

# Programación J2ME con Netbeans

*Interfaz gráfico para el tetris*  
Septiembre de 2006



**DECSAI**  
Departamento de Ciencias  
de la Computación e I.A.  
Universidad de Granada

Curso de Formación Continua de Programación de  
dispositivos móviles con Java (4<sup>a</sup> edición)

Septiembre de 2006



# Índice

<b>1. Introducción</b>	<b>5</b>
<b>2. Creación del proyecto</b>	<b>6</b>
2.1. Pasos iniciales de creación . . . . .	6
2.2. Refactorización del código . . . . .	7
2.3. Cambiar el nombre al MIDlet . . . . .	8
<b>3. Diseñador de flujo</b>	<b>9</b>
3.1. Creación de un screen alert . . . . .	10
3.2. Conexión del screen alert con el flujo de la aplicación . . . . .	11
<b>4. Visualización del código fuente generado por el IDE</b>	<b>11</b>
<b>5. Diseñador de pantallas</b>	<b>12</b>
5.1. Añadir nuevos componentes en una pantalla . . . . .	13
5.2. Añadir componentes no visuales . . . . .	14
5.3. Inserción de imágenes en el proyecto . . . . .	16
<b>6. Creación de un Canvas como panel del juego</b>	<b>17</b>
<b>7. Asignación de acciones a los comandos del Canvas y hello-Form</b>	<b>21</b>
<b>8. Pintar el tablero de juego en el Canvas</b>	<b>22</b>
<b>9. Movimiento de la figura del tetris</b>	<b>24</b>
<b>10. Movimiento a izquierda y derecha y rotación de la figura</b>	<b>26</b>
<b>11. Posibles mejoras del juego</b>	<b>27</b>



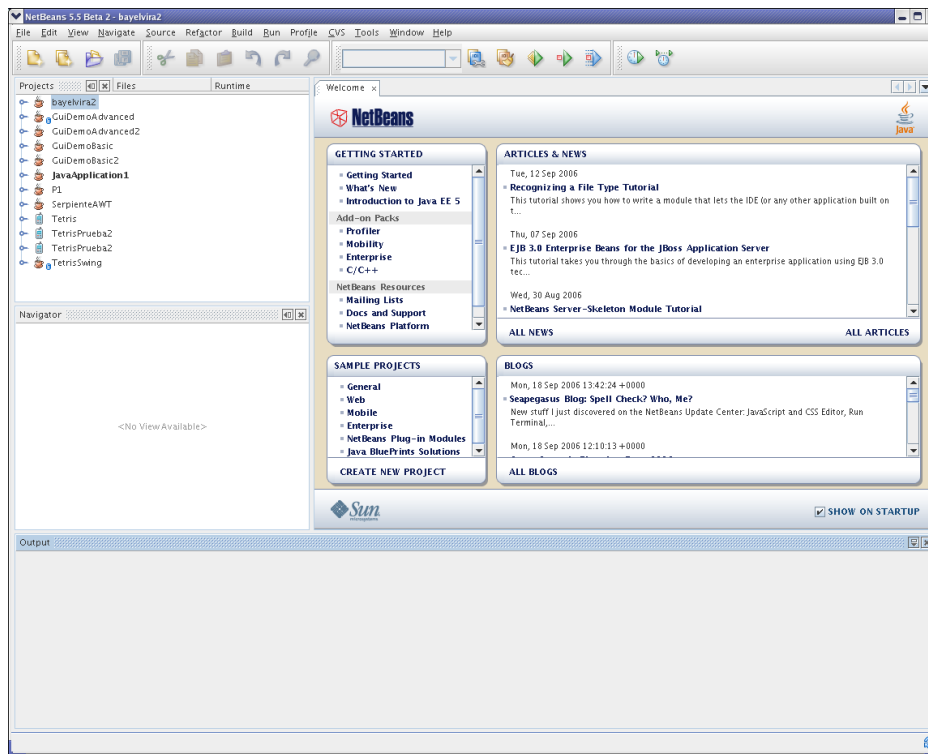


Figura 1: Ventana inicial de Netbeans

## 1. Introducción

**Netbeans** es un entorno de desarrollo integrado (IDE) que permite editar programas en java, compilarlos, ejecutarlos, depurarlos, construir rápidamente el interfaz gráfico de una aplicación eligiendo los componentes de una paleta, etc.

La idea de este guión es que el alumno aprenda por sí sólo las principales utilidades que proporciona **Netbeans** para construir el interfaz gráfico de una aplicación J2ME. Nosotros utilizaremos la versión 5.0 de **Netbeans** junto con el **Netbeans Mobility Pack 5.0**. Esto nos facilitará enormemente la tarea de construir aplicaciones J2ME, sobre todo en cuanto al interfaz gráfico.

Este software tanto para linux como para Windows, puede descargarse gratuitamente de la página web <http://www.netbeans.org/>. En esa página también pueden encontrarse diversos documentos para aprender a usar Netbeans. Existe un libro sobre Netbeans que puede encontrarse en pdf en la dirección

<http://www.netbeans.org/download/books/definitive-guide/index.html>

aunque el libro está hecho para la versión 3.5 (la última versión estable es la 5.0, aunque actualmente existe una versión beta2 de netbeans 5.5).

**Netbeans** 5.0 se encuentra instalado en las aulas de prácticas de la ETSII en linux fedora core 4, en el directorio `/usr/local/netbeans5.0`. Puedes ejecutar netbeans abriendo un terminal y ejecutando el comando:

```
netbeans
```



Figura 2: Tetris funcionando en el emulador

Tras unos segundos, aparecerá una ventana similar a la de la figura 1. En este gui3n construiremos un programa J2ME para jugar a una versi3n simple del juego del tetris. El aspecto que tendr3 el programa en el emulador una vez terminado es el que aparece en la figura 2.

## 2. Creaci3n del proyecto

### 2.1. Pasos iniciales de creaci3n

El primer paso consiste en crear el proyecto para el juego del tetris. Despu3s de iniciar el IDE de Netbeans, seleccionamos **Men3 File** → **New Project**. En la primera p3gina del wizard para *New Project* seleccionamos **Mobile** como categor3a y **Mobile Application** como tipo de proyecto (figura 3). Pulsamos ahora el bot3n **Next**.

En la siguiente ventana (figura 4) introducimos **Tetris5.5** como nombre del proyecto. En **Project Location** introduce el directorio (carpeta) donde guardas tus proyectos Netbeans. Este directorio debe estar creado previamente. En mi caso es el directorio `/home/gte/acu/CursoJavaMobil/ProjectsNB`.

Pulsa de nuevo el bot3n **Next** asegur3ndote antes de que las opciones *Set as Main Project* y *Create Hello Midlet* est3n seleccionadas. Aparece la siguiente ventana (figura 5) que nos permite seleccionar la plataforma con la que queremos compilar nuestro programa (versi3n de MIDP y CLDC). Dejaremos seleccionadas las opciones por defecto (CLDC-1.1 y MIDP-2.0).

En la siguiente ventana (figura 6) pulsamos el bot3n **Finish** para terminar de crear el proyecto, sin seleccionar ninguna opci3n.

