Programación para Ingeniería Telemática

Curso 2018-19

Práctica 4: Interfaces Gráficas de Usuario.

1. Objetivos de aprendizaje. Estructura de la práctica.

Objetivos

- Presentar ejemplos básicos de interfaces gráficas de usuario.
- Presentar el mecanismo de manejo de eventos en interfaces gráficas: eventos asociados a botones y al teclado.
- Definir representaciones gráficas (vistas) para los elementos para el juego.
- Ser capaces de representar gráficamente los elementos del juego a medida que éste se desarrolla (primera parte).
- Ser capaces de utilizar los eventos de teclado para controlar el movimiento del personaje principal (primera parte).

En esta práctica definiremos los gráficos del juego. Primero se presentarán una serie de ejemplos de interfaces gráficas en los que se mostrará cómo definir menús y botones, cómo organizar los elementos de la interfaz, cómo capturar eventos del teclado y de un temporizador y, finalmente, cómo dibujar sobre el panel de una ventana. A continuación se pedirá que los estudiantes apliquen estos conocimientos para realizar un editor de escenarios de juego.

Contenidos de la memoria

Esta memoria se estructura en cinco secciones con los siguientes contenidos:

Sección 1: Objetivos y contenido de la práctica.

Sección 2: Los paquetes `common`, `game`, `guis` y `views`. En esta sección se presenta el código de partida que tendrá que usarse para realizar esta práctica. Aparecen dos nuevos paquetes, `guis` y `views`, que se irán ampliando en sucesivas prácticas.

Sección 3: Enunciado de los ejercicios a realizar. Estos ejercicios están pensados para facilitar la realización de la práctica entregable correspondiente al bloque de la asignatura dedicado a la programación orientada a objetos. Además, los ejemplos y ejercicios que se vayan mostrando en las prácticas y o bien son semejantes a los que pueden pedirse en los exámenes de la signatura o bien constituyen un paso previo para ser capaces de realizarlos.

Antes de hacer los ejercicios es aconsejable tener una visión de conjunto, por ello **se recomienda leer detenidamente el boletín de prácticas y entender el código suministrado antes de abordar la resolución de los ejercicios.**

2. Los paquetes common, game, guis y views y la carpeta images

2.1. El paquete common

El paquete `common` contiene las clases `KeyBoard` y `FileUtilities` completamente implementadas y la interfaz `IToJSONObject`.

2.2. El paquete game

Contiene todas las interfaces y clases definidas en las prácticas anteriores (figura 1).

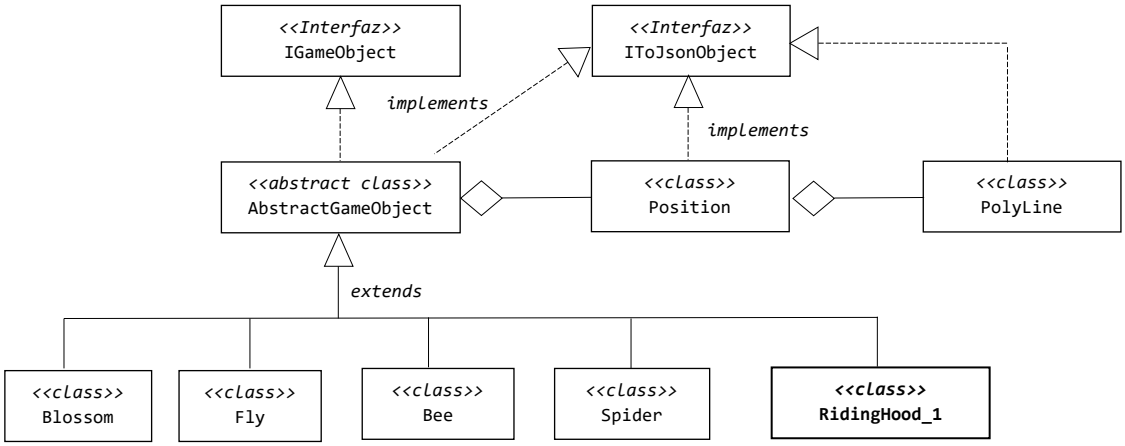


Figura 1: Tipos de datos

2.3. El paquete guis y la carpeta images

En el paquete `guis` se incluyen los ejemplos listados en la Tabla 1. Estos ejemplos van mostrando progresivamente algunas de las características más importantes de las clases e interfaces que vamos a utilizar para diseñar e implementar la interfaz gráfica del juego. También proporcionan ejemplos de distintos tipos de manejadores de eventos. Los detalles de los ejemplos pueden consultarse en el código fuente. Todas las clases contienen un método `main`.

