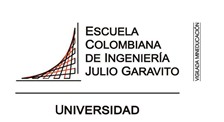
****

**Escuela Colombiana de Ingeniería Julio Garavito**

**Programación Orientada a Objetos 2024-2**

**Laboratorio V**

**Cristian Santiago Pedraza Rodríguez**

**Andersson David Sánchez Méndez**

**8 de noviembre de 2024**

# PROGRAMACIÓN ORIENTADA A OBJETOS INTERFAZ

**2024-2**

# Laboratorio 5/6

## OBJETIVOS

1. Desarrollar una miniaplicación gráfica considerando el patrón MVC.
2. Implementar el esquema de manejo de eventos con clases anónimas
3. Experimentar el comportamiento de las ventanas JFrame, JDialog y JOptionPane
4. Seleccionar los lienzos más apropiados para un diseño: JPanel, JScrollPane, JTabbedPane
5. Revisar las posibilidades de los estilos:FlowLayout, BorderLayout y GridLayout
6. Apropiar algunos componentes básicos: JLabel, JTextField, JButton, JMenuBar,
7. Apropiar algunos componentes especiales: JFileChooser y JColorChooser
8. Vivenciar las prácticas XP: [Acceptance tests](http://www.extremeprogramming.org/rules/functionaltests.html) *are run often and the score is published*

*When* [a bug is found](http://www.extremeprogramming.org/rules/bugs.html) *tests are create*

# ENTREGA

1. Incluyan en un archivo .zip los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.
2. En el espacio de entrega deben indicar el estado de avance de su laboratorio y los problemas pendientes por resolver.
3. Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada, en los espacios preparados para tal fin.

# CONTEXTO

El objetivo es implementar el juego **Clustering**

El trabajo se debe hacer desde **CONSOLA**

|  |  |
| --- | --- |
| El propuesto por ustedes ClusteringGUI | El acordado en laboratorio Clustering |
| **Vista - Controlador** | **Modelo** |

**Para la capa de presentación NO deben hacer pruebas de unidad ni diagramas de secuencia**

**ESTE LAB COMO TODO ES DESDE CONSOLA, SE EJECUTA LA PEQUEÑA APLICACIÓN CON MAVEN, LAS PRUEBAS TAMBIÉN (UNA CLASE CON TODAS LAS PRUEBAS(6)).**

**El comando para ejecutar y compilar con Maven es mvn clean compile exec:java estando en el directorio cd lab5, cd Clustering.**

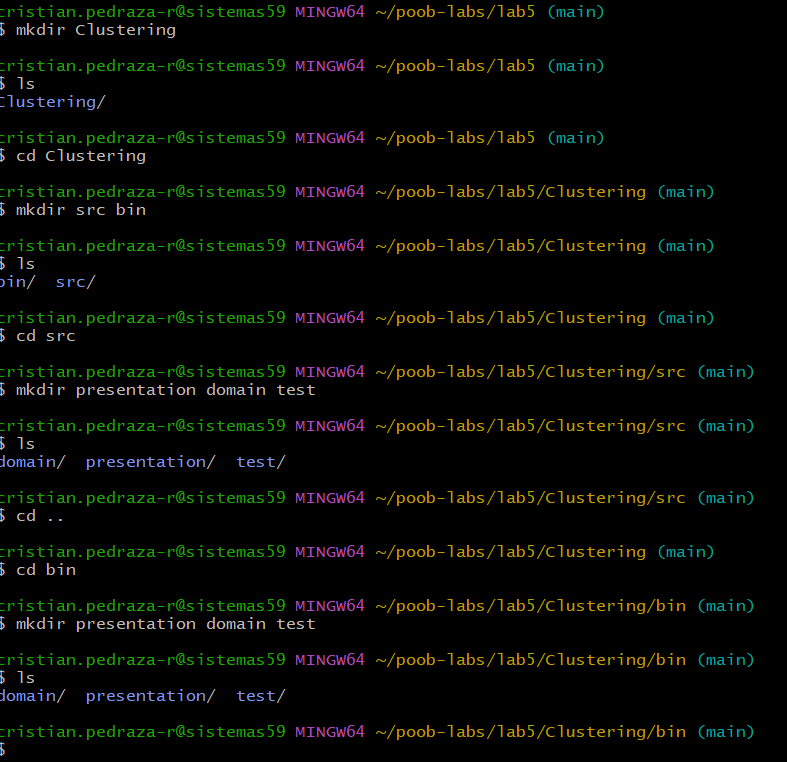
**Y las pruebas(1) con el comando mvn test, marcando todo en verde si no hay fallos o errores.**

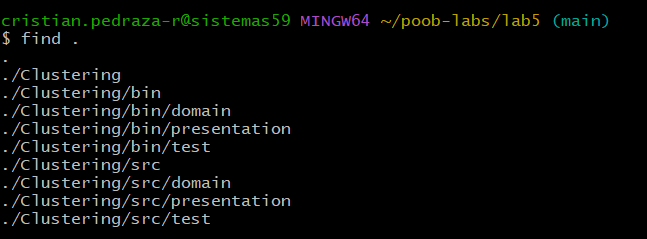
**DESARROLLO**

## Directorios

El objetivo de este punto es construir un primer esquema para el juego **Clustering**

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

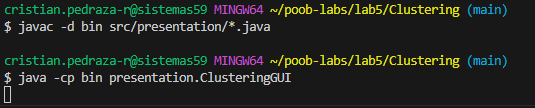


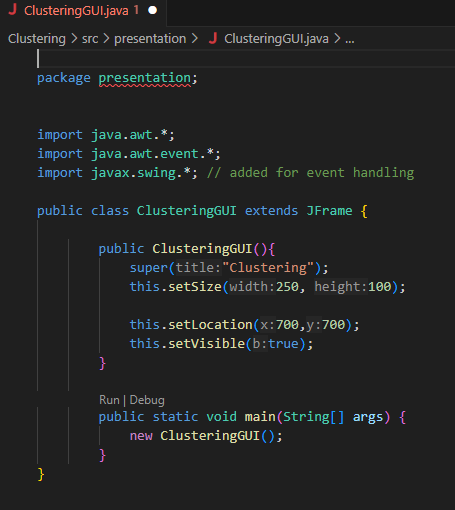


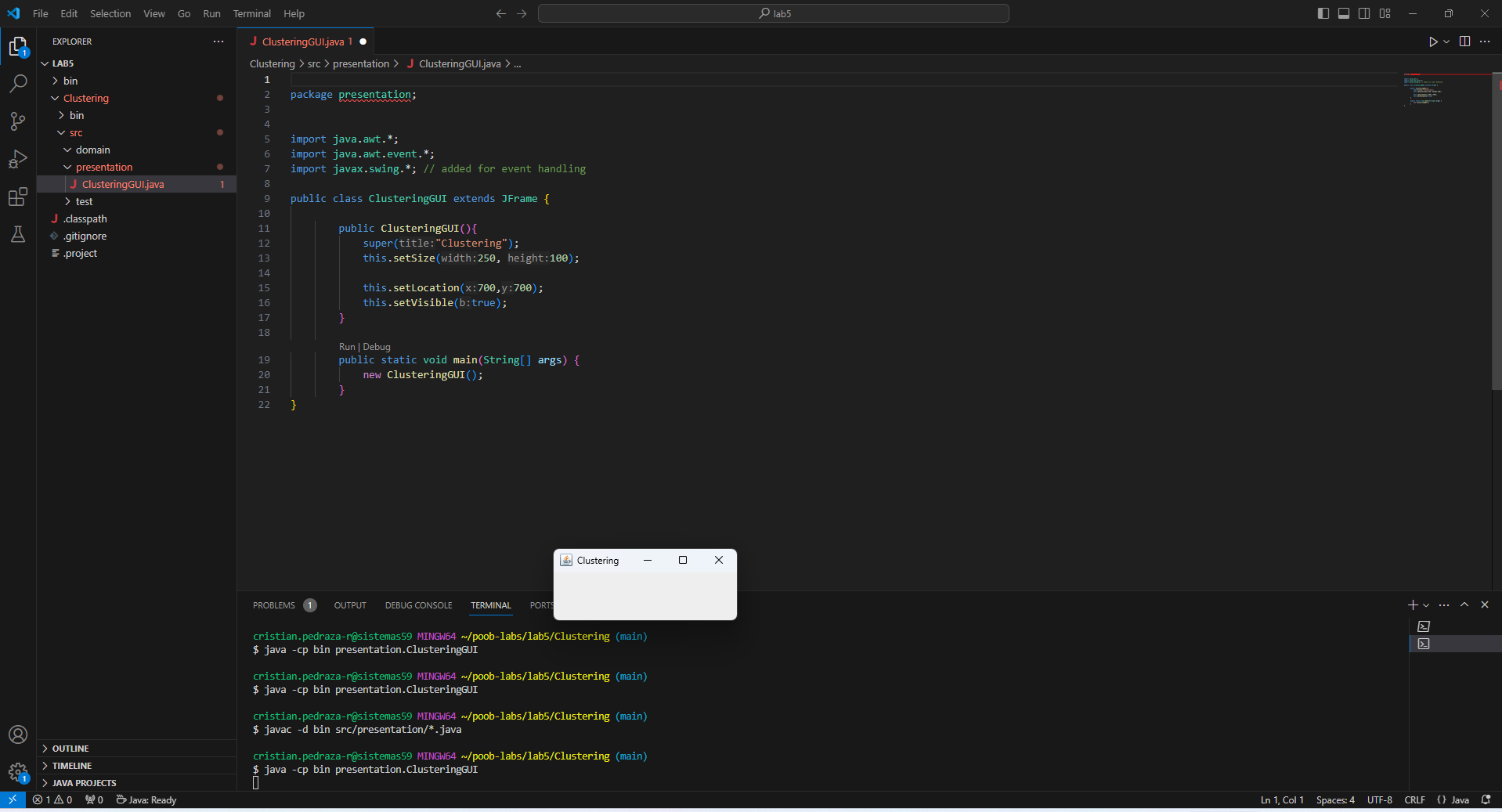
## Ciclo 0: Ventana vacía – Salir [En \*.java y lab05.doc]

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

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

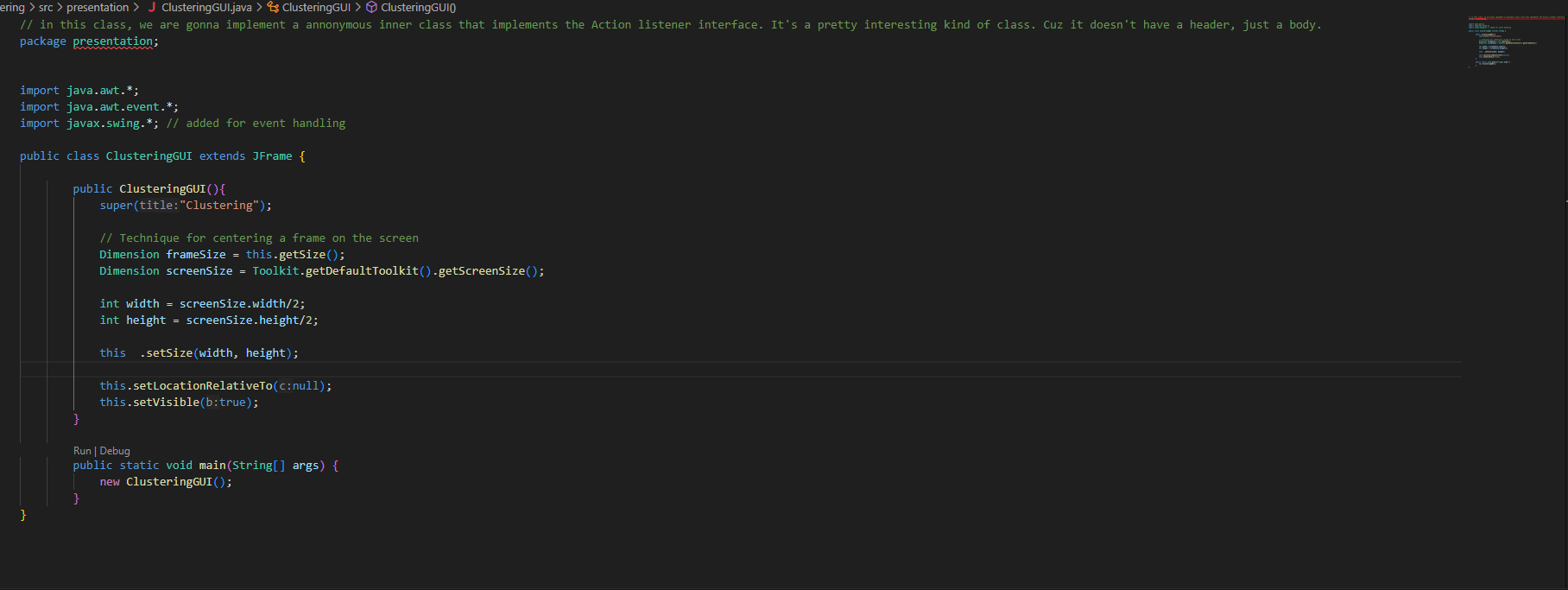


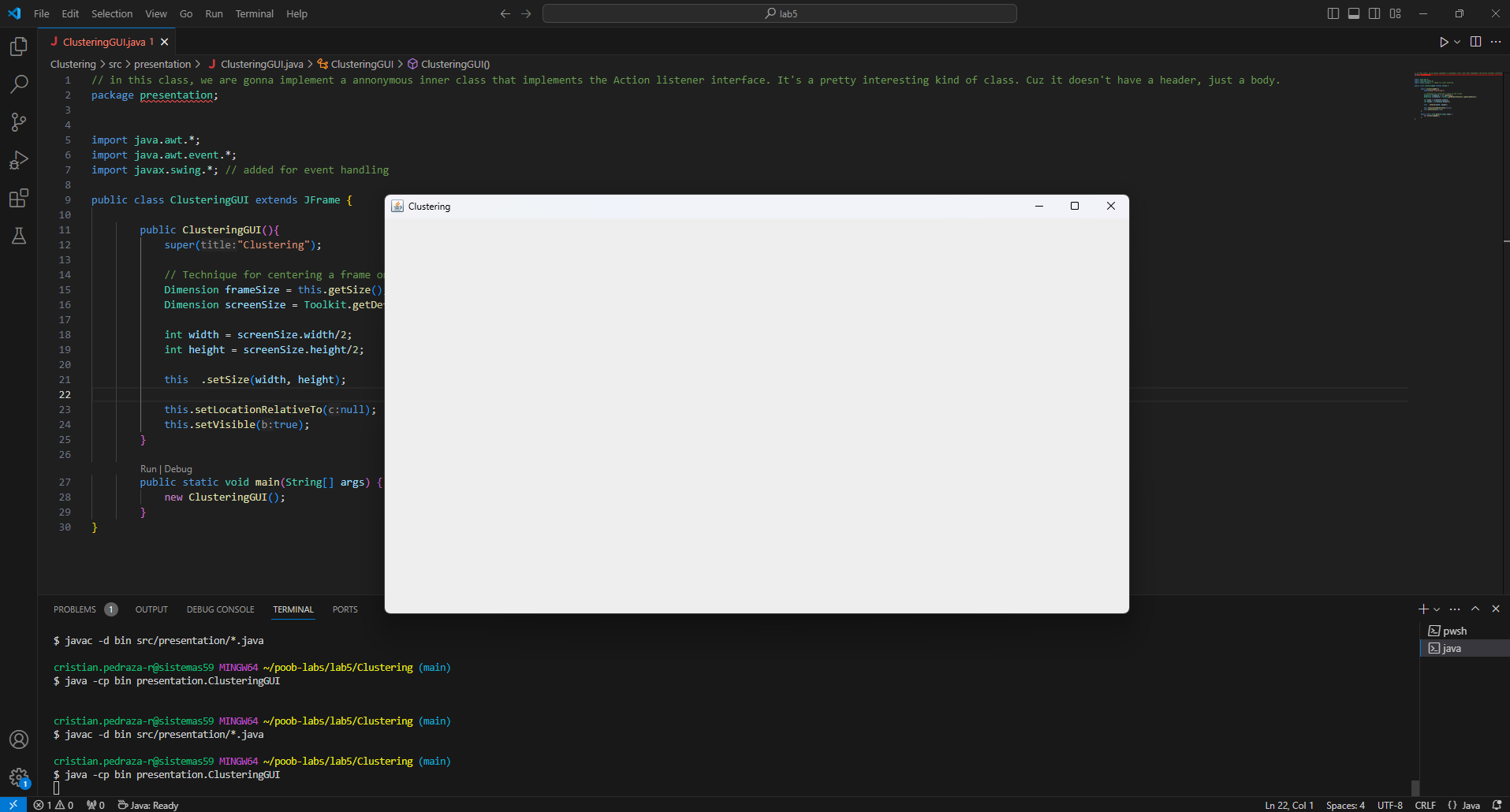




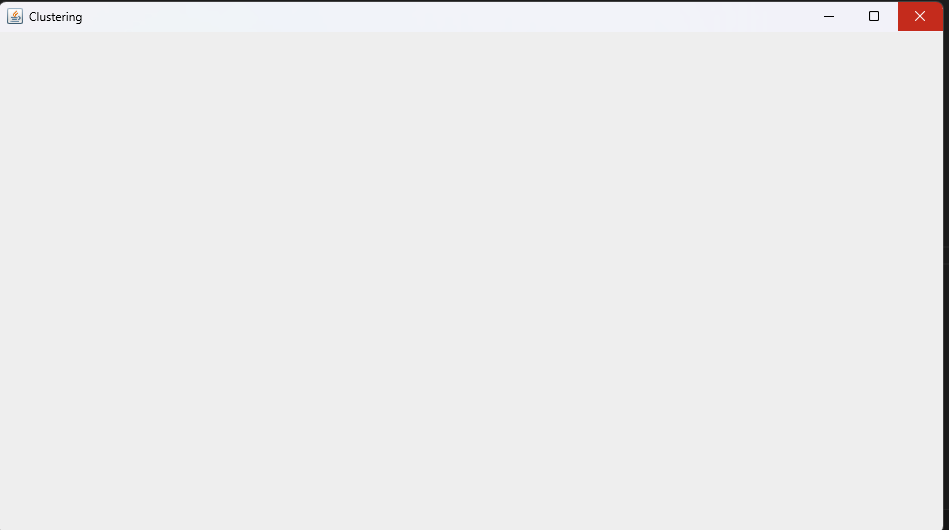
(Si la ventana principal no es la inicial en su diseño, después deberán mover el main al

componente visual correspondiente)

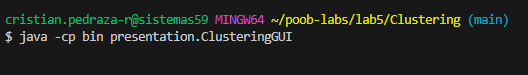
1. Modifiquen el tamaño de la ventana para que ocupe un cuarto de la pantalla y ubíquenla en el centro. Para eso inicien la codificación del método prepareElements. Capturen esa pantalla.
2. 



