Facultad de Informática – Universidad Complutense de Madrid

Fundamentos de Programación II – 2024/25

Proyecto Buscaminas (Minesweeper) - versión 1

Fecha de entrega: 31 de marzo de 2025 a las 9:00

INSTRUCCIONES - leed con atención

1. Escribid vuestros nombres y apellidos al principio del código.
2. Debe entregarse a través del Campus Virtual en **Entrega práctica (versión 1)**.
3. Se entregarán únicamente los ficheros de cabeceras `.h` y los ficheros fuente `.cpp` en un archivo comprimido de tipo `zip`.
4. El proyecto no puede tener errores de compilación.
5. Sigue los pasos que aparecen en el enunciado y ten en cuenta todas las indicaciones.
6. Aunque el proyecto se debe hacer de manera incremental, tal y como aparece en el enunciado, este debe leerse completamente antes de empezar.
7. En el Campus Virtual se encuentra un archivo `zip` con ficheros que puedes utilizar para ayudarte a dibujar los tableros, además de algunos tableros de ejemplo.

Primero resuelve el problema. Entonces, escribe el código.

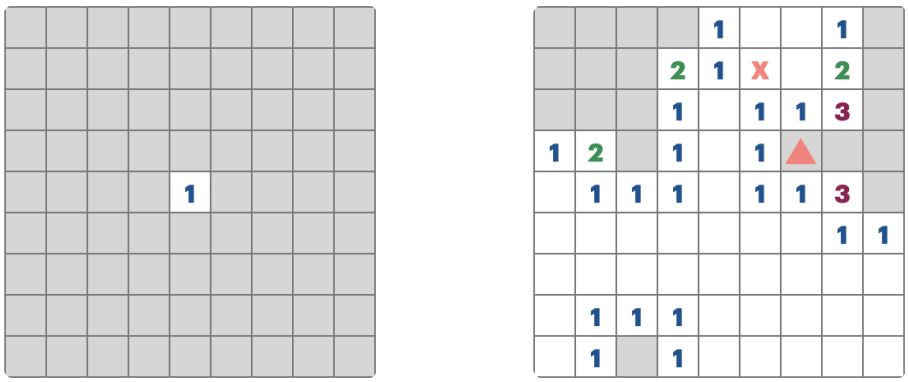
— John Johnson

El juego

El objetivo de este proyecto es desarrollar un programa que permita jugar a una versión del juego Buscaminas: `https://minesweeper.online/es/`. Según la información de wikipedia (`https://es.wikipedia.org/wiki/Buscaminas`), el origen de Buscaminas no está claro. Según *TechRadar*, la primera versión del videojuego fue *Microsoft Minesweeper* de 1990, pero *Eurogamer* dice que *Mined-Out* de Ian Andrew (1983) fue el primer videojuego de Buscaminas. Curt Johnson, el creador de *Microsoft Minesweeper*, reconoce que el diseño de su videojuego fue tomado prestado de otro videojuego, pero no era *Mined-Out*, y no recuerda de qué videojuego se trata.

El Buscaminas es un juego clásico de ordenador para un solo jugador cuyo objetivo es despejar un campo minado sin detonar ninguna mina. El juego consiste en una cuadricula con celdas, donde algunas contienen minas y otras están libres. El jugador selecciona celdas, una a una, para ver lo que contienen. En caso de seleccionar una celda con mina, pierde el juego. En otro caso, la celda puede estar vacía o contener un número. Si está vacía significa que ninguna de sus ocho celdas vecinas contiene una mina, y por lo tanto se descubren sus ocho celdas vecinas. Si la celda contiene un número, este número indica el número de minas que la rodean, y por tanto no se descubre ninguna de sus posiciones vecinas. En el juego original del Buscaminas, una vez descubierta una celda, este proceso se repite sobre las celdas vecinas que están vacías, hasta que no se puedan descubrir más celdas. También es posible "marcar" aquellas celdas sospechosas de contener una mina.

