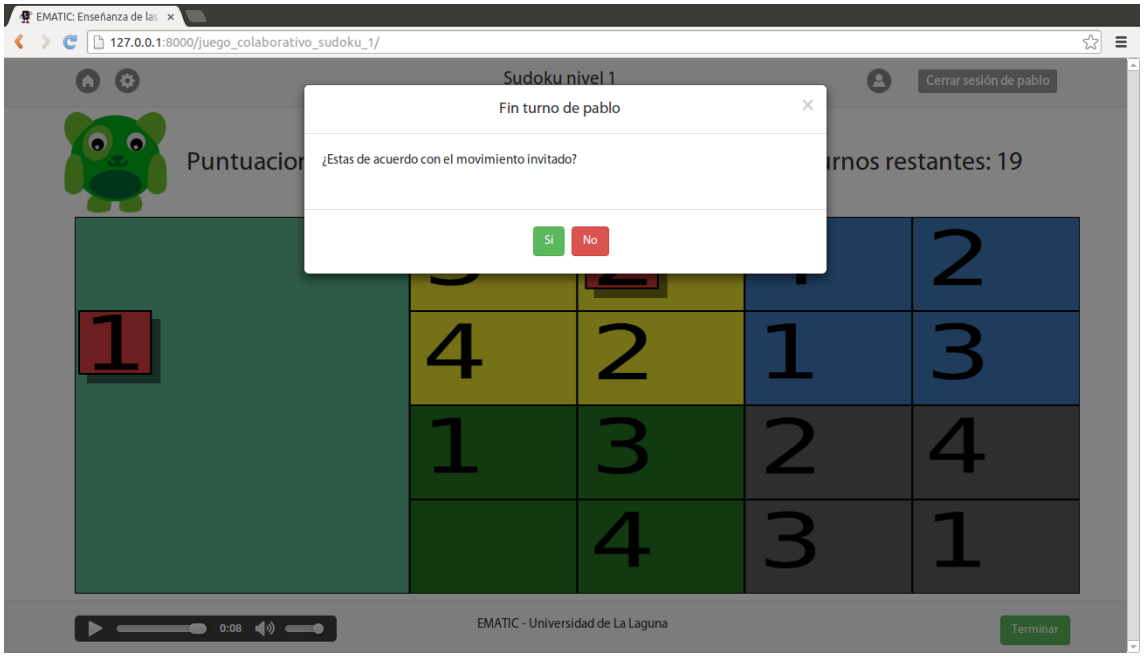


Figura 36 - Pantalla Sudoku 3

La siguiente captura se produce cuando un jugador pulsa el botón de terminar (Figura 37):

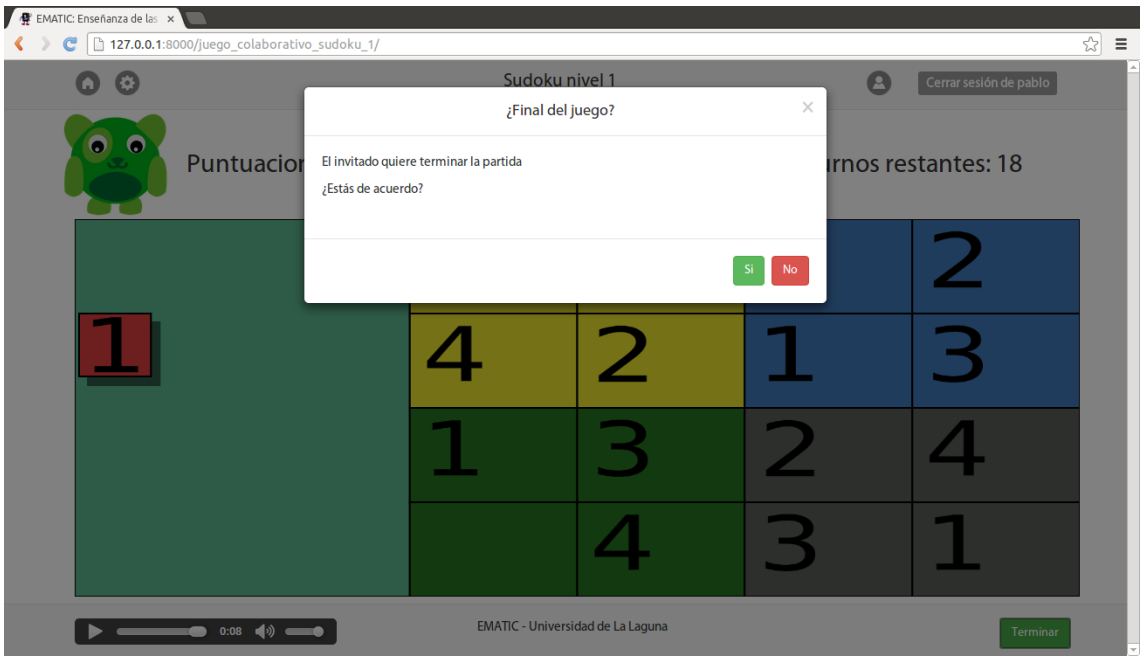


Figura 37 - Pantalla Sudoku 4

Finalmente mostramos la pantalla de final del juego, una vez ambos acuerdan acabar, o se le acaban los turnos restantes (Figura 38).

