Universitat de les Illes Balears

A parameterizable web game to analyze the spread of decisions

MSc Candidate
Nicolás Galindo López

A MSc thesis submitted to *Departament de Ciències Matemàtiques i Informàtica* of the University of Balearic Islands in accordance with the requirements for the degree of
Màster Universitari Enginyeria Informàtica (MINF)

Author
Nicolás Galindo López

MSc Supervisor
Isaac Lera Castro

MINF Director
Antònia Mas Pichaco

19/07/2016

Un juego web parametrizable para analizar la propagación de decisiones

Nicolás Galindo López
Tutor: Isaac Lera Castro
Trabajo de fin de Máster Universitario en Ingeniería Informática (MINF)
Universitat de les Illes Balears
07122 Palma de Mallorca
nicolas.galindo.91@gmail.com

Resumen

El proyecto llevado a cabo en este Trabajo de Fin de Máster ha consistido en la creación de un juego multijugador sobre una plataforma web mediante el cual se posibilita el estudio de la toma de decisiones de un número variable de jugadores. Como resultado del juego queda registrado cada movimiento de los jugadores de tal manera que se pueda analizar la toma de decisiones.

Este trabajo se ha realizado en colaboración con el IFISC, donde se diseñó el juego y surgió la necesidad de una plataforma tecnológica para ponerlo en funcionamiento.

Las tareas realizadas conforman la construcción de la plataforma web sobre la que se basa el juego: creación del servidor web, implementación del algoritmo para interconectar a los jugadores, dinamización del cliente web con el que los jugadores interactuarán, cálculo de los resultados finales y generación de los ficheros anteriormente mencionados.

Abstract

The project carried out in this Master's Thesis consisted of the creation of a multiplayer game on a web platform through which the study of the spread of decisions of a variable number of players will be made possible.

This work was done in collaboration with IFISC, where the game was designed and the need for a technological platform to operate it arose.

The tasks carried out are the needed ones for the construction of the game: creation of the web server, implementation of the algorithm to link the players, development of the web client with which players will interact, calculation of the final results and generation of the aforementioned files.

Palabras clave: toma de decisiones, juego web, experimento, node.js, socket.io, JavaScript

1. Introducción

El estudio de la toma de decisiones y su dispersión entre los miembros de una comunidad supone un reto inabarcable si se intenta acometer en el medio natural de los individuos de dicha comunidad, sea cual sea ésta. Por tanto, la generación de un entorno de menor complejidad donde todas las variables sean conocidas se muestra como una solución viable para acometer este objetivo.

La motivación que impulsó la creación de este proyecto partió de la intención de adaptar a humanos experimentos realizados anteriormente con animales tales como peces o monos. Dichos experimentos consistían en un sistema cerrado en el que los individuos podían elegir entre dos alternativas donde una ya había sido elegida por otros y la otra permanecía casi inédita. En los experimentos con animales se observaba que la forma de tomar la decisión era bayesiana[1]: la alternativa que hubiera sido la elegida por más individuos sería aquella por la que con mayor probabilidad se decantarían los siguientes.

En el IFISC decidieron modelizar la toma de decisiones en humanos mediante experimentos simples y controlados inspirados en los anteriormente mencionados. Construyeron una primera plataforma tecnológica basada en Flash para llevar a cabo una ejecución del experimento [1], pero su funcionamiento no fue el deseado: contenía errores y no soportaba grandes volúmenes de usuarios. Por este motivo se ha llevado a cabo este proyecto.

Entrando en profundidad en este experimento, el entorno es un juego web y la única variable es una palabra –una secuencia de dígitos, en realidad– aleatoria de longitud variable, que es la que los jugadores deberán tratar de deducir colectivamente.

[1] El término *bayesiano* proviene del teorema de Bayes, que expresa la probabilidad de que un evento tenga lugar condicionada por otro evento.

Esta palabra le será mostrada parcialmente a cada jugador y posteriormente ocultada. El jugador deberá retener en su memoria la parte de la palabra que habrá visto puesto que ésta, en combinación con las palabras de los demás jugadores, serán su vía para alcanzar la resolución de la palabra completa. Además, el jugador deberá tener en cuenta que las palabras de los demás jugadores pueden ser parcial o totalmente incorrectas, puesto que cada uno de ellos, a su vez, habrá tenido acceso a un fragmento –distinto del que ve el primer jugador– de la palabra objetivo, y el resto de la palabra que mostrará será resultado de sus propias cábalas. Confiar en ellas o no es otro de los desafíos a los que se enfrentará cada jugador.