En la siguiente figura, en la izquierda aparece un tablero donde la celda abierta está rodeada por una mina, por lo tanto no se descubre ninguna celda vecina. En la figura de la derecha, la celda elegida es la marcada con "X". En este caso la celda está vacía y se descubren sus ocho celdas vecinas. De las ocho vecinas, para aquellas que están vacías, se descubren también sus ocho celdas vecinas y así sucesivamente hasta que no se puedan descubrir más. Para seleccionar la siguiente celda se obtiene información de las celdas descubiertas. Por ejemplo, se sabe que la celda marcada con ▲ contiene una mina con toda seguridad, por lo tanto no debe elegirse.



Para ganar una partida del Buscaminas, todas las celdas que no sean minas deben abrirse sin destapar ninguna mina. No hay puntuación, pero se registra el número de pasos que se tarda en terminar la partida.

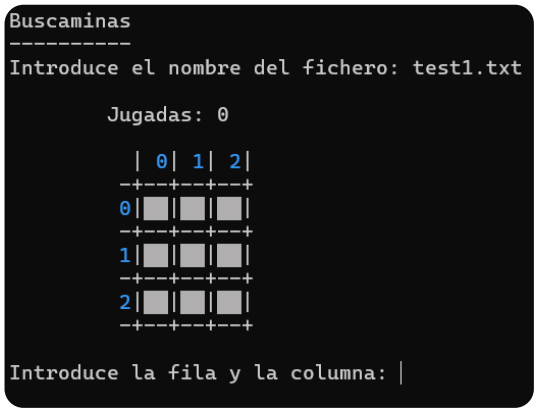
Nosotros vamos a comenzar con una **versión simple** del Buscaminas, en la que únicamente, una vez elegida una celda, si está vacía, se muestra la información de sus ocho casillas adyacentes. En versiones posteriores implementaremos el juego original.

Jugando al Buscaminas

Comencemos a desarrollar una aplicación que permita jugar al Buscaminas. Al comienzo, el usuario elegirá un archivo que contendrá la configuración inicial del tablero. El archivo tendrá un formato similar al siguiente:

```
3 3
1
0 0
```

donde la primera línea contiene las dimensiones del tablero (3 filas y 3 columnas), la segunda línea es el número de minas que contiene el tablero, y el resto de líneas son las posiciones en las que se encuentran colocadas las minas (fila y columna). El tablero se cargará y se mostrará en la consola de la siguiente manera:

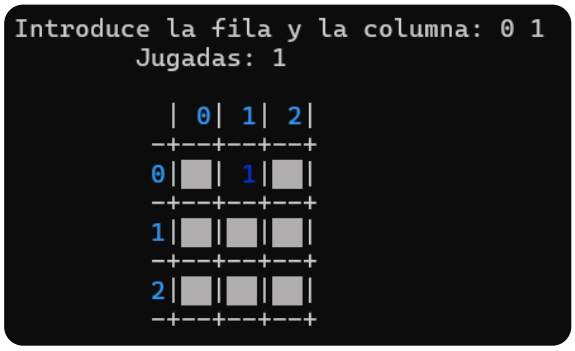


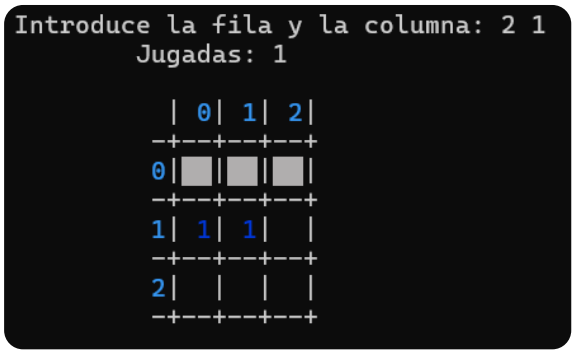
Las celdas blancas se corresponden con celdas ocultas (no visibles), y son las celdas que se podrán elegir para ser descubiertas. El juego discurre solicitando al usuario la elección de una posición, denotándola mediante una fila y una columna. Dependiendo de los valores introducidos se realizará una acción u otra. Concretamente:

- Si `fila = -1, columna = -1`, entonces el juego termina.
- Si `fila = -2, columna = -2`, se "**marca**" la celda que indique el jugador. Más tarde explicaremos su funcionamiento.
- Si `fila = -3, columna = -3`, se realiza una operación de "**undo**", volviendo el tablero a su configuración anterior. Posteriormente explicaremos como funciona la operación de *undo*.
- En otro caso, se ejecuta un paso en el juego. Si `fila` y `columna` no están dentro de las dimensiones del tablero, no se hace nada. Si son correctas, tenemos las siguientes posibilidades:
  - La celda ya está descubierta. Entonces no se hace nada.
  - La celda contiene una mina. Entonces el juego termina.
  - La celda está oculta y contiene un número. Entonces simplemente se descubre esa celda mostrando el número que contiene.
  - La celda está oculta y está vacía. Entonces se descubre la celda y sus ocho celdas vecinas.

Ejecución de un paso

Si partimos del tablero inicial mostrado con anterioridad, la figura de la izquierda corresponde a la elección de la posición `(0,1)`, donde, al ser una celda oculta con número, se descubre pero no se descubren sus celdas vecinas. En la figura de la derecha elegimos la posición `(2,1)`, que al ser una posición oculta sin minas a su alrededor (está vacía), se descubre, pero además también se descubran todas sus celdas vecinas.

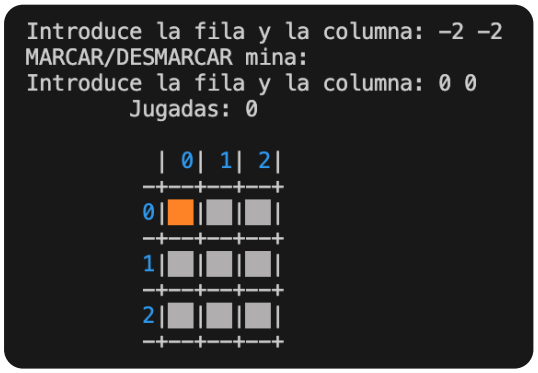


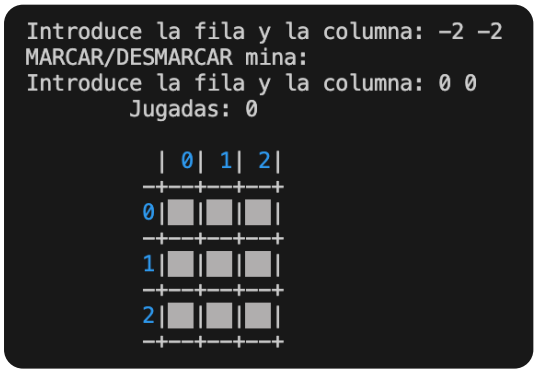


Marcar/Desmarcar minas

Existe además la posibilidad de marcar/desmarcar una celda oculta, para indicar que puede contener una mina. Para ello basta introducir la posición `(-2,-2)`. Entonces el juego nos solicitará la celda sobre la que realizar la operación de marcar/desmarcar. Hay dos posibilidades. Si la celda está oculta y no marcada, entonces sobre ella aparece una marca. Si la celda está marcada, entonces se desmarca. En cualquier otro caso no se hace nada.

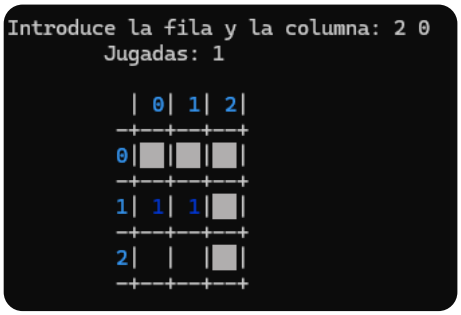
Partiendo de nuestro tablero original, la figura de la izquierda muestra una marca sobre la celda `(0,0)`, mientras que la figura de la derecha posteriormente la desmarca.

