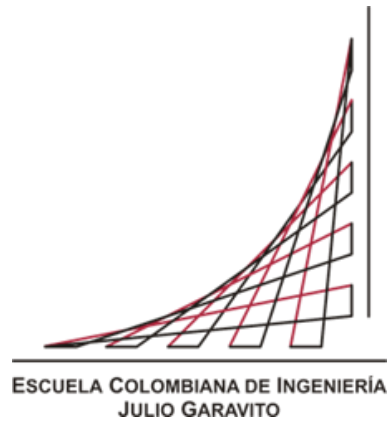


Escuela Colombiana de Ingeniería Julio Garavito Armero



LAB 5 POOB

GRUPO 2

Laura Valentina Rodríguez Ortégón

Joan Steven Acevedo Aguilar

2023-2

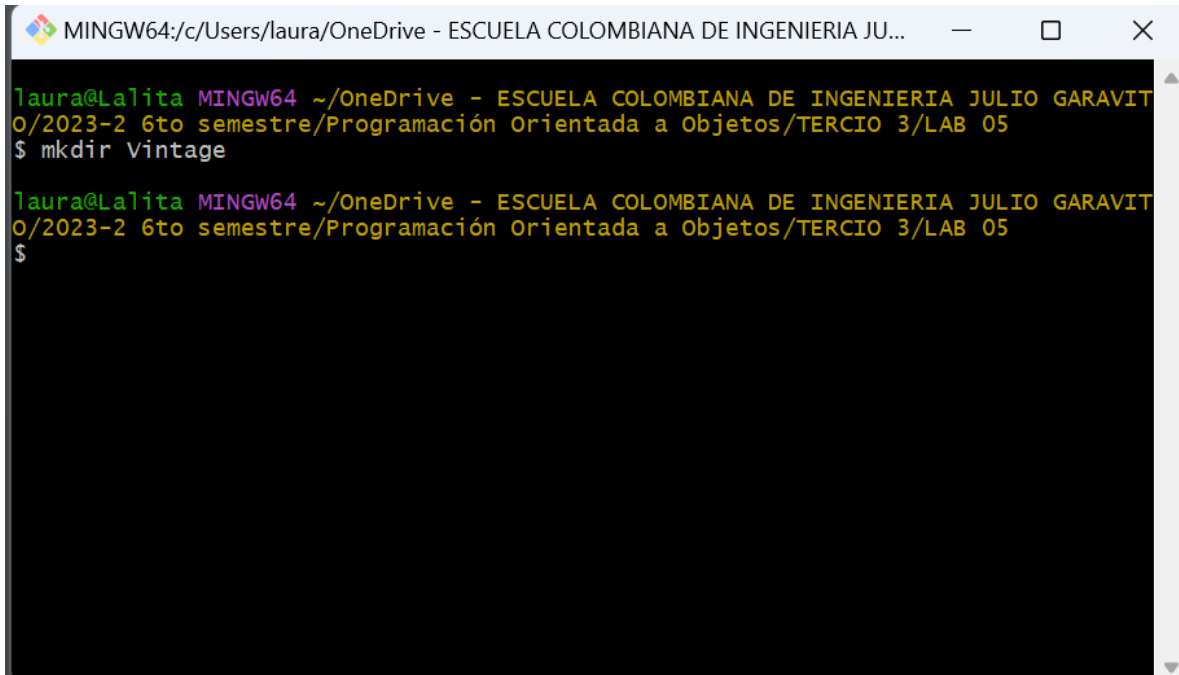
Bogotá (Cundinamarca)

DESARROLLO

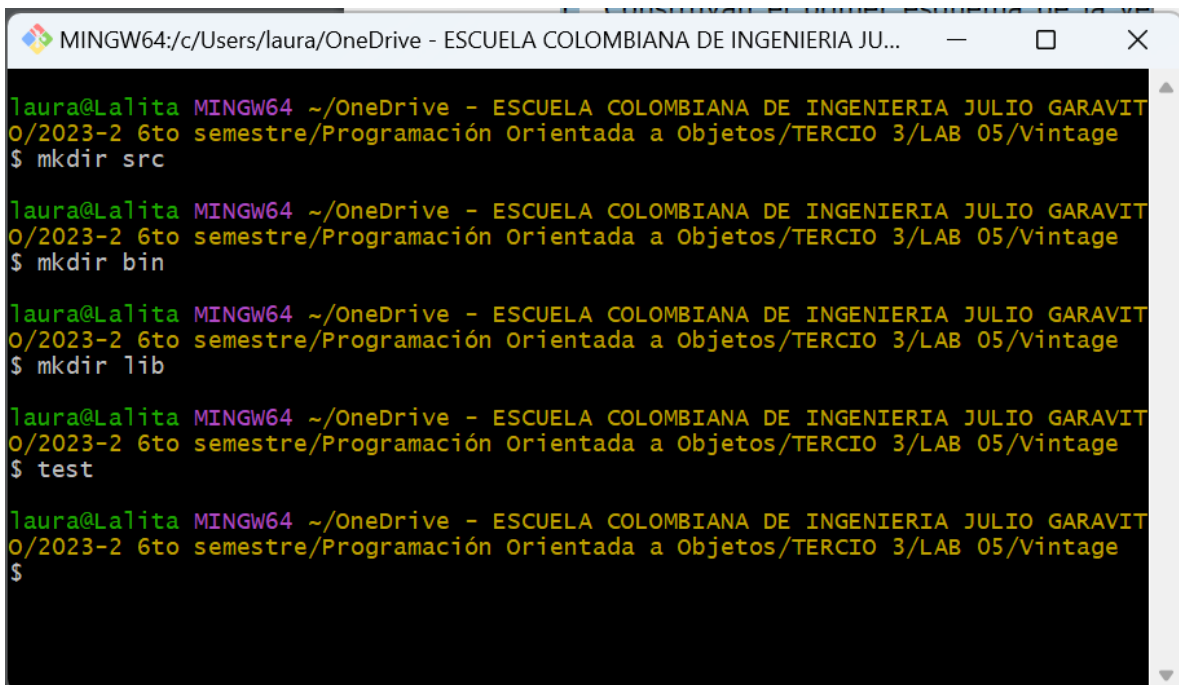
Directorios

El objetivo de este punto es construir un primer esquema para el juego Vintage.

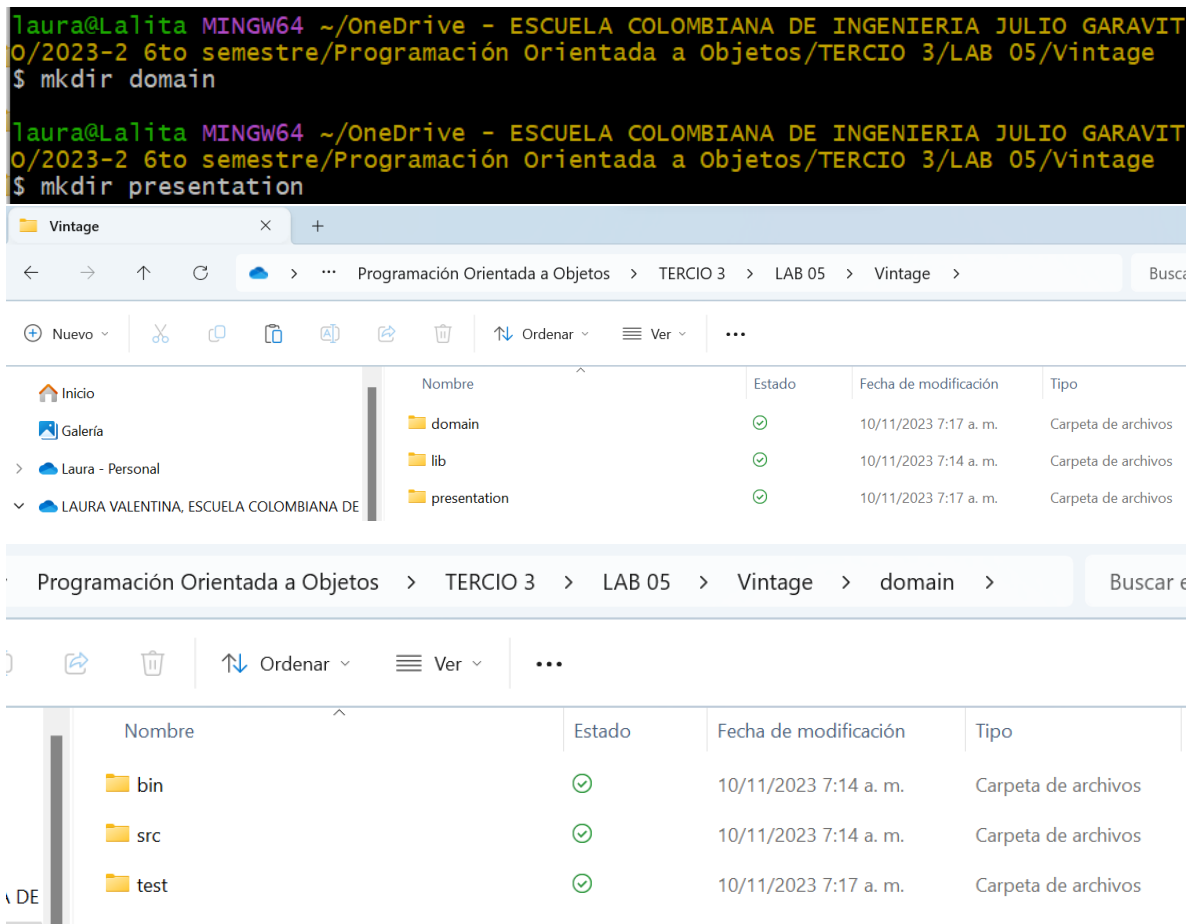
1. Preparen un directorio llamado Vintage con los directorios src y bin y los subdirectorios para presentación, dominio y pruebas de unidad.



```
MINGW64:/c/Users/laura/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JU...  
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVIT  
O/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05  
$ mkdir Vintage  
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVIT  
O/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05  
$
```



```
MINGW64:/c/Users/laura/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JU...  
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVIT  
O/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05/Vintage  
$ mkdir src  
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVIT  
O/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05/Vintage  
$ mkdir bin  
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVIT  
O/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05/Vintage  
$ mkdir lib  
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVIT  
O/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05/Vintage  
$ test  
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVIT  
O/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05/Vintage  
$
```



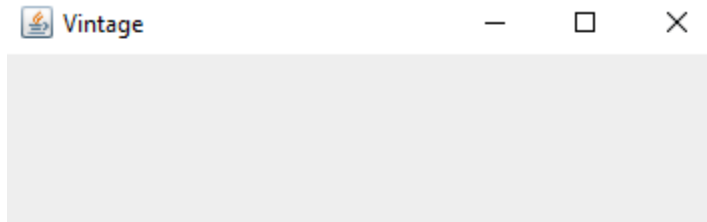
Ciclo 0: Ventana vacía – Salir

[En *.java y lab05.doc]

El objetivo es implementar la ventana principal de Vintage con un final adecuado desde el icono de cerrar. Utilizar el esquema de prepareElements-prepareActions.

1. Construyan el primer esquema de la ventana de Vintage únicamente con el título “Vintage”. Para esto cree la clase VintageGUI como un JFrame con su creador (que sólo coloca el título) y el método main que crea un objeto VintageGUI y lo hace visible. Ejecútenlo. Capturen la pantalla.

(Si la ventana principal no es la inicial en su diseño, después deberán mover el main al componente visual correspondiente)



```
import javax.swing.*;

public class VintageGUI extends JFrame{

    private String title = "";

    private VintageGUI(){
        prepareElements();
        prepareAcciones();
    }

    Run | Debug
    public static void main(String args[]){
        VintageGUI gui=new VintageGUI();
        gui.setVisible(b:true);
    }

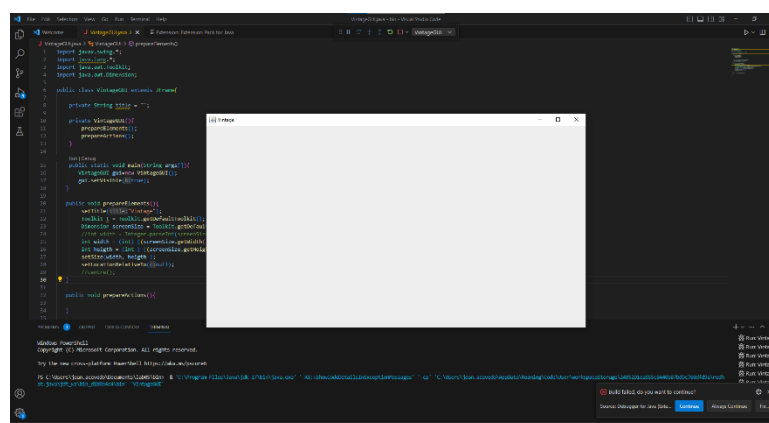
    public void prepareElements(){
        setTitle(title:"Vintage");
    }

    public void prepareAcciones(){

    }

}
```

2. Modifiquen el tamaño de la ventana para que ocupe un cuarto de la pantalla y ubíquela en el centro. Para eso inicien la codificación del método `prepareElements`. Capturen esa pantalla.



```

public void prepareElements(){
    setTitle(title:"Vintage");
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int width = (int) ((screenSize.getWidth()/2));
    int height = (int) ((screenSize.getHeight()/2));
    setSize(width, height);
    setLocationRelativeTo(c:null);
}

```

3. Traten de cerrar la ventana. ¿Termina la ejecución?

Cuando intentamos cerrar la ejecución, no lo está haciendo por sí mismo, si no que nosotros debemos cerrarlo, cuando se ejecuta un nuevo JFrame, queda en la pantalla de ejecución, es decir, se queda ejecutando en segundo plano sin finalizar el proceso.

> IntelCpHeciSvc Executable	0%	0.1 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	26.4 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	25.2 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	24.9 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	21.7 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	26.1 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	26.5 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	27.3 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	32.1 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	24.2 MB	0 MB/s	0 Mbps
Java(TM) Platform SE binary	0%	4.5 MB	0 MB/s	0 Mbps
mic tray icon	0%	0.2 MB	0 MB/s	0 Mbps

¿Qué deben hacer para terminar la ejecución?

Para poder hacerlo debemos forzar la ejecución nosotros mismos en el panel de control para poder cerrarlo.

4. Estudien en JFrame el método `setDefaultCloseOperation`. ¿Para qué sirve?

Sirve para llevar la aplicación a la bandeja de sistema, una vez que pulsamos cerrar en la ventana de la aplicación.

¿Cómo lo usarían si queremos confirmar el cierre de la aplicación?

