PRÁCTICA WORDLE

Fundamentos de programación
T1-Z01
Fernando Carmona Palacio y Ángel Ibáñez Sendino

VERSIÓN

Versión completa con ficheros en los que se guardan el diccionario y los resultados, comprobación de si el usuario hace trampas, y versión especial de poder introducir las palabras tanto en mayúsculas como en minúsculas.

ESTRATEGIA GENERAL DE RESOLUCIÓN

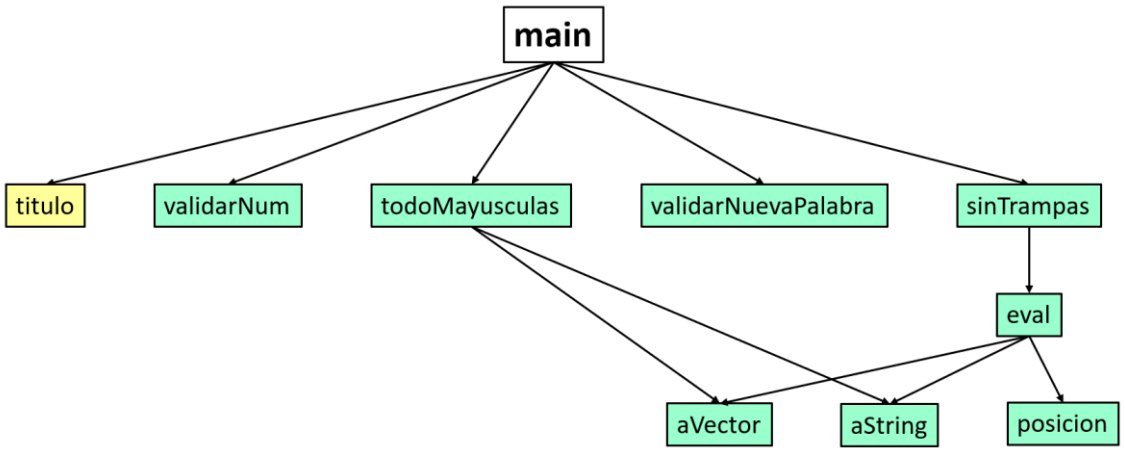
Nuestro algoritmo se basa en asociar a cada palabra del diccionario importado un valor sobre cómo de buena es esa palabra en relación con lo devuelto por el usuario. Para ello, por cada letra que coincida con un 2 devuelto, su valor asociado aumentará en 1000 (un número lo suficientemente grande como para que sea lo más importante), por cada 1 devuelto que coincida se sumará 1 (mucho menos significativo que los doses, pero que marca la diferencia entre las de igual valor en los millares), y por cada 0 que coincida se le restará 1 al valor (para que se propongan palabras con los menores ceros posibles). Después de esto también se comprobará si la palabra más probable a proponer ya se ha dicho, para que esto no sea así.

Por último, se comprueba con el método *eval* si el usuario ha hecho trampas en alguna de las entradas que haya hecho. Para comprobar esto se usa la palabra pensada, que introduce al finalizar el juego, y las palabras que la máquina ha ido proponiendo para comparar los resultados introducidos con los que se deberían haber introducido. Si hay al menos uno de ellos que no coincide, se considerará que el usuario ha hecho trampas (o se ha equivocado) y no se le dejará añadir su palabra al nuevo diccionario actualizado, ni la máquina se considerará perdedora a la hora de actualizar las estadísticas.

ESTRUCTURA DE MÓDULOS

Nuestro programa es sencillo desde el punto de modular, pues cuenta con 8 funciones, llamadas *validarNum*, *todoMayusculas*, *validarNuevaPalabra*, *posicion*, *aString*, *aVector*, *eval* y *sinTrampas*, y un único procedimiento llamado *titulo*.