1. Traten de cerrar la ventana. ¿Termina la ejecución? ¿Qué deben hacer en consola para terminar la ejecución?



Al cerrar la ventana, no se finaliza la ejecución, es necesario ejecutar control + c en la consola.

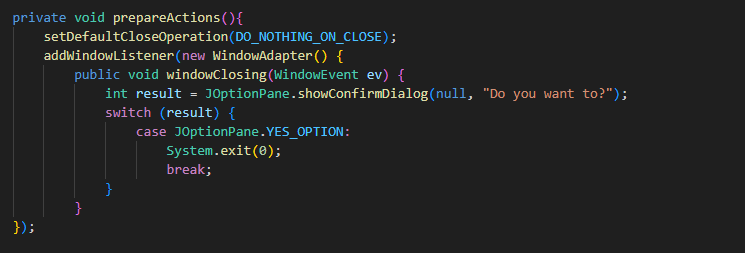


1. Estudien en JFrame el método setDefaultCloseOperation. ¿Para qué sirve? ¿Cómo lo usarían si queremos confirmar el cierre de la aplicación? ¿Cómo lo usarían si queremos simplemente cerrar la aplicación?

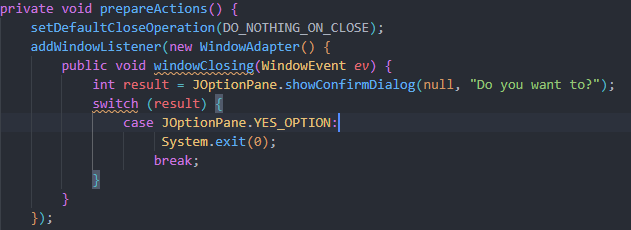
El método de la clase JFrame setDefaultCloseOperation nos permite definir el comportamiento que quiero que tenga mi programa una vez se dé la instrucción de cierre del programa.

Para nuestra aplicación, lo utilizaríamos como evento que al dispararse, se lo mandamos como argumento EXIT\_ON\_CLOSE para cerrar la aplicación.

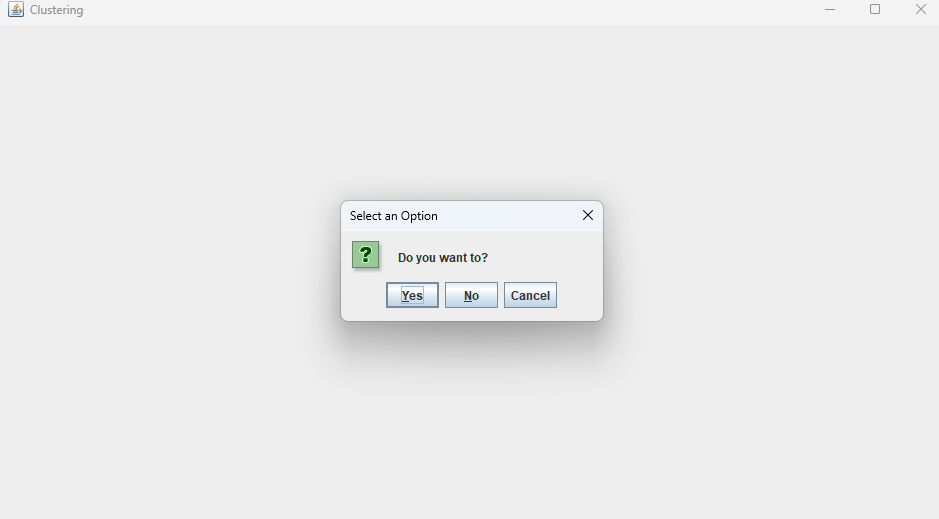
Al principio del método prepareActions se pondría:



1. 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.



Mensaje de confirmación



Si la respuesta es Yes, se cierra el programa, si es No o Cancel no se cierra y se quedará en la interfaz gráfica normal.

## 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. a) **Expliquen los componentes visuales necesarios para este menú.**

Para implementar el menú se requieren de componentes visuales como:

* **JMenuBar:** Se almacena la barra de menú principal
* **JMenu:** Cada menú en esa barra predefinida anteriormente.
* **JMenuItem:** Se dan las opciones: Nuevo, Abrir – Salvar y Salir.

Y mediante los separadores se da el orden adecuado a cada opción.

b) **¿Cuáles serían los atributos?**

De la misma en que se definieron los componentes visuales, así también se crearon los atributos en código.

JMenuBar menuBar,

JMenu menuFile,

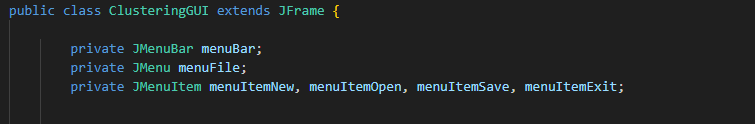
JMenuItem menuItemNew, menuItemOpen, menuItemSave, menuItemExit

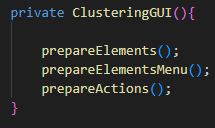
c) **¿Cuáles podrían ser variables del método prepareElements?**

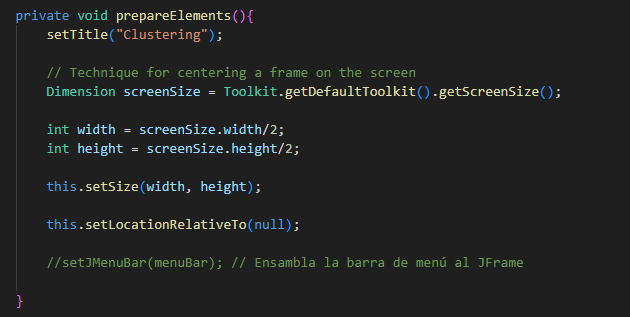
Lo que va cambiando en prepareElements es la instanciación de los objetos como menú (JMenu,JMenuItem,JMenuBar) que se desarrollan en el método prepareElementsMenu.

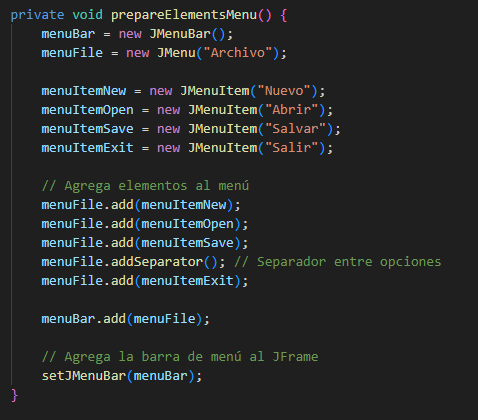
1. Construya la forma del menú propuesto (prepareElements - prepareElementsMenu).

Ejecuten. Capturen la pantalla.

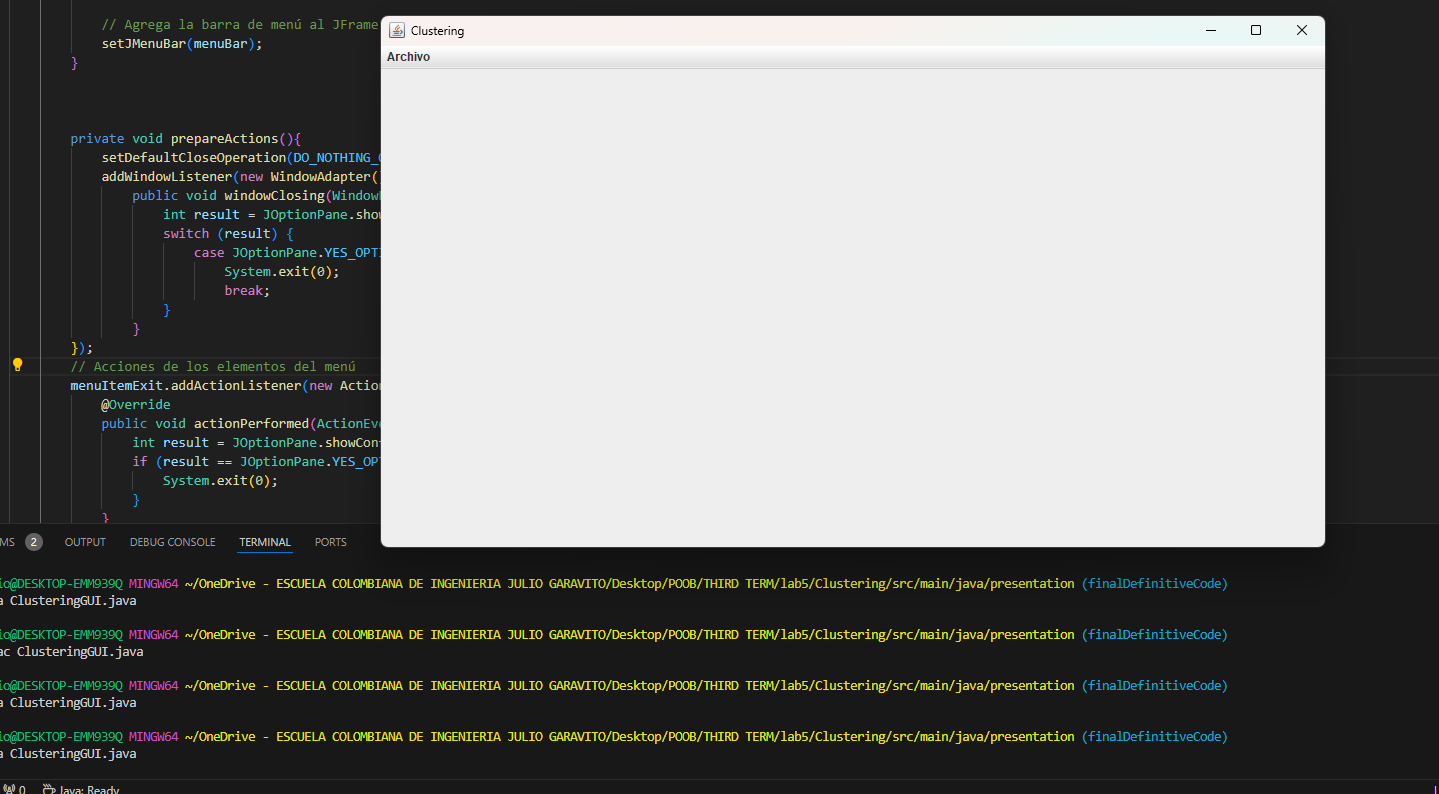


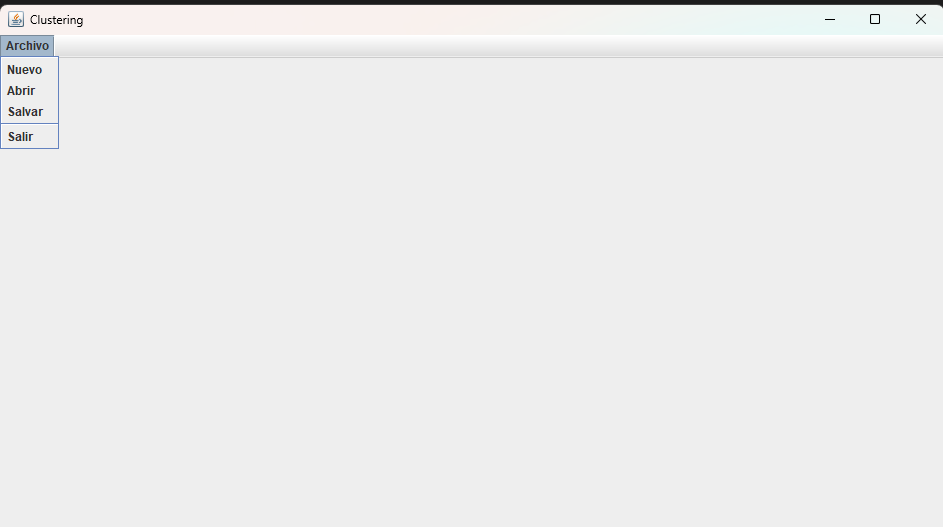




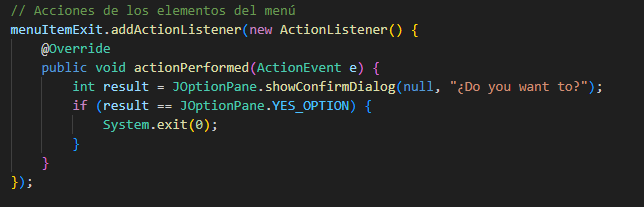


Al ejecutar con git bash, se ve esto:



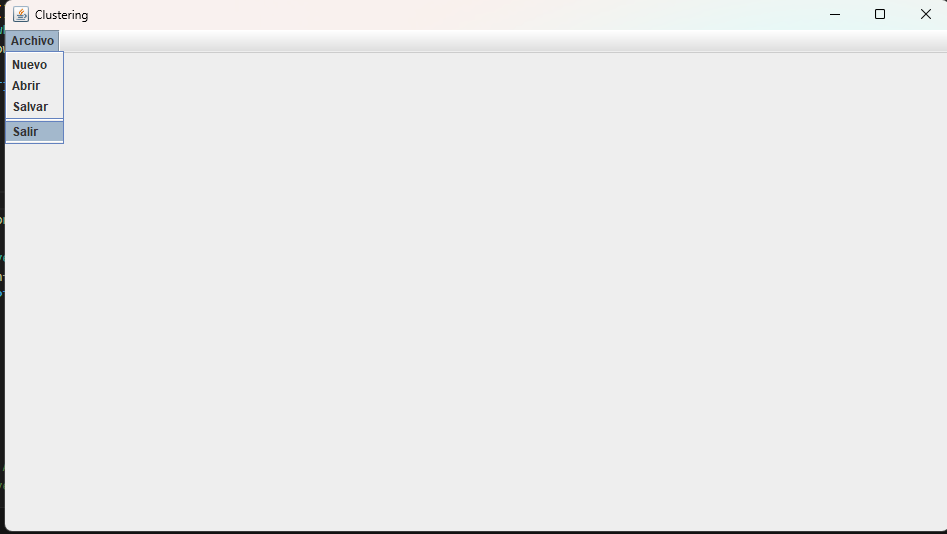


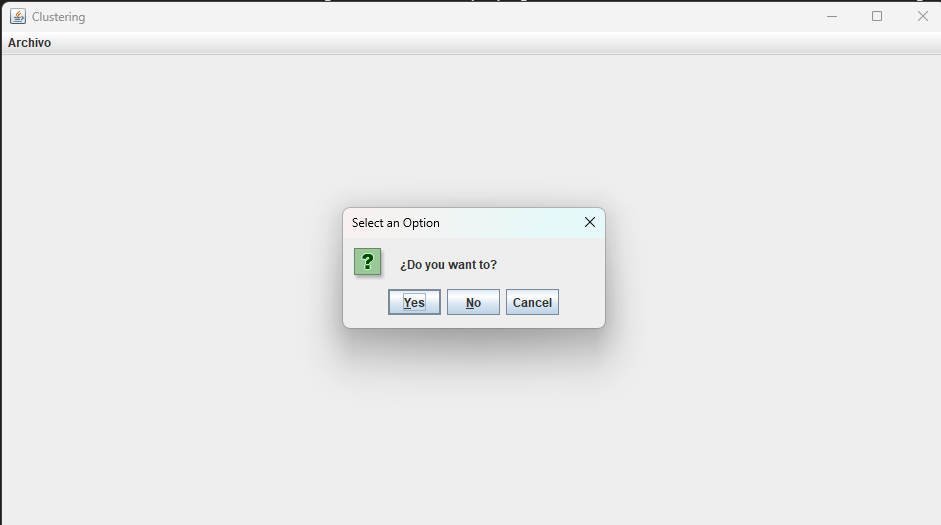
1. Preparen el “oyente” correspondiente al icono cerrar con confirmación (prepareActions - prepareActionsMenu). Ejecuten el programa y salgan del programa. Capturen las pantallas.



Entonces lo que ocurre cuando se dispara este evento es que dependiendo si la lista de opciones de menú que se puso es igual a Salir, entonces que genere el mensaje de confirmación todo tal cual partiendo del Window Listener.

Se ve así cuando se oprime Salir:





## Ciclo 2: Salvar y abrir [En \*.java y lab05.doc]

El objetivo es preparar la interfaz para las funciones de persistencia

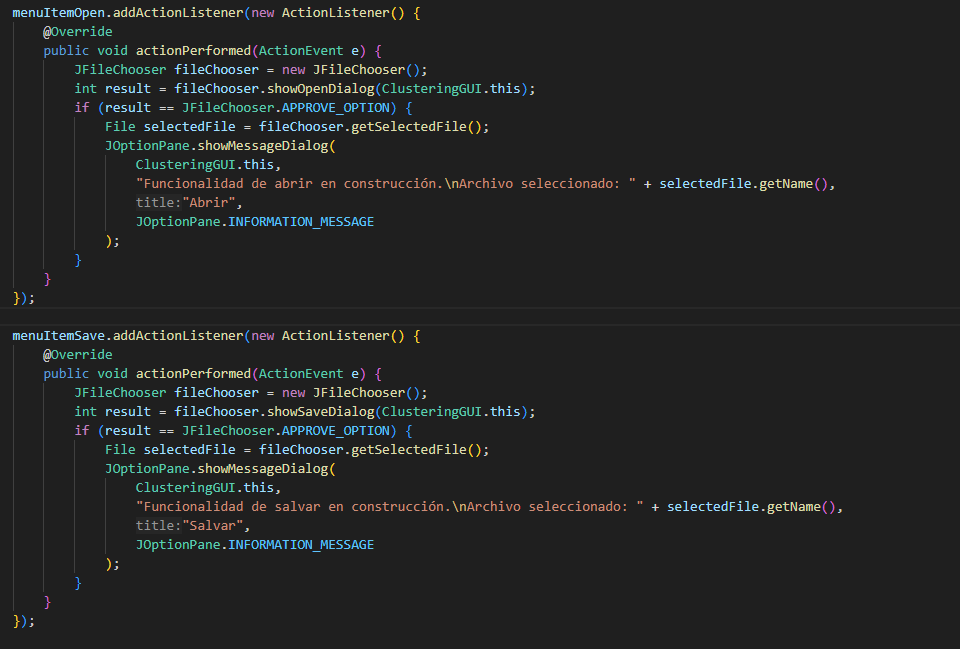
1. Detalle el componente JFileChooser especialmente los métodos: JFileChooser, showOpenDialog, showSaveDialog, getSelectedFile.

