Programación de ejercicios en soporte informático para el aula de E/LE

PASCUAL CANTOS GÓMEZ
Universidad de Murcia

Una alternativa a la adquisición de programas para la Enseñanza Asistida por Ordenador, en general, y la Enseñanza de Lenguas Asistida por Ordenador, en particular, (EAO y ELAO en lo sucesivo), es siempre la creación de éstos, para lo cual el autor o docente dispone de varias aplicaciones informáticas:

1. Herramientas de programación:
   a. lenguajes de programación
   b. lenguajes de autor
2. Sistemas de autor.

1. Herramientas de programación

Son las herramientas indispensables para crear aplicaciones informáticas. Requieren buenos conocimientos informáticos, puesto que hay que saber programar.

Programar consiste básicamente en comunicar al ordenador lo que tiene que hacer. Las diversas fases de las que consta la programación informática son:

1. Análisis (unificar criterios, prever errores, etcétera).
2. Diseño (¿cómo va a ser la secuenciación del programa?, ¿qué grado de interacción alumno- ordenador?), diseño de las pantallas, sonidos, colores, etcétera.
3. Codificación.

Tras la codificación viene la etapa más laboriosa, que consiste en la corrección de posibles errores sintácticos y/o lógicos y en la ejecución una y otra vez del programa en su totalidad o en parte, para comprobar las correcciones.

Una vez concluida la fase de programación es importante guardar organigramas, listados y copias del programa. Esta documentación nos puede servir en otro momento para codificar el programa en otro lenguaje o para volver a utilizar rutinas o partes del programa en otros posteriores.

Es indudable que escribir un programa para la EAO/ELAO requiere muchas horas y buenos conocimientos de los lenguajes de programación. Pero a veces la escasez de programas hace que una vez adquiridos los ordenadores por el centro de enseñanza, al docente sólo le quede la alternativa de introducirse en el mundo informático para poder así sacar un mejor provecho de los ordenadores.

Hay que tener siempre presente que el docente no es un programador profesional, y que programar es una de las disciplinas más complejas de la informática (Underwood, 1984:43). Al mismo tiempo, nunca se debe considerar al profesor-programador como intruso en el área de la programación informática. Está claro que si se dispusiese de una oferta amplia de programas para la EAO/ELAO, el docente tendría dónde elegir y se ahorraría mucho tiempo. La solución estaría en la perfecta comunión entre docentes e informáticos, simbiosis más que necesaria para el futuro desarrollo de la EAO/ELAO. *Según estadísticas recientes, para la elaboración de material didáctico de tipo informático pensado para ser utilizado por el alumno en un espacio concreto de tiempo, se requiere un trabajo previo en horas de trabajo, que es hasta diez veces superior que si se hubiera tratado de material sonoro* (Ruipérez, 1990:69).

El problema se acentúa, además, entre los docentes-programadores, porque éstos nunca consideran sus obras como finalizadas, introduciendo constantemente cambios y mejoras. Esto puede, a veces, producir en ellos ciertos sentimientos de frustración, que, a la postre, repercuten negativamente en su experiencia con la EAO/ELAO.

Dejando al margen los problemas con que pueden encontrarse los profesores-programadores y volviendo a las herramientas de programación, diremos que éstas son las responsables de la comunicación directa entre ordenador-hombre. Atendiendo al mayor o menor grado de dificultad que implica el trabajar con esta clase de programas, distinguimos los siguientes tipos:

a) lenguajes de programación o de procedimiento.
b) lenguajes de autor.

A diferencia de Ruipérez (1990:65), no incluimos aquí los sistemas de autor, ya que éstos no precisan de ningún conocimiento de programación informática y no modifican al programa en sí.

a. Lenguajes de programación

Los ordenadores operan internamente con un lenguaje denominado código de máquina o lenguaje binario. Este lenguaje resulta difícil y complicado. Por ello, para no tener que recurrir a este medio de comunicación, se han creado lenguajes de programación o de procedimiento, que, haciendo uso de otro tipo de lenguaje simbólico más comprensible, traducen todas las instrucciones escritas en éste a código de máquina.

a. 1. BASIC

El lenguaje de programación de mayor difusión ha sido y sigue siendo, aunque ya en menor medida, el BASIC (*Beginners' All Purpose Symbolic Instruction Code*), por la sencilla razón de que este lenguaje se suministraba y se sigue suministrando gratis a la hora de adquirir un ordenador.

Una baza importante de esta primacía inicial del BASIC, respecto a otros lenguajes de programación para la EAO/ELAO, es su capacidad y su facilidad de operación con cadenas de caracteres. La manipulación de estas secuencias de caracteres es de capital importancia para confeccionar aplicaciones para la EAO/ELAO. Además, el BASIC es quizá uno de los lenguajes de procedimiento más fáciles de aprender. Muchas órdenes y funciones son simples verbos y sustantivos tomados directamente de la lengua inglesa, por ejemplo PRINT, INPUT, READ, etcétera.

A raíz de la popularidad del BASIC en la EAO, además de las ventajas citadas, algunos autores (Higgins, 1987; Leuschner, 1987) llegan a afirmar que es el lenguaje de programación más apropiado para el desarrollo de programas para la ELAO. Este hecho queda también patente en algunas obras (Kenning y Kenning, 1983; Abdulaziz, Smalzer y Abdulaziz, 1985) repletas de técnicas y rutinas de programación en dicho lenguaje para la elaboración de programas propios para la ELAO.

Sin embargo, la falta de estandarización del lenguaje BASIC y la constante aparición de nuevos dialectos, ha hecho que mucho material escrito en este lenguaje no sea compatible con el resto de los equipos.