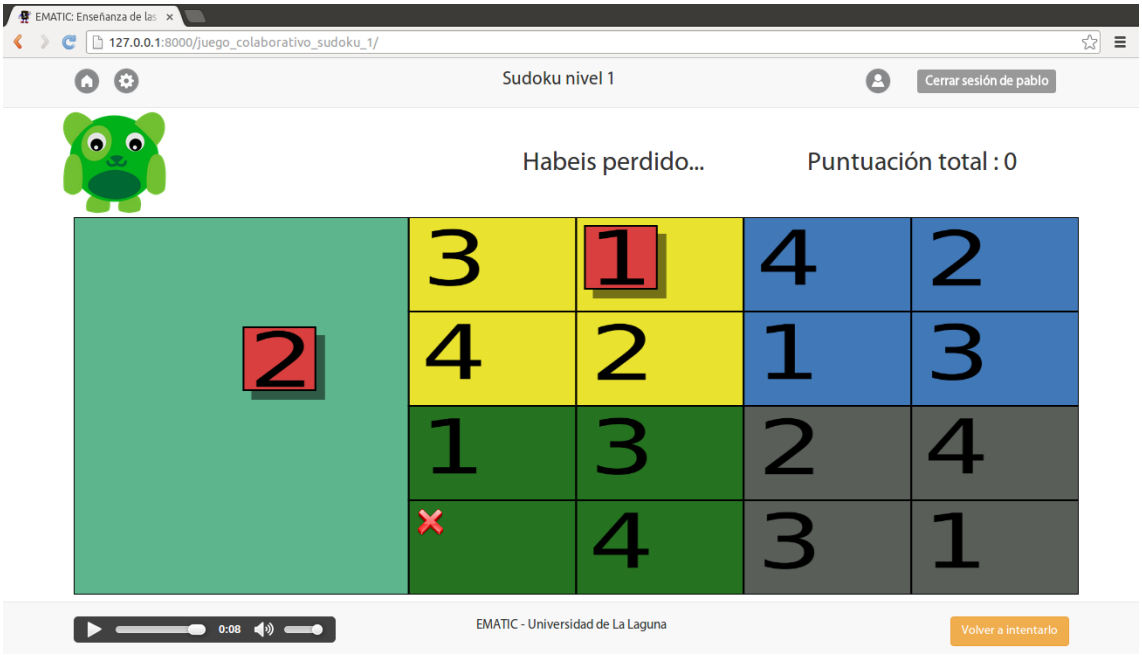


Figura 38 - Pantalla Sudoku 5

De este modo finalizamos el juego del Sudoku colaborativo.

Capítulo 7. Juego colaborativo de memoria

Introducción

En este capítulo describiremos en que consiste el juego colaborativo de memoria, haciendo un análisis de los patrones de juego que incluiremos y finalmente, explicaremos como se ha implementado este juego, presentando varias capturas de pantalla.

¿En qué consiste el juego?

Este juego consiste en descubrir parejas de imágenes iguales. Al comenzar el juego se muestran una serie de parejas de imágenes ocultas, de modo que los jugadores ven unos círculos azules, ocultándose las verdaderas imágenes tras estos círculos.

Ambos jugadores tendrán que colaborar para descubrir todas las parejas de cartas, y por tanto ganar el juego, si no descubren todas las parejas ocultas, pierden.

Patrones de juego

Siguiendo el modelo descrito en el Capítulo 3, se configuran las siguientes condiciones iniciales para el juego:

- Tipo de actividad
    - Descubrir todas las parejas de cartas que se encuentran ocultas en el tablero
- Naturaleza de los colaboradores
    - Interacción *peer-to-peer*
- Heterogeneidad del grupo

    - Tamaño del grupo: 2 personas
    - Género (Sexo): Mixto
    - Diferencias dentro del grupo: El juego se desarrolla entre niños de un rango de edad entre 9 y 14 años, en el que pueden existir diferencias en su nivel intelectual y a nivel cognitivo.

- Interdependencias positivas
    - Interdependencia positiva de meta: Esta interdependencia existe porque cada miembro del grupo necesita que el otro colabore para lograr completar el juego. El juego terminará con éxito solo si los dos miembros del grupo colaboran.
    - Interdependencia positiva de rol: Esta interdependencia existe porque existen dos roles de jugador, el primero corresponde al que levanta la primera carta, y el segundo corresponde al que tiene que buscar la pareja de la carta. Estos roles se intercambian durante el juego.
    - Interdependencia positiva de recompensa: Esta interdependencia existe porque cada jugador recibe una puntuación individual, que al final se sumarán dando como resultado una puntuación global.
- Ajuste de la colaboración
    - Mediada por ordenador o en clase
- Condiciones de la colaboración
    - Mediada por ordenador
- Periodo de colaboración
    - 15-30 minutos

La forma de estructurar la colaboración será la siguiente:

- Actividades
    - Global: Descubrir todas las parejas
    - Parcial: Encontrar la pareja de la carta que el compañero ha descubierto
    - Reglas: Si se descubre la pareja correctamente la pareja se queda virada y se cambian los roles de los jugadores, si no, se vuelven a

ocultar ambas cartas. Si el juego no se resuelve en un determinado número de turnos, el juego se acaba.

