SIMÓN DICE

Abstract. — El objetivo de este proyecto es coger soltura con el manejo de información cuando esta es organizada en arrays y se mantiene un control sobre los índices para conocer en todo momento la capacidad de los mismos. Al mismo tiempo, será necesario declarar algunos tipos enumerados, hacer uso de condiciones lógicas y, en las últimas versiones del proyecto, escritura/lectura de ficheros e implementación del algoritmo de ordenación de la burbuja.

1. Descripción del juego

Simón dice [**1**] es un juego electrónico, creado por Ralph Baer [**2**], 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.

2. Versión 1 del proyecto

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

```
Welcome to Simon dice!
What is your name? Homer (Enter)
Hello Homer, press ENTER to start playing (Enter)
```

En esta versión del proyecto 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:

I. El usuario comete un fallo y no acierta la secuencia.
II. 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 numero 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 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 este se corresponda con un color válido.

2.1. 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 `tColores`: `tColores[] secuenciaColores = new tColores[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, los siguientes métodos:

- `tColores charToColor (char _color)`: Este método 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.
    - `_color` representa el caracter del color introducido por el usuario.
- `tColores intToColor (int _numero)`: Este método recibe como parámetro un número entero y devuelve un color. El número recibido representa la posición del color en el tipo enumerado.
    - `_numero` representa el número entero.
- `void generarSecuencia (int _numColores)`: Este método genera una secuencia de colores de forma aleatoria dentro del intervalo [0, `_numColores`], la cual se almacenará en el array `secuenciaColores`. 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`.
    - `_numColores` representa el número de colores que tiene el tipo enumerado `tColores`. 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.
- `boolean comprobarColor (int _index, tColores _color)`: Método que comprueba si el color introducido por el usuario es correcto o no, devolviendo un booleano que indica si ha habido fallo. Si el color es correcto, la función devolverá `false`, en caso contrario, devolverá `true` (pues el usuario ha fallado).
    - `_index` es el índice que el color ocupa dentro del array `secuenciaColores`.
    - `_color` es el color introducido por el usuario.
- `void mostrarSecuencia (int _numero)`: El objetivo de este método 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 del siguiente bucle:
    - `_numero` es el número de la secuencia actual.

```
for (int i = 0; i < 50; ++i)
  System.out.println();
```

- `void play ()`: Método que llevará el control del juego. Será el encargado 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.

3. Versión 2 del proyecto

En la segunda versión de la práctica se va a implementar un modo más complicado del juego. Si la versión anterior era el "modo fácil", está será el "modo difícil". En este caso, los colores utilizados pasan de ser 4 a ser 7 (**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 terminar 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 este no recuerde un color. Obviamente, es 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.

```
Introduce la secuencia de 3 colores (X para la ayuda):
A (Enter)
X (Enter)
El siguiente color es el: Marron. Te quedan 2 ayudas.
B (Enter)
Enhorabuena, has acertado la secuencia numero 1
Secuencia numero 2: Azul Marron Blanco Blanco
Memoriza la secuencia y pulsa ENTER para continuar
```

3.1. Detalles de implementación. — Se usarán, al menos, estos métodos y atributos:

- Un tipo enumerado `tModo` para representar los modos de juego.
- Se deberán incluir dos constantes `MAX_COLORES_FACIL` y `MAX_COLORES_DIFICIL` para determinar el número de colores posibles en el modo fácil y en el modo difícil respectivamente.
- El método `play()` recibe el modo de juego como parámetro.
- No se permite el uso de variables globales.
- `boolean usarAyuda (int _index)`: Método que permite utiliza las ayudas disponibles. Si no quedan, deberá mostrar un mensaje informando sobre ello.
  - `_index` índice del color i-ésimo que desea comprobar.

