Fundamentos de la programación

Grado en Desarrollo de Videojuegos

Examen parcial Febrero 2017

Indicaciones generales:

- Se entregará un único archivo `Program.cs` (generado con MonoDevelop) con el programa completo.
- La línea 1 será un comentario de la forma `// Nombre Apellido1 Apellido2`
- La línea 2 será un comentario de la forma `// Numero de laboratorio, Puesto`
- **Lee atentamente el enunciado** e implementa el programa tal como se pide, con los métodos, parámetros y requisitos que se especifican.
- El programa debe ser correcto, y estar bien estructurado y comentado. Se valorarán la claridad, la concisión y la eficiencia.
- **Entrega:** la entrega se hará a través del servidor FTP de los laboratorio

---

En este ejercicio vamos a implementar el **juego del ahorcado**: el ordenador selecciona aleatoriamente una palabra de un conjunto de palabras disponibles y el usuario tratará de descubrirla tanteando letras que puedan formar parte de la misma. Inicialmente el ordenador mostrará una secuencia de guiones correspondientes a los huecos de las letras que forman la palabra. Después, en cada ronda el jugador escoge una letra cualquiera y el ordenador *destapa* todas las posiciones donde aparece esa letra. Si la letra no aparece en la palabra se contabiliza un fallo (en el juego original se hace un trazo del dibujo de un ahorcado). El juego termina cuando el jugador descubre todas las letras que forman la palabra y gana, o bien cuando alcanza un número prefijado de fallos y pierde. Una posible secuencia de jugadas, con la palabra "CELESTINA" (en consola se mostraría todo en una sola columna):

```
Adivina mi palabra

---------
Fallos: 0

Tu letra: a
--------A
Fallos: 0
```

```
Tu letra: o
--------A
Fallos: 1

Tu letra: e
-E-E----A
Fallos: 1

Tu letra: b
-E-E----A
Fallos: 2
```

```
Tu letra: c
CE-E----A
Fallos: 2

Tu letra: s
CE-ES---A
Fallos: 2

Tu letra:
```

Inicialmente solo sabemos que la palabra tiene 9 letras y el contador de fallos está a 0. En la primera ronda el usuario escoge la `'a'` y el ordenador destapa la `'a'` del final de la palabra. Después escoge la `'o'` y como la palabra no tiene ninguna, incrementa el número de fallos. A continuación, con la `'e'` destapa las dos apariciones, con la `'b'` incrementa los fallos, etc.

Para simplificar el programa no consideraremos vocales acentuadas, eñes, ni diéresis. Así mismo, internamente se manejarán todas las cadenas y caracteres en mayúscula (aunque se recoja alguna minúscula del input, automáticamente se convierte a mayúscula). **Inicialmente**, utilizaremos solo la palabra CELESTINA para facilitar el desarrollo y el método `Main` tendrá el siguiente aspecto:

```
public static void Main (){
  const int MAX_FALLOS = 10;  // numero maximo de fallos permitidos
  string pal; // palabra a adivinar
  ... // inicializacion de la palabra

  bool[] descubiertas = new bool[pal.Length];
  // inicializa el vector de descubiertas
  ...

  // bucle principal de juego
```

La variable `pal` contiene la palabra con la que juega el ordenador. Inicialmente, para desarrollar el programa, podemos hacer `pal = "CELESTINA";` (después haremos un método para seleccionar aleatoriamente una palabra de juego). El array `descubiertas` determina las posiciones que contienen letras ya acertadas por el usuario y servirá para mostrar dichas posiciones en la palabra, en cada iteración del juego. En el ejemplo anterior, cuando el usuario ha probado las letras `'a'`, `'o'`, `'e'`, `'b'`, `'c'`, `'s'`, tendremos la siguiente situación:

`pal`	C	E	L	E	S	T	I	N	A
`descubiertas`	true	true	false	true	true	false	false	false	true
en pantalla	C	E	-	E	S	-	-	-	A

Es decir, `pal` contiene la cadena "CELESTINA", el array de descubiertas tiene a `true` las posiciones correspondientes a las letras probadas hasta el momento (`'C'`, `'E'`, `'S'`, `'A'`) y a `false` el resto. Y en pantalla se mostrará la parte de la palabra con las posiciones descubiertas (guiones para las otras posiciones).

