DOCUMENTACIÓN IS

Proyecto PasaPalabra

Mikel Barcina
Jose Ángel Gumiel

Índice

Introducción

A lo largo de las siguientes páginas vamos a presentar nuestro proyecto. Explicaremos en qué consiste, cómo se desarrolla internamente, qué metodología hemos seguido para organizarnos y trabajar en grupo, qué conocimientos previos hemos necesitado y qué conclusiones hemos obtenido.

Además expondremos otros aspectos que nos parecen interesantes de recalcar.

Descripción del proyecto

Se trata de un juego programado en lenguaje Java. Está basado en la última prueba del programa de televisión "PasaPalabra". El jugador tendrá un tiempo predefinido en segundos y tendrá que responder las preguntas que se le formulen. En total hay unas 27 preguntas, una por cada letra de nuestro alfabeto. El usuario tiene la opción de responder o pasar a la siguiente pregunta. Una vez se haya completado la primera vuelta, el usuario tendrá que responder a las preguntas que no haya contestado.

Las respuestas pueden ser acertadas o falladas. Por cada acierto se suman puntos, pero los fallos penalizan y descuentan puntos. La puntuación no podrá ser nunca negativa, en el caso de que esto ocurra la puntuación final será cero.

Motivaciones

Es un proyecto que permite agrupar un conjunto de aspectos ya vistos en otras asignaturas de la carrera. Se aplican los conocimientos de diseño de diagramas de clases y de secuencias vistos en PMOO y en IS. También permite la reflexión sobre qué tipo de algoritmo pudiera ser más eficiente y apto para las listas, lo cual aprendimos en la asignatura EDA. Además, uno de los requisitos es también el uso de una interfaz gráfica, con lo que también tenemos que aplicar conocimientos adquiridos en IS.

En resumen, es un proyecto que requiere de conocer un poco de todo lo estudiado durante nuestra formación, con lo cual es apto para nuestro aprendizaje.

Aspectos Técnicos

Para el correcto desarrollo de este juego lo primero que hemos tenido que hacer son los diagramas de clases y de secuencia. Desde el primero que hicimos hasta el último que hemos utilizado para elaborar nuestro juego hay bastantes diferencias. Hemos tenido que hacer unos cuantos cambios y modificaciones hasta dar con el diseño correcto. Al final de este documento se incluirán en un anexo los diagramas que hemos utilizado y seguido.

A la hora de hacer las listas hemos intentado coger lo que más se adecúe a este juego. Para el diccionario queremos velocidad, es por ello que como debe albergar un conjunto de palabras y definiciones significativo, la mejor forma de hacerlo es a través de un HashMap.

En el rosco, observamos que es una lista que cuando llega al final, vuelve al principio, es circular. Por ello hemos decidido que el modelo que mejor se adapta es una lista ligada doble. Además

tiene la ventaja que sabemos la casilla en la que estamos, la siguiente y la anterior. Cuando una respuesta es contestada podemos anotar en ese mismo momento si es correcta o no. Al pasar a la siguiente letra podemos saber también si ha sido contestada o no, y en caso de estar contestada se puede eliminar de esa lista para que no se vuelva a repetir. Es un algoritmo muy versátil.

En cuanto a la interfaz, lo más primordial es que permita una interacción básica con el usuario y que sea fácil de entender. Como aspectos opcionales está la estética y los adornos que le queramos añadir. Cabe resaltar que para crear el rosco de forma circular, hemos tenido que recurrir a una fórmula matemática que explicaremos más adelante, cuando detallemos todos los métodos del programa en profundidad.

Objetivos

- ✓ Diseño de un diagrama de clases adecuado a la aplicación.
- ✓ Diseño de un diagrama de secuencia que detalle el proceso de la partida.
- ✓ Decidir qué estructuras de datos son las más adecuadas al proyecto.
- ✓ Implementación de los métodos.
- ✓ Pruebas para comprobar el correcto funcionamiento de la implementación.
- ✓ Diseño de la interfaz gráfica.

Diseño

Este apartado se dividirá en las clases que tiene el proyecto, explicaremos en profundidad la funcionalidad de cada clase y qué es lo que hacen sus métodos.