Por otra parte, a modo de dificultad añadida y con el objetivo de poner a prueba la memoria visual de los jugadores, en lugar de ver los dígitos de la palabra verán una hilera de colores. Cada uno de estos colores se corresponderá con uno de los dígitos, y variarán en cada partida.

Cada jugador se encontrará dentro de una sala en la que no verá al total del resto de los jugadores. Además, los jugadores no necesariamente se verán mutuamente. Por tanto, existirá una sala única para cada jugador.

Las conexiones entre los jugadores se generarán en forma de **red de mundo pequeño** o ***small-world network***. Una red de mundo pequeño es un tipo de grafo en el que la mayoría de los nodos no son vecinos entre sí –geográficamente sí pueden serlo, pero no necesariamente están interconectados–, pero son alcanzables desde cualquier otro nodo origen mediante un número pequeño de saltos. Formalmente, en una red de este tipo la distancia típica L entre dos nodos elegidos aleatoriamente crece proporcionalmente al logaritmo del número de nodos N presentes en la red, es decir, $L \propto \log N$. Esta estructura se muestra en numerosos ejemplos del mundo real, entre ellos, las redes sociales[2].

Como resultado del juego se generarán unos ficheros en los que quedará registrado cada movimiento que los jugadores realicen, de tal manera que se pueda estudiar su toma de decisiones y obtener patrones de conducta.

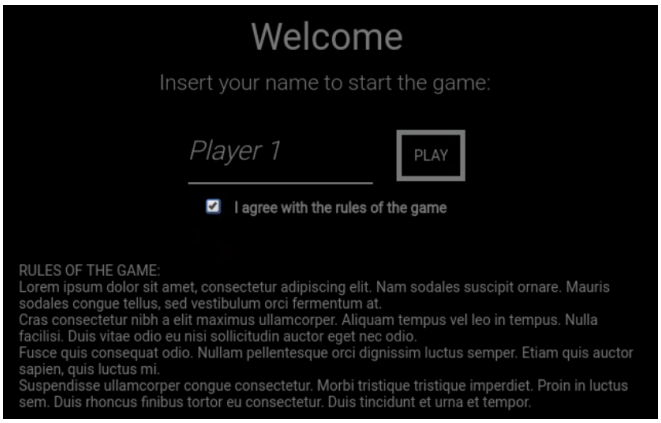
A partir de todos los detalles ya mencionados, se puede describir el experimento completo como una secuencia de fases, que son las siguientes:

1. Entrada de jugadores: se arranca el servidor y éste admite la entrada de jugadores hasta un parámetro fijado.
2. Inicio del experimento: una vez alcanzado el número de jugadores necesario, el experimento comienza. Su desarrollo está basado en un proceso iterativo que consta de varias partidas secuenciales, cada una de ellas con la siguiente estructura:
   a) Creación de la partida: se generan la palabra objetivo y las conexiones entre jugadores o salas. Se determina qué fragmento de la palabra objetivo verá cada jugador y cuáles serán los colores que representarán a cada dígito. Esta fase debe tener lugar durante un periodo de tiempo prácticamente imperceptible para los jugadores.
   b) Preparación de los jugadores: se le muestra a cada jugador durante un periodo de tiempo fijado en el que no puede realizar ninguna acción la parte de la palabra objetivo a la que tiene acceso.
   c) Inicio de la partida: se oculta la totalidad de la palabra al jugador. Desde este momento, debe tratar de deducir la palabra objetivo. Cuando envíe su primera propuesta, verá las palabras de los demás jugadores, que se actualizarán con cada cambio que ellos efectúen. El jugador puede enviar tantas propuestas como desee a lo largo de la partida –que tendrá una duración fijada–, y en ningún momento sabrá cuál es su grado de acierto.
   d) Fin de la partida: se efectúa el cálculo de las puntuaciones de cada jugador en base a la última propuesta que hayan enviado y se les muestra una tabla de resultados.

2. Descripción del juego

Tal como se ha explicado en la introducción, el experimento consta de una serie de fases. En esta sección se detallan las pantallas que verán los jugadores.