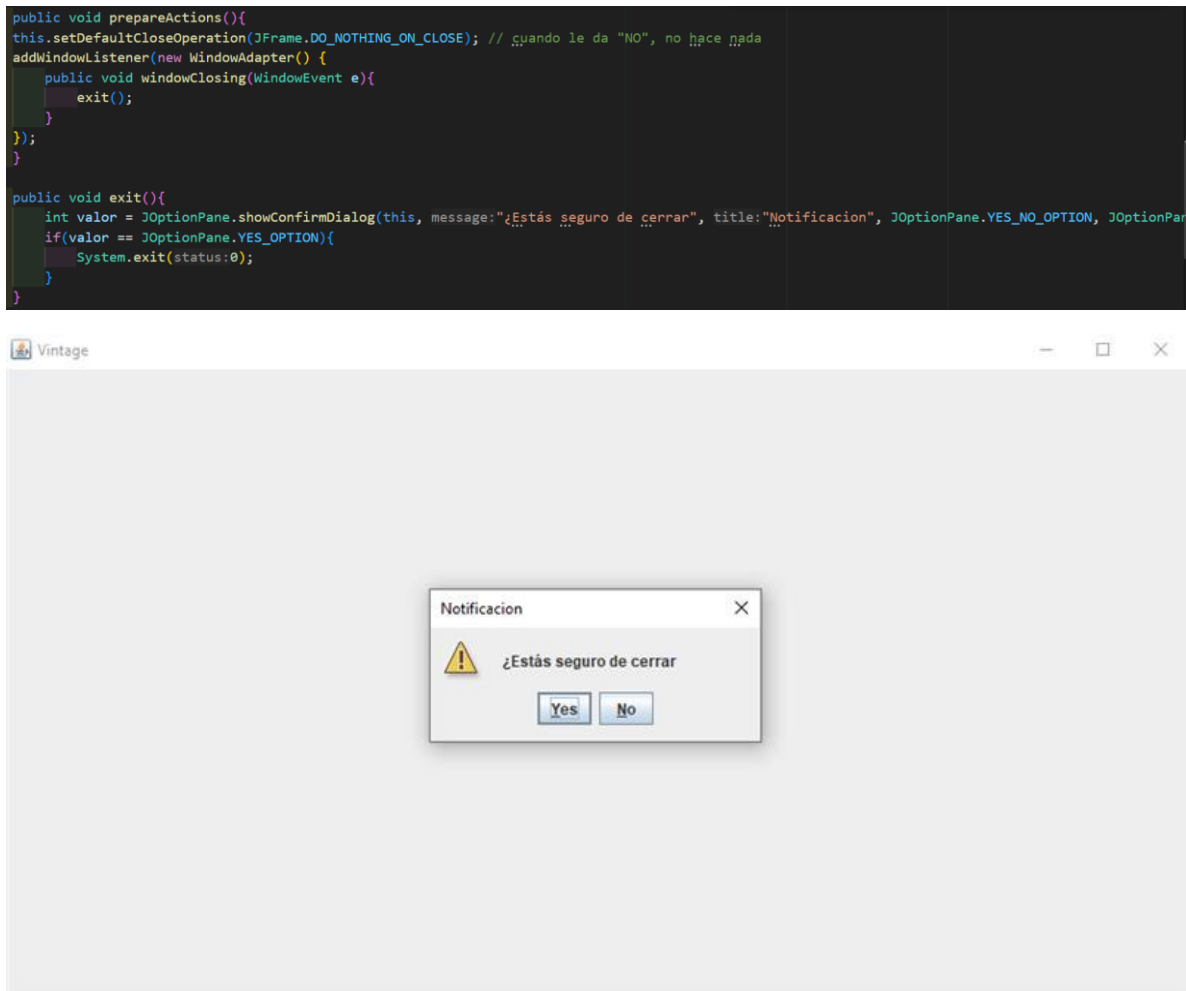
Si queremos confirmar el cierre, solo tenemos que devolver un mensaje de confirmación si quiere cerrar o no.

¿Cómo lo usarían si queremos simplemente cerrar la aplicación?

Solo le damos sobre el si de la confirmación y debería cerrar de igual forma en el panel de control.

5. Preparen el “oyente” correspondiente al icono cerrar que le pida al usuario que confirme su selección. Para eso inicien la codificación del método `prepareActions` y el método asociado a la acción (`exit`). Ejecuten el programa y cierren el programa.

Capturen las pantallas.



Ciclo 1: Ventana con menú – Salir

[En *.java y lab05.doc]

El objetivo es implementar un menú clásico para la aplicación con un final adecuado desde la opción del menú para salir. El menú debe ofrecer mínimo las siguientes opciones :Nuevo, Abrir – Salvar y Salir . Incluyan los separadores de opciones.

1. *Expliquen los componentes visuales necesarios para este menú.*

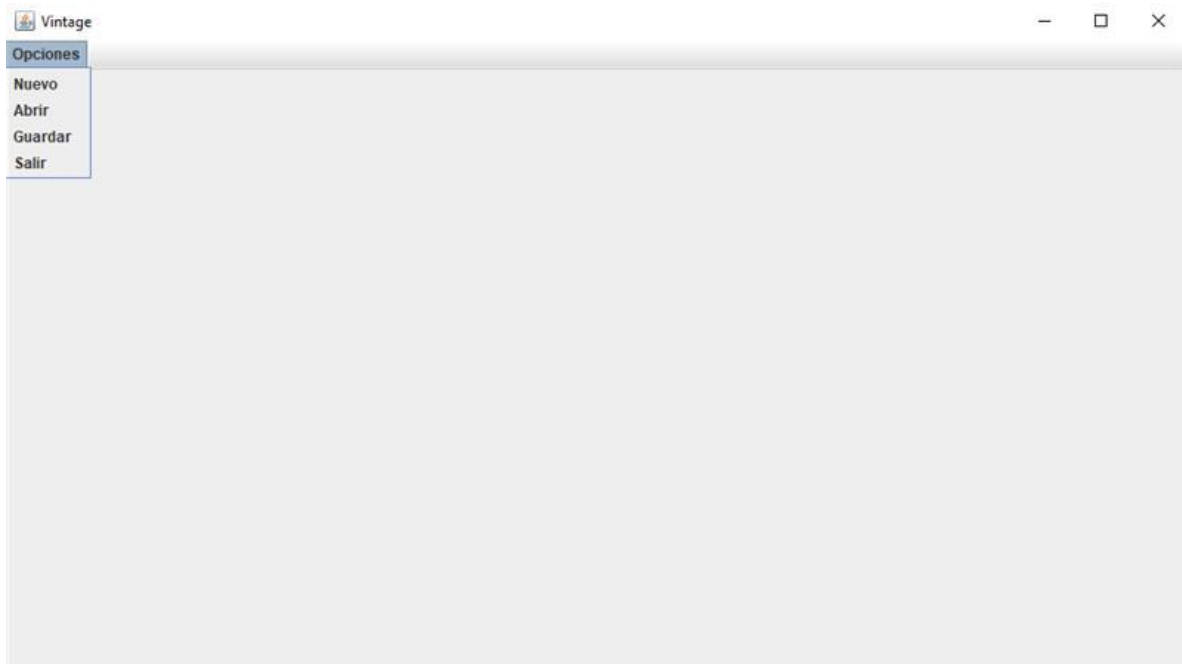
¿Cuáles serían atributos y cuáles podrían ser variables del método `prepareElements`? Justifique.

No tenemos atributos ya que no necesitamos de alguno para poder ejecutarlo.

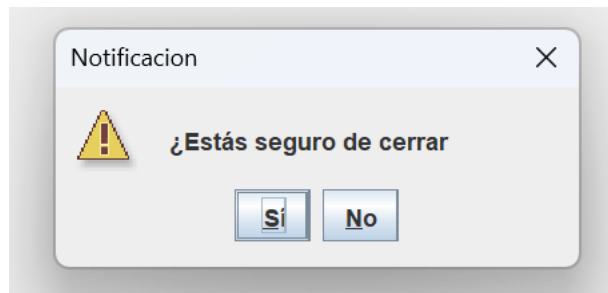
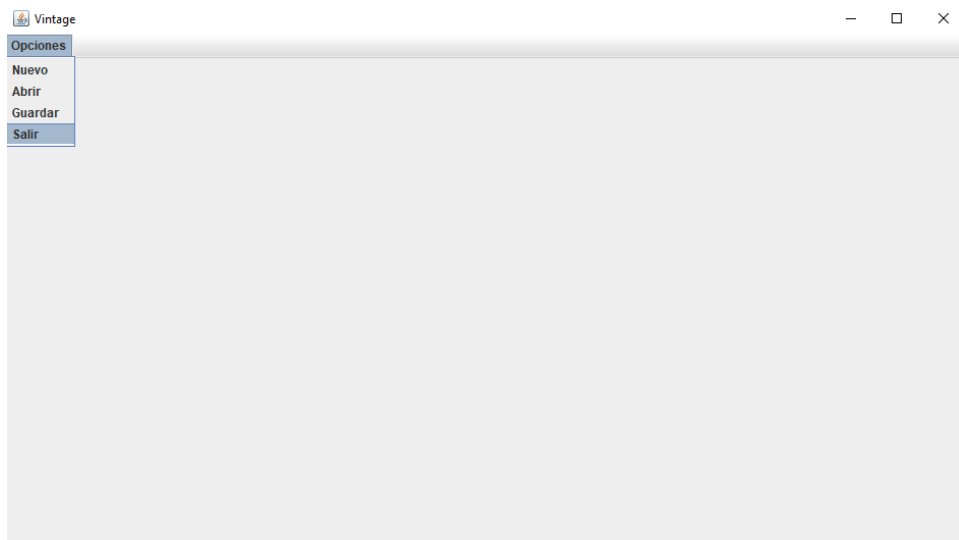
Nuestras variables son: o1, o2, o3, o4, opciones y mb (que vendría siendo nuestro menú), ya que tenemos que agregar cada una la de las opciones que se nos piden: Nuevo, Abrir, Salvar, Salir.

2. *Construya la forma del menú propuesto (`prepareElements` - `prepareElementsMenu`) .*

Ejecuten. Capturen la pantalla.



3. Preparen el “oyente” correspondiente al icono cerrar con confirmación (prepareActions - prepareActionsMenu). Ejecuten el programa y salgan del programa. Capturen las pantallas.



```

private void prepareActions(){
    this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE); // cuando le da "NO", no hace nada

    o4.addActionListener(
        new ActionListener() {
            public void actionPerformed(ActionEvent e){
                exit();
            }
        });

    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e){
            exit();
        }
    });
}

```

```

private void prepareElementsMenu(){
    setLayout(manager:null);
    mb = new JMenuBar();
    setJMenuBar(mb);
    opciones = new JMenu(s:"Opciones");
    mb.add(opciones);
    o1 = new JMenuItem(text:"Nuevo");
    opciones.add(o1);
    o2 = new JMenuItem(text:"Abrir");
    opciones.add(o2);
    o3 = new JMenuItem(text:"Guardar");
    opciones.add(o3);
    o4 = new JMenuItem(text:"Salir");
    opciones.add(o4);
}

```

Ciclo 2: Salvar y abrir

[En *.java y lab05.doc]

El objetivo es preparar la interfaz para las funciones de persistencia

1. Detalle el componente *JFileChooser* especialmente los métodos : *JFileChooser*, *showOpenDialog*, *showSaveDialog*, *getSelectedFile*.

JFileChooser es una clase en Java que proporciona una interfaz gráfica para que los usuarios elijan archivos o directorios. Aquí hay una descripción detallada de los métodos que se mencionan:

- Constructor *JFileChooser()* :

Este es el constructor predeterminado de la clase *JFileChooser*. Crea un objeto *JFileChooser* que se puede utilizar para mostrar un cuadro de diálogo de selección de archivos o directorios.

- Método *showOpenDialog(Component parent)* :

- a) Muestra un cuadro de diálogo para que el usuario seleccione un archivo para abrir. El método devuelve un valor que indica si el usuario seleccionó un archivo y si sí, cómo se realizó la selección.
- b) parentComponent: Componente que se utilizará como el propietario del cuadro de diálogo. Puede ser null.
- c) El valor devuelto (result) puede ser JFileChooser.APPROVE_OPTION si el usuario selecciona un archivo, JFileChooser.CANCEL_OPTION si el usuario cierra el cuadro de diálogo o elige cancelar, etc.
- Método showSaveDialog(Component parent) :

Muestra un cuadro de diálogo para que el usuario seleccione un archivo para guardar. Al igual que showOpenDialog(), devuelve un valor que indica cómo se realizó la selección.

- Método getSelectedFile() :

Devuelve el archivo que el usuario seleccionó en el cuadro de diálogo. Devuelve null si el usuario cancela el cuadro de diálogo o no selecciona ningún archivo.

2. Implementen parcialmente los elementos necesarios para salvar y abrir. Al seleccionar los archivos indiquen que las funcionalidades están en construcción detallando la acción y el nombre del archivo seleccionado.

Abrir:

```
o2.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e){
            open();
        }
    });

public void open(){
    JFileChooser fileChooser = new JFileChooser();
    int response = fileChooser.showOpenDialog(o2);
    File file = null;
    if (response == JFileChooser.APPROVE_OPTION){
        file = new File (fileChooser.getSelectedFile().getAbsolutePath());
        System.out.println(file);
    }
    JOptionPane.showMessageDialog(fileChooser, "La funcionalidad de abrir el archivo: "+file.getName()+" está en preparación.",title:"Notificación")
}
```

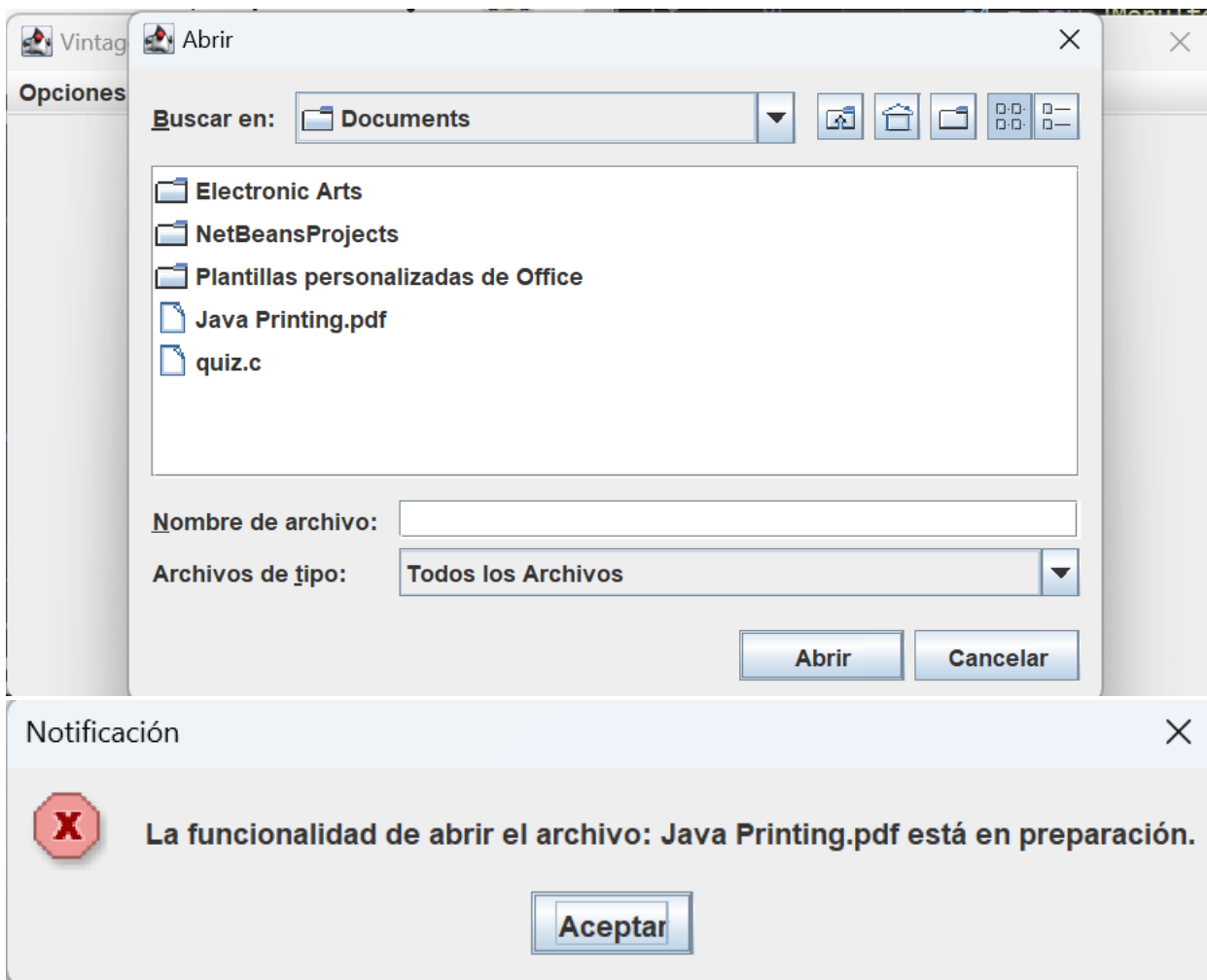
Guardar:

```
public void save(){
    int response = fileChooser.showSaveDialog(o3);
    if (response == JFileChooser.APPROVE_OPTION){
        file = new File (fileChooser.getSelectedFile().getAbsolutePath());
        System.out.println(file);
    }
    JOptionPane.showMessageDialog(fileChooser, "La funcionalidad de guardar el archivo: "+file.getName()+" está en preparación.",title:"Notificación")
}
```