Interfaz

Empezaremos detallando las clases correspondientes a la interfaz, que son las siguientes:

Principal

Es la primera pantalla de nuestro juego. En ella se pueden seleccionar diferentes opciones. La primera es “Jugar”, es un botón que nos permitirá iniciar la partida.

En segundo lugar se encuentra “Puntuaciones”. Pulsando este botón se mostrarán por pantalla las puntuaciones obtenidas en partidas anteriores.

La última opción es “Reglas”. Muestra por pantalla la dinámica del juego y la información que el usuario necesita conocer para desarrollar correctamente su partida.

Como extra tenemos un botón de selección que permite decidir si queremos que nuestra partida contenga sonidos, o si por el contrario, preferimos desactivarlos.

Reglas

Se trata de una ventana del tipo JDialog que muestra por pantalla las normas del juego.

La cabecera de arriba del todo es una JLabel con el texto “Reglas”.

El texto correspondiente a las reglas se encuentra también en una JLabel llamada “lblReglas”. Seteamos el texto para que se muestre en la ventana.

Esta pantalla tiene dos botones, “Volver” y “Salir”. El primero de ellos nos lleva a la pantalla anterior, es decir, el menú principal. El segundo, detiene la ejecución de la aplicación. Para que los botones funcionen y cumplan su objetivo tienen que llevar implementados los ActionListeners.

InterfazPuntuaciones

Muy similar al anterior. Se encarga de mostrar por pantalla las puntuaciones obtenidas. Coge la información del archivo xml, donde están las 10 mejores puntuaciones. Al igual que en la anterior, también nos encontramos con 2 botones, “Volver” y “Salir”.

InterfazRosco2

El nombre de esta interfaz es debido a que hemos tenido otras interfaces anteriores que no resultaron, y finalmente nos quedamos con esta sin cambiar el nombre. Esta es una de las pantallas más importantes, ya que es la que más interacción con el usuario va a tener.

Esta ventana es de tipo JFrame. Ha sido dividida en varias Layouts. Las layouts utilizadas han sido en su mayoría Group Layouts.

En esta pantalla vemos por un lado el rosco. Lo hemos hecho con JLabels. En primer lugar hemos creado 27 JLabels por cada color, es decir, el color azul representa pregunta sin responder, entonces hay 27 JLabels de color azul que tiene configurado el apartado “icon” con una imagen que se le corresponde, por ejemplo “botón_azul_mini_A.png”. Lo mismo con el color amarillo, que representa pregunta actual, la que está respondiendo el usuario, color verde, pregunta acertada, y color rojo, pregunta fallada. En resumen, 108 JLabels con su imagen correspondiente.

Las imágenes, para que no haya ningún problema respecto a rutas o ubicaciones, las hemos almacenado en un paquete llamado “packIconos”.

Dentro del rosco también tenemos un JLabel que muestra el tiempo del que disponemos.

Una parte interesante del rosco es la forma que hemos utilizado para que nos quede perfectamente redondo. Para ello hemos tenido que recurrir a las matemáticas y a la trigonometría. Para ello creamos un método al que llamamos “ordenar”. Ahí definimos theta como un float, los intefers x e y que usaremos para situar el objeto en pantalla, sería unareferencia en píxeles, y también hemos necesitado de una Dimensión a la que hemos llamado “dimPanel”.

Mientras theta sea diferente de 2PI significará que no hemos dado la vuelta a toda la circunferencia. Tenemos 27 elementos, así que decimos que theta es 2Pi/27. Para hallar la posición de x hacemos el coseno de theta multiplicado por la anchura del panel, y para la posición y hayamos el seno multiplicado por la altura del panel. Todo esto trabajando en integers, así que necesitamos del Cast. Como haciendo esto no nos quedaba centrado, tuvimos que hacer una pequeña trampa manual y sumarle un determinado número de píxeles para que quedase centrado. Al tener ya la x y la y, hacemos un setbounds del objeto y lo situamos en la posición adecuada. Las imágenes utilizadas han sido todas de 32x32 píxeles, así que en el setbounds también configuramos el tamaño.

También tenemos en otro layout el área donde el usuario lee y contesta la pregunta. La definición se representa a través de un JTextArea, y el usuario debe introducir la respuesta a través de un JTextField.

Por último tenemos los botones de “Ok”, “Pasar” y “Salir”.

