Memoria
Práctica 1
SumaTres

GIITIN01-2-006 Comunicación Persona-Máquina

Alumno:
UO283319
Juan Francisco Mier Montoto

Bloque	Incluido
1	Sí
2	Sí
3	Sí
4	Sí
5	Sí

Contenido

Introducción

En este informe se detallará tanto el código fuente, como el funcionamiento del programa SumaTres, el cual se presenta como proyecto de la primera parte de la asignatura Comunicación Persona-Máquina.

Temática del proyecto

SumaTres es un simple juego un jugador que trata de conseguir la mayor puntuación posible sumando piezas adyacentes del mismo valor, excepto las piezas 1 y 2 que solo se pueden sumar entre sí. La partida termina cuando el tablero esté lleno y el jugador no tenga la posibilidad de sumar ninguna ficha.

SumaTres no solo es el proyecto de esta primera parte de la asignatura, sino que es una versión muy desarrollada del proyecto final de la asignatura de Introducción a la Programación, entregado el 14 de enero de 2021.

Puesto que se trata de un proyecto expandido, TODA la interfaz es funcional, es decir, TODO lo que se puede activar, desactivar, seleccionar, modificar, tocar, o utilizar, sirve y hace su función, o al menos debería.

En su estado actual, al programa le faltan algunas funcionalidades planeadas y cambios al funcionamiento del juego, pero debido al estricto límite de tiempo impuesto, no me ha sido posible avanzar más allá. La adaptación del proyecto a los “estándares” que se han de cumplir en este trabajo, como incluir una interfaz desde cero, uso de threads, varias ventanas, diálogos predefinidos, etc., han significado alrededor de 50 horas.

Guía de usuario básica

El juego es bastante intuitivo debido a que es una variante del popular juego “2048”, disponible en plataformas móviles y a través de navegador. En concreto, SumaTres es una adaptación del juego *AddThree* presente en la aplicación “CrazyPlus”.

Prepartida

Antes de empezar una partida, el usuario debe escoger entre varias opciones que afectan ligeramente al trascurso de la partida.

Las dos más importantes será el tamaño del tablero y el modo de juego:

- El tamaño del tablero es de vital importancia, especialmente para la duración de la partida. Un tablero grande, por ejemplo de 7x7, puede llegar a tener una duración de partida de media hora, dependiendo de las acciones del usuario y de las fichas siguientes. Por lo tanto, para mantener una duración de partida adecuada y ya probada, se recomienda jugar con el tamaño por defecto, 5x5, para mantener el reto del juego y la atención del usuario.
- El modo de partida es un cambio implementado para tratar de ofrecer más opciones de juego y mejor jugabilidad al usuario. El modo clásico se mantiene fiel a las restricciones del enunciado original: salida por consola, cuatro direcciones de movimiento y poco más. El modo experimental trata de añadir un poco más al juego, al haber modificaciones dependiendo del tamaño del tablero, 8 direcciones de movimiento, más opciones de interfaz y la posibilidad de activar “trucos”.

Estas opciones se escogen siempre antes de empezar una partida, aunque se pueden guardar ficheros de opciones con extensión “.sto” y cargar dichas opciones en cualquier otra partida. Si el proceso detecta un fichero llamado “default.sto”, trata de cargar dichas opciones al comienzo de cada partida.

Controles

El usuario puede jugar con el teclado, usando “wasd” como esquema clásico de movimiento en videojuegos, o “qweadzxc” en modo experimental. También puede jugar mediante clics de ratón sobre las flechas que indiquen la dirección del movimiento que desea realizar.

Cómo jugar

Al comenzar cada partida, se le presenta al usuario el tablero del tamaño que haya escogido y una cantidad de piezas. A partir de esas piezas, el usuario debe sumar las piezas entre sí, sabiendo que las piezas que sean iguales o mayores que "3" se suman solo con otras piezas iguales y que las piezas con valor 1 y 2 solo se suman entre sí. Al sumar dos piezas, se genera otra pieza, con el valor de la suma, por lo que a lo largo de la partida se van a conseguir piezas de tamaño superior.

El objetivo del usuario puede ser o conseguir la máxima puntuación o conseguir la pieza con un valor más alto.

