Práctica 2: *Simón dice*

Fecha de entrega: 31 de enero de 2016

(Wikipedia.org)

0.- Descripción del juego

Simon dice es un juego electrónico, creado por Ralph Baer, que consiste en reproducir una secuencia de colores, con sus correspondientes sonidos, que son generados por la máquina de forma aleatoria. Una vez emitida la secuencia original, el usuario debe repetir la misma secuencia que generó la máquina, pulsando los botones (colores) correspondientes. Si el usuario repite correctamente la secuencia, la máquina generará una nueva más larga. Este proceso se repite hasta que el usuario comete un fallo o se llega a la secuencia con el límite máximo de colores.

1.- Versión 1 de la práctica

La primera versión de la práctica deberá permitir al usuario jugar una partida de *Simon dice*. Al inicio del programa se pide el nombre del jugador de la siguiente forma:

```
¡Bienvenido a Simon dice!
¿Cómo te llamas? Homer (Enter)
Hola Homer, pulsa una tecla para empezar a jugar. (Enter)
```

donde las letras marcadas en negrita representan las teclas pulsadas por el usuario.

En esta versión de la práctica se utilizarán 4 colores, como ocurre en la versión original del juego (**R**ojo, **V**erde, **A**zul y **D**orado).

La primera secuencia que genera la máquina estará compuesta por 3 colores. Si el usuario acierta una secuencia, se genera la siguiente secuencia a partir de ella añadiéndole al final un nuevo color.

Esto se repite hasta que:

a) El usuario comete un fallo y no acierta la secuencia.
b) El usuario acierta la última secuencia, formada por 12 colores.

Una vez la máquina ha generado la secuencia de colores aleatoriamente, el jugador podrá memorizarla hasta que presione una tecla. Seguidamente, se borrará la secuencia para que el usuario trate de adivinarla introduciendo los colores que considere oportunos.

```
Secuencia número 1: Rojo – Verde – Azul
Memoriza la secuencia y pulsa Enter para continuar… (Enter)
```

Seguidamente, el usuario introduce los colores por teclado, uno por cada línea. Para

```
Homer, introduce la secuencia de 3 colores:
R (Enter)
V (Enter)
A (Enter)

Enhorabuena, has acertado la secuencia número 1!
```

simplificar la entrada, cada color se representará con su letra inicial. En caso de que el carácter introducido por el usuario no se corresponda con un color, el programa deberá volver a pedir un carácter hasta que éste se corresponda con un color válido.

1.1 Detalles de implementación

Para este primer apartado de la práctica, los datos se representarán de la siguiente forma:

- Los colores se representarán con el tipo enumerado `tColores`.
- No se permitirá el uso de variables globales.
- La secuencia de colores se almacenará en un array de tipo `tSecuencia`:
  `typedef tColores tSecuencia[MAX_COLORES_SEQ]`

donde MAX_COLORES_SEQ será una constante mayor o igual que el número de colores de la secuencia (que en este caso es 12).

Además, se deberán implementar, al menos, las siguientes funciones:

```
tColores charToColor (char color);
```

Esta función recibe como parámetro un `char` que representa el color introducido por el usuario (primera letra) y devuelve su enumerado correspondiente. Para evitar inconsistencias, es recomendable comprobar la letra introducida, tanto en mayúscula, como en minúscula. No podrán existir dos colores cuyo nombre empiece con la misma letra.

Parámetros:

- `color`: Carácter que representa el color introducido por el usuario.

```
tColores intToColor (int numero);
```

Función recibe como parámetro un número entero y devuelve un color. El número recibido representa la posición del color en el tipo enumerado.

Parámetros:

- `numero`: Número entero.

```
void generarSecuencia (tSecuencia secuenciaColores, int numColores);
```

Esta función genera una secuencia de `numColores` colores de forma aleatoria, la cual se almacenará en el array `secuenciaColores` que se pasa como parámetro. Para generar los colores de forma aleatoria, se puede generar primero un número aleatorio y, seguidamente, utilizar la función `intToColor` para convertirlo al tipo `tColores`.