A nivel técnico, muchos programadores se quejan de la falta de estructuración que presenta, pues la lectura de largos listados escritos en este lenguaje puede resultar muy complicada. El autor pierde, a veces, la visión general del programa. Así, algunos detractores del BASIC lo han considerado como un lenguaje de poca calidad, y hay programadores que relegan el BASIC a los últimos lugares en la escala de «prestigio».

a. 2. Otros lenguajes de programación

La falta de prestigio y de estructuración del BASIC ha hecho que muchos profesores-programadores optaran por otros lenguajes, sobre todo el PASCAL y el C. Otros se han decidido por el LOGO, lenguaje diseñado con fines didácticos, muy eficaz a la hora de manipular cadenas de caracteres y crear gráficos. Cabe citar también PROLOG y LISP, «hermanos mayores» del LOGO, conocidos por estar vinculados con la inteligencia artificial.

En realidad no podemos decir que éste o aquél sean el mejor lenguaje de programación para la EAO/ELAO, a diferencia de lo que ocurre en otras áreas del conocimiento, ya que existen lenguajes creados específicamente para estos campos. Así, el COBOL está concebido más para aplicaciones comerciales, el FORTRAN para aplicaciones matemáticas y el LISP y PROLOG para la inteligencia artificial. Sin embargo, el BASIC, PASCAL y, en mayor medida, el C, son lenguajes que permiten aplicaciones generales en cualquier área, aunque, posiblemente, no con la eficacia de los lenguajes de programación específicos que acabamos de citar.

b. Lenguajes de autor

Son también conocidos como *programas de autor*. Los podríamos definir como lenguajes de procedimiento orientados a la creación de aplicaciones en la EAO/ELAO. Incluyen ya una serie de rutinas definidas. Así la confección de ejercicios interactivos para la EAO/ELAO puede resultar mucho más fácil, ya que son menos complejos que los de programación y requieren menos conocimientos informáticos. Son, por lo tanto, lenguajes de programación simplificados, menos exigentes en su sintaxis que los lenguajes estándar y proporcionan al autor rutinas ya definidas para muchas de las actividades habituales, como definición de pantallas, entrada de respuestas, evaluación de las mismas, estrategias de ramificación, etcétera.

Los lenguajes de autor han sido diseñados especialmente para docentes que sin amplios conocimientos informáticos quieran desarrollar sus propias aplicaciones para la EAO/ELAO.

b. 1. PILOT

Hay muchos lenguajes de autor: TES/T, EDUTEXT, TICCIT, EXTOL, MICROTEXT (Motteram 1990) o TUTOR, pero quizá los más conocidos y difundidos sean CALIS (y su versión para Windows: WINCALIS) y PILOT (*Programmed Instruction Learning or Teaching*; Sanders 1985) y su versión posterior para ordenadores IBM y compatibles, el SUPER-PILOT. Su gran capacidad de operar con cadenas de caracteres, gráficos y animación hacen de él una herramienta muy a tener en cuenta, además del ahorro de tiempo que implica a la hora de confeccionar material didáctico.

Por otro lado, PILOT sólo permite ejercicios interactivos del tipo pregunta-respuesta, para un modelo de enseñanza tutorial. Además, dado que lleva las rutinas ya definidas, el autor puede encontrarse algo «encorsetado» ya que, aunque sea más fácil de manejar que un lenguaje de programación, no goza de la versatilidad y flexibilidad de aquél. A esto se le suma la falta de estructuración y los problemas que esto conlleva. Los factores que acabamos de ver pueden dar una imagen pobre de lo que realmente se puede lograr en la EAO/ELAO y dificultar su expansión.

Debido principalmente a estas causas, el lenguaje de autor PILOT no ha tenido mucha aceptación ni difusión entre los docentes, y muchos de ellos siguen prefiriendo los lenguajes de programación, especialmente los dialectos estructurados del BASIC.

b. 2. Tipos de lenguajes de autor

En la actualidad se imponen principalmente dos tipos de lenguajes de autor, con interfaces muy diferentes, pero igualmente potentes y agradables: los lenguajes de estructuración gráfica, y los lenguajes con programación dirigida a objetos.

Los lenguajes de estructuración gráfica se programan mediante iconos que representan los principales eventos que ocurren durante la creación de un curso. Su principal atractivo y sencillez de uso reside en ofrecer una representación icónica del flujo y estructura del programa, un tratamiento de las variables y de la sintaxis mediante menús y la presencia de toda una serie de elementos «embellecedores» (editor gráfico potente orientado a objetos y complementado por otro editor de mapa de bits, editor de sonido, editor de animación, editor de control de videodisco, etcétera). Pertenecen a esta clase de lenguajes COURSE BUILDER, AUTHORWARE PROFESSIONAL, ICONAUTHOR, PROPY.

Por su parte, los lenguajes orientados a objetos son en su mayoría derivados de los gestores de hipertexto. Cuentan con la característica principal de poseer un estructuración nodal: la información se organiza y se alma-

cena en una red de *nodos* unidos por conexiones. La técnica consiste en crear zonas de pantalla que podríamos denominar «sensibles», como nudos que relacionan información situada a distintos niveles. Presionando sobre esta zona con el ratón gráfico aparece en pantalla la información ligada a esta zona, desapareciendo o no, o sólo en parte, la pantalla primera. Los gestores de hipertexto más conocidos son HYPERCARD y SUPERCARD (para entorno MACINTOSH), LINKWAY, HYPERPAD (para entorno WINDOWS), y GUIDE (para entorno WINDOWS y MACINTOSH).

2. Sistemas de autor

Con estos sistemas se pueden confeccionar programas para la EAO/ELAO de la manera más sencilla. Estas aplicaciones informáticas separan claramente lo que es el programa y los ficheros de datos. El docente se limita a introducir los datos necesarios para los tipos de ejercicios posibles que determina cada sistema de autor sin necesidad de conocimientos informáticos previos. Así el tiempo invertido es muy inferior y las posibles correcciones de los programas son mucho más sencillas de realizar.

