Proyecto: Juego del buscaminas

A lo largo del semestre, se implementarán cinco versiones del juego Buscaminas (Minesweeper). En cada una de ellas se irán introduciendo nuevos desafíos diseñados para reforzar lo aprendido en la asignatura.

Se realizarán dos entregas, una el 28 de marzo y otra el 11 de mayo. Es obligatorio realizar ambas entregas, con una calidad aceptable, para poder aprobar la asignatura por evaluación continua.

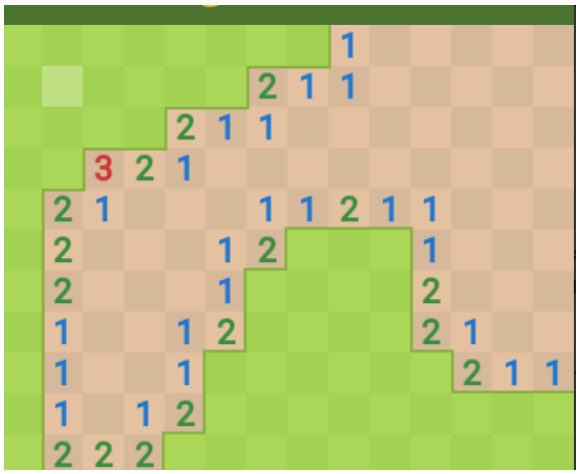
1 Descripción del juego

El buscaminas, en su versión básica, es un juego de un solo jugador. El jugador se enfrenta a un **tablero** dividido en **casillas** cuya apariencia inicial es idéntica. Algunas de estas casillas contienen **minas**, y el objetivo del jugador es **descubrir cuáles** son estas casillas y **marcarlas** con una bandera. Aquellas que no contengan una mina pueden estar **vacías o contener una pista** en forma de número. Dicho número indica la **cantidad total de minas** que existen en alguna de las **8 posiciones adyacentes** a la casilla, lo que se conoce como los **8-vecinos**. Son precisamente estas pistas numéricas las que el jugador debe utilizar para deducir dónde se encuentran localizadas todas las minas.

El jugador puede realizar 3 acciones diferentes sobre cada casilla:

- **Descubrir (step on)**: desvela el contenido de la casilla. Si se trata de una mina, el juego acaba y el jugador pierde. De lo contrario, se descubrirá la casilla correspondiente, vacía o pista.
- **Marcar (flag)**: el jugador cree que en la casilla existe una mina y lo señala colocando en ella un distintivo. En la mayoría de implementaciones gráficas, se coloca el icono de una bandera sobre la casilla. Una casilla con bandera no puede ser descubierta. El jugador posee un número limitado de banderas, que serán tantas como minas contiene el tablero.
- **Desmarcar (unflag)**: el jugador cambia de opinión sobre una casilla marcada con una bandera y la retira para que la casilla correspondiente pueda ser descubierta de nuevo.

Las casillas que **no tienen ninguna mina adyacente** se consideran **vacías**. Al descubrir una de éstas, se descubrirán automáticamente todas las **adyacentes** a la misma (será una de pista numérica o vacía). De las casillas descubiertas automáticamente, aquellas que sean vacías provocarán de nuevo el descubrimiento automático de sus casillas adyacentes. Esta reacción en cadena se propagará hasta que ninguna de las casillas descubiertas automáticamente sea vacía.



El jugador **pierde** en el momento que descubre alguna mina. Por el contrario, **gana** cuando consigue señalizar todas las casillas con mina. El juego mide el tiempo que el jugador tarda en alcanzar este estado.

Puedes probar una implementación del buscaminas y ver una demostración en vídeo de una partida en el siguiente enlace: http://buscaminas.eu/

2 Primera versión del buscaminas

En esta primera versión del proyecto implementaremos una versión del buscaminas en **modo texto**. El **tamaño del tablero** y la **cantidad de minas** será **fijo**: un cuadrado de 9x9 con un 12% de minas[1].

Al inicializar el juego las 9 minas se colocarán en posiciones aleatorias. Al descubrir una casilla vacía **no es necesario que se provoque una reacción en cadena de casillas vacías adyacentes**.

En cada iteración del juego, se mostrará en consola una matriz de caracteres que representan cada casilla. Inicialmente, todos los caracteres serán **"#", que representa una casilla sin descubrir**.