Tras unos instantes, el IDE habr3 creado un nuevo proyecto MIDP 2.0 que puede ser compilado y ejecutado desde el mismo entorno. Por ahora el proyecto contiene 3nicamente la clase *HelloMIDlet* y el paquete *hello*

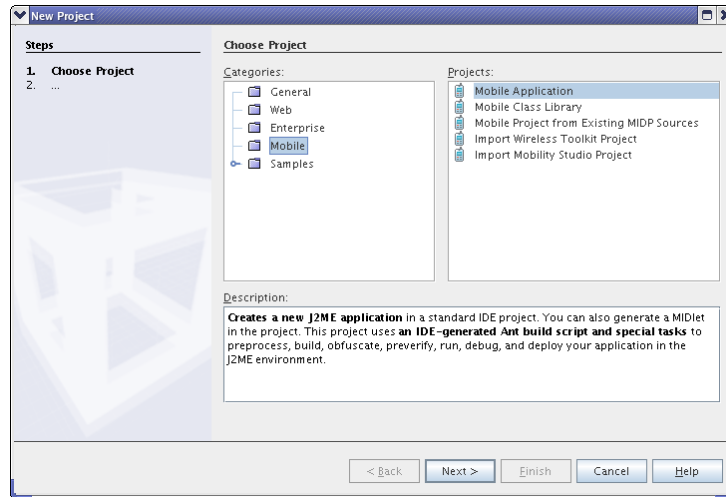


Figura 3: Creación de una nueva aplicación

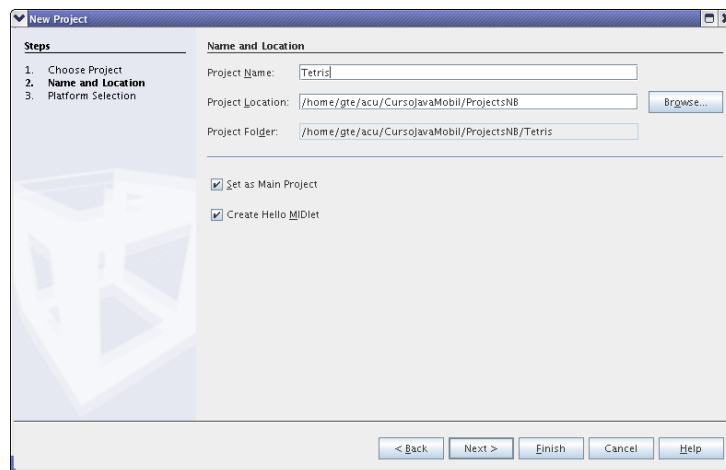


Figura 4: Definiendo el nombre del proyecto

como puede observarse en el panel superior izquierda del IDE (*panel de proyectos*), suponiendo que la solapa *Projects* está seleccionada (figura 7).

## 2.2. Refactorización del código

A continuación cambiaremos el nombre de la clase principal y del paquete usando otros nombres más adecuados para este proyecto. Usaremos como nombres de clase y paquete, *TetrisMidlet* y *tetris* respectivamente. Esto puede hacerse pinchando con el botón derecho del ratón en el nombre del paquete (en el panel de proyectos, donde aparece la estructura jerárquica de nuestro proyecto), y seleccionando **Refactor** → **Rename ...**. Aparecerá una ventana (figura 8) donde introduciremos *tetris* como nuevo nombre para el paquete. Marcaremos la casilla *Apply Rename on Comments* para que el cambio afecte también a los comentarios del código fuente, y pulsaremos el botón **Next**. Si en esa ventana teníamos seleccionado la casilla *Preview All Changes*, entonces será necesario pulsar el botón **Do Refactoring** en el *panel Output* (panel de Salidas) que está en

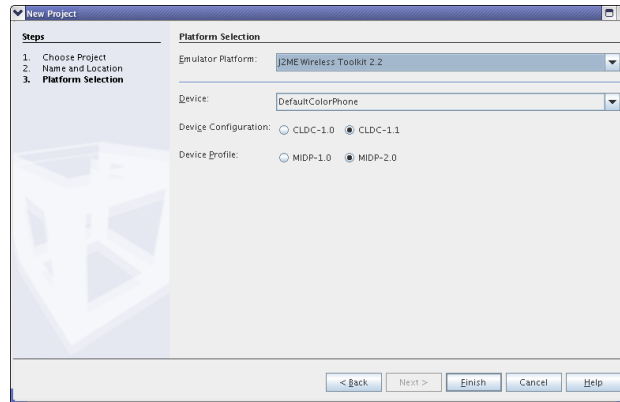


Figura 5: Ventana de configuración de la plataforma para el proyecto

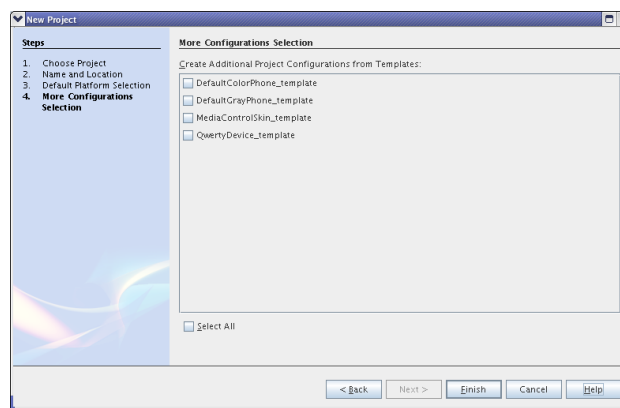


Figura 6: Ventana de creación de configuraciones de proyecto adicionales

la parte inferior izquierda del IDE (figura 9).

Repite la misma operación para el nombre de la clase (archivo `HelloMiddlet.java`) usando ahora el nombre *TetrisMidlet*.

### 2.3. Cambiar el nombre al MIDlet

Ahora vamos a poner *Tetris* como nuevo nombre para el MIDlet. Para ello selecciona **Menú File** → **Tetris Properties** para que se muestre el diálogo de propiedades del proyecto (10). Selecciona el item *MIDlets* dentro del nodo *Application Descriptor* y cambia el nombre del MIDlet de *HelloMIDlet* a *Tetris*. Finalmente, pulsa el botón **Ok** para aceptar los cambios.

El código fuente J2ME generado por netbeans, puede verse si pinchamos en la solapa *Source* del *panel de diseño* (panel del lado derecho). Aparece algo así como lo que muestra la figura 11.

Podemos ejecutar el proyecto mediante **Menú Run** → **Run Main Project**. Esto hará que se compile la aplicación y se lance el emulador J2ME. La aplicación no hace mucho por ahora.