Como contrapartida, el docente está supeditado a los tipos de ejercicios que permiten estos sistemas, además de la monotonía y falta de imaginación. Sirva como ejemplo el sistema de autor PROMPT. Éste sólo permite dos tipos de ejercicios: de eliminación (*'cloze'*, elisión parcial o total del texto), y de elección múltiple.

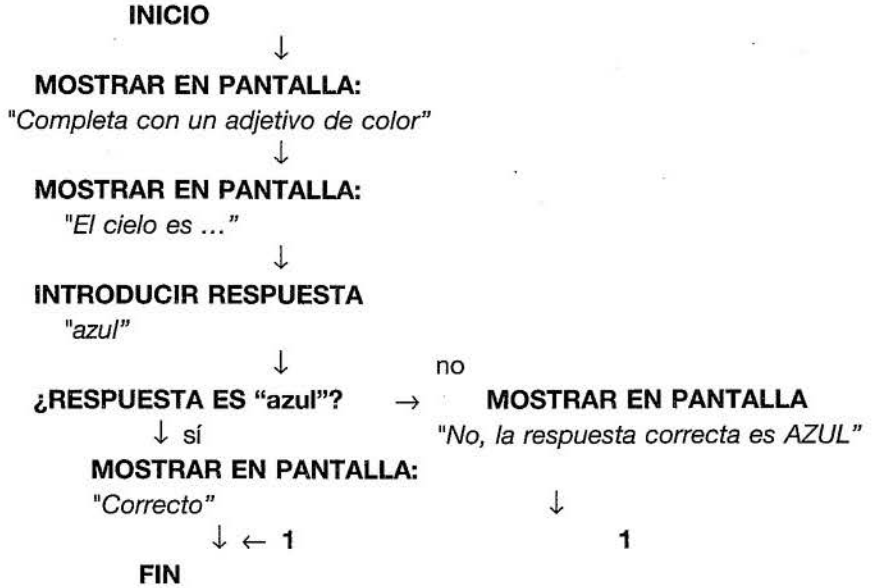
3. Ejemplos prácticos

Para ilustrar mejor todo lo dicho hasta aquí, vamos a proponer una serie de rutinas de utilidad para la creación de programas para la ELAO, así como mostrar algunos ejercicios creados mediante algunas de las aplicaciones ya mencionadas.

3.1. Ejemplos prácticos: lenguajes de programación

A pesar del título del epígrafe, los ejemplos que mostramos aquí están escritos con QBasic. No obstante pueden ser fácilmente reescritos en cualquier otro lenguaje, como C, por ejemplo.

Empezaremos con un sencillo diagrama de flujo que muestra el procedimiento lógico de un programa de diálogo o pregunta-respuesta:



Lo que traducido a QBasic equivaldría a:

```
PRINT "Completa con un adjetivo de color"
PRINT "El cielo es ..."
INPUT RESPUESTA$
IF RESPUESTA$ = "azul" THEN
  PRINT "Correcto"
ELSE
      PRINT "No, la respuesta correcta es AZUL"
END IF
END
```

Con los siguientes resultados (la respuesta del alumno en negrita y cursiva):

1) Completa con un adjetivo de color
   El cielo es ...
   azul
   Correcto

2) Completa con un adjetivo de color
   El cielo es ...
   amarillo
   No, la respuesta correcta es AZUL

El modelo anterior, a pesar de su sencillez, no varía sustancialmente de la gran mayoría de ejercicios interactivos que en la actualidad encontramos en el mercado. Adolece, por supuesto, de muchas sutilezas, que iremos presentando, pero la técnica básica no varía.

Podríamos añadir, por ejemplo, un pequeño contador de respuestas, que fuera contabilizando las respuestas correctas e incorrectas, para mostrarnos al final del ejercicio el porcentaje de aciertos sobre el total de respuestas, por ejemplo.

Pero hay cuestiones y aspectos más importantes y de mayor calado, especialmente, en lo que concierne a los programas para la ELAO, que tenemos que cuidar y prever. Destacaríamos: casos en los que hay más de una respuesta correcta posible; respuestas «abiertas»; control de mayúsculas y/o minúsculas en las respuestas; control de espacios entre palabras o espacios innecesarios que hayamos intercalado; grados de tolerancia en las respuestas (ausencia de acentos, erratas, etcétera) y suministro de pistas y ayudas.

3.1.1. Más de una respuesta correcta posible

Tomemos el ejemplo anterior e imaginemos que el alumno responde *celeste* en vez de *azul*, respuesta igualmente aceptable. Tal y como está diseñado el programa, la respuesta *celeste* sería incorrecta. Para subsanar esto, cabe la posibilidad de incluir una pequeña base de datos que contenga más de una respuesta correcta posible asociada a una misma pregunta. Sistemáticamente, el programa irá comparando la respuesta del discente con las posibles respuestas contenidas en la base de datos asociada, hasta encontrarla. En su defecto, una vez alcanzado el final de la base de datos y al no haber encontrado ninguna respuesta idéntica a la introducida por el alumno, mostrará un mensaje de error.