El juego solicitará una acción al usuario, que podrá ser **s** (step on), **f** (flag) o **u** (unflag). El usuario deberá introducir el carácter correspondiente, una **coordenada x** y una **coordenada y.**

[1] Al hacer los cálculos aplica redondeo al alza.

```
Ready to start
Time: 0 seconds
Flags left: 9
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]

movement (s | f | u):
s
1
1
```

En la siguiente iteración, el juego descubrirá la casilla indicada por el usuario y volverá a solicitar una acción.

```
Time: 53 seconds
Flags left: 9
[#, #, #, #, #, #, #, #, #]
[#, 1, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]

movement (s | f | u):
s
3
3
```

En cada iteración se volverá a mostrar el tablero, se informará del número de minas que quedan y del tiempo transcurrido en segundos desde el inicio del juego.

Si la casilla a descubrir contiene **una pista, se mostrará el número correspondiente**. Si está **vacía, se muestra un espacio en blanco**. En caso de **pisar una mina**, se descubrirá el tablero por completo y se informará de que el jugador ha perdido. Se utiliza el **carácter "@" para representar las minas**.

```
movement (s | f | u):
s
4
5
BOOOMMM!!!!
Time: 548 seconds
Flags left: 9
[ ,  ,  , 1, @, 1,  ,  ,  ]
[ ,  ,  , 1, 1, 1, 1, 1, 1]
[1, 2, 2, 1,  ,  , 1, @, 1]
[1, @, @, 1, 1, 2, 3, 2, 1]
[2, 3, 2, 1, 1, @, @, 1,  ]
[@, 1,  ,  , 2, 3, 4, 2, 1]
[1, 1,  ,  , 1, @, 2, @, 1]
[ ,  ,  ,  , 1, 1, 2, 1, 1]
[ ,  ,  ,  ,  ,  ,  ,  ,  ]

The game has finished
```

Para indicar posiciones del tablero, el origen de coordenadas se sitúa en la esquina superior izquierda de la matriz, y la coordenada mínima es cero. Es decir, en un tablero de 8x8, la esquina superior izquierda será la (0, 0) y la inferior derecha la (7, 7).

Cuando colocamos una bandera sobre una casilla con el comando (f)lag, el carácter a mostrar es el que aparece en la posición (0, 0) de la siguiente imagen:

```
movement (s | f | u):
f
0
0
Time: 48 seconds
Flags left: 8
[¶, #, #, #, #, #, #, #, #]
[#, 2, 1, #, #, #, #, #, #]
[#, 1,  , #, #, #, #, #, #]
[#, 1, #,  , #, #, #, #, #]
[#, #, #, #, 1, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
[#, #, #, #, #, #, #, #, #]
```

Para obtener dicho carácter se ha de ejecutar la siguiente sentencia:

```
Character.toString((char) 182);
```

3 Tarea

Junto con este enunciado se proporciona un proyecto Java que contiene una clase Main. Dicha clase es la que ha de ser ejecutada para lanzar el juego. Modifica el nombre de este proyecto a **primerapellido_segundoapellido_nombre_minesweeper_sprint1** e implementa las siguientes clases **tal y como se describen**. En este documento se **describen los métodos públicos** de cada clase, pero se podrán (y deberán) **utilizar tantos métodos privados y atributos como sea preciso**.

3.1 Clase Game

Gestiona la lógica principal de la partida. Es la clase que se encarga de ejecutar el bucle de iteraciones del juego. Únicamente desde esta clase se puede realizar interacción con el usuario.

Métodos públicos:

`Constructor Game(Board board)`
El constructor de Game recibe un objeto Board.

`void play()`
Inicializa el juego (mensajes al usuario, contador de tiempo, etc.). A continuación, ejecuta un bucle en el que se pide un movimiento al usuario y se descubre/(des)marca la casilla correspondiente. Cuando se detecta que el usuario ha perdido (ha pisado una mina) o el tablero está en situación de victoria (ya no quedan minas por encontrar y señalar), el juego se detiene, se muestra el tablero descubierto por completo y se informa al usuario de su victoria/derrota.

3.2 Clase Board

Representa el tablero del buscaminas y almacena el estado del juego en cada momento. Internamente, contiene una referencia a un array de objetos Square de dos dimensiones.

Métodos públicos:

`Constructor Board(int width, int height, int percentage)`
Construye un tablero de width x height posiciones, en la que el porcentaje de las casillas que contiene una mina lo marca percentage. El número de minas se redondeará al alza a partir del porcentaje.

`Constructor Board(int mines, Square[][] squares)`
Este constructor se utilizará únicamente para casos de test. Mines marca la cantidad de minas, y squares se asigna directamente a la referencia interna al array de casillas.

`boolean isExploded()`
Devuelve true si ha explotado alguna mina en el tablero, false en caso contrario.

`void stepOn(int x, int y)`
Si no está descubierta, descubre la casilla de coordenadas (x, y).