**JFileChooser:** definido como el constructor que se encarga de crear diálogos para selección de archivos. Los métodos que usa son:

* **showOpenDialog**: componente padre que muestra un cuadro de diálogo para que el usuario seleccione el archivo que desea abrir.
* **showSaveDialog**: componente padre que muestra un cuadro de diálogo para que el usuario seleccione una ubicación de memoria para guardar el archivo.
* **getSelectedFile()**: devuelve el archivo que uno seleccionó en el cuadro de diálogo.

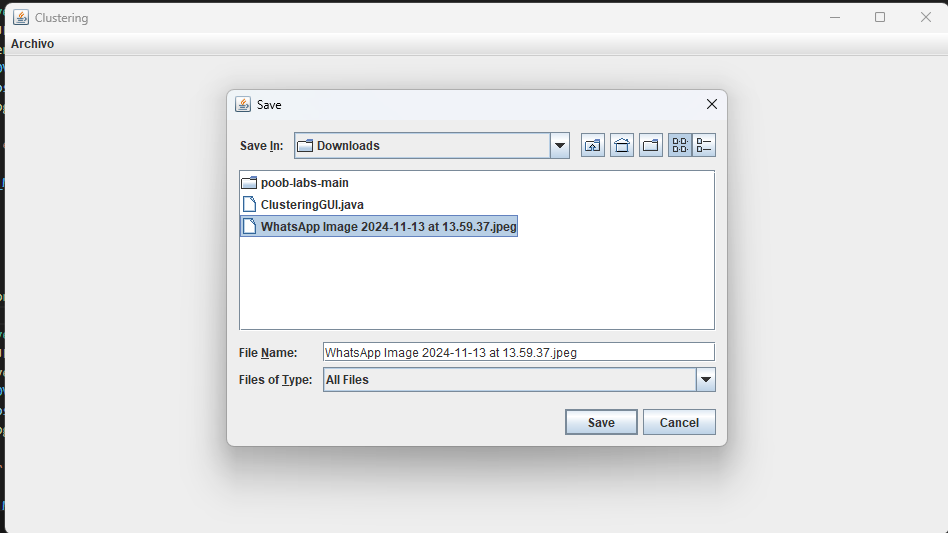
1. Implementen parcialmente los elementos necesarios para salvar y abrir. Al seleccionar

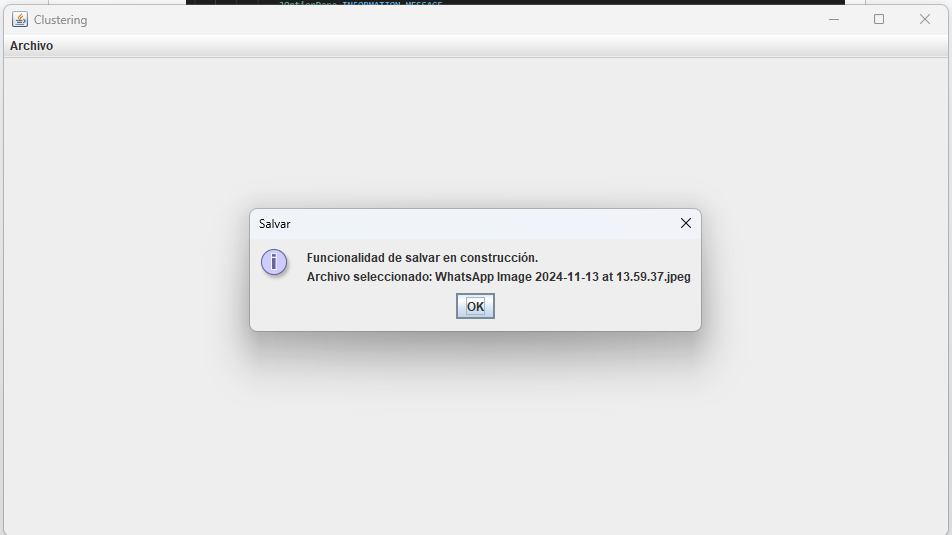
los archivos, indique que las funcionalidades están en construcción detallando la acción y el nombre del archivo seleccionado.



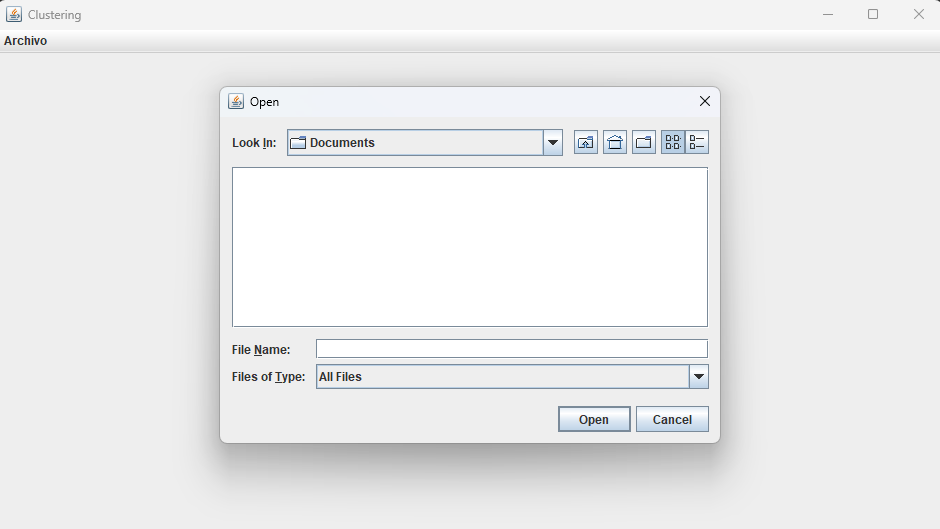
1. Ejecuten las dos opciones y capturen las pantallas más significativas.

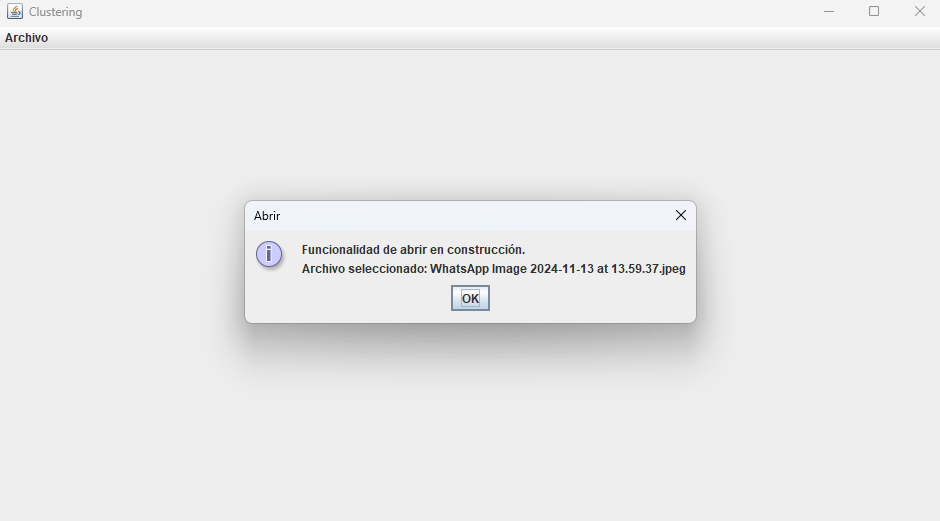
**Para salvar:**

****

****

**Para abrir:**

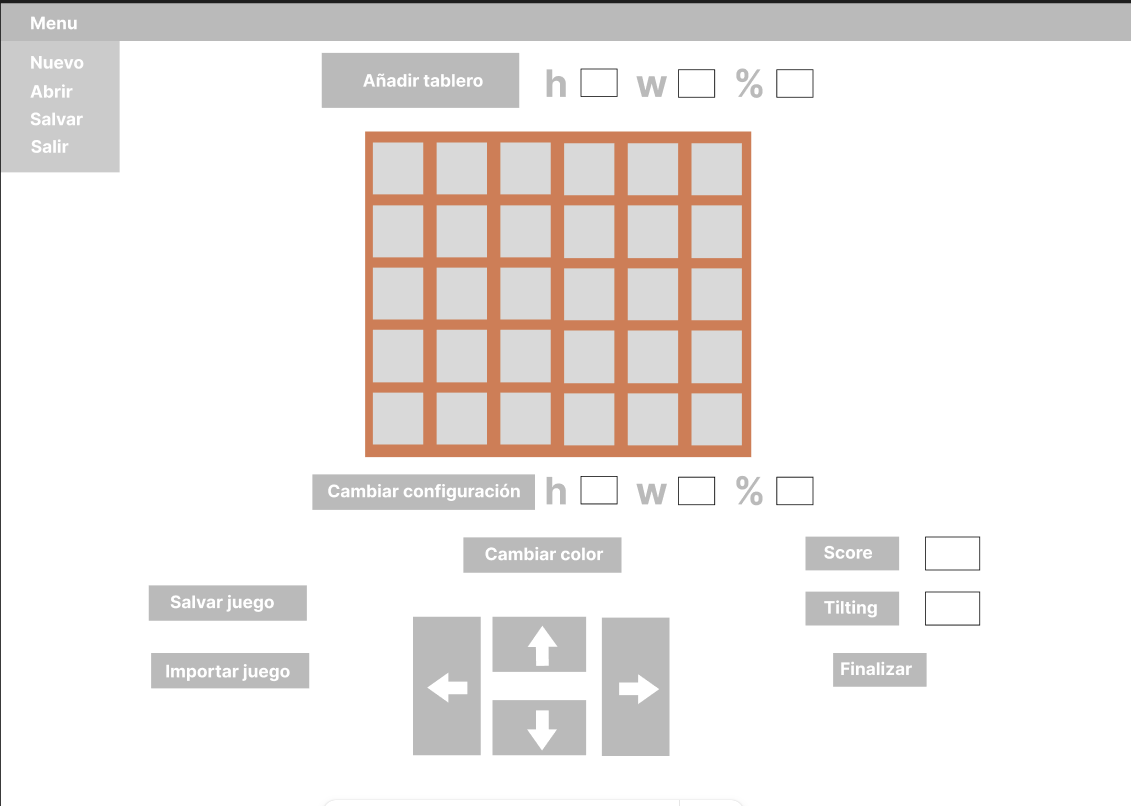
****

****

## 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)

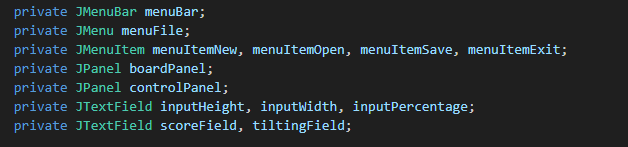
1. Presenten el bosquejo del diseño de interfaz con todos los componentes necesarios.



1. 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.

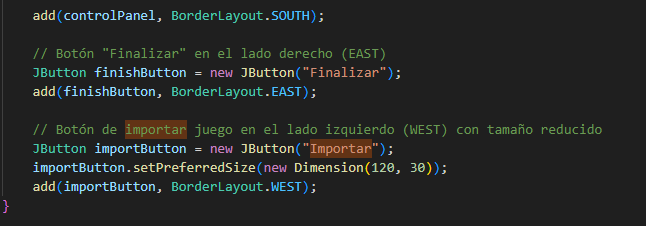
**Atributos necesarios**

****

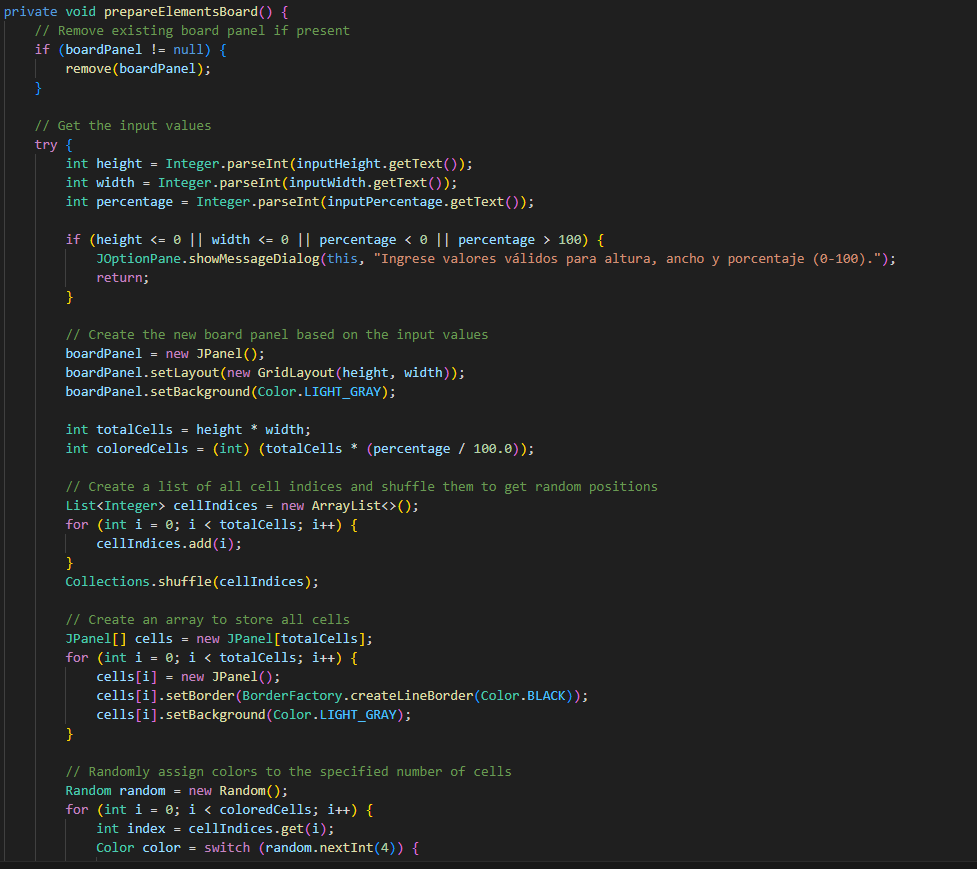
**prepareElements()**

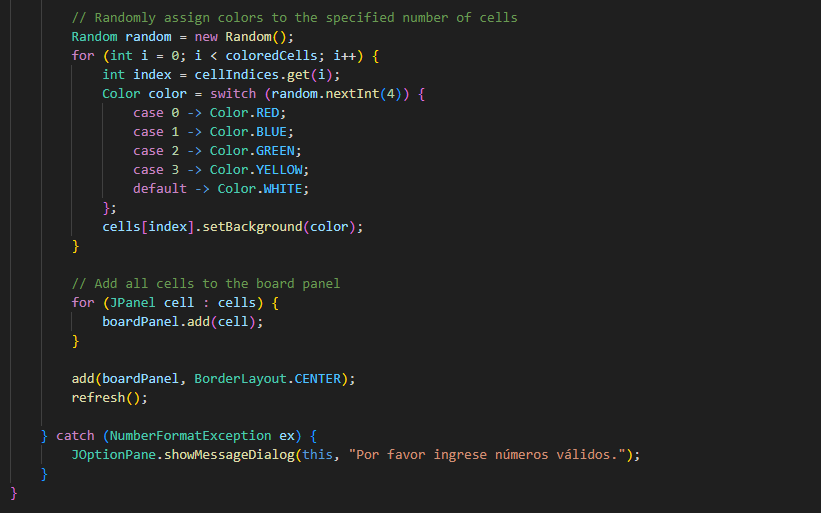




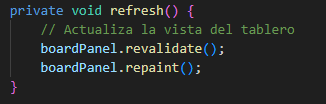


**prepareElementsBoard()**

****

****

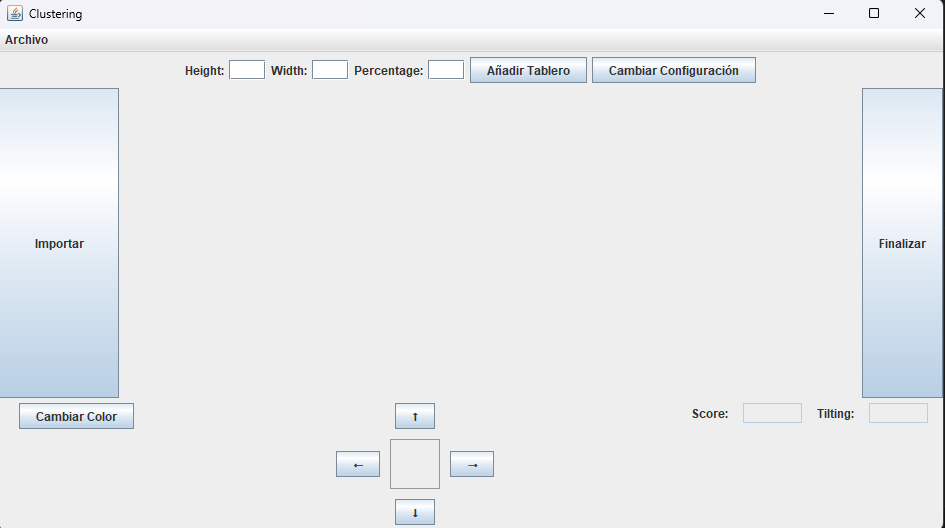
**refresh()**

****

**Después de implementar todo esto, se da una representación gráfica del bosquejo parecido hecho inicialmente.**

1. Ejecuten y capturen esta pantalla.

Antes de añadir h,w, porcentaje



Después de agregar



## Ciclo 4: Cambiar colores [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.

**Para la vista:**

Esta se compone del botón Cambiar Color ya implementado en la interfaz o el bosquejo diseñado, permitiendo al usuario elegir qué color desea modificar para todos los colores de las baldosas, usando JColorChooser.

