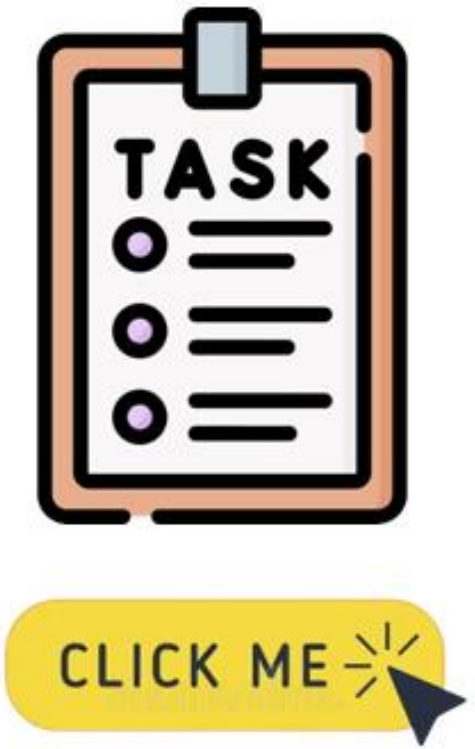
T20 – MAVEN



```java
public class TA20_Ej01_ventana{

    public static void main(String[] args) {
        // Crear y configurar el marco de la ventana principal
        JFrame frame = new JFrame("MegaVentana de la Muerte");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cerrar la
aplicación al cerrar la ventana
        frame.setSize(400, 300); // Establecer el tamaño inicial de la ventana

        // Crear una etiqueta con texto centrado
        JLabel label = new JLabel("¡WASSAAAAAAA!", SwingConstants.CENTER);
        frame.getContentPane().add(label, BorderLayout.CENTER); // Añadir la
etiqueta al centro del marco

        // Hacer visible la ventana
        frame.setVisible(true);
    }
}
```

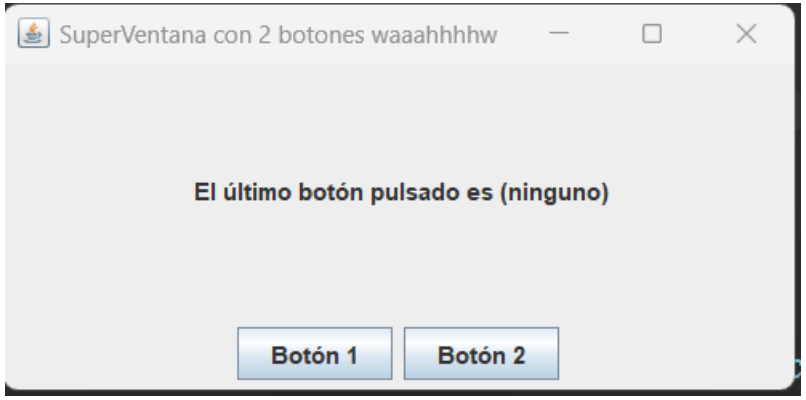

```java
private static ImageIcon loadImageFromDatabase() {
    ImageIcon imageIcon = null;

    // Información de la base de datos
    String url = "jdbc:mysql://localhost:3306/DB_IMAGES";
    String username = "root";
    String password = "";

    // Consulta SQL para obtener la imagen
    String query = "SELECT image FROM wassa WHERE name = 'wassa'";
```

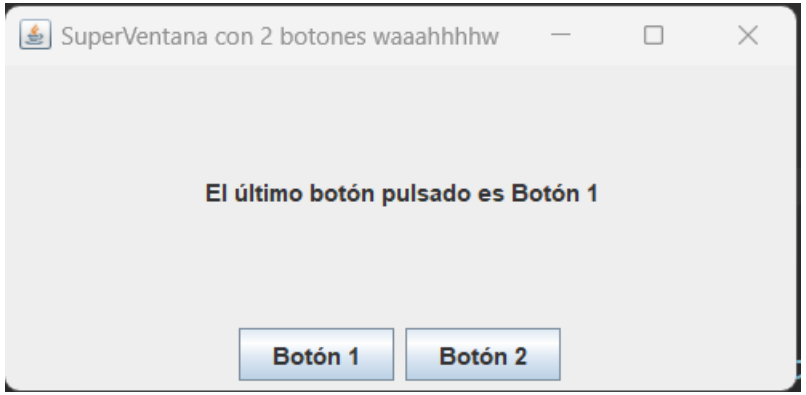
Hemos creado una ventana con título propio, una etiqueta y una imagen.