Juego

En esta parte detallaremos las clases y métodos del programa, todo aquello que el usuario no ve a simple vista pero que se ejecuta por debajo.

Jugador

Como su nombre indica, es la clase que contiene al jugador. Es una MAE, ya que es una instancia que es única, es decir, en nuestro diseño sólo puede haber un jugador jugando en una partida. Si fuese multijugador el diseño sería diferente, pero no es el caso.

Es una clase muy simple, lo único que tiene es el atributo “nombre”, que nos servirá para introducirlo posteriormente en la lista de puntuaciones. El método más interesante que tiene es “prepararJugador()”, que recoge el nombre introducido por pantalla en un String a través de un InputDialog, y posteriormente se lo asigna al objeto a través del método privado “setNombre()”. Por último, tiene los métodos “getNombre()” y “getJugador()”.

Definición

En este juego tenemos que responder preguntas, así que está clase tiene por un lado un atributo de tipo String llamado “definición”, contiene la pregunta a realizar, por otro lado tiene otro atributo llamado “término”, que es la respuesta a dicha pregunta, y por último un char llamado “letra”. En esta clase no puede faltar un atributo de tipo enumerable que hemos llamado “estado”. Esto es lo que nos permitirá saber si la pregunta está contestada, acertada o fallada.

Esta clase tiene sus correspondientes getters y setters, aunque los dos métodos más interesantes son “estaContestada()” y “comprobarRespuesta()”. El primer método puede parecer redundante, pero esa pregunta que le hacemos va a tener su utilidad en el rosco, ya que explicaremos como las preguntas contestadas son eliminadas. Comprobar respuesta lo que hace es mirar si la respuesta dada por el usuario se corresponde con el “término” de la definición. Si es igual se pone a true el estado a través de un método privado que cambia el estado a “acertado”. En caso contrario hace lo mismo pero lo pone como fallado. En caso de que la pregunta no se conteste, no se llama a ningún método de comprobación, y se queda como no contestada.

ListaDefiniciones

Es una lista como cualquiera con las que hemos trabajado que en su interior guarda objetos de tipo “Definición”. En este caso utilizamos la estructura de Array. Contiene los métodos “getDefinicion()”, “anadirDefinicion()” y “obtenerDefinicionAleatoria()”. El método más interesante es el último de todos, ya que es el que nos permite coger una posición aleatoria entre un número que se encuentre entre el 0 y el tamaño del array menos una posición. Es decir, si el array tiene 20 elementos nos coge un número al azar entre el 0 y el 19, que se corresponde con la posición de una celda del array que contiene una Definición. Esto nos permite que cada partida tenga un rosco diferente.

Diccionario

Esta clase tiene una particularidad, es una lista de listas. Hemos decidido separar las listas por letras, así que diccionario alberga unas 27 listas de palabras. Para que sea lo más eficiente posible hemos decidido aplicar los conocimientos adquiridos en la asignatura "EDA" y utilizar la estructura HashMap. La clave o key del HashMap será la letra, y el contenido, la lista de palabras que empiecen por dicha letra.

Para dejar a punto el diccionario, utilizamos una clase llamada "LectorXMLDiccionario", que comentaremos más tarde. La llamada a esta clase se hace mediante el método "prepararDiccionario()".

Esta clase también tiene un método "obtenerDefinicionAleatoria". En este caso se le pasa un char, que es la key del HashMap mediante la cual se puede acceder a la lista. El otro método también necesario es "anadirDefinicion()".

Rosco

Subiendo un poco más de nivel, tenemos la clase Rosco, que no funcionaría sin todo lo que hemos detallado anteriormente.

Al igual que con Jugador, Rosco es también una MAE, por lo tanto sólo podrá existir una instancia de esta clase. Esta clase tiene una particularidad, y es que hemos decidido de implementar un tipo de estructura de datos que Java no implementa por defecto, se trata de la DoubleLinkedList. Explicaremos más detalladamente esta decisión cuando expliquemos dicha fase, de momento adelantar que nos parece que se trata de la estructura de datos que más se asemeja con el funcionamiento del Rosco en la realidad.