Opciones durante la partida

Durante la partida, se pueden cambiar algunas opciones a través de la interfaz. La cantidad de opciones que se pueden cambiar se expande si se está jugando en modo experimental:

- Se pueden activar o desactivar casi cualquier elemento gráfico de la partida, excepto el tablero y las piezas. Además, se pueden activar otros elementos gráficos como el grid de tablero, o las coordenadas de las piezas.
- Se puede pasar entre el modo experimental y el modo clásico y viceversa.
- Se pueden activar "trucos", que pueden modificar la partida casi de cualquier manera, pero desactivan la puntuación final (el multiplicador de dificultad pasa a 0.0):
    - Añadir, modificar o eliminar cualquier pieza.
    - Insertar la pieza siguiente sin pasar un turno.
    - Modificar el valor de la pieza siguiente.
    - Añadir puntos manualmente al contador.
    - Permitir que el programa "juegue por ti" al hacer un bucle de jugadas aleatorias.
    - Volver al tablero del turno anterior.

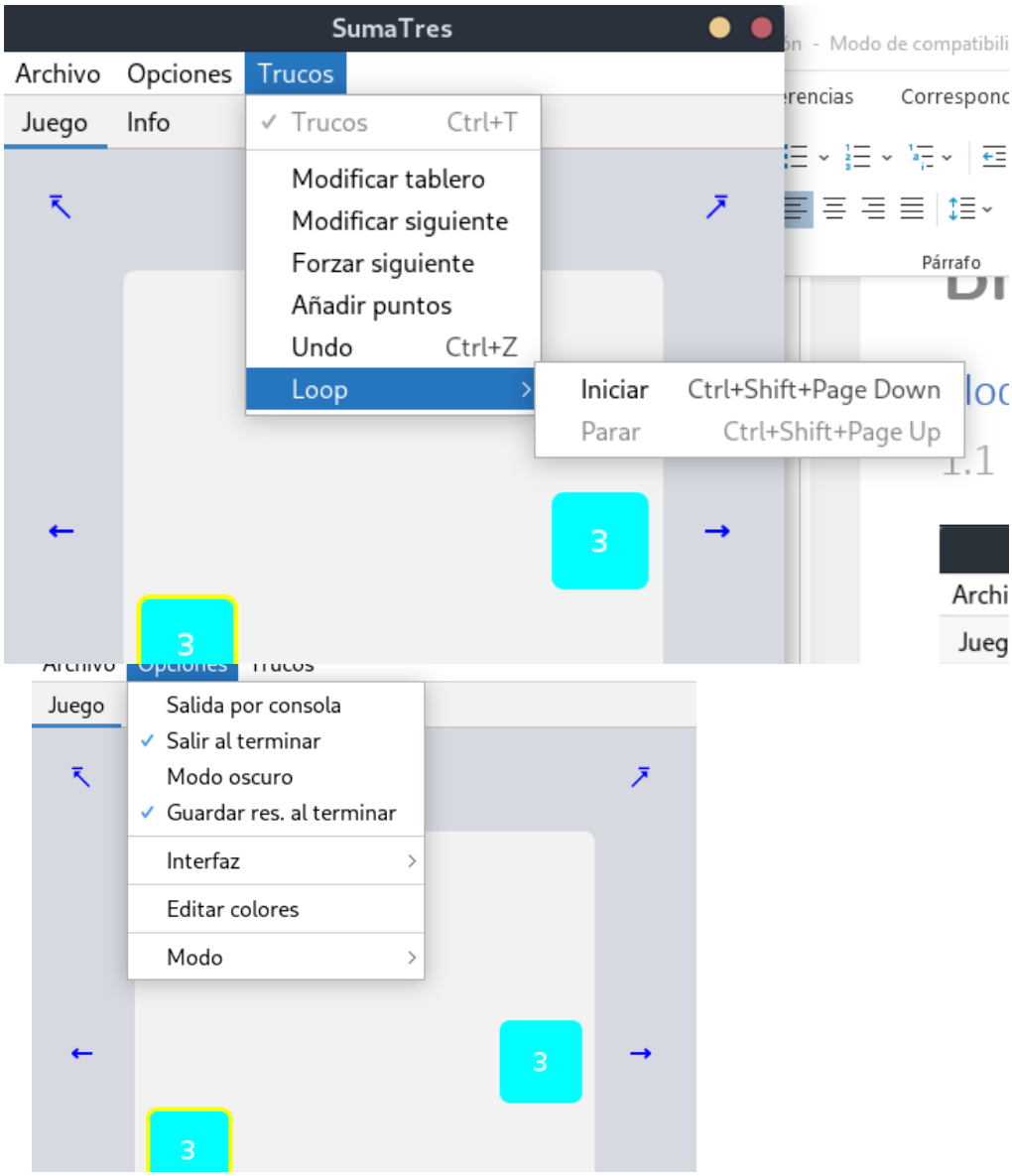
Además, se pueden cambiar otras opciones en cualquier modo:

- Se puede activar o desactivar el modo oscuro, en cualquier momento.
- Se puede cambiar el color de cualquier pieza, esté o no en pieza, siempre que ya tenga un color asociado.
- Se puede activar o desactivar la salida por consola, es decir, la actualización de la partida se imprime por consola en cada turno. Esta es la manera original de representación del enunciado original.
- Se puede activar o desactivar el guardado de los resultados al terminar la partida.
- Se puede activar o desactivar la salida automática de la partida al terminar.
- Se puede establecer el nivel de *verbosidad* por consola (0 – 2).

Bloques

Bloque 1. Ventana y eventos

1.1 Barra de menús con algunos menús y submenús



1.2 Algunos componentes en un *layout* diferente del default

Las interfaces dependen bastante del diseño "libre", por lo que se ha optado por incluir un "CardLayout" como forma de cumplir en este requisito aunque no cambie nada visualmente en la ventana principal.

Ventana	Componente	Layout
game.LauncherRF	JFrame	CardLayout

1.3 Componentes modificados por eventos

Toda la interfaz funciona a base de eventos, incluyendo sliders, botones, toggleButtons, checkboxes, menús desplegables…

Ventana	Componente		Clase	Evento
gui.Prepartida	Modificado	btnClásico	Button	btnClásicoActionPerformed(AE evt)
gui.Avanzadas	Impactado	chkConsoleOutput	Checkbox	

Código

```
    private void btnClásicoActionPerformed(java.awt.event.ActionEvent evt) {
        op = new Settings(op.getX(), op.getY(), false);
        avanzadas.readValues();
    }

    public void readValues() {
        Settings op = ventanaSecundaria.getSettings();

        btnExperimental.setSelected(op.isExperimental());
        btnClásico.setSelected(!op.isExperimental());

        chkDiagonalMovement.setEnabled(op.isExperimental());
        chkHUD.setEnabled(op.isExperimental());
        chkCheatsAvailable.setEnabled(op.isExperimental());
        chkMoreNextValues.setEnabled(op.isExperimental());
        chkBalancedStart.setEnabled(op.isExperimental());
        chkPaintArrows.setEnabled(op.isExperimental());
        ...
    }

// Existen muchos ejemplos de componentes que modifiquen otros componentes mediante
// eventos. El mejor de ellos son los botones de cambio de modo en la pantalla
// prepartida, ya que estos modifican todos las checkboxes en la pantalla Avanzadas
// simultáneamente.
```

1.4 Tratamiento de eventos de teclado

Las acciones del usuario se interceptan en la ventana principal y se pasan a una clase de la partida que permita modificar lo pertinente. Se hace así para poder modificar algún elemento (en este caso se actualizan valores de otra ventana) cuando el usuario haga una acción.

Ventana	Componente	Evento
game.LauncherRF (principal)	juego (game.SumaTres)	evento (KeyEvent / MouseEvent)

Código

```
        juego.addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent event) {
                new Keyboard(juego, event).keyboardHandler();
                ventanaColores.updateValues();
            }
        });

        juego.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent event) {
                new Mouse(juego, event).mouseHandler();
            }
        });
```

1.5 Modificación de componentes con un número variable de elementos

NOTA: no se refieren al mismo método, updateValues() es un método diferente en cada ventana, pero ambos modifican sus componentes respectivos.