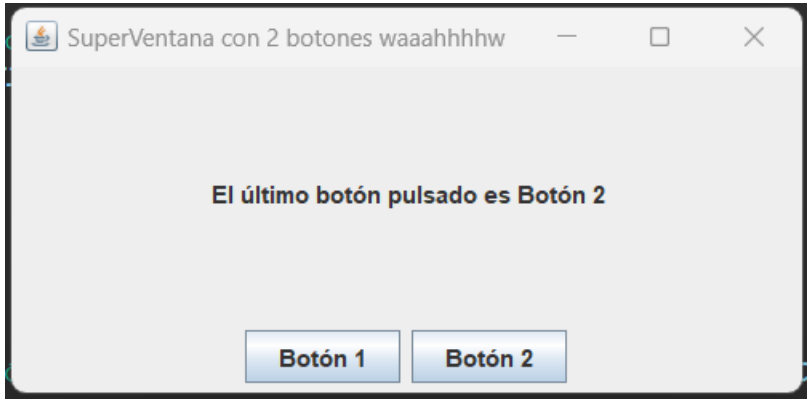
La imagen está descargada desde la base de datos que hemos creado (DB_IMAGES), y buscamos la imagen a través de la consulta por su nombre.

```java
public class TA20_Ej02_doubleButton
{
    public static void main(String[] args) {
        // Crear el frame principal
        JFrame frame = new JFrame("SuperVentana con 2 botones waaahhhhw");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(410, 200);
```

```java
// Añadir los botones al panel de botones
buttonPanel.add(button1);
buttonPanel.add(button2);

// Añadir el panel de botones al panel principal
panel.add(buttonPanel, BorderLayout.SOUTH);

// Añadir el panel principal al frame
frame.getContentPane().add(panel);

// Hacer visible el frame
frame.setVisible(true);
```







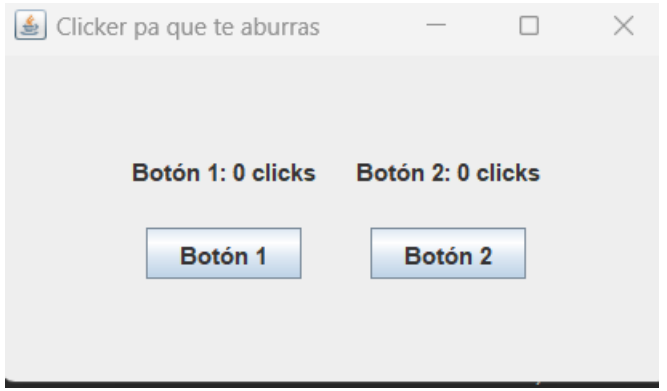
En este panel hemos añadido una etiqueta y dos botones.

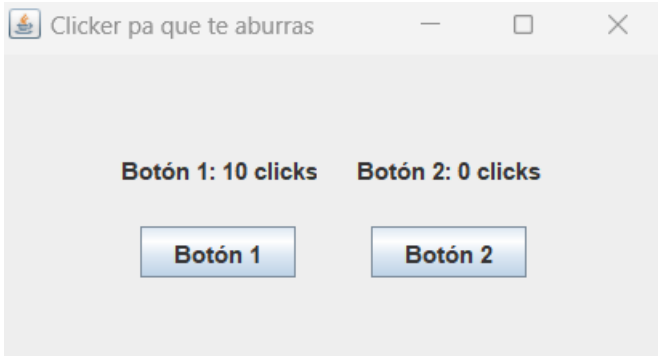
Los eventos de los botones hacen cambiar la etiqueta, expresando qué botón fue pulsado el último.

