Práctica 5:

Implementación de una interfaz gráfica de usuario básica siguiendo el patrón MVC

Añadiendo una interfaz gráfica a nuestro juego

Una vez finalizado el modelo de Irrgarten es el momento de añadirle una interfaz gráfica de usuario utilizando el patrón Modelo-Vista-Controlador.

La interfaz de texto que se entregó en la práctica 3 ya sigue este esquema; por tanto, el controlador lo podrás reutilizar sin apenas modificaciones.

Esta práctica se realizará únicamente sobre el proyecto Java del juego.

Añadiendo una interfaz común a las interfaces de usuario

Define una interfaz llamada *UI* común a la interfaz de usuario de texto y la gráfica que crearás en esta práctica. Créala en el mismo paquete en el que incluiste la interfaz de usuario de texto.

```
public Directions nextMove();
public void showGame(GameState gameState);
```

Fíjate que ya dispones de esos métodos en la interfaz de usuario de texto y en este caso solo tendrás que indicar que esa clase realiza la interfaz *UI*.

En el controlador cambia el tipo de las variables que hacen referencia a la interfaz de usuario para que usen objetos del tipo *UI*. Ahora el controlador aceptará también la interfaz gráfica como interfaz de usuario ya que esta también realizará la nueva interfaz.

Nota: cuando en las siguientes secciones se indica que debes crear clases derivadas de *JFrame* o *JDialog* debes realizar esta operación en Netbeans desde File → New File → Swing GUI Forms → JFrame/JDialog Form.

Creando la ventana principal del juego

La interfaz de usuario del juego constará, por simplicidad, de una nueva clase que herede de *JFrame* y que realice la interfaz indicada en el apartado anterior (*UI*). Al crear esta clase, verás en el código fuente que se ha creado automáticamente que la clase tiene un método *main*. Borra este método *main*, pues el programa principal que se va a usar será uno similar al que creaste en la práctica 3 para jugar con la interfaz de texto; solo que en esta ocasión se instanciará la interfaz gráfica que estás creando.

Otro cambio que harás en el código que se ha creado de manera automática para esta clase es añadir, en el constructor, después de la instrucción *initComponents()* la instrucción *setVisible(true)* para que la ventana de la aplicación sea visible.

En ese objeto JFrame se crearan instancias de JTextArea para mostrar el estado del laberinto, los

monstruos, los jugadores y el log. Para el resto de información, como si hay un ganador y el nombre del jugador actual se pueden utilizar etiquetas (*JLabel*) o campos de texto (*JtextField*).

El método *showGame* simplemente leerá todos los campos del objeto de tipo *GameState* y los mostrará en los elementos gráficos indicados en el párrafo anterior.

Al final del método *showGame* llama al método *repaint* para asegurar que los elementos gráficos se actualizan con la nueva información.

Añadiendo la entrada de la dirección de movimiento del personaje

Para recoger la dirección en la que el jugador humano desea que se mueva el jugador se creará una nueva clase (denominada *Cursors*) que herede de *JDialog* (cuadro de diálogo). Esta tendrá un botón para representar cada dirección de movimiento y una pareja atributo-consultor que permita obtener la dirección elegida. Aquí también verás que, de manera automática, la clase también tiene un método *main*, elimínalo igualmente.

Dicho consultor, *getDirection()* tiene que hacer visible el cuadro de diálogo antes de devolver el atributo *direction*. Así, cuando se le solicite una dirección, se mostrará el cuadro de diálogo (*setVisible(true)*) quedando a la espera de que el usuario pulse algún botón de dirección para devolver el atributo *direction*.

La programación de cada uno de los botones de dirección es igual de fácil: cuando se pulse sobre cada uno de los botones, se guardará la dirección elegida en el atributo y se ejecutará el método *dispose* para que deje de estar visible el cuadro de diálogo (esto provocará que *getDirection()* continúe y devuelva el atributo que se acaba de actualizar.

En la clase de la interfaz de usuario principal tendremos que hacer algunos cambios más: se añadirá un atributo privado de la clase *Cursors*, y se inicializará en el constructor con la instrucción *new Cursors(this,true);* el primer parámetro es una referencia al propio *JFrame*, que es la ventana que lanza el cuadro de diálogo; y el segundo parámetro activa el modo *modal*, para que el cuadro de diálogo no ceda el foco mientras esté abierto.

Después, en el método *nextMove*, que es el encargado de solicitarle al usuario una dirección y devolverla, se añadirá la instrucción *return cursors.getDirection();* es decir, delegamos en el cuadro de diálogo la comunicación con el usuario.

Resumiendo, cuando desde el controlador se ejecute el método *nextMove*, este delegará en la instancia de *Cursors* para obtener la dirección (*cursors.getDirection()*). La ejecución de *getDirection* hará que el cuadro de diálogo se haga visible a si mismo y espere a que el usuario pulse sobre el botón que representa la dirección correspondiente. La acción asociada a cualquiera de los botones que representan la dirección de movimiento deseada es almacenar la dirección seleccionada y ejecutar el método *dispose* del cuadro de diálogo para que deje de estar visible, devuelva el valor elegido de dirección y vuelva a permitir interactuar con la vista principal.

Comprobando lo aprendido hasta ahora

Una vez finalizada esta práctica y las anteriores deberías saber y entender los siguientes conceptos:

- Los indicados en el guion anterior.
- Saber implementar el patrón Modelo-Vista-Controlador y conocer la responsabilidad de cada módulo
- Saber cómo se comunican dichos módulos entre sí y con el usuario.