En primer lugar, cuando los jugadores acceden a la plataforma del experimento se les recibe con una pantalla de bienvenida en la que deberán introducir su nombre:

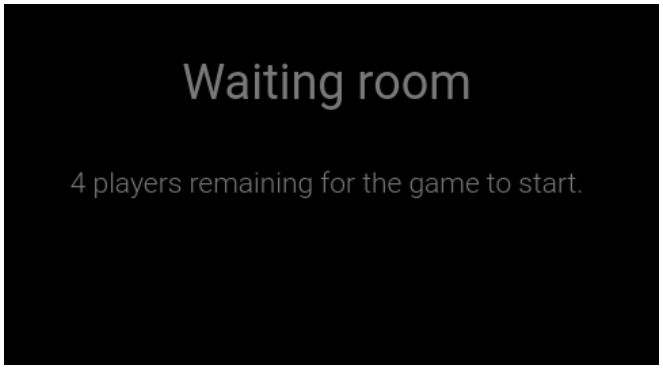


Además, deberán confirmar que aceptan las reglas del juego, así como las políticas de protección y cesión de datos, antes de poder acceder a él.

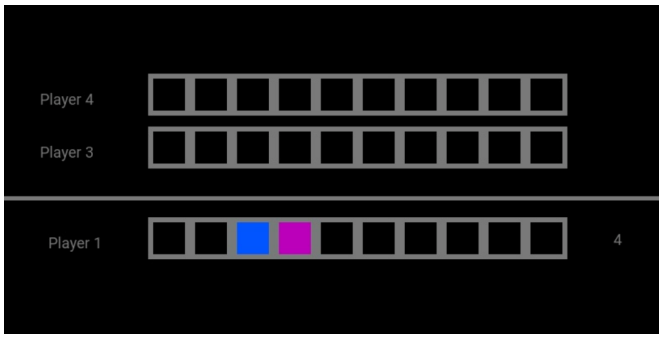
Una vez introducido su nombre, los jugadores pasarán a la

[2]Otro ejemplo muy conocido de red de mundo pequeño es la teoría de los **seis grados de separación**, propuesta por Frigyes Karinthy, que enuncia que toda persona en el mundo puede estar conectada a cualquier otra a través de una cadena de conocidos que no tiene más de cinco intermediarios.

sala de espera, donde se les indicará el recuento de jugadores que faltan para poder iniciar el experimento:

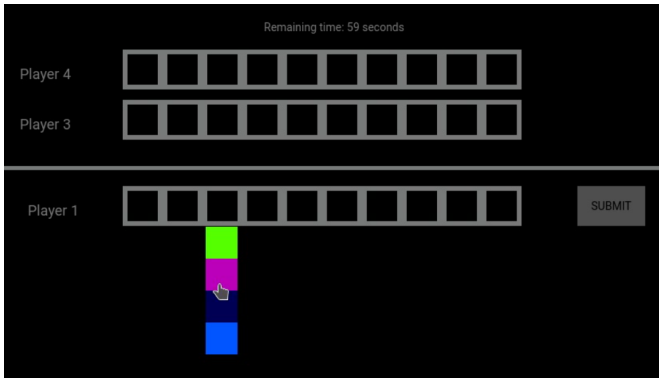


El siguiente paso es automático. Una vez alcanzado el número de jugadores necesario, el experimento comienza y se le muestra al jugador su palabra inicial:

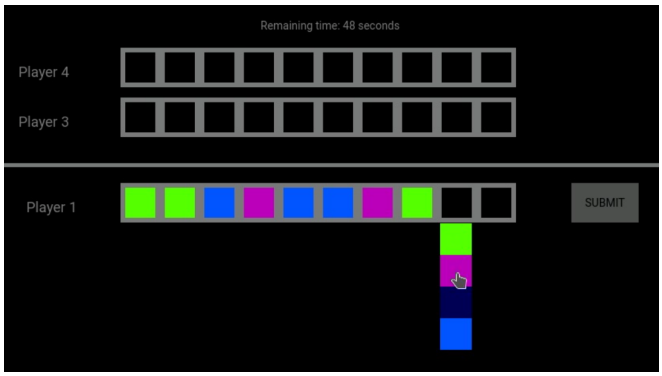


A la derecha de la palabra aparece una cuenta atrás indicándole cuántos segundos faltan para que la palabra se le oculte. En este ejemplo, el jugador ve la tercera y cuarta letras de la palabra, y además sabe que en su sala hay otros dos jugadores, el 3 y el 4.