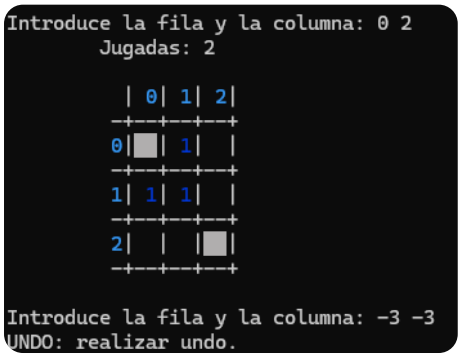


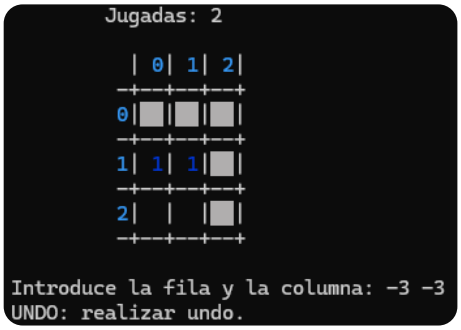


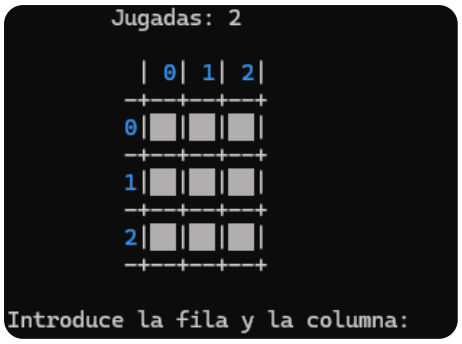
Realizar un "*undo*"

Para realizar una operación de "*undo*" debe introducirse la posición `(-3,-3)`. En este caso se vuelve al tablero anterior. Siempre será posible deshacer los últimos `MAX_UNDO` movimientos realizados en el tablero. La operación de "*undo*" no afecta a las celdas marcadas, que en caso de haberlas seguirán marcadas tras la operación. Esto es así porque las celdas se pueden marcar/desmarcar de forma manual. Los gráficos siguientes, de izquierda a derecha y de arriba a abajo, muestran el efecto de dos operaciones "*undo*".









¡Vamos a implementar nuestro propio juego!

Detalles de implementación del Buscaminas

El desarrollo de este proyecto se debe realizar con un **enfoque modular**. Esto facilita la ampliación y el mantenimiento del sistema más adelante, por ejemplo, cuando desarrollemos la *versión* 2 del mismo juego.

En las descripción de las clases sólamente especificamos el nombre de los subprogramas y parámetros. No aparece ni el tipo devuelto, ni la especificación de E/S de los parámetros. Serás tú quien debas incluirlos de forma adecuada siguiendo las indicaciones del enunciado.

Módulo `celda`

Encapsula la información de una celda. Una celda puede estar vacía, tener una mina o contener un número. Utiliza el tipo enumerado `typedef enum { NUMERO, VACIA, MINA } tEstado` para representar el estado de una celda.
El módulo celda tendrá al menos las siguientes estructuras y subprogramas:

```
typedef struct {
  bool visible;
  tEstado estado;
  int numero;
  bool marcada;
} tCelda;

inicializar(celda);
dame_estado(celda);
dame_numero(celda);
es_visible(celda);
```

```
es_mina(celda);
esta_vacia(celda);
contiene_numero(celda);
esta_marcada(celda);
descubrir_celda(celda);
ocultar_celda(celda);
poner_mina(celda);
marcar_celda(celda);
desmarcar_celda(celda);
poner_numero(celda, num);
```

