Práctica 3
Tortugas robóticas

Fecha de entrega: 1 de abril

1. Descripción del juego

La práctica consiste en desarrollar un programa en C++ para jugar a un juego inspirado en el **Robot Turtles**, un juego de mesa del que se puede obtener información en (www.robotturtles.com/instructions).

Robot Turtles se desarrolla en un tablero de 8x8 casillas y pueden jugar entre 1 y 4 jugadores. El objetivo del juego es ser el primero en conseguir una joya. A continuación se muestra una imagen tomada de [https://www.kickstarter.com/projects/danshapiro/ robot-turtles-the-board-game-for-little-programmer)](https://www.kickstarter.com/projects/danshapiro/ robot-turtles-the-board-game-for-little-programmer):

Nosotros vamos a implementar una variante inspirada en este juego.

Preparación del tablero

Antes de comenzar el juego, han de colocarse sobre el tablero los siguientes elementos:

- Una tortuga por jugador (de 1 a 4).
- Tantas joyas como jugadores haya. El objetivo del juego es ser el primero en conseguir una de estas joyas.
- Entre 0 y 20 muros de piedra. Estos muros son fijos e impiden avanzar a las tortugas.
- Entre 0 y 12 bloques de hielo. Estos bloques también impiden avanzar a las tortugas pero se pueden derretir si se usa una pistola láser.
- Entre 0 y 8 cajas. Las cajas se pueden empujar en la dirección del movimiento de la tortuga siempre que haya una casilla libre justo detrás en esa dirección donde ponerla. Al empujar la caja, tanto la caja como la tortuga avanzan una casilla.

Las configuraciones iniciales de tablero se cargarán siempre de fichero, y garantizarán que todos los jugadores puedan conseguir una joya.

El mazo de cartas

Para alcanzar la joya, la tortuga tiene disponibles cartas de varios tipos:

- Carta “Avanza”: permite a la tortuga avanzar una casilla en la dirección en la que mira si dicho movimiento es posible. La tortuga no puede salirse del tablero, ni atravesar un muro, ni ocupar el lugar que ya está ocupado por otra tortuga, ni empujar una caja que detrás no tenga una casilla libre, por lo que en esos casos la carta no tendrá efecto.
- Carta “Gira a la derecha”: permite a la tortuga girar en el sentido de las agujas del reloj permaneciendo en la misma casilla.
- Carta “Gira a la izquierda”: permite a la tortuga girar en el sentido contrario al de las agujas del reloj permaneciendo en la misma casilla.
- Carta “Pistola láser”: la pistola se dispara en la dirección hacia la que mira la tortuga y golpea lo primero que alcanza en esa dirección.
    - Si es un muro de hielo este se deshace.
    - En cualquier otro caso no sucede nada.

Cada jugador dispone de un grupo de cartas llamado “mano” con las que puede jugar y otro grupo de cartas llamado “mazo” de donde puede tomar más cartas para incorporar a su mano.

El *mazo* inicial de cada jugador consta de 38 cartas:

- 18 cartas “Avanza”
- 8 cartas “Gira a la derecha”
- 8 cartas “Gira a la izquierda”
- 4 cartas “Pistola láser”

y en él las cartas están mezcladas de manera aleatoria.

El jugador coge tres cartas de la parte superior de su *mazo* para formar la *mano* inicial.

El juego

Los jugadores están numerados del 1 hasta el número de jugadores, y en cada ronda juegan en orden creciente de número.
Cuando un jugador tiene el turno puede realizar una de las acciones siguientes:

- Robar una carta de la parte superior de su *mazo* y añadirla a su *mano*.
- Formar una secuencia con un subconjunto de las cartas de la *mano* y ejecutar dicha secuencia según las normas explicadas anteriormente. Las cartas utilizadas volverán a la parte inferior de su *mazo* (en cualquier orden).

El jugador que primero coloca su tortuga sobre una joya es el ganador. En caso de que lo consiga mientras está ejecutando una secuencia, se detendrá la ejecución de la misma.