En los atributos del rosco tenemos definido el tiempo de la partida en milisegundos. Hemos puesto unos 250 segundos, que nos parece un tiempo razonable para que el usuario conteste. Tal vez no disponga de mucha holgura, pero no es imposible acabar el juego en ese tiempo. El rosco también va apuntando los aciertos y los fallos, tiene un atributo llamado "sonido" por si el usuario decide que haya sonidos al responder las preguntas, y "definicionActual", que es un puntero a la pregunta en la que nos encontramos.

Entre los métodos tenemos los getters y setters, métodos para calcular aciertos, fallos y hallar la puntuación obtenida y otros más interesantes como "prepararRosco()" y "comprobarRespuesta()". El primero de todos mira la configuración del jugador, prepara las preguntas del rosco, pone el reloj a cero y hace lo mismo con el número de aciertos y de fallos. El último lo único que hace es delegar la comprobación en Definición, y ejecutar los métodos que correspondan. Por ejemplo, si hay opción de sonido llamará al método "reproducirSonido()" presente en esta misma clase, si la respuesta es acertada se sumará el acierto y se cargará la siguiente definición a través del método "cargarDefinicion()", etc.

DoubleLinkedList

Esta clase tiene gran interés. Hemos escogido una lista ligada doble porque realmente la estructura es circular, y es lo que desde nuestro punto de vista más se asemeja al rosco. Además nos permite otras muchas funciones, por ejemplo, podemos saber el contenido del siguiente nodo mediante un apuntador, o incluso podemos borrar del rosco las preguntas que ya están contestadas para que no se vuelvan a repetir y ahorrarnos fallos. El no tener que pasar dos veces por una casilla que ya ha sido contestada es ya de por sí algo positivo, al menos para el diseño de este juego.

En esta lista cabe destacar que tenemos el primer elemento y el actual. En caso de querer ir al siguiente elemento tendremos que entrar dentro del Nodo y pedirle el siguiente elemento.

Para obtener las definiciones, se delega en diccionario, al que se le pasa un Array con los chars, que hacen referencia al alfabeto de nuestra lengua. Por cada letra se obtiene una definición aleatoria que se añade a un nuevo nodo.

El método “quedanPreguntas()“ lo que hace es comprobar el tamaño de la lista, si es igual a cero, es que todas están ya respondidas. “CargarSiguientePregunta()” se encarga de poner al atributo “actual” la siguiente Definición.

Nodo

Acabamos de ver la DoubleLinkedList, y esta se compone a su vez de Nodos, o para decirlo de otra forma, apuntadores.

El nodo contiene una Definicion, y a su vez contiene otros Nodos, que son el anterior y el siguiente. Esto nos permite conocer los elementos anteriores y posteriores para poder trabajar con ellos. Aparte de sus getters y setters, esta clase contiene el método “comprobarRespuesta()”, que llama a definición pasándole un String para que se compruebe allí, es decir, delega la tarea en Definición.

Puntuación

Esta clase tiene dos atributos, “nombre” y “puntuación”. Entre los métodos tiene los correspondientes getters y setters, “imprimir()” y “compararPuntuacion()”. Este último nos es útil porque en la lista de puntuaciones no va a haber nombres repetidos. Así que si un jugador juega una partida más de una vez, la puntuación guardada se actualizará con la más alta.

ListaPuntuaciones

Es una lista común. Lo más relevante que se puede decir es que obtiene la información a partir de un fichero xml. La lista queda ordenada por la puntuación, el jugador con mayor número de puntos encabezará la lista.

Reloj

Se trata de un cronómetro. Se le da un valor de inicio y va restando el tiempo. Es capaz de decirnos el tiempo restante que nos queda, parar el tiempo en un momento determinado y volver a reanudar la cuenta atrás. Trabaja en milisegundos, así que para nuestro juego, tenemos que hacer que pueda utilizar segundos.

LectorXMLPuntuaciones y LectorXMLDiccionario

Son clases que se encargan de leer de un archivo con extensión xml. Son útiles para guardar y leer datos y poder cargarlos en una lista. Posteriormente explicaremos el proceso.

Modificaciones

Durante el desarrollo del proyecto nos hemos visto en la obligación de realizar modificaciones, de forma que algunas cosas de las que están escritas en páginas atrás ya no son así porque se han realizado cambios. Dividiremos en apartados aquellos cambios que hayamos realizado.

