Aplicación de técnicas de aprendizaje activo a la enseñanza de la programación de ordenadores

Eva Gibaja[1][0000−0002−0184−8789], María Luque[1][0000−0001−7735−8340], and Amelia Zafra[1][0000−0003−3868−6143]

Universidad de Córdoba, Departamento de Informática y Análisis Numérico, Córdoba, España
{egibaja, mluque, azafra}@uco.es

Resumen El alumno tiende a ser un elemento pasivo en el proceso de enseñanza-aprendizaje, limitándose a recibir lecciones magistrales. En este trabajo realizamos una propuesta para el programa de prácticas de la materia Programación, impartida en el Grado en Ingeniería Informática de la Universidad de Córdoba, enfocado a que el alumno se implique en su propio proceso de aprendizaje experimentando con problemas de complejidad media, que le acercan al mundo real. La propuesta se basa en el aprendizaje activo y pretende que el alumno adquiera las competencias de la materia mediante la experimentación con casos prácticos.

Keywords: Aprendizaje activo · Programación de ordenadores · Ingeniería informática.

1. Introducción

La enseñanza tradicional es un modelo aún presente en las aulas universitarias. Está basado en un concepto de transmisión de la enseñanza donde el profesorado imparte clases magistrales en las que intenta enseñar a su alumnado los aspectos teóricos de la materia. Desde el punto de vista de los alumnos, supone un aprendizaje fundamentalmente memorístico-conceptual, ya que suelen ser agentes pasivos en su proceso de aprendizaje en el aula. Además, al menos en sus aspectos teóricos, supone un acercamiento muy superficial a la forma en que se construye el conocimiento científico. A pesar de que, en una primera lectura, podría parecer que este tipo de enseñanza es descartable por completo, esta afirmación no es del todo acertada. Primero, porque una clase magistral sobre conocimientos específicos, impartida por el profesorado, es un excelente recurso para que los alumnos conozcan los aspectos centrales de cada tema. Por otro lado, en aulas masificadas resulta imposible poder llevar a cabo otro método de enseñanza.

Sin embargo, una enseñanza basada exclusivamente en clases magistrales puede llegar a considerarse, en cierto modo, obsoleta, dado que los estudiantes disponen de un gran número de fuentes de información alternativas a la mera transmisión de conocimientos. Por tanto, en la medida de lo posible, hay que incorporar otras actividades en las que el alumnado sea agente activo de su proceso

de aprendizaje. Como docentes debemos facilitar que los estudiantes desarrollen sus aptitudes para aprender y producir conocimiento, comprometiéndole de una manera activa en los complejos procesos de búsqueda, comprensión e implicación en su futuro laboral y personal [11]. Es fundamental la motivación y enseñarles a que quieran y puedan continuar aprendiendo al abandonar las aulas. Así, los alumnos deben *aprender a aprender*, sobre todo, en un campo como la Informática, cuyo desarrollo es vertiginoso. Esto implica que, cuando los estudiantes que ahora están en el aula se conviertan en profesionales, es probable que los instrumentos de que dispongan en el ejercicio de su actividad y las técnicas que empleen sean diferentes a las que se les haya podido transmitir. Lo que ahora importa, no es tanto poseer una información determinada, sino haber adquirido la capacidad para descubrir y saber encontrar esa información. No vale cualquier tipo de enseñanza, sino aquella que facilita y estimula el aprendizaje de las competencias humanas consideradas valiosas [10].

