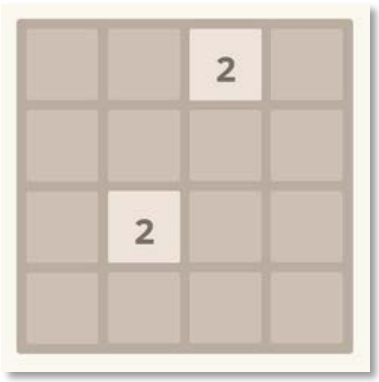
Práctica 3
2048

Fecha de entrega: 29 de marzo

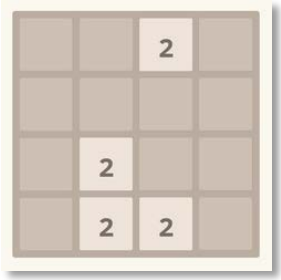
1. Descripción del juego

La práctica consiste en desarrollar un programa en C++ para jugar a **2048**, un juego inventado en marzo de 2014 por Gabriele Cirulli, un joven desarrollador web italiano.

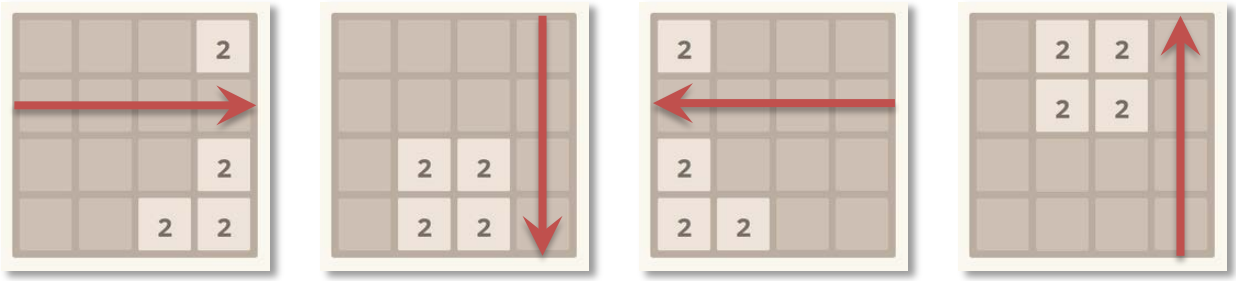
El juego utiliza un tablero de 4 x 4 baldosas que pueden tener fichas de distintos valores, siempre potencias de 2. Inicialmente se ponen dos fichas de valor 2 o 4 en baldosas elegidas aleatoriamente (con una probabilidad más alta de ser un 2 que de ser un 4).



A partir de ese momento el jugador realizará sucesivas jugadas, cada una moviendo el tablero en una dirección (arriba, abajo, derecha o izquierda). Todas las fichas *caerán* en esa dirección, cubriendo los huecos libres para que todas las fichas de cada fila o columna queden juntas hacia el borde correspondiente. Por ejemplo, con este tablero[1]...

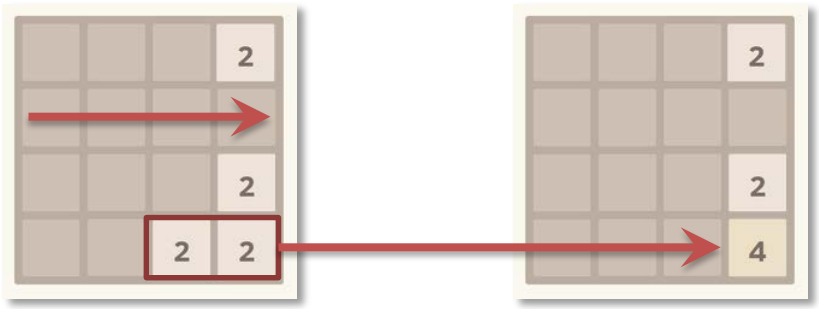


...cada movimiento desplaza las fichas como se muestra a continuación:



[1] Ilustraciones basadas en la que aparece en la entrada de la Wikipedia *2048 (videojuego)*

Una vez desplazadas las fichas, si quedan dos del mismo valor pegadas en alguna fila o columna del sentido del movimiento, entonces se combinan ambas en una sola con la suma de los valores, sustituyendo la que esté más hacia el borde de la dirección elegida. Por ejemplo, moviendo hacia la derecha:

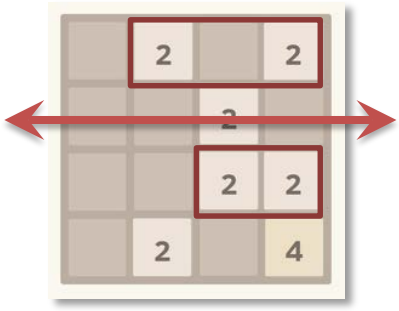


Los valores de las fichas que se consiguen con las combinaciones son obviamente potencias de 2: 4, 8, 16, 32, 64, 128, 256, 512, 1024 y 2048.

