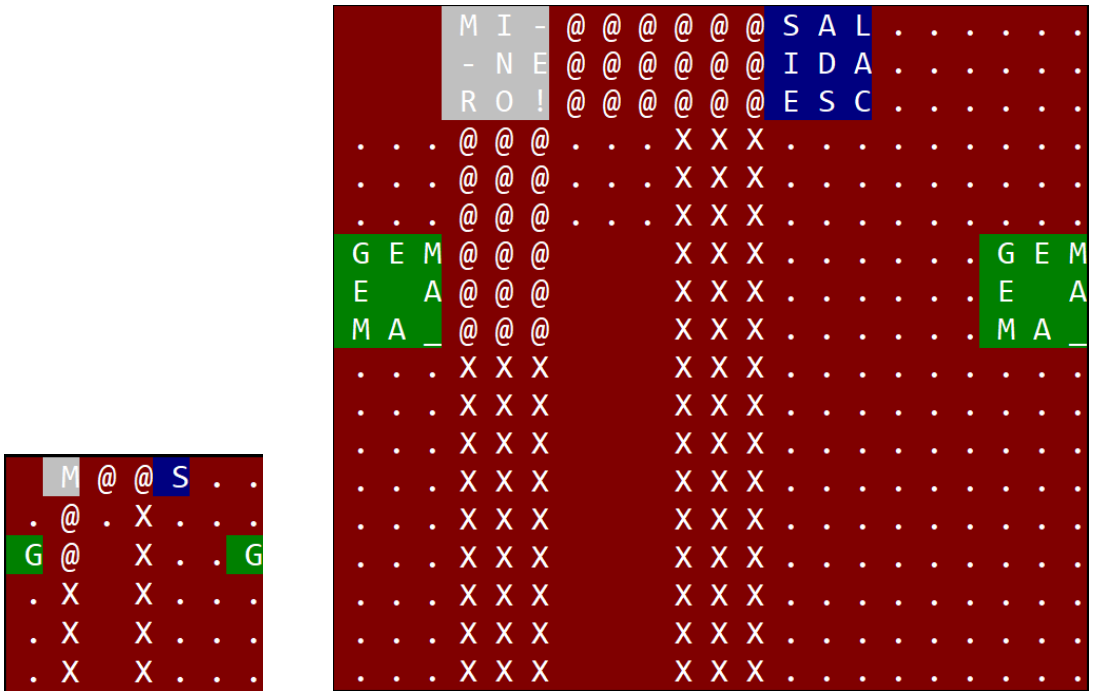
Práctica; primera parte
Master miner

Fecha de entrega: 13 de abril

1. Descripción del juego

La práctica consiste en desarrollar un programa en C++ que permita jugar a una versión del **Master Miner** (https://en.wikipedia.org/wiki/Master_Miner)**.** Este juego, creado por Dan Illowsky, trata de un minero cuyo objetivo es recoger gemas. En nuestra versión, cuantas más gemas recoja, más puntos acumula, no siendo necesario que recoja todas las gemas para pasar de nivel. El objetivo del juego es recolectar el máximo número de gemas y luego dirigirse a la casilla de salida.
El juego se desarrolla sobre un plano de la mina que representa una mina cortada en vertical. A continuación, se muestra un posible plano de la mina.



Planos de la misma mina a escala 1:1 y 1:3

El minero puede moverse dentro del plano hacia la derecha o la izquierda manteniendo su nivel con respecto a la profundidad de la mina, o hacia arriba o hacia abajo cambiando el nivel de profundidad dentro de la mina. Si el minero se desplaza en horizontal, aunque

la casilla de debajo esté excavada mantiene su nivel, no se cae. El minero no puede atravesar muros, ni piedras.
Al realizar un movimiento el minero se desplaza a la casilla contigua, según el movimiento. El movimiento se puede llevar a cabo si la casilla contigua está vacía (excavada anteriormente), o con tierra, o si tiene una gema. De hecho, si tiene una gema la recolecta y ocupa la casilla. Si la casilla a la que quiere ir tiene una piedra, entonces debe empujarla para ocupar ese sitio. Una piedra puede moverse si la casilla a la cual se va a desplazar está vacía.