- Roles
    - Jugador que levanta la primera carta: El jugador elige cualquier carta que se encuentre virada en el tablero.
    - Jugador que busca la pareja: El jugador vira la carta que es pareja de la carta que se encuentra ya virada.
- Herramientas
    - Tabletas u ordenador
- Objetos
    - Tablero de cartas
    - Cartas
    - Colección de trofeos

Por último se acuerdan las siguientes pautas para mantener la colaboración:

- Existe un facilitador en forma de personaje del juego que explica el juego y orienta a los jugadores
- Fase inicial: Al inicio del juego se sincronizan a los jugadores decidiendo que rol juega cada uno para comenzar.
- Fase de desarrollo: Durante el desarrollo del juego, se permite que los jugadores compartan información y se aconsejen que cartas levantar.

Implementación del juego

Como ya hemos definido el modelo de datos para identificar juegos al crear los Sudokus, ahora simplemente creamos 3 nuevos registros, uno para cada nivel del juego. Recordamos que la clase Juego estaba formada por "titulo", "descripcion", "texto_resumen" y "nombre_vista". Procedemos por tanto a crear los registros.

En "titulo" guardamos el nombre del juego, por ejemplo, "Memo nivel 1". En "descripcion", el texto que queremos que el agente diga al inicio del juego, en "texto_resumen" una explicación de en qué consiste el juego, y en "nombre_vista" la *url* del *template* del juego.

Después añadimos las rutas para cargar el inicio de los tres niveles y la carga de soluciones para cada nivel en el archivo *"urls.py"*.

El siguiente paso es crear una función para cada nivel en el archivo *"views.py"*. Estas funciones tienen como nombre "juego_colaborativo_cartas_*nivel*". En cada una de estas funciones cargamos del modelo los datos del juego que vamos a jugar, las imágenes que forman las parejas que hay que descubrir en el juego, y la imagen del círculo azul que se muestra como reverso de las parejas de imágenes. Tras la carga de datos, renderizamos el *template* de inicio de juego con los datos que hemos solicitado al modelo.

A continuación creamos un *template* para cada nivel del juego al que llamaremos *"juego_colaborativo_sudoku_nivel.html"*. En cada uno de estos *templates* desarrollaremos el código para colocar las imágenes en su posición correcta, y permitir que los jugadores puedan jugar la partida.

En primer lugar creamos la estructura HTML donde vamos a colocar los distintos elementos que componen el juego, y que es la siguiente (Figura 39):

```
{% block content %}
<div class="row">
    <div id="container" class="col-md-12">

        <div id="turnos" class="col-md-12">
                <p id="texto_turno" class="centrado2"> Turno de {{ user }} </p>
        </div>

    <div id="fila1" class="col-md-12">
        </div>
    <div id="fila2" class="col-md-12">
        </div>

        <div id="turnos2" class="col-sm-6 col-md-6">
                <p id="resto_turno" class="centrado3"></p>
        </div>

        <div id="puntuacion" class="col-sm-6 col-md-6">
                <p id="puntuaciones" class="centrado3"></p>
        </div>

        </div>
</div>
    <div id="dim">
    </div>
{% endblock %}
```

Figura 39 - Estructura HTML del juego de memoria

Seguidamente añadimos las cartas a sus filas correspondientes usando *javascript*. Hemos de insertar un círculo azul por cada una de las imágenes que se deben descubrir.

El siguiente paso es ocultar las imágenes usando el método de JQuery ".hide()" (Figura 40).

```
$("#carta1").hide();
$("#carta2").hide();
$("#carta3").hide();
$("#carta4").hide();
$("#carta5").hide();
$("#carta6").hide();
```

Figura 40 - Ocultar imágenes

A continuación hemos de asociar unas instrucciones al evento *"click"* de los círculos azules que hemos insertado y que son visibles para el usuario al comienzo del juego, en lugar de las imágenes. Estas instrucciones son, en primer lugar, ocultar el círculo azul, y mostrar la imagen asociada a ese círculo que estaba oculta. A continuación comprobamos si es la primera imagen que se descubre, o si ya existe otra descubierta. Si es la primera imagen, guardamos en una variable el ID de la imagen que ha sido descubierta, y finalizamos el turno.

Si ya existe otra imagen descubierta, comprobamos si las dos imágenes que se han descubierto son iguales. Si las dos imágenes son iguales comprobamos si quedan parejas de imágenes por descubrir, en caso de que no queden finalizamos el juego, si quedan parejas por descubrir, dejamos la parejas de imágenes descubiertas hasta el final del juego y el jugador que descubrió la segunda imagen continuado jugando, no se finaliza el turno.

