Simulación de carreras de coches

Fecha de entrega de la versión 3: Martes 5 de diciembre de 2023

(Imagen extraída de *http://motorsportsimulator.com*)

En esta versión de la práctica mejoraremos el simulador desarrollado en la versión anterior. Esencialmente, se introducirán dos cambios significativos. Por un lado, la carretera contendrá varios carriles, donde por cada uno de ellos circulará un coche. Además, se gestionará una lista de clasificación de forma que podamos consultar el orden de llegada de los coches en cada carrera.

1. La simulación en detalle

En la versión 2 de esta práctica (v2) el programa controlaba un único coche circulando por un carril, el cual se configuraba utilizando datos leídos de un archivo de texto. En este caso, dado que la carretera puede contener varios carriles, el archivo contendrá todos los datos necesarios para poder cargarlos. Cabe remarcar que la funcionalidad de las posiciones `clavo` y `sorpresa` es la misma que en la versión v2. Igualmente, los coches empiezan en la *posición inicial* 0 y en cada turno, avanzan o permanecen parados (en un pinchazo), según corresponda. Cada carrera estará identificada por un nombre que introducirá el usuario.

Consideramos que la simulación de la carrera ha finalizado cuando todos los coches involucrados en la misma han llegado a meta. Cuando ocurre esto, se muestra el clasificación, se almacena en la lista de clasificaciones y se pregunta al usuario si desea realizar otra simulación. Si la respuesta es afirmativa, se vuelven a colocar los coches en la posición inicial con su tiempo parado igual a 0 y se comienza una nueva simulación. En caso contrario, la lista de clasificaciones se almacena en el archivo de texto “`clasificacion.txt`”.

El modo de ejecución será el mismo que el utilizado en las versiones anteriores, es decir, tendremos el modo *normal* y el modo *depuración*.

2. Carretera con varios carriles, cada uno con un coche distinto.

Con el fin de mantener la consistencia, dejaremos la posición inicial vacía (posición normal), es decir, no podrá ser ni una posición `clavo` ni una posición `sorpresa`. Además, cada carril puede tener una configuración distinta, es decir, los clavos y sorpresas pueden estar en posiciones diferentes.

Por ejemplo, el archivo **"`carriles.txt`"** que suministramos con el código, es de la forma:

```
CLAVO 3 2 6 8
SORPRESA 4 3 5 7 9
XX
CLAVO 1 5
XX
SORPRESA 1 6
CLAVO 1 3
CLAVO 1 7
XX
```

donde puedes observar que existen tres carriles, cada uno delimitado por el centinela **"XX"**. En el primer carril existen 3 clavos y 4 sorpresas (primer número de cada secuencia), ubicados en las posiciones 2,6, 8 y 3,5,7,9 respectivamente. El segundo carril contiene un clavo en la posición 5 y ninguna posición sorpresa. El tercer y último carril contiene una sorpresa en la posición 6 y dos clavos, uno en la posición 3 y otro en la posición 7. Observa que el número de carriles es una constante del programa.

Es importante remarcar que, en cada carril, tanto la configuración de los clavos como de las sorpresas, puede estar contenida en una o varias líneas, tal y como se muestra en los clavos del tercer carril y como ocurría ya en la versión 2. Por ello, tu programa debe tener esto en cuenta y procesar los identificadores CLAVO y SORPRESA existentes en cada carril para realizar la carga de la carretera correctamente.

Con el fin de simplificar el proceso de carga, podemos asumir que los ficheros procesados serán correctos, esto es, los tipos de las posiciones sólo serán CLAVO y SORPRESA, cada posición será un número entero dentro de los límites de la carretera, y la misma posición de un mismo carril no puede tener tipos distintos, por ejemplo, un CLAVO y una SORPRESA. El resto de posiciones del carril, es decir, las que no aparecen en el fichero, se consideran posiciones normales.