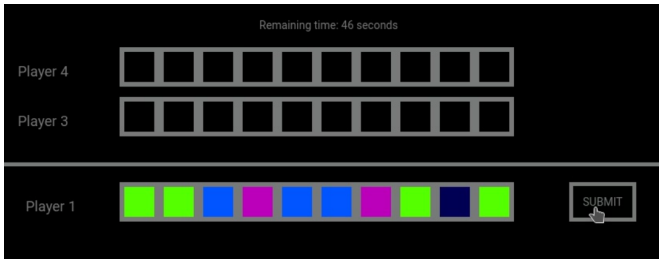
Una vez finalizada la cuenta atrás, se le oculta al jugador su palabra inicial y ya se le permite actuar. Cuando pulse sobre una casilla se le desplegarán las opciones de colores para que pueda asignar una valor a dicha casilla:



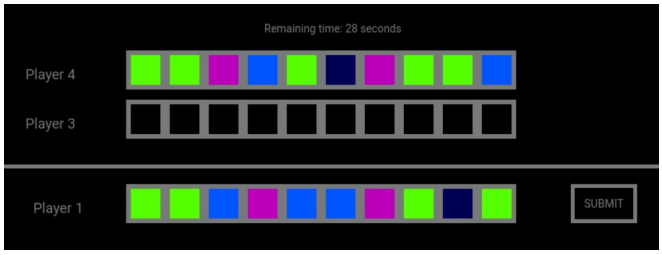
Deberá completar este proceso para cada casilla antes de poder enviar su propuesta:



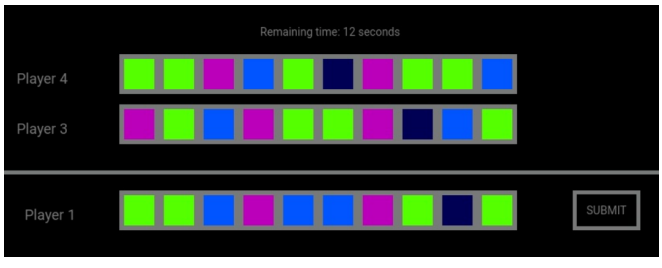
Una vez haya creado una palabra completa, el botón se mostrará habilitado y el jugador ya podrá enviar su propuesta:



A continuación, se le mostrarán las palabras de los demás jugadores de su sala que ya hayan enviado una propuesta:



Conforme los demás jugadores vayan enviando sus propuestas iniciales o las modifiquen, la información se le actualizará al jugador:



Esta información será en la que se podrá basar el jugador para modificar su propia propuesta.

Finalmente, al término de la partida se les mostrarán a los jugadores sus puntuaciones:

Las puntuaciones se dividen en dos columnas, una para la puntuación de la ronda que acaba de terminar y otra para las puntuaciones acumuladas a lo largo del experimento.

Cada vez que un jugador ejecute alguna acción a lo largo de las fases mencionadas en este apartado, ésta quedará registrada en el instante de tiempo en el que se produzca, permitiendo así el posterior análisis de toma de decisiones.

Además, la configuración de cada uno de los juegos que conforman el experimento es parametrizable: pueden modificarse su duración, el número de opciones distintas, la longitud de la palabra y el tiempo durante el cual se les mostrará inicialmente su palabra a los jugadores antes de que se oculte. También son configurables los parámetros del algoritmo que genera la red de mundo pequeño, por lo que ésta puede tener en cada juego un grado de regularidad distinto.

3. Implementación

La implementación de este juego web ha constado de dos partes claramente diferenciadas: la capa de servidor y la de cliente.

3.1. La capa de servidor

El servidor web, encargado tanto de la creación de los experimentos como de la recopilación de los resultados, ha sido implementado en **node.js**[3]. El motivo de la elección de un *framework* para JavaScript por delante de otros lenguajes más habituales para esta tarea –como puedan ser Django sobre Python o Laravel sobre PHP– se debe a la arquitectura del sistema, explicada a continuación.

Tras un estudio preliminar del producto a generar, se comprobó que éste tendría las siguientes características:

- Sería prioritaria la efectividad y facilidad de intercambio de datos constante entre usuarios y servidor en tiempo real.
- El servidor generaría salas en las que admitiría la entrada de usuarios. La asignación de los usuarios a cada sala sería dinámica a lo largo de la duración del experimento.
- El sistema carecería de base de datos.