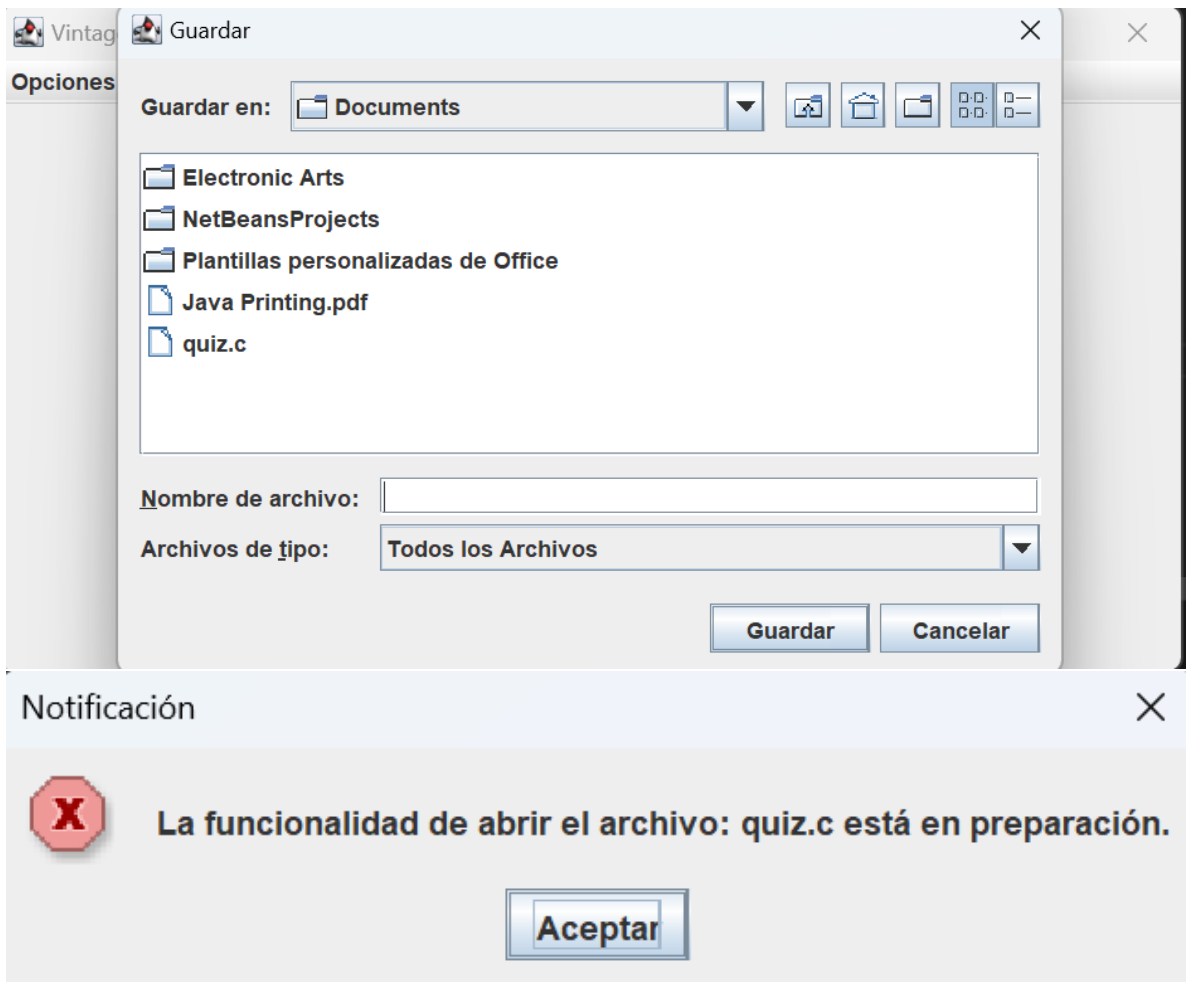
```
o3.addActionListener(  
    new ActionListener() {  
        public void actionPerformed(ActionEvent e){  
            save();  
        }  
    });
```

3. Ejecuten las dos opciones y capturen las pantallas más significativas.

Abrir:



Guardar:

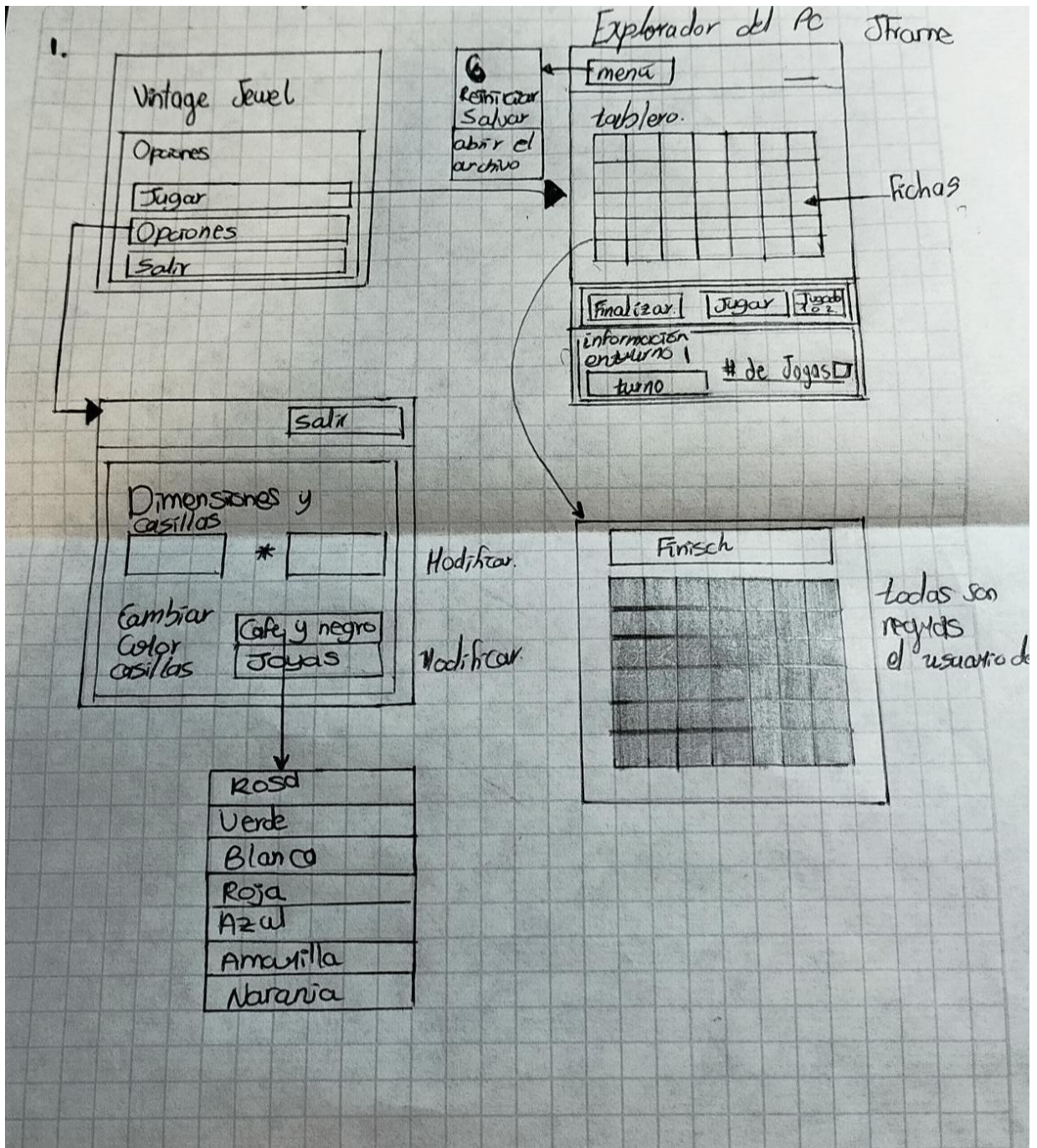


Ciclo 3: Forma de la ventana principal

[En *.java y lab05.doc]

El objetivo es codificar el diseño de la ventana principal (todos los elementos de primer nivel)

1. *Presenten el bosquejo del diseño de interfaz con todos los componentes necesarios.*



2. Continúe con la implementación definiendo los atributos necesarios y extendiendo el método `prepareElements()`.

Para la zona del tablero defina un método `prepareElementsBoard()` y un método `refresh()` que actualiza la vista del tablero considerando, por ahora, el tablero inicial por omisión. Este método lo vamos a implementar realmente en otros ciclos.

```

private void prepareElementsBoard(){
    setLayout(new BorderLayout());
    add(new Button(label:"Refrescar"), BorderLayout.NORTH);
    add(new Button(label:"Información del jugador"), BorderLayout.SOUTH);
    add(new Button(label:"Color"), BorderLayout.EAST);
    add(new Button(label:"# de jugadas"), BorderLayout.WEST);

    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout(rows:12, cols:12));

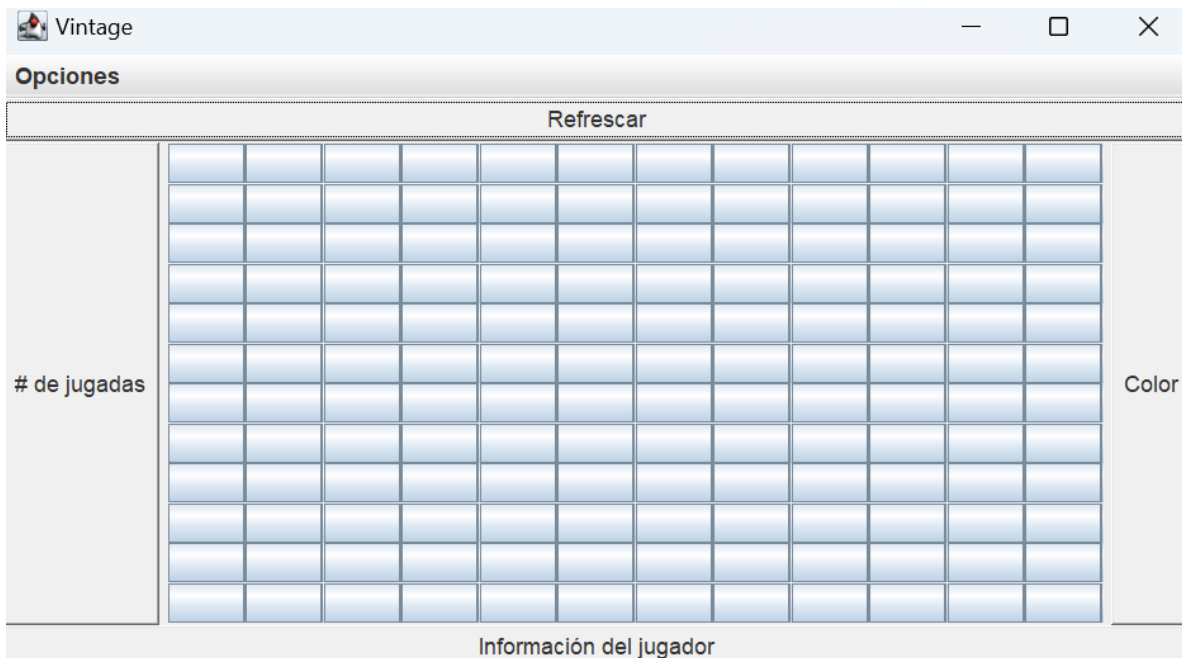
    JButton[][] board = new JButton[12][12];
    for (int i = 0; i < 12; i++) {
        for (int j = 0; j < 12; j++) {
            board[i][j] = new JButton();
            buttonPanel.add(board[i][j]);
        }
    }

    add(buttonPanel, BorderLayout.CENTER);
}

public void refresh() {
    System.out.println(x:"Se ha actualizado el tablero.");
}

```

3. Ejecuten y capturen esta pantalla.



Ciclo 4: Cambiar color

[En *.java y lab05.doc]

El objetivo es implementar este caso de uso.

1. Expliquen los elementos (vista – controlador) necesarios para implementar este caso de uso.

Modelo:

En este caso, el modelo podría ser la lógica subyacente de tu juego, el estado del tablero, las reglas del juego, etc. Aunque no está completamente definido en tu código proporcionado, asumiré que hay una lógica del juego detrás de la interfaz gráfica.

Vista:

La vista es la representación gráfica de tu aplicación. En tu caso, la interfaz gráfica que ha construido con botones, menús y un tablero.

Controlador:

El controlador maneja las interacciones del usuario y actúa como intermediario entre la vista y el modelo. En tu código, los métodos `open()`, `save()`, y `exit()` en conjunto con los `ActionListener` podrían considerarse parte del controlador. Sin embargo, puedes modularizar aún más las acciones en un controlador separado si tu aplicación se vuelve más compleja.

2. Detalle el comportamiento de `JColorChooser` especialmente el método estático `showDialog`

El método estático `showDialog` de la clase `JColorChooser` se utiliza para mostrar un diálogo que permite al usuario seleccionar un color.