El término *aprendizaje activo* [2], hace referencia a un conjunto de métodos de aprendizaje que enfocan la responsabilidad del aprendizaje en los estudiantes. Estos métodos pretenden involucrar al estudiante en tareas en las que, además de actuar, reflexione sobre la acción que desarrolla [8]. Todos ellos incluyen, entre otros aspectos: a) la implicación de los estudiantes en algo más que la escucha pasiva, b) el énfasis en el desarrollo de habilidades en los estudiantes y c) la realización de tareas que requieren procesos de pensamiento de cierta complejidad [9]. El aprendizaje activo hace referencia a que el estudiante se identifica con un proceso en el que investiga. No es un proceso en el que se busca una culminación concreta, sino que es el propio estudiante quien va a volver a aprender y reconstruir como herramienta y estrategia a lo largo del aprendizaje. De ahí, que se necesite un sujeto que indague y sienta curiosidad por lo que es desconocido (*learning by doing* [6]). El papel que debe desempeñar el alumno es un papel activo, autónomo, estratégico, reflexivo, cooperativo y responsable [7]. Esta estrategia ha sido utilizada con éxito tanto en otros estudios de Grado ligados a la ingeniería [14] [5], como de Ingeniería Informática [13] [4] [3] [12].

De forma más particular, en este trabajo se proponen un conjunto de actividades inspiradas en el aprendizaje activo para el desarrollo del programa práctico de la materia *Programación* impartida en el título de Grado en Ingeniería Informática de la Universidad de Córdoba. Esta materia abarca dos asignaturas denominadas *Introducción a la Programación (IP)* y *Metodología de la Programación (MP)*.

La necesidad de aplicar técnicas de aprendizaje activo surgió al detectar un bajo rendimiento de los estudiantes al finalizar el curso. A la hora de preparar los exámenes, es común la falta de consultas bibliográficas y dejar todo el estudio para el final, lo que implica una baja asimilación de los conceptos de la asignatura, una escasa asociación entre los contenidos de ésta y la dificultad de llevar los conceptos de la teoría a la práctica. Por ello, mediante el uso de aprendizaje activo, pretendemos plantear a los alumnos el desarrollo de tareas de complejidad media, basadas en el mundo real, que les permitan reflexionar

sobre los contenidos teóricos de la materia y aplicarlos en la resolución de dicha tarea.

En la Sección 2 se presenta una introducción al contexto de la materia Programación dentro del plan de estudios. A continuación, las Secciones 3 y 4 describen las propuestas presentadas y, finalmente, la Sección 5 presenta un conjunto de conclusiones.

2. Contexto académico

El título de Grado en Ingeniería Informática, en el ámbito internacional se corresponde con las titulaciones denominadas de forma genérica *Computer Science*, *Computer Engineering*, *Information Systems*, *Information Technology* y *Software Engineering*, según las recomendaciones de ACM [1].

En la Universidad de Córdoba, el objetivo general del título es la formación de profesionales en el ámbito de la Ingeniería Técnica en Informática con una base amplia y generalista de conocimiento en Ingeniería Informática y una formación que garantice la adquisición de los conocimientos específicos de las especialidades de Ingeniería del Software, Ingeniería de Computadores y Computación. Además, el título proporciona la capacidad de aplicación de dichos conocimientos a las actividades propias de la profesión del Ingeniero Técnico en Informática. El Grado consta de 4 cursos, el primero de ellos constituye el *módulo de formación básica* con contenidos de Informática, Matemáticas, Física y Empresa. Dentro de este módulo se encuentra la materia *Programación*, que abarca dos asignaturas: *Introducción a la Programación* y *Metodología de la Programación*.

3. Programa práctico para IP

Introducción a la Programación es impartida en el primer cuatrimestre con un total de 6 ECTS. La Tabla 1 muestra un resumen de las competencias y contenidos de la asignatura. El lenguaje de programación utilizado es C. Para el desarrollo del programa práctico, se proponen dos actividades. La primera de las actividades propuestas está basada en el álgebra lineal que hay detrás de los buscadores de internet y la segunda consiste en el desarrollo del juego de la vida. La Tabla 2 muestra un esquema de los contenidos teóricos de la asignatura cubiertos por cada una de las propuestas de prácticas presentadas.