Parámetros:

- `secuenciaColores`: Array que contendrá la secuencia de colores generados aleatoriamente.
- `numColores`: Número de colores que debe contener la secuencia generada. Esto quiere decir que la secuencia se genera completa al principio del juego y al jugador solamente se le muestra en cada momento una parte cada vez más larga de ella.

```
bool comprobarColor (const tSecuencia secuenciaColores,
                     int indice, tColores color);
```

Función que comprueba si el color introducido por el usuario es correcto o no. Si el color es correcto, la función devolverá `true`, en caso contrario, devolverá `false`.

Parámetros:

- `secuenciaColores`: Secuencia de colores actual.
- `indice`: Índice del color que se desea comprobar.
- `color`: Color introducido por el usuario.

```
void mostrarSecuencia(const tSecuencia secuenciaColores, int numero);
```

El objetivo de esta función es mostrar la secuencia actual por pantalla para que el usuario la pueda memorizar. Al pulsar una tecla, la secuencia debe borrarse para permitir que el usuario introduzca los colores. Para borrar la pantalla se puede hacer uso de la llamada `system("cls")`.

Parámetros:

- `secuenciaColores`: Secuencia de colores actual.
- `numero`: Número de la secuencia actual.

```
void comenzarJuego(string nombre);
```

Función que llevará el control del juego. Será la encargada de controlar la secuencia en la que se encuentra el juego, así como de leer de teclado los colores que teclee el usuario y sacar por pantalla los correspondientes mensajes. Para generar las secuencias de colores y comprobar los colores introducidos por el usuario, se deberán utilizar las funciones anteriores.

Parámetros:

- `nombre`: Nombre del usuario que va a jugar.

2.- Versión 2 de la práctica

En la segunda versión de la práctica se va a implementar un modo más complicado del juego. Si consideramos la versión desarrollada en el apartado anterior como "modo sencillo", en esta versión se implementará el "modo difícil".

En este caso, el número de colores utilizado pasa de 4 a 7, estando la nueva gama de colores compuesta por **R**ojo, **V**erde, **A**zul, **D**orado, **B**lanco, **M**arrón y **N**aranja. Además, en este nuevo modo de juego existen 15 secuencias para acabar el juego, de forma que el juego es más largo y, a su vez, más complicado.

Para facilitar las cosas a los jugadores, se debe implementar un sistema de ayudas. Una ayuda consiste en solicitar a la máquina el color actual de la secuencia, de forma que la máquina debe ayudar al jugador cuando éste se olvide de un color. Obviamente, el número de ayudas es limitado, siendo su valor por defecto de 3 ayudas por partida. Las ayudas estarán disponibles en los dos modos de juego.

Para solicitar una ayuda, el usuario deberá introducir una 'x'. Si el usuario tenía ayudas disponibles, la máquina le facilitará el color actual de la secuencia. En caso contrario, no se facilitará ningún color y el usuario deberá intentar adivinarlo.

El siguiente ejemplo muestra la secuencia de acciones durante una partida en modo difícil:

```
Ayudas disponibles: 3
Homer, introduce la secuencia de 5 colores:

R (Enter)
N (Enter)
x (Enter)

El siguiente color es el Blanco. Te quedan 2 ayudas!
…
```

Finalmente, se debe implementar un menú donde se permita al usuario poder elegir entre las siguientes opciones:

```
Elija una opción para continuar:
0: Salir.
1: Jugar en modo sencillo.
2: Jugar en modo difícil.
```

La opción 0 se utiliza para salir del programa.

Las opciones 1 y 2 comenzarán la partida en los modos sencillo y difícil, respectivamente.

2.1 Detalles de implementación

Para este segundo apartado de la práctica, los datos se representarán de la siguiente forma:

- Se deberá modificar el tipo enumerado `tColores` para que contenga los nuevos colores.

- Se utilizará un tipo enumerado `tModo` para representar cada modo de juego.
- Se deberá incluir una nueva constante `MAX_COLORES_DIFICIL` que contenga el número de colores posibles en el modo difícil y otra `MAX_COLORES_FACIL` que contenga el número de colores posibles en el modo fácil. La constante `MAX_COLORES_SEQ` ha de ser mayor que ambas, de forma que el tipo `tSecuencia` no ha de modificarse, y sirve igualmente para ambos modos.
- No se permitirá el uso de variables globales.