La estructura es la siguiente:



(En amarillo aparecen los procedimientos y en turquesa las funciones)

Las características y objetivos de cada método son:

validarNum

Función con un solo parámetro *String* "s" que devuelve el valor *boolean* "validar". El objetivo de este método es validar si un número, introducido en forma de *String* cumple las siguientes condiciones: si su longitud es 5, y si todos los caracteres son '0', '1' o '2'. De esta forma, la función devuelve *true* si se cumplen con las especificaciones y *false* si no lo hacen. Se usará para comprobar si los números que introduce el usuario para cada palabra son válidos, a fin de volver a pedírselo si no es así.

todoMayusculas

Función con un parámetro *String* "s" que devuelve otro *String* "devuelto". El objetivo de este método es transformar un *String* a ese mismo *String*, pero cambiando las minúsculas por mayúsculas. Se utiliza a lo largo del código para dar una mayor facilidad al usuario a la hora de introducir palabras.

validarNuevaPalabra

Función con un solo parámetro *String* "s" que devuelve el valor *boolean* "validar". El objetivo de este método es validar si una palabra, introducida en forma de *String* cumple las siguientes condiciones: si su longitud es 5 y si todos sus caracteres son mayúsculas del alfabeto inglés. De esta forma, la función devuelve *true* si se cumplen con las especificaciones y *false* si no lo hacen. Se usará para validar, en el caso de que el ordenador no gane, si la palabra que había pensado el usuario es correcta para añadirla al diccionario, con la finalidad de volver a pedírsela si no es así.

posicion

Función con dos parámetros, *char[ ]* "v" y *char* "c", que devuelve el valor *int* "p". El objetivo de este método es devolver la posición del carácter "c" dentro del vector "v". Si el carácter no está en el vector, la función devuelve -1. Se llamará desde el método *eval* (ver más adelante).

aString

Función con un solo parámetro *char[ ]* "v" que devuelve el *String* "s". El objetivo es transformar el vector de caracteres a una sola palabra, en el orden del vector. Se llamará desde los métodos *eval* y *todoMayusculas*.

aVector

Función con un solo parámetro *String* "s" que devuelve el *char[ ]* "v". El objetivo es transformar el *String* a un vector de caracteres, en el orden de la palabra. Se llamará desde los métodos *eval* y *todoMayusculas*.

eval

Función con dos parámetros, *String* "oculta" y *String* "propuesta", que devuelve el *String* "respuesta". En el *String* "respuesta" se devuelve la cadena de ceros, unos y doses que devolvería el *Wordle* original como evaluación de la palabra propuesta respecto a la oculta. Como precondición tenemos que ambos *Strings* tienen que ser de la misma longitud, y que ninguno de ellos contienen los caracteres '+' o '-'. Su funcionamiento se basa en hacer varias pasadas, en una primera se apuntan los doses (y se tachan esas letras para que no se vuelvan a marcar), y en la segunda se apuntan los unos (y se tachan para que no se vuelvan a marcar). Este método se llama desde la función *sinTrampas* (ver a continuación).

sinTrampas

Función con tres parámetros, *String[ ]* "respuestas", *String[ ]* "palIntentadas" y *String* "nuevaPalabra"; que devuelve el valor *boolean* "sinTrampas". Su objetivo es comparar dos vectores de *Strings*, y devuelve el valor *true* si los *Strings* en la misma posición de ambos vectores coinciden. Se usa para comprobar si las respuestas dadas por el usuario a cada palabra propuesta y las respuestas que se deberían haber dado coinciden para saber si el usuario ha hecho trampas o se ha equivocado al introducirlos.

titulo

Procedimiento usado para imprimir el título del programa usando almohadillas (#). Su función es simplemente ahorrar código dentro del *main*.

ESTRUCTURAS DE DATOS UTILIZADAS

Para llevar a cabo esta práctica, hemos utilizado gran variedad de las estructuras de datos aprendidas en la materia, tales como vectores, matrices y ficheros. A continuación, se comentan las usadas.

Primeramente, hemos empleado dos ficheros, llamados "Diccionario.txt" y "Resultados.txt", en los que se almacenan, respectivamente, las palabras del diccionario de juego y el número de partidas jugadas y las ganadas por la máquina.