Ventana	Componente		Evento
gui.EditarColores	Modificado	lstValores (lista)	updateValues()
gui.ModifyBoardDialog	Modificado	cmbValores (combobox)	updateValues()

Código

```
    public void updateValues() {
        model.removeAllElements();
        for(Integer i : Pieza.getColores().keySet()) {
            model.addElement(i);
        }
        model.removeElement(-2);
        model.removeElement(-1);
        model.removeElement(0);
    }

    public void updateValues() {
        cmbValores.removeAllItems();
        for(Integer i : Pieza.getColores().keySet()) {
            cmbValores.addItem(String.valueOf(i));
        }

        // se eliminan los valores -2, -1 y 0 del combobox.
        cmbValores.removeItem("-2");
        cmbValores.removeItem("-1");
        cmbValores.removeItem("0");
    }
```

1.6 Elementos no vistos

A esta sección hay que añadir que la clase principal del juego (game.SumaTres) extiende a la clase JPanel y por lo tanto es un componente complejo.s

Ventana	Componente	Evento
gui.ModifyBoardBoard	btnAñadir, btnModificar, btnEliminar (toggleButton)	btnAñadirActionPerformed(ActionEvent), btnModificarActionPerformed(ActionEvent), btnEliminarActionPerformed(ActionEvent)
gui.loadSaveDialog	pswCode (Password Field)	*No implementado.*

Código

```
     private void btnAñadirActionPerformed(java.awt.event.ActionEvent evt) {
        mode = 0;
        cmbValores.setEnabled(true);
    }

    private void btnModificarActionPerformed(java.awt.event.ActionEvent evt) {
        mode = 1;
```

```
        cmbValores.setEnabled(true);
    }

    private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
        mode = 2;
        cmbValores.setEnabled(false);
    }


// El código en el que se añade el objeto de la clase SumaTres:
public void launch(Settings op) {
        juego = new SumaTres(op);
        ...

       // Propiedades del JPanel de la partida.
        juego.setSize(Graphic.defineX(juego) + 15, Graphic.defineY(juego) + 39);
        juego.setLocation(0,0);
        juego.setBackground(op.isDarkModeEnabled() ? Paint.DARK_BACKGROUND :
Paint.LIGHT_BACKGROUND);

        // añade el panel de la partida y establece el orden correcto.
        jTabbedPane1.addTab("Juego", juego);
        jTabbedPane1.setComponentAt(0, juego);
```

* El objeto de la partida principal extiende por definición a JPanel y se añade manualmente a la ventana “LauncherRF” al mostrarse.

Bloque 2. Varias ventanas

2.1 La ventana principal modifica componentes de otra ventana

LauncherRF actualiza los valores de la lista de la ventana “EditarColores” cada vez que el usuario realiza una acción o cuando se accede al menú “Editar colores”.

Ventana principal	Ventana no principal	Componente
game.LauncherRF	gui.EditarColores	lstValores (su DefaultListModel se actualiza mediante updateValues())

Código

```
            public void keyPressed(KeyEvent event) {
                new Keyboard(juego, event).keyboardHandler();
                ventanaColores.updateValues();
            }

    public LauncherRF() {
        FlatLightLaf.setup(); // Se establece el modo claro por defecto.
        initComponents();
        ventanaColores = new EditarColores(this);
        ventanaColores.setVisible(false);
        secundaria = new PrePartida(this);
        secundaria.setVisible(true); // Se lanza la ventana de opciones prepartida.
    }
```

2.2 Una ventana no principal modifica componentes de la principal

La ventana “Prepartida” cambia el estado del modo oscuro de la ventana principal y de sus delegadas antes de que comience la partida. Además, se encarga de lanzar la ventana principal cuando el usuario le da al botón “JUGAR”, lo que modifica el TabbedPane de la ventana principal, habilita y deshabilita opciones de los menús, y más modificaciones.

Ventana principal	Ventana no principal	Componente
game.LauncherRF	gui.Prepartida	-

Código

```
    private void checkDarkMode() {
        principal.toggleDarkMode(op.isDarkModeEnabled());
        SwingUtilities.updateComponentTreeUI(this);
        SwingUtilities.updateComponentTreeUI(avanzadas);
    }

    public PrePartida(LauncherRF p) {
        this();
        principal = p;
        avanzadas = new Avanzadas(this);

        // las opciones por defecto son: tamaño 5x5, modo experimental.
        op = new Settings(5, 5, true);
        if(MAX_SIZE[1] < 15) sldHorizontal.setMaximum(MAX_SIZE[1]);
        if(MAX_SIZE[0] < 15) sldVertical.setMaximum(MAX_SIZE[0]);
        avanzadas.readValues();
        File defaultSettings = new File("default.sto");
        if(defaultSettings.exists()) {
            openSettingsFile(defaultSettings);
            System.out.println("Cargado archivo de opciones por defecto.");
        }
```

```
    }
```

Además, la ventana “EditarColores” edita los colores de las piezas de la ventana principal.