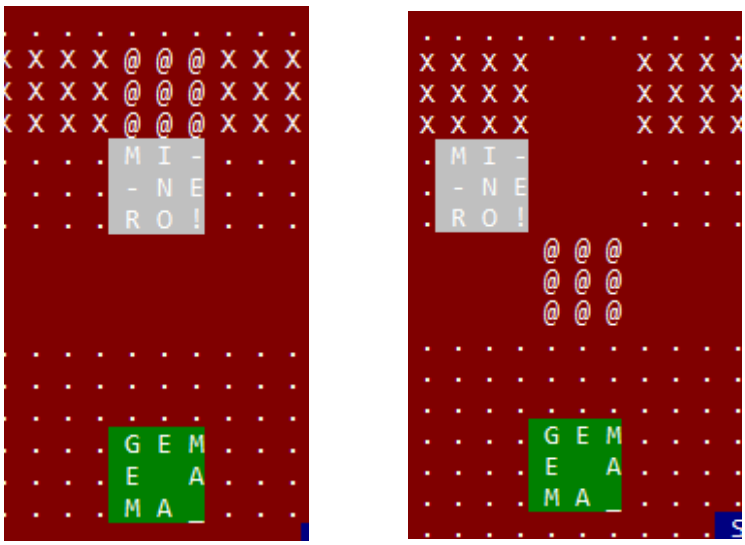
Las casillas pueden contener:

a. Al minero

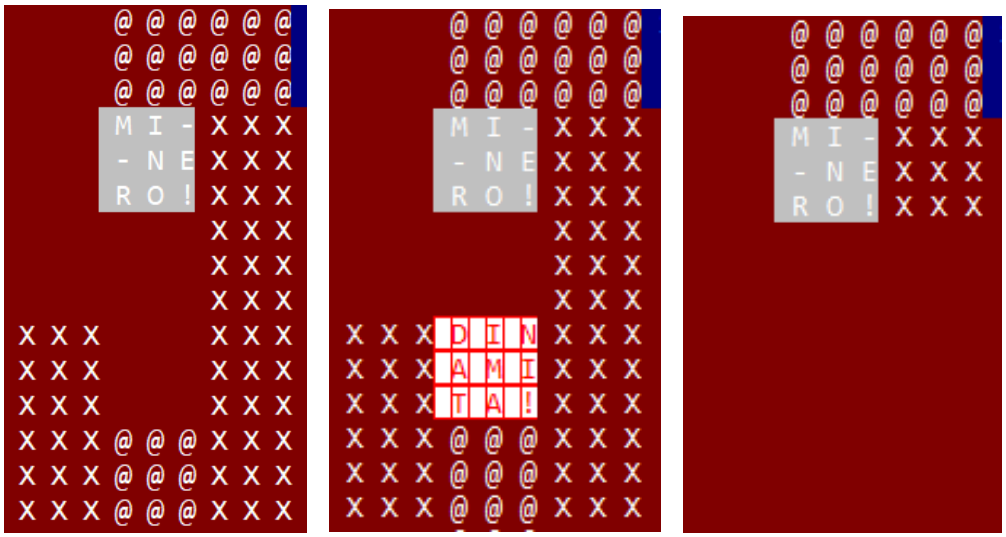
b. Muro. El muro no se puede atravesar, ni empujar para ocupar esa casilla. La única forma de destruir un muro es con dinamita.

c. Piedra. Las piedras impiden el paso del minero. Las puede empujar en horizontal si la casilla siguiente está vacía. Si no está vacía (tierra, muro, gema u otra piedra) no se puede desplazar. Para desplazarla debe avanzar hacia ella.