Patrón Observador-Observable

Tras tener hecha la implementación y estar funcionando se nos pidió que implementásemos el patrón observador-observable. Es un patrón de diseño que define una dependencia del tipo uno-

a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.

Esto lo hemos tenido que implementar en la clase “InterfazRosco2”. Lo que nos permite es que cuando haya un cambio tras la acción del usuario, se cambien todos los elementos de la interfaz que dicha acción implica. Por ejemplo, si el usuario contesta una pregunta y la acierta, al pasar a la siguiente querremos que la bola correspondiente a la letra anterior se ponga de color verde, que la bola correspondiente a letra actual ya no sea azul, sino que sea de color amarillo, que se escriba por pantalla la nueva pregunta, se actualice el tiempo, el número de aciertos, etc.

ListaBolas

Como dijimos anteriormente, la clase “InterfazRosco2” se componía de más de una centena de JLabels, ya que la gran mayoría era para las bolas. Teníamos 27 bolas multiplicadas por cuatro colores. Ahora eso va a cambiar.

En la clase “InterfazRosco2”, hemos eliminado todas las JLabels para tener cuatro “ListaBolas”, una por color utilizado. Para eso hemos tenido que crear una nueva clase llamada “ListaBolas” de rellenar un HashMap de bolas, accesible por la letra como key, y de administrarlo correctamente.

El método “ordenar()”, que se encarga de mostrar por pantalla las bolas en forma circular, también se ha visto afectado, y ahora ha quedado mucho más corto y comprensible que en un inicio.

Por último, ahora en vez de definir una a una cada JLabel correspondiente a cada letra y color, tenemos un método llamado “anadirLabelsAlPanel()” que lo hace de manera automática sin tener que llenar toda la clase con esas líneas innecesarias.

Respuestas múltiples

Otro de los cambios que hemos hecho en el código es contemplar la posibilidad de que una misma definición tenga más de una posible respuesta, es decir, que existan sinónimos.

Para ello hemos aplicado la herencia. Tenemos la clase definición, como antes, pero ahora heredan de ella dos nuevas clases: “DefinicionUnTermino” y “DefinicionVariosTerminos”.

Definición es ahora una clase abstracta, ya que no nos interesa tener instancias de ella sino que la empleamos para que otras clases hereden de ella. Esto se debe a que por las características de nuestro lenguaje algunas palabras pueden contener acentos de modo que si el Jugador escribe una respuesta sin acento se podría dar el caso de que no se reconociese la palabra debido a la falta de dicho acento. Además, también se puede dar el caso de que una pregunta pueda tener varias respuestas validas –por ejemplo en nuestro juego tenemos ese problema con las palabras kiosco y kiosko que son ambas aceptadas-

Por tanto, vamos a tener palabras con varias respuestas y palabras con una respuesta. Es por ello que para no desperdiciar memoria, las palabras que admiten varias respuestas serán del tipo DefinicionVariosTerminos mientras que las que solo admiten una respuesta serán del tipo DefinicionUnTermino.

En esta clase Definicion vamos a tener por tanto, la definición, o pregunta, el estado de dicha Definicion, (si ha sido acertada, fallada, o si aún no ha sido contestada), y la letra por la que empieza, esto último debido a que se emplea en la interfaz. Además consta de varios métodos que ayudan a la interfaz a obtener la definición y la letra, otro que informa si ha sido respondida

la pregunta o no y que es empleado por el Rosco para asegurarse de que no va a hacer al usuario una pregunta que ya ha acertado previamente, y uno último que es abstracto y es el que dada la respuesta del usuario nos dice si ha acertado o no. Este último método es abstracto y no se implementa en esta clase sino que lo heredarán las clases que extienden a Definicion y lo implementarán como necesiten para poder comprobar cuando el usuario acierta o falla con su respuesta.

DefinicionUnTermino hereda la gran mayoría de sus atributos y métodos de la clase Definicion. Se le añade un atributo, el término, que es la respuesta correcta a dicha pregunta. En la constructora, por tanto, se le pasa este parámetro, además de la letra y la definición que serán los atributos enviados a la constructora de la clase padre.