Ventana principal	Ventana no principal	Componente
game.LauncherRF	gui.EditarColores	-

Código

```
public EditarColores(LauncherRF p) {
    this();
    principal = p;
}

private void refresh(int selectedValue) {
    updateColorOfValue(selectedValue);
    updateBrightnessOfValue(selectedValue);
    updateColorLabel();
    principal.getPartida().repaint();
}
```

2.3 Una ventana no principal modifica componentes de otra ventana no principal

La ventana “Avanzadas” actualiza la selección de los botones del modo en la ventana “Prepartida”. Además, dichos botones también modifican los botones presentes a su vez en la ventana “Avanzadas”, además de establecer todos los checkboxes a su valor indicado dependiendo del modo seleccionado.

Ventana modificadora	Ventana modificada	Componente
gui.Avanzadas	gui.Prepartida	btnClásico, btnExperimental

Código

```
private void btnClásicoActionPerformed(java.awt.event.ActionEvent evt) {
    ventanaSecundaria.setClassic();
    readValues();
}

private void btnExperimentalActionPerformed(java.awt.event.ActionEvent evt) {
    ventanaSecundaria.setExperimental();
    readValues();
}

public Avanzadas(PrePartida p) {
    this();
    ventanaSecundaria = p;
}
```

2.4 Una ventana inicial que no sea la principal

La ventana de Prepartida siempre se lanza antes de la ventana principal para permitir el usuario cambiar cualquier opción que desee.

Ventana
gui.Prepartida
Código

```
    private PrePartida() {
        initComponents();
        btnJugar.requestFocus(); // permite comenzar la partida al darle enter nada
más iniciar.
    }

    public PrePartida(LauncherRF p) {
        this();
        principal = p;
        avanzadas = new Avanzadas(this);

        // las opciones por defecto son: tamaño 5x5, modo experimental.
        op = new Settings(5, 5, true);
        if(MAX_SIZE[1] < 15) sldHorizontal.setMaximum(MAX_SIZE[1]);
        if(MAX_SIZE[0] < 15) sldVertical.setMaximum(MAX_SIZE[0]);
        avanzadas.readValues();
        File defaultSettings = new File("default.sto");
        if(defaultSettings.exists()) {
            openSettingsFile(defaultSettings);
            System.out.println("Cargado archivo de opciones por defecto.");
        }
    }
```

Bloque 3. Diálogos

3.1 Un diálogo usando JOptionPane

Se utilizan múltiples JOptionPane a lo largo del programa, tanto como para output como para input. **La clase "Dialog" se encarga de mostrar los JOptionPane y devolver los valores que interesen.**

Ventana
game.LauncherRF

Código

```
    private void jmiModoClassicActionPerformed(java.awt.event.ActionEvent evt) {
        boolean check = true;
        if(juego.getMultiplier() != 0.0) check = Dialog.confirm("Esta acción cambia
el mutiplicador de puntos a cero.\n¿Desea continuar?");
        if(check) {
            juego.setMultiplier(0.0);
            juego.getSettings().setExperimentalMode(false);
            jmiTrucos.setEnabled(false);
            setCheatsEnabled(false);
            juego.repaint();
            actualizarPneInfo();
        } else {jmiModoExperimental.setSelected(true);}
    }

    private void jmiTrucosForzarSiguienteActionPerformed(java.awt.event.ActionEvent
evt) {
        try {
            juego.colocarSiguiente();
            juego.repaint();
            if(!Turno.ableToMove(juego)) juego.finalDePartida();
        } catch (NullPointerException ex) {Dialog.showError("No hay hueco para otra
pieza.");}
    }

    private void jmiResultsActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            if (Desktop.isDesktopSupported() &&
Desktop.getDesktop().isSupported(Desktop.Action.OPEN))
                Desktop.getDesktop().open(SumaTres.ARCHIVO);
            else throw new IOException();
        } catch (IOException ex) {jmiResults.setEnabled(false); Dialog.showError("No
se pudo abrir el archivo de resultados.");}
    }

    private void jmiTrucosActionPerformed(java.awt.event.ActionEvent evt) {
        if(Dialog.confirm("Activar los trucos cambiará el multiplicador de puntos a
0.\n¿Desea continuar?")) {
        ...
        } else {jmiTrucos.setSelected(false);}
    }
```

3.2 Un diálogo predefinido

Se utilizan tanto JFileChooser como JColorChooser.