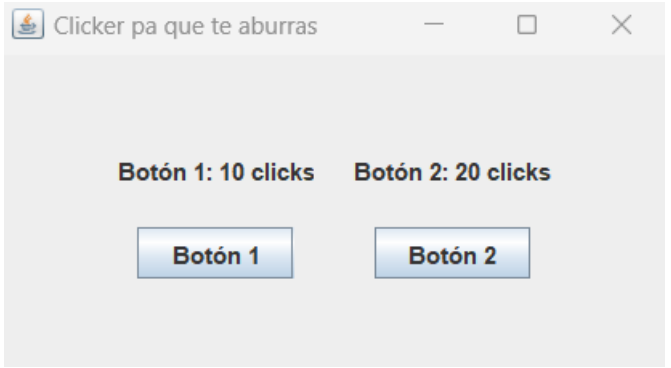
Si aún no han sido pulsados, se muestra la etiqueta por defecto.

```
public class TA20_Ej03_Clicker {
```

```
// Crear panel con GridBagLayout
JPanel panel = new JPanel(new GridBagLayout());
// Definir la posición y el comportamiento de cada componente dentro del GridBagLayout:
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets(10, 10, 10, 10); // Espaciado entre componentes

// Configurar y agregar componentes al panel
gbc.gridx = 0;
gbc.gridy = 0;
gbc.fill = GridBagConstraints.HORIZONTAL;
panel.add(label1, gbc);

gbc.gridx = 1;
gbc.gridy = 0;
panel.add(label2, gbc);

gbc.gridx = 0;
gbc.gridy = 1;
gbc.fill = GridBagConstraints.NONE;
panel.add(button1, gbc);

gbc.gridx = 1;
gbc.gridy = 1;
panel.add(button2, gbc);
```