Nuestro programa anterior quedaría pues (la secuencia "*xxxfinalxxx*" indica el final de la base de datos):

```
PRINT "Completa con un adjetivo de color"
PRINT "El cielo es ..."
INPUT RESPUESTA$
uno: READ RESPOS$
IF RESPOS$ = "xxxfinalxxx" THEN
    PRINT "No, la respuesta correcta es AZUL o CELESTE"
ELSEIF RESPUESTA$ = RESPOS$ THEN
    PRINT "Correcto"
ELSE GOTO uno
END IF
DATA azul,celeste,xxxfinalxxx
END
```

La rutina procede de la forma siguiente:

1) Completa con un adjetivo de color
El cielo es ...
azul
Correcto

2) Completa con un adjetivo de color
El cielo es ...
celeste
Correcto

3) Completa con un adjetivo de color
El cielo es ...
amarillo
No, la respuesta correcta es AZUL

Esta es, por supuesto, solamente una de las posibles soluciones, pero creemos que es suficiente para ilustrar que es posible incluir más de una respuesta correcta.

Respuestas «abiertas»

Son muchos los ejercicios en los que las respuestas que el alumno puede proporcionar son demasiadas y a veces hasta imprevisibles. En ejercicios de comprensión lectora, por ejemplo, el discente tiene que contestar a preguntas «abiertas» del tipo *¿Quién descubrió América?*, en donde la solución parece clara, a priori, sin demasiadas complicaciones para procesar y automatizar, ya que la respuesta es *Colón* o *Cristóbal Colón*. No obstante, el alumno podrá igualmente contestar correctamente: *América la descubrió Colón*, *Colón descubrió América*, *Cristóbal Colón descubrió América*, entre otras muchas. De modo que una pregunta tan sencilla como la que acabamos de ver se complica debido a la gran variedad de respuestas que puede tener.

La solución que ofrecemos aquí será, como se verá más adelante, parcial y consiste en detectar si el discente utiliza en su respuesta una o varias palabras clave. En el caso anterior no cabe duda que la palabra clave para determinar si la respuesta es correcta o no sería: *Colón*. Así, dejando de lado la complejidad sintáctica de la respuesta del alumno, ésta, siempre que contenga la palabra clave, será evaluada como correcta. Para ello necesitamos una rutina que lea la cadena de caracteres, la respuesta del alumno, y verifique si la palabra clave se encuentra o no en ella (función INSTR).

```
PRINT "¿Quién descubrió América"
INPUT RESPUESTA$
IF INSTR(1, RESPUESTA$, "Colón") <> 0 THEN
   PRINT "Correcto"
ELSE
   PRINT "No, la respuesta correcta es COLÓN"
END IF
END
```

La ejecución tiene el resultado siguiente:

1) ¿Quién descubrió América?
   Fue Cristóbal Colón quien descubrió el Nuevo Mundo
   Correcto

2) ¿Quién descubrió América?
   Francis Drake
   No, la respuesta correcta es COLÓN

Tal como adelantábamos, la solución es solamente parcial, ya que admite respuestas del tipo: *Xcfcf Colón kdla akdla kalkd* o *Colón nunca descubrió América*, como correctas.

Control de mayúsculas y minúsculas en las respuestas

La comprobación automática de las respuestas que da el alumno con las respuestas previsibles contenidas en la base de datos tiene además varios problemas asociados. Uno de ellos es que *azul*, *Azul* y *AZUL* son para el ordenador respuestas completamente distintas, como lo pueden ser *azul* y *celeste*. Para subsanar esto podríamos, o bien incluir todas las respuestas posibles en la base de datos asociada (ver 3.1.1.) o bien, incluir una pequeña rutina que cambie las respuestas del alumno de mayúsculas a minúsculas, o viceversa, indistintamente de cómo haya redactado el discente la respuesta, y así adecuarla al formato de la base de datos de respuestas. Es decir, si en la base de datos las respuestas están todas redactadas en mayúsculas, tendremos que cambiar todo el input del alumno, de minúsculas a mayúsculas, o sustituir solamente las letras minúsculas por sus equivalentes mayúsculas —caso de *Azul*— o si los datos introducidos están ya en mayúsculas, dejarlos tal cual.

```
PRINT "Completa con un adjetivo de color"
PRINT "El cielo es ..."
INPUT RESPUESTA$
```

```
GOSUB mayúsculas
IF RESPUESTA$ = "AZUL" THEN
   PRINT "Correcto"
ELSE
      PRINT "No, la respuesta correcta es AZUL"
END IF
END
mayúsculas:
I = I + 1
IF I > LEN(RESPUESTA$) THEN
   RESPUESTA$ = RESP1$
   RETURN
ELSE
   LM$ = MID$(RESPUESTA$, I, 1)
   IF LM$ <= "Z" THEN
      GOTO uno
   ELSE
      LM$ = CHR$(ASC(LM$) - 32)
   END IF
END IF
uno:
RESP1$ = RESP1$ + LM$
GOTO mayúsculas
```

Esta rutina añadida verifica si los caracteres de la cadena están o no en mayúsculas (...LM$<="Z...) para, en caso afirmativo, simplemente anexionarlos a la cadena resultante (RESP1$=RESP1$+LM$), o, en caso negativo, reemplazarlo por su homólogo en mayúscula (LM$=CHR(ASC (LM$)−32) y seguidamente añadir el carácter a la cadena final (RESP1$=RESP1$+LM$). Los códigos ASCII correspondientes a las letras "*A-Z*", van del 65 al 90 y para las minúsculas "*a-z*", del 97 al 122, es decir que la diferencia entre el código ASCII de una mayúscula y su equivalente minúscula es de -32.

El resultado de esta rutina queda pues:

```
1) Completa con un adjetivo de color
   El cielo es ...
   azul
   Correcto

2) Completa con un adjetivo de color
   El cielo es ...
   AZUL
   Correcto
```

3) Completa con un adjetivo de color
El cielo es ...
Azul
Correcto

Control de espacios entre palabras

De igual modo que *azul* es distinto a *Azul* para un ordenador, *en casa* y *en casa*, también lo son. Aquí se debe a que en el primer caso, las palabras están separados por un único espacio mientras que en el segundo hay más de uno.