Las siguientes funciones deberán ser modificadas, con respecto a las desarrolladas en la versión anterior, para que cumplan la funcionalidad pedida:

```
void comenzarJuego(string nombre, tModo modo, int ayudas);
```

Función que llevará el control del juego. En esta versión se añadirá un parámetro para indicar el modo de juego al que se va a jugar, así como el número de ayudas que dispondrán los jugadores. De forma similar al apartado anterior, será la encargada de controlar la secuencia en la que se encuentra el juego, así como de leer de teclado los colores que teclee el usuario y sacar por pantalla los correspondientes mensajes.

Parámetros:

- `nombre`: Nombre del usuario que va a jugar.
- `modo`: Modo de juego.
- `ayudas`: Número de ayudas disponibles durante la partida.

```
void generarSecuencia (tSecuencia secuenciaColores,
                       tModo modo, int numColores);
```

En este apartado se modificará esta función incluyendo el modo de juego. De esta forma, es posible calcular cómo generar aleatoriamente los colores según el modo de juego seleccionado.

Parámetros:

- `secuenciaColores`: Secuencia de colores generados aleatoriamente.
- `modo`: Modo de juego.
- `numColores`: Número de colores que debe contener la secuencia.

Además, se deberá implementar, al menos, la siguiente función:

```
bool utilizarAyuda (const tSecuencia secuenciaColores,
                    int indice, int &numAyudas);
```

Función que se encarga de comprobar si se puede, o no, ayudar al usuario. Inicialmente, se debe comprobar si al usuario le quedan ayudas disponibles. En caso afirmativo, se restará una ayuda y se mostrará el color siguiente por pantalla. En caso negativo, se mostrará por pantalla que el usuario no dispone de más ayudas. La función devolverá `true` en caso que se utilice una ayuda y `false` en otro caso (si no quedan ayudas).

Parámetros:

- `secuenciaColores`: Secuencia de colores actual.
- `indice`: Índice del color actual.
- `numAyudas`: Ayudas disponibles para el usuario.

3.- Versión 3 de la práctica

Para hacer más interesante el juego, vamos a dotar al mismo de un sistema de puntuaciones. Este sistema se aplicará tanto al modo sencillo como al modo difícil.

Las puntuaciones van a consistir en lo siguiente:

a) Cada jugador empieza su partida con 0 puntos.
b) Por cada color acertado sumará 5 puntos.
c) Por cada secuencia completa acertada, sumará 10 puntos.
d) Por cada ayuda utilizada, restará 7 puntos.
e) Si el jugador está en modo difícil, multiplicará cada apartado anterior por 1.5.
f) No se podrán obtener puntuaciones negativas. Es decir, como mínimo un jugador obtendrá 0 puntos.

El programa deberá mantener una lista con las puntuaciones de los 10 mejores jugadores. Éstos se almacenarán en el fichero "top.txt" (sin comillas) de la siguiente forma:

a) Cada línea contiene 2 valores.
    a. El primer valor es la puntuación del jugador (`float`).
    b. El segundo valor es el nombre del jugador (`string`).

Las estructuras de datos que contienen la información de los 10 mejores jugadores deberán cargarse, de forma implícita, al iniciar el programa, es decir, sin necesidad de que el usuario lo indique con una opción del menú. Así, al arrancar el programa, la información existente en el fichero "top.txt" (sin comillas) quedará almacenada en memoria. En caso que no exista este fichero, se inicializarán las estructuras necesarias asumiendo que no hay, todavía, mejores jugadores. Una vez se vayan jugando partidas, se irán completando estas estructuras.

El menú deberá mostrar una nueva opción para visualizar la lista de los mejores jugadores, tal como muestra el siguiente cuadro:

```
Elija una opción para continuar:
0: Salir.
1: Jugar en modo sencillo.
2: Jugar en modo difícil.
3: Ver 10 mejores jugadores.
4: Ver jugador(es) con la puntuación más alta.
```

donde la opción 3 mostrará por pantalla los 10 mejores jugadores. Si en un determinado momento en la ejecución del juego, no existen los 10 jugadores, sólo se mostrarán los existentes.

La opción 4 mostrará el jugador (o jugadores) con la puntuación más alta. En caso que existan varios jugadores con la puntuación más alta, deberán mostrarse todos ellos.

Al finalizar la ejecución del programa, es decir, cuando el usuario seleccione la opción 0 del menú, las estructuras de datos con la información de los 10 mejores jugadores deberán grabarse en el fichero "top.txt" (sin comillas).