Además tenemos el método Comprobar respuesta que hace es mira si la respuesta dada por el usuario se corresponde con el "término" de la definición. Si es igual se pone a true el estado a través de un método privado que cambia el estado a "acertado". En caso contrario hace lo mismo pero lo pone como fallado. Cabe destacar que a este método solo se le llamará cuando el usuario haya proporcionado una respuesta, ya que en la clase Rosco antes de llamar a este método se filtran las respuestas y en caso de que este en blanco se llama directamente al método pasar de la propia clase Rosco.

DefinicionVariosTerminos hereda la gran mayoría de sus atributos y métodos de la clase Definicion. Se le añade un atributo, un array de términos, que es una lista que contiene todas las respuestas a dicha pregunta que son admitidas como correctas. En la constructora, por tanto, se le pasa este parámetro, además de la letra y la definición que serán los atributos enviados a la constructora de la clase padre.

Además tenemos el método Comprobar respuesta que hace es mira si la respuesta dada por el usuario se corresponde con alguno de los "términos" de la definición. En caso de que alguno coincida se pone a true el estado a través de un método privado que cambia el estado a "acertado". En caso contrario hace lo mismo pero lo pone como fallado. Cabe destacar que a este método solo se le llamará cuando el usuario haya proporcionado una respuesta, ya que en la clase Rosco antes de llamar a este método se filtran las respuestas y en caso de que este en blanco se llama directamente al método pasar de la propia clase Rosco.

Almacenamiento persistente de datos: XML

En este juego hay datos que queremos almacenar. Esta información se corresponde al diccionario y a las puntuaciones de los jugadores. Para poder mantener esa información usamos archivos de tipo XML.

¿Qué es XML?

XML es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

No está destinado únicamente para su aplicación para Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

Se trata de una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

¿Por qué la usamos?

Para empezar, es un lenguaje muy fácil de entender y que se basa en las etiquetas. Estas etiquetas sirven para calificar la información que las rodea.

Las etiquetas no se entrecruzan, la primera en abrirse es la primera en cerrarse.

Un archivo XML es fácil de leer y entender para cualquiera, ya sea un ordenador o un humano. Con un ejemplo veremos cómo se almacena la información.

```
<?xml version="1.0" encoding="UTF-8"?>
 <diccionario>
    <listadefiniciones>
        <letra>A</letra> <!-- letra del alfabeto al que corresponde -->
       <palabras>
            <palabra>
                <termino>Atajo</termino>
                <definicion>Senda o lugar por donde se abrevia el camino.</definicion>
            </palabra>
       </palabras>
    </listadefiniciones>
</diccionario>
```

Lo primero que vemos es que podemos definir la codificación del texto. Usamos UTF-8, que tiene el inconveniente de que al usar únicamente 8 bits hay caracteres de nuestra lengua que no va a tener, pero como se ve, eso es modificable.

Vemos que empieza por diccionario. Recordemos que diccionario era un conjunto de listas de definiciones, que a su vez una de esas listas es un conjunto de definiciones, o como en el XML se define, palabras, y que cada definición tiene un atributo definición y un término.

Vemos así que el archivo XML guarda una coherencia con la forma en la que guardamos la información en nuestro programa una vez obtenidos los datos.

Además una de las ventajas que tiene el XML y por la cual nosotros la usamos es porque Java tiene ya implementado un método que permite la lectura de archivos XML.

¿Cómo se usa?

La lectura y obtención de datos se realiza a través de las clases LectorXMLDiccionario y LectorXMLPuntuaciones

Estas clases se encargan de leer los ficheros XML en los que guardamos dos aspectos esenciales del juego como son el Diccionario y la Lista de Puntuaciones. Para poder realizar la lectura utilizamos la clase SAXParser que contiene los métodos startElement, endElement y characters que tan útiles resultan a la hora de leer un fichero XML.