Estas características, en conjunto, recordaron inmediatamente a un sistema ya conocido: las salas de chat. Además, se decidió que resultaba conveniente llevar a cabo una implementación basada en *WebSockets*. Por tanto, la herramienta elegida –debido a su total cumplimiento de todos los requisitos– fue **socket.io**[4] y, en consecuencia, la implementación natural del servidor se realizaría mediante node.js.

La necesidad del uso de *WebSockets*

WebSocket es un protocolo que proporciona comunicación full-duplex[5] sobre una misma conexión TCP. En otras palabras, genera una vía entre servidor y cliente mediante la cual ambos pueden transmitir y recibir.

Este sistema, para el caso de la aplicación a desarrollar en este Trabajo de Fin de Máster, supone una ventaja sobre su alternativa clásica: AJAX. La causa de dicha ventaja recae sobre la bidireccionalidad de los *WebSockets*, puesto que permiten que el servidor envíe mensajes al cliente sin que éste los solicite. Esto no es posible usando AJAX, por lo que esta técnica obligaría a implementar una estrategia basada en *polling*: el cliente debería solicitar continuamente al servidor los eventuales nuevos datos que se hubieran podido generar.

El uso de *WebSockets* también supone una gran ventaja sobre la tecnología en la que fue implementada la primera versión del experimento, Adobe Flash. Esto es debido no solamente al hecho de que los navegadores de muchos dispositivos –principalmente dispositivos portátiles– ya no soportan esta tecnología, sino también a que, incluso los que sí lo hacen, no tienen soporte nativo para la implementación de sockets realizada en Flash, mientras que la mayoría sí que soportan *WebSockets*.

3.1.1. Fases de la implementación

El proceso de implementación de la capa de servidor se ha llevado a cabo por fases, cada una de ellas correspondientes a las tareas a desarrollar. En este apartado se detalla cada una de estas fases.

La creación del servidor

Una vez decidido que el servidor estaría desarrollado sobre node.js, los pasos a seguir eran claros: obtener su documentación y seguir las directrices especificadas en ella. Node.js

[3] `https://nodejs.org/en/`

[4] `http://socket.io/`

[5] Sistema de comunicación punto a punto en el que se permite la comunicación en dos direcciones simultaneamente.

es un *framework* con una curva de aprendizaje muy inclinada, esto es, su aprendizaje es rápido. Además, existe una gran cantidad de plantillas para el arranque de un proyecto, por lo que resulta casi inmediato lograr la construcción de un servidor web funcional. Sin ir más lejos, en la propia página web de socket.io pueden conseguirse esqueletos de proyectos sobre node.js orientados al uso de socket.io.

Así, tras una búsqueda de la mejor plantilla para arrancar un proyecto y algunas tareas de configuración mínimas, la base del código del servidor estaba lista para ser usada.

La implementación del algoritmo

Como ya se ha explicado, las conexiones entre los jugadores tendrían la forma de una red de mundo pequeño. Este tipo de redes conforman un grafo en el que la mayoría de nodos no son vecinos entre sí y, sin embargo, casi todos pueden ser alcanzados desde cualquier nodo origen a través de un número relativamente corto de saltos. Existen múltiples modelos orientados a la construcción de grafos de este tipo, y el empleado en este caso –por especificación explícita de los usuarios finales del sistema– es el de Watts-Strogatz.

Antes de tratar de emprender una implementación manual de este algoritmo en JavaScript, se llevó a cabo una búsqueda de aproximaciones ya existentes y probadas que pudieran suponer una solución fiable. Así se encontró, entre otras, la librería **randomgraph**[6]. Una primera revisión del código de su función `randomgraph.WattsStrogatz.beta()` permitió comprobar que aparentemente se ajustaba al algoritmo, y las pruebas posteriores lo confirmaron.

En la Figura 1, muestra de dichas pruebas, se observa que ningún nodo carece de conexiones, por lo que queda garantizado que cualquiera de ellos es accesible desde cualquier otro, ya sea de forma directa o mediante un número bajo de saltos. Además, como dato relevante, queda latente otra de las características ya mencionadas del modelo de Watts-Strogatz: los nodos no necesariamente están conectados mutuamente. En este ejemplo, la conexión es bidireccional únicamente entre los nodos `2` y `5`.