`showDialog(Component parent, String title, Color initialColor):`

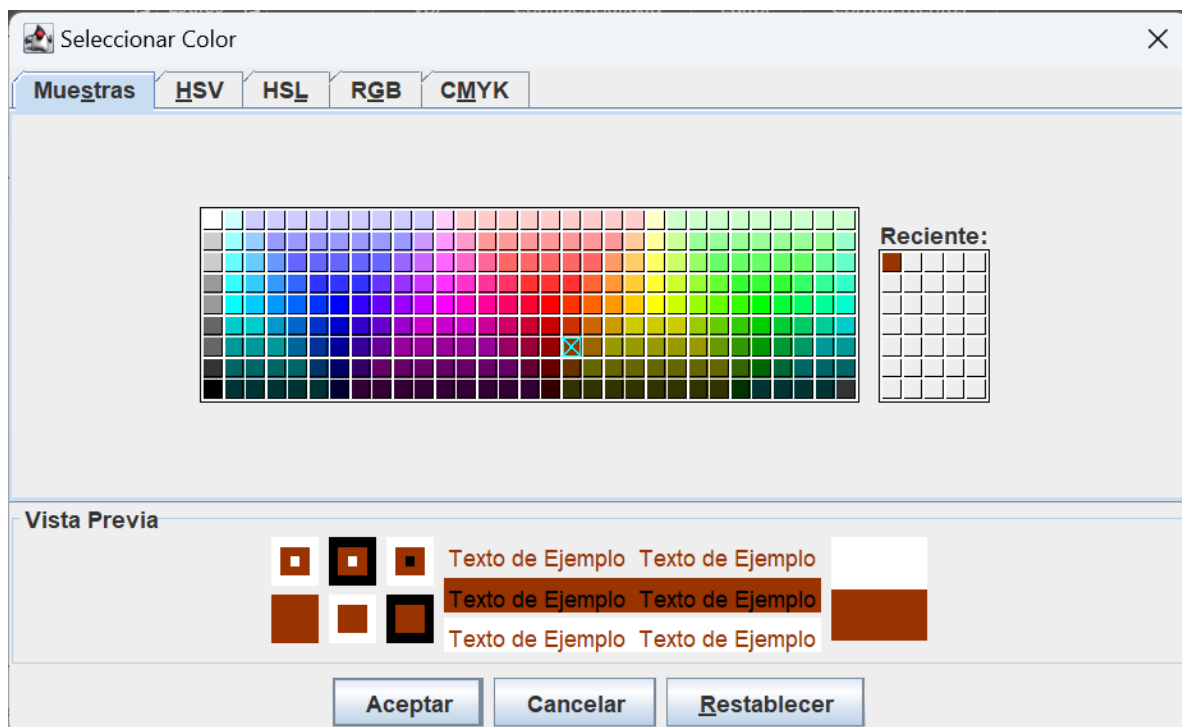
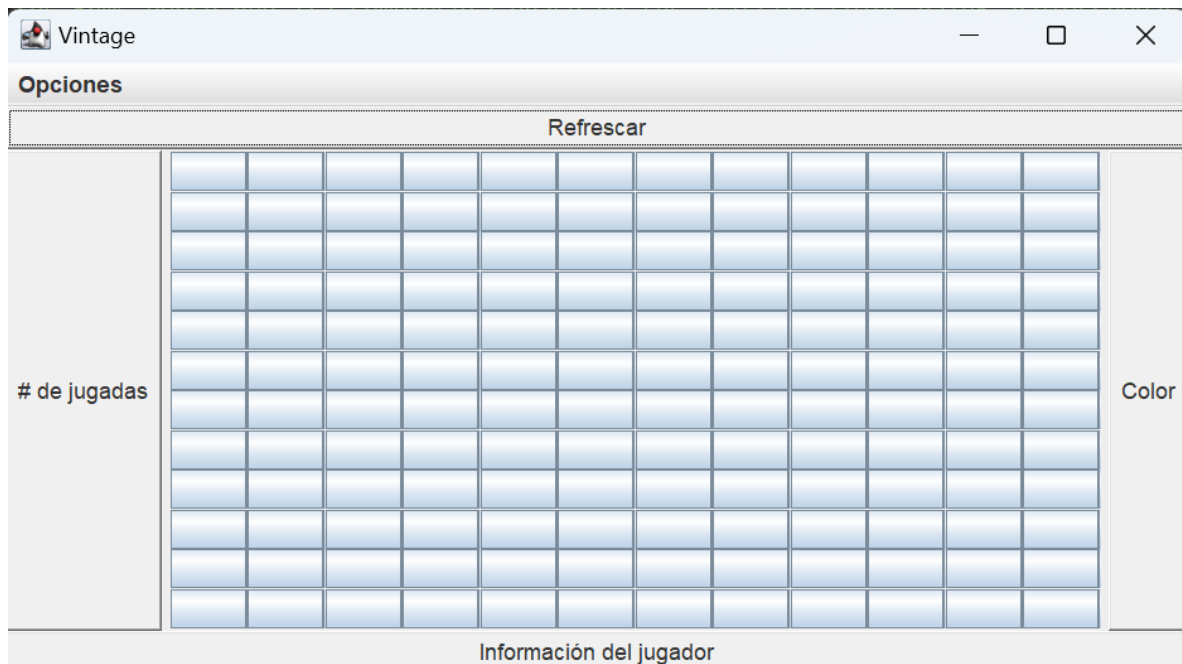
- `parent`: El componente padre que se utilizará como el propietario del cuadro de diálogo. Puede ser `null`.
- `title`: El título del cuadro de diálogo.
- `initialColor`: El color inicial que se muestra en el cuadro de diálogo.

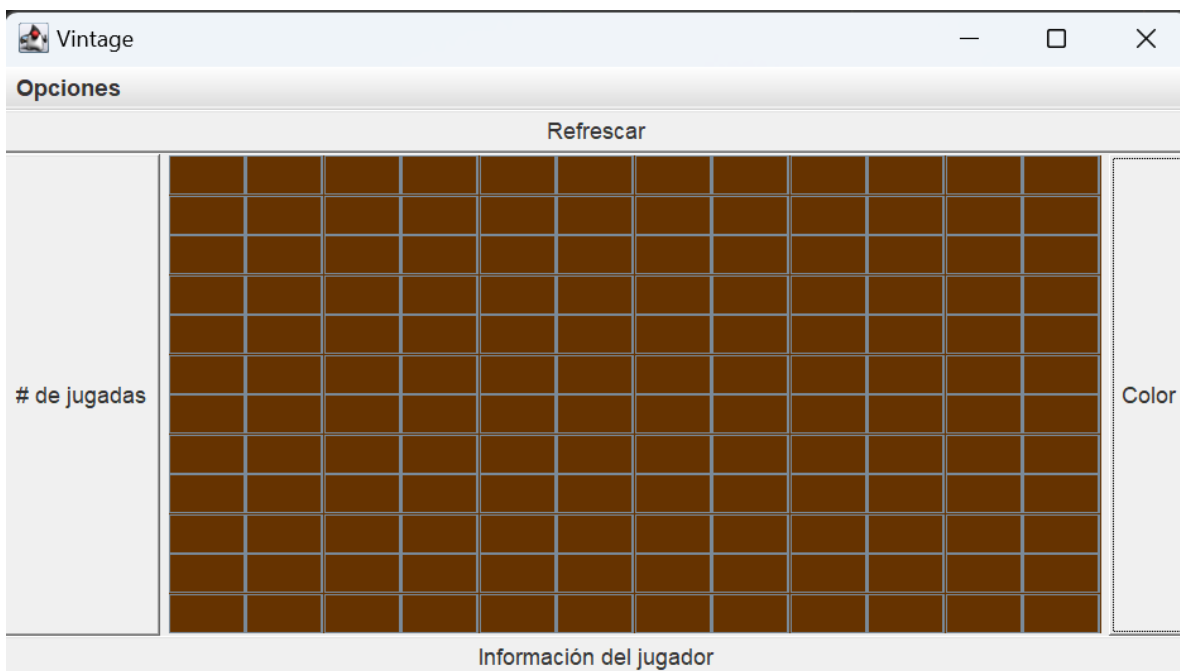
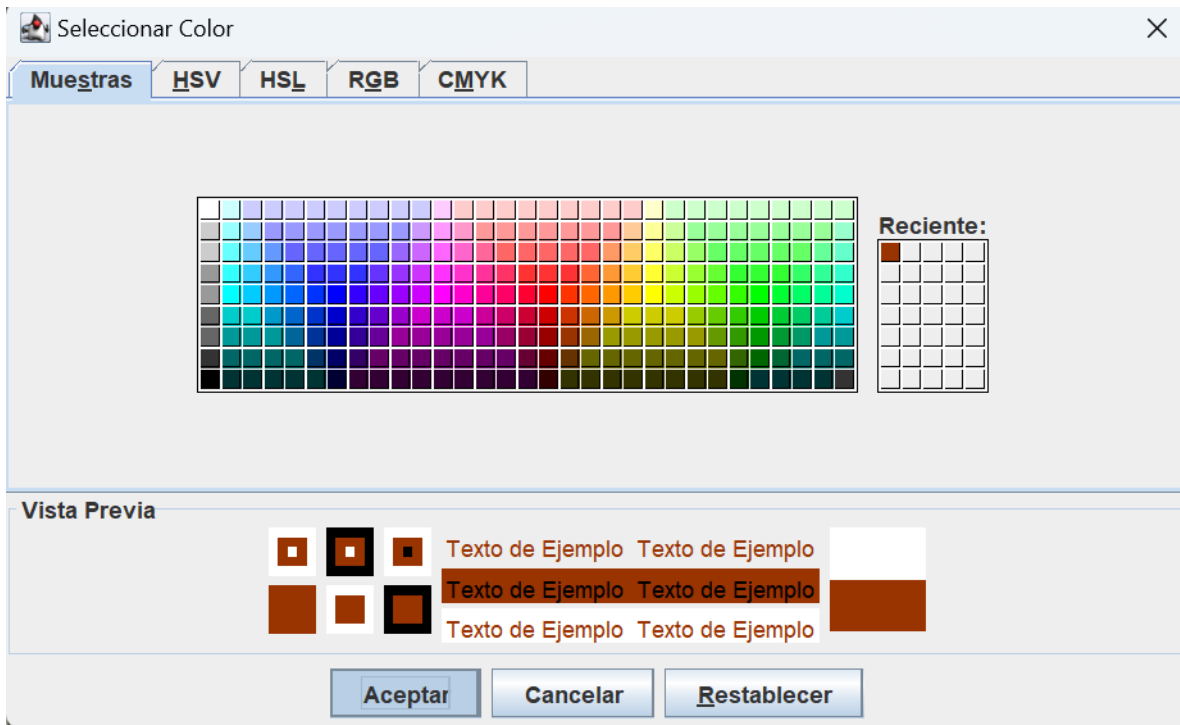
Devuelve el color seleccionado por el usuario o `null` si el usuario cancela el diálogo.

3. Implementen los componentes necesarios para cambiar el color del tablero y de las fichas.

```
/**
 * Changes the color of the game board and its components based on the selected color using JColorChooser.
 * The method prompts the user to choose a color and updates the background of the board and individual buttons.
 */
private void changeBoardColor() {
    Color selectedColor = JColorChooser.showDialog(this, title: "Seleccionar Color", Color.WHITE);
    for (int i = 0; i < heightBoard; i++) {
        for (int j = 0; j < widthBoard; j++) {
            board[i][j].setBackground(selectedColor);
            this.setVisible(b:true);
        }
    }
}
```

4. Ejecuten el caso de uso y capture las pantallas más significativas





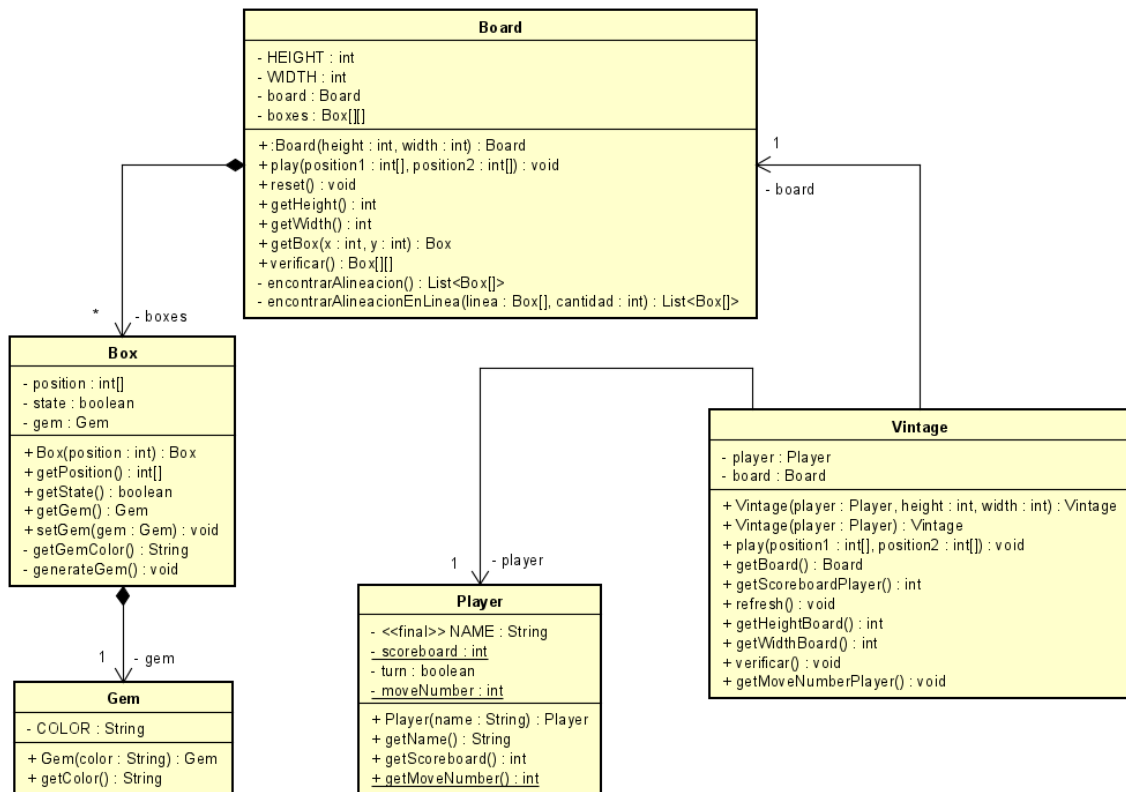
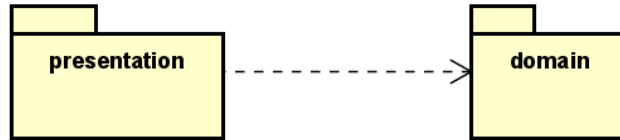
Ciclo 5: Modelo Vintage

