Buscaminas 1.0

Fundamentos de la Programación II

Grados de la Facultad de Informática (UCM)

Fecha máxima de entrega: 31/03/2025 a las 9:00

Normas de realización de la práctica

1. La fecha límite para entregar la práctica es el **31/03/2025 a las 9:00**.
2. Debe entregarse a través del Campus Virtual, en la actividad Entrega práctica (versión 1).
3. Sigue los pasos que aparecen en el enunciado y ten en cuenta todas las indicaciones.
4. Aunque recomendamos que vayas haciendo la práctica de manera incremental, tal y como aparece en el enunciado, debes leerlo completamente antes de empezar.
5. En el Campus Virtual encontrarás 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

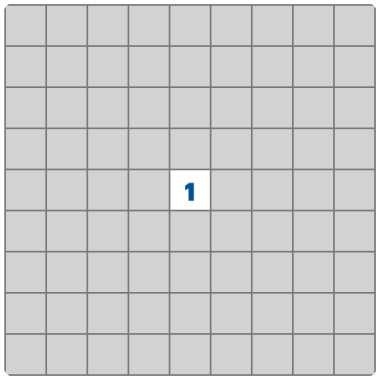
El juego

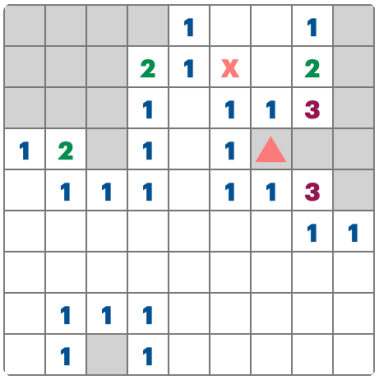
En esta práctica vamos a explorar la implementación de un clásico juego llamado *Buscaminas* (en inglés *Minesweeper*). Según la información de wikipedia (`https://es.wikipedia.org/wiki/Buscaminas`), Buscaminas es un género de videojuegos de lógica que generalmente se juega en computadoras personales. El videojuego presenta un tablero de celdas en las que se puede hacer clic, donde hay "minas" ocultas (representadas como minas navales en el videojuego original) esparcidas por todo el tablero. El objetivo es limpiar el tablero sin detonar ninguna mina, con la ayuda de pistas sobre el número de minas vecinas en las celdas circundantes.

Al Buscaminas se juega en un tablero, donde inicialmente todas las celdas están ocultas, y hay que irlas descubriendo sin descubrir ninguna celda con una mina. Las minas están distribuidas al azar. Aquellas celdas que no contienen una mina, contendrán un número que representa el número de celdas con mina que la rodean. Para ganar, debes descubrir todas las celdas que no contienen minas.

Cuando el jugador selecciona por primera vez una celda en el tablero, si la celda contiene una mina, entonces el juego termina. En otro caso se descubre esa celda. 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 las ocho celdas. 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 celdas 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, a la izquierda aparece un tablero donde la celda descubierta 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 descubrirse sin descubrir ninguna mina. No hay puntuación, pero se registra el número de pasos que se tarda en terminar la partida.

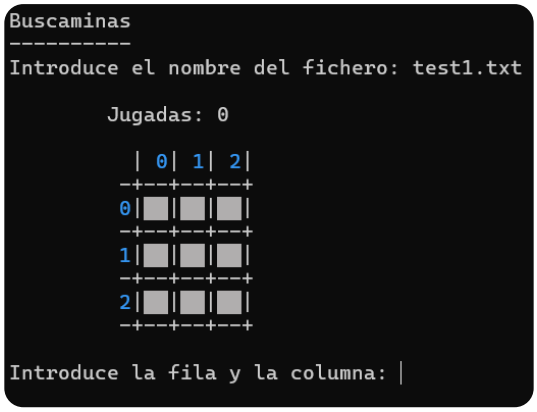
Nosotros vamos a comenzar con una versión simple del Buscaminas, en la que únicamente, una vez elegida una celda, si está vacía, se descubre la información de sus ocho casillas adyacentes. En versiones posteriores implementaremos el juego original.