En el caso de que las imágenes descubiertas sean distintas, si no quedan turnos finalizamos el juego, y si quedan turnos ocultamos las dos imágenes, mostramos nuevamente los círculos azules, y finalizamos el turno (Figura 41).

```
$("#reverso_carta1" ).click(function() {
    if (ira == 0) {
            $("#reverso_carta1").hide();
            $("#carta1").show();
            if (primera_carta == 0) {
                  ira = 1;
                  setTimeout( function () {
                          primera_carta = 1;
                          if (jugador == 1) {
                                  jugador = 2;
                                  $('#initjugador1').modal('show');
                                  $("#texto_turno").html("Turno del invitado");
                                  $("#puntuaciones").html("Puntuación del invitado : " + puntuacion_invitado)
                          } else {
                                  jugador = 1;
                                  $('#initjugador2').modal('show');
                                  $("#texto_turno").html("Turno de {{ user }}");
                                  $("#puntuaciones").html("Puntuación de {{ user }} : " + puntuacion_usuario)
                          }
                          ira = 0;
                          turnos_restantes--;
                          $("#resto_turno").html("Turnos restantes: " + turnos_restantes);
                  }, 1500);
            }
            else {
                  segunda_carta = 1;
                  comprobar_igualdad();
            }
    }
});
```

Figura 41 - Código asociado al evento *click* de los círculos azules

A continuación creamos los *popup* necesarios para que el juego funcione, que son, finalizar el turno de un jugador, el cual se activa automáticamente cuando un jugador hace *click* en un círculo azul y no hay otra imagen descubierta, un *popup* para mostrar que se ha descubierto una pareja de imágenes correctamente, un *popup* para mostrar que no se ha descubierto una pareja de cartas correctamente, y el de inicio del juego.

El *popup* inicial nos da una instrucciones para jugar, y tras pulsar el botón de comenzar partida comenzamos el juego. El *popup* de finalizar el turno informa del que el turno del jugador que estaba jugando se ha acabado y debe de jugar el otro jugador a continuación.

La ventana emergente para mostrar que se ha descubierto correctamente una pareja de cartas, felicita a los jugadores por conseguirlo, e informa de que no se finaliza el turno del jugador que descubrió la segunda carta, pues en este momento se cambian los roles de los jugadores.

El *popup* para mostrar que no se ha descubierto una pareja de imágenes correctamente, informa a los jugadores del fallo, y del final del turno del jugador que descubrió la segunda imagen.

El siguiente paso es añadir un límite de turnos al juego. Para eso creamos una variable *javascript* que vaya contando los turnos que quedan por jugarse. Se inicializa en un valor distinto para cada uno de los 3 niveles, en el caso del nivel 1 es de 28 turnos, en el del nivel 2 es de 20 turnos, y en el nivel 3 es de 20 turnos. Cada vez que mostramos el *popup* de fin de turno, de acierto descubriendo una pareja, o de fallo descubriendo una pareja restamos en una unidad esta variable, y si llega a 0 en algún momento, finalizamos el juego automáticamente.

A continuación incluimos las puntuaciones en el juego. Cada jugador posee una puntuación individual, y en el caso de ambos es de 100 puntos para todos los niveles. Esta puntuación se verá incrementada si los jugadores aciertan una pareja de imágenes. En el nivel 1, se sumarán 25 puntos, mientras que en el nivel 2 y el 3 se sumarán 20 puntos al jugador que descubra la segunda imagen correctamente. Además, la puntuación de los jugadores puede aminorar. Si un jugador se equivoca al descubrir la segunda imagen y no acierta la pareja correctamente, se le restarán 20 puntos en todos los niveles (Tabla 4).

Acción	**Puntuación Jugador 1**	**Puntuación Jugador 2**
Inicio del juego	100	100
Nivel 1: Jugador 2 descubre correctamente una segunda imagen acertando la pareja	Sigue igual	+25
Nivel 2: Jugador 2 descubre correctamente una segunda imagen acertando la pareja	Sigue igual	+20
Nivel 3: Jugador 2 descubre correctamente una segunda imagen acertando la pareja	Sigue igual	+20
Jugador 2 falla al descubrir una segunda imagen	Sigue igual	-20

Tabla 4 - Puntuación de memoria

Ahora creamos la función Ajax que se ejecutará cuando un jugador finalice la partida, cuando se acaben los turnos disponibles o cuando se descubran correctamente todas las parejas. En ella colocamos los siguientes datos: los turnos restantes, si ha sido resuelto o no, las parejas restantes por descubrir, y la puntuación de los invitados. Para acabar con la función Ajax llamamos a la función que crearemos en *"views.py"* enviándole los datos que acabamos de mencionar, y cargamos el resultado, que será un nuevo *template*, en una etiqueta vacía previamente creada. De este modo finalizamos este *template "juego_colaborativo_cartas_nivel.html"*.

