

PRÁCTICA Navidades. Juego Buscaminas

La práctica consiste en implementar en **JAVA** el juego del **Buscaminas**, para jugar en **modo consola**.

La práctica se realizará **por parejas o grupos de 3 personas**.

Antes de empezar a programar debes tener muy claro en qué consiste el juego y **diseñar en papel los algoritmos** necesarios para resolver los distintos problemas que se plantean.

Descripción general del juego

El Buscaminas se juega sobre un tablero formado por casillas ocultas. Algunas de estas casillas contienen minas.

El jugador podrá descubrir casillas o marcar banderas para indicar dónde cree que hay una mina.

El objetivo del juego es **descubrir todas las casillas que no contienen minas y marcar correctamente todas las minas con banderas**.

Representación del tablero

El tablero se representará mediante una **matriz bidimensional**.

Las casillas se mostrarán por consola de la siguiente forma:

- Casilla oculta: .
- Casilla con número: se mostrará el número correspondiente
- Casilla blanca (sin minas alrededor): espacio en blanco
- Casilla con mina: M
- Casilla con bandera: B

	0	1	2	3	4	5
0			1	.	.	0
1			2	.	.	1
2			2	.	.	2
3		1	2	.	2	3
4		1	.	.	.	4
5		1	B	.	.	5
	0	1	2	3	4	5

Clases del programa

La implementación del juego seguirá, al menos, las siguientes clases:

Clase Casilla

que contendrá los siguientes atributos **privados**:

- `boolean mina` → indica si la casilla contiene una mina
- `boolean blanco` → indica si la casilla es blanca (sin minas adyacentes)
- `boolean bandera` → indica si la casilla tiene una bandera
- `boolean visible` → indica si la casilla está descubierta
- `int numero` → número de minas adyacentes

Todos los atributos deberán inicializarse correctamente y estar encapsulados mediante **getters y setters**.

Clase Tablero

Cada posición del tablero estará representada por un objeto de la clase Casilla

- Matriz de casillas
- `int numMinas`
- `int numFilas`
- `int numColumnas`

El constructor se encargará de inicializar cada atributo según parámetros y creará el tablero con Casillas “vacías”,

La clase Tablero tendrá un método de objeto para inicializarTablero, el cual añadirá minas de forma aleatoria y calculará los números de cada casilla en función de las minas que haya en las casillas de alrededores.

La clase Tablero tendrá un método toString para devolver el tablero como cadena

Clase Juego

Atributo

- `Tablero tablero;`

El constructor se encargará de crear el tablero inicial del Juego para poder empezar a jugar.

Métodos de objeto:

descubrirCasilla(fila,col) devolverá true/false en función de si el juego ha terminado o no → **DIFÍCIL**

Comportamiento de las casillas:

- Si se descubre una casilla con mina:
 - Se muestran todas las minas
 - El juego termina (derrota)

- Si se descubre una casilla con número:
 - Se muestra el número de minas adyacentes
- Si se descubre una casilla blanca:
 - **Se descubre automáticamente el área de casillas blancas adyacentes**

ponerBandera(fila,col)

quitarBandera(fila,col)

Requisitos a tener en cuenta

- No se podrá descubrir una casilla marcada con bandera
- No se podrá colocar una bandera en una casilla ya visible
- El programa debe controlar que las posiciones introducidas estén dentro del tablero

Clase Buscaminas – clase Principal

gestiona la interacción con el usuario, el menú de opciones...

Esta será la clase que interactúe con el usuario, por lo que deberá hacer uso de la clase Teclado, proporcionada en el .jar.

Al iniciar el programa se pedirá al usuario la configuración inicial del juego:

```

      0 1 2 3 4 5
0 | . . . . . 0
1 | . . . . . 1
2 | . . . . . 2
3 | . . . . . 3
4 | . . . . . 4
5 | . . . . . 5
      0 1 2 3 4 5

```

Eliga el tipo de operacion

1->descubrir

2->Poner Bandera

3->Quitar Bandera

- Número de filas del tablero
- Número de columnas del tablero
- Número de minas

se creará el Tablero y se pintará, sacando un menú como el de la siguiente imagen:

Para cada acción, el jugador deberá introducir la fila y la columna correspondientes y se ejecutará la opción.