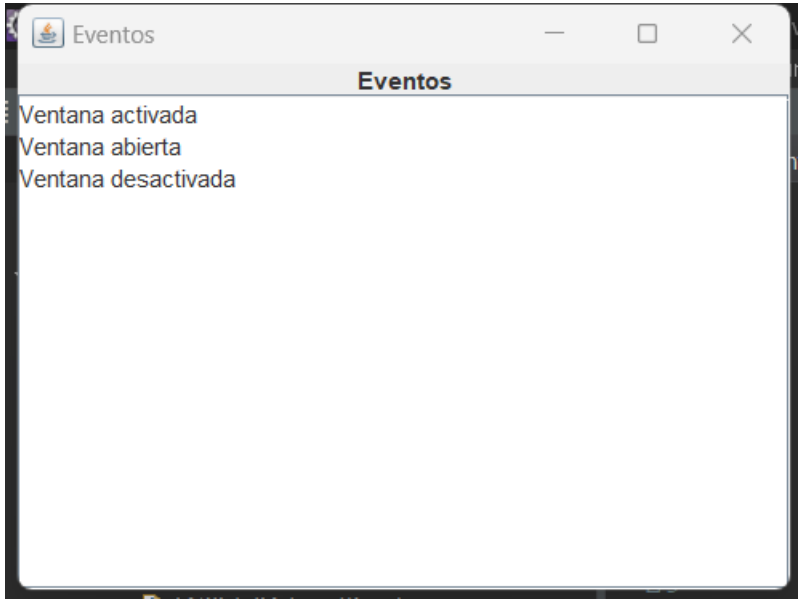






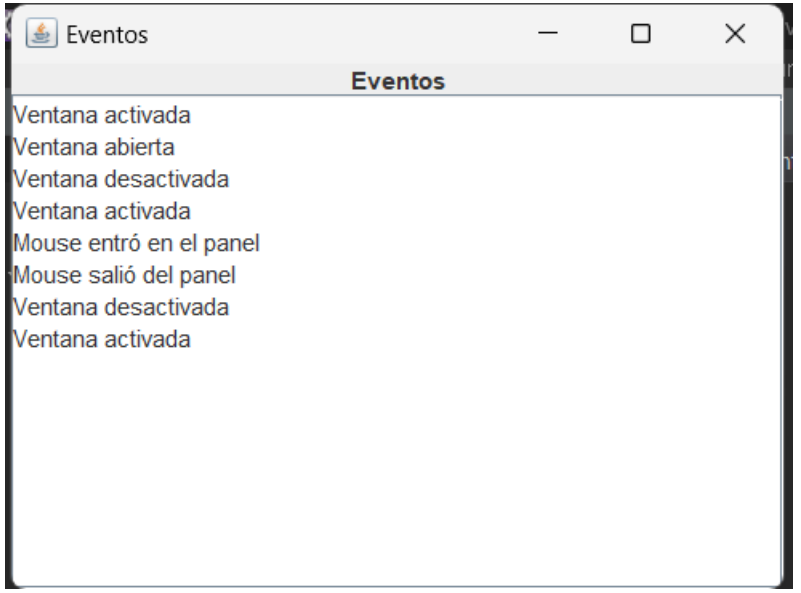
Hemos utilizado un GridBagConstraints sin definir, al cual le hemos ido añadiendo las label y los botones que queríamos; también hay un espaciado entre los componentes para una mayor diferenciación visual de los componentes.
Los label se van sobreescribiendo según el Evento de los botones.
Cada click al botón suma 1 al contador de clicks de ese botón; el resultado se muestra en el label correspondiente.

```java
public class TA20_Ej04_EventPanel extends JFrame {

    private JTextArea eventTextArea;
    private JLabel eventLabel;

    public TA20_Ej04_EventPanel() {
        // Configuración del marco
        setTitle("Eventos");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```




```java
        // Agregar listeners
        addWindowListener(new WindowAdapter() {

        panel.addMouseListener(new MouseAdapter() {
```

```java
@Override
public void windowOpened(WindowEvent e) {
    eventTextArea.append("Ventana abierta\n");
}

@Override
public void windowClosing(WindowEvent e) {
    eventTextArea.append("Ventana cerrándose\n");
}
```



En este panel hemos añadido una TextArea que va recogiendo los “eventos” que suceden al panel, como puede ser minimizar a ventana, pasar el mouse por encima, etc. Todos esos eventos son @Override’s, situados en los Listener adecuados.

MEMORY GAME ESPACIAL

ALEX, AURORA Y JOSE.

Tareas

MEMORY_GAME

- [x] Generar imagenes con CANVA
- [x] Crear la base de datos

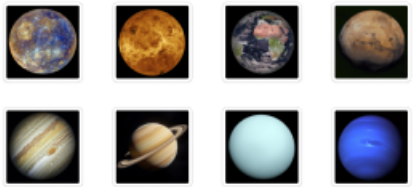
Código

- [x] 1. Generar interfaz con Swing
- [x] 2. Introducir las imagenes a la base de datos y cargarlas en el programa
- [x] 3. Generar la lógica del juego

Mejoras

- [x] Añadir contador
- [x] Generar barra de menú

IMÁGENES



DORSO

Fuente

NATIONAL GEOGRAPHIC

RECURSOS

http://localhost/phpmyadmin/index.php?route=/database/structure&db=db_images

CÓDIGO

Dimensiones del tablero y números de cartas necesarias:

```
public class MemoryGame {
    private static final int ROWS = 4;
    private static final int COLS = 4;
    private static final int NUM_CARDS = ROWS * COLS;
```

Conexión a DB:

```
    private static final String DB_URL = "jdbc:mysql://localhost:3306/db_images";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "";
```

Consulta para cargar imágenes desde DB:

```
    private static final String SQL_QUERY = "SELECT name, image FROM Cartas2 WHERE id BETWEEN 1 AND 8";
```

Declaración variables para el dorso, el array de imágenes y los botones que harán de cartas.