[6] `https://github.com/gka/randomgraph.js/tree/master`

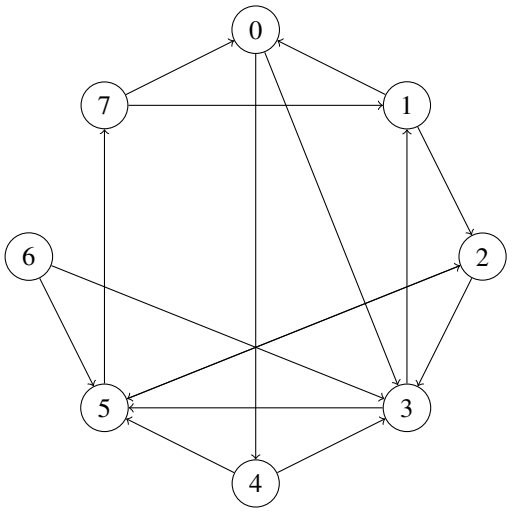


Figura 1: Red generada por la librería randomgraph para 8 nodos

El siguiente paso antes de optar por esta solución consistía en comprobar su rendimiento, puesto que se encargaría de uno de los elementos clave del juego y no serían tolerables tiempos de respuesta altos –esto es, del orden de segundos– que mantuvieran a los jugadores en espera. Verificar si este aspecto cumplía con los requisitos fue sencillo: simplemente fue necesario ejecutar el algoritmo para un número variable de jugadores y comprobar los tiempos de ejecución. Los resultados obtenidos pueden observarse en las Figuras 2 y 3, que demuestran que fueron satisfactorios. El crecimiento de los tiempos de ejecución es aproximadamente lineal y, en cualquier caso, éstos nunca alcanzan siquiera los 500 milisegundos para un número de jugadores extremadamente alto. Por su parte, el crecimiento del uso de memoria presenta una pendiente más pronunciada, pero sus valores son despreciables.

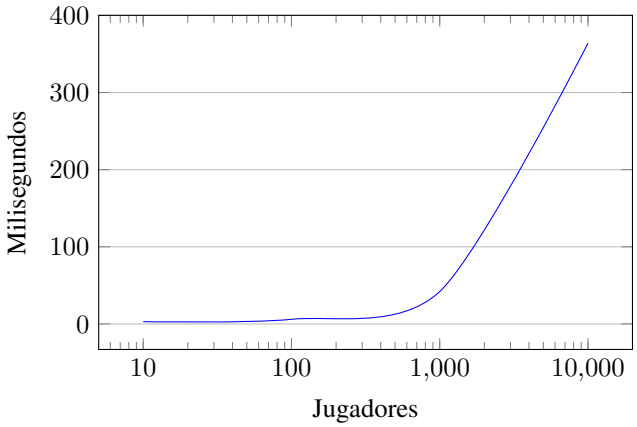


Figura 2: Tiempos necesarios para la generación de las conexiones entre los jugadores con una CPU Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz

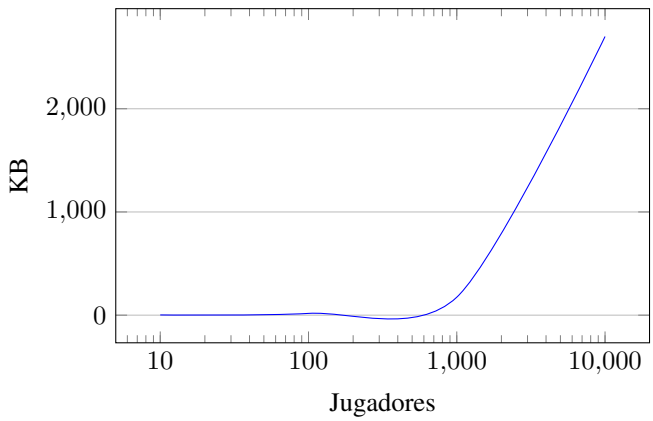


Figura 3: Usos de memoria durante la generación de las conexiones entre los jugadores

Por tanto, tras comprobar que esta implementación del algoritmo cumplía los requisitos necesarios, fue la que se usó para distribuir a los jugadores en las salas.

El cálculo de puntuaciones

Tras cada ronda, el sistema lleva a cabo un cálculo de los valores finales en los que comprueba cuál es el grado de acierto de cada jugador.

La forma de llevar a cabo el recuento de los aciertos no es compleja: consiste en comprobar cuáles de las letras incluidas en la última palabra que ha enviado cada jugador forman parte de la palabra real y otorgar a éste una cantidad parametrizable de puntos por cada acierto. En el caso de que el jugador haya acertado la totalidad de la palabra, recibirá además una bonificación que se añadirá a su puntación en la ronda.