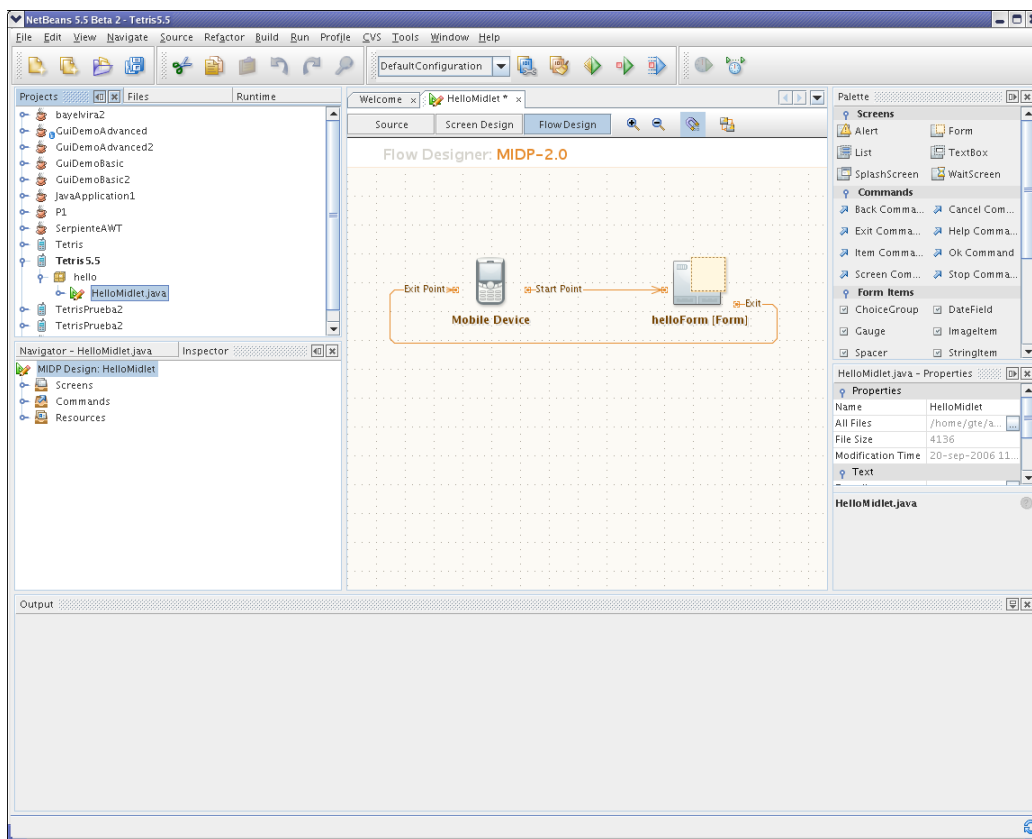


Figura 7: IDE tras crear el proyecto

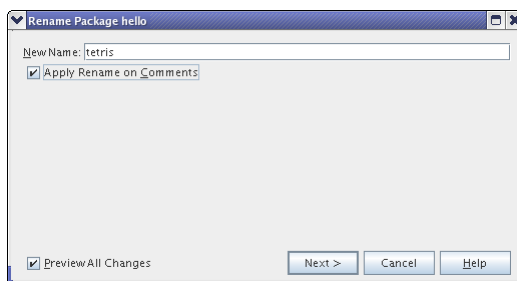


Figura 8: Renombrar el paquete

### 3. Diseñador de flujo

El diseñador de flujo (*Flow Designer*) nos muestra una representación gráfica de alto nivel del flujo de la aplicación, esto es, de las relaciones entre las diferentes pantallas (*screens*) del midlet incluyendo las transiciones que comienzan y finalizan la aplicación. El IDE de Netbeans genera automáticamente la mayoría del código fuente, lo que hace que sea una forma muy simple y adecuada de desarrollar el interfaz de usuario de una aplicación.

El diseñador de flujo aparece en la parte central del IDE de Netbeans. Para ello, la solapa *TetrisMidlet* correspondiente a nuestro proyecto debe estar seleccionada. También debe estar seleccionada la solapa *Flow Designer* (figura 12).

El siguiente paso que haremos será el de añadir una pantalla de pre-

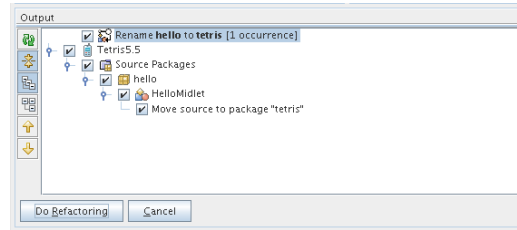


Figura 9: Realizar la refactorización

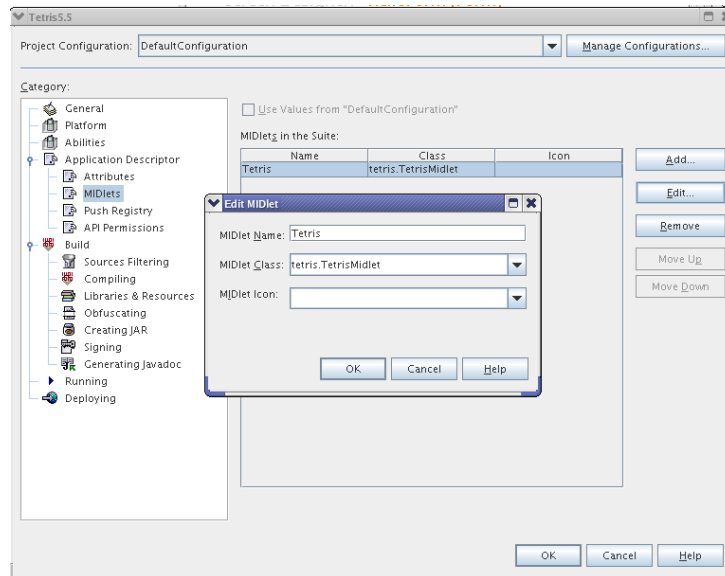


Figura 10: Cambio de nombre al midlet

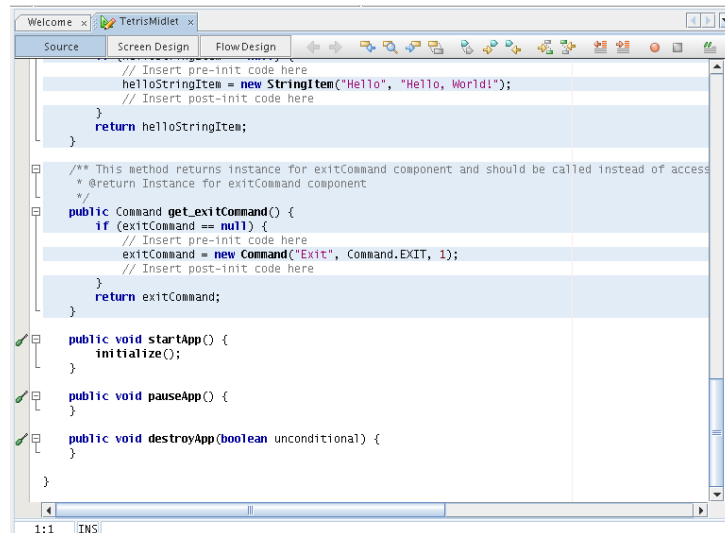
sentación a nuestro proyecto. Nos servirá para mostrar información sobre la aplicación cuando ésta comience a ejecutarse. Nosotros emplearemos para ello una alerta de MIDP (*alert screen*).

### 3.1. Creación de un screen alert

Para añadir la *alert screen* a nuestra aplicación arrastramos el ítem *Alert* de la paleta de componentes (situado a la izquierda de la ventana *Flow Designer*, que está dentro de la categoría *Screens*) hacia el área blanca del diseñador (figura 13).

Selecciona el nuevo screen alert en la ventana del diseñador de flujo para modificar sus propiedades *Title* y *String*. En la ventana de propiedades (*Properties*) situada a la derecha del IDE, introduce el valor que tú quieras en los campos *Title* y *String*. En mi caso he introducido *Java Tetris* y *Bienvenido al juego del Tetris* respectivamente. En *Timeout* introduce 2 segundos (valor 2000).

Finalmente, en el apartado *Code Properties*, renombra el ítem *Instance Name* con el nombre *splashAlert*. Esto hará que en el código fuente de *TetrisMidlet*, se cree una variable llamada *splashAlert* que representará la nueva *screen* creada.



```

// Insert pre-init code here
helloStringItem = new StringItem("Hello", "Hello, World!");
// Insert post-init code here
}
return helloStringItem;
}

/** This method returns instance for exitCommand component and should be called instead of access
 * @return Instance for exitCommand component
 */
public Command get_exitCommand() {
    if (exitCommand == null) {
        // Insert pre-init code here
        exitCommand = new Command("Exit", Command.EXIT, 1);
        // Insert post-init code here
    }
    return exitCommand;
}

public void startApp() {
    initialize();
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
}
    
```

Figura 11: Código fuente

### 3.2. Conexión del screen alert con el flujo de la aplicación

Conectemos ahora la alerta con el flujo de la aplicación. Pincha con el botón derecho del ratón sobre el icono *Mobile Device* en la ventana *Flow Designer*. Selecciona *Properties* en el menú, para abrir las propiedades de diseño del Midlet (figura 14).

En el diálogo resultante pulsa el botón etiquetado con ... de la propiedad *Start Point Action*. Esto hace que se abra un nuevo diálogo para esta propiedad (figura 15). En este diálogo, modificar la pantalla destino (*Target screen*) a *splashScreen* y *Alert Forward screen* a *helloForm*. Pulsa el botón *Ok* y cierra la ventana de propiedades de diseño del midlet pulsado el botón *Close*.

Podemos ver ahora en la ventana del *Flow Designer* que la alerta ha sido insertada en el flujo de la aplicación (figura 16). Ejecuta ahora el proyecto para ver cual es el aspecto de la aplicación tras inserta la alerta.