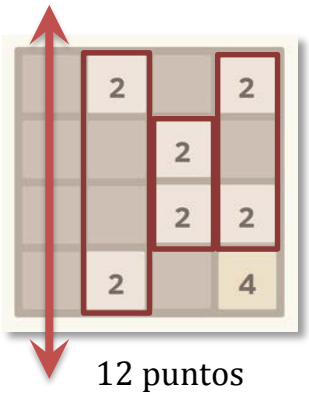
Tras cada jugada aparece otra ficha, en una baldosa libre, elegida aleatoriamente, de valor 2 o 4 (con mayor probabilidad de ser un 2 que de ser un 4).

A medida que van apareciendo nuevas fichas y se van realizando combinaciones, el tablero se va poblando de fichas con distintos valores:

El objetivo del juego es conseguir una ficha con valor 2048. En el momento en que se consigue, el juego termina (en la versión original se puede seguir). La puntuación total obtenida es igual a la suma de las puntuaciones de cada jugada. La puntuación de cada jugada es la suma de las combinaciones de fichas que se consigan:



8 puntos



12 puntos

Si, no habiendo conseguido la ficha 2048, se llena el tablero de fichas y no queda ningún movimiento posible, entonces el jugador pierde el juego.

2. El programa

El programa simulará de forma realista la dinámica del juego, aunque obviamente en modo consola. Utilizará *caracteres gráficos* para dibujar el tablero, así como colores de fondo para las fichas, y leerá teclas especiales, como las teclas de flecha o la tecla Esc.

Principalmente el programa usa un array bidimensional de 4 x 4 de enteros para mantener el estado del tablero. Usa el valor 1 (2^0) para las baldosas libres y el valor de la ficha para las baldosas ocupadas. El programa usa también un tipo enumerado `tAccion` con los siguientes valores: `Arriba`, `Abajo`, `Derecha`, `Izquierda`, `Salir` y `Nada`.

2.1. Versión 1: Inicialización, visualización y carga de tableros

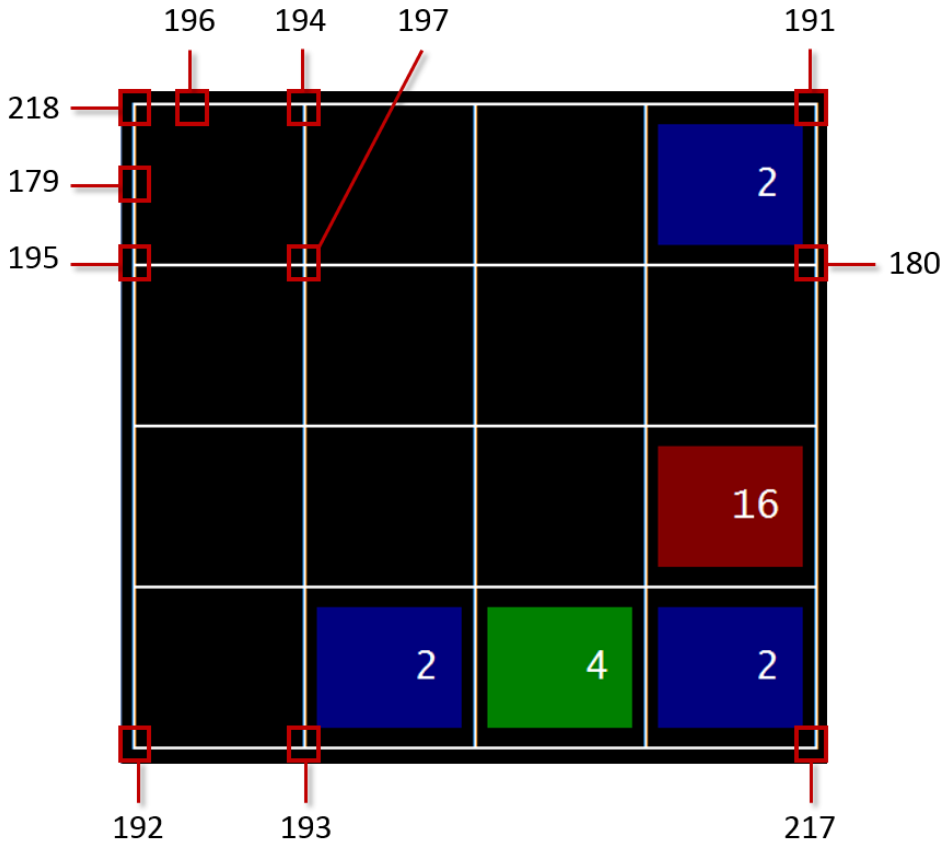
Lo primero que habrá que hacer es inicializar el tablero haciendo que todas las baldosas estén libres (valor 1) menos dos elegidas aleatoriamente que contendrán un 2 o un 4, siendo la probabilidad de ser un 2 del 95% y no pudiendo ser ambas 4.