Quizá la solución más sencilla sea simplemente eliminar todos los espacios de las respuestas que da el alumno. En tal caso, habrá que adaptar previamente, también, las respuestas del ejercicio (las que constan en el fichero de respuestas) a tal formato.

```
PRINT "Transforma la siguiente frase en femenino"
PRINT "Mi hermano está en casa"
INPUT RESPUESTA$
GOSUB espacios
IF RESPUESTA$ = "Mihermanaestáencasa" THEN
   PRINT "Correcto"
ELSE
      PRINT "No, la respuesta correcta es MI HERMANA ESTÁ EN CASA"
END IF
END
espacios:
X = LEN(RESPUESTA$)
FOR I = 1 TO X
   RS$ = MID$(RESPUESTA$, I, 1)
   IF RS$ = " " THEN
      GOTO nxt
   ELSE
      R2$ = R2$ + RS$
   END IF
nxt:
NEXT
RESPUESTA$ = R2$
RETURN
```

La rutina se ejecuta con el resultado siguiente:

1) Transforma la siguiente frase en femenino
Mi hermano está en casa

Mi hermana está en casa
Correcto

2) Transforma la siguiente frase en femenino
Mi hermano está en casa
Mi hermana está en casa
Correcto

Grados de tolerancia en las respuestas

No siempre es aconsejable ni justo mostrar un mensaje de error, ya que, a veces, la respuesta del alumno se diferencia de la del modelo en un único carácter o en la ausencia de un acento, por ejemplo.

La rutina que a continuación proponemos compara la respuesta del discente con la que contiene el programa, mostrando un mensaje de advertencia, siempre que el error sea de sólo un carácter, y solicitando al aprendiz que subsane la errata.

```
inicio:
PRINT "Escribe el antónimo de PRIMERO"
INPUT RESPUESTA$
FOR I = 1 TO LEN(RESPUESTA$)
    IF MID$(RESPUESTA$, I, 1) <> MID$("último", I, 1) THEN
        ER = ER + 1
        X$ = X$ + "–"
    ELSE
        X$ = X$ + MID$(RESPUESTA$, I, 1)
    END IF
NEXT
IF ER > 1 THEN
    PRINT "No, la respuesta correcta es ÚLTIMO"
ELSE
    IF ER = 0 THEN
        PRINT "Correcto"
    ELSE
            PRINT "Corrige la errata:", X$
        X$ = ""
        ER = 0
        GOTO inicio
    END IF
END IF
END
```

Esta utilidad controla la respuesta del alumno, y solamente mostrará un mensaje de error cuando la respuesta difiera en más de un carácter (ER>1)

de la del modelo. En caso contrario se visualizará un mensaje de advertencia, solicitando que se vuelva a reescribir la respuesta.

La rutina se ejecutaría así:

1) Escribe el antónimo de PRIMERO
último
Correcto

2) Escribe el antónimo de PRIMERO
ultimo
Corrige la errata: -ltimo

3) Escribe el antónimo de PRIMERO
ultino
No, la respuesta correcta es ÚLTIMO

Pistas y ayudas

Es acosejable contar con rutinas de ayuda y/o pistas que reconduzcan al alumno para que éste pueda dar con la respuesta correcta.

La propuesta de ayuda que presentamos seguidamente se limita a mostrar al discente las dos primeras letras de la respuesta correcta. Para ello tendrá, previamente, que introducir la palabra clave *ayuda*.

```
inicio:
PRINT "Completa con un adjetivo de color"
PRINT "El cielo es ..."
INPUT RESPUESTA$
IF RESPUESTA$ = "azul" THEN
    PRINT "Correcto"
ELSE
    IF RESPUESTA$ = "ayuda" THEN
        PRINT "az..."
        GOTO inicio
    ELSE
                PRINT "No, la respuesta correcta es AZUL"
    END IF
END IF
END
```

En acción, tendríamos así:

1) Completa con un adjetivo de color
El cielo es ...
azul
Correcto

2) Completa con un adjetivo de color
El cielo es ...
negro
No, la respuesta correcta es AZUL

3) Completa con un adjetivo de color
El cielo es ...
ayuda
az...

En este apartado hemos intentado, de forma muy sucinta, acercar al lector a algunas rutinas básicas, que, creemos, deben tenerse en cuenta a la hora de confeccionar cualquier programa para la ELAO. Por su puesto, no son todas ni las únicas, pero la puesta en común de todas ellas puede resultar en aplicaciones didácticas realmente consistentes y fiables.

No queremos cerrar este apartado sin citar algunas otras utilidades interesantes a la hora de crear programas para la ELAO, tales como: la inclusión de algún mecanismo que altere, al azar, la secuenciación de las preguntas cada vez que se ejecuta el programa (lo que evitará la repetición del orden de las preguntas cada vez que se ejecuta un mismo programa, produciendo una sensación de estar trabajando cada vez con un programa distinto), y la separación del programa base de los ficheros de preguntas/respuestas. Esto tiene la ventaja adicional de que una vez diseñado, codificado y corregido el programa, simplemente basta con cambiar y confeccionar nuevos ficheros de preguntas/respuestas, para disponer de aplicaciones hechas a medida, según las necesidades del aula, alumnos o currículo, sin tener que preocuparnos de la codificación de un nuevo programa: los ficheros pueden ser fácilmente editados con cualquier editor/procesador de texto).

3.2. Ejemplos prácticos: lenguajes de autor

Ruipérez (1990:80) define los lenguajes de autor como lenguajes de procedimiento orientados a confeccionar aplicaciones en la EAO. Por ello, la elaboración de ejercicios interactivos mediante estas herramientas resulta mucho más fácil, debido a que estos lenguajes están provistos de instrucciones específicas para este fin.