Una vez configurada la carretera tras procesar el fichero, se solicitará un identificador para la carrera, y ésta empezará por el coche situado en el primer carril. Cuando finalice el turno del coche actual, el turno pasará al coche siguiente. Cuando el turno llegue al coche del último carril, y se haya procesado correctamente, éste pasará al coche del primer carril. Así transcurrirá la simulación hasta que todos los coches lleguen a la meta. En este punto, se almacenará la clasificación en la lista de clasificaciones, se mostrará la clasificación actual con el orden de llegada de los coches, y el usuario decidirá si se ejecuta una nueva simulación, o si termina definitivamente la ejecución del programa. En caso de que se decida terminar la simulación, la lista de clasificaciones se almacenará en el archivo de texto **"`clasificacion.txt`"**.

2.1. Detalles de implementación

En esta versión de la práctica se utilizarán las mismas constantes que las ya utilizadas en la versión anterior, pero tendremos que añadir alguna más: CENTINELA toma el valor XX correspondiente al centinela del fichero de entrada de datos; `NUM_CARRILES` definirá el número de carriles de una carretera, y `MAX_CARRERAS` se utilizará para representar el número máximo de carreras que se pueden almacenar en la lista de clasificaciones.

Tened en cuenta que ahora la carretera, al contar con varios carriles, debe separar los mismos. Utilizaremos el carácter `CHAR_LINEA_HORIZONTAL = 205` (código ASCII asociado al carácter `'='`).

Por ejemplo, la situación inicial para la carretera representada en el fichero "`carriles.txt`", con las constantes `NUM_CARRILES`=3 y `LONG_CARRETERA=10` sería:

2.1.1 Estructuras de datos

Para representar un coche en la carrera utilizaremos el tipo de datos `tCoche` de la siguiente forma:

```
struct tCoche {
	int pos;
	int tiempoParado;
};
```

donde `pos` indica la posición del coche en el carril, y `tiempoParado` indica la cantidad de turnos que el coche debe estar parado por un pinchazo. Cuando este número sea 0, podrá moverse como corresponda.

De forma similar a como definimos la carretera en la versión anterior, ahora definiremos `tCarril` como un struct que agrupa la información del coche y del array de tipo `tTipoPosicion` que representa el carril de la carretera por donde circula el coche.

```
struct tCarril {
	tCoche coche;
	tTipoPosicion posiciones[LONG_CARRETERA];
};
```

donde `tTipoPosicion` será un enumerado que define los valores NORMAL, CLAVO y SORPRESA.

Una vez definido el carril, resulta intuitivo definir la carretera como un array de carriles (`tCarril`):

```
using tCarretera = tCarril[NUM_CARRILES];
```

La gestión de la clasificación de cada carrera se llevará a cabo con una lista. Para ello, debemos definir primero una estructura que contenga el identificador de la carrera, un array con el orden de llegada de cada coche y un contador con el número de coches que ya han llegado.

```
struct tClasificacion{
      string idCarrera;
      int clasificacion[NUM_CARRILES];
      int cont;
};
```

La estructura anterior contiene, en un *string*, el identificador de la carrera que se introduce por teclado antes de iniciar la misma, así como una lista con el orden de llegada de los coches. Puede observarse que `clasificacion` es un array que contiene NUM_CARRILES números enteros. El contenido de la posición i-ésima del array representa el carril del coche que ha llegado i-ésimo en la clasificación.
Por último, definiremos una lista para almacenar las clasificaciones de las diferentes carreras que se corran:

```
struct tListaClasificacion {
      tClasificacion lista[MAX_CARRERAS];
      int cont;
};
```

2.2.2 Subprogramas

En esta sección se describen los subprogramas (funciones y/o procedimientos) utilizados en esta versión de la práctica. Están agrupados según su funcionalidad. En la mayor parte de los casos, deberás adaptar los subprogramas de la versión anterior para que se utilicen las estructuras de datos definidas en la sección 2.2.1.

Subprogramas para leer del archivo de texto y cargar la carretera.

`void iniciaCoche(tCoche& coche)`: inicializa un coche, estableciendo su posición inicial y el número de turnos que debe permanecer parado a 0.

`void iniciaCarril(tCarril& carril)`: inicializa un carril asignando a todas sus posiciones el tipo NORMAL, además de inicializar el coche que contiene.