Paso 1. Jugar 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, `fila` y `columna`, en las que se encuentran colocadas las minas. El tablero se cargará y se mostrará en la consola de la siguiente manera:



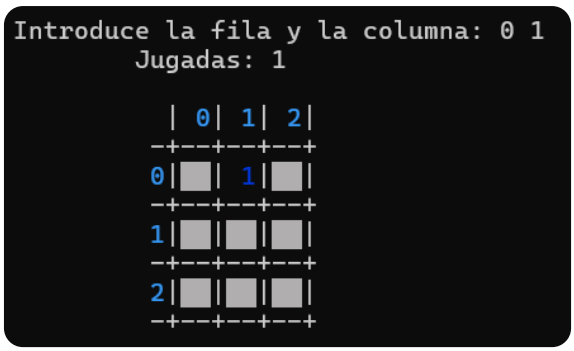
Las celdas blancas se corresponden con celdas ocultas, y son las celdas que se podrán elegir para ser descubiertas. El juego discurre solicitando al usuario la elección de una posición, denotándola mediante una `fila` y una `columna`. Dependiendo de los valores introducidos se realizará una acción u otra. Concretamente:

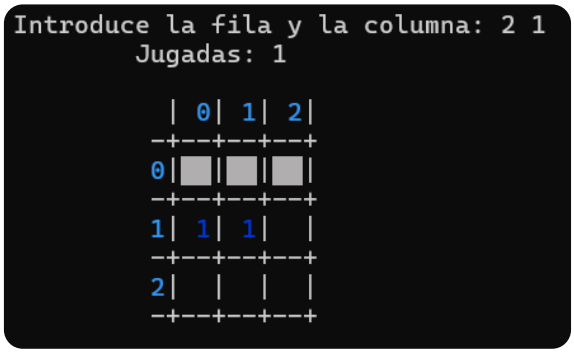
- Si `fila = -1, columna = -1`, entonces el juego termina.
- Si `fila = -2, columna = -2`, se "**marca**" la celda que indique el jugador. Más tarde explicaremos su funcionamiento.
- Si `fila = -3, columna = -3`, se realiza una operación de **undo**, volviendo el tablero a su configuración anterior. 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, se descubre, pero además también se descubren todas sus celdas vecinas.



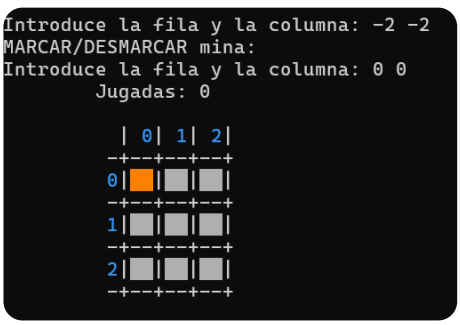


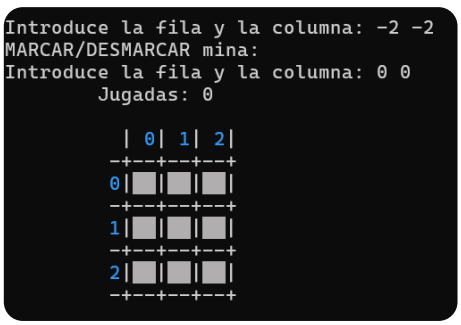
marcar/desmarcar celdas

Existe 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 se marca.
- Si la celda está marcada, entonces se desmarca.
- En otro caso no se hace nada.