Las piedras pueden caer hacia abajo por efecto de la gravedad cuando la casilla que tienen debajo está vacía. Para que la casilla de abajo quede vacía, el minero debe excavar la tierra que hay, quedando él debajo de la piedra, pero esta no le aplasta (es un minero muy fuerte), cuando se mueve de debajo de la piedra, la casilla queda libre y la piedra cae tantas casillas como haya libres. Si la piedra tiene otras piedras o gemas encima, entonces estás caen con ella. Si el minero se mueve hacia abajo teniendo una o varias piedras encima, éstas caen encima del minero, pero no le aplastan. Si cae sobre cualquier otro elemento, como la tierra o un muro, queda sobre dicho elemento.



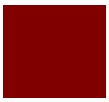
d. Dinamita. La dinamita la lleva el minero y cuando decide soltarla cae en vertical por acción de la gravedad, al llegar abajo explota. La explosión provoca que las 8 casillas que hay alrededor queden vacías. Es la forma de eliminar muros y piedras. Si explota afectando a una gema ésta se elimina, y si explota afectando al minero éste muere. El minero tiene todas las dinamitas que necesite.



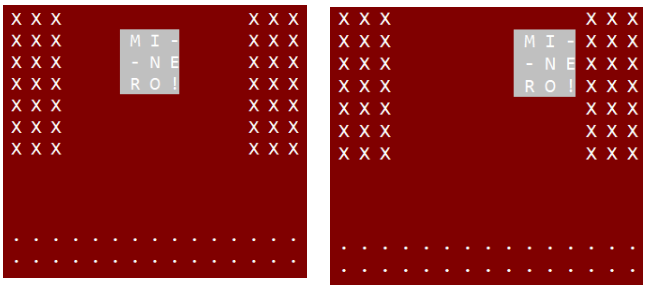
e. Gema. Cuando el minero llega a una casilla que tiene una gema la recolecta y la casilla queda vacía. Las gemas bloquean el movimiento de piedras, es decir, no se puede mover una piedra si detrás hay una gema.

f. Salida. Hay una única salida a la que hay que llegar para pasar al siguiente nivel.

g. Libre. es una casilla que pertenece a un túnel ya excavado o a un hueco que está desde el inicio de la partida.



El minero puede moverse en todas las direcciones, sin caerse al vacío.



El juego termina con éxito cuando el minero ha llegado a la casilla de salida. El juego termina con fallo si la explosión de una dinamita alcanza al minero o porque se ha quedado encerrado entre piedras o muros y no puede alcanzar la salida y debe pulsar la tecla de escape para finalizar.

2. El programa

El programa simulará la dinámica del juego en modo de consola. Inicialmente mostrará un menú con tres opciones:

```
1. Jugar partida a escala 1:1.
2. Jugar partida a escala 1:3.
0. Salir
```

Las opciones 1 y 2 permiten jugar una partida. La opción 1 muestra el plano de la mina en escala 1:1, mientras que la opción 2 lo muestra a escala 1:3 como se explica en la subsección 2.3. La partida se inicia cargando el primer nivel desde el fichero “1.txt”, si logra alcanzar la salida de este nivel, tendrá acceso al siguiente nivel definido en el fichero “2.txt”, y así sucesivamente hasta alcanzar el último nivel, cuando se supere este

nivel aparecerá un mensaje de "Game Over". El último nivel viene definido como una constante del programa.

Se dará la opción de introducir los movimientos desde teclado o introducirlos utilizando un fichero, en este caso, se le pedirá al usuario el nombre de este fichero. Se podrán jugar partidas hasta que se elija la opción 0. Para ello se dará un fichero que contiene una letra por movimiento. La letra 'D' es la dinamita, 'M' es derecha, 'N' izquierda, 'A' arriba y 'Z' abajo. Esta opción es útil para hacer pruebas.

2.1 Datos del programa

Define un tipo enumerado `tCasilla` que permita representar al menos los diferentes elementos que puede haber en la mina: `LIBRE, TIERRA, GEMA, PIEDRA, MURO, SALIDA, MINERO, DINAMITA.`

Para mantener el estado de la mina el programa usa un array bidimensional de tipo `tCasilla`. Declara la correspondiente constante `MAX=50` y el tipo `tPlano` para representar el array bidimensional.