`void leeCarril(ifstream& archivo, tCarril& carril)`: carga un carril completo desde el archivo. El código es el utilizado en la función `cargaCarretera` de la versión 2, donde solo se cargaba un carril.

`bool cargaCarretera(tCarretera& carriles)`: pide el nombre del archivo que contiene la información de la carretera y, si el fichero se abre correctamente, realiza la carga de los carriles usando la función `iniciaCarril`. Si el archivo no existe, o no puede abrirse correctamente, devuelve falso. Antes de cargar el carril, la función inicializa el carril.

Subprogramas encargados de controlar la lógica de la simulación.

`bool esSorpresa(const tTipoPosicion pos[], int p)`: devuelve `true` sí y sólo sí `pos[p]` contiene SORPRESA, en caso contrario devuelve `false`.

`bool esClavo(const tTipoPosicion pos[], int p)`: devuelve `true` sí y sólo sí `pos[p]` contiene CLAVO, en caso contrario devuelve `false`.

`int buscaSiguientePosSorpresa(const tCarril& carril, int incr)`: devuelve la siguiente/anterior posición sorpresa donde debe situarse el coche del carril. Si el parámetro incr toma el valor 1, el coche avanza hasta la siguiente sorpresa. Si toma el valor -1, el coche retrocede hasta la sorpresa anterior. Si el coche se encuentra en la última posición sorpresa, el coche pasa a la posición inicial 0.

`int avanza(int posCoche)`: calcula el avance del coche teniendo en cuenta el modo de ejecución. Su definición es similar a la de la versión 2 de la práctica.

`bool haLlegado(int posCoche)`: devuelve `true` si la posición del coche es mayor o igual a LONG_CARRETERA. Igual que en la versión 2.

`bool calculaPosicion(tCarril& carril)`: analiza la posición del coche. Si dicha posición contiene un clavo, entonces actualiza el valor del tiempo de parada del coche a la constante correspondiente. En caso de ser una posición sorpresa, decide, de forma aleatoria, si hay que avanzar o retroceder. En modo depuración se pedirá al usuario si se avanza o retrocede con la sorpresa. Posteriormente invoca a la función `buscaPosicionSorpresa` para actualizar la posición del coche. Además, esta función devuelve true sí y solo sí la posición del coche es posición sorpresa. Similar a la función de la versión 2 de la práctica.

`bool avanzaCarril(tCarretera carretera, int i)`: Si el coche del carril `i` no ha llegado a la meta y no está parado por un pinchazo, se calcula su avance (utilizando la función `avanza`). Seguidamente, se analiza la nueva posición alcanzada. Si se ha llegado a la meta, colocamos el coche en la posición LONG_CARRETERA. En otro caso, se dibuja la carretera y se invoca a `calculaPosicion`. Si ha caído en una posición sorpresa, la función anterior devuelve `true` y calcula la nueva posición a la que salta el coche. Por tanto, para visualizar dicho salto, se vuelve a dibujar la carretera. Devuelve cierto si el coche ha llegado a la meta y falso en caso contrario. Similar a la función de la versión 2 de la práctica.

`void avanzaCarriles(tCarretera carretera, tClasificacion& clasificacion)`: realiza el avance, en orden ascendente, de todos los coches. El avance de cada coche se realiza con el código de la función `avanzaCarril`. El parámetro `clasificacion` debe utilizarse para almacenar el orden de llegada de los coches. Concretamente `clasificacion[i]=j` significa que el coche en el carril `j` ha llegado en la posición `i`.

`tClasificacion simulaCarrera(tCarretera carretera)`: subprograma que controla la simulación de la carrera completa. Tal y como se ha comentado anteriormente, la carrera finaliza cuando todos los coches llegan a la meta. Básicamente ejecuta un bucle de tal forma que,

mientras no hayan llegado todos los coches a la meta, invoca a `avanzaCarriles`. Devuelve la clasificación de la carrera que se ha simulado.

Subprogramas encargados de dibujar el estado de la carrera.