```
    private static ImageIcon backImage;
    private static ArrayList<ImageIcon> cardImages;
    private static JButton[] cardButtons;
```

Declaración variables para simular la selección de las cartas a voltear:

```
    private static int firstCardIndex = -1;
    private static int secondCardIndex = -1;
```

Declaración variables para contabilizar el número de intentos y el número de parejas encontradas (msg: Congrats!). También se añade la etiqueta para mostrar por pantalla el número total de intentos en vivo.

```
    private static int moveCount = 0;
    private static int pairsFound = 0; // Contador de pares encontrados
    private static JLabel moveCounterLabel;
```

Bucle del volteo de cartas:

1. Asegura voltear carta#1 y carta#2, mostrando sus imágenes (según index).

```
@Override
public void actionPerformed(ActionEvent e) {
    if (firstCardIndex == -1) {
        firstCardIndex = index;
        cardButtons[index].setIcon(scaleImageIcon
                (cardImages.get(index), cardButtons[index].getSize()));

    } else if (secondCardIndex == -1 && index != firstCardIndex) {
        secondCardIndex = index;
        cardButtons[index].setIcon(scaleImageIcon
                (cardImages.get(index), cardButtons[index].getSize()));
```

2. Actualiza el contador de intentos.

```
moveCount++;
moveCounterLabel.setText("Movimientos: " + moveCount);
```

3. Contrasta las cartas seleccionadas y verifica si son pareja o no.

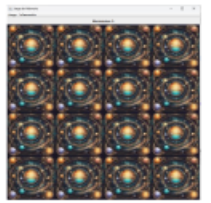
```
// Verificar si las cartas coinciden
if (cardImages.get(firstCardIndex).getDescription().
        equals(cardImages.get(secondCardIndex).getDescription())) {