Después creamos en el archivo *"views.py"* las funciones que serán llamadas en las funciones Ajax desde cada uno de los *template*s que acabamos de ver. Cada una de estas tres funciones se llamará "ajax_results_cartas_*nivel*". En cada una de ellas en primer lugar rescatamos los datos que se nos envían en el POST de la función Ajax. El siguiente paso es calcular la puntuación conseguida. Si el juego se ha resuelto correctamente se suman las puntuaciones de los jugadores a la puntuación global del jugador que ha iniciado sesión, y se guarda como nuevo nivel alcanzado en el que se encuentre, siempre y cuando sea mayor al máximo nivel alcanzado que se encuentra guardado en la base de datos. Si no se resuelve correctamente la puntuación obtenida es de 0 puntos.

Finalmente en este archivo renderizamos el *template* que mostrará al usuario su puntuación en la pantalla, y si ha ganado o no.

El último paso es crear los *template*s que acabamos de mencionar, uno para cada nivel. Tendrán como nombre *"solucion-cartas-nivel.html"*. En ellos simplemente cargaremos un *popup* en el que mostramos al usuario si ha ganado o perdido, su puntuación, y el tiempo empleado, modificamos los textos que se mostraban en la pantalla, y modificamos el *footer* para que el usuario pueda volver a jugar.

Para finalizar mostraremos 5 capturas de pantalla de como ha quedado el juego tras realizar este trabajo. En primer lugar mostramos la pantalla inicial del juego (Figura 42):

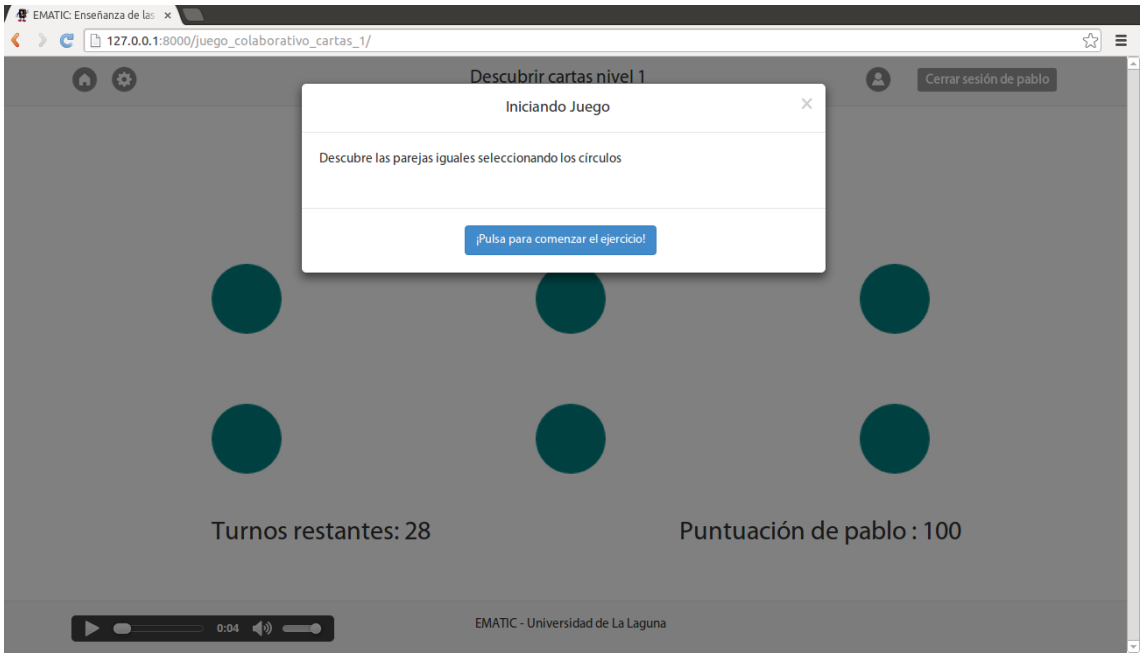


Figura 42 - Pantalla memoria 1

Mostramos a continuación la pantalla que se muestra después de que un jugador pulse un círculo azul, sin que haya otra imagen descubierta (Figura 43):