La carpeta `images` contiene ficheros jpg que se utilizarán en los ejemplos. Para que los ejemplos funcionen correctamente debe copiar esta carpeta en el directorio del proyecto con la siguiente ruta:

proyecto`/src/main/resources/`**`images`**

Tabla 1: Ejemplos del paquete `guis`	
Clase	**Descripción**
`UnaVentana`	Crea una ventana vacía con un nombre.
`MenusYBotones`	Crea una ventana vacía con un nombre. Añade una *barra de menú* que a su vez incluye dos *menús*, uno de ellos con submenús y dos botones.
`MenusYBotonesConManejadores`	Igual que la anterior, pero con ***manejadores de eventos*** asociados a las entradas de menú y a los botones.
`PanelesYLayouts`	Se crea una ventana y se le añaden 5 paneles y una serie de botones para cambiar la disposición de los paneles en la ventana (cambio de ***layout***). En este ejemplo se muestran ejemplos de ***manejadores de eventos*** y de ***uso de manejadores de layouts*** típicos para controlar la disposición de los contenidos en un contenedor.
`PanelConImagen`	Crea una ventana visible y le añade un panel modelado como una clase interna. En este panel se reproduce una imagen que se toma del fichero: *proyecto*`/src/main/resources/images/PUERTO.jpg` La carpeta `images` puede obtenerse en el aula virtual y debe copiarse en el directorio que acaba de indicarse.
`PanelConImagenYScroll`	Semejante al anterior, pero incluyendo un **Scroll Panel** con dos barras de deslizamiento.
`PanelConScrollYEventosDeRaton`	Es una versión del ejemplo anterior con algunos elementos adicionales. Se ha añadido una etiqueta en la parte inferior de la ventana de la aplicación. El panel de la imagen **maneja los eventos de ratón que se producen sobre él**, escribiendo en la etiqueta el nombre del método que se ejecuta en respuesta al evento de ratón y las coordenadas en las que se ha producido el evento.
`PanelConCuadriculaYTeclado`	Este ejemplo introduce algunos elementos especialmente interesantes de cara a la programación del juego: – Manejo de **eventos de teclado,** en concreto de las teclas de flecha (***arrow keys***). – Se define una **clase derivada de Panel en la que se redefine el método de "pintado del panel"** para dibujar una cuadrícula y un cuadrado que se desplaza a través de ella de acuerdo con las teclas de flecha pulsadas. – Se incluye una etiqueta en la que se muestran las coordenadas actuales en las que se encuentra el cuadrado.
`PanelConCuadriculaYTimer`	Se introduce el **manejo de los ticks de un temporizador**. El cuadrado avanza en cada tick del timer una dirección en la dirección indicada por la última tecla de flecha pulsada.
`CampoDeTextoYDialogo`	Ventana con una etiqueta y un campo de texto. – Maneja los **eventos del campo de texto**. – Muestra un **cuadro de diálogo** que informa de los errores del operador.

2.4. El paquete `views`

El paquete `views` contiene las interfaces y clases que modelan e implementan las vistas de los elementos del juego (figura 2). Los detalles pueden consultarse en el propio código fuente.