**Para el controlador:**

Encargado de responder cuando el usuario haga clic en el botón Cambiar Color. Abrirá el cuadro showDialog que permitirá seleccionar un color.

Después de seleccionar el color, el controlador debe aplicar ese nuevo color a todas las baldosas que tengan el color previamente seleccionado (por ejemplo, todas las baldosas rojas se cambiarán al nuevo color seleccionado).

* 1. Detalle el comportamiento de JColorChooser especialmente el método estático

showDialog

**JColorChooser** es un componente de Swing que permite al usuario seleccionar colores de manera gráfica.

**Método Estático showDialog** muestra un cuadro de diálogo adaptable al usuario para seleccionar un color.

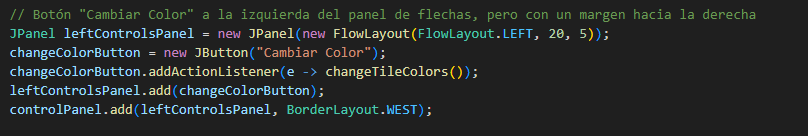
El método devuelve el color elegido por el usuario o null si el usuario cierra el cuadro de diálogo sin hacer una selección.

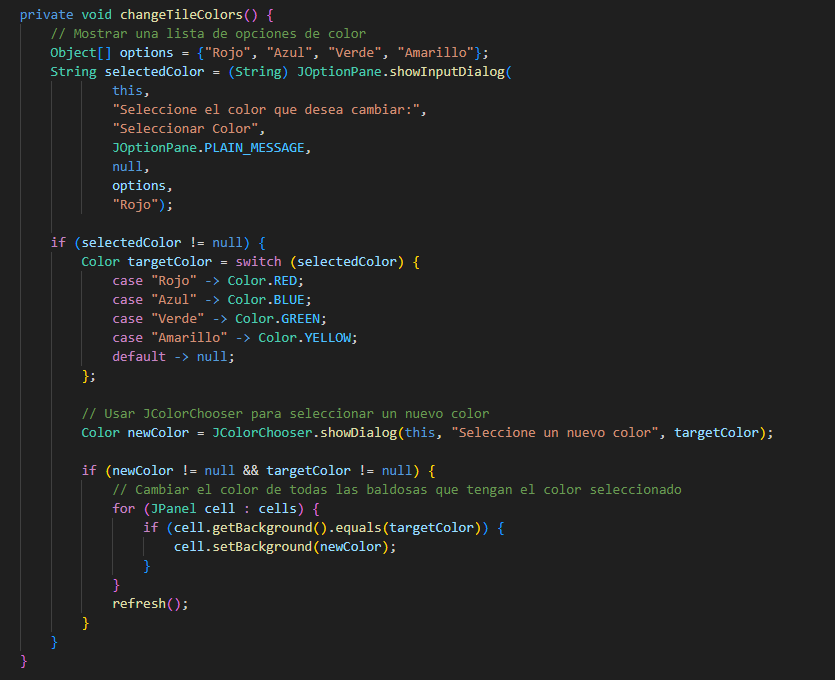
* 1. Implementen los componentes necesarios para cambiar el color de las fichas.

Con base al botón:

****

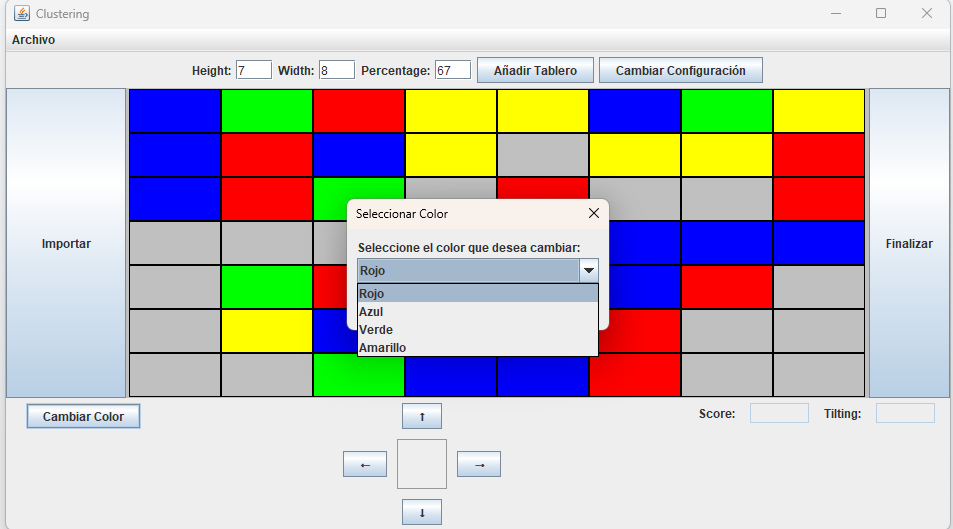
Entonces para implementar esto se necesitó de:



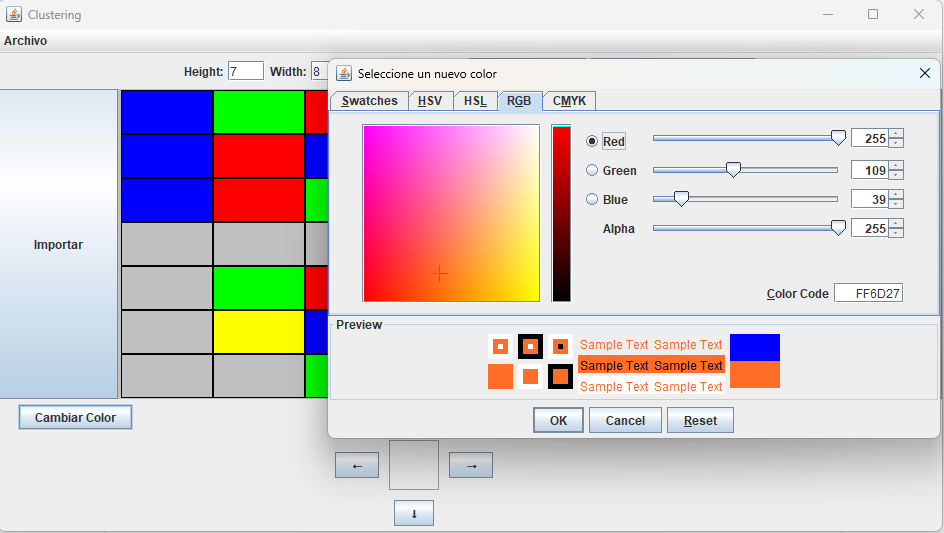


* 1. Ejecuten el caso de uso y capture las pantallas más significativas.

Opciones para mirar qué color de baldosas se quiere cambiar a otro color usando JChooseColor:



Entonces, por ejemplo, queremos cambiar todas las baldosas azules a un color escogido desde la API de swing.



Debería cambiar en el tablero todas las baldosas azules por ese color:



CONTINUACIÓN

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**V**

## Ciclo 5: Modelo Clustering [En \*.java y lab05.doc]

El objetivo es implementar la capa de dominio para **Clustering**

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

Los métodos básicos son:

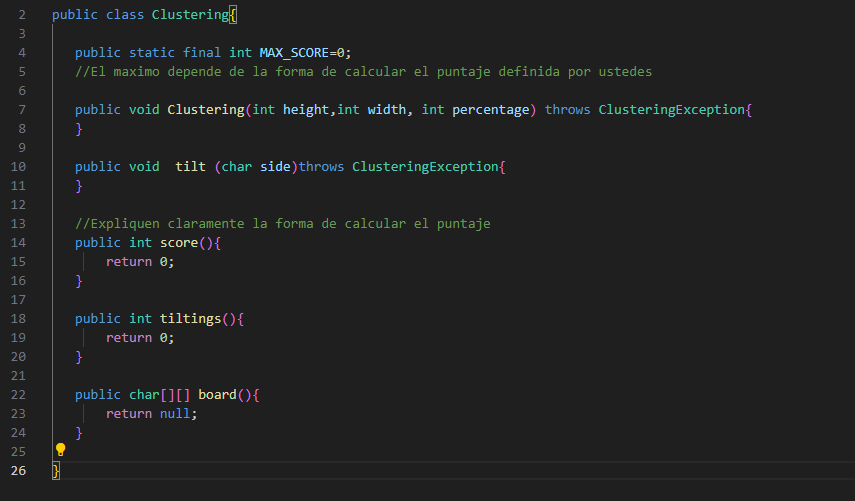
**tilt(char side) =** su función es mover el tablero hacia la dirección opuesta donde hizo clic, es decir, si hace tilt hacia arriba, entonces todas las baldosas deben moverse, pero solo 1 posición hacia abajo.

**score =** puntaje que mide el agrupamiento de las fichas. Este método solo funciona tanto para baldosas de colores definidos principalmente (r,gb,y), como para los nuevos colores elegidos con JChooseColor, es decir, si se cambia el color en las baldosas, y si de esas nuevas baldosas existen adyacentes a ese color, entonces también las cuente.

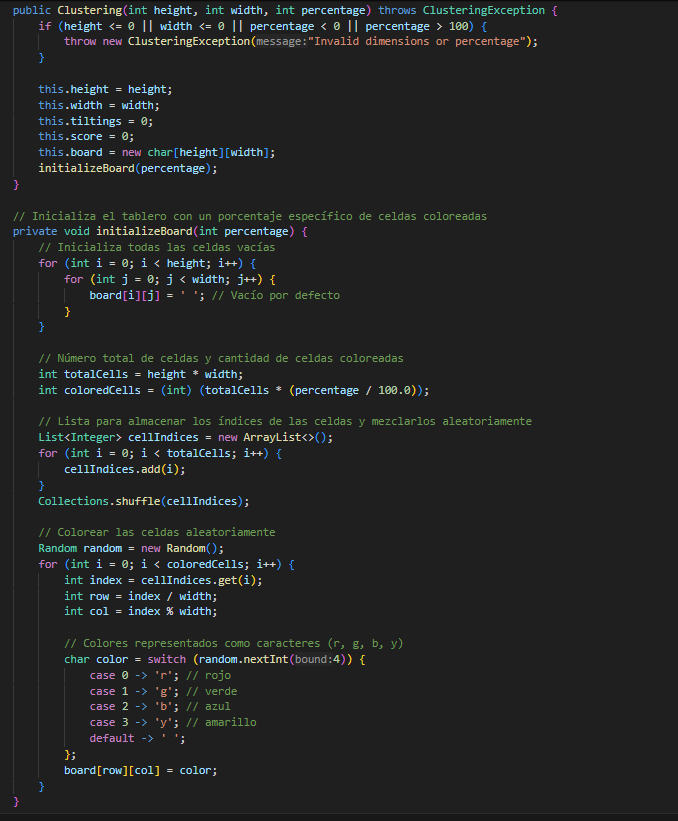
**tiltings =** informar el número de movimientos, es decir, cada vez que se hace tilt hacia cualquier dirección, se suma 1 a tiltings.

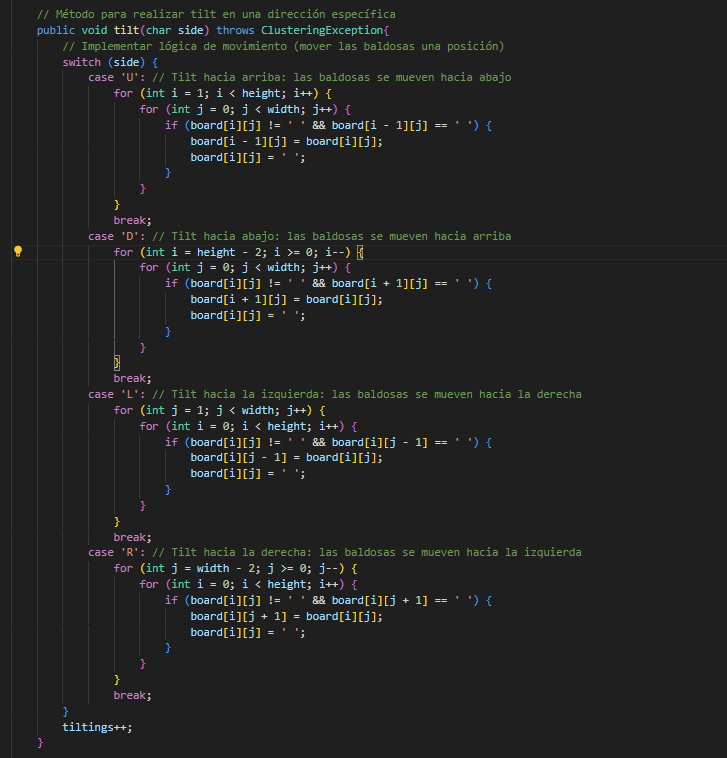
**board(char [][]) =** modificar la configuración del tablero con h,w, y porcentaje.

