

# MEMORIA PRÁCTICA MP

Por Iván Martín Gómez y Markos Aguirre Elorza

## Introducción

La práctica que hemos realizado es el del juego clásico “Snake” donde una serpiente controlada por el jugador come frutas para crecer y ganar más puntos. En el caso de que la serpiente choque contra su propio cuerpo o contra las paredes el jugador perderá.

Hemos destacado en esta memoria los aspectos de implementación más relevantes así como fragmentos del diagrama UML que faciliten su entendimiento. Para consultar el diagrama completo se debe dirigir al repositorio de GitHub, ahí también se podrán consultar diferentes enfoques del modelo, vista y controlador.

Los requisitos (vienen escritas implícitamente), instrucciones de uso etc. los podrá encontrar en esta memoria con la ayuda del siguiente índice:

## Índice

- Instrucciones de Uso (pág. 2)
- Hebras (pág. 3,4)
  - Clase Thread, Interface Runnable
- MVC (pág. 5, 6, 7)
  - Vista
  - Controlador
  - Modelo
    - Generación de eventos almacenados en una cola
- Observer (pág. 8)
- Eventos (pág. 8)
- Funcionalidades destacables (pág. 9)

# Instrucciones de Uso

Para empezar, en el “Pantalla Principal”, como se puede apreciar en la foto, consta de un gran recuadro que va a ser el escenario del juego (donde aparecen la serpiente y los frutos). Este recuadro consta de varios cuadrados que funcionan como píxeles, se iluminan, apaga o cambian de color dependiendo de la situación. De esta forma, se pueden escribir palabras (como el Game Over del final), dibujar los movimientos de la serpiente o mostrar donde está el fruto. Cabe destacar que los límites de esta pantalla son los límites por los que la serpiente puede moverse.

Más abajo, hay una barra que siempre que se inicie el programa se inicializa con la frase “Introduzca nombre del jugador, el color y pulsar iniciar”. Es aquí donde el usuario debe escribir su nombre y pulsar enter, una vez escrito, el nombre aparecerá en el panel que está inmediatamente debajo y que le antecede la palabra “Nombre”.

Durante la partida, la serpiente se va a ir moviendo y la posición en la que se encuentra (que es el cuadrado donde se encuentra la cabeza de la serpiente” va a ser expresada en coordenadas en el “Pantalla Principal” donde aparece “Eje Y” y “Eje X”.

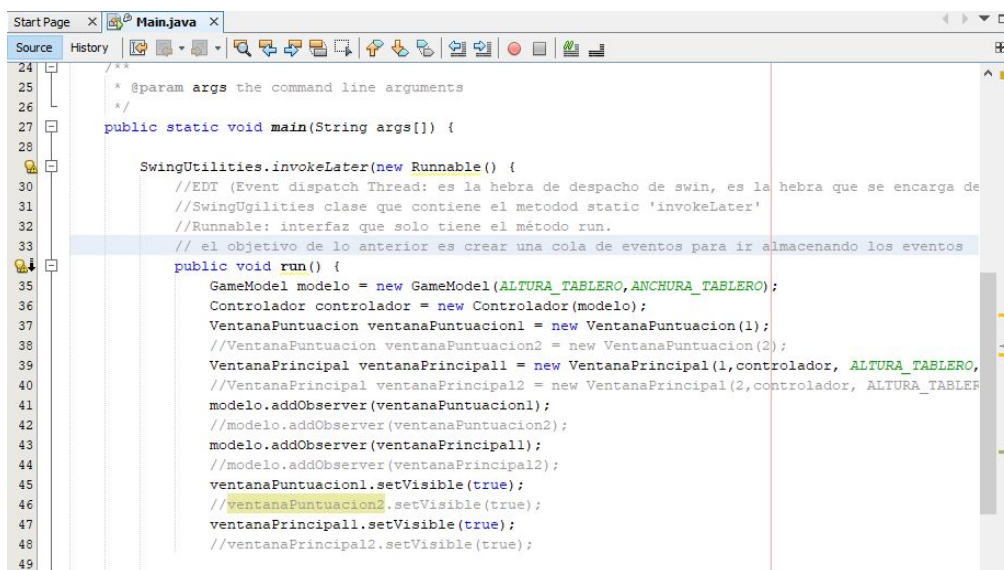
El color de la serpiente no es fijo y se puede modificar mediante el recuadro donde pone “Color”, al clicar ahí se puede elegir el color de la serpiente dentro de una variedad predeterminada de colores del desplegable.

Cuando el jugador haya introducido el nombre y el color tiene que pulsar el botón “Iniciar”. Este botón hará aparecer a la serpiente y la fruta en el gran recuadro y la serpiente se podrá mover mediante los botones del interfaz o sino mediante los del teclado.