Cada vez que se finalice una partida, se deberán actualizar las estructuras que gestionen las 10 mejores puntuaciones, así como el nombre de sus jugadores. Cuando estas estructuras contengan menos de 10 jugadores, bastará simplemente con insertar en el hueco correspondiente la nueva puntuación. Cuando existan 10 jugadores en el sistema, se deberá sustituir al jugador con menor puntuación, por el jugador actual. Ya que, de lo contrario, no habrá hueco suficiente para todos los jugadores. En caso de existir varios jugadores con la puntuación mínima, se podrá elegir a cualquiera de ellos para ser eliminado.

Adicionalmente, y de manera opcional, se pueden mantener en memoria las puntuaciones ordenadas, de forma que al mostrar por pantalla esta información, los primeros jugadores sean aquéllos con las puntuaciones más altas.

3.1 Detalles de implementación

Para este tercer apartado de la práctica, los datos se representarán de la siguiente forma:

- Se utilizará una constante para representar el número máximo de mejores jugadores, `MAX_JUGADORES`, que tendrá un valor de 10.
- Se puede utilizar el centinela "-1" (sin comillas) para marcar el final de fichero.
- Para representar la lista de los mejores jugadores se ofrecen dos posibilidades:
    a. Utilizar 2 arrays de tamaño `MAX_JUGADORES`. El primero será de tipo `string` y contendrá los nombres de los mejores jugadores. El segundo será de tipo `float` y contendrá las puntuaciones de los jugadores. De esta forma, se mantiene emparejado a su jugador con su puntuación ya que se utiliza el mismo índice para acceder a ambos arrays.
    b. Utilizar un único *array* de *structs*, donde cada posición contenga una estructura con dos campos: el nombre del jugador (`string`) y su puntuación (`float`).

Las siguiente función deberá ser modificada, con respecto a la desarrollada en las versiones anteriores, para que cumpla la funcionalidad pedida:

```
float comenzarJuego(string nombre, tModo modo, int ayudas);
```

En esta versión de la práctica, esta función devolverá la puntuación obtenida por el jugador, realizando además la misma funcionalidad que la descrita en el apartado anterior.

Parámetros:

- `nombre`: Nombre del usuario que va a jugar.
- `modo`: Modo de juego.
- `ayudas`: Número de ayudas disponibles durante la partida.

Además, se deben incluir las funciones indicadas a continuación para gestionar las puntuaciones. Los parámetros de estas funciones están sin especificar. Dependiendo de

cómo se haya decidido implementar la lista de los 10 mejores jugadores, se deberán pasar a las siguientes funciones los parámetros correspondientes.

```
void cargarRanking (…);
```

Esta función se debe invocar al arrancar el programa. Su función es comprobar la existencia del fichero "top.txt". Si este fichero existe, se deberán cargar las estructuras de datos pertinentes con el contenido del fichero. En caso contrario, estas estructuras permanecerán vacías. Como parámetros, esta función recibirá las estructuras de datos para albergar la puntuación de los 10 mejores jugadores.

```
void escribirRanking (…);
```

Esta función se debe invocar antes de salir de la aplicación. Su función es grabar en el fichero "top.txt" los mejores jugadores que se encuentren en las estructuras de datos correspondientes. Como parámetros, esta función recibirá las estructuras de datos que contienen la puntuación de los 10 mejores jugadores.

```
void visualizarRanking (…);
```

Función que muestra por pantalla los mejores jugadores. Como parámetros, esta función recibirá las estructuras de datos que contienen la puntuación de los 10 mejores jugadores.

```
void visualizarBestPlayer (…);
```

Función que muestra por pantalla el jugador (o jugadores) con la puntuación más alta. En caso de empate, se mostrarán todos los jugadores con esta puntuación. Como parámetros, esta función recibirá las estructuras de datos que contienen la puntuación de los 10 mejores jugadores.

4. Entrega de la práctica

La práctica se entregará en el Campus Virtual por medio de la tarea Entrega de la Práctica 2, que permitirá subir un archivo comprimido conteniendo el archivo con el código fuente de la versión 3 (archivo main.cpp) y el archivo con las mejores puntuaciones de los jugadores (archivo top.txt).

Asegúrate de poner el nombre de los miembros del grupo en un comentario al principio del archivo de código fuente.

Y recuerda, ¡las prácticas no se copian! Una práctica copiada implica suspender la asignatura.

Fecha de entrega: **31 de Enero de 2016**