Ventana
gui.EditarColores

Código

```
    private void btnEditarActionPerformed(java.awt.event.ActionEvent evt) {
        if(!lstValores.isSelectionEmpty()) {
            int selectedValue = (int)
 Pieza.getColores().keySet().toArray()[lstValores.getSelectedIndex() + 3];
 //          int selectedValue = Integer.parseInt(lstValores.getSelectedValue()); //
 <-- esto no funciona! no es culpa mía, es java.
            Color newColor = JColorChooser.showDialog(null, "Seleccione nuevo color",
 Pieza.getColores().get(selectedValue), false);
            if(newColor != null) {
                Pieza.getColores().put(selectedValue, newColor);
                refresh(selectedValue);
            }
        }
    }
```

3.3 Un diálogo creado por el usuario que pida información al usuario

Se utiliza el diálogo "ModifyBoardDialog" para preguntarle al usuario qué pieza quiere añadir, modificar o eliminar y además el valor de la nueva pieza, si es que se quiere modificar y no eliminar. Utiliza el método showDialog(), el atributo pressedOk y botones de cancelar y aceptar como visto en clase.

Ventana diálogo

gui.ModifyBoardDialog

Código

```
 public class ModifyBoardDialog extends javax.swing.JDialog {

    private boolean pressedOk;
    private SumaTres s;
    private PiezaDisplayer displayer;
    private int mode;

    /**
     * Creates new form CoordenadasMatriz
     */
    private ModifyBoardDialog(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        displayer = (PiezaDisplayer) pnlPieza;
    }

    public ModifyBoardDialog(SumaTres si) {
        this(null, true);
        this.s = si;
        sldHorizontal.setMaximum(si.getSettings().getY() - 1);
        sldVertical.setMaximum(si.getSettings().getX() - 1);
        updateValues();
   }

    public boolean showDialog() {
        pressedOk = false;
        this.setVisible(true);
        update();
        return pressedOk;
    }

 ...
 // La clase son 400 líneas de código.
}
```

LauncherRF llama al diálogo cuando el usuario utiliza el truco “Modificar tablero”.

Ventana
game.LauncherRF
Código

```
    public boolean modificarTablero() {
        boolean check = false;
        ModifyBoardDialog dialog = new ModifyBoardDialog(this);
        if (dialog.showDialog()) {
            if(dialog.getMode() == 0 || dialog.getMode() == 1)
setTab(dialog.getCoordsX(), dialog.getCoordsY(), dialog.getValue());
            else setTab(dialog.getCoordsX(), dialog.getCoordsY(), 0);
            update();
            check = true;
        }
        deactivateSelected();
        return check;
    }
```

Bloque 4. Interfaz en primer plano

4.1 Métodos set para dar información a la tarea

En realidad, la tarea no necesita más información que la aportada a través del constructor. No tengo ni idea de por qué es un requerimiento. Aún así, se puede establecer el límite de jugadas aleatorias que realiza el bucle si se desea.

Clase
game.LauncherRF → thread.LoopTask
Código

```
        loopComms.setLimit(-1);

    public void setLimit(int val) {
        if(val >= 0) limit = val;
        else if(val == -1) limit = Integer.MAX_VALUE;
    }
```

Además, se puede establecer la cantidad de tiempo en milisegundos que el bucle espera antes de realizar otra jugada.

Clase
game.LauncherRF → thread.LoopTask
Código

```
        loopComms.setSlowdown(50);

    public void setSlowdown(int val) {
        if(val >= 0) slowdown = val;
   }
```

4.2 Métodos que envíen información de la tarea a la interfaz

La tarea informa del progreso del bucle a la interfaz, que lo muestra en la sección “Info”. El progreso es un cálculo sencillo pero costoso: la cantidad de piezas en tablero entre la cantidad de huecos del tablero, asumiendo que se está hacia el final del bucle si quedan pocos espacios vacíos en mesa.

Clase
thread.LoopComms → game.LauncherRF
Código

```
    @Override
    public void Progress(int val) {
        ventanaPrincipal.setProgress(val);
    }
```

4.3 Posibilidad de hacer un stop - continue

El método que permite continuar la tarea es el mismo que la inicia.
El método de la tarea que continúa el bucle es el mismo que el que lo para.