3.1. Actividad 1: Clasificación de páginas Web

Supongamos que todas las páginas web accesibles desde cualquier navegador se pueden clasificar en cinco grupos diferenciados, en función del contenido de éstas; y que se tiene la matriz de probabilidad de la ecuación (1), donde un elemento $M(i, j)$, indica la probabilidad de pasar de una página del grupo j, a una página del grupo i. Por ejemplo, el elemento $M(2, 3)$ vale 0,20, lo que indica que la probabilidad de pasar de una página del grupo 3 a otra del grupo 2 es de

Tabla 1: Contenidos impartidos en la asignatura IP

Competencias
CB2: Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas en el campo de la Ingeniería Informática
CB4: Que los estudiantes puedan transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado
CEB3: Capacidad para comprender y dominar los conceptos básicos de matemática discreta, lógica, algorítmica y complejidad computacional, y su aplicación para la resolución de problemas propios de la ingeniería
CEB4: Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería
CEB5: Conocimiento de la estructura, organización, funcionamiento e interconexión de los sistemas informáticos, los fundamentos de su programación, y su aplicación para la resolución de problemas propios de la ingeniería
Contenidos
Introducción a la informática
Algoritmos y Programas
Fundamentos de programación: Tipos de datos, operadores, expresiones, estructuras de control
Autodocumentación y estilos de programación
Tipos de datos compuestos: estructuras, arrays, cadenas
Programación estructurada y modular

Tabla 2: Contenidos de IP practicados en cada propuesta

	Algorítmica	Tipos datos	Operadores expresiones	Estructuras control	Documentación	*Arrays* cadenas	Funciones
Páginas Web	✓	✓	✓	✓	✓	✓	✓
Juego vida	✓	✓	✓	✓	✓	✓	✓

0,20. La suma de los elementos de cualquier columna ha de ser 1 (a este tipo de matrices se les denomina *estocásticas*).

$$\mathbf{M} = \begin{pmatrix} 0{,}09 & 0{,}08 & 0{,}10 & 0{,}16 & 0{,}12 \\ 0{,}11 & 0{,}12 & 0{,}20 & 0{,}04 & 0{,}08 \\ 0{,}09 & 0{,}08 & 0{,}09 & 0{,}10 & 0{,}15 \\ 0{,}20 & 0{,}15 & 0{,}19 & 0{,}15 & 0{,}10 \\ 0{,}51 & 0{,}57 & 0{,}42 & 0{,}55 & 0{,}55 \end{pmatrix} \tag{1}$$

Para representar la página en la que estamos ubicados en un momento dado, se utiliza un vector de estados, de cinco elementos, en el cual todos los elementos son 0, excepto el elemento correspondiente al grupo al que pertenece la página en la que estamos ubicados, que vale 1. Por ejemplo, si estamos inicialmente en una página perteneciente al grupo 2, el vector de estados $\mathbf{v_0}$ será:

$$\mathbf{v_0} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2}$$

Si, por ejemplo, estamos en una página del grupo 2, y accedemos a otra página, podemos obtener un nuevo vector $\mathbf{v_1}$ que nos indica la probabilidad de

estar en una página perteneciente a cualquier grupo, de la siguiente forma:

$$\mathbf{v_1} = \mathbf{M} * \mathbf{v_0} = \begin{pmatrix} 0{,}08 \\ 0{,}12 \\ 0{,}08 \\ 0{,}15 \\ 0{,}57 \end{pmatrix} \tag{3}$$

Generalizando, después de cambiar de página n veces, se podría obtener el vector de estados $\mathbf{v_n}$, donde $\mathbf{M}^n$ es la potencia n-ésima de la matriz $\mathbf{M}$, como:

$$\mathbf{v_n} = \mathbf{M}^n * \mathbf{v_0} \tag{4}$$

Propuesta de trabajo

Implementar un programa en C, en el que se obtenga la potencia n-ésima de la matriz $\mathbf{M}$, así como el vector de estados $\mathbf{v_n}$, partiendo de una página inicial dada por el usuario (se indicará a qué grupo pertenece, valor comprendido entre 1 y 5). La matriz se inicializará en el momento de su declaración con los valores de la matriz de ejemplo (1). Se implementará una función que multiplique dos matrices, de cualquier dimensión (siempre que se puedan multiplicar), y otra función, que invocando a la anterior, obtenga la potencia n-ésima de una matriz cuadrada. También se implementará una función para visualizar la matriz $\mathbf{M}^n$.