El programa usa también un tipo enumerado `tTecla` para representar los movimientos que puede realizar el minero: `ARRIBA, ABAJO, DCHA, IZDA, SALIR, NADA` y `TNT.`

Define el tipo estructurado `tMina` para describir el estado completo de la mina, conteniendo al menos:

- ✓ El array bidimensional de tipo `tPlano`.
- ✓ El número de filas `nfilas` y de columnas `ncolumnas` del array bidimensional (ambas <= `MAX`).
- ✓ La `fila` y la `columna` donde se encuentra colocado el minero.

Define también el tipo estructurado `tJuego` que describe el estado del juego conteniendo al menos:

- ✓ El estado de la `mina (plano y posición del minero)` de tipo `tMina`.
- ✓ El contador de gemas recolectadas desde que se inició la partida.
- ✓ El número de movimientos
- ✓ El número de dinamitas que utiliza

2.2 Cargar el plano de la mina

Los planos de juego de los distintos niveles se leerán de un archivo de texto.

Por ejemplo, el plano del ejemplo anterior corresponde al archivo `1.txt`:

```
7 10
GM J MMMMM
GM   MMTGT
TM G MMTTT
GTTG  MTTT
TTTTTTTTTT
TGTTMGTMST
TTTTMTTMTT
```

La primera línea define el número de filas y columnas de la mina. A continuación aparece una matriz de caracteres, donde ' ' (espacio en blanco) representa un espacio 'LIBRE', es decir un hueco en la mina. 'T' representa la tierra, 'G' una gema, 'P' una piedra, 'M' un muro, 'S' la salida y 'J' al jugador o minero. Distintos archivos con distintos niveles se dejarán en el Campus Virtual.

2.3 Visualización del plano de la mina

El plano de la mina se puede visualizar en dos escalas: 1:1 y 1:3. Al comienzo de una partida el usuario elige si quiere utilizar una escala u otra. En la escala 1:1 cada casilla se corresponde con un carácter en la consola, mientras que en la escala 1:3 cada casilla se corresponde con una matriz de 3*3 caracteres de la consola.

Cada vez que se vaya a visualizar el estado del plano, primero se borra el contenido de la ventana de consola, de forma que siempre se muestre el plano en la misma posición y la sensación sea más visual. Para borrar la consola utiliza: `system("cls");.`

En la proyección 1:1 se utilizan los siguientes caracteres para representar los datos de la mina: '@' piedra, 'X' muro, '.' Tierra, 'M' minero, 'D' dinamita, 'G' gema y 'S' salida.

La proyección a escala 3:3 agranda en 3 unidades el plano original y solo se utiliza en el momento de visualizar el juego. Los caracteres utilizados pueden ser los mismos que en la proyección 1:1 repetidos 9 veces o se puede utilizar algún diseño original. Por ejemplo:

Para llevar a cabo la proyección se deben crear las siguientes matrices que se utilizarán conjuntamente para visualizar la mina. Usaremos las definiciones de tipos `tPlanoCaracteres` y `tPlanoColores` como tipos definidos de

`char planoCaracteres[3*MAX][3*MAX]:` esta matriz guarda los caracteres por cada casilla.

`int planoColores[3*MAX][3*MAX]:` esta matriz guarda el color de cada casilla.

Esta proyección deja la mina un poco "estrecha", si queremos ampliar la anchura podemos aplicar setw de la siguiente forma:

```
cout << setw(2) << planoCaracteres[_][_];
```

2.4 Poner color al plano de la mina.

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.

2.5 Lectura del movimiento del minero

El minero se mueve en horizontal y vertical utilizando las flechas, para lanzar la dinamita se utiliza la tecla "D" y para salir en cualquier momento del juego se utiliza la tecla de escape.

Para facilitar las pruebas se pueden guardar los movimientos en un fichero de texto cuyo nombre se dará al comienzo del programa. Cuando se utilice esta opción las teclas del movimiento serán: A arriba, Z abajo, N izquierda, M derecha y D la dinamita.