Con la ayuda de SAXParser vamos leyendo el fichero línea por línea y elemento por elemento permitiéndonos manejar cada elemento del modo deseado. Los métodos startElement y endElement son empleados para hallar cuando comienzan y terminan los elementos, mientras que characters es empleado para recoger las palabras que haya entre los tags. Por ejemplo si tuviésemos lo siguiente en un fichero xml:

```
<palabra>
		<termino>Arpa</termino>
		<definicion>Instrumento musical de forma triangular, con cuerdas
colocadas verticalmente y que se tocan con ambas manos.</definicion>
</palabra>
```

Lo primero que haría SAXParser sería leer el tag <palabra> en la primera línea. El método startElement se accionaría recibiendo como argumento palabra y a partir de ahí nosotros podemos prepararnos para recibir la palabra. Después, saltaría a la segunda línea <termino>Arpa</termino> pero como hemos dicho, solo lee elemento por elemento luego solamente leerá <termino> volviendo a accionar el método startElement que esta vez recibiría el argumento término. A continuación recibiría el término, en este caso "Arpa" y ahora no accionaría StartElement sino que se llamaría a characters donde, en caso de haber preparado bien la recepción, se recogería "Arpa" en una variable llamada término. Finalmente llegaríamos a </termino> que accionaría el método endElement y donde podríamos añadir el término a la definición en caso de que la hubiésemos creado ya. Nosotros creamos la Definicion al final por lo que solamente mantenemos el término en un atributo y pasamos a la siguiente línea.

Ahora vendría la línea con la definición. Se lee el tag <definición> igual que antes, se prepara para recibir la definición y cuando se lee la definición el método characters la almacena en otra variable. Finalmente, se llega a </definición> que como hemos dicho anteriormente no hace nada en nuestro caso.

Por último llegamos a </palabra> donde, ahora sí, al llamar a endElement, creamos una Definicion con el término y la definición recogida anteriormente en las variables.

Hemos explicado todo esto con el diccionario, pero con las puntuaciones es prácticamente igual.

```
<?xml version="1.0" encoding="UTF-8"?>
	<listapuntuaciones>
		<puntuacion>
			<nombre>Moore</nombre>
			<puntos>750</puntos>
		</puntuacion>
	</listapuntuaciones>
```

Imágenes del juego

A continuación podemos ver algunas capturas de pantallas obtenidas del juego.

Esta es la primera pantalla que vemos al visualizar el juego.

Se puede elegir "Jugar" para iniciar una nueva partida, "Puntuaciones" para ver las 10 mejores puntuaciones o "Reglas", para saber el funcionamiento del juego.

Por último tenemos una "checkbox" que nos permite decidir si queremos que se reproduzcan sonidos durante la partida o no. Los sonidos se reproducirían al acertar o fallar una respuesta.

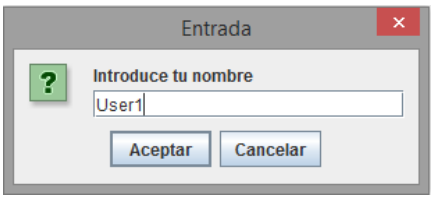
Aquí tenemos la ventana de puntuaciones. Tiene el listado con los 10 mejores jugadores. En este caso tenemos únicamente siete porque el archivo XML de donde se obtienen las puntuaciones no tiene más información.

También tenemos dos opciones, que son “Volver” y “Salir”.

“Volver” retorna al usuario al menú principal.

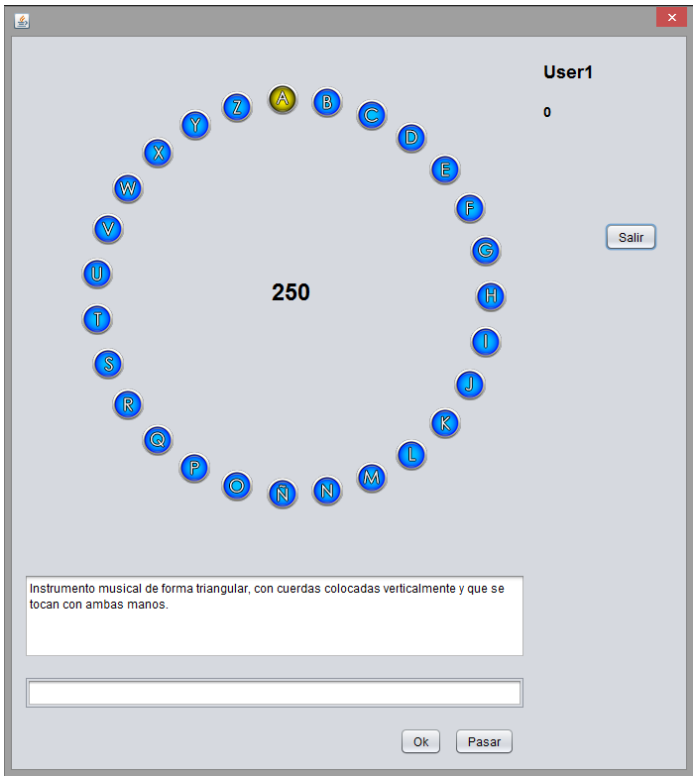
“Salir” cierra la aplicación.