Vamos a implementar este juego registrando además las puntuaciones de los jugadores en un fichero `Puntuaciones.txt`, que se cargará al empezar la ejecución de la aplicación y se guardará a su finalización. En él se almacenan los nombres de los jugadores junto con su puntuación.

El ganador obtiene una puntuación que coincide con el número de jugadores. Así, por ejemplo, si juegan cuatro jugadores el que gana consigue 4 puntos; si juegan dos, el ganador consigue 2.

1. Datos del programa

El tipo `tCasilla` nos permite representar las casillas del tablero:

```
enum tDir {NORTE, SUR, ESTE, OESTE};
enum tEstadoCasilla {VACIA, HIELO, MURO, CAJA, JOYA, TORTUGA};
struct tTortuga {int numero; tDir direccion;};
struct tCasilla {tEstadoCasilla estado; tTortuga tortuga;};
```

En él se indica el tipo de casilla y, en caso de que la casilla contenga una tortuga, el número del jugador que la maneja y la dirección en la que mira la tortuga.

El tipo `tTablero` es un array bidimensional de `tCasillas`.

El tipo `tCarta` permite representar el tipo de cartas del mazo:

```
enum tCarta {AVANZAR, GIROIZQUIERDA, GIRODERECHA, LASER};
```

Define también un tipo enumerado `tAccion` para representar las dos clases de acciones que puede llevar a cabo un jugador en su turno.

Define los siguientes tipos estructurados:

- `tCoordenada` para representar la fila y la columna de una casilla.
- `tMazo` para representar los mazos y las secuencias de ejecución. Es una secuencia de cartas.

y el tipo array

- `tMano` para representar la mano de los jugadores. Registra cuantas cartas de cada tipo tiene el jugador en la mano.

El tipo de un jugador `tJugador` es también un tipo estructurado que lleva:

- ✓ El nombre del jugador.

- ✓ El mazo del jugador (`tMazo`).
- ✓ La mano del jugador (`tMano`).
- ✓ Las coordenadas actuales (`tCoordenada`).

El tipo `tJuego` es un tipo estructurado con información de:

- ✓ El número de jugadores.
- ✓ El turno actual.
- ✓ Un array de registros `tJugador` que contiene los cuatro posibles. El jugador de número i está colocado en la posición i-1 de este array.
- ✓ El tablero de juego.

El programa usa también un tipo enumerado `tTecla` con los siguientes valores: `AVANZA`, `DERECHA`, `IZQUIERDA`, `LASER`, `SALIR` y `NADA`.

Define un tipo `tPuntuaciones` para mantener la información de las puntuaciones de los jugadores durante la ejecución de la aplicación.

Recuerda definir las constantes que consideres necesarias para evitar utilizar números directamente en el código, por ejemplo:

```
const int MAX_JUGADORES = 4;
const int NUM_FILAS = 8;
const int NUM_TIPOS_CASILLAS = 6;
```

2. Visualización del estado del juego: tablero, manos de los jugadores y turno

Cada vez que vayas a visualizar el estado del tablero borra primero el contenido de la ventana de consola, de forma que siempre se muestre el tablero en la misma posición y la sensación sea más visual. Para borrar la consola utiliza:

```
system("cls");
```

Por defecto, el color de primer plano, aquel con el que se muestran los trazos de los caracteres, es blanco, mientras que el color de fondo es negro. Podemos cambiar esos colores, por supuesto, aunque debemos hacerlo utilizando rutinas que son específicas de Visual Studio, por lo que debemos ser conscientes de que el programa no será portable a otros compiladores.

Disponemos de 16 colores diferentes entre los que elegir, con valores de 0 a 15, tanto para el primer plano como para el fondo. El 0 es el negro y el 15 es el blanco. Los demás son azul, verde, cian, rojo, magenta, amarillo y gris, en dos versiones, oscuro y claro.