Una vez que hayamos inicializado el tablero lo mostraremos por pantalla tal y como se puede ver en la ilustración anterior. Las baldosas libres y los bordes tienen fondo negro. Cada ficha usa un recuadro de 3 x 6 con un color de fondo que depende del valor de la ficha y un valor que se muestra dentro en color blanco. Sobre el tablero se muestra la última puntuación y la puntuación total.

Cada vez que vayas a visualizar el tablero borra primero el contenido de la ventana, 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");
```

Tras mostrar las puntuaciones parcial y total en una primera línea (dejando una en blanco antes), construye el tablero, teniendo en cuenta que para los bordes de las baldosas necesitas caracteres especiales. A continuación te indicamos los códigos que tienes que usar con `char()` para obtener cada *carácter gráfico*:



Sin contar los bordes, cada baldosa comprende tres líneas de pantalla y seis columnas. Si la baldosa está vacía, el fondo es negro, pero si hay una ficha, los 6 x 3 caracteres se mostrarán con un color de fondo que dependerá del valor de la ficha. El valor de la ficha se mostrará en la fila del medio, con un espacio a ambos lados y una anchura de campo de 4.

Al dibujar el tablero, obviamente por filas, ten en cuenta que las esquinas superiores (códigos 218, 194 y 191) e inferiores (códigos 195, 197, 180, 192, 193 y 217) usan caracteres distintos de los bordes interiores (código 196 y 179). Ten en cuenta también que los bordes interiores horizontales (código 196) se repiten seis veces y los bordes interiores verticales (código 179) se repiten tres veces.

Colores de fondo en la consola

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. En nuestro programa el color de primer plano siempre es blanco, pero queremos poder cambiar el color de fondo para representar cada ficha en un color diferente. Podemos cambiar esos colores

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:

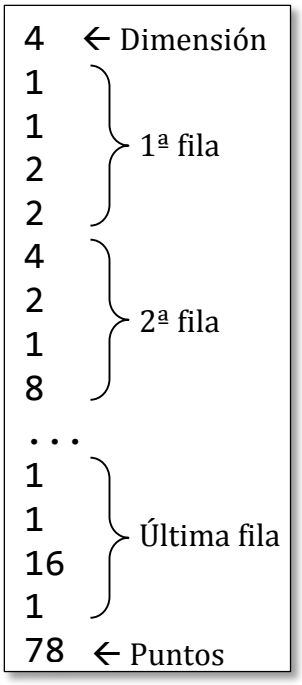
```
void colorFondo(int color) {
   HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(handle, 15 | (color << 4));
}
```

Basta proporcionar un color para el fondo (0 a 15) y esa rutina lo establecerá, con el color de primer plano siempre en blanco (15).

Para cada ficha, utilizaremos como color de fondo la potencia de 2 de su valor, por ejemplo, para las baldosas con valor 2 usaremos como color de fondo el 1 (azul), para las baldosas con valor 4 el 2 (verde)… Debes cambiar el color de fondo cada vez que tengas que *dibujar* parte de una ficha y volverlo a poner a negro (0) a continuación.

Carga de partidas