Se debe implementar un menú donde se permita al usuario poder elegir entre las opciones disponibles

```
Elige una opcion para continuar:
    0 - Salir.
    1 - Jugar en modo facil.
    2 - Jugar en modo dificil.
```

4. Versión 3 del proyecto

Para hacer más interesante el juego, vamos a dotar al mismo de un sistema de puntuaciones. Este sistema se aplicará tanto al modo fácil como al modo difícil. Las puntuaciones van a consistir en lo siguiente:

I. Cada jugador empieza con 0 puntos.
II. Por cada color acertado se sumarán 5 puntos.
III. Por cada secuencia acertada se sumarán 10 puntos.
IV. Por cada ayuda utilizada se restarán 8 puntos.
V. Si el jugador está en modo difícil, se multiplicará la puntuación por 2.
VI. No se podrán obtener puntuaciones negativas.

El programa deberá mantener una lista con las puntuaciones de los jugadores. Estos se almacenan en un fichero `top.txt` en dos columnas, donde la primera de ellas almacena la puntuación del jugador y la segunda el nombre.

```
84 Tomas
75 Guillermo
55 Felipe
42 Maria
42 Marcos
```

Cada uno de estos pares, cuando sean leídos desde fichero, será instanciado como un objeto de la clase genérica `Pair`. Además, los datos del fichero deberán cargarse al iniciar el programa, es decir, sin necesidad de que el usuario lo indique como una opción del menú. Así, al iniciar el programa, la información existente en el fichero quedará almacenada en memoria.

El menú deberá mostrar nuevas opciones tal y como se muestra en el siguiente ejemplo:

```
Elige una opcion para continuar:
    0 - Salir.
    1 - Jugar en modo facil.
    2 - Jugar en modo dificil.
    3 - Ver 10 mejores jugadores.
    4 - Ver jugador(es) con la puntuacion mas alta.
```

donde la opción 3 mostrará por pantalla los 10 mejores jugadores. Si existen menos jugadores, sólo se mostrarán los existentes. Si existen más, sólo se mostrarán los 10 primeros leídos desde fichero. Por otro lado, la opción 4 mostrará el jugador (o jugadores) con la puntuación más alta. En caso de que existan varios jugadores con la puntuación más alta, deberán mostrarse todos ellos. Al finalizar la ejecución del programa (cuando se pulse la opción 0), los datos deben ordenarse de mayor a menor puntuación y grabarse en el fichero `top.txt`.

Por último, cuando existan 10 o más jugadores en el fichero, el sistema, al comenzar una nueva partida, deberá sustituir al último jugador por el jugador actual.

4.1. Detalles de implementación. — Se usarán, al menos, los métodos y atributos que se describen a continuación. Además, la estructura del proyecto se ve modificada debido al uso de las clases de lectura y escritura de ficheros. El árbol debe tener este aspecto:

```
Proyecto
    src
        main package
            Engine
            Jugador
            Main
            Record
        data package
            top.txt
        files package
            CustomReadFile
            CustomWriteFile
            ICustomReadFile
            ICustomWriteFile
            Pair
```

donde `ICustomReadFile` y `ICustomWriteFile` son interfaces. Por lo tanto, las clases asociadas deben implementar los métodos que se declaren en dichas interfaces. En concreto, debemos tener en cuenta:

- Una constante `MAX_JUGADORES` para indicar el número máximo de jugadores.
- El método `play (tModo _modo)` debe devolver la puntuación obtenida cuando una partida ha finalizado.
- `void showRanking ()`: Muestra el ranking de los 10 mejores jugadores.
- `void showBestPlayer ()`: Muestra el jugador (o jugadores) con la puntuación más alta.
- `void cargarRanking ()`: Este método crea un objeto de tipo `CustomReadFile` y llama al método que se encarga de leer de fichero, que como no podía ser de otra forma se encuentra en la propia clase `CustomReadFile` (de ahí que debamos instanciar un objeto de esa clase).
- `void escribirRanking ()`: Este método crea un objeto de tipo `CustomWriteFile` y llama al método que se encarga de escribir en fichero, que como no podía ser de otra forma se encuentra en la propia clase `CustomWriteFile`. Específicamente, este método creará una cadena de tipo `String` con los jugadores y sus puntuaciones, respetando las dos columnas. Esta cadena será pasada por parámetro al método correspondiente en la clase `CustomWriteFile` para que así pueda escribir la información en el fichero.

El profesor proporcionará parte del código de las clases `CustomReadFile`, `CustomWriteFile`, `ICustomReadFile`, `ICustomWriteFile` y `Pair`.