Las puntuaciones obtenidas por los jugadores en cada ronda se acumulan durante todo el experimento, por lo que al término de cada ronda ven tanto la puntuación obtenida en dicha ronda como el total acumulado.

En el supuesto de que un jugador abandone el juego, su puntuación seguirá apareciendo a los demás jugadores durante todo el experimento.

La generación de los ficheros con los resultados

La auténtica utilidad de este juego recae en los ficheros generados (*logs*), en los que queda registrada toda la actividad que tiene lugar durante cada experimento. A estos *logs*, generados en formato CSV, se les añade el contenido en varias tareas diferenciadas.

La primera de estas tareas tiene lugar inmediatamente antes del inicio del juego y, a su vez, se subdivide en otras dos partes.

- Por una parte, se registran datos tales como la longitud de la palabra, el número de jugadores necesarios para el experimento, los parámetros empleados para el algoritmo de Watts-Strogatz, la duración de la partida... es decir, los datos referentes a la configuración del juego.
- Por otra parte, se registran los datos referentes a la partida y los jugadores: la palabra con la que se jugará, las palabras iniciales que ve cada jugador –partes de la palabra que le quedarán visibles u ocultas– y las conexiones entre todos ellos.

La segunda parte se produce a lo largo del juego, y consiste en registrar secuencialmente cada nueva palabra que un jugador envía al servidor. Es con este dato con el que puede comprobarse cómo evoluciona la partida y en qué movimientos de un jugador se basa otro para llevar a cabo los suyos propios. Por tanto, este registro es el producto clave derivado del juego.

Finalmente, se registran también los abandonos que puedan tener lugar durante el juego, ya que sin este dato resultaría complicado saber si un jugador no realiza movimientos porque está a la espera de nuevas ideas o porque ha dejado de participar en el juego.

3.2. La capa de cliente

El cliente web –comúnmente el navegador– es la interfaz mediante la cual los usuarios interactúan con la aplicación. En el caso de esta aplicación, se encarga de mostrar las palabras iniciales a los jugadores, capturar sus interacciones para enviar sus propuestas al servidor y mostrarles los resultados.

La dinamización del cliente se ha llevado a cabo mediante HTML, CSS y, de nuevo, JavaScript, aunque en este caso, al tratarse del lenguaje natural para los navegadores web, su uso es más común que en la capa de servidor. La única librería para JavaScript empleada ha sido jQuery[7], que simplifica las interacciones con el DOM[8]. Por su parte, para HTML y CSS se ha empleado Bootstrap[9], que aglutina la mayoría de estilos útiles para generar la base de una página web.

La estructura seguida para implementar toda la capa de cliente ha consistido en generar una clase para cada vista –pantalla de bienvenida, pantalla de juego, pantalla de puntuaciones, etc.– siguiendo el paradigma de programación orientada a objetos de la manera que JavaScript permite este paradigma. Así, cada clase tiene la siguiente estructura:

[7] `https://jquery.com/`

[8] Document Object Model: representación estructural de los documentos HTML y XML que permite la modificación de su contenido o su presentación visual.

[9] `http://getbootstrap.com/`

```
var Clase = function() {
    var privada;  // Declaración de variables privadas

    /**
     * Declaración de las propiedades de la clase que
     * deben ser públicas
     */
    return {
        /**
         * Constructor
         */
        init: function() {
            privada = ...;
        },
        // Referencia a la función definida más abajo
        // para que sea pública
        metodoPublico: metodoInterno
    };

    function metodoInterno() {
        ...
    }

    /**
     * Esta función nunca será accesible desde el exterior
     * porque no está referenciada arriba
     */
    function metodoPrivado() {
        ...
    }
};
```

Una vez creada la clase, la instanciación se produce invocando a su constructor:

```
var clase = new Clase();
clase.init();
```

o, sencillamente,

```
new Clase().init();
```

A partir de esta estructura de clases, la responsabilidad de la inicialización de toda la aplicación en el cliente recae sobre una clase principal –llamada `Main()`– que se inicializa automáticamente y se encarga de instanciar a las demás. Esta inicialización automática tiene lugar cuando se dispara el evento `ready` del DOM, tal como se muestra en el siguiente fragmento de código:

```
$(document).ready(function() {
    new Main().init();
});
```