A continuación presentamos un ejercicio interactivo realizado mediante PILOT, en donde se pueden apreciar estas virtudes. Cada línea del programa viene comentada. Asímismo, están resaltadas en otro color las instrucciones específicas del lenguaje de autor PILOT (a pie de pantalla se simula el funcionamiento de la aplicación creada; ver ejercicios del tipo '*cloze*', fig. 1).

Otro lenguaje de autor muy difundido en la actualidad entre los docentes que utilizan la ELAO en el aula es CALIS, de la Universidad de Duke. Su funcionamiento es sencillo y transparente. A modo de ejemplo, basta con editar un texto en DOS y colocar una barra anterior y otra posterior a la palabra que deseamos mutilar del texto, y tras compilar el texto mediante el módulo de autor, ya podemos ejecutar la actividad (figs. 2 y 3).

CALIS permite la creación de dos tipos de ejercicios bastante estandarizados: técnica *cloze* y comprensión lectora (pregunta-respuesta).

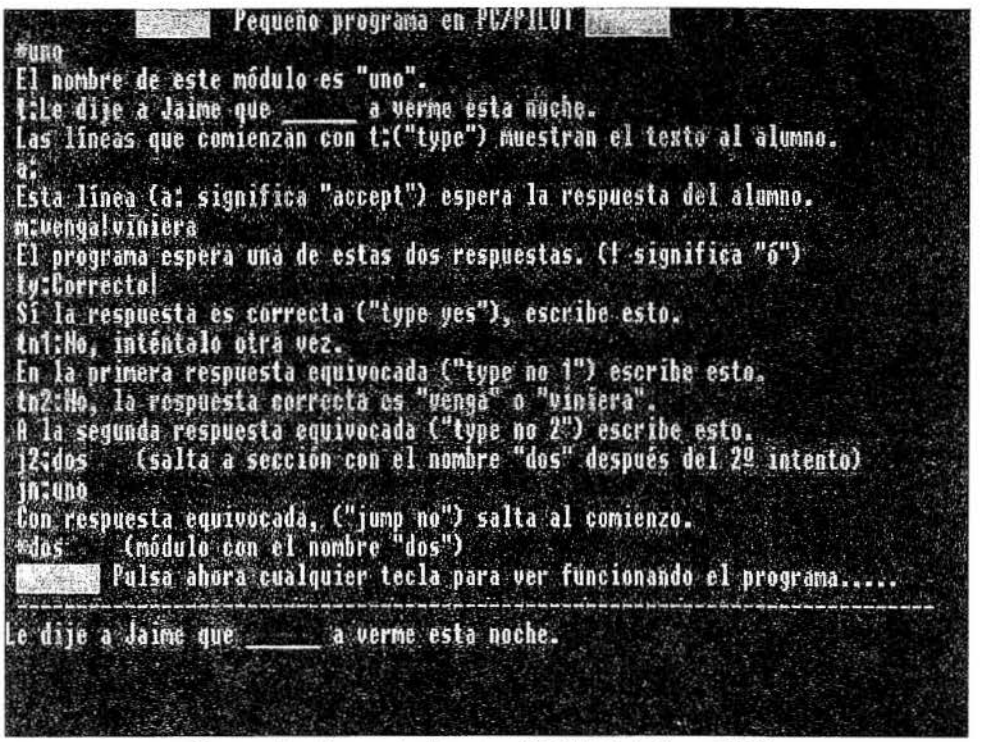


FIGURA 1.—*Lenguaje de autor PILOT* (Ruipérez, 1990).