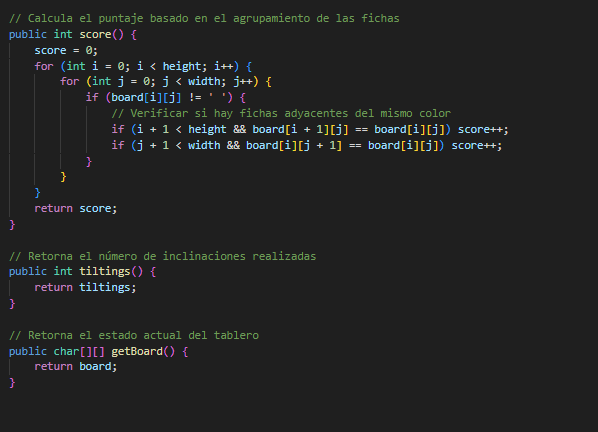
**Antes**

****

**Después**

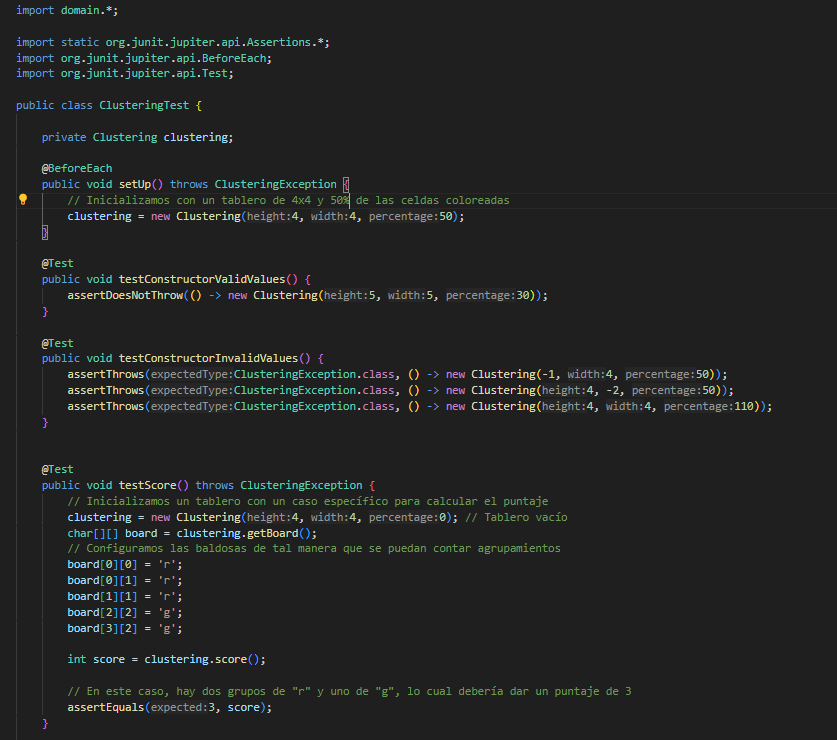
****

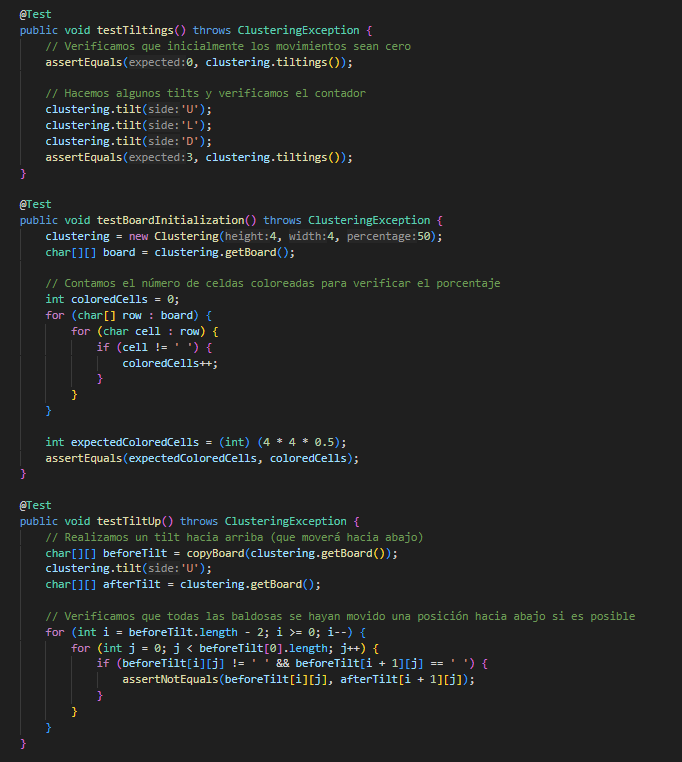
****

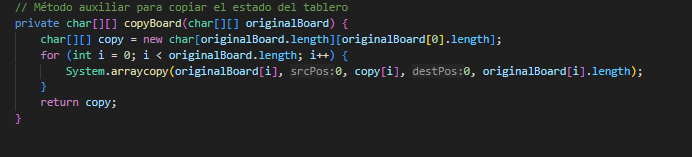
****

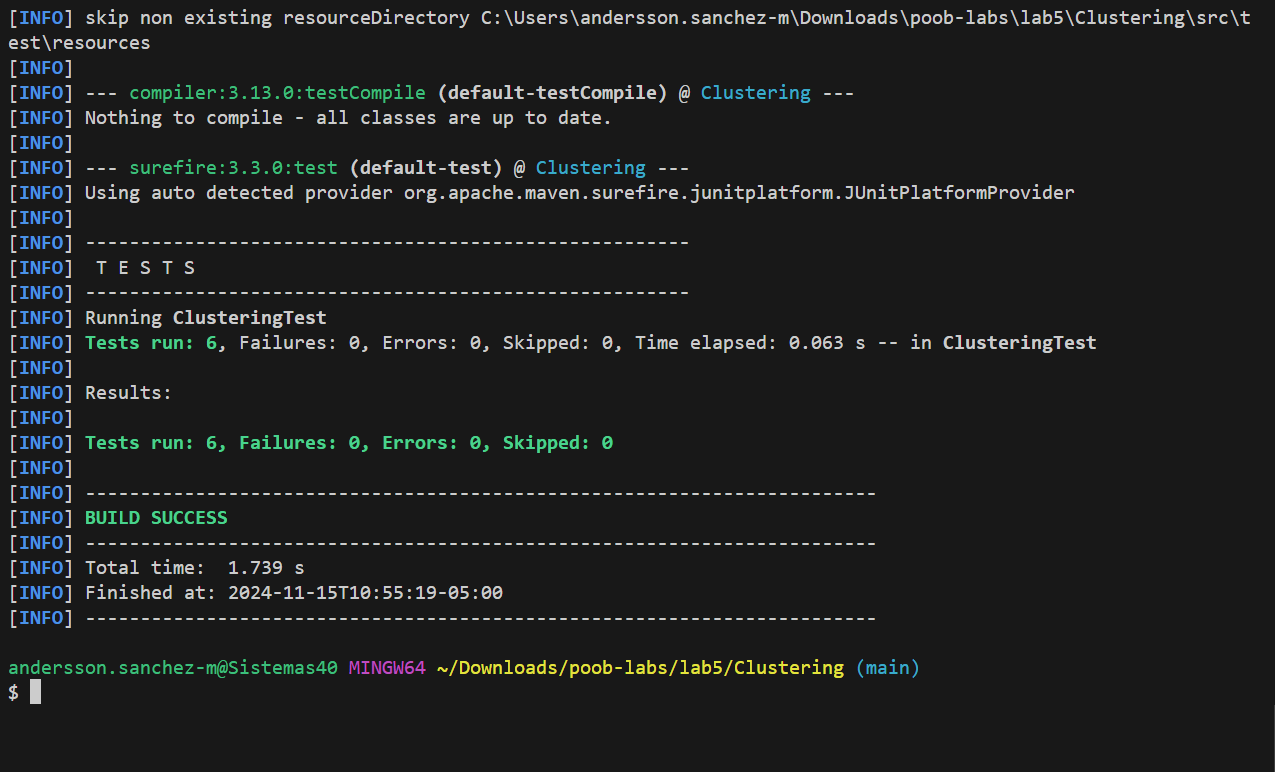
1. Ejecuten las pruebas y capturen el resultado.

Se probaron todos los métodos descritos anteriormente. En total se hicieron 6 pruebas describiendo el correcto funcionamiento de los métodos, todo se probó usando Maven.



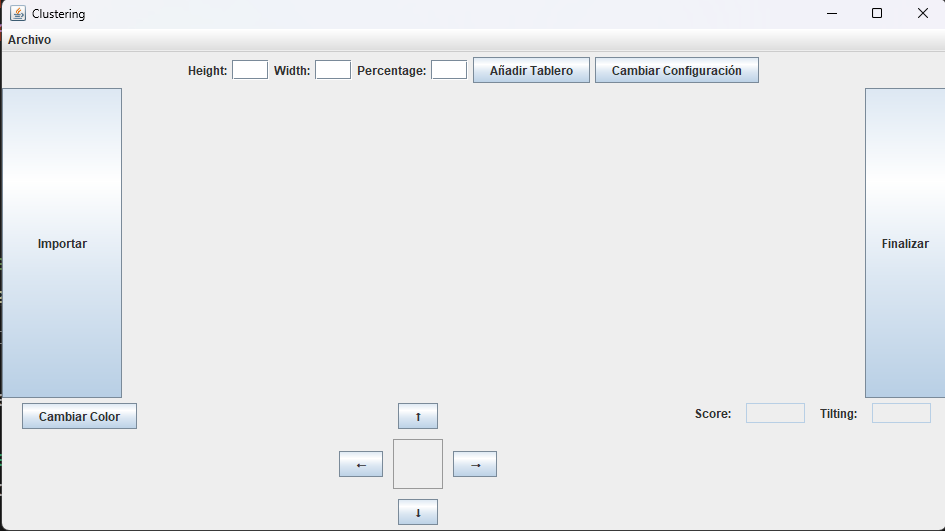






Tomando en cuenta un caso de uso para probar los métodos, entonces:

Partiendo de ejecutar y compilar la aplicación con el comando mvn clean compile exec:java, se muestra la interfaz

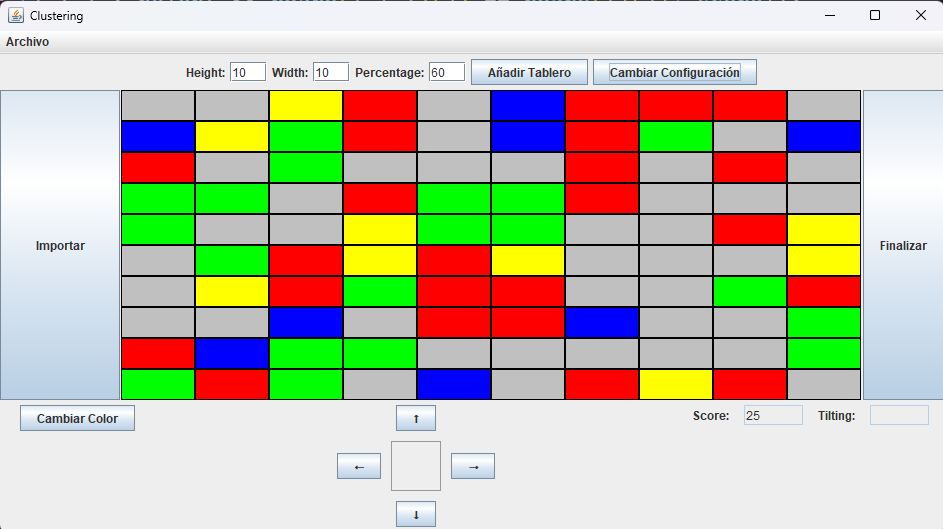


Entonces, si se pone un h,w,p con 6,7,50, respectivamente, se muestra el tablero (GridLayout) con esos valores



Se ve que con el porcentaje seleccionado, se ubican las baldosas con color aleatorio dependiendo del porcentaje.

Ahora, digamos que quiero cambiar la configuración inicial, entonces con el botón y con un nuevo h,w,percentage, se crea el nuevo tablero.



## Ahora se ve que el score marca 25 lo cual es correcto porque cumple el requerimiento de este método, que cuente el número de maneras en que puede haber baldosas adyacentes del mismo color.

Entonces ahora cada vez que se hace tilt, que en este caso es con las flechas, entonces, funciona correctamente, y tiltings debería dar 3 si hacemos 3 tilts.

1 tilt hacia arriba



2 tilt hacia la izquierda



3 tilt hacia la derecha



Entonces con este caso de uso se probaron todos los métodos a implementar.

## Ciclo 6: Jugar

**[En \*.java y lab05.doc]**

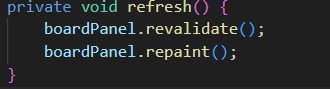
El objetivo es implementar el caso de uso jugar.

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



De por sí, jugar se implementó desde el principio haciendo interacciones con clics con los diferentes botones o el h,w, percentage para escribir los valores.

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



Este verifica si el h,w, y percentage son valores válidos; y si cumple con la condición, entonces revalida y repinta el boardPanel que es de tipo JPanel.

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

Jugar implica:

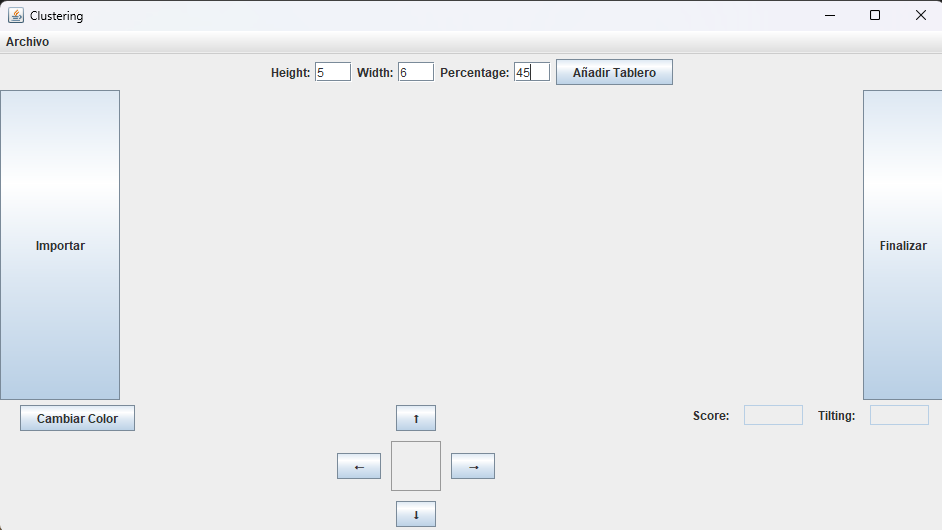
* Configurar los oyentes para los botones de control.
* Asegurar que cada método que se implementó en el ciclo 5 se muestre en la interfaz gráfica (ClusteringGUI) con sus respectivos botones.
* También se necesita la clase de excepción propia para manejar los posibles errores de la aplicación.

Partiendo del mockup inicial, en este ciclo el modo de interacción y lógica de algunos botones se cambiaron en menor medida.

**¿Qué cambió?**

Cuando se pone la primera configuración, que es, añadir tablero (h,w, percentage), esa opción debe desaparecer una vez se cree; después de esto solo aparece el botón de cambiar configuración cuando el usuario desee oprimir ese botón.

Así se ve:





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

¿Por qué?

**Oyentes para los botones de las flechas** (ActionListener): Cada flecha tiene su propio ActionListener que permite al jugador inclinar el tablero en la dirección opuesta una sola posición.

**Oyente para el botón Cambiar Color**: Un oyente es suficiente para manejar la lógica de selección de color y cambiar los colores de las baldosas específicas.

**Oyente para el botón Generar Tablero:** Este oyente maneja la creación del tablero según los parámetros especificados por el usuario.

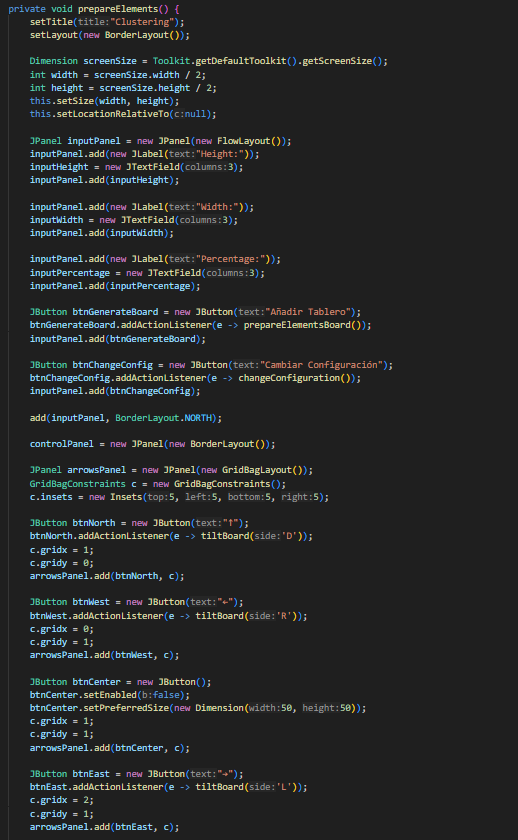
**Oyente para Cambiar Configuración:** Requiere un oyente para actualizar el tablero con nuevas configuraciones.

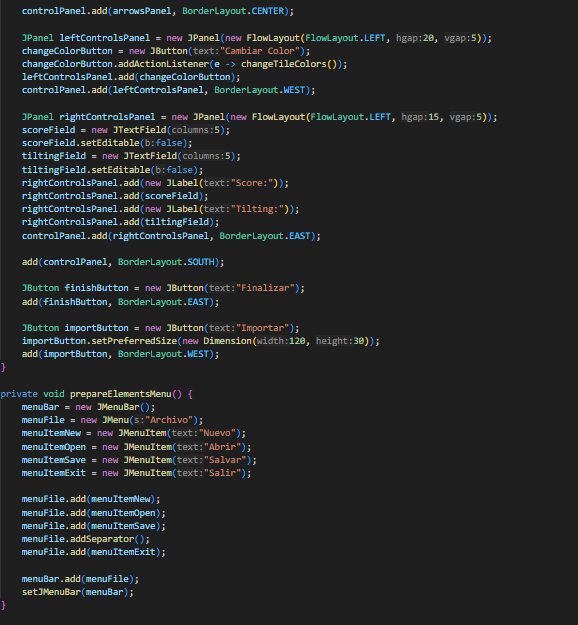
**Oyente de Ventana (WindowAdapter):** Este se utiliza para mostrar una confirmación antes de cerrar la ventana principal.

**Implementación**

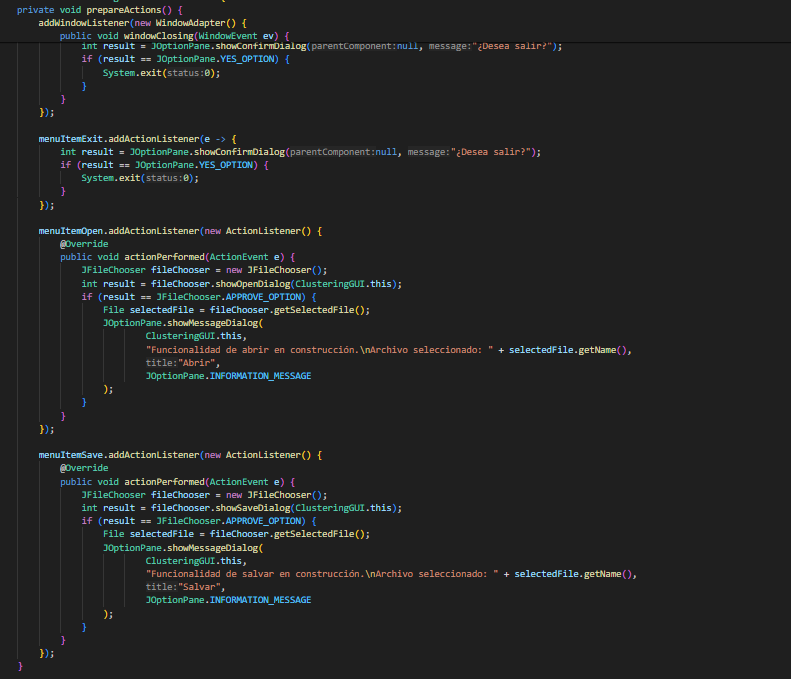
Partiendo desde la creación del atributo private Clustering clustering se hace toda la GUI en la capa de presentación, así, como los eventos que se disparan cuando se hace clic en el método que uno desee.