Tenemos estos dos botones ya que esta misma ventana se mostrará cuando un usuario haya terminado su partida.



En el caso de que iniciemos una partida nueva nos saldrá esta ventana que permite la introducción de texto. En ella daremos el nombre con el que queramos jugar. Para esta prueba hemos dado el nombre de “User1”.

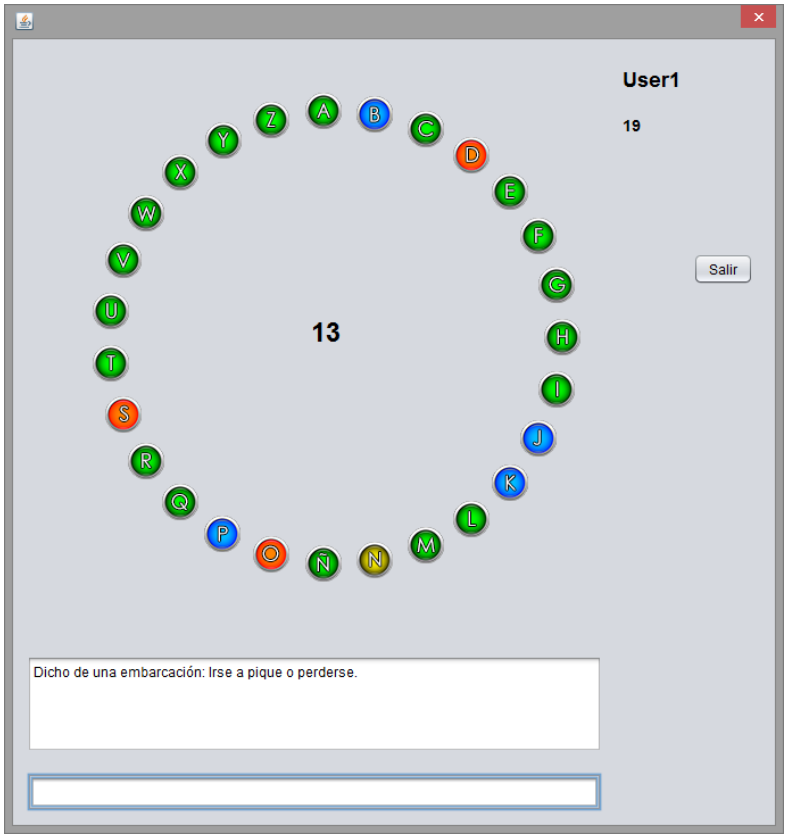
Una vez introducido el nombre nos debería salir una ventana como esta.



Vemos que en la esquena derecha está nuestro nombre y el número de preguntas acertadas. En este caso, como aún no hemos empezado, tenemos cero.

En el centro se encuentra nuestro rosco, perfectamente alineado y con todas las letras. En el centro de ese rosco podemos ver el tiempo del que disponemos para contestar.

En la parte inferior se puede ver la pregunta formulada y un campo de introducción de texto para contestarla.



Aquí se puede ver la partida a punto de finalizar. Vemos que está completamente diferente a como lo estaba al principio.

Ahora los aciertos han cambiado, tenemos 19.

Las bolas del rosco han cambiado. En azul se muestran las que están sin contestar, en ámbar la pregunta actual, en verde los aciertos y en rojo los fallos.

Vemos también que el tiempo ha sido actualizado.

Cuando el tiempo se acabe no se podrá contestar más, se podrá contestar la última, pero al igual que en el programa televisivo, no entrará porque estaremos fuera de tiempo, así que el programa finalizará y se mostrará la puntuación.

Así es como ha quedado la tabla. Se puede ver que ahora está "User1" con 445 puntos. En la primera foto que hemos mostrado no estaba.

Si "User1" vuelve a jugar y mejora su puntuación, se actualizará. En caso contrario, figurará en la lista la puntuación más alta obtenida con ese nombre.