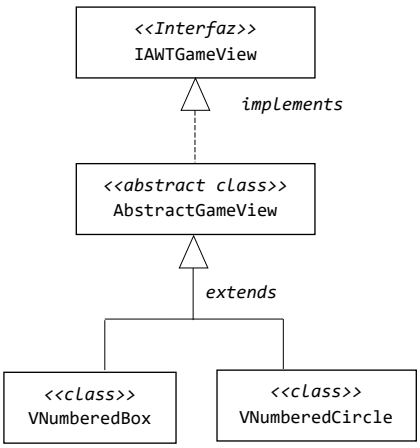


Figura 2: Modelado de vistas

- La interfaz **`IAWTGameView`** define un único método, que deben implementar todas las vistas.

  ```
  void draw(Graphics g);
  ```

  Cuando **`draw`** es invocado sobre una vista, dicha vista se dibuja a sí misma utilizando para ello el objeto g de la clase Graphics que se le pasa como argumento.

  La clase **`java.awt.Graphics`** puede verse como un pincel para dibujar sobre una superficie, con métodos para fijar el color y el grosor del pincel y para dibujar líneas y figuras geométricas.

- La clase abstracta **`AbstractGameObject`** proporciona una implementación parcial por defecto para los objetos `IAWTGameView`. Esta clase es abstracta, ya que no implementa el método draw, sin embargo, proporciona dos variables de instancia y un constructor para fijarlas.

  ```
  AbstractGameView(IGameObject obj, int length) throws Exception
  ```

  Para construir la vista se necesita el elemento de juego al que van a representar (`obj`) y el tamaño con el que se debe representar (`length`). La vista puede obtener de `obj` el tipo, la posición y estado del objeto que debe representar[1].

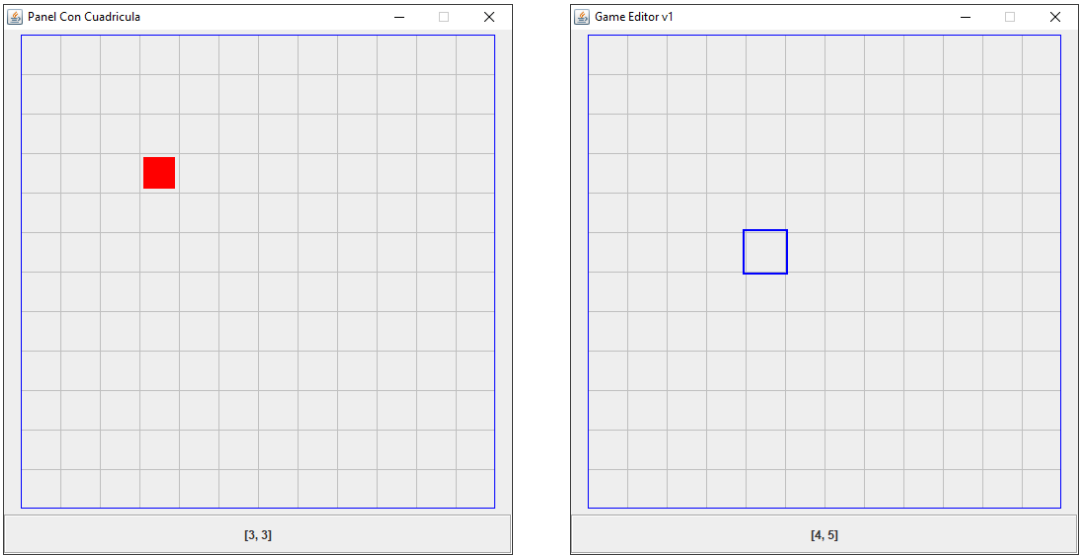
- Las clases concretas **`VNumberedBox`** y **`VNumberedCircle`** son dos subclases muy sencillas de `AbstractGameView`, dibujan un cuadrado y un circulo respectivamente (figura 4.d), a los que se puede añadir un pequeño texto y fijar su color (véanse constructores en código fuente).

[1] En realidad no hace falta pasarle a la vista todos los datos del objeto. Se podría haber definido un tipo de datos más sencillo para pasárselo a la vista, pero se ha preferido no complicar el diseño.