## 4. Visualización del código fuente generado por el IDE

El diseñador de flujo (*Flow Designer*) permite manejar las pantallas (*screens*) y los comandos asociados con estas pantallas. Las líneas representan transiciones desde una pantalla a otra en respuesta a un comando. Aunque todo esto lo estamos haciendo de forma gráfica, realmente lo que está ocurriendo es que el *Flow Designer* está insertando código Java en el fichero *Tetrismidlet.java* para llevar a cabo todas esas tareas. Podemos ver el código generado (figura 17) seleccionando la solapa *Source* (a la izquierda de la solapa *Flow Design*). Parte del código fuente nos aparece sombreado en azul. Esto nos indica que ese código no puede modificarse con el editor de código fuente (con el teclado), sino que tendremos que ha-

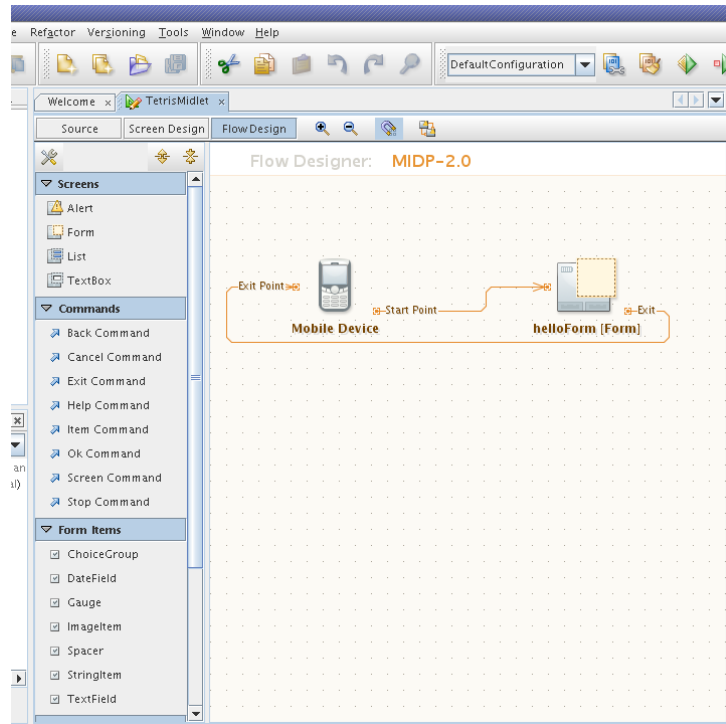


Figura 12: Flow Designer

cer uso de las herramientas visuales que estamos viendo en este tutorial. Netbeans no permite editar el código sombreado en azul, y nunca deberíamos de hacerlo con un editor externo, pues entonces ya no podríamos seguir modificando el proyecto con netbeans.

Cada pantalla y comando tiene asociadas dos propiedades, *Pre-Init User Code* (código fuente que se ejecuta antes de instanciar la pantalla o comando) y *Post-Init User Code* (código fuente que se ejecuta después de instanciar la pantalla o comando) que pueden usarse para insertar código fuente adicional. Si queremos insertar algún código en una de estas dos partes, lo haremos directamente en la parte adecuada del código fuente donde pone `// Insert pre-init code here` o bien `// Insert post-init code here`.

## 5. Diseñador de pantallas

El diseñador de pantallas (*screen designer*) se usa para editar las pantallas de la aplicación. Hay dos formas de acceder al diseñador de pantallas:

1. La forma más simple es desde la pantalla del *Flow Designer*, haciendo doble click en la pantalla que queremos editar.
2. Pulsando en la solapa *Screen Design* que está a la izquierda de la solapa *Flow Design* y seleccionando la pantalla (*screen*) deseada en *Edited Screen* (figura 18).

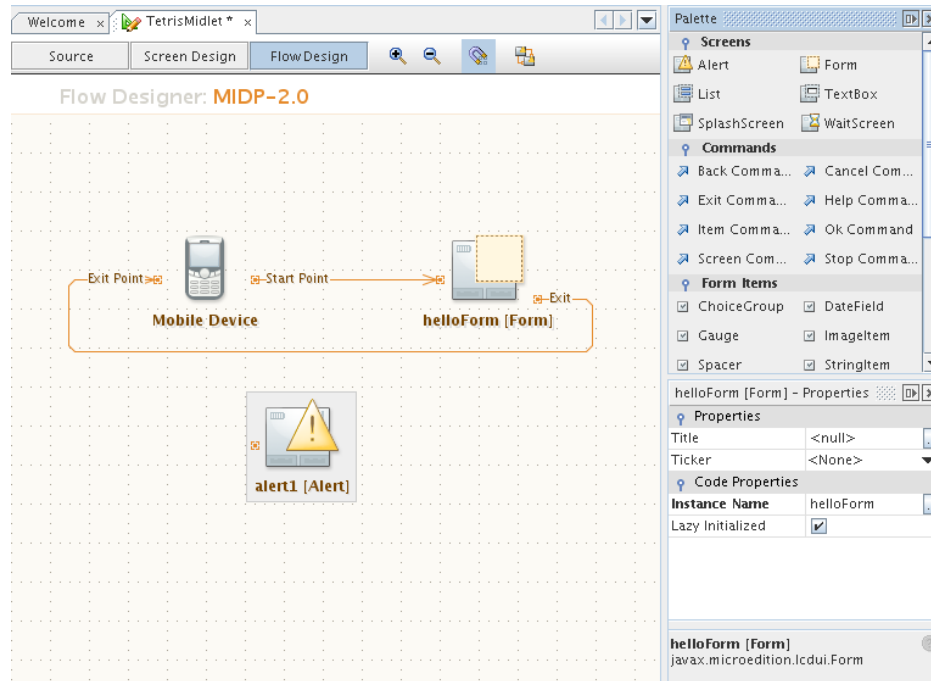


Figura 13: Creación de un screen alert

Comenzaremos modificando el único componente que hay ahora mismo en el Form *helloForm*, que es el *String Item* cuya etiqueta (*Label*) es *Hello* y cuyo *String* asociado es *Hello, World!*. Para ello seleccionamos el *String Item* con el botón izquierdo del ratón y cambiamos el *Label* y *String*, que aparecen en la ventana de propiedades (derecha del IDE), por los valores *Hola* y *Bienvenido al juego del Tetris*.

## 5.1. Añadir nuevos componentes en una pantalla

El *screen designer* funciona como otros editores visuales de componentes. O sea, se crea el interfaz de usuario arrastrando componentes desde la paleta de componentes hasta la pantalla que se esté editando.

A continuación crearemos un *ChoiceGroup* en el que incluiremos tres *ChoiceItem* de tipo *EXCLUSIVE*. Este *ChoiceGroup* nos servirá para elegir la operación que queremos hacer una vez que el programa comience: Jugar, Opciones o Ver Records. Para ello haremos los siguientes pasos:

- Arrastra un *ChoiceGroup* desde la paleta de componentes hacia el form *helloForm* (debemos tener visualizado *helloForm* en el *screen designer*).
- Pincha con el botón izquierdo del ratón para seleccionar el *ChoiceGroup* y modificar la etiqueta que se ha puesto por defecto (*choiceGroup1*). Asigne la etiqueta **Menú**.
- En la ventana de propiedades del *ChoiceGroup* (a la derecha del IDE) cambia la propiedad *Type* del valor *MULTIPLE* al valor *EXCLUSIVE*. Esto hará que el usuario sólo pueda seleccionar uno de los items del *ChoiceGroup*.

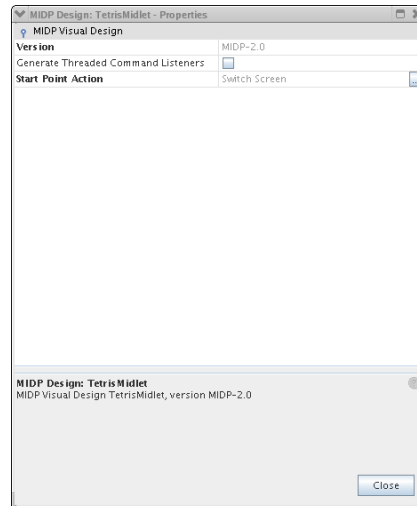


Figura 14: Acceso a las propiedades de diseño del midlet

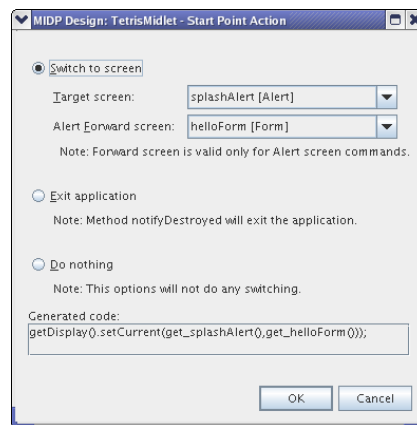


Figura 15: Acceso a las propiedades de diseño del midlet

- Añade ahora tres *Choice Elements* arrastrándalos desde la paleta de componentes hacia el *ChoiceGroup*.
- Cambia la propiedad *String* de los anteriores *Choice Elements* por los valores *Jugar*, *Opciones* y *Ver records* respectivamente.