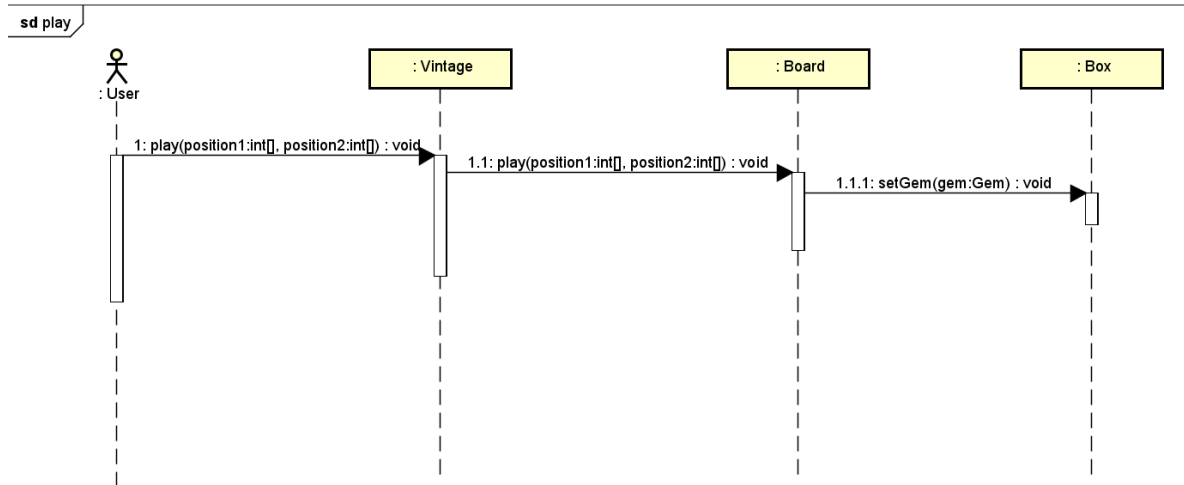
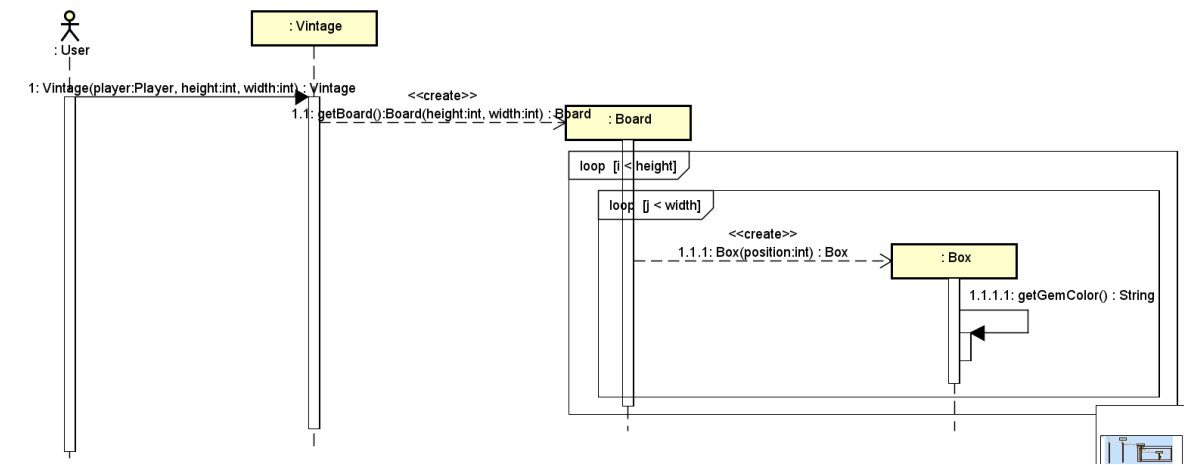
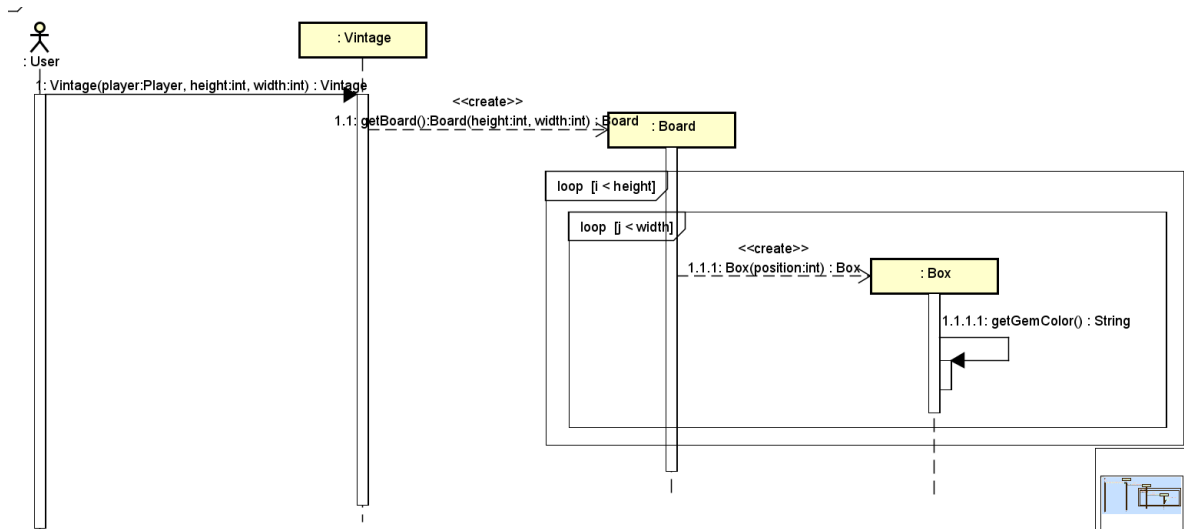
[En *.java y lab05.doc]

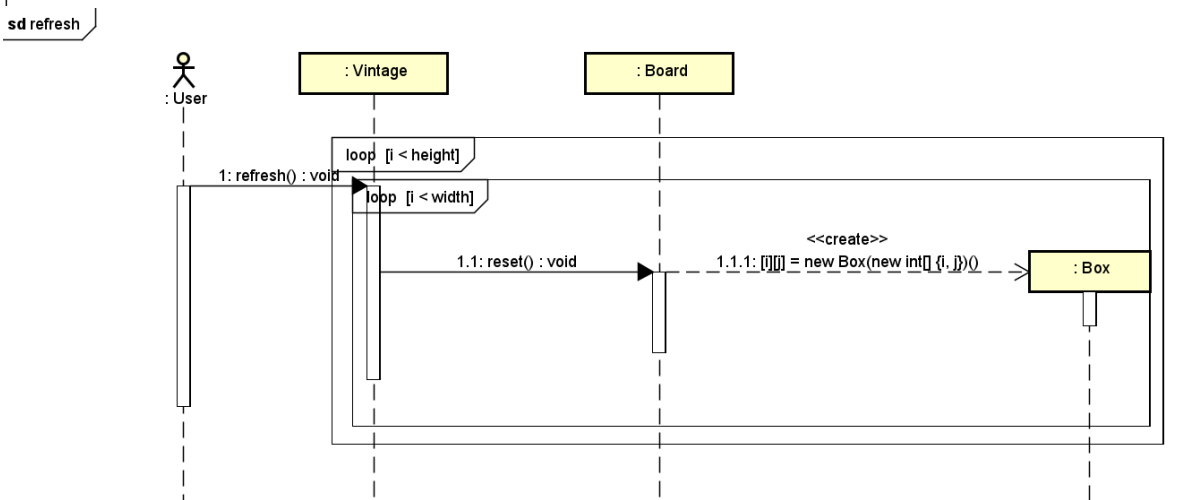
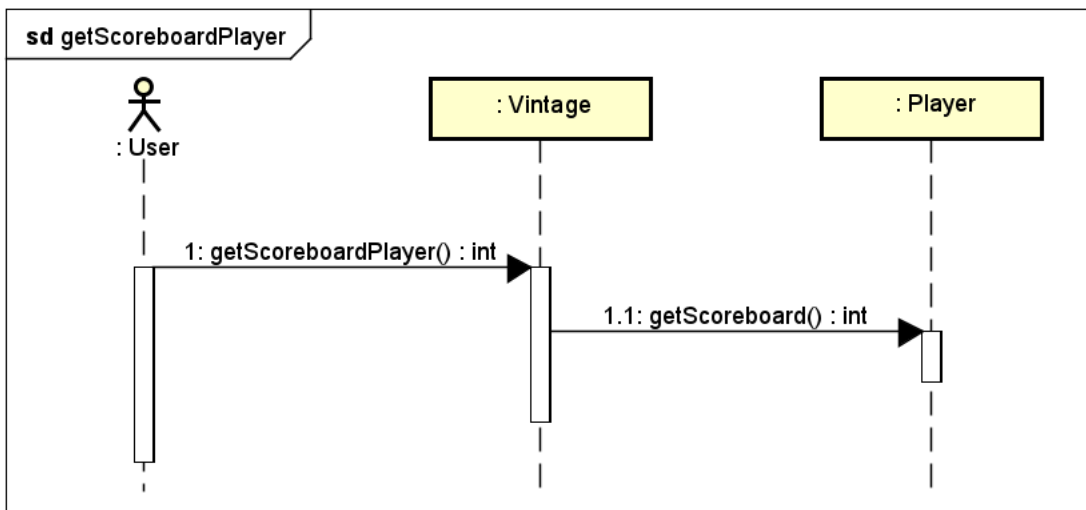
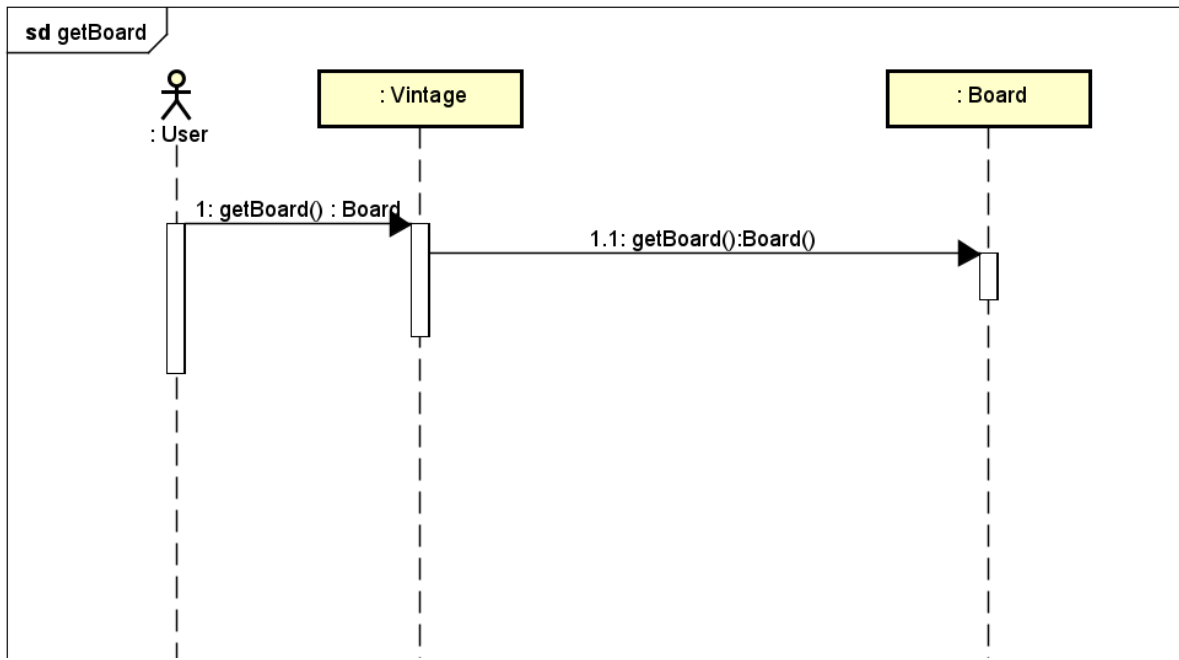
El objetivo es implementar la capa de dominio para Vintage.