Además, nos hemos ayudado de una gran cantidad de vectores, los cuales iremos desgranando a continuación:

El primero, y más importante, es el *array* de *Strings* "diccionario". En él se guardan, al comenzar el juego, las palabras que hay en el fichero con el mismo nombre. De este modo, realizar la mayoría de las funciones que se llevan a cabo en el código resulta más sencillo ya que podemos utilizar las características de los vectores, más preferibles (en este caso) a las de los ficheros. De estas características destacan el acceso por índice (y no secuencial) y el tamaño fijo. Gracias a ello, podemos utilizar el bucle *for* en multitud de ocasiones.

"valorAsoc" es un vector de enteros que se usa de forma complementaria a "diccionario". Su nombre hace referencia a que el valor entero de cada posición va asociado al *String* con la misma posición en el vector "diccionario". Este valor se obtiene mediante una serie de interacciones y condicionales que calculan un valor para cada palabra que representa su semejanza a la pensada. Para ello usa la respuesta introducida por el usuario en el correspondiente intento y la palabra impresa en pantalla en el anterior intento (a excepción del primer intento, en el cual esta palabra se elige aleatoriamente de entre las existentes en el diccionario). De este modo, cuanto mayor sea el valor asociado a una palabra, más se parece a la palabra oculta.

Hemos nombrado "respuestas" a un unidimensional de tamaño 6, que nos sirve para almacenar las respuestas introducidas por el usuario. Su razón de ser en el programa es servirnos para realizar la comprobación de si el usuario ha realizado trampas o se ha equivocado, para lo cual se utilizan otras estructuras de datos de las que hablaremos posteriormente.

Otro vector utilizado es "palIntentadas", donde almacenamos las palabras que ya ha impreso en pantalla la maquina durante esta partida, de modo que podemos usarlas a efector de contraste en la función *sinTrampas*, de la que ya hemos hablado. Su tamaño es de 6, dado que este es el número máximo de palabras que puede utilizar la máquina para intentar adivinar la palabra oculta en cada partida.

Las estructuras de datos comentadas anteriormente son declaradas dentro del main y, por tanto, su ámbito es este y los métodos.

DICCIONARIO INICIAL.

Nuestro diccionario inicial cuenta con las siguientes 20 palabras elegidas: SOBRE, CREMA, LAPIZ, TEMOR, LETRA, RUEDA, LIBRO, LLAVE, GENTE, FAUNA, SALUD, LENTE, PINAR, PODER, METAL, MUELA, PARED, RUIDO, SILLA y CARNE (todas estas palabras están en mayúsculas y no se tienen en cuenta las tildes).

Cabe mencionar que, dado el reducido tamaño del diccionario, que puede contar hasta un máximo de 100 palabras, en comparación con un idioma tan rico en léxico como es el español, las posibilidades que tiene la máquina de ganar son bastante reducidas.

Por ello, las palabras que se encuentran en el diccionario inicial son fruto de una investigación con el fin de averiguar cuáles son algunas de las palabras con 5 letras más frecuentes en español. El propósito es que el juego sea lo más dinámico posible al tener la maquina mayores posibilidades de ganar.

BATERÍA DE PRUEBAS.

Casos	Entrada	Salida esperada	Salida obtenida
1	00000 00000 00000 00000 00000 00000 *Una palabra que haya sido impresa en pantalla	Te hemos pillado haciendo trampas así que en esta partida voy a considerar que he ganado también	Te hemos pillado haciendo trampas así que en esta partida voy a considerar que he ganado también
2	22222	¡¡¡HE GANADO!!!	¡¡¡HE GANADO!!!
3	Patata (cuando hay que introducir números)	ERROR. La entrada no es correcta Tienes que meter 5 números, y deben ser 0, 1 o 2	ERROR. La entrada no es correcta Tienes que meter 5 números, y deben ser 0, 1 o 2
4	Zanahoria (cuando se pregunta la palabra pensada)	La palabra oculta no es válida La palabra debe tener 5 letras, todas ellas mayúsculas o minúsculas, del alfabeto inglés y sin espacios	La palabra oculta no es válida La palabra debe tener 5 letras, todas ellas mayúsculas o minúsculas, del alfabeto inglés y sin espacios
5	Jijas (cuando se pregunta sí o no)	ERROR, debe responder SI o NO	ERROR, debe responder SI o NO