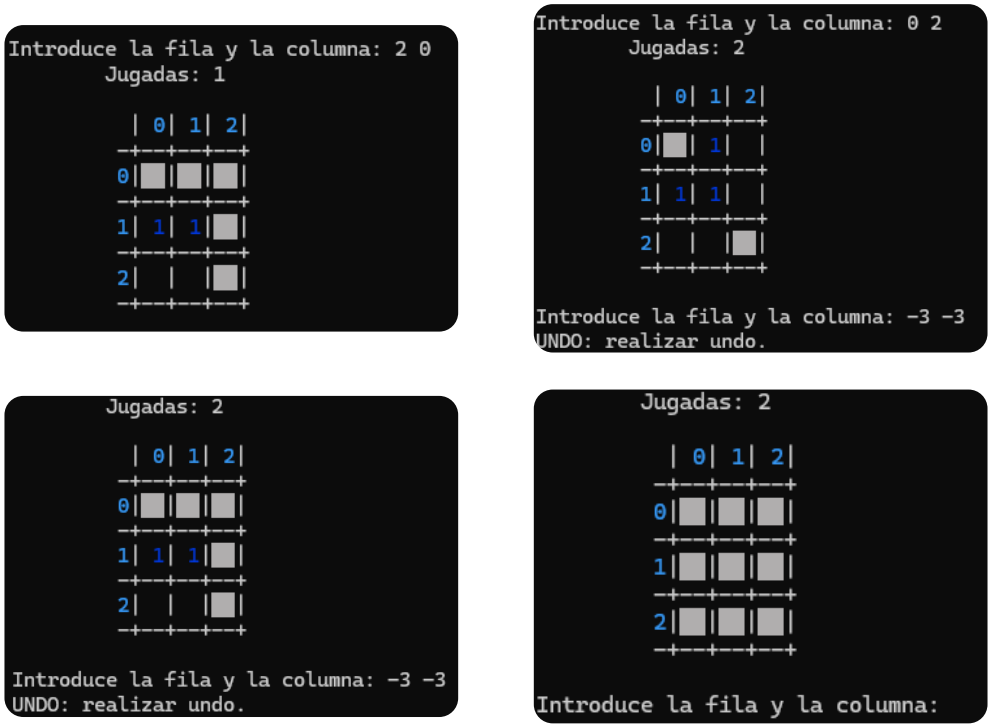
Partiendo de nuestro tablero original, la figura de la izquierda muestra una marca sobre la celda `(0,0)`, mientras que la figura de la derecha posteriormente la desmarca.





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!

Paso 2: Implementación

Con el fin de asegurar un progreso efectivo, vamos a utilizar un enfoque modular en el desarrollo de la aplicación. Esto facilitará la ampliación y el mantenimiento del sistema a lo largo del tiempo, simplificando la adición de nuevas características y funcionalidades, así como la gestión de modificaciones.

En las descripción de las clases sólamente especificamos el nombre de los métodos 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. También deberás indicar qué métodos son `const`.

Módulo `Celda`

Encapsula la información de una celda. Una celda puede estar vacía, ser una mina o contener un número. Utiliza el tipo **privado** `typedef enum { NUMERO, VACIA, MINA } Estado` para representar el estado de una celda.

La clase `Celda` tendrá al menos los siguientes campos y métodos.

```
class Celda {
private:
    typedef enum { NUMERO, VACIA, MINA } Estado;
    bool descubierta;
    Estado estado;
    int numero;
    bool marcada;
public:
    Celda();
    void destruye();
    dame_numero();
    esta_descubierta();
    contiene_mina();
    contiene_numero();
    esta_vacia();
    esta_marcada();
    descubrir_celda();
    ocultar_celda();
    poner_mina();
    poner_numero(n);
    marcar_celda();
    desmarcar_celda();
};
```

donde el campo `descubierta` determina si la celda está descubierta u oculta. El campo `estado` indica el tipo de celda que es. `numero` contiene (si es necesario) el número de minas que rodean a la celda. Finalmente `marcada` indica si la celda tiene una marca o no la tiene. Los métodos hacen lo siguiente:

- `Celda()`: constructora por defecto. Pone la celda oculta y no marcada, su estado a `VACIA` y su número a 0.
- `destruye()`: mismo código que la constructora por defecto.
- `dame_numero()`: devuelve el valor del campo `numero` de la celda.
- `esta_descubierta()`, `contiene_mina()`, `contiene_numero()`, `esta_vacia()`, `esta_marcada()`: devuelven `true` si y solo si la celda está, respectivamente, descubierta, contiene una mina (su `estado` es `MINA`), contiene un número (su `estado` es `NUMERO`), está vacía (su estado es `VACIA`) o está marcada.
- `descubrir_celda()`: descubre la celda.
- `ocultar_celda()`: oculta la celda.
- `poner_mina()`: cambia el estado de la celda a `MINA`.
- `poner_numero(n)`: pone el estado de la celda a `NUMERO` y su `numero` al valor `n`.
- `marcar_celda()`: marca la celda.
- `desmarcar_celda()`: desmarca la celda.