## 5.2. Añadir componentes no visuales

El *screen designer* puede usarse también con componentes no visuales. Por ejemplo, para añadir un comando, arrastra el tipo de comando desde la paleta de componentes hacia la pantalla. La pantalla *helloForm* ya dispone de un comando *exit*. Por tanto añadiremos un comando *Ok*:

- Arrastra un comando *Ok* desde la paleta de componentes hacia el screen de *helloForm*. El screen no cambia, pero un nuevo comando *OkCommand1* aparece en el inspector de componentes dentro del apartado *Assigned Commands* (figura 19).
- En el inspector de componentes, pinchar con el botón derecho del ratón sobre el nuevo comando *okCommand1* y selecciona la opción

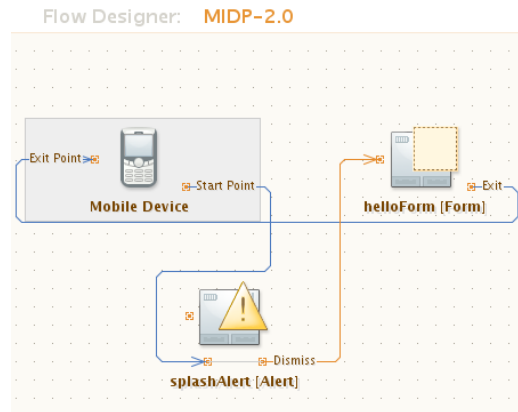


Figura 16: La alerta está ahora insertada en el flujo

*Edit* en el menú que aparece. Aparecerá una *Action* (figura 20) que permite controlar la acción que se hará cuando el usuario pulse ese comando, por ejemplo a qué screen se cambia. Por ahora no cambiamos nada y pulsamos el botón *Ok*.

Pinchamos la solapa *Source* para visualizar el código fuente y buscamos el siguiente método:

```

/** Called by the system to indicate that a command has been invoked on a
 * particular displayable.
 * @param command the Command that ws invoked
 * @param displayable the Displayable on which the command was invoked
 */
public void commandAction(Command command, Displayable displayable) {
    // Insert global pre-action code here
    if (displayable == helloForm) {
        if (command == exitCommand) {
            // Insert pre-action code here
            exitMIDlet();
            // Insert post-action code here
        } else if (command == okCommand1) {
            // Insert pre-action code here
            // Do nothing
            // Insert post-action code here
        }
    }
    // Insert global post-action code here
}
    
```

Ahora añadiremos el código siguiente después del comentario

```
\\Insert pre-action code here
```

que hay tras la sentencia

```
else if (command == okCommand1) {
```

```
System.out.println("He pulsado el botón Ok");
```

El método quedará según aparece en la figura 21. Este nuevo código hará que cuando el usuario pulse el botón *Ok* del dispositivo móvil, estando en el screen *helloForm* aparezca el mensaje *He pulsado el botón Ok* por la salida estándar.

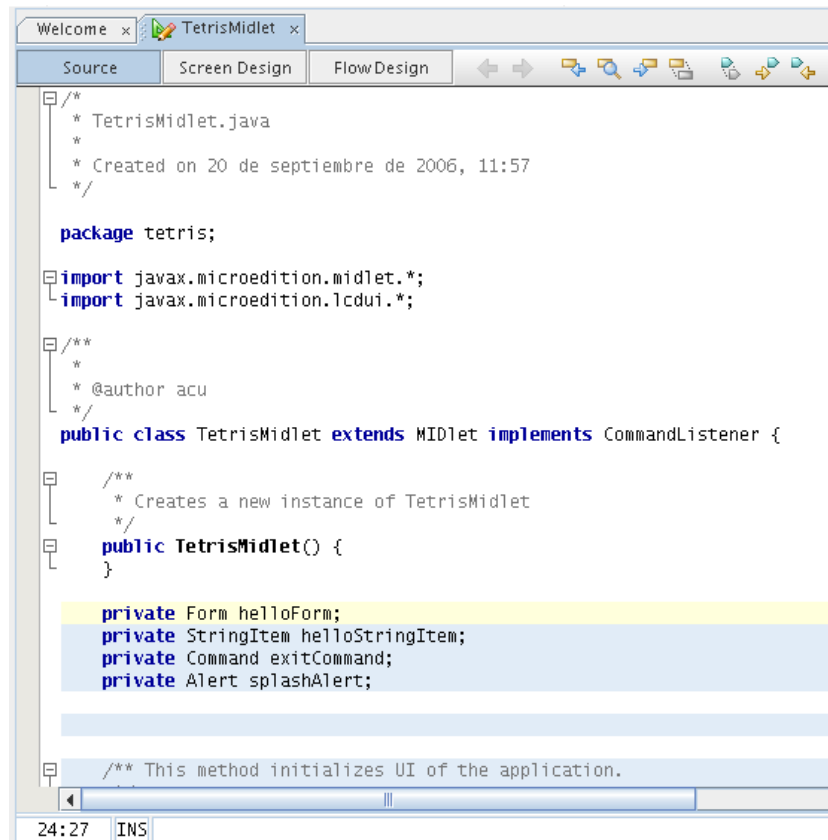


Figura 17: Ventana de edición de código para Tetrismidlet

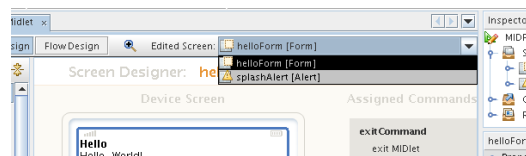


Figura 18: Seleccionar la pantalla a editar

### 5.3. Inserción de imágenes en el proyecto

Se pueden añadir imágenes a nuestro proyecto que serán guardados dentro del fichero Jar del Midlet. Las imágenes pueden usarse por ejemplo para asignarlas a un *image item* o como icono del Midlet. Veamos los pasos que hay que dar para utilizar una imagen como icono de nuestra aplicación:

- Usando el sistema de ficheros copia el fichero **Fuentes/piezaTetris.png** (o cualquier otra imagen en formato png) al directorio *src* de tu proyecto.
- Dentro de Netbeans, arrastra un componente *Image* de la sección *Resources* de la paleta de componentes, hacia el *Screen Designer*, pero fuera de los límites del screen visualizado en este momento. Esto hará que aparezca un nuevo item imagen *image1* en el inspector de componentes dentro de la sección *Resources*.

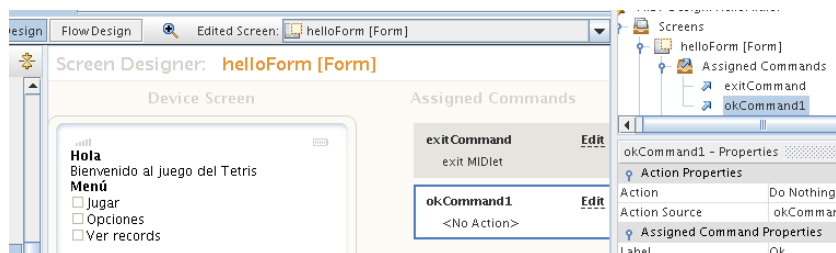


Figura 19: Añadir un nuevo comando

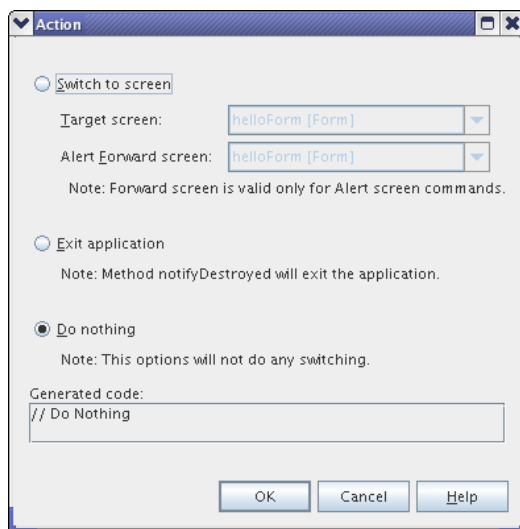



Figura 20: Ventana de acción de un comando

- Pincha sobre el anterior item en el inspector de componentes con el botón derecho del ratón y selecciona *Properties*.
- En la ventana de propiedades de la imagen pincha en el botón etiquetado con ... de la propiedad *Resource path* para definir el path donde se encuentra la imagen.
- Selecciona el fichero de imagen `/piezaTetris.png` (figura 22).
- Para poner esta imagen como icono del proyecto debes abrir la misma ventana que utilizamos en la sección 2.3 para cambiar el nombre del midlet (figura 10).