**En prepareElements():**

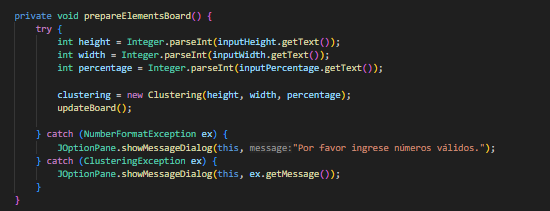




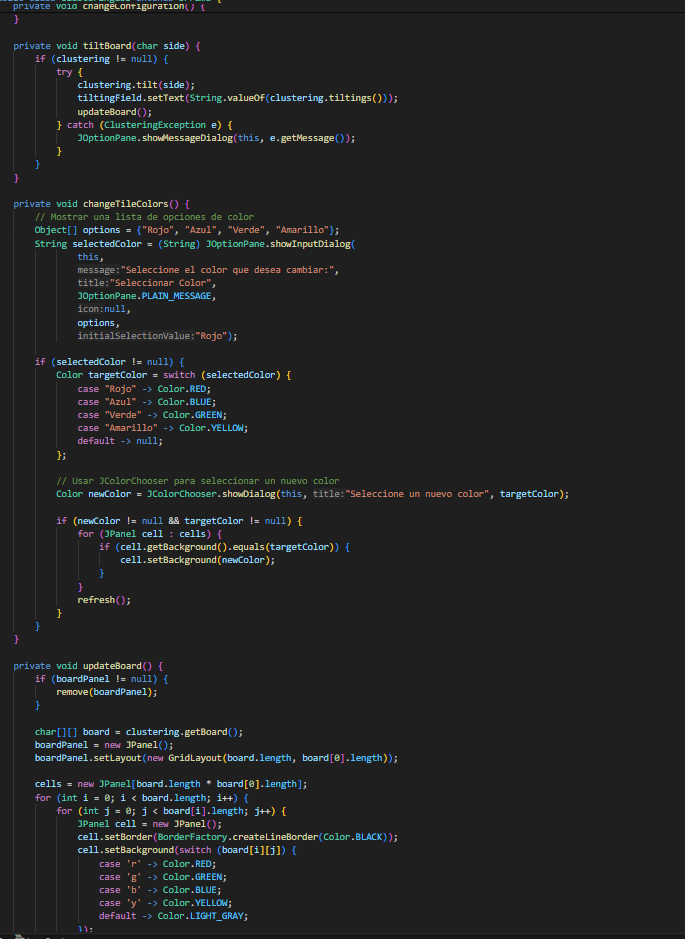
**En prepareActions()**

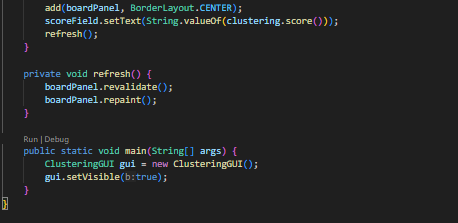
****

**En prepareElementsBoard**

****

**Todos los otros métodos para hacer posible la visualización de toda la aplicación.**

****

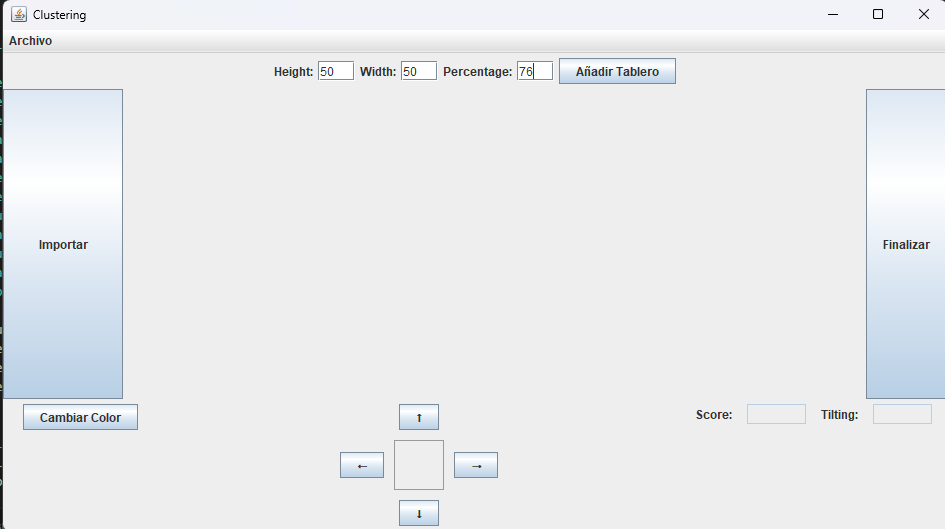
****

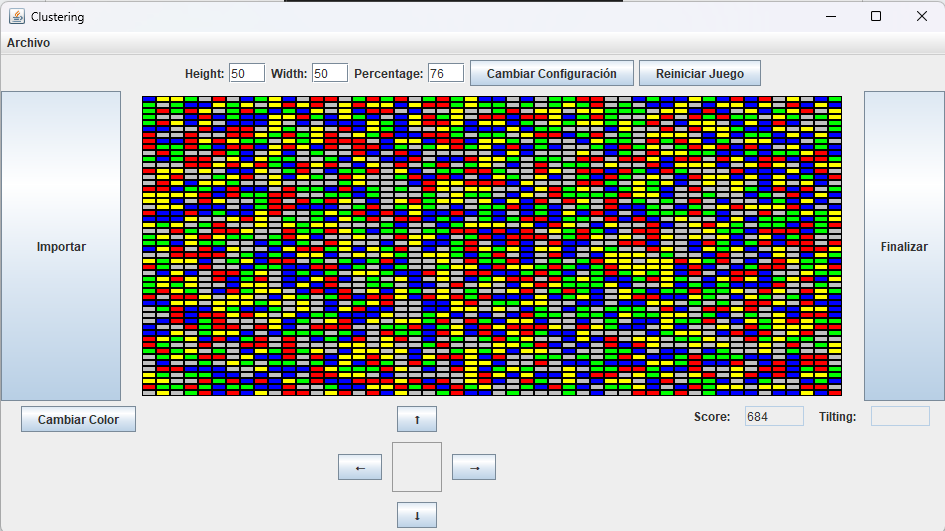
1. Ejecuten el caso de uso y capture las pantallas más significativas.

El caso de uso es igual al que se implementó arriba con todos los métodos, ahora la única diferencia es la novedad que se implementó.

Sin embargo, aquí se muestra otro caso:

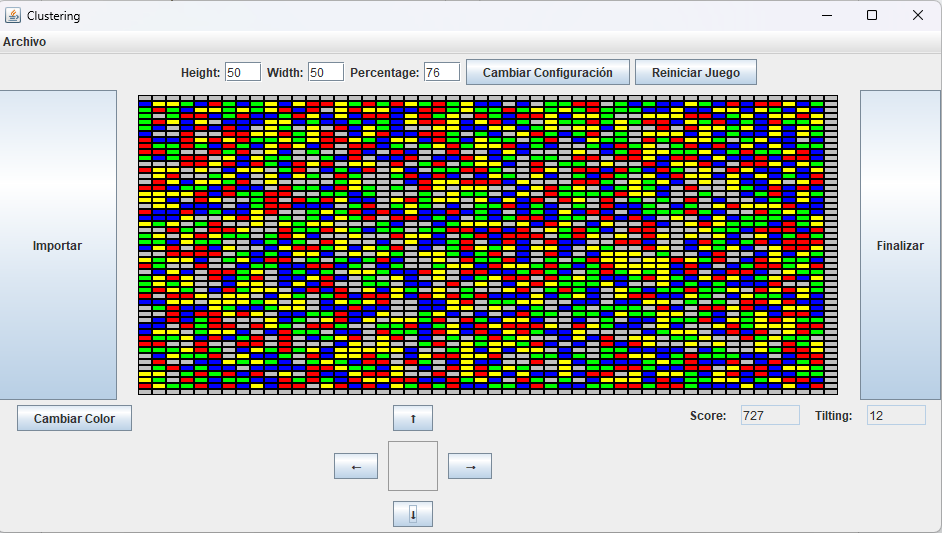
Tablero de 50 x 50 con porcentaje de 76



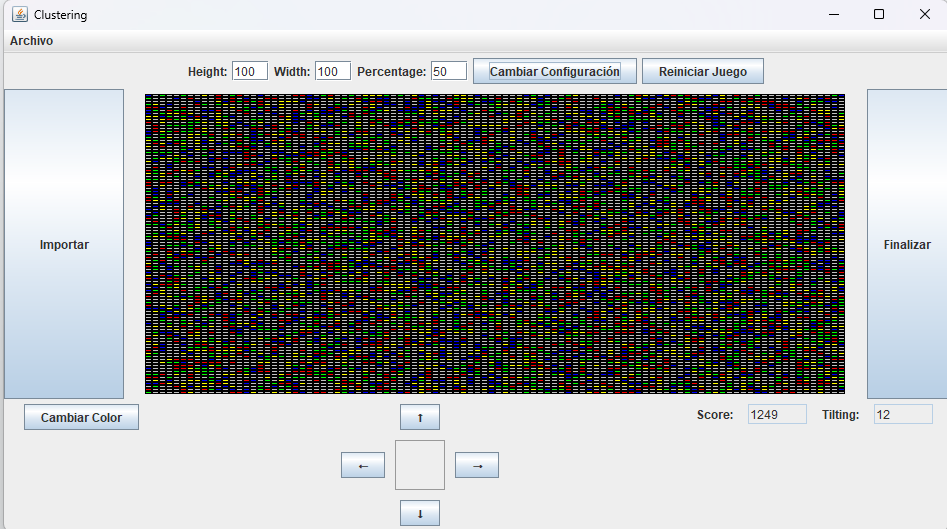


Ahora como requerimiento se pone que el score puede ser mayor de 300 porque el GridLayout permite crear tableros con dimensión muy grande y valdría la pena ver cómo se comporta score y los otros métodos con este caso.

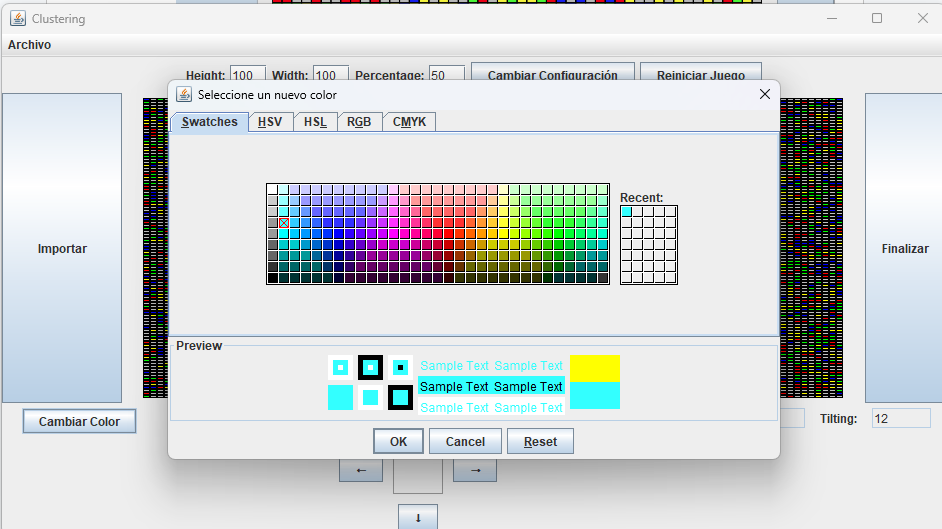
Ahí nos damos cuenta que automáticamente calcula el score de 684, y por cada tilt que se hace si se va sumando 1 en tilting

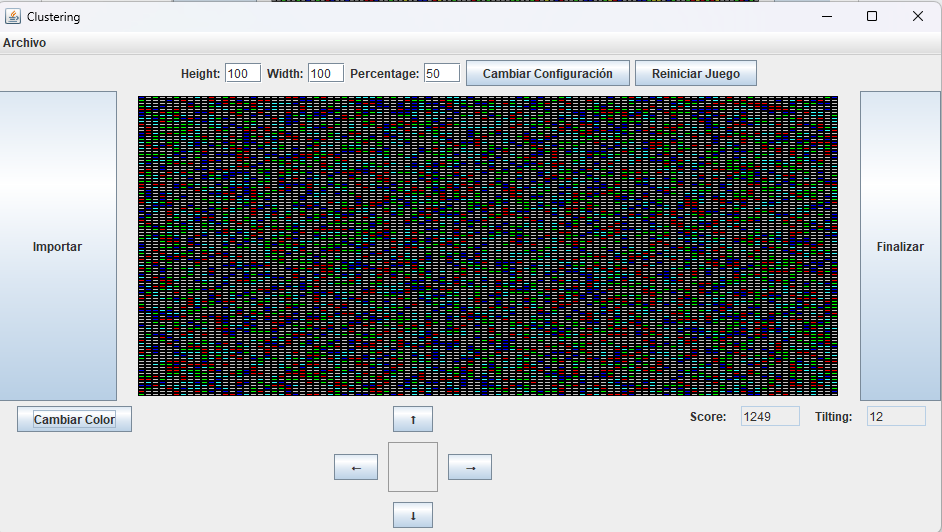


Ahora, digamos que quiero cambiar la configuración a uno de 100 x 100 con p=50.



Todo se ejecuta normal, y vemos el score, ahora también se puede ver que se puede cambiar de color, digamos todas las baldosas amarillas a azul claro.





## 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.

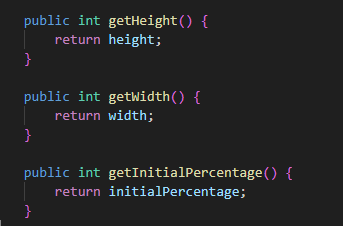
Los elementos para tener en cuenta para lograr reiniciar el juego dependiendo si tiene la primera configuración o después si cambió de configuración son:

* En la capa de presentación (ClusteringGUI), el botón resetButton se utiliza para reiniciar el estado del juego. Se activa luego de crear el tablero, y al presionarlo, llama al método resetGame().
* El botón permite que el jugador reinicie el estado del tablero al original, ya sea desde la configuración inicial o la configuración más reciente.
* El primer tablero se almacena en una copia (initialBoardState) para que se pueda restaurar más adelante al reiniciar el juego.
* copyBoard(char[][]) se utiliza para crear una copia profunda del estado del tablero.
* Dependiendo de si se está en la configuración inicial (isInitialConfig es verdadero), se vuelve al estado inicial del tablero (initialBoardState). Si se ha cambiado la configuración, se crea un nuevo tablero con la última configuración establecida.
* Clustering maneja el movimiento de las baldosas (tilt()), la configuración inicial (initializeBoard()), y la restauración del tablero (setBoard()).
* Al reiniciar, se crea una nueva instancia de Clustering que toma las dimensiones y el porcentaje inicial para restaurar el tablero a un estado equivalente al original.

1. Implementen los elementos necesarios para reiniciar.

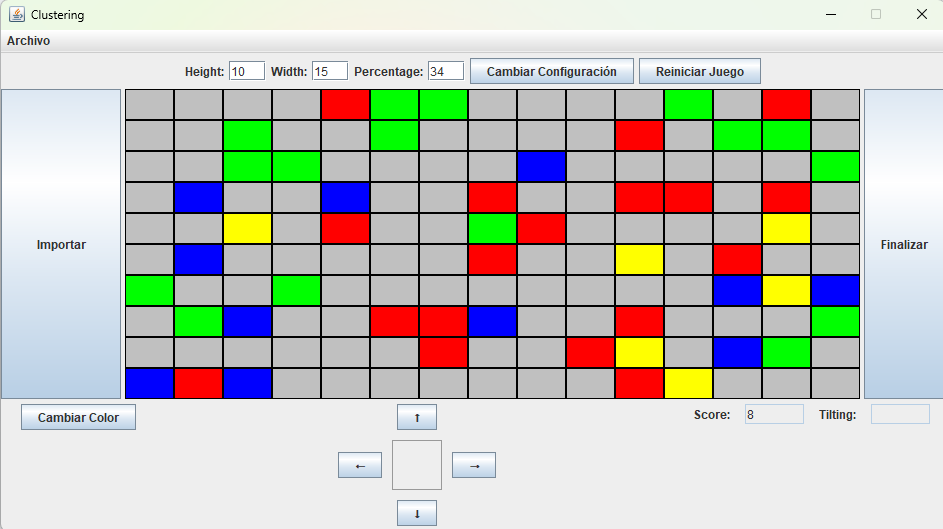




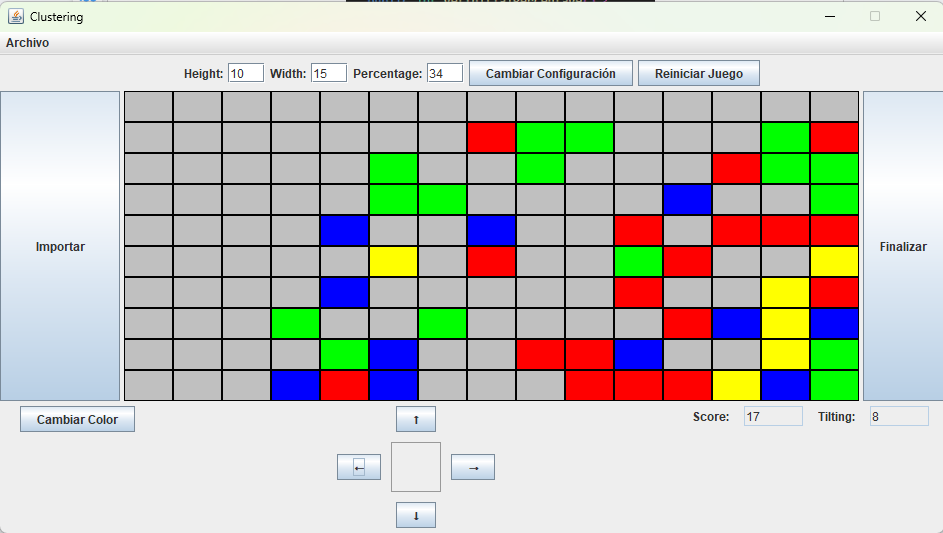


1. Ejecuten el caso de uso y capturen las pantallas más significativas.

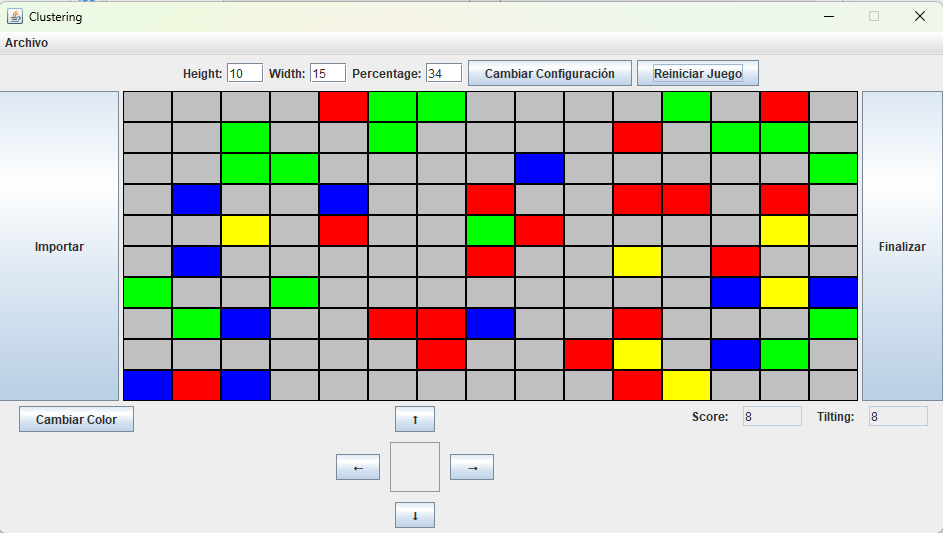
Ahora, para probar este botón que quedará al lado de cambiar configuración, se añade un tablero de cualquier tamaño.



Ahora, ya que tenemos la configuración inicial del tablero, procederemos a hacer varios tilt(8).