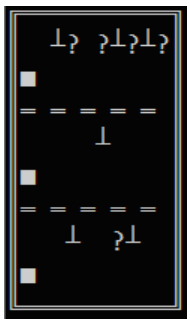
`void dibujaLineaHorizontalInferior():` dibuja el borde inferior de la carretera, de forma similar a la versión 2.

`void dibujaCarril(const tCarril& carril):` dibuja el carril pasado en el primer parámetro. Se implementa de forma similar a la versión 2.

`void dibujaLineaDiscontinua():` Dibuja la línea discontinua que hace de separador de carriles.

`void dibujaLineaHorizontalSuperior ():` dibuja el borde superior de la carretera. Es igual que el correspondiente subprograma de la versión 2.

`void dibujaCarretera(const tCarretera carriles):` dibuja la carretera completa, esto es, el borde superior, los carriles seguidos de una línea discontinua, excepto el último, y el borde inferior. Por ejemplo, si `NUM_CARRILES = 3`, y `LONG_CARRETERA = 10`, la carretera se dibujaría como sigue:



Si cambiamos `LONG_CARRETERA = 13`, el aspecto de la carretera sería:

Subprogramas encargados de gestionar y mostrar la clasificación.

`std::ostream& operator<<(std::ostrean& salida, tClasificacion const& cl):` sobrecarga del extractor para escribir una clasificación con el formato pedido en la salida por consola y en el fichero de salida.

`void iniciaListaClasificacion(tListaClasificacion& listaC)`: inicia la lista de clasificaciones poniendo el contador a 0.

`void eliminaClasificacion (tListaClasificacion& listaC, int pos)`: Elimina la clasificación de la posición `pos` de la lista `listaC`, manteniendo el orden de las clasificaciones.

`void insertaClasificacion (tListaClasificacion& listaC, const tClasificacion& clasificacion)`: inserta la clasificación al final de la lista de clasificaciones. Si la lista está completa, elimina la clasificación de la primera carrera para poder añadir la nueva carrera al final de la lista.

`bool guardaListaClasificacion(const tListaClasificacion& listaC)`: guarda en el fichero "`clasificacion.txt`" la lista completa de clasificaciones. Debe utilizar la sobrecarga del extractor para una clasificación.