1. Construya los métodos básicos del juego (No olvide MDD y TDD)

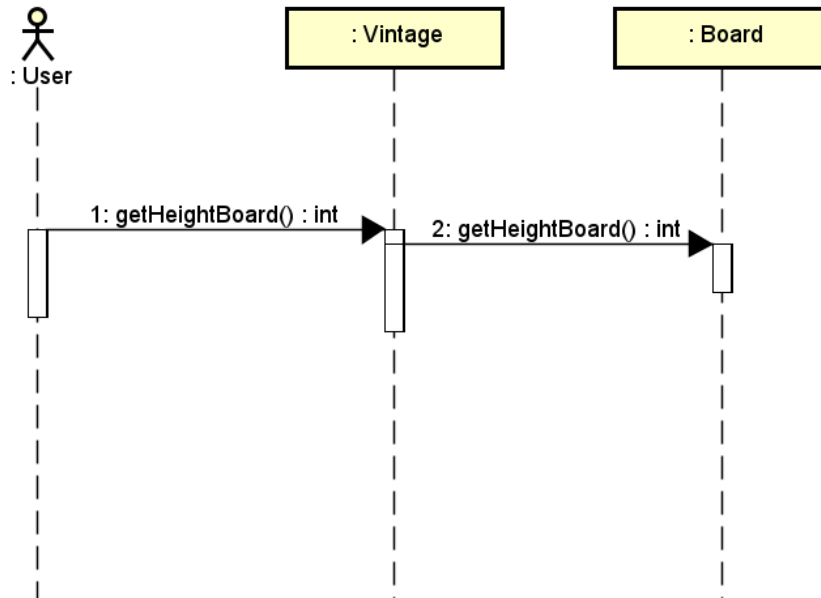
pkg



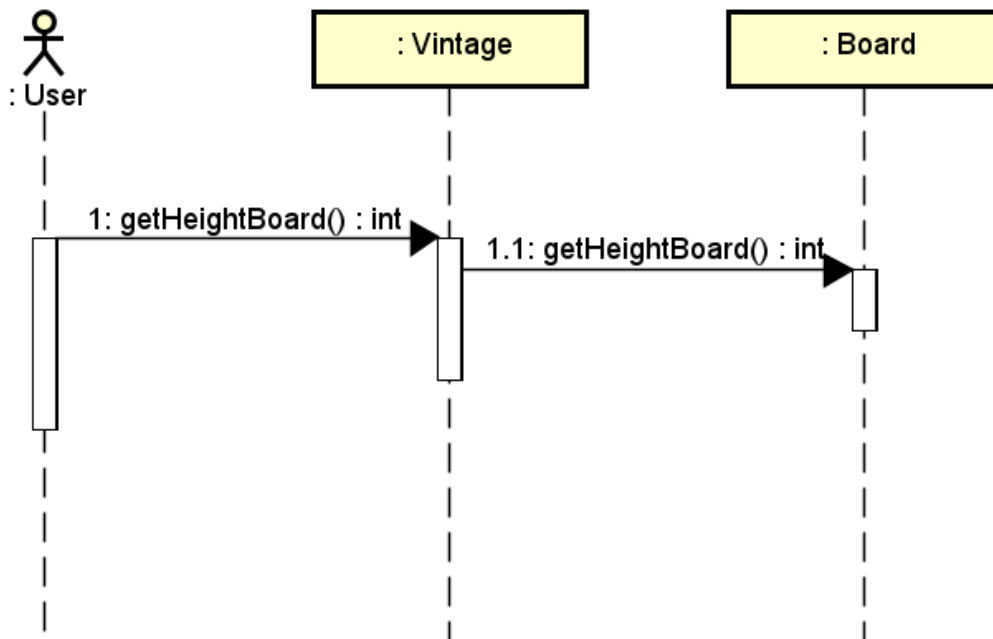


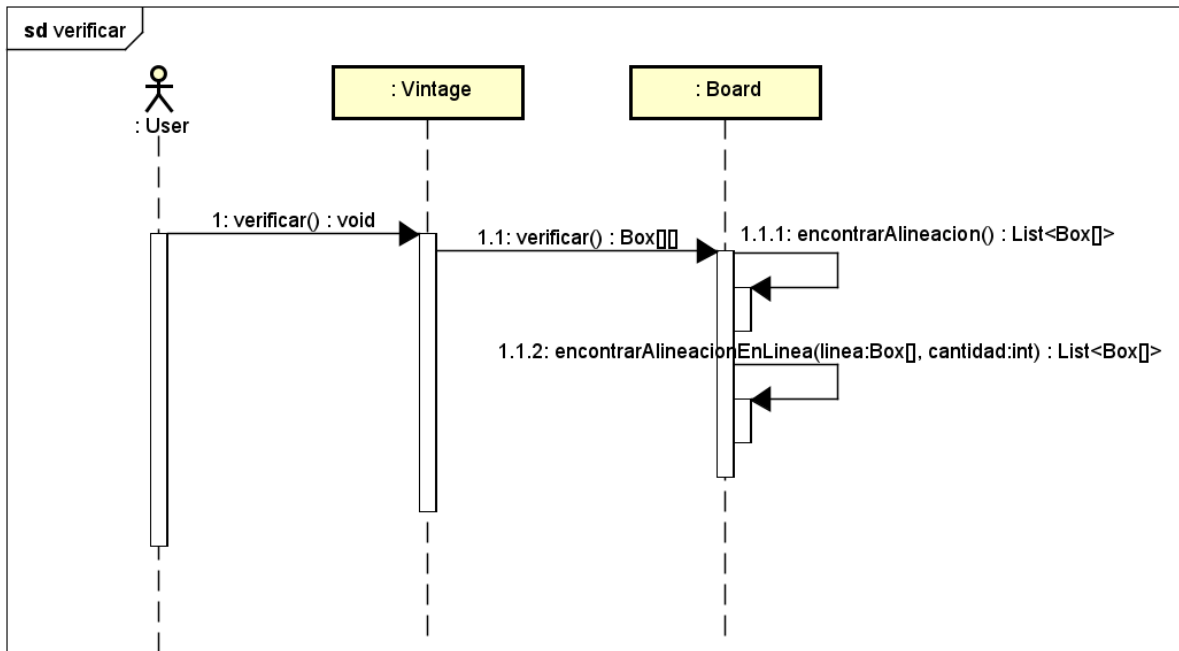


sd getHeightBoard

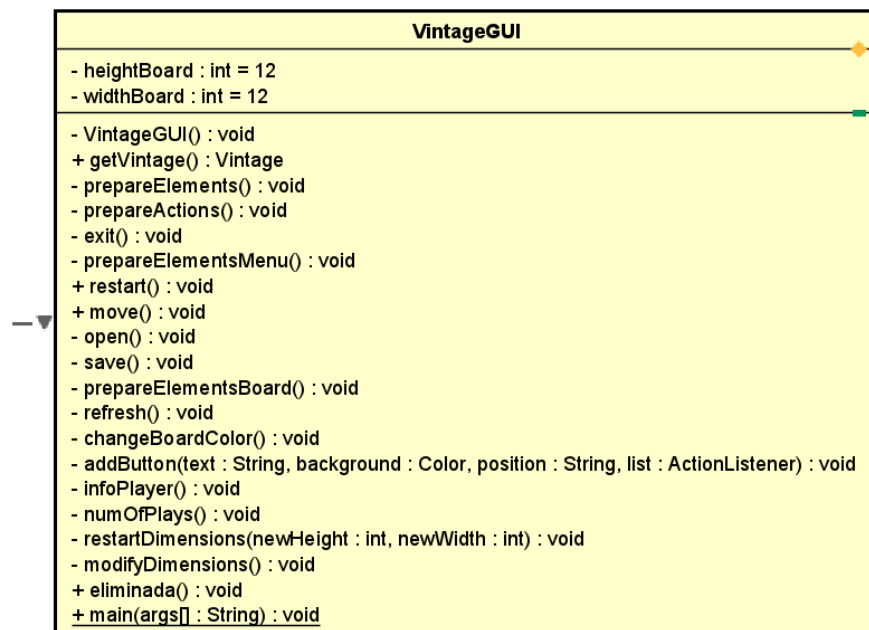


sd getWidthBoard





pkg presentation



2. Ejecuten las pruebas y capturen el resultado.

```
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import domain.*;
class VintageTest {

    private Vintage vintage;
    private Player player;

    @BeforeEach
    public void setUp() {
        player = new Player("TestPlayer");
        vintage = new Vintage(player);
    }

    @Test
    public void play_validMove_shouldMoveGems() {
        int[] initialPosition = {0, 0};
        int[] finalPosition = {1, 1};
        vintage.play(initialPosition, finalPosition);
        assertNotNull(vintage.getBoard().getBox(finalPosition[0], finalPosition[1]).getGem());
    }

    @Test
    public void getScoreboardPlayer_initialScore_shouldReturnZero() {
        assertEquals(0, vintage.getScoreboardPlayer());
    }

    @Test
    public void refresh_resetBoard_shouldHaveEmptyBoard() {
        int[] initialPosition = {0, 0};
        int[] finalPosition = {1, 1};
        vintage.play(initialPosition, finalPosition);
        vintage.refresh();
        assertNotNull(vintage.getBoard().getBox(initialPosition[0], initialPosition[1]).getGem());
    }

    @Test
    public void getHeightBoard_initialBoard_shouldReturnEight() {
        assertEquals(8, vintage.getHeightBoard());
    }

    @Test
    public void getWidthBoard_initialBoard_shouldReturnEight() {
        assertEquals(8, vintage.getWidthBoard());
    }

    @Test
    public void play_invalidMove_shouldNotMoveGems() {
        int[] initialPosition = {0, 0};
        int[] finalPosition = {8, 8};
        assertEquals(8, vintage.getHeightBoard());
    }

    @Test
    public void getScoreboardPlayer_afterWinningMove_shouldIncreaseScore() {
        int[] initialPosition = {0, 0};
        int[] finalPosition = {1, 1};
        vintage.play(initialPosition, finalPosition);
        assertFalse(vintage.getScoreboardPlayer() > 0);
    }
}
```

```
@test
public void refresh_afterMultipleMoves_shouldHaveEmptyBoard() {
    int[] initialPosition1 = {0, 0};
    int[] finalPosition1 = {1, 1};
    int[] initialPosition2 = {2, 2};
    int[] finalPosition2 = {3, 3};
    vintage.play(initialPosition1, finalPosition1);
    vintage.play(initialPosition2, finalPosition2);
    vintage.refresh();
    assertNotNull(vintage.getBoard().getBox(initialPosition1[0], initialPosition1[1]).getGem());
}
```

```
laura@Lalita MINGW64 ~/OneDrive - ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVITO/2023-2 6to semestre/Programación Orientada a Objetos/TERCIO 3/LAB 05/Vintage
$ java -cp ".;lib\junit-4.8.jar;lib\hamcrest-2.2.jar;bin" org.junit.runner.JUnit4
Core test.VintageTest
JUnit version 4.8.2
Time: 0,039

OK (5 tests)
```

Ciclo 6: Jugar

[En *.java y lab05.doc]

El objetivo es implementar el caso de uso jugar.

1. *Adicione a la capa de presentación el atributo correspondiente al modelo.*

Como tenemos nuestra capa de presentación y dominio, entonces necesitamos conectarla, de tal forma que esté relacionada entre si y pueda ir a la par con las acciones que se hagan al momento de jugar, reiniciar o refrescar el juego.

```
/**
 * Constructs a new VintageGUI instance, initializing the graphical user interface.
 */
private VintageGUI(){
    vintage = new Vintage(8,8);
    prepareElements();
    prepareActions();
}
```

2. *Perfeccionen el método refresh() considerando la información del modelo de dominio.*

Para la clase Vintage:

```
/**
 * Resets the state of the board.
 */
public void refresh() {
    board.reset();
}
```

Para la clase Board:

```

/**
 * Resets the state of the board.
 */
public void reset() {
    Random random = new Random();
    for (int i = 0; i < HEIGHT; i++) {
        for (int j = 0; j < WIDTH; j++) {
            boolean isBlack = (i + j) % 2 == 0;
            boxes[i][j] = new Box(new int[]{i, j});
        }
    }
}

```

3. Expliquen los elementos necesarios para implementar este caso de uso.

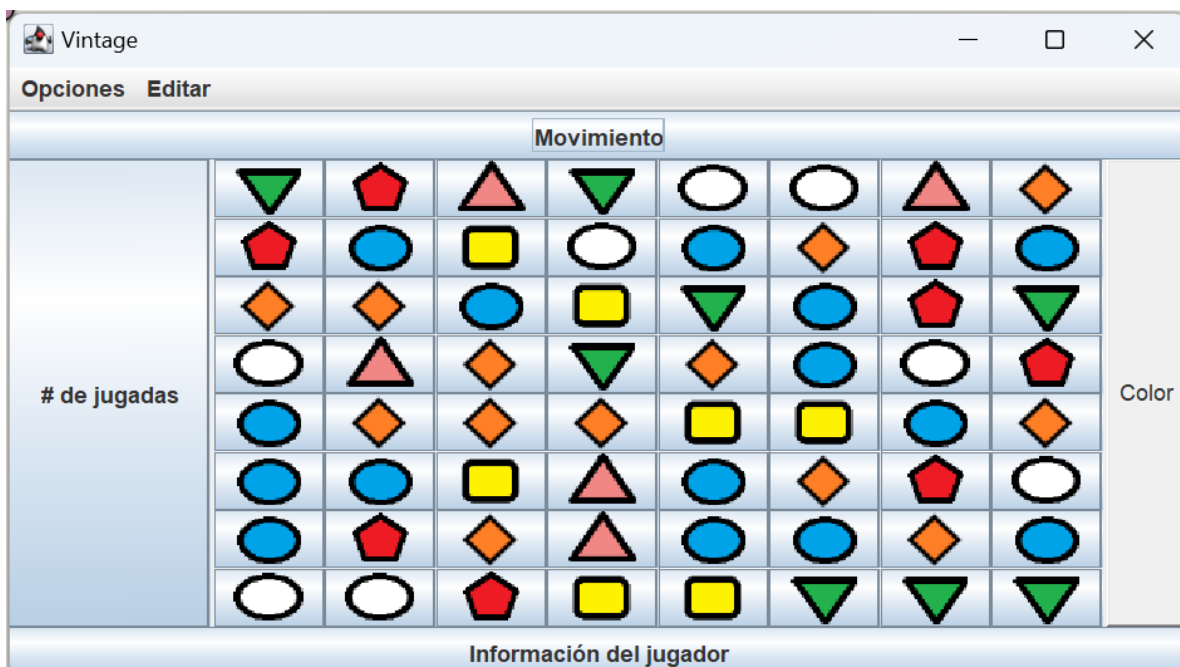
Es importante crear varios métodos para verificar varios estados del tablero,

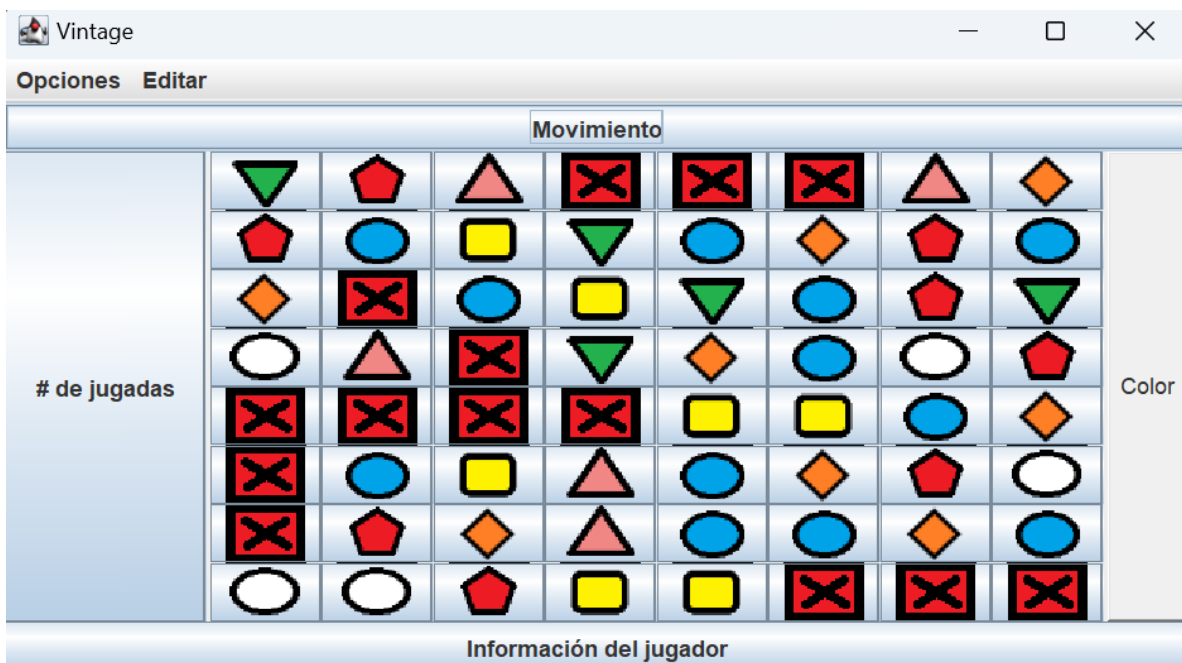
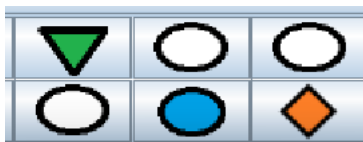
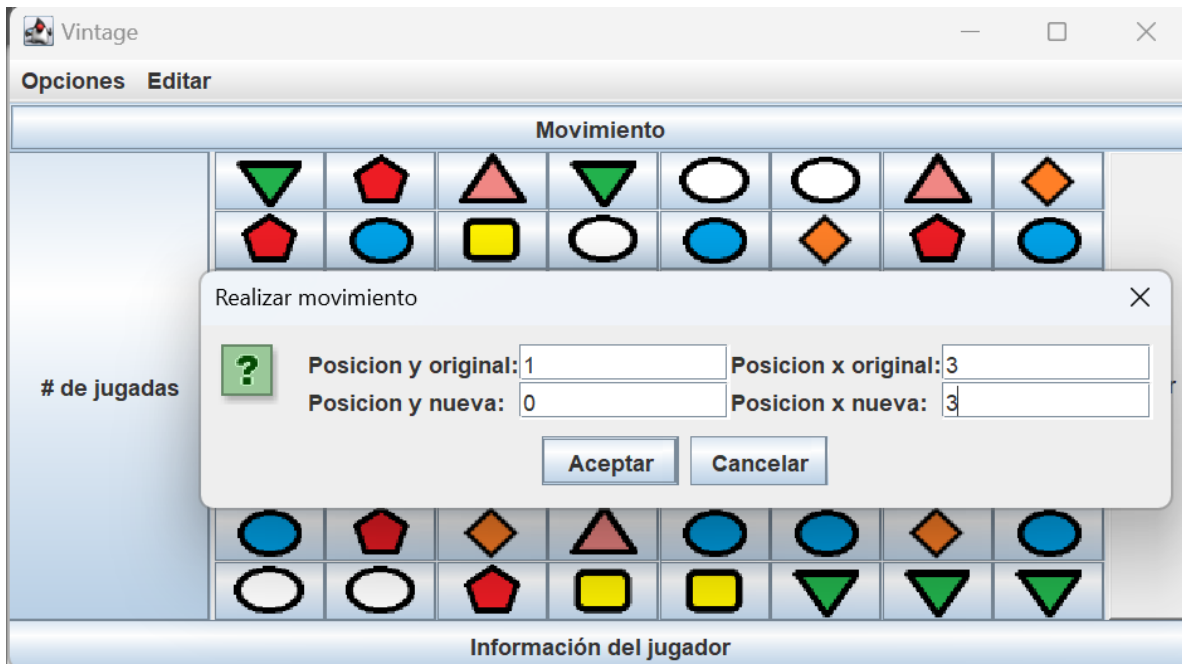
4. Implementen los componentes necesarios para jugar. ¿Cuántos oyentes necesitan?