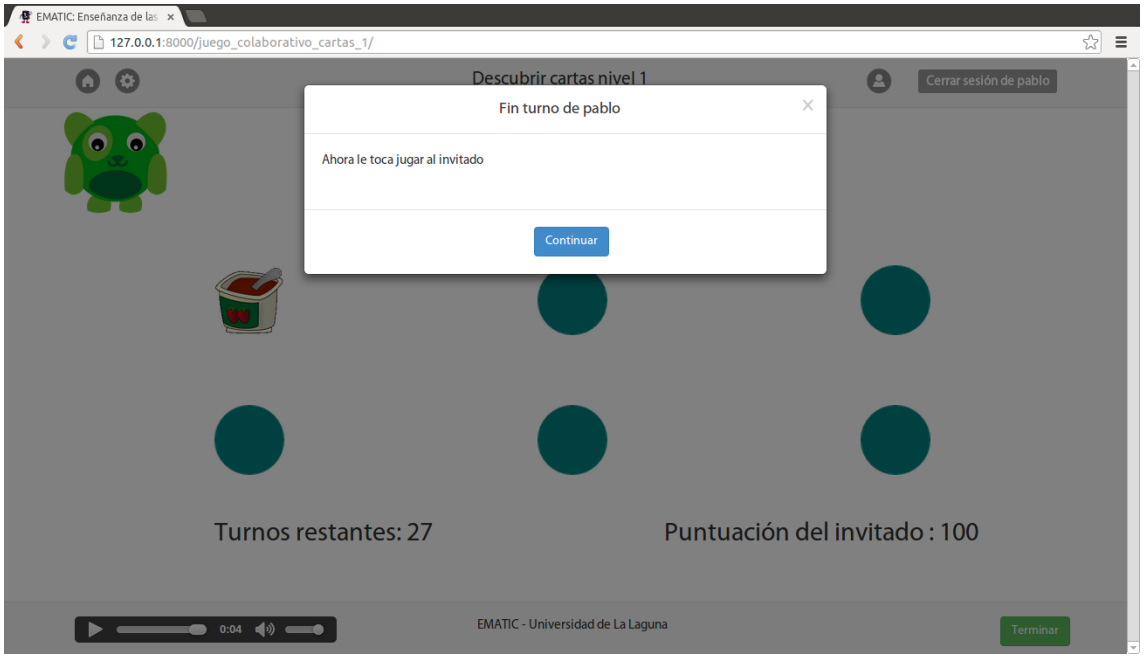


Figura 43 - Pantalla memoria 2

Mostramos ahora la pantalla que se muestra cuando un usuario descubre una pareja de imágenes correctamente (Figura 44):

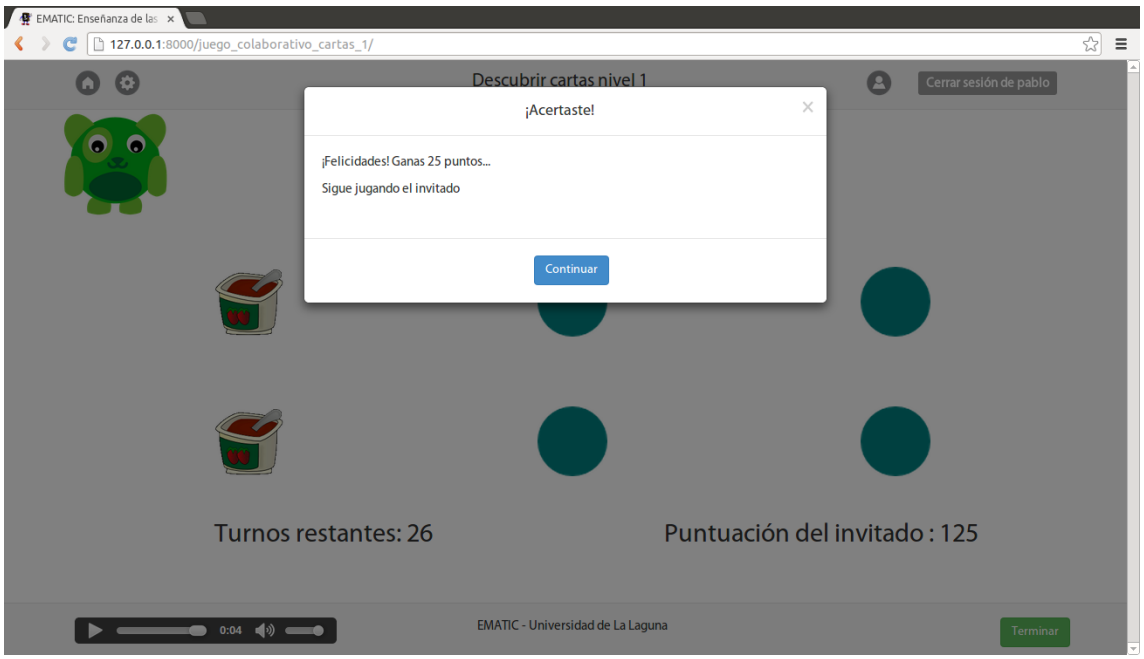


Figura 44 - Pantalla memoria 3

La siguiente pantalla muestra el *popup* que se carga cuando se finaliza una partida y se gana (Figura 45):