También queremos poder cargar tableros de partidas a medias. Como podemos cambiar la dimensión del tablero para jugar con tableros más grandes (6x6, 10x10,...), los archivos de tableros deben comenzar con una línea que indique cuál es esa dimensión (DIM). El tablero sólo se cargará si esa dimensión coincide con la dimensión actual de tablero en el programa. Si es así, simplemente se leerán DIMxDIM enteros y se colocarán, por filas, en el tablero. Y, como necesitamos los puntos alcanzados hasta ese momento en la partida, el archivo termina con esos puntos. No hay centinela (no se necesita). A la derecha puedes ver un ejemplo de archivo con los datos de una partida.



Implementación

Para esta primera versión crea una función `main()` que comience iniciando el generador de números aleatorios y preguntando al usuario si quiere cargar una partida; si es así, pedirá el nombre del archivo y cargará la partida. Si no se quiere cargar una partida, o no se ha podido, inicializará el tablero. Visualizará el tablero y terminará solicitando al usuario que pulse la tecla `Enter` para a continuación cerrarse.

Implementa, al menos, los siguientes subprogramas:

- ✓ `void inicializa(tTablero tablero)`: Inicializa el `tablero`, haciendo que todas las baldosas estén libres, excepto dos elegidas aleatoriamente. Esas dos baldosas contendrán fichas con valor 2 o 4, siendo la probabilidad de ser un 2 del 95% y no pudiendo ser ambas 4.
- ✓ `void visualiza(const tTablero tablero, int puntos, int totalPuntos)`: Muestra el `tablero`, con los `puntos` de la última jugada y el total de puntos.
- ✓ `int log2(int num)`: Devuelve la potencia de 2 que da como resultado `num` ($1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$, ..., $2048 = 2^{11}$). Por ejemplo, si num es 8 log2 devolverá un 3.
- ✓ `void carga(tTablero tablero, int &puntos, bool &ok)`: Pide un nombre de archivo e intenta cargar en el `tablero` los datos de archivo, así como los `puntos`. Si no se puede cargar el archivo, inicializa el `tablero` y pone a cero los `puntos`.

2.2. Versión 2: Inclinación del tablero

Vamos a añadir a la versión 1 la posibilidad de *inclinar* el tablero y hacer que las fichas *caigan* en esa dirección, cubriendo los huecos libres para que todas las fichas de cada fila o columna queden juntas hacia el borde correspondiente. Para ello, tras mostrar el tablero, el programa solicitará al usuario la dirección en la que desea *inclinarlo*. El usuario introducirá dicha dirección con las teclas de flecha. El tablero se inclinará repetidas veces hasta que el usuario pulse *Esc*, momento en que el programa terminará.

Lectura de teclas especiales

La tecla Esc genera un código ASCII (27), 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. Si se trata de una tecla especial habrá que llamar dos veces a `_getch()`. Esta función requiere que se incluya la biblioteca `conio.h`. El código para leer las teclas especiales sería:

```
cin.sync();
dir =_getch(); // dir: tipo int
if (dir ==  0xe0) {
   dir = _getch();
   // ...
}
// Si no, si dir es 27, es la tecla Esc
```

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 ↓ 80 → 77 ← 75

Implementación

Modifica la función `main()` de la primera versión para que después de mostrar el tablero inicial lo vaya inclinando tantas veces como desee el usuario. Implementa, al menos, los siguientes subprogramas:

- ✓ `tAccion leeAccion()`: Devuelve un valor de dirección distinto de `Nada`. `Salir`, si se pulsa la tecla Esc, o la dirección en la que *inclinar* el tablero. Esta función usará `_getch()` para detectar las teclas de flecha (o `Esc`). Se seguirán leyendo teclas mientras éstas sean teclas normales o especiales distintas de las de flecha o Esc.
- ✓ `void mueveFichas(tTablero tablero, tAccion accion)`: Desplaza las fichas del `tablero` en la dirección indicada por `accion`.

Además, modifica el subprograma de visualización para que termine mostrando el mensaje `Usa las teclas de flecha (Esc para salir)...`

2.3. Versión 3: Juego completo

En esta última versión añadimos todo lo necesario para poder jugar al 2048 tal y como se explicó en la Sección 1. Ahora una vez que el usuario indica la dirección en la que *inclinar* el tablero, tenemos que realizar dos tareas: desplazar las fichas en la dirección y realizar las posibles combinaciones de fichas. Tras cada jugada debe aparecer una nueva ficha, en una baldosa libre, elegida aleatoriamente, de valor 2 o 4 (con una probabilidad del 95% de ser un 2 y una probabilidad del 5% de ser un 4). Ahora el programa debe terminar si:

- El usuario pulsa Esc.
- El usuario consigue una baldosa con el valor 2048. En este caso el usuario gana.
- El tablero se llena, no queda ninguna baldosa vacía, y no se puede realizar ninguna combinación. En este caso el usuario pierde.

Implementación

Modifica la función `main()` de la versión anterior para poder jugar una partida completa de 2048. Implementa, al menos, los siguientes subprogramas:

- ✓ `int mayor(const tTablero tablero)`: Devuelve el valor mayor encontrado en las fichas del `tablero`. Si ese valor es 2048, entonces se ha ganado el juego.
- ✓ `bool lleno(const tTablero tablero)`: Devuelve `true` si no quedan baldosas libres en el `tablero`; `false` en otro caso.
- ✓ `void combinaFichas(tTablero tablero, tAccion accion, int &puntos)`: Combina las fichas del `tablero` que hayan quedado juntas tras haber movido el tablero en la dirección indicada por `accion`. Devuelve el total de puntos conseguido.

- ✓ `void nuevaFicha(tTablero tablero)`: Crea, en una posición del `tablero` elegida aleatoriamente y que esté libre, una nueva ficha (con un 95% de probabilidades de que sea un 2 y un 5% de que sea un 4).

2.4. Versión 4: Mejores puntuaciones y salvado de partidas (opcional)

Modifica la versión 3 para que el programa mantenga un archivo con las hasta diez mejores puntuaciones de los que hayan conseguido ganar el juego (llegar a 2048).

El archivo `records.txt` tiene un nombre (sin espacios) y una puntuación, separados por un espacio, en cada línea. Termina con ??? como nombre (centinela). Ejemplo:

```
Luis 21763
Ana 20967
Eva 20682
???
```

Además, queremos poder guardar partidas en archivos.

Implementación

Añade, al menos, los siguientes subprogramas:

- ✓ `void nuevoRecord(int puntos)`: Carga del archivo de mejores puntuaciones (`records.txt`) una lista de nombres y puntuaciones, pide al usuario su nombre, inserta en la lista el nombre y los puntos del nuevo record, y finalmente guarda en el archivo la lista actualizada (hasta diez records; si ya había diez, el de menor puntuación desaparece de la lista).
- ✓ `void muestraRecords()`: Muestra el contenido del archivo `records.txt`.
- ✓ `void guarda(const tTablero tablero, int puntos)`: Pide un nombre de archivo y guarda en él la dimensión del `tablero`, su contenido (por filas) y los `puntos` conseguidos hasta ese momento.

Actualiza la función `main()` de la versión 3 para que cuando un jugador gane una partida se actualice el archivo de mejores puntuaciones con la suya y a continuación muestre las mejores puntuaciones al usuario. Además, cuando el usuario pulse la tecla Esc, el programa le dará la oportunidad de guardar la partida antes de salir.

3. Requisitos de implementación

No olvides declarar las constantes necesarias para hacer tu código reutilizable y fácil de modificar. No olvides incluir los prototipos de tus funciones.

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

No pueden usarse en la resolución de la práctica elementos de C++ no vistos en clase. El programa no debe utilizar instrucciones de salto no estructuradas como `exit`, break (salvo en las cláusulas de la instrucción switch) y `return` (salvo en la última instrucción de las funciones que devuelven un valor).

4. Entrega de la práctica

La práctica se entregará a través del Campus Virtual. Se habilitará una nueva tarea: **Entrega de la Práctica 3** que permitirá subir un archivo con el código fuente de la última versión. El fichero subido deberá tener el siguiente nombre: Practica3GrupoXXX.cpp, siendo XXX el número de grupo. Uno sólo de los miembros del grupo (no los dos) será el encargado de subir la práctica.

La práctica debe funcionar usando Visual Studio, que es la herramienta vista en clase. La práctica subida debe compilar y cumplir con la funcionalidad descrita en el enunciado.

No se corregirá ninguna práctica que no cumpla estos requisitos.

Fin del plazo de entrega: **29 de marzo a las 23:55**