Recordemos que los caracteres individuales de un `string` son accesibles del mismo modo que las componentes de un vector, con la notación de corchetes (solo para ver el contenido, no para modificarlo). Es decir, `pal[0]` vale `'C'`, `pal[1]` vale `'E'`, etc.

Se pide, implementar los siguientes métodos (se especifican los parámetros con su tipo, pero **debe decidirse la forma de paso** entrada/salida/entrada-salida y añadir los modificadores correspondientes):

- [**1 pt**] `void Muestra(string pal, bool [] descubiertas, int fallos)`: dada una palabra `pal` y el vector de posiciones `descubiertas`, muestra en pantalla la parte de la palabra ya adivinada y guiones la parte no descubierta, tal como se ha explicado arriba. Además mostrará el número de `fallos` tal como se ve en el ejemplo.

- [**1 pt**] `bool PalabraAcertada(bool [] descubiertas)`: determina si todas las posiciones del vector `descubiertas` son `true`.

- [**1 pt**] `char LeeLetra()`: solicita una letra de teclado, la lee, la convierte a mayúsculas (con el método `ToUpper()`) y devuelve el resultado.

- [**1 pt**] `void DescubreLetras(string pal, bool [] descubiertas, char let, bool acierto)`: pone a `true` las posiciones del vector `descubiertas` que contienen la letra `let` en la palabra `pal`, i.e., destapa las letras acertadas en una ronda. Además, `acierto` tomará el valor `true` en caso de que haya alguna letra acertada y `false` en caso contrario. Por ejemplo, con la palabra "CELESTINA" y la letra `'E'` se pondrán a `true` las posiciones correspondientes a las dos `'E'` de dicha palabra, y `acierto` tomará el valor `true`.

- [**2 pt**] Completar el método `Main` descrito arriba utilizando los métodos previos para implementar el juego. En particular, el bucle principal solicitará letras al usuario hasta que adivine la palabra o alcance el máximo de fallos.

- [2 pt] `void Selecciona(string file, string palabra)`: este método hará juego más realista, seleccionando **aleatoriamente** la palabra de un archivo de candidatos especificado en `file` y devolviéndola en `pal`. El archivo de entrada tendrá la forma:

```
6
COSA
CELESTINA
NARANJA
AMAPOLAS
ESTRATOS
ANIMAL
```

  Donde la primera línea indica el número de palabras y a continuación, cada línea contiene una palabra (asumimos que habrá exactamente el número de palabras indicado por el número de la primera línea).

El juego puede refinarse para que al solicitar una nueva letra en una ronda, el usuario sepa qué letras no ha probado aún y no tenga que memorizarlas. Para implementar esta idea, declararemos e inicializaremos un nuevo array en el método `Main`:

```
bool[] probadas = new bool[((int)'Z') - ((int)'A') +1];
for (int i = 0; i < probadas.Length; i++) probadas [i] = false;
```

Recordemos que la expresión `((int)'Z') - ((int)'A') + 1` calcula el número de letras de la `'A'` a la `'Z'`. De este modo la componente `probadas[0]` determina si la `'A'` ya ha sido probada, `probadas[1]` lo mismo para la `'B'`, etc.

Después implementaremos un **método alternativo** para recoger la letra del usuario (comentar la llamada al anterior `Selecciona`, ¡no borrarlo!):

- [2 pt] `char LeeLetra2(bool [] probadas)`: este método muestra en pantalla las letras que aun no han sido probadas por el usuario, recoge una nueva letra comprobando que aun no está marcada en el array `probadas` y la devuelve como salida. Con este nuevo método, para la palabra "CELESTINA", tras haber probado `'A'`, `'B'`, `'C'`, `'E'` tendríamos:

```
CE-E----A
Fallos: 1
Disponibles: D F G H I J K L M N O P Q R S T U V W X Y Z
Tu letra:
```

  Pistas: para convertir una letra `l` de la secuencia `'A'`, `'B'`, ..., `'Z'` al correspondiente índice $0, 1, \ldots, 25$ se puede utilizar la expresión `(int) l -'A'`. A la inversa, para convertir uno de esos índices en la letra correspondiente se puede hacer `(char) ('A' + i)`.