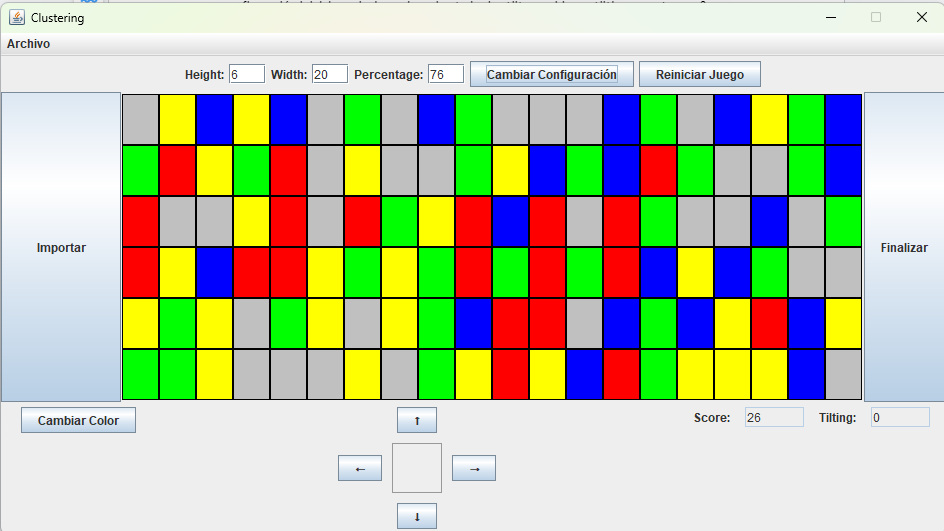


Luego de esto, ahora si oprimimos reiniciar juego, lo que debe pasar es que volvemos a la configuración inicial, es decir, se devuelve todos los tilts que hizo, y tiltings se setea en 0.

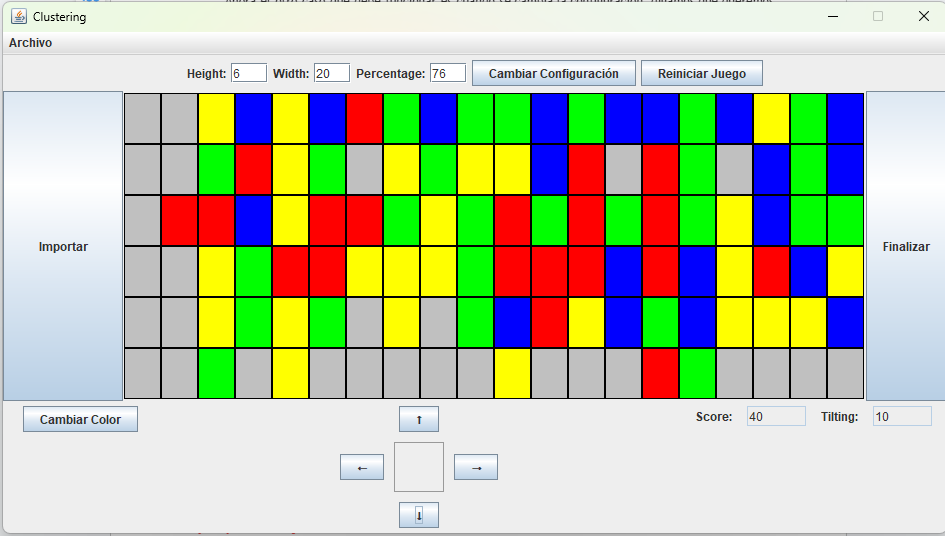


Vemos que coincide…

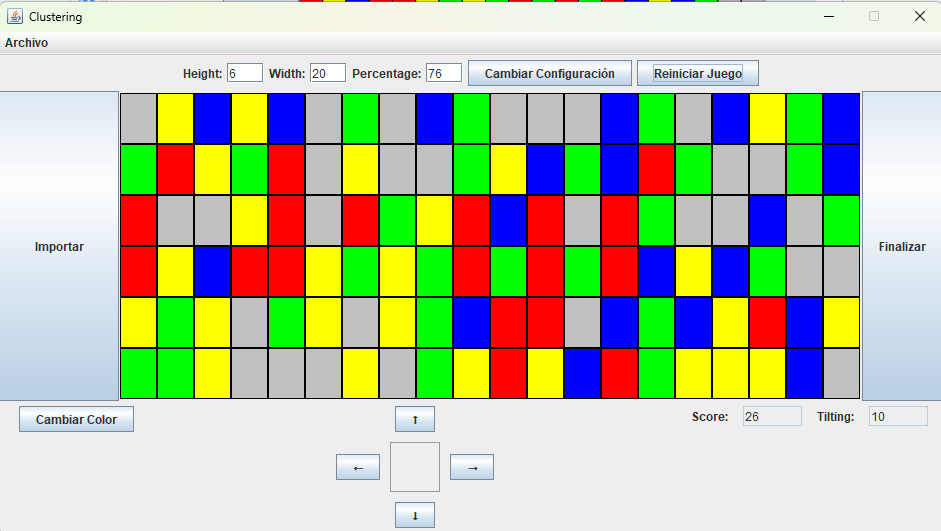
Ahora el otro caso que debe funcionar es cuando se cambia la configuración, digamos que queremos poner otro h, w, percentage, y hacemos varios tilts, si se hace clic en reiniciar juego, debería volver al estado original, pero no de la configuración inicial del tablero, sino al estado del tablero cuando se cambió la configuración.

Cuando se cambió de configuración:

Después de hacer tilt:



Si se reinicia el juego, queda con estado original sin hacer tilt:



## 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.

Solo es necesario definir otro método que se va a llamar prepareChangeScreenSize(), y se va a añadir otro botón al lado de score y tilting, que cuando se le dé clic, cambie el tamaño de la pantalla, en este caso, lo que implementamos es un botón que “maximiza” la pantalla, pero no usando setResizable(true), sino de la misma forma como se implementó de cuando se inicia el juego, que es con un cuarto de la pantalla, solo que ahora el screenSize va a ser el tamaño de la pantalla, y así se logra que cuando se hace uso del botón, desaparece la opción porque ya no es posible devolverse a jugar en un cuarto de pantalla.

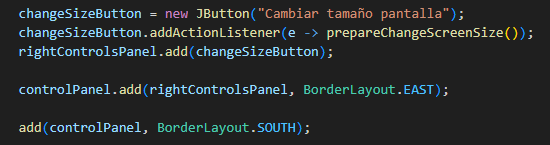
Se usa la API Dimension para saber el tamaño de la pantalla, y setSize para acomodar el nuevo tamaño de la pantalla referido al height, width.

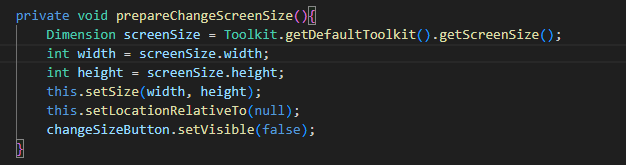
También para la creación del botón se utiliza JButton, y cuando se oprime, se acciona el evento que cambia el tamaño de la pantalla, en este caso en SOUTH, pero en EAST.

1. Implementen los elementos necesarios para cambiar el tamaño del juego.



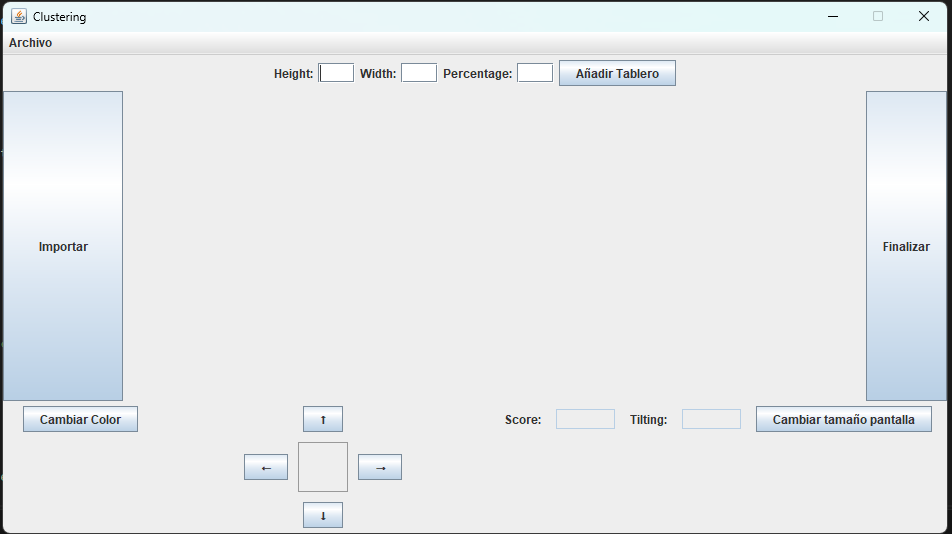




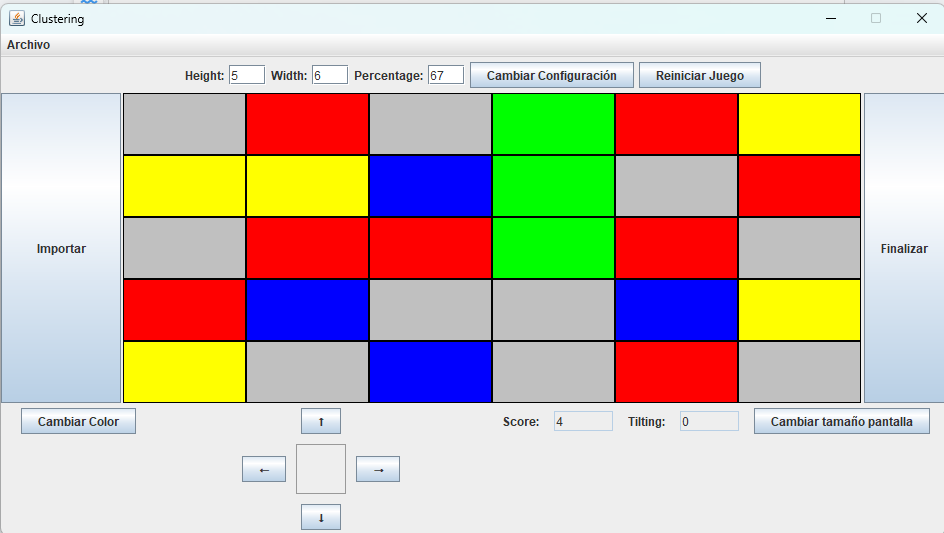


1. Ejecuten el caso de uso y capturen las pantallas más significativas.

Siempre empezamos desde un cuarto de pantalla, aquí se ve el botón cambiar tamaño pantalla:



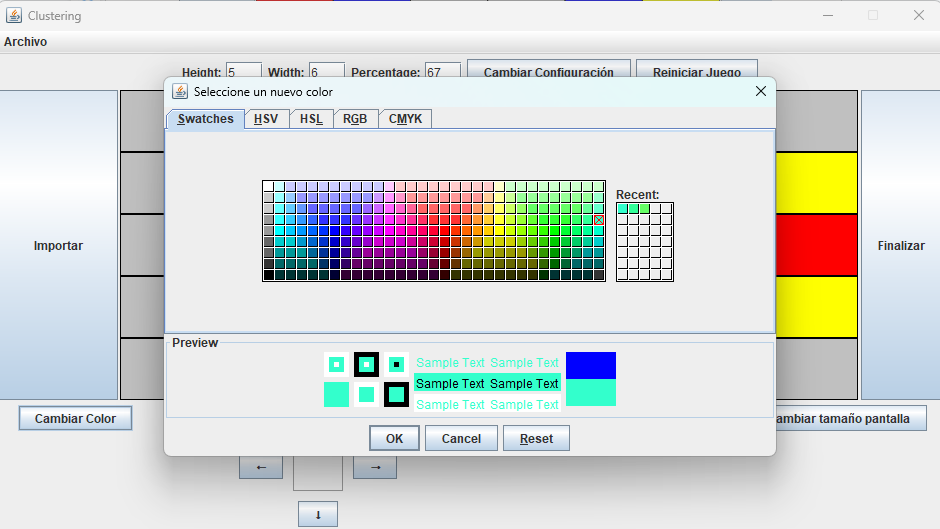
Ahora, el único caso de uso implementado para este nuevo botón, es que digamos que creo un tablero.



Hago tilt varias veces



También, cambio las baldosas azules a otro color:

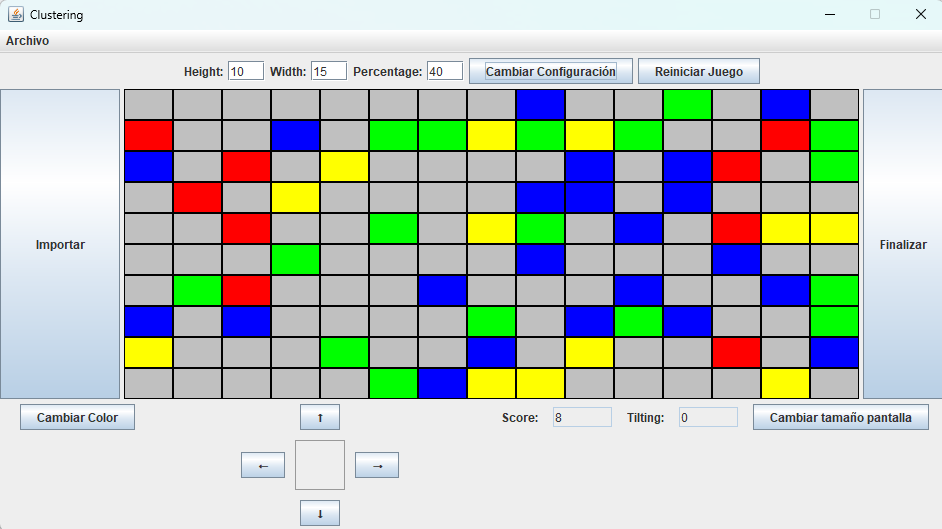




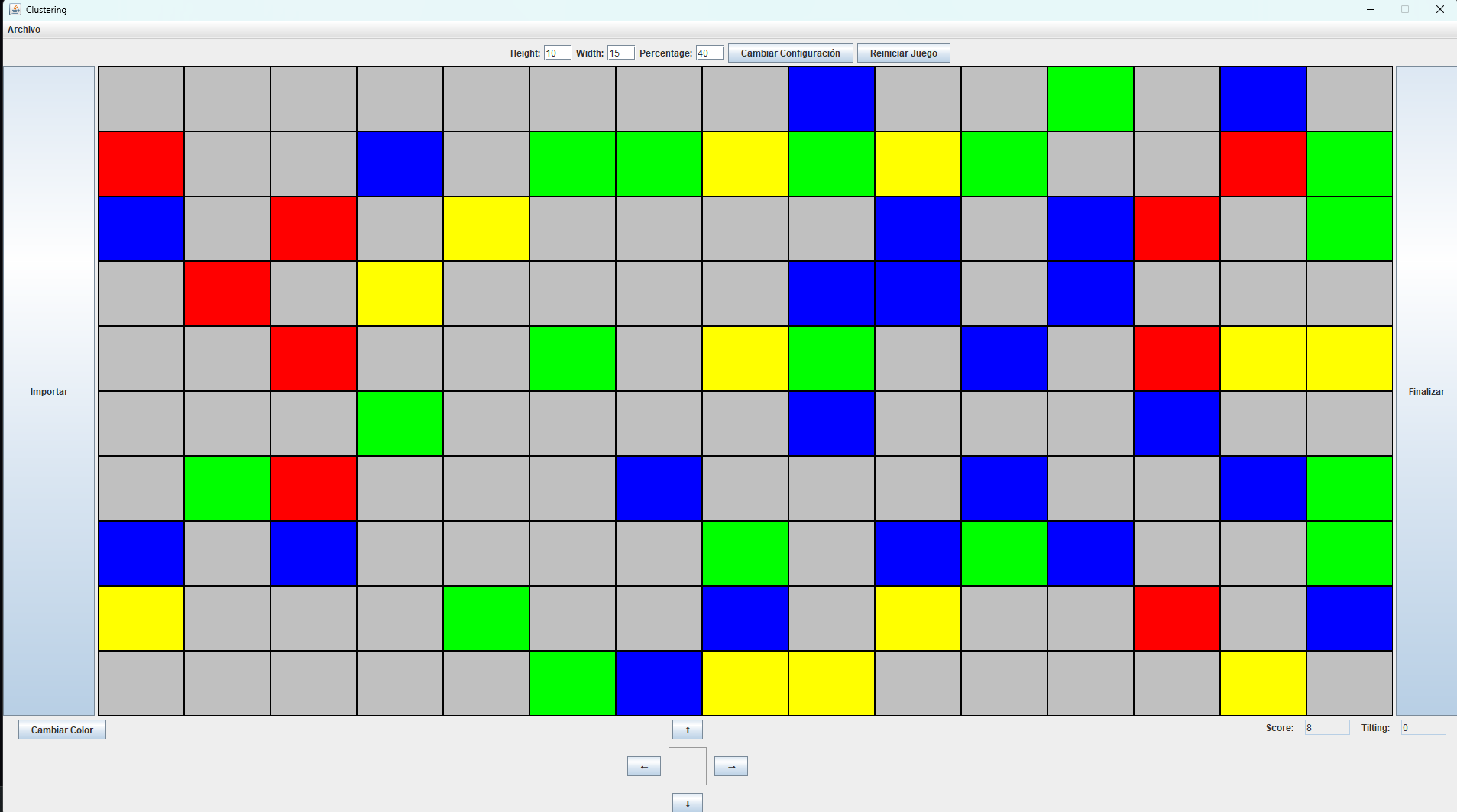
Además, ahora hago tilt con estas baldosas:



Ahora, quiero cambiar la configuración por otro h, w, percentage:

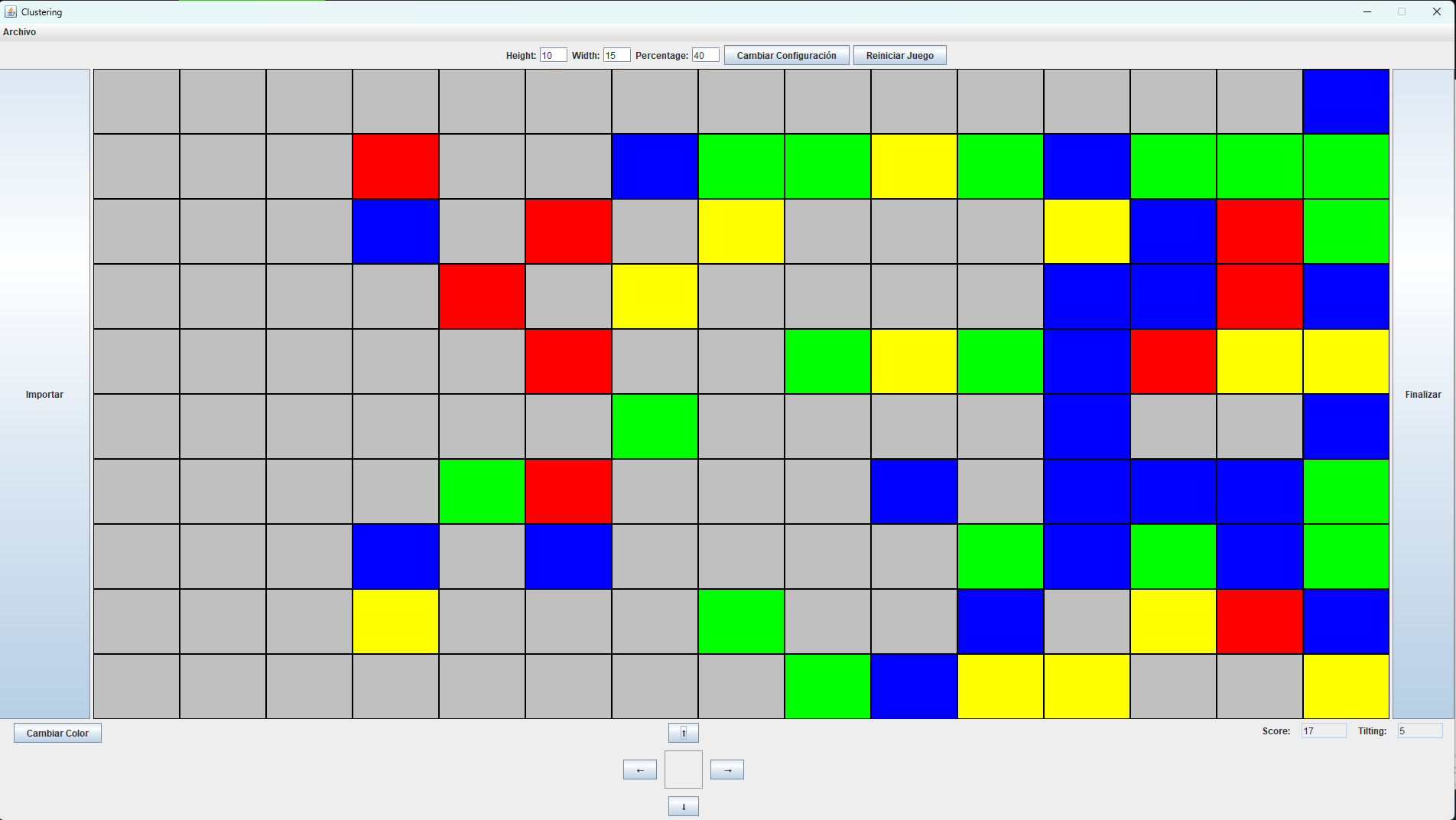


Para adicionar en este caso de uso e implementar el requisito de este ciclo, entonces ahora esto que estoy viendo en un cuarto de pantalla, ahora se va a ver en pantalla completo usando el botón cambiar tamaño pantalla.

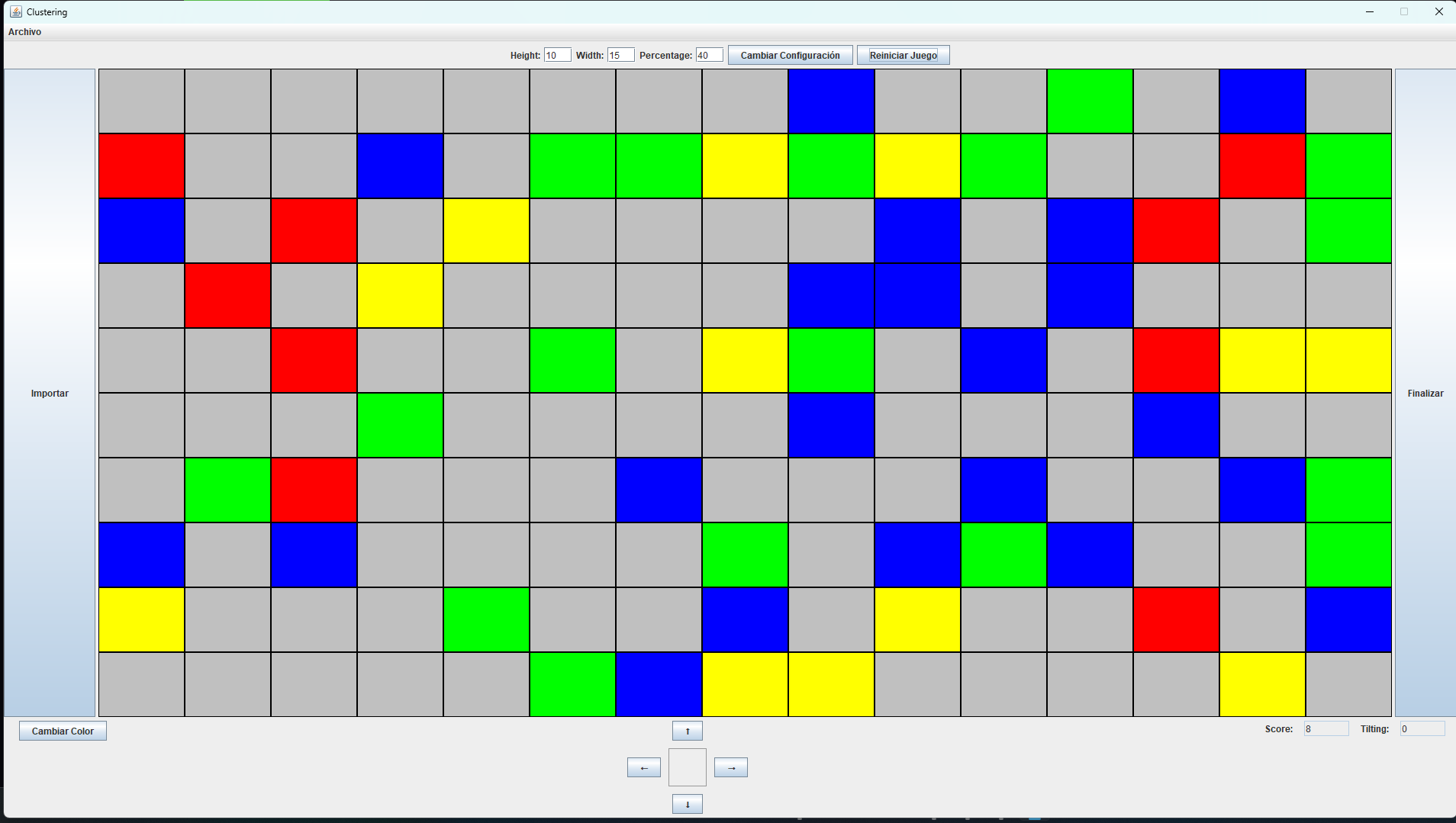


Si nos damos cuenta, ya no se puede devolver al cuarto de pantalla, ahora solo se juega en pantalla completa.

Y para evidenciar, se hace tilt y todo funciona con normalidad



Y si le damos reiniciar juego, debe mostrarse la configuración inicial después de cambiarla, reiniciando también tiltings.



CONTINUACIÓN

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**|**

**V**

**COMO UN RESUMEN DE TODO LO VIVENCIADO DE ESTE JUEGO CON LOS OBJETIVOS PROPUESTOS, Y LA IMPLEMENTACIÓN QUE SE HACÍA CADA VEZ QUE SE AVANZABA DE CICLO, HASTA LLEGAR AL CICLO FINAL.**

### OBJETIVOS CUMPLIDOS

1. **Patrón MVC:**
   * La clase Clustering representa la capa del modelo, almacenando la lógica del juego y el estado del tablero. La interfaz gráfica (vista) y los manejadores de eventos (controlador) se implementaron en la clase ClusteringGUI.
2. **Manejo de eventos con clases anónimas:**
   * Los botones y componentes interactivos (como JButton y menús) utilizan clases anónimas para implementar eventos como clics y selecciones; como los movimientos (tilts) se manejan mediante ActionListener anónimos asignados a cada botón de dirección.
3. **Uso de ventanas (JFrame, JDialog, JOptionPane):**
   * ClusteringGUI utiliza JFrame.
   * JOptionPane para confirmar acciones como salir del juego o notificar errores de entrada.
   * JDialog se usa para componentes como el selector de colores (JColorChooser).
4. **Selección de lienzos apropiados:**
   * Contenedores como JPanel para las celdas del tablero y JScrollPane para mejorar la navegación en tableros grandes.
   * JTabbedPane no fue usado en este caso, ya que no era necesario para la estructura de la aplicación.
5. **Uso de estilos (FlowLayout, BorderLayout, GridLayout):**
   * FlowLayout en los paneles de control y entrada.
   * BorderLayout para estructurar la interfaz general, así como se definió el bosquejo inicial.
   * GridLayout para organizar las celdas del tablero.
6. **Componentes básicos (JLabel, JTextField, JButton, JMenuBar):**
   * JLabel para etiquetas en campos de entrada y en la interfaz.
   * JTextField para capturar dimensiones del tablero y porcentaje.
   * JButton para todas las acciones principales (generar tablero, cambiar configuración, etc.).
   * JMenuBar con menús para nuevo, abrir, guardar y salir del juego.
7. **Componentes especiales (JFileChooser, JColorChooser):**
   * JFileChooser permite abrir y guardar el estado del juego en archivos.
   * JColorChooser permite cambiar los colores de las baldosas.
8. **Prácticas XP:**
   * Las pruebas de aceptación regularmente se hacían cuando pasaban NullPointerException, entonces debugueando o revisar el código, bien nos dábamos cuenta de que solo faltaba inicializar alguna variable en algún método. También el código compila correctamente, pero visualmente a veces se evidenciaba métodos que no deberían comportarse de esa manera, así que se revisó la lógica del código y listo. De resto encontramos excepciones definidas en el laboratorio de h, w, percentage inválidas o tilt (dirección inválida), o en otro caso, cuando las dimensiones del tablero no coinciden con el tamaño actual, en cuanto se cambiaba la configuración inicial, así, solo se hicieron pruebas robustas para los métodos principales definidos en el lab.

### COMPONENTES DE LA INTERFAZ

1. **Botones (JButton):**
   * **↑, ↓, ←, →:** Realizan los movimientos del tablero en las cuatro direcciones. Es decir, es el tilt, que cuando se hace ←, mueve todas las baldosas hacia → una sola posición.

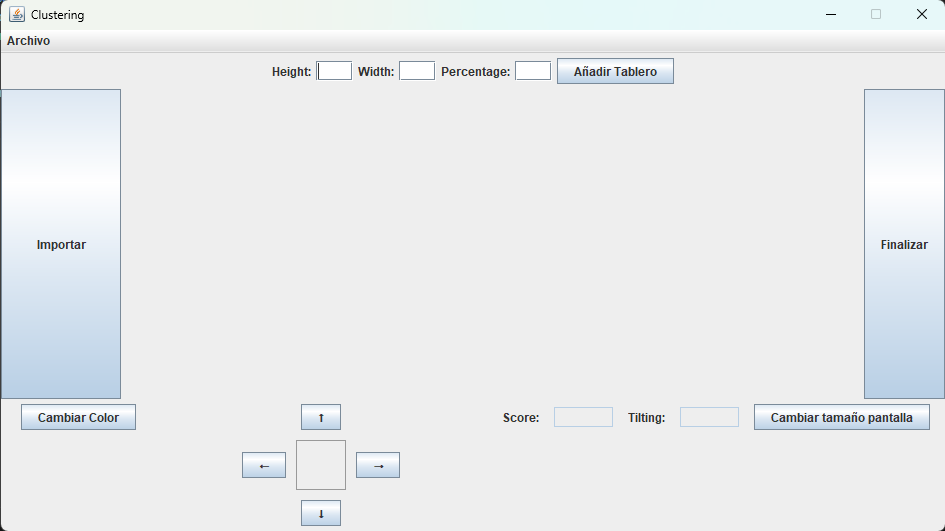
* + **Añadir Tablero:** Genera el tablero con las dimensiones y porcentaje ingresados válidos.
  + **Cambiar Configuración:** Permite cambiar las dimensiones y porcentaje del tablero.
  + **Cambiar Color:** Cambia el color de manera personalizada de las baldosas utilizando JColorChooser.
  + **Reiniciar Juego:** Reinicia el tablero a su estado inicial.
  + **Cambiar el tamaño de la pantalla:** “Maximiza la ventana, pero usando API Dimension”.
  + **Importar juego:** Importa el juego, pero como no está este requerimiento en el lab, entonces se tomó por ahora como si importar = abrir, no es verdad esto, pero así se toma por ahora.
  + **Finalizar:** Sale del juego.

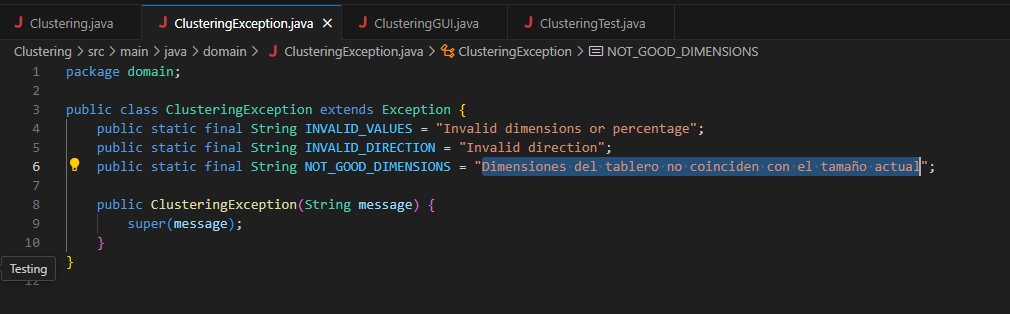
1. **Paneles (JPanel):**
   * **Tablero:** Contiene las celdas organizadas en un GridLayout.
   * **Controles:** Contiene botones de dirección y opciones en un diseño de BorderLayout.
2. **Menús (JMenuBar):**
   * Archivo: Incluye opciones como "Nuevo", "Abrir", "Guardar" y "Salir" usando JFileChooser para el funcionamiento de abrir, guardar.

Ahora describiré lo nuevo que se implementó, que creíamos que es necesario, así no lo hayan puesto como requisito.

* Cuando se añade tablero, después desaparece ese botón, y solo aparece cambiar configuración.
* Se cambia el color de las baldosas de un solo color a otro color personalizado, entonces al hacer tilt, se debe comportar igual, moviéndose una posición hacia la dirección clickeada, e igualmente si hay baldosas adyacentes de ese nuevo color, también lo cuenta en el score, tiltings de la misma manera.
* Con el nuevo color de baldosas, si se cambia la configuración, debe solo poner baldosas con colores aleatorios básicos (r, g, b, y).
* Ahora, si se tiene esos colores personalizados y se hacen varios tilt, y se le da reiniciar juego, vuelve a la configuración inicial de ese tablero, así tenga colores personalizados porque así fue el tablero inicial.
* Cabe aclarar que cuando se reinicia o se cambia de configuración, tiltings se setea en cero.
* Ahora hay tres modos de terminar el juego, con la x, con el botón finalizar, o desde menú salir.
* Se da la funcionalidad a menú-Nuevo, este lo que hace es que cuando se oprime Nuevo, setea todo el juego, es decir, toca otra vez poner h,w, percentage, añadir tablero, y empezar a jugar, entonces con JOptionPane se genera el mensaje apropiado.
* También cuando se oprime en el botón cambiar tamaño pantalla, se hace invisible ese botón una vez se haya cambiado el tamaño, esto con el fin de que el jugador no pueda volver a jugar con un cuarto de pantalla.
* Se crea la clase excepción con sus respectivos mensajes.

Así queda la interfaz definitiva:





Cada método está documentado, para saber qué hace cada uno, con sus respectivos comentarios, si es necesario.

# RETROSPECTIVA

1. **¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)**

El tiempo total invertido fue de más 24 horas en trabajo en equipo.

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

Laboratorio terminado porque se fue hasta el nivel de detalle más mínimo para cumplir con todos los objetivos establecidos. Trabajo en equipo.

1. **Considerando la práctica XP del laboratorio, ¿por qué consideran que es importante?**

Consideramos que “When a bug is found tests are create” fue la práctica más significativa porque no todo era pruebas de aceptación en código, sino más, viendo visualmente que pasaba si uno oprimía algún botón, y no generaba lo que no debería pasar, entonces nos dio la necesidad de hacer pruebas implícitas debugueando para así cumplir con lo requerido. Y pues las pruebas de aceptación en general nos sirvieron para probar los métodos principales. Entonces también nos obligó a generar la clase Excepción para mostrar de forma bonita los errores con JOptionPane.

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

El mayor logro, como siempre, es terminar el laboratorio con todos los requerimientos y añadir funcionalidades implícitas que no están descritas.

El mayor problema fue al hacer tilt con baldosas que cambian de color de manera personalizada con el botón, y el de cambiar configuración adecuado a este mismo. Se solucionó muy bien analizando de cómo se actualizaba el board cuando se hacía tilt, y vimos que no se estaba actualizando correctamente esas baldosas de otro color, estaban estáticas; por eso, esos dos métodos nos costaron problema.

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

Nos comprometemos a usar esta práctica XP con más frecuencia para lograr una mejor eficiencia en trabajo en equipo porque algunas veces solo nos guiábamos de hacer solo pruebas unitarias, probando todo por separado, pero es que a veces ver primero visualmente lo que debería pasar es necesario antes de revisar código, ya sabiendo lo que toca arreglar, se corrige, y se da la solución de forma correcta.