Clase
thread.loopComms
Método stop / continue:

```
     public void setStop(boolean t) {
         this.isStopped = t;
    }
```

Llamada al método stop:

Clase
game.LauncherRF
Código

```
     private void jmiTrucosLoopEndActionPerformed(java.awt.event.ActionEvent evt)
 {
         jmiTrucosLoopEnd.setSelected(true);
         loopComms.setStop(true);
    }
```

Bloque 5. Gráficos

NOTA: por lo general, toda la lógica de Paint se encuentra en la clase `util.Paint`.

5.1 Clase hija de un componente del que se refine su método Paint

Se utiliza el método paintComponent() (que es más correcto que Paint()) para pintar TODO el objeto de la partida, incluyendo texto.

Clase

game.SumaTres

Código

```
    /**
     * Método que redirige los gráficos a {@link #paint(Graphics)}. <p>
     * También pide el focus de la ventana al comenzar a pintar, para evitar que
se pierdan
     * pulsaciones de teclado al comenzar la partida.
     */
    @Override
    public void paintComponent(Graphics g) {
        requestFocusInWindow(); // <- IMPORTANTE! necesita el focus para poder
redirigir el teclado
        super.paintComponent(g);
        new Paint(this, g).paint();
    }
```

5.2 Métodos set para modificar lo que se pinta en la clase anterior

A la hora de pintar el tablero se realizan MUCHAS modificaciones por turno a la herramienta Graphic2D, como se puede apreciar en los ejemplos de abajo. Son tan solo ejemplos, se cambia muchas veces de color y de tamaño de fuente por turno.

Clase

util.Paint

Código

```
        g.setRenderingHints(rh);
        g.setStroke(new BasicStroke(STROKE_SIZE));
        g.setFont(new Font("Helvetica", Font.PLAIN, size));
        g.setColor(BOARD_COLOR);
```

Código

```
Declaración de las estructuras de datos.
```

5.3 Utilización del método repaint

El método repaint se utiliza numerosas veces a lo largo de la ejecución como manera de actualizar el estado del tablero, piezas y otra información.

Clase

…

Código

```
Found 23 matches of repaint() in 10 files.
SumaTres.java
                    repaint();      [position 357:4]
SumaTres.java
                    repaint();      [position 416:4]
             repaint();      [position 593:3]
SumaTres.java
                    repaint();      [position 416:4]
             repaint();      [position 593:3]
SumaTres.java
                    repaint();      [position 357:4]
Tablero.java
                    repaint();      [position 48:4]
GetMatrixCoordsDialog.java
             s.repaint();      [position 185:15]
LauncherRF.java
             juego.repaint();      [position 472:19]
         juego.repaint();      [position 507:15]
         juego.repaint();      [position 600:15]
         juego.repaint();      [position 605:15]
         juego.repaint();      [position 619:15]
             juego.repaint();      [position 629:19]
         juego.repaint();      [position 669:15]
         juego.repaint();      [position 674:15]
         juego.repaint();      [position 679:15]
SumaTres.java
         repaint();      [position 590:9]
         repaint();      [position 830:9]
EditarColores.java
         principal.getPartida().repaint();      [position 204:32]
         displayer.repaint();      [position 233:19]
ModifyBoardDialog.java
         displayer.repaint();      [position 294:19]
         s.repaint();      [position 295:11]
```

5.4 Elemento no visto en clase ni en apuntes

El objeto de tipo “Graphic” se transforma a “Graphic2D”, se le aplican pistas de antialiasing y otras mejoras gráficas y se dibuja el tablero utilizando dichas mejoras, para eliminar dientes de sierra y otros desperfectos.

Clase

util.Paint

Código

```
        var rh = new RenderingHints(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        rh.put(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);
        rh.put(RenderingHints.KEY_COLOR_RENDERING,
RenderingHints.VALUE_COLOR_RENDER_QUALITY);
        rh.put(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BICUBIC);
        rh.put(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
        /*
         *  Gracias al antialiasing,  los bordes de las fichas, son mucho más
```

```
suaves y
         *  aptos para una pantalla de gran dpi y resolución. No afecta al
rendimiento del programa.
         *  Algunas de estas mejoras no suponen una clara diferencia en la
calidad visual del juego.
         */

        g.setRenderingHints(rh);
        g.setStroke(new BasicStroke(STROKE_SIZE));
```