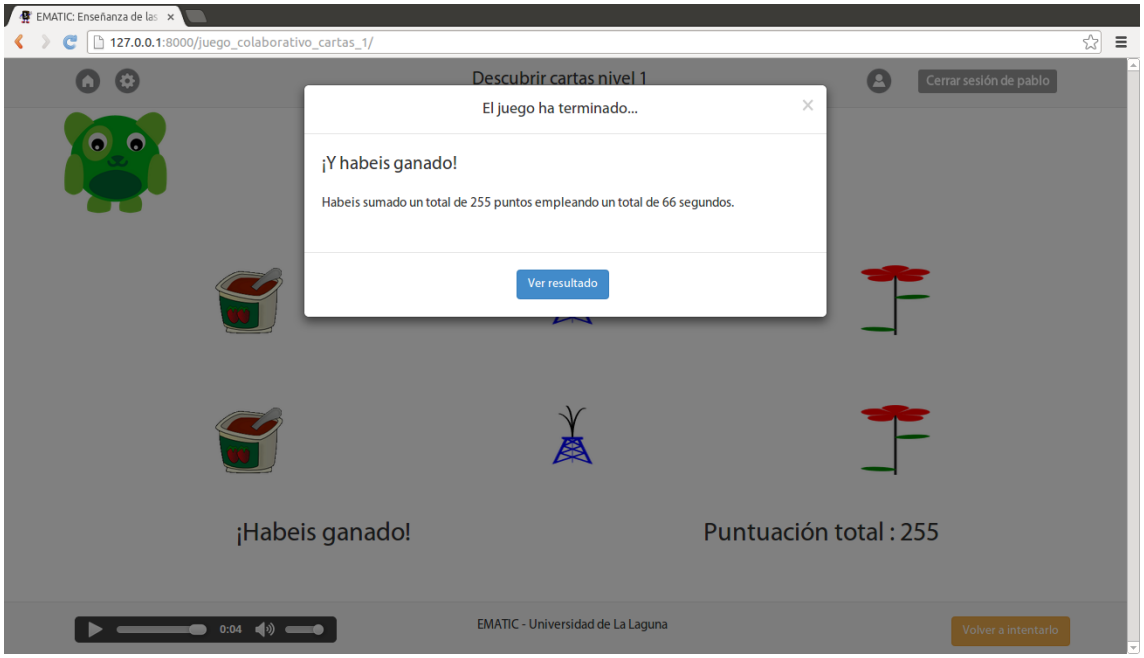


Figura 45 - Pantalla memoria 4

Por último mostramos la pantalla que aparece cuando pulsamos “Ver resultado” en la pantalla que acabamos de ver (Figura 46):

Figura 46 - Pantalla memoria 5

De este modo finalizamos el juego de memoria colaborativo.

Capítulo 8. Sistema de recompensas en actividades

Introducción

En este capítulo explicaremos el sistema de recompensas que se ha implementado en las actividades ya existentes, y que completa por tanto el trabajo de fin de grado.

Sistema de recompensas utilizado

Como último objetivo se propone añadir un sistema de puntuaciones en las actividades ya existentes, con la intención de que los usuarios de la aplicación tengan un aliciente más para completar las actividades correctamente.

Para ello hacemos uso de la clase "PuntuacionJugador" creada anteriormente en el modelo, ya que ahí almacenamos las puntuaciones que van consiguiendo los jugadores. Un jugador solo sumará puntos en una actividad si la completa correctamente. Si completa correctamente la actividad calculamos su puntuación en función del tiempo que haya empleado en superarla, si ha tardado menos de 30 segundos, se le recompensa con 300 puntos, si ha tardado entre 30 o 40 segundos se le recompensa con 200 puntos, si tarda entre 40 y 50 segundos se le recompensa con 100 puntos, y si tarda más de 50 segundo no se le recompensa con puntos.

Tras calcular la puntuación, modificamos el registro que almacena la puntuación del jugador sumándole los puntos conseguidos en la actividad que ha completado.

Estas operaciones las desarrollamos en el archivo *"views.py"*, escribiendo las instrucciones que acabamos de describir en cada una de las funciones que controlan las distintas actividades de la aplicación.

De este modo finalizamos la implementación del sistema de recompensas.

Conclusiones y trabajos futuros

Se ha actualizado el aspecto de la aplicación, utilizando el *framework Bootstrap*. Se ha elegido *Bootstrap* porque permite realizar un *"responsive design"*, es decir, lo que desarrollemos en este *framework* se podrá visualizar correctamente en dispositivos móviles, *tablets* y pc, porque al usar HTML5 y CSS3 es compatible con casi todos los navegadores y dispositivos, y porque permite un desarrollo fácil, rápido e intuitivo (Fran Amaterasu, 2014). Se ha utilizado además JQuery, pues es un *framework* de *javascript* que nos facilita el manejo del DOM, 'Modelo de Objetos del Documento" (Álvarez, 2008), facilita el manejo de funciones Ajax, y se complementa perfectamente con *Bootstrap* (Bootstrap - Javascript, 2014). Este cambio revaloriza la aplicación, pues los jugadores sentirán que la estética de la aplicación es más atractiva, lo que les motivará a jugar en ella. Además se han introducido puntuaciones a las actividades para mejorar las capacidades matemáticas de los jugadores que ya existían, y distintas categorías de jugadores, dos elementos más que motivarán a los jugadores a superarse a sí mismos, y a jugar con más ganas. Estos incentivos que hemos creado harán que el aprendizaje de los jugadores mejore, pues no debemos olvidar que el objetivo de este proyecto es que los jugadores mejoren sus capacidades y se mejore su aprendizaje.

Se han creado dos juegos colaborativos, con distintos niveles, puntuaciones y trofeos asociados a los mismos. Estos juegos aportan una nueva dimensión a la aplicación, ya que hasta ahora las actividades que existían eran individuales, y el objetivo de estos juegos es que dos jugadores colaboren para resolverlos correctamente. De este modo, no será posible que un jugador alcance el éxito sin que exista una colaboración mutua, lo cual permitirá que además de que aprendan matemáticas o mejoren su capacidad de memorización, desarrollen habilidades y capacidades de trabajo en grupo, las cuales son vitales para desarrollar correctamente una profesión en el mundo en el que vivimos.