Eliga el tipo de operacion

1->descubrir

2->Poner Bandera

3->Quitar Bandera

1

Coordenada Fila

3

Coordenada columna

4

	0	1	2	3	4	5	
0	0
1	1
2	2
3	1 .	. .	3
4	4
5	5
	0	1	2	3	4	5	

La partida finaliza cuando:

- El jugador descubre una mina → **Has perdido**
- Todas las minas están marcadas con banderas y el resto de casillas están visibles → **Has ganado.**

División del proyecto Buscaminas (grupos de 3)

Alumno 1 — Casilla y Tablero

Tareas

- **Diseñar e implementar la clase Casilla:**
 - Atributos.
 - Getters y setters
 - ToString
- **Diseñar e implementar la clase Tablero:**
 - Atributos
 - Constructor
 - inicializarTablero: colocar las minas de forma aleatoria sin repetir posiciones y calcular los números de minas adyacentes: Actualizar `numero` y marcar casillas si blanco (`numero = 0`)

Para esto se podrán implementar métodos auxiliares, tener en cuenta los límites del tablero.
 - ToString (para pintar el tablero)

Alumno 2 — Juego

Tareas

- **Diseñar e implementar la clase Juego:**
 - Atributos.
 - Constructor
 - ToString
- Implementar las acciones del jugador:
 - Descubrir casilla
 - Poner bandera
 - Quitar bandera
- Controlar reglas:
 - No descubrir casillas con bandera

- No poner bandera en casillas visibles
- No repetir jugadas sobre casillas visibles
- Implementar la lógica de descubrimiento:
 - Casilla mina → fin del juego
 - Casilla numérica → visible
 - Casilla blanca → descubrimiento en cascada
- Gestionar el estado del juego:
 - Partida en curso
 - Victoria
 - Derrota

Alumno 3 — Buscaminas

Tareas

- main de la aplicación
- Solicitar entradas al usuario:
 - Filas, columnas, número de minas
- Validar entradas:
 - Valores dentro del tablero
 - Opción de acción válida
- Mostrar mensajes del juego:
 - Victoria
 - Derrota
 - Errores de entrada
- Coordinar el bucle principal del juego

Trabajo conjunto

Todos los miembros deben colaborar en:

- Definir interfaces/métodos comunes
- Integrar las partes del proyecto
- Probar el juego completo
- Corregir errores

- Documentar el código

Entrega y trabajo colaborativo (GitFlow simplificado)

El proyecto deberá gestionarse mediante **Git y GitHub**, aplicando un flujo de trabajo colaborativo basado en **GitFlow simplificado**.

Repositorio

- El proyecto se subirá a **GitHub** en un único repositorio por grupo.
- El repositorio deberá contener todo el código fuente del proyecto.

Ramas obligatorias

El repositorio deberá tener, como mínimo, las siguientes ramas:

- `main` → rama estable con la versión final del proyecto
- `develop` → rama de integración del equipo
- Una rama por cada alumno del grupo (por ejemplo):
 - `feature-alumno1`
 - `feature-alumno2`
 - `feature-alumno3`

Forma de trabajo

- Cada alumno deberá trabajar **exclusivamente en su propia rama**.
- No se permite trabajar directamente sobre `develop` ni `main`.
- Cada alumno irá realizando commits frecuentes y descriptivos en su rama.

Integración del código

1. Cuando un alumno termine su parte:
 - Deberá fusionar (**merge**) su rama con `develop`.
 - El merge deberá realizarse sin errores y resolviendo conflictos si los hubiera.
2. Una vez que todo el equipo haya integrado su trabajo en `develop`:
 - Se probará el proyecto completo.
 - Si el proyecto funciona correctamente, se realizará el merge final de `develop` en `main`.

Reglas importantes

- La rama `main` solo debe contener versiones **funcionales y finales** del proyecto.
- Los conflictos de merge deberán resolverse de forma consensuada por el grupo.
- El historial de commits debe ser claro y entendible.

Entrega final

- Se entregará el **enlace al repositorio de GitHub**.
 - El proyecto debe compilar y ejecutarse correctamente desde la rama `main`.
 - Se valorará el uso correcto del flujo de trabajo con ramas.
-

Presentación del proyecto

Además de la entrega del código, el proyecto deberá ser **presentado oralmente en clase**.

Requisitos de la presentación

- La presentación se realizará **por grupos**.
- Todos los miembros del grupo deberán participar.
- Se explicará:
 - El funcionamiento general del juego
 - La estructura del proyecto y las clases principales
 - El reparto de tareas dentro del grupo
 - El uso de Git y las ramas (GitFlow simplificado)

Demostración

- Durante la presentación se realizará una **demostración en directo** del juego por consola.
- El programa deberá ejecutarse correctamente.

Evaluación de la presentación

Se valorará:

- Claridad en la explicación
- Participación equilibrada de todos los miembros
- Correcta demostración del funcionamiento del juego
- Uso adecuado del vocabulario técnico