Visual Studio incluye una biblioteca, `Windows.h`, que tiene, entre otras, rutinas para la consola. Una de ellas es `SetConsoleTextAttribute()`, que permite ajustar los colores de fondo y primer plano. Incluye en el programa esa biblioteca y esta rutina:

```
void colorFondo(int color) {
   HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(handle, 15 | (color << 4));
}
```

Basta proporcionar un color para el fondo (1 a 14) y esa rutina lo establecerá, con el color de primer plano en blanco (15). Debes cambiar el color de fondo cada vez que tengas que *dibujar* una casilla y volverlo a poner a negro (0) a continuación.

Ten en cuenta que cada tortuga tiene un color de fondo distinto, y diferente de los demás elementos del juego. Para facilitar la visualización usa siempre el mismo color para el resto de los elementos de cada tipo, por ejemplo, el gris para los muros de piedra, rojo para las joyas, azul clarito para el hielo y granate para las cajas.

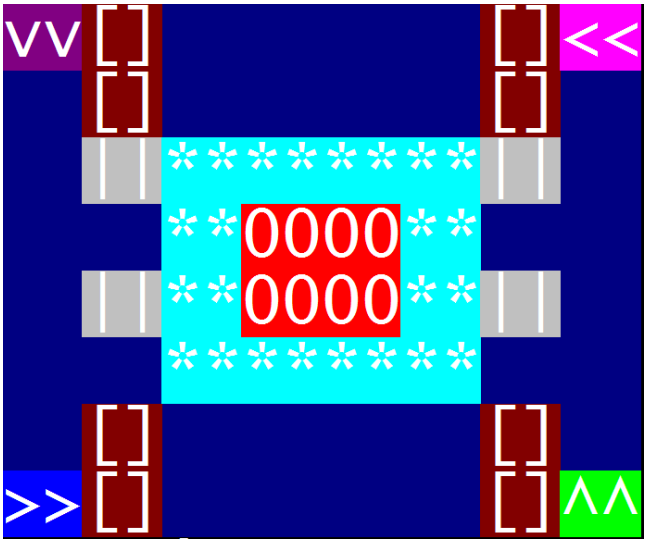
Puedes utilizar una paleta de colores fija para las casillas, los jugadores y las cartas. Por ejemplo:

```
const int PALETA[NUM_TIPOS_CASILLAS+MAX_JUGADORES] =
            {1,11,7,4,12,5,13,9,10,3};
```

A través de esta paleta de colores se puede conocer los colores de los elementos del juego allí donde hagan falta.

Utiliza los siguientes símbolos para visualizar los elementos: || para los muros de piedra, ** para los muros de hielo, [] para las cajas, 00 para las joyas; y para las tortugas que miran en las cuatro direcciones posibles usa >>, <<, ^^ y vv.

Este es un ejemplo de tablero de juego con cuatro jugadores:



Las manos de los jugadores se mostrarán al lado de su nombre (en el color correspondiente) por orden, apareciendo un símbolo > delante del que tiene el turno.

Las cartas se visualizarán con un símbolo del tipo de carta y al lado el número de cartas de ese tipo: ^ para las cartas avanzar, < para girar a la izquierda, > para girar a la derecha, ~ para el láser.

Un ejemplo de cómo aparecen los jugadores debajo del tablero:

3. Carga del juego

Los tableros de juego se leerán de un archivo de texto que contiene un tablero para cada número de jugadores: primero figura el número de jugadores y a continuación el tablero. En el siguiente ejemplo se muestra la parte del fichero de texto que corresponde al tablero de arriba:

```
4
DC      CL
 C      C
 #@@@@@#
  @$$@
 #@$$@#
  @@@@
 C      C
RC      CU
```

'#' representa muro de piedra, '@' es muro de hielo, ' ' (blanco) es casilla vacía, '$' es joya y 'C' es caja . Para representar una tortuga aparece una letra que indica la dirección en la que mira U, D, R, L (up, down, right y left). Los jugadores se van añadiendo en el orden en que los encontramos al leer los datos de arriba abajo y de izquierda a derecha.