3.2. Actividad 2: El juego de la vida

Consideremos una población de K insectos en una matriz de dimensiones $M \times N$, de modo que en cada celda de la matriz hay, como máximo, un insecto. Por lo tanto, cada insecto tiene, como máximo, 8 insectos vecinos. La población está en desarrollo continuamente debido a los nacimientos y muertes que se producen siguiendo las siguientes reglas de evolución:

- Aquellos insectos que tienen 0, 1, 4, 5, 6, 7 u 8 vecinos mueren.
- Los insectos que tienen 2 o 3 vecinos sobreviven.
- En las celdas vacías con 2 o 3 vecinos nace un nuevo insecto.
- Los insectos que nacen o mueren no afectan a las reglas hasta que se ha completado un ciclo evolutivo, entendiendo por éste un ciclo en el que se ha decidido la supervivencia o muerte de los insectos (vivos al comenzar el ciclo) de acuerdo con las reglas mencionadas. Así, al calcular los nuevos estados de las celdas de la matriz las modificaciones se deben hacer sobre una matriz auxiliar que, finalmente, se copiará en la matriz original.

Propuesta de trabajo

Escribir una función *Evolucion* que simule la evolución de la población y que:

1. Reciba como entrada los enteros positivos K, N y M, un *array* de estructuras con las coordenadas x e y de las celdas en las que se encuentran los K insectos iniciales de la población y un entero positivo L que indica cuántos ciclos se van a simular.

2. Muestre por pantalla el estado inicial de la población, marcando con un asterisco (*) las celdas ocupadas por insectos.
3. Simule la evolución de la población durante los L ciclos evolutivos y represente en la pantalla el estado de la población en cada ciclo representándolo como en el siguiente ejemplo:
   - Con una matriz de 10×10, 3 insectos colocados inicialmente en las posiciones $(0,0)$, $(0,1)$ y $(1,0)$ y 3 ciclos de evolución tendríamos la configuración inicial de la Tabla 3a y las configuraciones de las Tablas 3b, 3c y 3d para la primera, segunda y tercera evolución respectivamente.

Tabla 3: Evolución de la población

(a) Configuración incial

```
0 1 2 3 4 5 6 7 8 9
* *
*
```

(b) Primera iteración

```
0 1 2 3 4 5 6 7 8 9
* *               *
* *               *
```

(c) Segunda iteración

```
0 1 2 3 4 5 6 7 8 9
  *               *
  * *             * *
* *                 *
```

(d) Tercera iteración

```
0 1 2 3 4 5 6 7 8 9
    *       * *
  * *     * *
* *         * *
```

Para ello serán necesarias, al menos, las siguientes funciones auxiliares:

- *Pinta*. que recibe la matriz con los insectos y los datos necesarios para pintarla por pantalla utilizando asteriscos, tal y como muestra la Tabla 3d.
- *CuentaVecinos*, que recibe como parámetro la matriz de insectos, las coordenadas de una posición dentro de la matriz y los datos necesarios para devolver el número de insectos vecinos a esa posición. Hay que considerar la matriz circular, es decir, si tenemos una matriz de 4×4 y estamos inspeccionando la casilla $(3,3)$ (esquina inferior derecha), los vecinos serán las casillas $(2,2)$, $(2,3)$, $(2,0)$, $(3,2)$, $(3,0)$, $(0,2)$, $(0,3)$ y $(0,0)$ (véase Tabla 4).

Tabla 4: Vecinos de la casilla $(3, 3)$

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	**3,3**

- *ProximaGeneracion*, que recibe la matriz de insectos y los datos necesarios para devolver, en la misma matriz, el estado de la matriz después de una generación donde se han aplicado las reglas del juego de la vida (internamente, la función utilizará una matriz auxiliar).