## 6. Creación de un Canvas como panel del juego

El panel donde se dibujará el tablero del juego y la figura que cae actualmente será implementado creando una subclase de la clase *Canvas*. Este panel lo construiremos creando una subclase de la clase *Canvas* desde el IDE de Netbeans. Esta subclase no puede crear de forma visual como lo hemos hecho con la alerta de la sección 3.2. Para crear el Canvas daremos los siguientes pasos:



```

    }
    // Insert global post-action code here
}

/** Called by the system to indicate that a command has been invoked on a particular
 * @param command the Command that ws invoked
 * @param displayable the Displayable on which the command was invoked
 */
public void commandAction(Command command, Displayable displayable) {
    // Insert global pre-action code here
    if (displayable == helloForm) {
        if (command == exitCommand) {
            // Insert pre-action code here
            exitMidlet();
            // Insert post-action code here
        } else if (command == okCommand1) {
            // Insert pre-action code here
            System.out.println("He pulsado el botón Ok");
            // Do nothing
            // Insert post-action code here
        }
    }
    // Insert global post-action code here
}
    
```

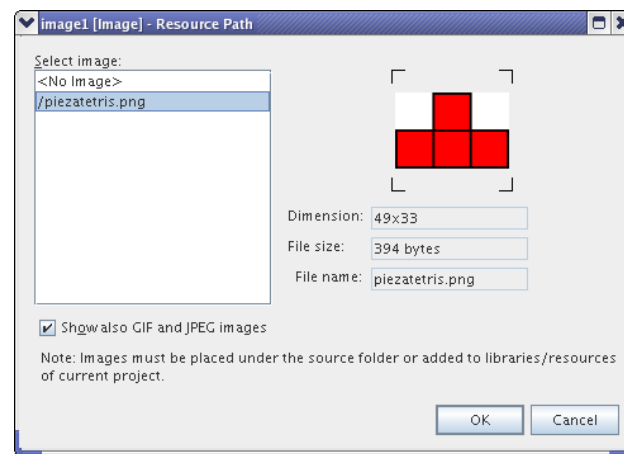
 Figura 21: Inserción de *pre-action user code*


Figura 22: Selección del fichero para la imagen

- Seleccionar **Menú File** → **New File**.
- En la ventana *New File* que aparece seleccionar la categoría *MIDP* y tipo de fichero (*File Types*) *MIDP Canvas* (figura 23) y pulsa el botón *Next*.
- Asigna *MiCanvas* como nombre de la nueva clase MIDP (*MIDP Class Name*) e introduce *tetris* en *Package* si no estaba ya introducido (figura 24). Pulsa el botón *Finish*. Al hacerlo se habrá incluido *MiCanvas* como nueva clase de nuestro proyecto. Esta nueva clase aparece editada en el IDE de Netbeans (figura 25). Puede observarse que el código fuente de esta nueva clase ya contiene un constructor y otros métodos como *paint(Graphics g)*, *keyPressed(int keyCode)*, etc.
- Incluye el dato miembro *TetrisMidlet tetrisMidlet* en la clase *MiCanvas*. Este dato miembro será utilizado para poder acceder al objeto *TetrisMidlet* donde se ha incluido el *Canvas* y así poder acceder a algunos métodos de la clase *TetrisMidlet* que necesitaremos utilizar desde *MiCanvas*.
- Modifica el constructor por defecto de *MiCanvas* para que tenga un

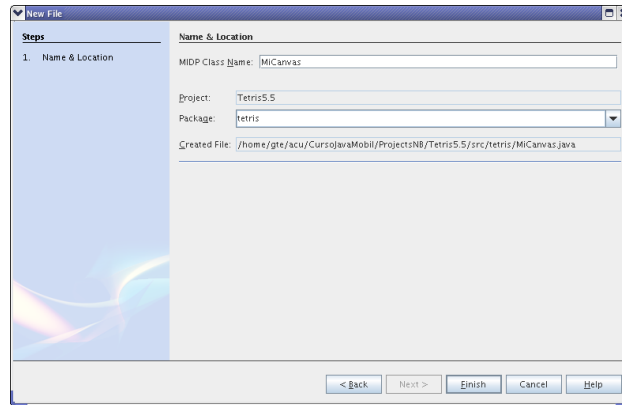


Figura 23: Creación del Canvas MIDP

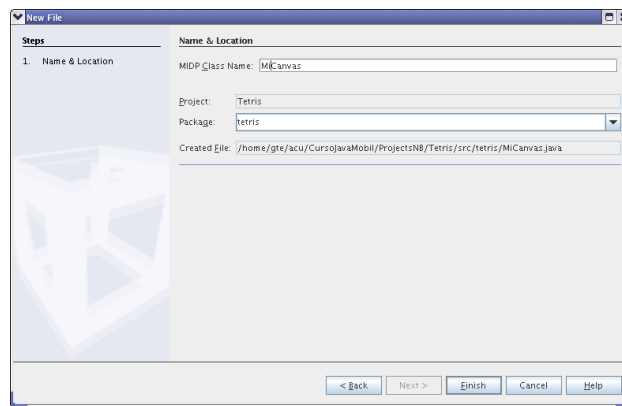


Figura 24: Asignando el nombre al Canvas MIDP

parámetro *TetrisMidlet t*. El constructor utilizará esta variable para inicializar el dato miembro introducido en el paso anterior (*tetrisMidlet*). El constructor quedaría como sigue:

```
public class MiCanvas extends Canvas implements CommandListener {
    /**
     * Referencia al TetrisMidlet donde se incluye este MiCanvas
     */
    TetrisMidlet tetrisMidlet;

    /**
     * constructor
     */
    public MiCanvas(TetrisMidlet t) {
        try {
            // Set up this canvas to listen
            // to command events
            setCommandListener(this);
            // Add the Exit command
            addCommand(new Command("Exit", Command.EXIT, 1));
            tetrisMidlet = t;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- Abre el código fuente de la clase *TetrisMidlet* pinchando en el IDE, la solapa *TetrisMidlet* y luego la solapa *Source*.

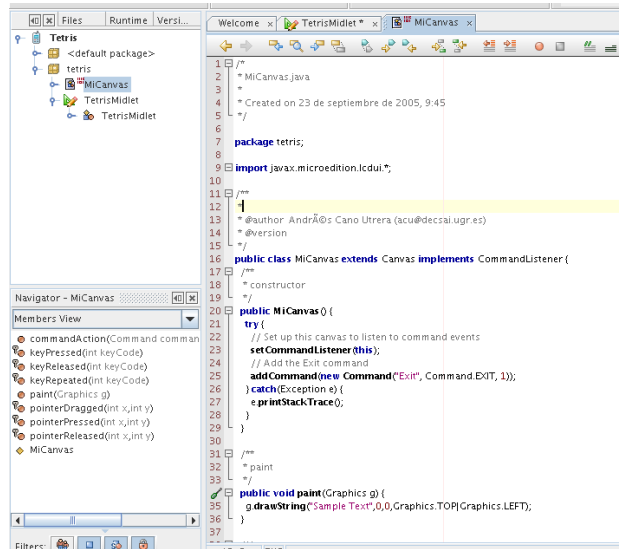


Figura 25: Clase MiCanvas dentro del IDE



Figura 26: Edición de la acción del comando okCommand1

- Incluye el dato miembro *MiCanvas miCanvas* en la clase *TetrisMidlet*, inmediatamente antes del constructor.
- Dentro del constructor de *TetrisMidlet* asigna a la variable *miCanvas* un nuevo objeto de la clase *MiCanvas*:

```
public class TetrisMidlet extends MIDlet implements
    javax.microedition.lcdui.CommandListener {
    MiCanvas miCanvas;

    /**
     * Creates a new instance of TetrisMidlet
     */
    public TetrisMidlet() {
        miCanvas = new MiCanvas(this);
    }
}
```

- Modifica la acción para el comando *okCommand1* introduciendo la siguiente línea de código en el método *commandAction()* de *TetrisMidlet.java*, para conseguir que el flujo del programa cambie al Canvas cuando el usuario pulse el comando Ok, sea cual sea la opción seleccionada en el *Choice Group*.