3. Ejercicios a realizar

Ejercicio 1:

Vamos a partir del ejemplo `PanelConCuadriculaYTeclado` y vamos a modificarlo ligeramente para que en lugar de pintar un cuadrado rojo un poco más pequeño que el cuadro de la cuadrícula (figura 3 a)), pinte el contorno de un cuadrado azul un poco más grande que el cuadro de la cuadrícula (figura 3 b)).



a) Ejemplo `guis.PanelConCuadriculaYTeclado` b) Ejemplo `game.GameEditor`

Figura 3: Transformación de ejemplo `PanelConCuadriculaYTeclado` en un editor de elementos de juego. Parte I.

Para ello, copie el fichero `PanelConCuadriculaYTeclado.java` en el paquete `game` y cámbiele el nombre a `GameEditor_1.java`. Realice los siguientes cambios y adiciones que se indican a continuación (y otros que estime necesarios para el buen funcionamiento del programa):

(1) Cambie el nombre de la ventana a Game Editor v1.

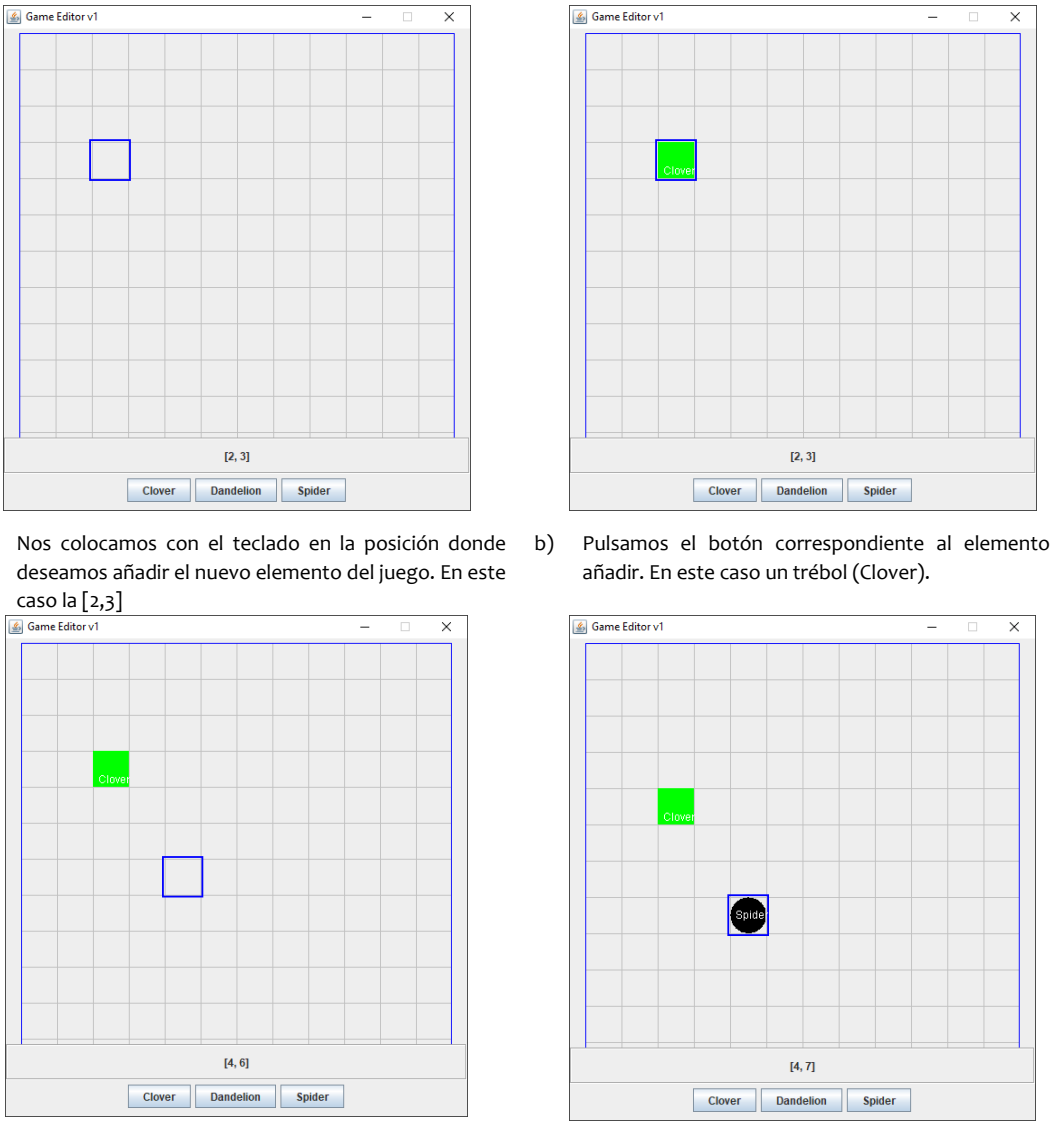
(2) Implemente un método privado drawSquare que pinte un cuadrado azul en una cuadro dado de la cuadrícula e invóquelo en el método paintComponent cuando sea conveniente.