La práctica deberá completarse con la implementación de un programa completo que haga uso de la función *Evolución* para la simulación de una población de insectos.

4. Programa práctico para MP

Metodología de la Programación es impartida en el segundo cuatrimestre con un total de 6 ECTS. La Tabla 5 muestra un resumen de las competencias y contenidos de la asignatura. El lenguaje de programación utilizado es C. Para el desarrollo del programa práctico, se proponen tres actividades. La primera y la segunda propuestas son una aplicación de los contenidos de la asignatura a un problema de gestión de datos complejos. Finalmente, la tercera consiste en un problema de gestión de colas para ventanillas de atención al público. La Tabla 6 muestra un esquema de los contenidos teóricos de la asignatura cubiertos por cada una de las propuestas de prácticas presentadas.

Tabla 5: Contenidos impartidos en la asignatura MP

Competencias
CB4: Que los estudiantes puedan transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado
CU2: Conocer y perfeccionar el nivel de usuario en el ámbito de las TIC
CEB4: Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería
CEB5: Conocimiento de la estructura, organización, funcionamiento e interconexión de los sistemas informáticos, los fundamentos de su programación, y su aplicación para la resolución de problemas propios de la ingeniería
Contenidos
Punteros
Ficheros: Texto y Binarios
Estructura de un programa en tiempo de ejecución
Memoria dinámica
Recursividad
Estructuras lineales dinámicas de datos: Listas, Pilas, Colas
Algoritmos básicos de búsqueda y ordenación y su complejidad algorítmica
Aspectos metodológicos de la programación: Documentación y Pruebas
Herramientas: generación automática de proyectos, documentación, bibliotecas, depuradores

Tabla 6: Contenidos de MP practicados en cada propuesta

	Memoria	Ficheros	Listas	Pilas colas	Punteros	Make	Argumentos *main*	Bibliot.	Orden
Conejos	√	√	√		√	√	√	√	√
Polígonos	√	√	√	√	√	√	√	√	√
Colas	√		√	√	√		√		√

4.1. Actividad 1: Gestión de una granja de conejos

Una empresa ganadera, dedicada a la cría y venta de conejos, dispone de un fichero binario, denominado *conejos.bin*, con los datos de los conejos criados en el año 2019. Los registros del fichero poseen la siguiente estructura:

- *Código* del conejo (cadena de 6 caracteres).
- *Edad* del conejo (en días) de tipo entero.
- *Peso* del conejo (en gramos) de tipo *double*.

La empresa ha detectado, que debido al peso, no todos los conejos pueden ser vendidos. Así, los conejos que no sobrepasan un determinado peso, no son deseados por los consumidores, y los conejos que sobrepasan un determinado peso tienen un exceso de grasa y no son muy aptos para el consumo.

Propuesta de trabajo

- Implementar un programa en C, que realice secuencialmente las siguientes operaciones:
  1. Paso de los registros del fichero a una lista doblemente enlazada.
  2. Cálculo del peso medio, m, de los conejos y de la desviación típica de los pesos, s, utilizando la lista doblemente enlazada.
  3. Eliminación de la lista doble de aquellos conejos cuyo peso sea inferior a $m - 1{,}5 * s$, y cuyo peso sea superior a $m + 1{,}5 * s$.
  4. Paso de los elementos de la lista resultante a un vector dinámico.
  5. Ordenación de los elementos del vector en orden decreciente del peso.
  6. Almacenamiento de los elementos del vector en un fichero de texto.
- Se usarán los siguientes archivos:
  - Un archivo *principal.c*, que contendrá la función *main*.
  - Los archivos *ficheros.c*, *listas.c*, y *vectores.c*, que contendrán las funciones correspondientes a ficheros, listas y vectores respectivamente. Cada uno de estos tres archivos llevará su correspondiente archivo de cabecera (*.h*).
  - Los tipos de datos para la lista y el vector, se implementarán en un archivo de cabecera denominado *tipos.h*.
- Se creará un fichero *makefile* con las siguientes caracteríticas,
  - Se creará una biblioteca, *conejos.lib* con los archivos *ficheros.o*, *listas.o*, y *vectores.o*.
  - El ejecutable se creará a partir de la biblioteca y el archivo *principal.o*.
- Los nombres de los archivos de entrada y salida (*conejos.bin* y *conejosSeleccionados.txt*) se introducirán en línea de órdenes.