- `inicializar(celda)` inicializa la celda a `VACIA`, no visible, sin marcar y con anotación numérica a 0.
- `dame_estado(celda)` devuelve el estado de la celda.
- `dame_numero(celda)` devuelve el número de la celda.
- `es_visible(celda)` dada una celda devuelve `true` si su contenido es visible.
- `es_mina(celda)` dada una celda devuelve `true` si esta tiene una mina.
- `esta_vacia(celda)` dada una celda devuelve `true` si esta está vacía.
- `contiene_numero(celda)` dada una celda devuelve `true` si tiene número asociado.
- `esta_marcada(celda)` dada una celda devuelve `true` si está marcada.
- `descubrir_celda(celda)` hace la celda visible.
- `ocultar_celda(celda)` oculta la celda.
- `poner_mina(celda)` cambia la celda a mina.
- `marcar_celda(celda)` marca la celda.
- `desmarcar_celda(celda)` desmarca la celda.
- `poner_numero(celda, num)` pone el estado de la celda a `NUMERO` y su numero al valor `num`.

Módulo `tablero`

Esta unidad funcional representa los tableros como matrices bidimensionales. Declara el tipo de datos `tTablero` para la gestión de la matriz bidimensional, con al menos las siguientes operaciones:

```
typedef struct {
  int nFils, nCols;
  tCelda datos[MAX_FILS][MAX_COLS];
} tTablero;

inicializar(tablero);
inicializar(tablero, nfils, ncols);
num_filas(tab);
num_columnas(tab);
```

```
dame_celda(tablero, fila, columna);
es_valida(tablero, fila, columna);
poner_celda(tablero, fila, columna, celda);
```

El comportamiento de las operaciones es el siguiente:

- `inicializar(tablero)` inicializa un tablero vacío, es decir, sin celdas.
- `inicializar_tablero(tablero, nfils, ncols)` inicializa un tablero de dimensión `nfils x ncols` con todas las celdas vacías.
- `num_filas(tab)` devuelve el número de filas del tablero.
- `num_columnas(tab)` devuelve el número de columnas del tablero.
- `dame_celda(tablero, fila, columna)` devuelve la celda contenida en la posición `(fila,columna)`, que supone correcta.
- `es_valida(tablero, fila, columna)` devuelve `true` si, y sólo si, la posición `(fila,columna)` es una posición válida dentro del tablero.
- `poner_celda(tablero, fila, columna, celda)` asigna el valor `celda` a la posición `(fila,columna)`, que supone correcta.

Módulo `juego`

Este módulo es el más relevante en cuanto a lógica, de la aplicación. Define el tipo de datos `tJuego` que representa la configuración actual de un juego, incluyendo el tablero, el número de jugadas, el número de minas y el número de celdas descubiertas (visibles). También se agrupan y establecen las *reglas del juego*.
Sobre el tipo de datos `tJuego` operan al menos los siguientes subprogramas (puedes utilizar tantas funciones y/o procedimientos privados como necesites para implementar las operaciones públicas):

```
typedef struct {
  tTablero tablero;
  int num_jugadas;
  bool mina_explotada;
  int num_minas;
  int num_descubiertas;
} tJuego;

inicializar(juego);
inicializar(juego, nfils, ncols);
dame_num_jugadas(juego);
dame_num_filas(juego);
dame_num_columnas(juego);
dame_num_minas(juego);
contiene_mina(juego, fila, columna);
es_visible(juego, fila, columna);
esta_marcada(juego, fila, columna);
esta_vacia(juego, fila, columna);
contiene_numero(juego, fila, columna);
```

```
dame_numero(juego, fila, columna);
esta_completo(juego);
mina_explotada(juego);
esta_terminado(juego);
poner_mina(juego, fila, columna);
marcar_desmarcar(juego, fila, columna);
ocultar(juego, fila, columna);
juega(juego, fila, columna, lista_pos);
```