(3) Elimine las líneas de código que pintan el cuadrado rojo.

Ejercicio 2:

Vamos a implementar ahora una nueva clase GameEditor_2, en la que vamos a añadir un panel con tres botones (Figura 4) para crear un elemento del juego en una posición dada y representarlo en la cuadrícula.

El funcionamiento sería el siguiente: nos desplazamos con el teclado a la posición de la cuadrícula donde queremos añadir un elemento del juego (figuras 4.a y 4.c) y pulsamos el botón correspondiente al elemento a añadir (figuras 4.b y 4.d). Al pulsar el botón el nuevo elemento se añade al juego y se representa en la cuadrícula.



a) Nos colocamos con el teclado en la posición donde deseamos añadir el nuevo elemento del juego. En este caso la [2,3]

b) Pulsamos el botón correspondiente al elemento a añadir. En este caso un trébol (Clover).

c) Nos desplazamos con el teclado a la siguiente posición donde deseamos añadir el nuevo elemento del juego. En este caso la [4,6]

d) Pulsamos el botón correspondiente al elemento a añadir. En este caso una araña (Spider).

Figura 4: Transformación de ejemplo `PanelConCuadriculaYTeclado` en un editor de elementos de juego. Parte II.

Vamos a dividir este problema en dos partes. La primera correspondiente a este ejercicio y la segunda al siguiente.

En este ejercicio vamos a incluir el objeto seleccionado en la aplicación, pero sin representarlo todavía gráficamente. En lugar de ello escribiremos una traza en consola indicando todos los elementos incluidos en el juego cada vez que se inserte uno nuevo.

Para ello, parta del código de `GameEditor_1.java` y realice los siguientes cambios y adiciones (y otros que estime necesarios para el buen funcionamiento del programa):

(1) Cree en `game` una nueva clase `GameEditor_2.java` y copie en ella los contenidos de `GameEditor_1.java`

(2) Cambie los nombres de los constructores para evitar errores de compilación.

(3) Cambie el nombre de la ventana a Game Editor v2.

(4) Defina las variables de instancia que va a necesitar para implementar la funcionalidad adicional: los tres botones (`JButton`), un array de elementos de juego, donde ir añadiendo los nuevos elementos, y un contador de elementos de juego añadidos (no puede sobrepasarse el tamaño del array).

(5) Añada los botones a la ventana del juego.

(6) Defina los manejadores de eventos correspondientes a los botones y realice la suscripción correspondiente (todos son iguales, solo variará el tipo de elemento a añadir). Para ello defina un manejador de eventos anónimo para cada botón. En cada manejador deberá:

- Comprobar que hay sitio en el array para añadir un nuevo elemento.
- Crear el nuevo elemento y añadirlo al array.
- Imprimir por consola la lista de todos los elementos de juego añadidos hasta el momento. Para esto último implemente un método privado `printGameItems` e invóquelo en el manejador.
- La última línea del código del manejador debe ser **`requestFocusInWindow()`**, en caso contrario dejarán de capturarse los eventos de Teclado que necesita para moverse por la cuadrícula.

Pruebe cada manejador una vez que esté hecho (no espere a tenerlos todos).