```
javax.microedition.lcdui.Display.getDisplay(this).setCurrent(miCanvas);
```

- Ejecuta el proyecto para comprobar el funcionamiento.

## 7. Asignación de acciones a los comandos del Canvas y helloForm

A continuación modificaremos la acción del comando Ok de *helloForm* que hemos editado en la sección anterior para conseguir que sólo nos cambiemos al Canvas cuando esté seleccionada la opción *Jugar* en el *Choice Group* de *helloForm*. También le asignaremos una acción al comando *Exit* del Canvas.

- Edita de nuevo la acción del comando *okCommand1* de *helloForm* eliminando las dos sentencias que introducimos anteriormente:

```
System.out.println("He pulsado el botón Ok");
javax.microedition.lcdui.Display.getDisplay(this).setCurrent(miCanvas);
```

sustituyéndolas por el siguiente trozo de código para que sólo se muestre el el canvas cuando tenemos seleccionado la opción *Jugar* en el *ChoiceGroup*:

```
if(choiceGroup1.getSelectedIndex()==0){
    System.out.println("Ha seleccionado Jugar");
    javax.microedition.lcdui.Display.getDisplay(this).setCurrent(miCanvas);
} else if(choiceGroup1.getSelectedIndex()==1){
    System.out.println("Ha seleccionado Opciones");
} else if(choiceGroup1.getSelectedIndex()==2){
    System.out.println("Ha seleccionado Ver records");
}
```

- Añade el dato miembro *private Command exitCommand* a la clase *MiCanvas* que nos permitirá asociar un comando a esta clase.
- Modifica el constructor de *MiCanvas* para que la variable *exitCommand* referencie al objeto de la clase *Command*.incluido en el canvas:

```
/**
 * constructor
 */
public MiCanvas(TetrisMidlet t) {
    try {
        // Set up this canvas to listen to command events
        setCommandListener(this);
        // Add the Exit command
        exitCommand = new Command("Exit", Command.EXIT, 1);
        addCommand(exitCommand);
        tetrisMidlet = t;
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

- Modifica el método *commandAction(Command command, Displayable displayable)* de *MiCanvas* para que el programa regrese al screen *helloForm* cuando el usuario pulse el comando *Exit* desde el Canvas.

```

/**
 * Called when action should be handled
 */
public void commandAction(Command command, Displayable displayable) {
    if (command == exitCommand) {
        javax.microedition.lcdui.Display.getDisplay(
            tetrisMidlet).setCurrent(tetrisMidlet.helloForm);
    }
}
    
```

- Ejecuta el proyecto para comprobar que funcionan bien los comandos en *helloForm* y en el Canvas.

## 8. Pintar el tablero de juego en el Canvas

En esta sección incluiremos el código necesario para que aparezca dibujado el tablero de juego. Para ello realiza los siguientes pasos:

- Copia los ficheros **Fuentes/Rejilla.java**, **Fuentes/Figura.java** y **Fuentes/Elemento.java** en el directorio *src/tetris* de tu proyecto usando el sistema de ficheros. La clase *Rejilla* es básicamente una clase que representa una matriz bidimensional donde cada celda puede contener los valores VACIA, BLOQUE o PIEZA. La clase *Figura* representa la figura que cae actualmente en el juego. La clase *Elemento* es utilizada por la clase *Figura* para representar cada una de las celdas ocupadas por la Figura.
- Añade el dato miembro *private int anchoCelda* a la clase *MiCanvas*. Esta variable representa el número de píxeles de cada celda de la Rejilla cuando se dibuje en el Canvas. La variable *anchoCelda* se inicializa con  $-1$ . Más adelante introduciremos código en el método *paint(Graphics g)* para calcular su valor correcto que se adapte al dispositivo en el que se ejecuta el programa.

```

/**
 * Referencia al TetrisMidlet donde se incluye este MiCanvas
 */
TetrisMidlet tetrisMidlet;

/**
 * Número de píxeles del ancho y alto de cada celda de
 * este tablero de juego
 */
private int anchoCelda = -1;
    
```

- Añade las variables *Rejilla rejilla* y *Figura figura* a la clase *TetrisMidlet*. Estos datos miembros nos servirán para acceder al objeto *Rejilla* y *Figura* respectivamente desde los métodos de *TetrisMidlet*.

```

public class TetrisMidlet extends MIDlet implements
    javax.microedition.lcdui.CommandListener {
    MiCanvas miCanvas;
    Rejilla rejilla;
    Figura figura=null;
}
    
```

- Añade al constructor de *TetrisMidlet* el código para que se cree una nueva *Rejilla* con tamaño 12 celdas de ancho por 22 de alto:

```
public TetrisMidlet() {
    miCanvas = new MiCanvas(this);
    rejilla = new Rejilla(12,22);
}
```

- Añade los métodos *getRejilla()* y *getFigura()* a *TetrisMidlet* que permitan obtener una referencia a la *Rejilla* del juego, y la *Figura* que cae actualmente respectivamente.

```
/**
 * Obtiene una referencia a la Rejilla del juego
 * @return una referencia a la Rejilla del juego
 */
public Rejilla getRejilla(){
    return rejilla;
}

/**
 * Obtiene una referencia a la Figura que cae actualmente en el juego
 * @return una referencia a la Figura actual
 */
public Figura getFigura(){
    return figura;
}
```

- Añade los métodos *dibujaRejilla(Graphics g)* y *dibujaFigura(Figura fig, Graphics g)* a la clase *MiCanvas*

```
/**
 * Dibuja los bordes del tablero de juego y las celdas ocupadas por trozos
 * de figura ya colocadas en el tablero
 * @param g el Graphics donde se dibujará
 */
public void dibujaRejilla(Graphics g){
    int i,j;
    Rejilla rejilla=tetrisMidlet.getRejilla();

    int xoffset=(getWidth()-rejilla.getAnchura()*anchoCelda)/2;
    for(i=0;i<rejilla.getAnchura();i++){
        for(j=0;j<rejilla.getAltura();j++){
            if(rejilla.getTipoCelda(i,j) == Rejilla.BLOQUE){
                g.setColor(0,0,0);
                g.drawRect(xoffset+i*anchoCelda,j*anchoCelda,anchoCelda,
                    anchoCelda);
            } else if(rejilla.getTipoCelda(i,j) == Rejilla.PIEZA){
                g.setColor(255,255,0);
                g.fillRect(xoffset+i*anchoCelda,j*anchoCelda,anchoCelda,
                    anchoCelda);
                g.setColor(255,0,0);
                g.drawRect(xoffset+i*anchoCelda,j*anchoCelda,anchoCelda,
                    anchoCelda);
            }
        }
    }
}

/**
 * Dibuja la Figura fig en el Graphics g pasado como parámetro
 * (normalmente el asociado a este Canvas)
 * @param fig la Figura a dibujar
 * @param g el Graphics donde se dibujará
 */
void dibujaFigura(Figura fig,Graphics g){
    if (fig!=null){
        Elemento elemento;
        Rejilla rejilla=tetrisMidlet.getRejilla();
    }
}
```

```

int xoffset=(getWidth()-rejilla.getAnchura()*anchoCelda)/2+
    fig.getXOrigen()*anchoCelda;
int yoffset=fig.getYOrigen()*anchoCelda;
for(int i=0;i<fig.getNElements();i++){
    elemento=fig.getElementAt(i);
    g.setColor(255,255,0);
    g.fillRect(xoffset+elemento.getColumna()*anchoCelda,
        yoffset+elemento.getFila()*anchoCelda, anchoCelda,
        anchoCelda);
    g.setColor(255,0,0);
    g.drawRect(xoffset+elemento.getColumna()*anchoCelda,
        yoffset+elemento.getFila()*anchoCelda, anchoCelda,
        anchoCelda);
}
}
}