4.2. Actividad 2: Gestión de un fichero con polígonos

Se tiene un fichero de texto que almacena la información necesaria para representar polígonos en el plano cartesiano. Cada línea del fichero representa un polígono de la siguiente forma:

- Un número entero, n, seguido de un espacio en blanco, que indica el número de lados del polígono.
- A continuación, $2 * n$ enteros separados por espacios en blanco y que representan las coordenadas (x, y) de cada punto del polígono (véase Archivo 1.1).

Archivo 1.1: Ejemplo de fichero de polígonos

```
9 185 4 142 1 113 12 71 74 75 115 97 149 167 172 183 169 231 132
4 217 4 192 33 201 37 221 28
2 10 297 8 308
8 200 315 170 297 145 298 108 352 120 381 126 387 187 392 212 348
9 195 0 194 0 190 2 188 4 186 12 198 21 198 21 201 21 205 17
```

Propuesta de trabajo

Realizar un programa que implemente una serie de funciones para procesar y visualizar listas de polígonos. El programa realizará las siguientes operaciones:

1. Leer de un fichero de texto los polígonos y almacenarlos en dos listas (Figura 1):
   - *listaTriangulos*: contendrá los polígonos que tengan tres lados. El resultado se mostrará por pantalla.
   - *listaGeneral*: contendrá los polígonos que no tengan tres lados. En este caso se realizará una inserción ordenada por el número de lados, que podrá ser ascendente o descendente. Este apartado se implementará utilizando punteros a funciones. El resultado se mostrará por pantalla.
2. Para la lista *listaTriángulos*:
   - Calcular el perímetro de cada uno de los elementos de la lista.
   - Borrar de la lista aquellos polígonos cuyo perímetro sea inferior a 400. El resultado se mostrará por pantalla.
3. Mostrar los polígonos de *listaGeneral* de N lados (N se leerá desde el teclado).

Requisitos y recomendaciones:

- El programa se ejecutará secuencialmente, sin menús.
- El programa se estructurará en 7 ficheros:
  1. *main.c*: contiene el programa principal, *main*.
  2. *listas.c*: funciones relacionadas con las listas.
  3. *listas.h*: fichero de cabecera para *listas.c*.
  4. *ficheros.c*: funciones relacionadas con la lectura de fichero de polígonos.
  5. *ficheros.h*: fichero de cabecera para *ficheros.c*.
  6. *miscelanea.c*: para el resto de las funciones.
  7. *miscelanea.h*: fichero de cabecera para *miscelanea.c*.