Ejercicio 3:

En este ejercicio vamos a actualizar los gráficos de la aplicación para que cada vez que se inserte un nuevo elemento del juego se represente en la cuadrícula.

Para ello, ampliaremos el código de `GameEditor_2.java` de la siguiente manera:

(1) En la clase interna `Canvas` añada un array de objetos de juego (`IGameObject objects[]`) y el siguiente un método público[2]:

```
public void drawGameItems(IGameObject [] objs){
    this.objects = objs;        // actualiza los objetos a representar por el canvas
    repaint();                  // fuerza el repintado del canvas.
}
```

Invoque dicho método en los manejadores de evento de los botones donde corresponda (añada una traza al método para comprobar que se llama cuando corresponde).

(2) Nuevamente, en la clase interna `Canvas` implemente método privado `void drawGameItems(Graphics g)` que pinte una vista, subclase de `AbstractGameObject`, de cada elemento del juego contenido en `objects[]` de acuerdo con su tipo. Invoque dicho método en `paintComponent`.

[2] En realidad, este método no es necesario, ya que la clase interna `Canvas` puede acceder al array definido en `GameEditor_2`, pero en la implementación del juego se pedirá que `Canvas` sea una clase independiente de la ventana en la que se utilice.

Ejercicio 4:

Vamos a implementar ahora una nueva clase GameEditor_3, en la que vamos a añadir un menú con dos entradas (Figura 5) para salvar los elementos creados en un fichero de texto o cargar elementos salvados previamente en un fichero de texto.

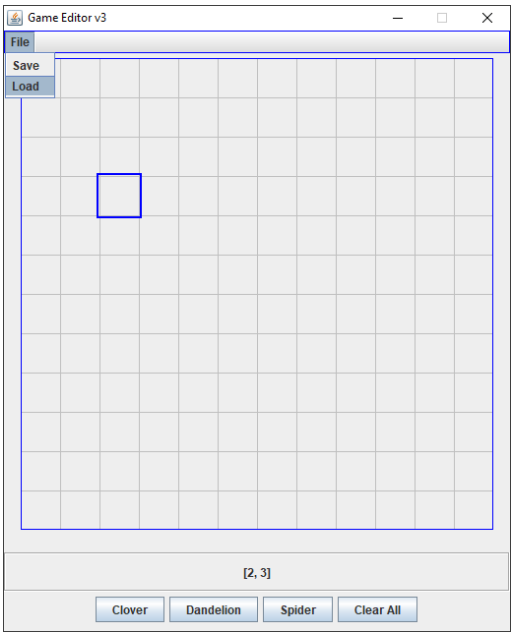


Figura 5: Transformación de ejemplo `PanelConCuadriculaYTeclado` en un editor de elementos de juego. Parte III.

Para ello, parta del código de `GameEditor_2.java` y realice los siguientes cambios y adiciones (y otros que estime necesarios para el buen funcionamiento del programa):

(1) Cree en `game` una nueva clase `GameEditor_3.java` y copie en ella los contenidos de `GameEditor_1.java`

(2) Cambie los nombres de los constructores para evitar errores de compilación.

(3) Cambie el nombre de la ventana a Game Editor v3.

(4) Añada un botón para borrar todos los elementos previamente incluidos en la aplicación (como aparece en la figura #).

(5) Declare las siguientes variables de instancia:

```
String path = "src/main/resources/games/game.txt";
JMenuBar menuBar;
JMenu menuFile;
JMenuItem itSave, itLoad;
```

Y extienda la aplicación de forma que aparezca la barra de menú con el menú menuFile con sus dos entradas, correspondientes a salvar los elementos del juego en el fichero path o a cargarlos.

(7) Defina los manejadores de eventos correspondientes a las entradas de menú y realice la suscripción correspondiente. Para ello defina un manejador de eventos anónimo para cada entrada de menú.

- Cuando se seleccione la entrada salvar se guardarán los elementos del juego en el fichero path en formato JSON.
- Cuando se seleccione la entrada cargar se leerán los elementos del juego guardados en el fichero path en formato JSON, se almacenarán en el array de objetos y se mostrarán en la cuadrícula.