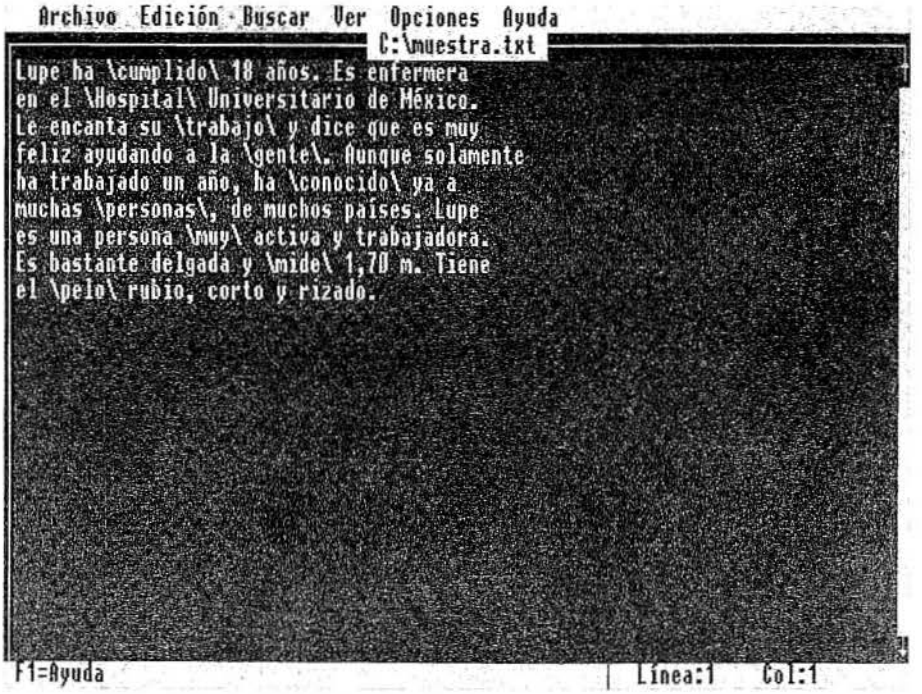


FIGURA 2.—*Texto base.*

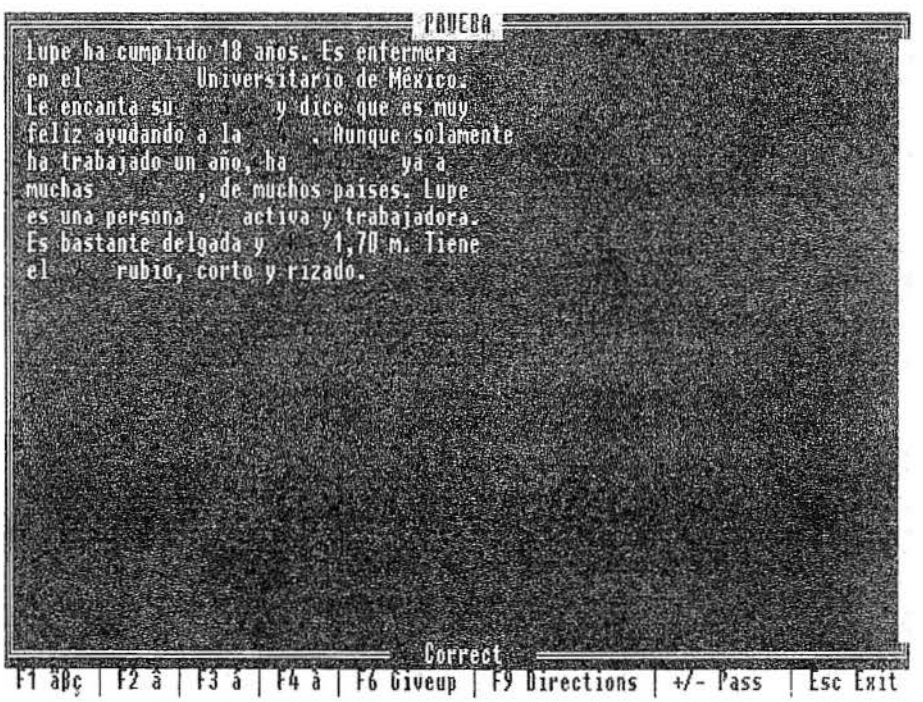


FIGURA 3.—*Formato* cloze *en CALIS del texto base.*

3.3. Ejemplos prácticos: sistemas de autor

Si la creación de ejercicios mediante los lenguajes de autor resulta fácil, los sistemas de autor la hacen aún más sencilla. En estas aplicaciones el docente se limita a introducir únicamente los datos, lo que reduce notablemente el tiempo de elaboración de aplicaciones para la ELAO. No obstante, tienen una desventaja y es que el docente está totalmente supeditado a la variedad de ejercicios que el sistema presenta: no ofrecen creatividad.

Para que el lector pueda hacerse una idea aproximada de lo que son los sistemas de autor, presentamos a continuación PROMPT, desarrollado por las Universidades canadienses de Calgary y Guelph. PROMPT permite la creación de ejercicios de elisión y de elección múltiple (figs. 4 y 5).

El menú principal ofrece al autor la posibilidad de introducir un texto base, elaborar un pequeño diccionario bilingüe (a la lengua materna del discente y que este puede consultar siempre que no comprenda una palabra del texto), introducir las preguntas relacionadas con el texto (tipo elección múltiple), señalar cuál es la respuesta correcta, además de añadir comentarios relacionados con las respuestas que dé el alumno (aciertos o fallos), probar la aplicación elaborada antes de dejarla en manos de los alumnos y finalmente grabar la aplicación creada para ser ya ejecutada por los discentes.