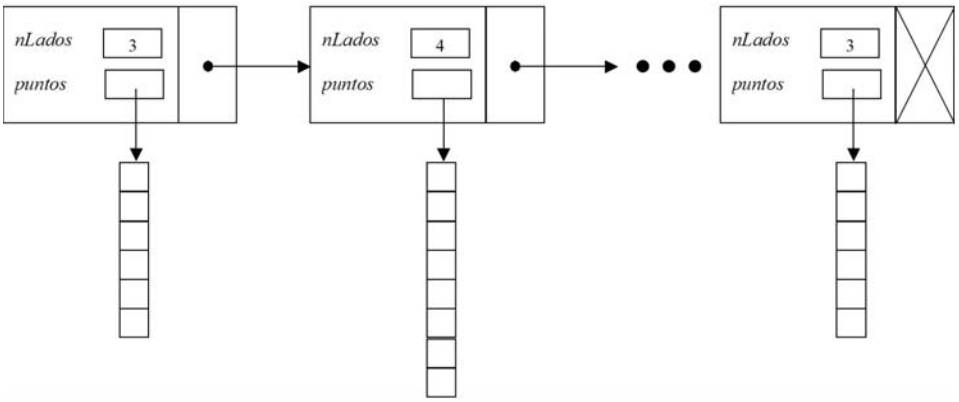

Figura 1: Ejemplo de lista de polígonos

- Se utilizará un fichero *makefile*.
  - Se desarrollará una biblioteca a partir de los ficheros *listas.o*, *ficheros.o* y *miscelanea.o*.
  - El ejecutable se creará a partir de *main.o* y de la biblioteca.
- El nombre del fichero de polígonos y el orden de inserción en *listaGeneral* (ascendente o descendente) se introducirá como un parámetro en la llamada al programa. Es decir, si el programa se llama *poligonos*, la llamada sería:
  - *poligonos ficheroPoligonos.txt a (orden ascendente)*
  - *poligonos ficheroPoligonos.txt d (orden descendente)*
- Se implementará una función para comprobar que la llamada al programa se ha realizado correctamente. Esto es, que el número de argumentos es el correcto y que el fichero de polígonos existe. En caso contrario, terminará la ejecución del programa y se mostrará al usuario un mensaje de error:
  - *Error: en el numero de argumentos. Sintaxis correcta: poligonos ficheroPoligonos.txt a | d*
  - *Error: el fichero de polígonos <ficheroPoligonos.txt> no existe*

4.3. Actividad 3: Gestión de colas

Normalmente cuando accedemos a un servicio atendido por varias ventanillas (estación de tren, estación de autobuses, etc.), nos colocamos en la ventanilla cuya cola tiene menos personas esperando y después suele suceder que esa cola es atendida más lentamente que las demás porque hay uno o varios usuarios que requieren más tiempo de atención que los demás.

Propuesta de trabajo

Simular el funcionamiento de tres ventanillas (A, B, C), siguiendo tres estrategias distintas a la hora de seleccionar la cola en la cual se ubica el usuario, para evaluar cual es la mejor y ver las diferencias entre ellas. A continuación se detallan los pasos a seguir:

1. Introducir por teclado el número de personas, n, que vamos a utilizar para simular el funcionamiento.
2. Reservar un vector dinámico de tipo entero de n elementos, en el cual cada elemento será un número aleatorio comprendido entre 1 y 5. Este número será el tiempo de atención que requiere cada usuario.
3. Recorrer el vector desde el elemento 1 hasta el n, asignando cada elemento a la cola que le corresponda, siguiendo una de las tres estrategias que se detallarán más adelante. Cada elemento de la cola, tendrá tres campos:
   - Carácter: indica la cola en la que está (A, B o C).
   - Entero: indica su tiempo de atención (valor obtenido en el paso 2).
   - Entero: indica el tiempo de permanencia en la cola. Se obtendrá sumando el tiempo de atención de dicha persona a los tiempos de atención de todas las personas que le preceden en su cola.
4. Una vez que las colas están llenas, después del paso 3, se irán vaciando simultáneamente en orden creciente de los tiempos de permanencia, y los elementos se almacenarán en una lista, llamada *salida*, en la cual se almacenarán todas las personas. Las colas serán liberadas una vez que se vacíen.
5. Una vez completa la lista de salida, se visualizarán sus elementos en orden creciente y se calculará el tiempo medio de permanencia de todas las personas que componen esta lista.

Para simplificar el problema, supondremos que hasta que las personas no se han distribuido en las tres ventanillas, no serán atendidas y no saldrán de su cola. Las tres estrategias a seguir para seleccionar una ventanilla serán las siguientes:

1. *Estrategia 1.* Seleccionar la cola con menos personas.
2. *Estrategia 2.* Seleccionar aleatoriamente una de las tres colas.
3. *Estrategia 3.* Seleccionar aquella cola en la cual la suma de los tiempos de atención de los individuos que ya están en ella es el menor.