Con respecto al uso de CSS, se ha empleado un preprocesador, SCSS[10], que permite la creación de variables y el anidamiento de reglas. El código SCSS debe compilarse para producir el código CSS útil para el navegador. Así, por ejemplo, el siguiente fragmento de SCSS:

```
.clase-principal {
  color: red;
  &.seleccionada {
    font-size: 18px;
  }
  .clase-interna {
    color: black;
  }
}
```

produce tras su compilación el siguiente fragmento de CSS:

```
.clase-principal {
    color: red;
}
.clase-principal.seleccionada {
    font-size: 18px;
}
.clase-principal .clase-interna {
    color: black;
}
```

Puede apreciarse en el ejemplo cómo se anidan las reglas y cómo el uso del ampersand (`&`) permite concatenar clases.

Finalmente, referente a la estructura del código HTML, el código en su totalidad está contenido en un único fichero, puesto que el contenido mostrado se selecciona dinámicamente según la pantalla que deban ver los jugadores, pero sin realizar redirecciones. Así, el esqueleto del código tiene la siguiente estructura:

```
<body>
    <!-- Loading screen -->
    <div id="loading-screen" class="screen">
        ...
    </div>
    <!-- Welcome screen -->
    <div id="welcome-screen" class="screen hidden">
        ...
    </div>
    ...
</body>
```

Cada pantalla posee la clase `screen`, que concentra los estilos comunes, y un identificador propio, que es el que permite mostrarlas y ocultarlas una por una.

Por otra parte, hay también un apartado de plantillas que contienen el código HTML necesario para mostrar las palabras de los jugadores. Por ejemplo, para la palabra del jugador actual:

[10] `http://sass-lang.com/`

```
<div id="board-template">
    <div class="col-xs-12 own-game row">
        <div class="col-xs-2 name-box">
            <span class="name"></span>
        </div>
        <div class="col-xs-8">
            <div class="game-board">
                <div class="dropdown box">
                    <button class="game-button
                            dropdown-toggle disabled"
                            data-position=""
                            data-toggle="dropdown"
                            data-background=""
                            type="button">
                    </button>
                    ...
                </div>
            </div>
        </div>
    </div>
</div>
```

Estas plantillas son las que se completan mediante JavaScript con los valores recibidos desde el servidor (longitud de la palabra, colores, número de jugadores que verá cada jugador, etc.) para poder mostrar las pantallas de cada juego. Puede observarse que los atributos `data-*` están vacíos, ya que su contenido se asigna dinámicamente.

4. Resultados y conclusiones

Una vez concluída la implementación del juego, se procedió a su instalación en un servidor para que pudiera ser probado por los usuarios finales del IFISC. Su valoración fue muy positiva; consideraron que todos los requisitos habían sido implementados y que los tiempos de ejecución eran idóneos para la puesta en funcionamiento del experimento. Por tanto, podrá lanzarse este experimento con la nueva plataforma tecnológica.

En el apartado técnico, la implementación en node.js era algo novedoso para mí, por lo que conllevó una búsqueda inicial de información y una pequeña adaptación al *framework*. Aun así, superada esta etapa inicial, el resto del trabajo no resultó dificultoso y, por consiguiente, el desarrollo tuvo lugar con normalidad y siguiendo los plazos previstos. Sin embargo, no se han podido llevar a cabo pruebas con un gran volumen de usuarios –cerca del centenar, en un caso ideal– tanto para poder comprobar el rendimiento real como para ejecutar una depuración exhaustiva del código. Todas las pruebas realizadas han contado con un máximo de cinco usuarios.

A modo de trabajo futuro, se pretende modificar la aplicación para adaptarla a un tablero bidimensional –esto es, mostrar una matriz en lugar de una palabra– que simule una superficie rugosa, de tal manera que los jugadores deban encontrar las coordenadas con mayor puntuación a partir de las puntuaciones logradas por los demás jugadores. Las generación de relaciones entre usuarios se mantendrá, pero la interfaz será distinta.

Finalmente, cabe mencionar que las tareas llevadas a cabo en este Trabajo de Fin de Máster han estado relacionadas con las asignaturas 11568 - Ingeniería Web y 11569 - TIC para la Educación.

Referencias

[1] Toni Pérez, Jordi Zamora, and Victor M. Eguiluz. Collective intelligence: aggregation of information from neighbors in a guessing game. Technical report, 2016.