Módulo `Tablero`

Es el módulo para representar tableros como matrices bidimensionales. Declara el tipo de datos `Tablero` para la gestión de la matriz bidimensional, con al menos las siguientes operaciones:

```
class Tablero {
private:
   int filas, columnas;
   Celda celdas[MAX_FILS][MAX_COLS];
public:
   Tablero();
   Tablero(int fils, int cols);
   destruye();
   num_filas();
   num_columnas();
   es_valida(fila, columna);
   dame_celda(fila, columna);
   poner_celda(fila, columna, celda);
};
```

El comportamiento de las operaciones es el siguiente:

- `Tablero()`: constructora por defecto que inicia las filas y columnas del tablero a `0`.
- `Tablero(fils, cols)`: inicia las filas y columnas del tablero a los valores de los parámetros, y cada una de las celdas del tablero a través de su constructora por defecto.
- `destruye()`: destruye todas las celdas de la matriz, y pone el número de filas y columnas a `0`.
- `num_filas()`, `num_columnas()`: sirven para consultar, respectivamente, el número de filas y columnas del tablero.
- `es_valida(fila, columna)`: devuelve `true` si y solo si, la posición (`fila`, `columna`) del tablero es válida.
- `dame_celda(fila, columna)`: devuelve la celda contenida en la posición `fila`, `columna`, que se supone correcta.
- `poner_celda(fila, columna, celda)`: asigna el valor `celda` a la posición `fila`, `columna` que supone correcta.

Módulo `Juego`

Este es el módulo más importante de la aplicación. Define el tipo de datos `Juego` que representa la configuración actual de un juego, incluyendo el tablero, el número de minas, el número de marcas y el número de celdas descubiertas. También contiene la lógica del juego, es decir, es donde se agrupan y establecen las *reglas del juego*.