Para poder simular el funcionamiento de las tres estrategias, usando los mismos datos, los pasos 1 y 2 se harán solo una vez, y los pasos 3, 4 y 5 se repetirán tres veces, una vez para cada tipo de estrategia.

5. Conclusiones

Este trabajo ha presentado una serie de propuestas para aprendizaje activo de la materia *Programación* basadas en plantear al alumno problemas de complejidad media del mundo real que deben ser resueltos mediante el análisis, diseño e implementación de un programa informático. Al experimentar con casos prácticos reales, los alumnos deben ser capaces de reflexionar e interiorizar los conceptos de la asignatura y tomar una actitud activa en su proceso de aprendizaje. Así, la aplicación del aprendizaje activo para el desarrollo del programa práctico de la materia ha resultado muy adecuado para desarrollar las competencias esperadas en un futuro ingeniero y promover el trabajo autónomo del alumno.

Referencias

1. The ACM-IEEE Joint Task Force for Computing Curricula 2005: Computing Curricula 2005 (CC2005): The Overview Report (2005)
2. Bonwell, C.C., Eison, J.A.: Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports. ERIC (1991)
3. García-Peñalvo, F.J., Moreno García, M.N., Bravo Martín, S., Conde González, M.Á.: Aprendizaje basado en problemas para la parte práctica de la materia ingeniería del software. https://gredos.usal.es/jspui/handle/10366/81695 (2010)
4. González, A., Rodríguez, M., Olmos, S.: Aprendizaje activo en ingeniería técnica informática, esp. gestión. sistemas informáticos. In: Actas de las I Jornadas de Innovación Educativa de la Escuela Politécnica Superior de Zamora. pp. 627–640 (2006)
5. Jair E. Rocha, Carlos A. Arango, J.L.: Aprendizaje activo en ingeniería industrial. In: Eleventh LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2013). Cancun, Mexico (aug 2013)
6. Jovanovic, B., Nyarko, Y.: Learning by doing and the choice of technology. Econometrica pp. 1299–1310 (1996)
7. March, A.F.: Metodologías activas para la formación de competencias. Educatio siglo XXI **24**, 35–56 (2006)
8. Mestre, M.J.O., Palao, C.G., Peris, M.F., Navarro, M.B.: Aprendizaje activo y desempeño del estudiante: diseño de un curso de dirección de la producción (active learning methods and student performance: A design of a production management course). WPOM-Working Papers on Operations Management **3**(2), 84–100 (2012), iSSN: 1989-9068
9. Navarro, L.P.: Aprendizaje activo en el aula universitaria: el caso del aprendizaje basado en problemas. Miscelánea Comillas. Revista de Ciencias Humanas y Sociales **64**(124), 173–196 (2006), iSSN: 2341-085X
10. Nutall, G.: The cultural myths and realities of classrooms. Teaching and learning: a Reacher College Record pp. 895–934 (2005)
11. Pérez, A., Soto, E., Sola, M., Serván, M.: In: AKAL, E. (ed.) Aprender en la Universidad. El sentido del Cambio en el EEES, Espacio Europeo de Educación Superior, vol. 1 (2009)
12. Rojas Ruiz, F.: Estrategia de aprendizaje activo y cooperativo para periféricos y dispositivos de interfaz humana en el grado de ingeniería informática. Enseñanza y Aprendizaje de Ingeniería de Computadores (3), 69–76 (2013), iSSN: 2173-8688
13. Spinel, S.C., Ortiz, J.C.R.: Prácticas docentes que promueven el aprendizaje activo en ingeniería civil. Revista de ingeniería (18), 48–55 (2003), iSSN 0121-4993
14. Trujillo Suárez, C.A., González Agudelo, E.M.: Aprendizaje activo en cursos básicos de ingeniería: un ejemplo en la enseñanza de dinámica. Uni-pluri/versidad **10**(2), 65–67 (2010), iSSN: 2665-2730