Como líneas futuras, este trabajo ha abierto las siguientes:

- Introducir/Integrar mecánicas de gamificación en todas las actividades de EMATIC.

- Mejorar los sistemas de *"awareness"* sobre el aprendizaje y los logros obtenidos durante la ejecución del juego, con el fin de incrementar la motivación del estudiante.
- Introducir un sistema de monitorización de la colaboración que busque patrones de interacción que promuevan el aprendizaje colaborativo.

Summary and Conclusions

Has been updated appearance of the aplication, using the *Bootstrap framework*. Was chosen *Bootstrap* because it allows you make an "responsive design", that is, what we develop in this *framework* can be displayed properly on mobile devices, *tablets* and pc, because by using HTML5 and CSS3 is compatible with almost all browsers and devices and because it allows an easy, quick and intuitive development (Fran Amaterasu, 2014). JQuery is also used, as it is a *javascript framework* that facilitates our management DOM, of Ajax functions, and perfectly complements *Bootstrap* (Bootstrap - Javascript, 2014). This change revalued the aplication, as players feel that the aesthetics of the website is more attractive, which will motivate them to play in it. Ratings have also been introduced to the activities to improve the mathematical skills of the players that already existed, and various categories of players, two more that will motivate players to better themselves, and play with more energy elements. These incentives have created will make learning better players, because we must not forget that the aim of this project is that players improve their skills and improve their learning.

Have been created two collaborative games, with different levels, scores and trophies associated with them. These games bring a new dimension to the aplication, because until now there were individual activities, and the goal of these games is that two players work together to solve them properly. Thus, it is not possible for a player to achieve success without a mutual collaboration, which will allow them to learn math well or improve their memory skills, develop skills and

teamwork skills, which are vital to properly develop a profession in the world in which we live.

As future, this work has opened the following:

- Input /Integrate mechanics gamification in all activities Ematic.
- Improve systems for "awareness" about learning and achievements during the execution of the game, in order to increase student motivation.
- Establish a monitoring system that seeks collaboration patterns of interaction that promote collaborative learning.

Bibliografía

Álvarez, M. A. (2008). Qué es el DOM. Obtenido de http://www.desarrolloweb.com/articulos/que-es-el-dom.html

Bootstrap - Javascript. (2014). JavaScript. Obtenido de http://getbootstrap.com/javascript/

Bootstrap. (2014). Manual de Bootstrap. Obtenido de http://getbootstrap.com/

Collazos, C. A., Guerrero, L. A., Pino, J. A., Ochoa, S. F., & Stahl, G. (2007). Designing Collaborative Learning Environments Using. *Journal of Universal Computer Science*, 3-5.

Collazos, C., González, C. G., Gutiérrez, F., & Guerrero, L. (2014). Patterns for Monitoring and Evaluating Collaborative Learning Processes in Videogames.

Conocimiento Abierto. (2014). Qué es Twitter Bootstrap y cómo aprender a usarlo. Obtenido de http://conocimientoabierto.es/que-es-twitter-bootstrap-y-como-aprender-a-usarlo/657/

Django Project. (2011). Manual de Django 1.3.1. Obtenido de https://docs.djangoproject.com/en/1.3/

Fran Amaterasu. (2014). 5 razones por las que deberías diseñar tu landing page con bootstrap3. Obtenido de http://www.franamaterasu.com/articulo/razones-diseno-landing-page-bootstrap-3/

Garaizar, P. (2012). La ludificación en la educación. Obtenido de http://blog.catedratelefonica.deusto.es/la-ludificacion-en-la-educacion/

Hosted Appliance. (2014). Using Bootstrap Framework to Build Beautiful and Fast Front-end Web. Obtenido de http://www.hostedappliance.net/Resources/using-bootstrap-framework-to-build-beautiful-and-fast-front-end-web-93782

JQueryMobile. (2012). Manual de JQueryMobile 1.1.1. Obtenido de http://demos.jquerymobile.com/1.1.1/

KineticJS. (s.f.). Manual de KineticJS. Obtenido de http://www.html5canvastutorials.com/tutorials/html5-canvas-element/

MarianoGuerra. (2012). ORMs. Obtenido de http://python.org.ar/ORMs

Montero, S. I. (s.f.). Curso Django: Entendiendo como trabaja Django. *Maestros del Web*. Obtenido de http://www.maestrosdelweb.com/editorial/curso-django-entendiendo-como-trabaja-django/

Pavón, J. (2009). El patrón Modelo-Vista-Controlador. Obtenido de https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf

Sudoku-online.org. (2014). ¿Qué es el sudoku? Obtenido de http://www.sudoku-online.org/que-es-el-sudoku.php

w3schools. (s.f.). Ajax. Obtenido de http://www.w3schools.com/ajax/ajax_intro.asp

Zichermann, G., & Cunningham, C. (2011). Gamification by Design.