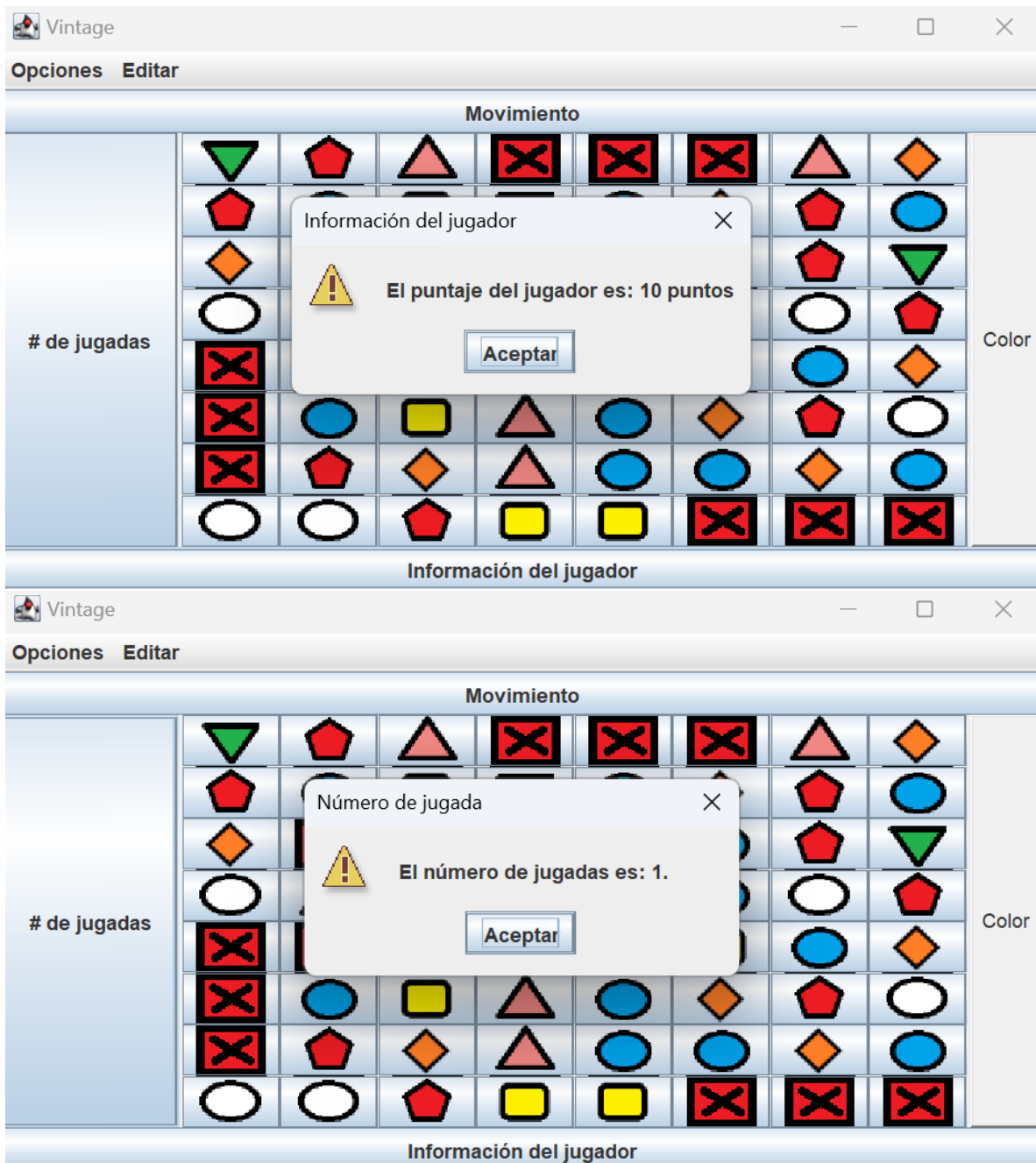
¿Por qué?

Solo implementamos un oyente, ya que necesitábamos obtener las posiciones para poder cambiar las fichas del tablero, solo se usa uno porque lo demás ya está implementado en la parte de dominio con su respectiva lógica.

5. Ejecuten el caso de uso y capture las pantallas más significativas.







Ciclo 7: Reiniciar

[En *.java y lab05.doc]

El objetivo es implementar este caso de uso.

1. Expliquen los elementos a usar para implementar este caso de uso.

Debemos implementar los Listener para saber sobre que botón se va a realizar la acción para poder reiniciar el tablero con sus respectivos valores e interfaz predeterminada, creando así un método con ciertas características que reinicie el tablero con valores iniciales o en 0.

2. Implementen los elementos necesarios para reiniciar.

```
/**
 * Refreshes the game board in the graphical user interface with default dimensions (8x8).
 */
private void restart() {
    restartDimensions(newHeight:8, newWidth:8);
}
```

```
infoPlayerListener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        infoPlayer();
    }
};

numOfPlaysListener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        numOfPlays();
    }
};

colorListener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        changeBoardColor();
    }
};
```

```

/**
 * Refreshes the game board in the graphical user interface with the specified dimensions.
 *
 * @param newHeight The new height of the game board.
 * @param newWidth The new width of the game board.
 */
private void restartDimensions(int newHeight, int newWidth) {
    heightBoard = newHeight;
    widthBoard = newWidth;
    Container contentPane = getContentPane();
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.CENTER));
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.SOUTH));
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.WEST));
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.EAST));
    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout(newHeight, newWidth));
    board = new JButton[newHeight][newWidth];
    for (int i = 0; i < newHeight; i++) {
        for (int j = 0; j < newWidth; j++) {
            Color darkCoffeeColor = new Color(rgb:0x3E2723);
            board[i][j] = new JButton();
            board[i][j].setBackground(darkCoffeeColor);
            buttonPanel.add(board[i][j]);
        }
    }
    Color lightBlueColor = new Color(r:173, g:216, b:230);
    add(buttonPanel, BorderLayout.CENTER);
    addButton(text:"Información del jugador", lightBlueColor, BorderLayout.SOUTH, infoPlayerListener);
    addButton(text:"# de jugadas", lightBlueColor, BorderLayout.WEST, numOfPlaysListener);
    addButton(text:"Color", lightBlueColor, BorderLayout.EAST, colorListener);
    revalidate();
    repaint();
}

```

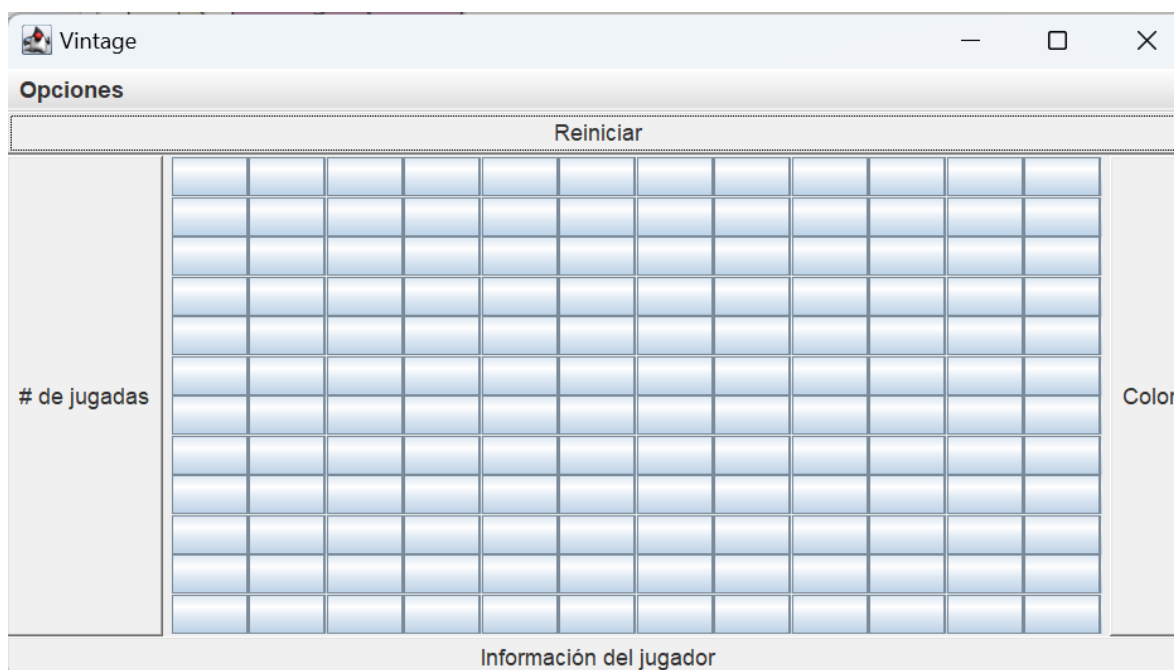
```

/**
 * Adds a JButton to the specified position in the BorderLayout of the container.
 *
 * @param text The text to be displayed on the button.
 * @param background The background color of the button.
 * @param position The position where the button should be added (e.g., BorderLayout.SOUTH).
 * @param listener The ActionListener to be attached to the button.
 */
private void addButton(String text, Color background, String position, ActionListener listener) {
    JButton button = new JButton(text);
    button.setBackground(background);
    button.setForeground(Color.BLACK);
    button.addActionListener(listener);
    switch (position) {
        case BorderLayout.SOUTH:
            add(button, BorderLayout.SOUTH);
            break;
        case BorderLayout.WEST:
            add(button, BorderLayout.WEST);
            break;
        case BorderLayout.EAST:
            add(button, BorderLayout.EAST);
            break;
        default:
            break;
    }
    revalidate();
    repaint();
}

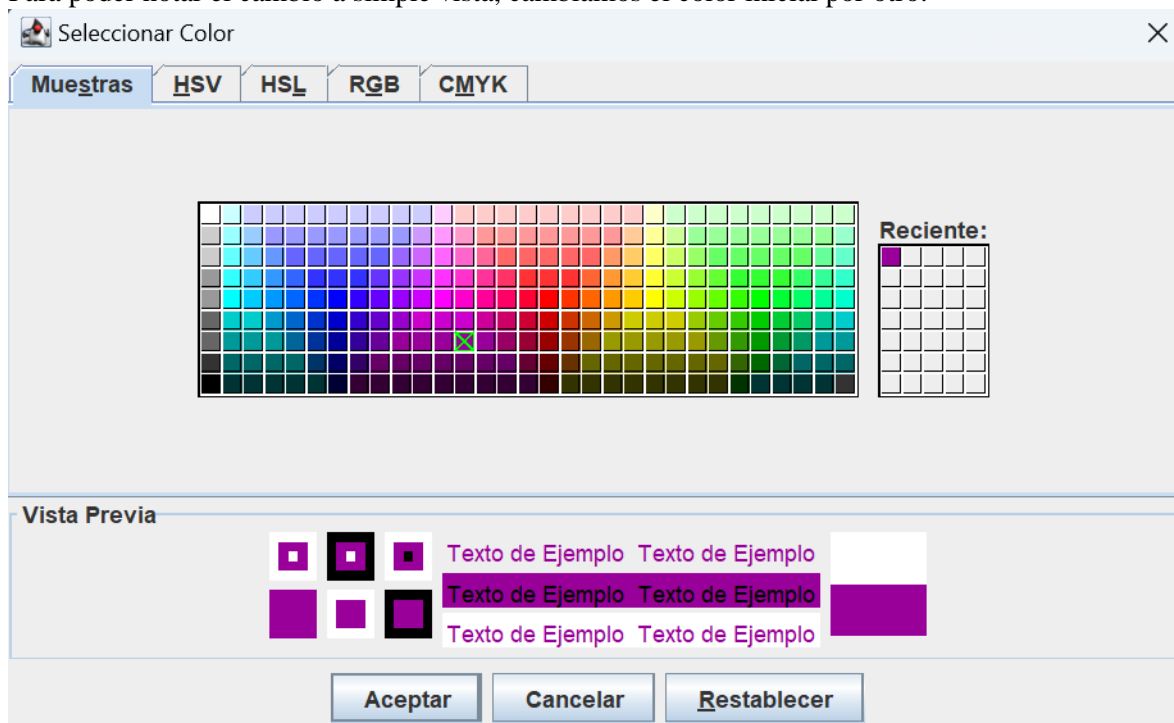
```

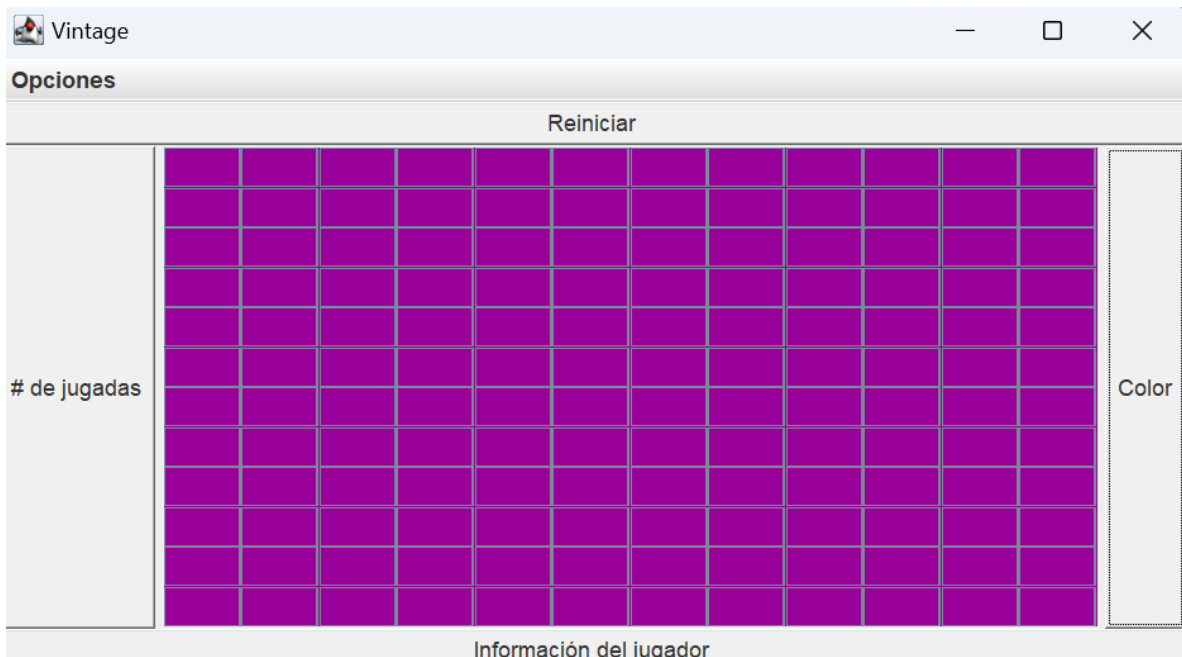
3. Ejecuten el caso de uso y capture las pantallas más significativas.