    // Cartas coinciden, se dejan descubiertas
    pairsFound++;
    firstCardIndex = -1;
    secondCardIndex = -1;
    checkGameOver();
} else {
    // Cartas no coinciden, se voltean después de un breve retraso
    Timer timer = new Timer(1000, new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            cardButtons[firstCardIndex].setIcon(scaleImageIcon(backImage,
                    cardButtons[firstCardIndex].getSize()));
            cardButtons[secondCardIndex].setIcon(scaleImageIcon(backImage,
                    cardButtons[secondCardIndex].getSize()));
            firstCardIndex = -1;
            secondCardIndex = -1;
        }
    });
    timer.setRepeats(false);
    timer.start();
```

4. Cuando se han emparejado todas las cartas, se notifica por pantalla

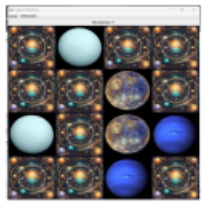
```
private void checkGameOver() {
    if (pairsFound == NUM_CARDS / 2) {
        JOptionPane.showMessageDialog(null,
                "¡FELICIDADES, HAS GANADO!");
    }
}
```

RESULTADO

Inicio del juego

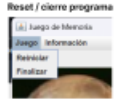


Partida in-game



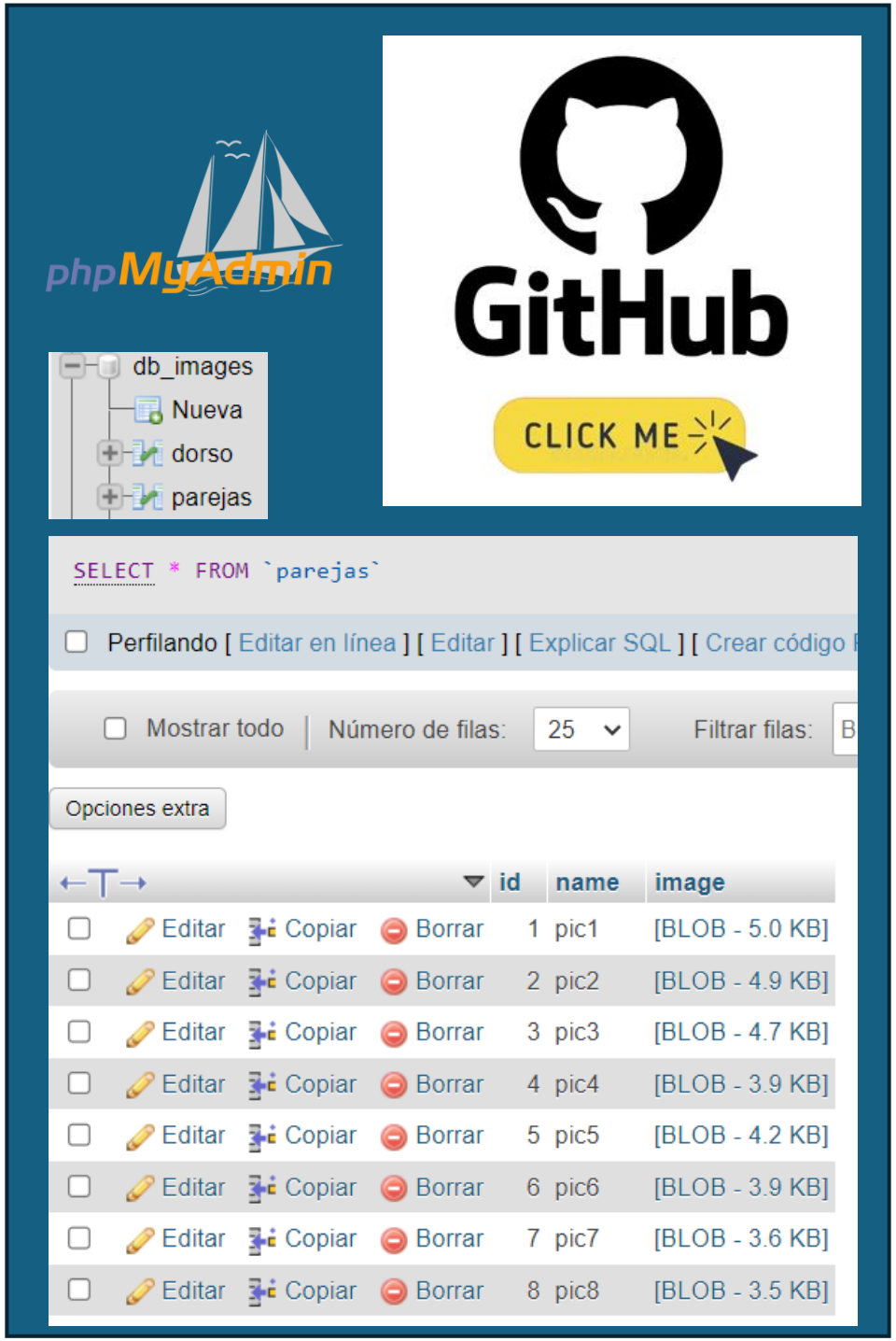
Partida finalizada

Reset / cierre programa



Créditos

CLICK ME



Hemos realizado una breve presentación del código.

Jose se ha encargado de la introducción y función del programa.
Aurora ha explicado cómo hemos trabajado y superado los retos que nos hemos encontrado durante el proceso.
Yo he explicado el código, cuyas líneas claves están visibles y explicadas en la presentación del Figma.

Hemos comentado entre todos que:

- Al inicio, utilizamos ToggleButton, pero éstos no nos dejaban incluir imágenes, por lo que cambiamos a Jbutton.
- Empezamos a trabajar con imágenes, las cuales se duplicaban para formar parejas.
- Pasamos de cargar las imágenes desde local a una DB.
- La DB (DB_images) no era compartida; la íbamos actualizando mediante *export* e *import* de queries.
- Tuvimos algunos problemas con el tamaño de las imágenes, pero pudimos solventarlo.

Como mejora, podríamos haber añadido un modo 2 jugadores, en el cual, al acertar parejas consecutivas, se contase el intento al fallar o ganar, por lo que podríamos añadir un contador de parejas (J1 vs J2).
También, aprovechando la DB, podríamos generar un histórico a modo de Récord de intentos, donde se guardase el récord con menor intentos y se sobrescribiera si se superase.