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 `Salir`, si se pulsa la tecla Esc; 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) y la tecla Esc (salir). La tecla Esc sí 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 y se puede llamar una segunda vez para obtener el segundo código, si se trata de 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 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:

Flecha arriba 72, abajo 80, derecha 77, e izquierda 75.

3. Movimiento del minero

Una vez que el usuario indica la dirección, tenemos que realizar el movimiento del jugador sobre el plano. Para esto, implementa la siguiente función:

- ✓ `bool hacerMovimiento(tJuego &juego, tTecla tecla)`: realiza el movimiento del minero en la dirección indicada. Esta función define este nuevo estado de la mina teniendo en cuenta el movimiento del minero. Por ejemplo, si intenta desplazarse a una casilla que contiene un muro, como no puede picarlo se queda dónde está sin realizar ninguna acción. Por otra parte, si se quiere desplazar a una casilla que contiene una piedra, lo primero a hacer es comprobar que detrás de la piedra no hay nada para poder desplazar la piedra. Una vez que la piedra está desplazada, hay que comprobar si cae al vacío por acción de la gravedad.

4. Módulos a implementar

3.1 Módulo mina

En este módulo se define el tipo `tCasilla`, el tipo `tPlano`, y el tipo `tMina`. Para implementar la funcionalidad de la mina debes utilizar al menos las siguientes funciones:

- ✓ `void cargar_Mina(ifstream& fichero, tMina& mina)`: lee los datos de un fichero y guarda la posición del minero.
- ✓ void `dibujar1_1`(const tMina& mina): dibuja la mina a escala 1:1.
- ✓ void `dibujar1_3`(const tMina& mina): dibuja la mina a escala 1:3 y usa la siguiente función `dibuja3x3`.
- ✓ `void dibuja3x3(tCasilla casilla, tPlanoCaracteres caracteres, tPlanoColores colores, int i, int j)`: dibuja las casillas aumentadas tres veces. En concreto, la casilla sirve para actualizar el plano de caracteres y colores en las coordenadas i, j.

3.2 Módulo del juego

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

- ✓ `bool cargar_Juego(tJuego& juego, int nivel)`: carga el juego del nivel indicado, es decir, inicializa el juego y carga la mina. También debe gestionar si el fichero de carga no está disponible. El nivel del juego coincide con el nombre del fichero. En el Campus Virtual se dejarán unos ficheros de carga llamados: "1.txt", "2.txt", "3.txt" y "4.txt".
- ✓ `bool hacerMovimiento(tJuego& juego, tTecla tecla)`: cuando el minero se mueve provoca que el estado de la mina cambie y el contador de movimientos. Posiblemente el contador de gemas y el número de dinamitas cambie también. Cualquier acción del minero cuenta como movimiento, incluidas dejar caer la dinamita y no poder avanzar por tener delante un muro o piedra.
- ✓ `void dibujar(const tJuego& juego):` muestra el estado de la mina teniendo en cuenta la escala. Además, debe mostrar el número de gemas recogidas hasta el momento, el número de movimientos realizados y las dinamitas que ha usado el minero.

5. Navegación

El programa principal mostrará el menú

```
1. Jugar partida a escala 1:1.
2. Jugar partida a escala 1:3.
3. Salir
```

y leerá la opción seleccionada por el usuario. A continuación se pide si el usuario utilizará las teclas o si prefiere que la computadora le los movimientos de un fichero.

```
1. Introducir movimientos por teclado.
2. Introducir movimientos por fichero.
0. Salir
```

Cuando se completa un nivel, se le ofrece la posibilidad de ir al siguiente nivel o abandonar.

```
1. Jugar siguiente nivel.
0. Salir.
```

6. Programa principal

El programa principal mostrará el menú y leerá la opción seleccionada por el usuario.

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 de FP2, primera parte**, que permitirá subir archivos `.cpp` y `.h` con el código fuente. Uno de los dos miembros del grupo, y sólo uno, será el encargado de subirlo, no lo suben los dos.

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

Fin del plazo de entrega: **13 de abril a las 23:55 horas**