Se inicia el juego con total normalidad



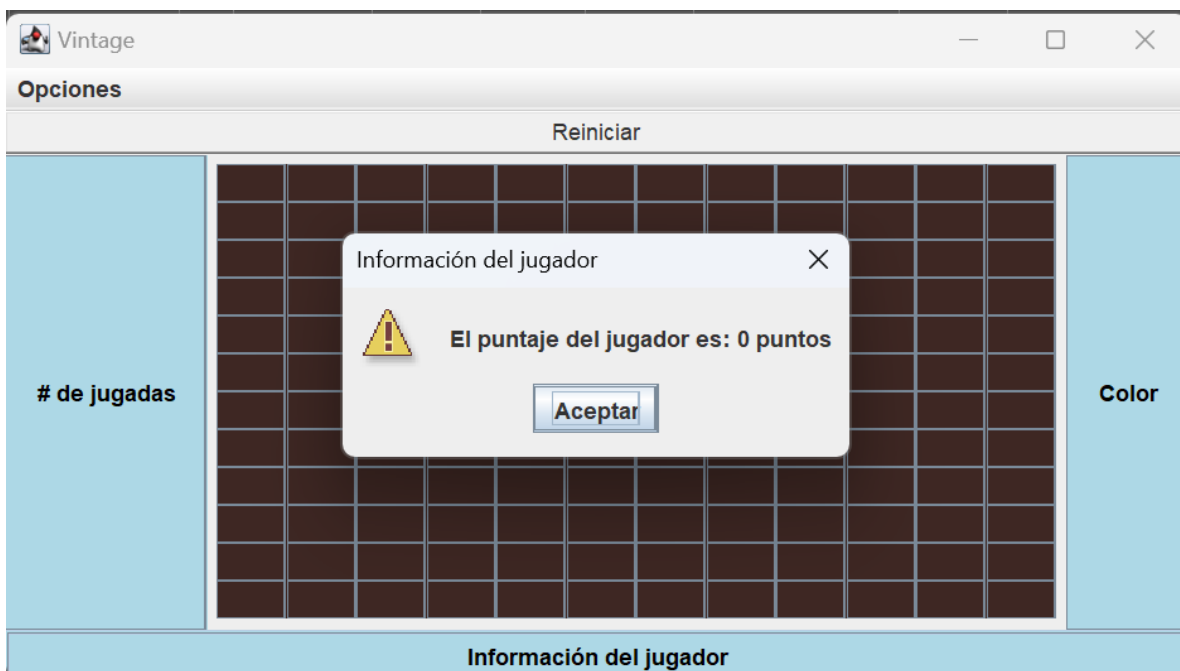
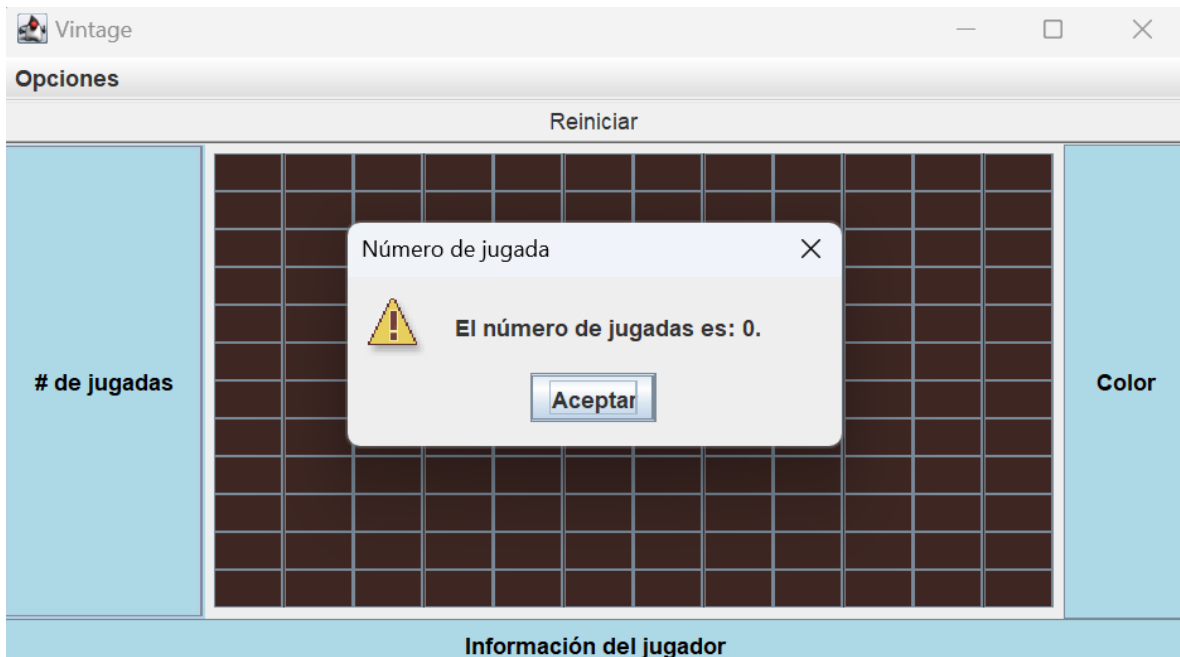
Para poder notar el cambio a simple vista, cambiamos el color inicial por otro:





Y luego le damos en reiniciar al tablero y se verá con ese color predeterminado:





Ciclo 8: Cambiar el tamaño

[En *.java y lab05.doc]

El objetivo es implementar este caso de uso.

1. Expliquen los elementos a usar para implementar este caso de uso.

Se debe crear un método para poder modificar las dimensiones iniciales de la matriz e implementar el método de reiniciar porque al cambiar las dimensiones, lo que estamos haciendo es que

obligatoriamente sea una partida nueva. De igual forma agregar el oyente correspondiente para que al darle en el botón de iniciar- modificar dimensiones, salga un cuadro de dialogo pidiendo las nuevas dimensiones; eso si no debe pasar de 20 su tamaño y solo admite valores de tipo entero.

2. *Implementen los elementos necesarios para cambiar el tamaño del juego.*

```
o5.addActionListener(  
new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        modifyDimensions();  
    }  
});
```

```
/**  
 * Refreshes the game board in the graphical user interface with the specified dimensions.  
 *  
 * @param newHeight The new height of the game board.  
 * @param newWidth The new width of the game board.  
 */  
private void restartDimensions(int newHeight, int newWidth) {  
    heightBoard = newHeight;  
    widthBoard = newWidth;  
    Container contentPane = getContentPane();  
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.CENTER));  
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.SOUTH));  
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.WEST));  
    contentPane.remove(((BorderLayout) contentPane.getLayout()).getLayoutComponent(BorderLayout.EAST));  
    JPanel buttonPanel = new JPanel();  
    buttonPanel.setLayout(new GridLayout(newHeight, newWidth));  
    board = new JButton[newHeight][newWidth];  
    for (int i = 0; i < newHeight; i++) {  
        for (int j = 0; j < newWidth; j++) {  
            Color darkCoffeeColor = new Color(rgb:0x3E2723);  
            board[i][j] = new JButton();  
            board[i][j].setBackground(darkCoffeeColor);  
            buttonPanel.add(board[i][j]);  
        }  
    }  
    Color lightBlueColor = new Color(r:173, g:216, b:230);  
    add(buttonPanel, BorderLayout.CENTER);  
    addButton(text:"Información del jugador", lightBlueColor, BorderLayout.SOUTH, infoPlayerListener);  
    addButton(text:"# de jugadas", lightBlueColor, BorderLayout.WEST, numOfPlaysListener);  
    addButton(text:"Color", lightBlueColor, BorderLayout.EAST, colorListener);  
    revalidate();  
    repaint();  
}
```



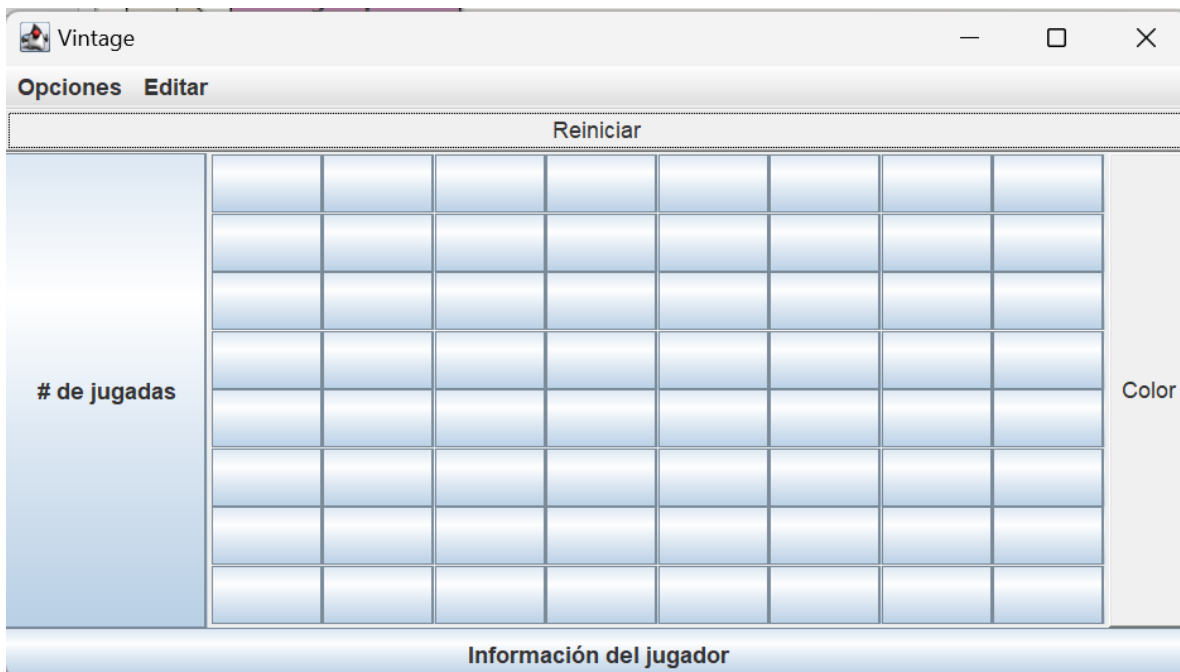
```

/**
 * Displays a dialog to modify the dimensions of the game board matrix.
 * Prompts the user to input new height and width values, then updates the game board accordingly.
 */
private void modifyDimensions() {
    JPanel inputPanel = new JPanel(new GridLayout(rows:2, cols:2));
    JTextField heightField = new JTextField();
    JTextField widthField = new JTextField();
    inputPanel.add(new JLabel(text:"Alto:"));
    inputPanel.add(heightField);
    inputPanel.add(new JLabel(text:"Ancho:"));
    inputPanel.add(widthField);
    int result = JOptionPane.showConfirmDialog(this, inputPanel, title:"Modificar Dimensiones", JOptionPane.OK_CANCEL_OPTION);
    if (result == JOptionPane.OK_OPTION) {
        try {
            int newHeight = Integer.parseInt(heightField.getText());
            int newWidth = Integer.parseInt(widthField.getText());
            if (newHeight <= 20 && newWidth <= 20) {
                heightBoard = newHeight;
                widthBoard = newWidth;
                restart();
            } else {
                JOptionPane.showMessageDialog(this, message:"Las dimensiones deben ser menos o iguales a 20.", title:"Error", JOptionPane.ERROR_MESSAGE);
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, message:"Por favor, ingrese números válidos para alto y ancho.", title:"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}

```

3. Ejecuten el caso de uso y capture las pantallas más significativas.

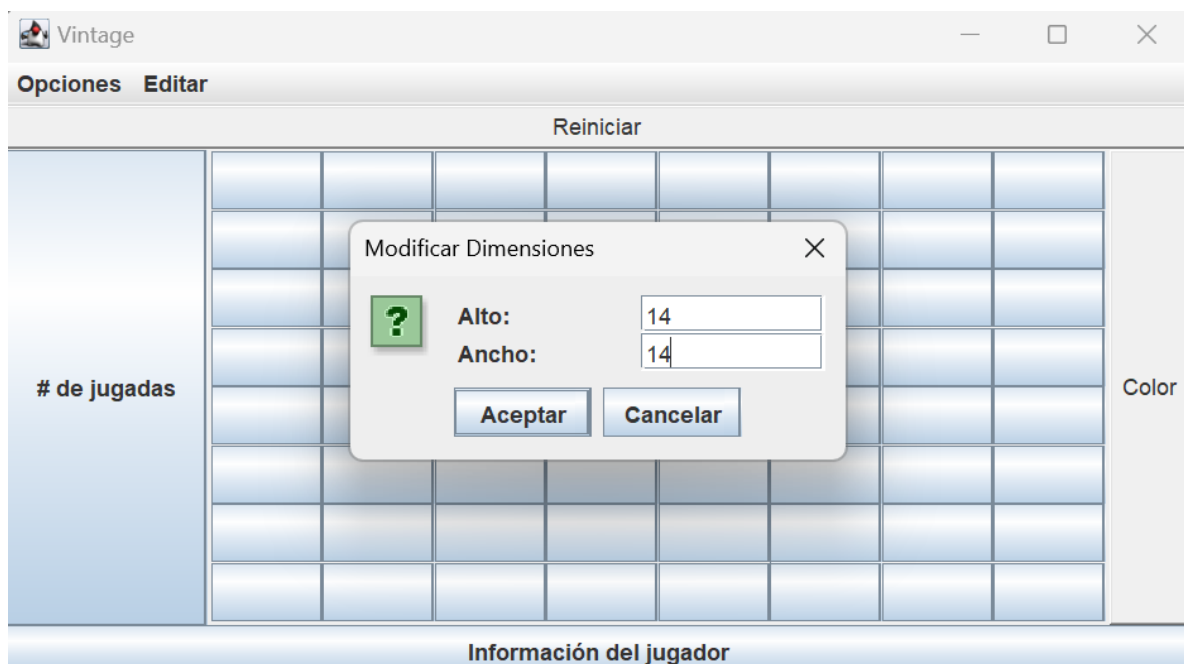
Inicialmente viene predeterminada por las casillas 8 X 8 y sin ningún color inicial.



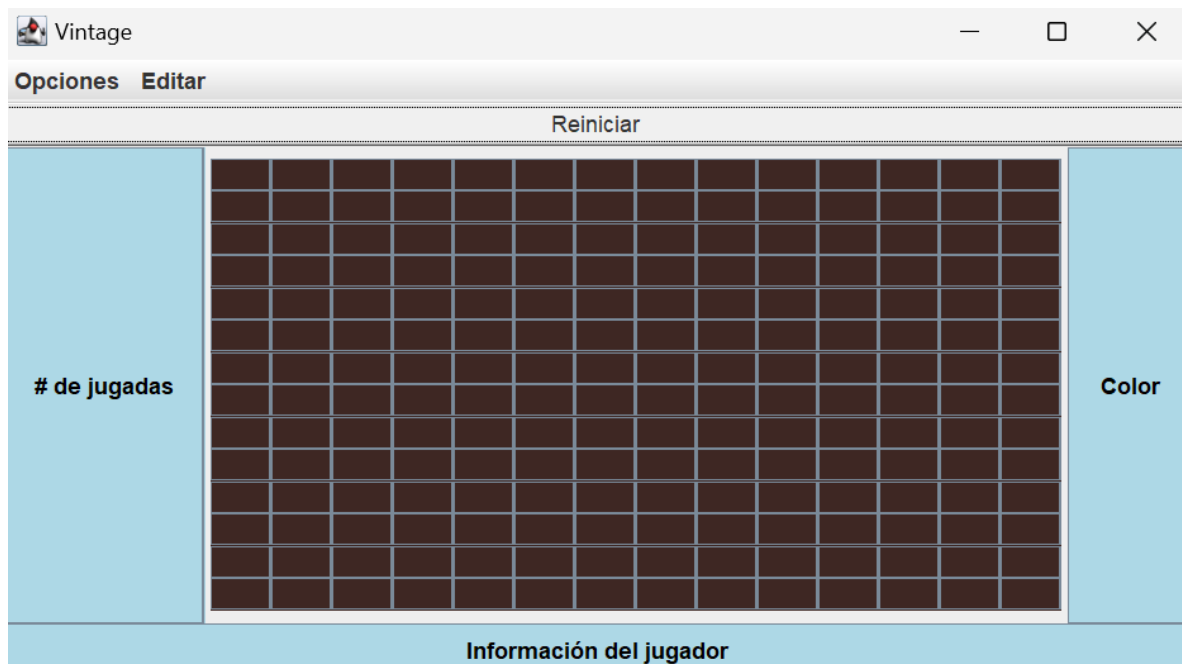
Cuando le demos en Editar – Modificar dimensiones va salir un cuadro de dialogo para cambiar el tamaño de las dimensiones de nuestra matriz



Le asignamos valores correctos en los campos necesarios.



Y se reiniciara con un nuevo tablero de las dimensiones que se deseen.



RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes?

(Horas/Nombre)

Acevedo-> 15 horas

Rodríguez-> 15 horas

2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

Está en buen estado ya que los requerimientos se cumplieron, con respecto a lo que se pedía.

3. Considerando la práctica XP del laboratorio ¿por qué consideran que es importante?

Las pruebas de aceptación fueron de gran importancia para este laboratorio, ya que todo se verificaba a partir de un caso de uso, ósea tener un paso a paso a la hora de ejecutar lo que se pedía y como se desarrollaba con respecto a los requerimientos.

4. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?

El mayor logro fue lograr aprender como se realiza una interfaz desde cero con todo lo básico para que funcione una aplicación, en este caso con un juego, lo más complicado fue lograr conectar la capa de presentación con la capa de dominio, ya que hay varias cosas que no son fáciles de identificar, para saber si realmente son físicas o lógicas, aquí es importante manejar bocetos bases para que sea más sencillo a la hora de implementar las cosas.

5. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Trabajamos a la par para el desarrollo del laboratorio, haciendo lo que se nos pedía de forma ordenada y con un diseño que teníamos previo.