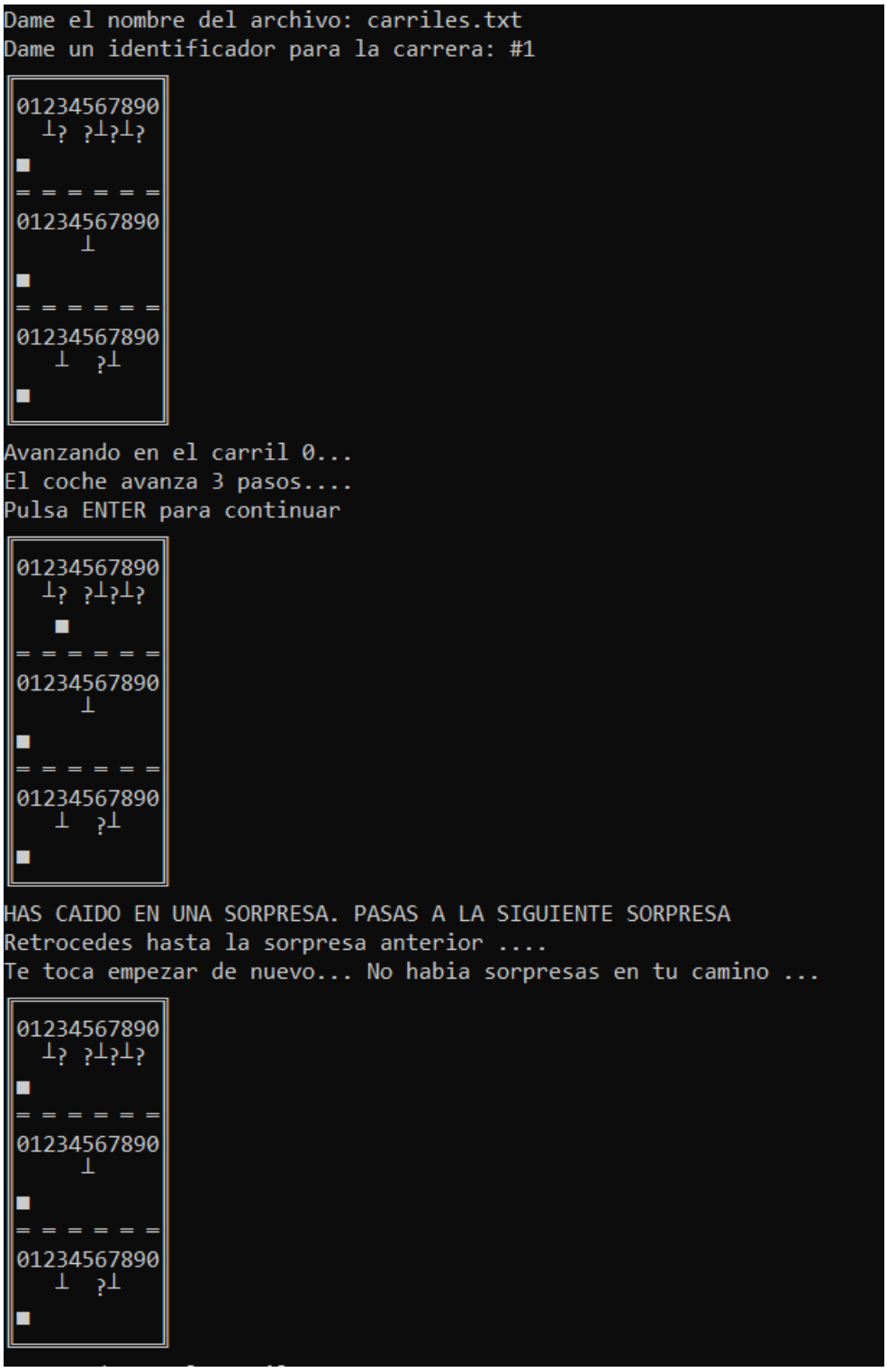
La función `main`

La función `main`, primero carga la carretera utilizando el fichero indicado por el usuario, se solicita un identificador para la carrera y se inicializa la lista de clasificaciones. Seguidamente comienza la simulación. Cuando ésta finaliza, se muestra la clasificación por pantalla, se almacena en la lista de clasificaciones y se vuelve a preguntar al usuario si desea realizar otra simulación. En caso afirmativo, se inicializan los coches y se repite el proceso. Cuando el usuario indica que ya no se realizan más simulaciones, se guarda la lista de clasificaciones actual en el archivo "`clasificacion.txt`".

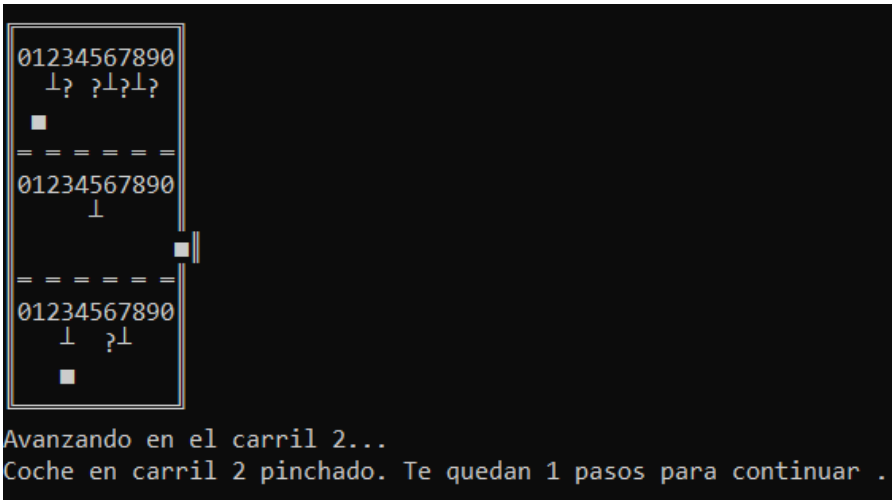
A continuación mostramos algunos ejemplos de ejecución:

3. Ejemplos de ejecución

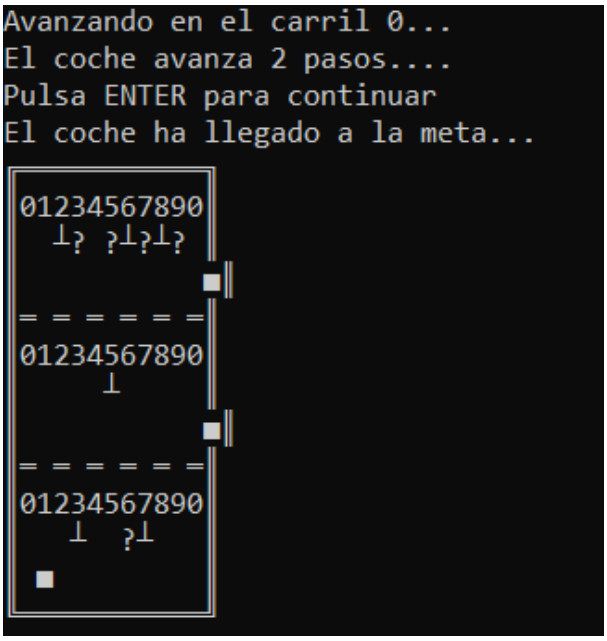
En la siguiente ejecución hemos utilizado las constantes: LONG_CARRETERA=11, MAX_PASOS=3, NUM_CARRILES=3 y TIEMPO_PARADO=2, así como el modo normal. Mostramos solo algunos pasos de la simulación. Además, hemos incorporado en la parte superior de cada carril los números de las posiciones. Observa que pasando de la 10, volvemos a numerar con 0. La simulación comienza:
Los números de las posiciones no hay que incorporarlos.



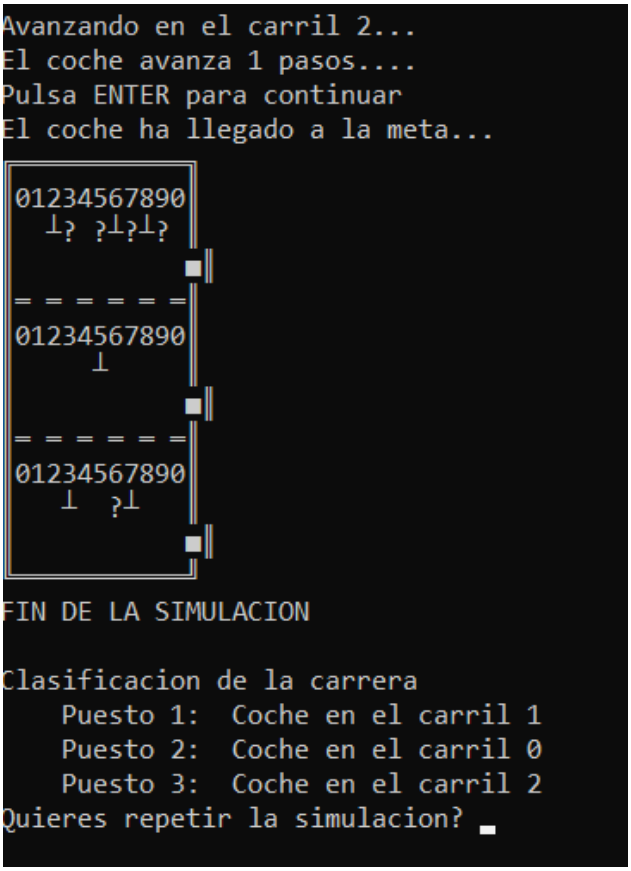
Ahora se muestra la llegada del primer coche (el del carril 1), a la meta:



El siguiente coche en llegar es el del carril 0:



Finalmente llega el coche del carril 2, y se acaba la simulación, mostrándose la clasificación:



En el siguiente ejemplo usamos el modo depuración, y usamos sólo dos carriles para visualizar como funciona los turnos cuando un coche ha pinchado.