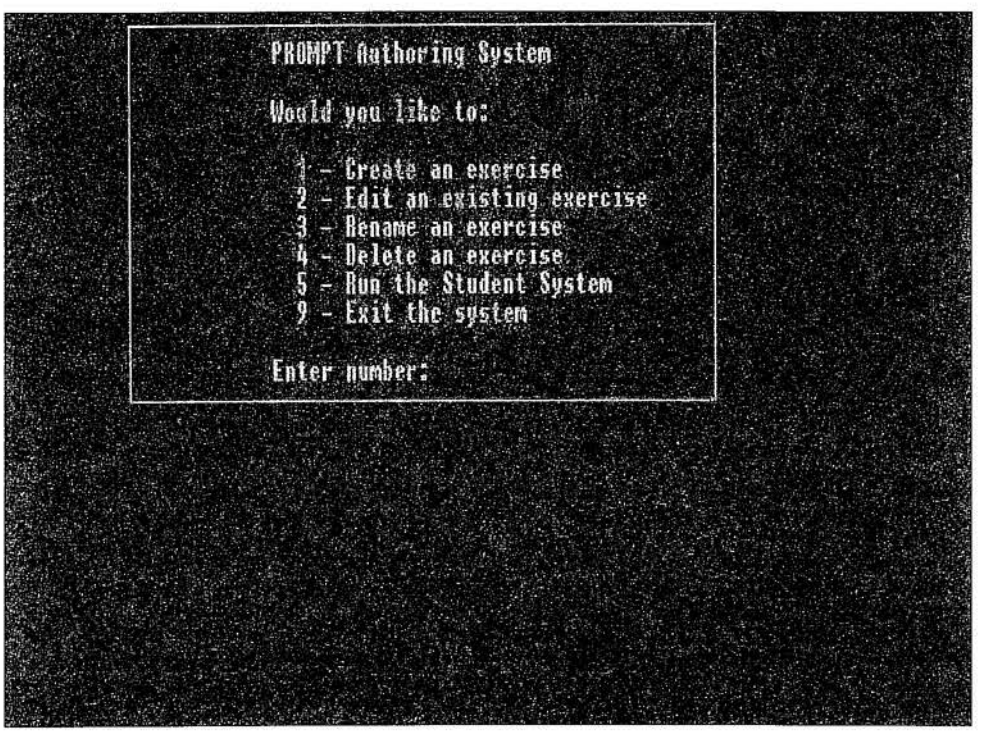


FIGURA 4.—*Menú inicial de PROMPT.*

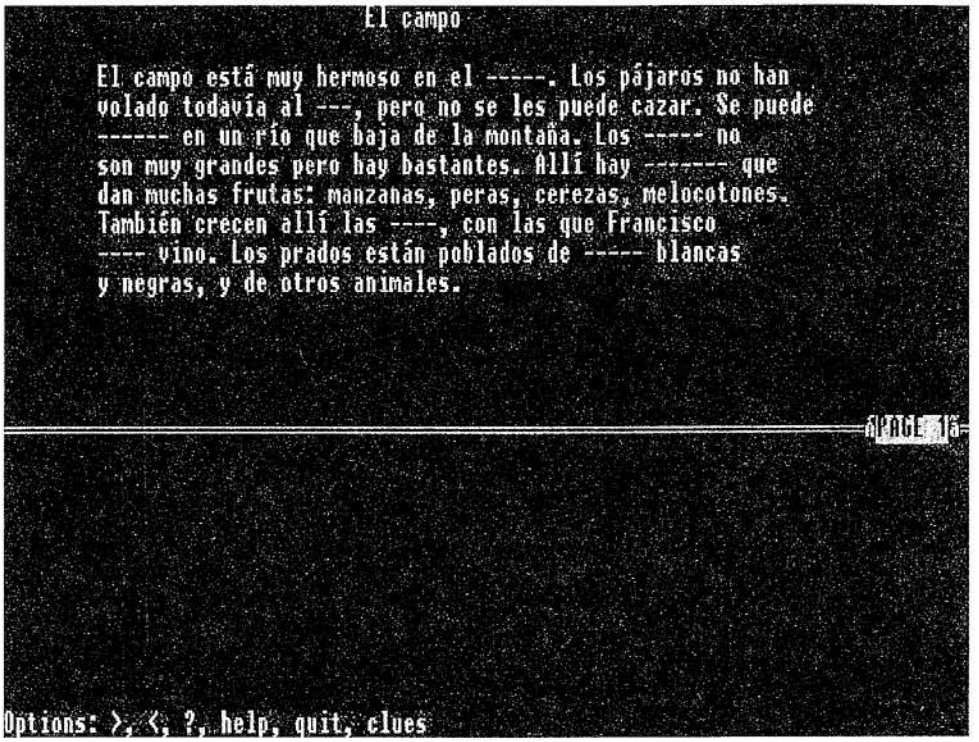


FIGURA 5.—*Ejercicio de elisión creado con PROMPT* (Ruipérez, 1990).

4. Conclusiones

En esta aproximación al software de programación para la ELAO, hemos repasado, primero, los lenguajes de programación o de procedimiento, especialmente QBasic, mediante rutinas y programas cortos de especial utilidad para la ELAO, para seguir con los lenguajes de autor y acabar con los sistemas de autor.

La complejidad de los lenguajes de procedimiento es mayor y requiere más conocimiento de programación para lograr aplicaciones de usuario aceptables para la ELAO. Algo que sin embargo resulta mucho más fácil con los lenguajes de autor, tipo PILOT y/o CALIS, por tratarse de lenguajes de programación con instrucciones específicas para la creación de aplicaciones de EAO/ELAO, y no digamos ya con los sistemas o programas de autor.

No obstante, este decremento en la complejidad de manejo para la confección de aplicaciones específicas para la EAO/ELAO tiene también un coste o contrapartida, y es que a mayor facilidad corresponde menor flexibilidad. Es decir, hay una relación inversamente proporcional entre la com-

plejidad del software de programación y lo que éste permite al usuario. Un lenguaje de programación permite crear aplicaciones altamente complejas, variadas y con una multitud de rutinas y utilidades impensables en los lenguajes o sistemas de autor, aunque, eso sí, exigen mucho más: amplios conocimientos de programación. De ahí que cuanto más bajemos en la escala de complejidad informática, lenguajes de programación, lenguajes de autor y sistemas de autor, también estamos acotando las posibilidades de variedad y tipología de actividades, hasta llegar al punto de estar completamente a expensas de las reducidas facilidades y opciones del software. El docente tendrá, por lo tanto, que encontrar un equilibrio entre lo que sabe y lo que espera conseguir, para, así, no crearse ni crear falsas expectativas: algo que lamentablemente está perjudicando seriamente la expansión de los nuevos medios tecnológicos en el aula.

Bibliografía

ABDULAZIZ, M., SMALZER, W. y ABDULAZIZ, H. (1985), *The Computer Book. Programming and Language Skills for Students of ESL*, New Jersey, Prentice-Hall.

HIGGINS, J. (1987), «The Coming of the Clone», *System*, 15, 2: 224-225.

JONES, C. y FORTESCUE, S. (1987), *Using Computers in the Language Classroom*, Essex, Longman.

KENNING, M. J. y KENNING, M.-M. (1983), *An Introduction to Computer Assisted Language Teaching*, Oxford, Oxford University Press.

LEUSCHNER, B. (1987), «Comal's the Thing - Letter to John Higgins», *System*, 15, 3: 373-376.

RUIPÉREZ, G. (1990), *Introducción a la enseñanza de lenguas asistida por ordenador*, Madrid, Universidad Nacional de Educación a Distancia.

SANDERS, R. H. (1985), «PILOT, SNOBOL, and LOGO as Computing Tools for Foreign-Language Instruction», *Calico Journal*, 3:41-47.

UNDERWOOD, J. (1984), *Linguistics, Computers and the Language Teacher. A Communicative Approach*, Rowley, Mass., Newbury House.