Los campos `num_minas`, `num_descubiertas` y `num_jugadas` almacenan el número de minas, de celdas descubiertas y de pasos realizados en el juego. El campo booleano `mina_explotada` será `true` si ha explotado una mina, lo cual determinará que el juego ha finalizado.
En cuanto a la implementación, el comportamiento de las operaciones es el siguiente:

- **`inicializar(juego)`** crea un juego vacío, con el número de jugadas, minas y celdas descubiertas a `0`. Lógicamente `mina_explotada` debe ser `false`. El tablero se debe inicializar a vacío.
- **`inicializar(juego, nfils, ncols)`** crea un juego como en el procedimiento anterior. El tablero se inicializa con dimensión `fils x cols` y sus celdas vacías.
- **`dame_num_jugadas(juego)`** devuelve el número de jugadas del juego.
- **`dame_num_filas(juego)`** devuelve el número de filas del tablero del juego.
- **`dame_num_columnas(juego)`** devuelve el número de columnas del tablero del juego.
- **`dame_num_minas(juego)`** devuelve el número de minas del juego.
- **`contiene_mina(juego, fila, columna)`** devuelve `true` si y solo si, la celda en la posición `(fila,columna)` es una mina. Se asume en esta función, y en las 4 siguientes, que la posición `(fila,columna)` es válida.
- **`es_visible(juego, fila, columna)`** devuelve `true` si y solo si, la celda en la posición `(fila,columna)` es visible.
- **`esta_marcada(juego, fila, columna)`** devuelve `true` si y solo si, la celda en la posición `(fila,columna)` está marcada.
- **`esta_vacia(juego, fila, columna)`** devuelve `true` si y solo si, la celda en la posición `(fila,columna)` está vacía.
- **`contiene_numero(juego, fila, columna)`** devuelve `true` si y solo si, la celda en la posición `(fila,columna)` contiene un número.
- **`dame_numero(juego, fila, columna)`** devuelve el número asociado a la celda que ocupa la posición `(fila,columna)`. Asume que la posición es válida.
- **`esta_completo(juego)`** devuelve `true` si y solo si todas las celdas del tablero, que no son minas, son visibles.
- **`mina_explotada(juego)`** devuelve `true` si y solo si alguna mina ha sido descubierta.

- `esta_terminado(juego)` devuelve `true` si y solo si todas las celdas que no son minas están descubiertas o una mina ha sido descubierta.
- `poner_mina(juego, fila, columna)` si la posición `(fila, columna)` es válida y no contiene ya una mina, entonces coloca una mina en esa posición y actualiza sus posiciones vecinas incrementando, cuando sea posible, su número en `1`. Ten en cuenta que si una celda vecina está vacía, entonces su estado cambiará a `NUMERO` y su número será `1`.
- `marcar_desmarcar(juego, fila, columna)` si la posición `(fila, columna)`, es válida y está marcada, la desmarca. En caso de que la posición no esté marcada, entonces la marca.
- `ocultar(juego, fila, columna)` si la posición `(fila, columna)` es válida, y la celda en esa posición es visible, entonces la oculta.
- `juega(juego, fila, columna, lista_pos)` intenta descubrir la celda colocada la posición `(fila,columna)`, para lo que debe comprobar que la posición sea válida, la celda en esa posición no sea visible, y además no esté marcada. Si esto es así, descubre la celda y añade `(fila,columna)` a la lista `lpos`. Una vez descubierta, si no es una mina ni contiene un numero mayor que `0`, actualiza sus celdas adyacentes. La actualización consiste en descubrir todas las celdas vecinas que estén ocultas y no estén marcadas, e ir insertando las posiciones descubiertas en la lista `lista_pos`.