El tipo de datos `Juego` contiene al menos las siguientes operaciones públicas (puedes utilizar tantas operaciones privadas como necesites para implementar las operaciones públicas):

```
class Juego {
private:
    Tablero tablero;
    int num_jugadas;
    bool mina_pisada;
    int num_minas;
    int num_descubiertas;
public:
    Juego();
    Juego(int fils, int cols);
    destruye();
    dame_num_jugadas();
    dame_num_filas();
    dame_num_columnas();
    dame_num_minas();
    contiene_mina(fila, columna);
    esta_completo();
    mina_explotada();
    esta_descubierta(fila, columna);
    esta_marcada(fila, columna);
    esta_vacia(fila, columna);
    contiene_numero(fila, columna);
    dame_numero(fila, columna);
    poner_mina(fila, columna);
    marcar_desmarcar(fila, columna);
    ocultar(fila, columna);
    juega(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. La booleana `mina_pisada` determinada si el juego ha finalizado porque se ha explotado una mina.

En cuanto a la interfaz, el comportamiento de las operaciones es el siguiente:

- `Juego()`: es el constructor por defecto. Crea un juego vacío, con el numero de jugadas, minas y celdas descubiertas a `0`. Lógicamente `mina_pisada` debe ser `false`. El tablero se inicializa a través de su constructora por defecto.
- `Juego(fils, cols)`: igual que la constructora anterior, pero el tablero se inicializa con los parámetros.
- `destruye()`: destruye el tablero, y el resto de atributos los pone al mismo valor que la constructora por defecto.
- `dame_num_jugadas()`, `dame_num_filas()`, `dame_num_columnas()`, `dame_num_minas()`: devuelven, respectivamente, el número de jugadas, filas y columnas, y número de minas del juego.
- `contiene_mina(fila,columna)`, `esta_descubierta(fila,columna)`, `esta_marcada(fila,columna)`, `esta_vacia(fila,columna)`, `contiene_numero(fila,columna)`: devuelven `true` si y solo

si, la celda en la posición `fila`, `columna`, es, respectivamente, una mina, esta descubierta, está marcada, está vacía o contiene un número. Se asume en todos ellos que la posición `fila`, `columna` es válida.

- `esta_completo()`: devuelve `true` si y solo si todas las celdas del tablero, que no son minas, están descubiertas.
- `mina_explotada()`: devuelve `true` si y solo si alguna mina ha sido descubierta.
- `dame_numero(fila,columna)`: devuelve el número asociado a la celda que ocupa la posición `fila`, `columna`. Asume que la posición es válida.
- `poner_mina(fila,columna)`: si la posición `fila`, `columna` es válida, no está descubierta y no contiene ya una mina, entonces coloca una mina en esa posición y actualiza sus posiciones vecinas incrementando, cuando sea posible, los números de las celdas vecinas 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(fila,columna)`: si la posición `fila`, `columna` es una posición válida, entonces si la posición está oculta, marca la celda. Si la celda está marcada, entonces la desmarca. En otro caso no hace nada.
- `ocultar_celda(fila,columna)`: Si la posición `fila`, `columna` es válida, y la celda en esa posición está descubierta, entonces la oculta.
- `juega(fila,columna,lista_pos)`: intenta descubrir la celda colocada en la posición `fila`, `columna`. Para ello debe comprobar que la posición sea válida, la celda en esa posición esté oculta, y además no esté marcada. Si esto es así, descubre la celda y añade la posición `fila`, `columna` a la lista de posiciones `lista_pos`. Una vez descubierta la celda, si no es una mina ni contiene un número mayor que `0`, entonces actualiza sus celdas vecinas. La actualización consiste en descubrir todas las celdas vecinas que no estén ocultas o marcadas, e ir insertando las posiciones descubiertas en la lista `lista_pos`.

Módulo `ListaPosiciones`

Implementa una lista de posiciones que necesitaremos para realizar las operaciones de *undo*.

```
class ListaPosiciones {
private:
    typedef struct {
        int posx;
        int posy;
    } Posicion;
    int cont;
    Posicion lista[MAX_LISTA];
public:
    ListaPosiciones();
    destruye();
    void insertar_final(x, y);
    longitud();
    dame_posX(i);
    dame_posY(i);
};
```

La constructora crea la lista por defecto poniendo su contador a `0`. Los métodos `longitud()`, `dame_posX(i)`, `dame_posY(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(x,y)` almacena, si es posible, la posición `(x,y)` al final de la lista. La función `destruye()` se implementa igual que la constructora por defecto.

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*.

```
class ListaUndo {
private:
    int cont;
    ListaPosiciones lista[MAX_UNDO];
public:
    ListaUndo();
    destruye();
    insertar_final(lista_pos);
    ultimo_elemento();
};
```

La constructora por defecto inicializa el número de elementos a `0`. La función `destruye()` destruye todos los elementos de la lista y pone el contador a `0`. La operación de `insertar_final (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`. Y finalmente inserta `lista_pos` en la última posición. El método `ultimo_elemento()` devuelve el último elemento de la lista.

Módulo InputOutput

En este módulo se van a implementar todas las funciones relacionadas con la entrada/-salida de la aplicación. Ofrece las siguientes funciones públicas, que deben declararse en `InputOutput.h`:

```
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, Juego& juego);
  ```

  y usalo 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. Cuando el juego termina se muestra el resultado y se destruye el juego. Recuerda que las posiciones `(-2,-2)` y `(-3,-3)` son especiales. La primera marca una posición y la segunda realiza un *undo*. Para el resto de posiciones será el juego quien ejecute el paso de acuerdo a sus reglas. Para ello implementa la función `juega(juego,fila,columna,lista_undo)`, que analiza los valores de `fila`, `columna` y realiza la acción asociada a dichos valores.

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