```

- Modifica el método *paint(Graphics g)* de *MiCanvas* de la siguiente forma para que se dibujen la Rejilla y Figura actual:

```

/**
 * paint
 */
public void paint(Graphics g) {
    //g.drawString("Sample Text",0,0,Graphics.TOP|Graphics.LEFT);

    if(anchoCelda==-1){
        anchoCelda=Math.min(getWidth()/tetrisMidlet.getRejilla().getAnchura(),
            (getHeight()-10)/tetrisMidlet.getRejilla().getAltura());
    }
    g.setColor(255,255,255);
    g.fillRect(0,0,getWidth(),getHeight());
    g.setColor(0,0,0);
    g.translate(0,12);
    dibujaRejilla(g);
    dibujaFigura(tetrisMidlet.getFigura(),g);
    g.translate(0,-12);
}

```

- Ejecuta el proyecto de nuevo. Ahora al seleccionar la opción *Jugar* en *helloForm* y pulsar el botón Ok, en el emulador debe aparecer algo como lo de la figura 27. Puedes comprobar que aun no aparece la figura.

Lo único que nos falta es hacer que aparezca la figura en el tablero de juego, y que ésta vaya cayendo hacia abajo. Esto lo haremos en la siguiente sección.

## 9. Movimiento de la figura del tetris

Para conseguir que una figura se mueva continuamente hacia abajo hasta que choque contra otra figura o bien contra la parte inferior del tablero, debemos construir una hebra que se encargue de ello.

- Añadir los métodos *nuevaFigura()* e *inicializaJuego()* a la clase *TetrisMidlet*:

```

/**
 * Obtiene una nueva figura cuyo tipo es seleccionado de forma aleatoria
 */
public void nuevaFigura(){

```



Figura 27: Programa tras dibujar la Rejilla

```

    figura = Figura.nuevaFigura();
}
/**
 * Deja VACIA todas las celdas de la Rejilla, la inicializa
 * de nuevo. Además genera una nueva Figura de tipo aleatorio
 */
public void inicializaJuego(){
    rejilla.initRejilla();
    nuevaFigura();
}
    
```

- Añade el método *MiCanvas getCanvas()* a la clase *TetrisMidlet*:

```

/**
 * Obtiene una referencia al Canvas (panel donde se dibuja) del juego
 * @return una referencia al Canvas del juego
 */
public MiCanvas getCanvas(){
    return miCanvas;
}
    
```

- Copia la clase **Fuentes/Mueve.java** al directorio *src/tetris* del proyecto. Esta clase es la que implementa la hebra que se encarga de mover la pieza que cae actualmente en el juego.
- Añade el dato miembro *Mueve mueve* a la clase *TetrisMidlet* y crea con él un objeto de la clase *Mueve* dentro del constructor de *TetrisMidlet* de la siguiente forma:

```

/**
 * Creates a new instance of TetrisMidlet
 */
public TetrisMidlet() {
    miCanvas = new MiCanvas(this);
    rejilla = new Rejilla(12,22);
    mueve=new Mueve(this, 2);
}
    
```

- Edita nuevamente la acción del comando Ok de *helloForm* para que se inicialice el juego mediante la llamada al método *inicializaJuego()* (deja vacía la rejilla y genera una figura de tipo aleatorio) de *TetrisMidlet* y empiece a mover la figura actual mediante la llamada al método *reanudar()* del objeto *mueve* (clase *Mueve*) que hemos creado en el constructor de la clase *TetrisMidlet*.

```

if(choiceGroup1.getSelectedIndex()==0){
    System.out.println("Ha seleccionado Jugar");
    javax.microedition.lcdui.Display.getDisplay(this).setCurrent(miCanvas);
    inicializaJuego();
    mueve.reanudar();
} else if(choiceGroup1.getSelectedIndex()==1){
    System.out.println("Ha seleccionado Opciones");
} else if(choiceGroup1.getSelectedIndex()==2){
    System.out.println("Ha seleccionado Ver records");
}
    
```

- Ejecuta el proyecto. Puedes comprobar que en el tablero ya aparece una figura que se va moviendo hacia abajo pero que todavía no puede controlarse con las teclas para moverla a izquierda o derecha, o para rotarla.

## 10. Movimiento a izquierda y derecha y rotación de la figura

En esta sección añadiremos el código necesario para que la figura que cae actualmente se mueva a izquierda y derecha con el teclado (teclas izquierda y derecha). Además utilizaremos la tecla de flecha hacia arriba para que la figura rote en sentido contrario a las agujas del reloj. La tecla de flecha hacia abajo se utilizará para que la figura avance más rápidamente hacia abajo. Los pasos que hay que realizar son los siguientes:

- Modificar el método *keyPressed(int keyCode)* de la clase *MiCanvas* para que quede de la siguiente forma:

```

protected void keyPressed(int keyCode) {
    if (keyCode == getKeyCode(LEFT)){
        if(!tetrisMidlet.getRejilla().seChoca(tetrisMidlet.getFigura(),
            Figura.IZQUIERDA)){
            tetrisMidlet.getFigura().mueve(Figura.IZQUIERDA);
            if(tetrisMidlet.getCanvas()!=null)
                tetrisMidlet.getCanvas().repaint();
        }
    } else if (keyCode == getKeyCode(RIGHT)){
        if(!tetrisMidlet.getRejilla().seChoca(tetrisMidlet.getFigura(),
            Figura.DERECHA)){
            tetrisMidlet.getFigura().mueve(Figura.DERECHA);
            if(tetrisMidlet.getCanvas()!=null)
                tetrisMidlet.getCanvas().repaint();
        }
    } else if (keyCode == getKeyCode(UP)){
        tetrisMidlet.getFigura().rotar(tetrisMidlet.getRejilla());
        if(tetrisMidlet.getCanvas()!=null)
            tetrisMidlet.getCanvas().repaint();
    } else if (keyCode == getKeyCode(DOWN)){
        if(!tetrisMidlet.getRejilla().seChoca(tetrisMidlet.getFigura(),
            Figura.ABAJO)){
            tetrisMidlet.getFigura().mueve(Figura.ABAJO);
        }
    }
}
    
```

```

        if(tetrisMidlet.getCanvas()!=null)
            tetrisMidlet.getCanvas().repaint();
    }
}

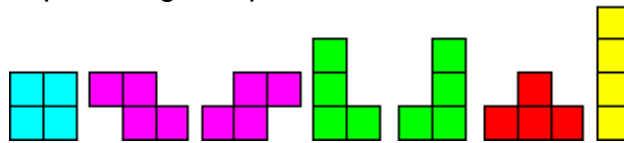
```

- Ejecuta el proyecto de nuevo y podrás comprobar que el juego ya está terminado.

## 11. Posibles mejoras del juego

Algunas posibilidades de ampliación del juego que se dejan como ejercicio son las siguientes:

- Permitir que el juego sea detenido al pulsar una determinada tecla.
- Hacer que se visualice la siguiente Figura que va a aparecer.
- Que cada tipo de Figura aparezca en un color diferente.



- Mostrar el tiempo que ha pasado desde que se comenzó el juego.
- Hacer uso de niveles. Se podría hacer que cuando se completen 20 líneas se pasaría a un nivel superior en el que las piezas caen a una mayor velocidad. Esto es relativamente sencillo pues el constructor de la clase *Mueve* ya está preparado para pasarle un parámetro que indica el nivel.
- Contar los puntos conseguidos. Por ejemplo 5 puntos por línea.