`void flag(int x, int y)`
Si no hay ya una bandera y la casilla no está descubierta, coloca una bandera en la casilla de coordenadas (x, y).

`void unflag(int x, int y)`
Si hay una bandera, retira una bandera en la casilla de coordenadas (x, y).

`void unveil()`
Descubre todas las casillas del tablero.

`int getFlagsLeft()`

Devuelve el número de banderas que aún no han sido colocadas. Distinto de getlMinesLeft().

`int getMinesLeft()`

Devuelve el número de minas que aún no ha sido cubierto con una bandera.

`void markAsExploded()`

Sitúa al estado interno del tablero como explotado (debido a que se ha descubierto alguna mina).

`char[][] getStatus()`

Devuelve un array de caracteres que representa el estado del tablero de juego. El carácter de la posición (x, y) representará el estado de la posición (x, y) del tablero.

`Square[][] getSquaresForTest()`

Devuelve una referencia al array de Square manejado internamente. Este método solo se utilizará para realizar tests.

3.3 Clase Square

Representa el contenido y estado de cada casilla en el tablero. Cada Square tiene tres posibles estados: CLOSED (no descubierto), OPEN (descubierto, se ve lo que contiene la casilla) o FLAGGED (marcado). Además, tiene un valor numérico asociado (lo que contiene la casilla), con 3 posibles casos.

- 0 → Casilla vacía.
- >=1 y <=8 → Casilla de pista numérica.
- -1 → Casilla con mina.

Estos son sus métodos públicos:

`void stepOn()`

Pasa la casilla a estado OPEN si la está CLOSED. En caso contrario (OPEN o FLAGGED) no hace nada.

`String toString()`

Devuelve el carácter que representa la casilla gráficamente de acuerdo a su valor y estado actual.

`void flag()`

SI la casilla está cerrada, su estado pasa a FLAGGED. En caso contrario no hace nada.

`void unflag()`

Si la casilla está marcada, su estado pasa a CLOSED. En caso contrario no hace nada.

`void open()`

Pasa la casilla a estado OPEN incondicionalmente.

`void getValue()`
Devuelve el valor numérico de la casilla.

`void setValue(int value)`
Establece el valor numérico de la casilla.

`boolean hasMine()`
Devuelve true si la casilla contiene una mina y false en caso contrario.

`void putMine()`
Coloca el valor numérico correspondiente a una mina en la casilla.

`boolean hasFlag()`
Devuelve true si el estado de la casilla es FLAGGED y false en caso contrario

`boolean isOpen()`
Devuelve true si el estado de la casilla es OPEN y false en caso contrario.

3.4 Test

Se deben realizar pruebas unitarias para los métodos de las clases mencionados a continuación. En el enunciado te proporcionamos los casos, deberás implementar el código correspondiente haciendo las comprobaciones de estado oportunas en cada caso.

3.4.1 Clase Game

`UnveilTests`
Comprobar que todas las casillas están abiertas tras ejecutar unveil con:
- Tablero completamente cubierto.
- Tablero con alguna bandera.
- Tablero con alguna casilla ya descubierta.

`UnflagTests`
Casos:
- Desmarcar casilla de mina con bandera.
- Desmarcar casilla de mina sin bandera.
- Desmarcar casilla sin bandera.
- Desmarcar una casilla dos veces.

FlagTests
Casos:
- Bandera sobre casilla ya marcada.
- Bandera sobre casilla desmarcada de mina.
- Bandera sobre casilla desmarcada sin mina.
- Bandera sobre una casilla dos veces consecutivas.

`StepOnTests`

Casos:

- stepOn en casilla ya descubierta.
- stepOn en casilla con bandera con mina.
- stepOn en casilla con bandera sin mina.
- stepOn en casilla con mina.
- stepOn en casilla con pista numérica.
- stepOn en casilla vacía.

`GetStatusTests`

Casos:

- Tablero completamente cubierto.
- Tablero completamente descubierto.
- Tablero en situación de juego intermedia: alguna casilla con bandera, alguna casilla descubierta vacía, alguna casilla descubierta con pista numérica.

`ConstructorTests`

Casos:

- Comprobar que la dimensión de la matriz y la cantidad de minas se corresponden con lo especificado por parámetro.

3.4.2 Clase Square

`StepOnTests`

Casos:

- stepOn sobre casilla OPEN.
- stepOn sobre casilla CLOSED.
- stepOn sobre casilla FLAGGED.

`UnflagTests`

Casos:

- unflag sobre casilla OPEN.
- unflag sobre casilla CLOSED.
- unflag sobre casilla FLAGGED.

FlagTests

Casos:

- flag sobre casilla OPEN.
- flag sobre casilla CLOSED.
- flag sobre casilla FLAGGED.