Módulo `listaPosiciones`

Implementa una lista de posiciones necesarias para realizar las operaciones de "*undo*".

```
typedef struct {
  int posx;
  int posy;
} tPosicion;

typedef struct {
  tPosicion lista[MAX_LISTA];
  int cont;
} tListaPosiciones;

inicializar(lista_pos);
insertar_final(lista_pos, x, y);
longitud(lista_pos);
dame_posX(lista_pos, i);
dame_posY(lista_pos, i);
```

El procedimiento `inicializar_listaPosiciones(lista_pos)` crea la lista vacía, inicializando su contador a `0`.
Los métodos `longitud(lpos)`, `dame_posX(lista_pos, i)`, `dame_posY(lista_pos, i)`, devuelven, respectivamente, el número de posiciones ocupadas en la lista, y el valor de `posX` (`posY`) del elemento almacenado en la posición `i`.
El método `insertar_final(lista_pos, x, y)` almacena, si es posible, la posición `(x,y)` al final de la lista.

Módulo `listaUndo`

Es una lista cuyos elementos son listas de posiciones. Concretamente se irán almacenando las posiciones que se descubran en cada paso del juego, para así poder ocultarlas al realizar la operación de "*undo*".

```
typedef struct {
  tListaPosiciones lista[MAX_UNDO];
  int cont;
} tListaUndo;

inicializar(lista_undo);
insertar_final(lista_undo, lista_pos);
ultimos_movimientos(lista_undo);
```

El procedimiento `inicializar(lista_undo)` inicializa el número de elementos a `0`.
La operación de `insertar_final(lista_undo, lista_pos)` inserta `lista_pos` al final de la lista. Si no hay espacio desplaza todos los elementos hacia la izquierda, descartando el colocado en la posición `0`. Finalmente inserta en la última posición.
La función `ultimos_movimientos(lista_undo)` devuelve el último elemento de la lista.

Módulo `inputOutput`

En este módulo se deben implementar todas las funciones relacionadas con la entrada/salida de la aplicación. Ofrece las siguientes funciones públicas:

```
mostrar_cabecera();
pedir_pos(fila, columna);
mostrar_resultado(juego);
mostrar_juego_consola(juego);
carga_juego(juego);
```

donde:

- `mostrar_cabecera()` muestra la cabecera del juego. Concretamente:

  ```
  Buscaminas
  ----------
  ```

- `pedir_pos(fila,columna)`: solicita al usuario que introduzca una `fila` y una `columna`.
- `mostrar_resultado(juego)`: muestra el resultado final del juego. Si el jugador ha ganado, si ha perdido (explotado una mina), o si ha decidido abandonar la partida.
- `mostrar_juego_consola(juego)`: muestra la información del juego por consola, incluido el tablero. El código de esta función está disponible junto con el enunciado.
- `cargar_juego(juego)`: solicita al usuario el nombre de un fichero e intenta cargar el juego. En caso de que la apertura no haya sido correcta devuelve `false`. Define el operador:

  ```
  istream& operator>> (istream& in, tJuego& juego);
  ```

  y úsalo para implementar la función.

Módulo `main`

La función `main` carga un juego. Si la carga es correcta solicita una posición al usuario y la ejecuta. Repite este proceso hasta que o el juego termina, o el usuario introduce `(-1,-1)` para abandonar la partida. Recuerda que las posiciones `(-2,-2)` y `(-3,-3)` son especiales. La primera se emplea para marcar una posición y la segunda para realizar un "*undo*". Para el resto de posiciones será el juego quien ejecute el paso de acuerdo a sus reglas. Para ello implementa, al menos, la función:

```
juega(juego,fila,columna,lista_undo)
```

que analiza los valores de fila, columna y realiza la acción asociada a dichos valores. Puedes añadir tantas funciones como sean necesarias.

Y recuerda

Comentar el código es como limpiar el cuarto de baño;
nadie quiere hacerlo, pero el resultado es siempre
una experiencia más agradable para uno mismo y sus invitados.

— Ryan Campbell