4. El juego en acción

4.1 Funcionamiento general

Al comenzar la ejecución de la aplicación, se carga la información de las puntuaciones de los jugadores desde el fichero `Puntuaciones.txt` en la estructura `tPuntuaciones`. En el fichero aparece en cada línea el nombre de un jugador (una sola palabra, sin espacios en blanco) y a continuación, separado por un blanco, un número natural que representa sus puntos. Se puede suponer que los nombres de los jugadores no están repetidos en el fichero.

Al usuario se le ofrece inicialmente, y de nuevo después de cada opción, un menú con las siguientes opciones:

```
1. Jugar
2. Mostrar puntuaciones

0. Salir
```

cada una de las cuales lleva a cabo la acción indicada.

Si el usuario elige Jugar, se solicita el nombre del fichero de tableros, el número de jugadores y a continuación el nombre de dichos jugadores, y se carga desde ese fichero el tablero con ese número de jugadores.

Cuando un jugador alcanza una tortuga se actualiza su puntuación en la estructura `tPuntuaciones`. Cuando se elige la opción Salir, se guarda la información de la estructura `tPuntuaciones` en el fichero.

4.2 Solicitud de jugada

Cuando es el turno de un jugador se le pregunta qué tipo de jugada quiere realizar mediante la pulsación de una tecla: **R** para robar y **E** para crear y ejecutar una secuencia.

En caso de haber pulsado **E**, se le solicita la secuencia que desea crear, mediante las teclas de dirección ↑ para avanzar, → para girar a la derecha, ← para girar a la izquierda, y espacio para el láser. La tecla `ENTER` marcará el fin de la secuencia. Será necesario comprobar que la mano permite generar dicha secuencia.

A medida que se introduce la secuencia, por pantalla se puede mostrar con los mismos símbolos utilizados para la mano para que todos los jugadores puedan verla.

Para leer las teclas pulsadas por el usuario implementa el siguiente subprograma:

- ✓ `tTecla leerTecla()`: devuelve un valor de tipo `tTecla`, que puede ser una de las cuatro direcciones si se pulsan las flechas de dirección correspondientes; el valor `LASER` si se pulsa la tecla espacio; el valor `SALIR`, si se pulsa la tecla `ENTER`; o `NADA` si se pulsa cualquier otra tecla.

La función `leerTecla()` detectará la pulsación por parte del usuario de teclas especiales, concretamente las teclas de flecha (direcciones), el espacio y la tecla `ENTER` (salir). La tecla `ENTER` sí genera un código ASCII (13), al igual que el espacio (32), pero las de flecha no tienen códigos ASCII asociados. Cuando se pulsan en realidad se generan dos códigos, uno que indica que se trata de una tecla especial y un segundo que indica de cuál se trata.

Las teclas especiales no se pueden leer con `get()`, pues esta función sólo devuelve un código. Podemos leerlas con la función `_getch()`, que devuelve un entero, y se puede llamar una segunda vez para obtener el código, si el entero devuelto en la primera llamada corresponde a una tecla especial.

Esta función requiere que se incluya la biblioteca `conio.h`.

```
cin.sync();
dir =_getch(); // dir: tipo int
if (dir ==  0xe0) {
   dir = _getch();
```

```
   // ...
}
// Si aquí dir es 13, es la tecla ENTER;  si es 32 es la tecla espacio
```

Si el primer código es `0xe0`, se trata de una tecla especial. Sabremos cuál con el segundo resultado de `_getch()`. A continuación puedes ver los códigos de cada tecla especial:

↑ 72 → 77 ← 75

4.3 Módulos a implementar

4.3.1 Módulo secuencia de cartas

En este módulo se define, entre otros, el tipo `tMazo` que sirve para representar las cartas del mazo así como la secuencia de cartas a ejecutar. Implementa al menos los siguientes subprogramas:

- ✓ `void crearVacia(tMazo & mazo)`: crea una secuencia vacía de cartas.
- ✓ `bool sacar(tMazo & mazo, tCarta& carta)`: saca la primera carta. Devuelve `true` si se ha podido sacar la carta o `false` en caso contrario.
- ✓ `void insertar(tMazo & mazo, tCarta carta)`: inserta la carta al final del mazo.
- ✓ `void crearMazoAleatorio(tMazo & mazo):` crear un mazo aleatorio. Para ello rellenamos un mazo inicial con las cartas disponibles y aplicamos el método `random_shuffle` de la librería `<algorithm>` para generar una mezcla aleatoria del mismo.

4.3.2 Módulo del juego

En este módulo se define, entre otros, el tipo `tJuego` y se implementan las operaciones necesarias para gestionar el juego. Implementa al menos los siguientes subprogramas:

- ✓ `bool crearJuego(tJuego & juego)`: solicita al usuario el nombre del fichero y el número de jugadores, y carga el tablero correspondiente desde ese fichero. También inicializa los mazos y manos de los jugadores.
- ✓ `void mostrarJuego(const tJuego & juego)`: visualiza el estado actual del juego.
- ✓ `bool ejecutarTurno (tJuego &juego)`: lee la acción del jugador actual y la ejecuta. El booleano indica si el jugador que tiene el turno ha alcanzado una joya.
- ✓ `bool accionRobar(tJuego & juego)`: el jugador con el turno roba una carta de su mazo si hay cartas disponibles. El booleano indica si ha sido posible robar la carta.
- ✓ `bool accionSecuencia(tJuego & juego, tMazo & cartas)`: el jugador con el turno ejecuta una secuencia de cartas. El segundo parámetro contiene un subconjunto de las cartas que hay en la mano del jugador actual y se va consumiendo a medida que se ejecuta. El booleano indica si el jugador que tiene el turno ha alcanzado una joya en la ejecución de esta secuencia.
- ✓ `void cambiarTurno (tJuego &juego)`: cambia el turno al jugador siguiente.

- ✓ `void incluirCarta(tMano &mano, tCarta carta)`: incluye una nueva carta en la mano del jugador.

4.3.3 Módulo puntuaciones

En este módulo se define, al menos, el tipo `tPuntuaciones` y se implementan las funciones necesarias para gestionarlas. Implementa al menos los siguientes subprogramas:

- ✓ `bool cargarPuntuaciones(tPuntuaciones &puntos)`: carga las puntuaciones de los jugadores a partir del fichero.
- ✓ `void guardarPuntuaciones(const tPuntuaciones & puntos)`: guarda las puntuaciones de los jugadores en el fichero.
- ✓ `void mostrarPuntuaciones(const tPuntuaciones & puntos):` muestra las puntuaciones de los jugadores.
- ✓ `bool actualizarPuntuacion(tPuntuaciones & puntos, const string & nombre, int nuevos)`: si el jugador ya estaba, se incrementan sus puntos en `nuevos` puntos; si no está en el listado lo inserta con los `nuevos` puntos (si no hay espacio, se insertará solamente si hay algún jugador en el listado con menos puntuación que él, y en tal caso ocupará el lugar del que menos puntos tiene). Devuelve `false` si no se ha podido insertar.

Aspectos de implementación

No uses variables globales: cada subprograma, además de los parámetros para intercambiar datos, debe declarar sus propias variables locales, aquellas que necesite usar en el código.

No uses instrucciones de salto como `exit`, break (más allá de las cláusulas de un `switch`) y `return` (en otros sitios distintos de la última instrucción de una función).

Entrega de la práctica

La práctica se entregará en el Campus Virtual por medio de la tarea **Entrega de la Práctica 3**, que permitirá subir un archivo con el código fuente. Uno de los dos miembros del grupo será el encargado de subirlo, no lo suben los dos.

Recordad poner el nombre de los miembros del grupo en un comentario al principio del archivo de código fuente.