La función del “Pantalla Secundaria” es mostrar la puntuación del jugador que es el número de frutas que ha comido.

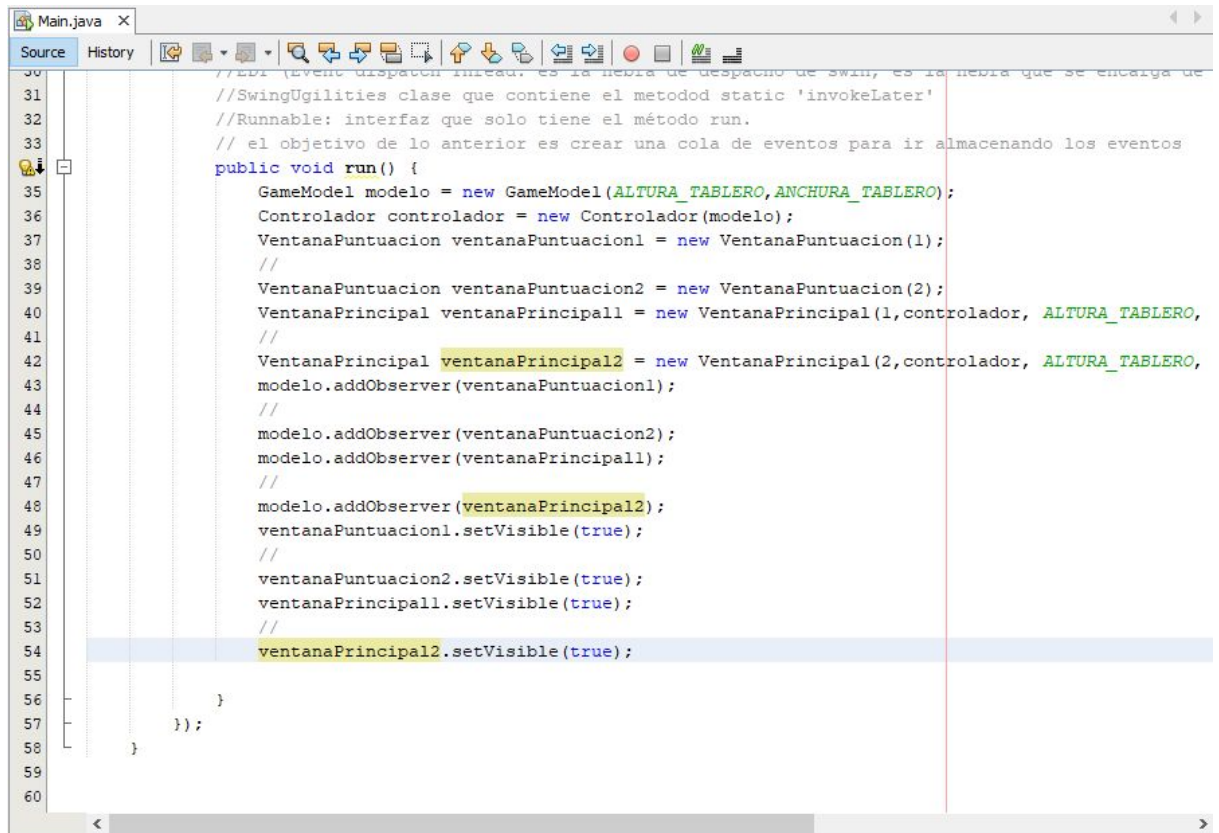
Para seleccionar el número de Jugadores hay que hacer una pequeña modificación en el código. En la clase Main se deben de quitar los comentarios y escribirlos como código tal y como aparece en las siguientes fotos:

## UN JUGADOR



```
24  /**
25   * @param args the command line arguments
26   */
27  public static void main(String args[]) {
28
29      SwingUtilities.invokeLater(new Runnable() {
30          //EDI (Event dispatch Thread: es la hebra de despacho de swin, es la hebra que se encarga de
31          //SwingUtilities clase que contiene el metodos static 'invokeLater'
32          //Runnable: interfaz que solo tiene el método run.
33          // el objetivo de lo anterior es crear una cola de eventos para ir almacenando los eventos
34
35          public void run() {
36              GameModel modelo = new GameModel(ALTURA_TABLERO,ANCHURA_TABLERO);
37              Controlador controlador = new Controlador(modelo);
38              VentanaPuntuacion ventanaPuntuacion1 = new VentanaPuntuacion(1);
39              //VentanaPuntuacion ventanaPuntuacion2 = new VentanaPuntuacion(2);
40              VentanaPrincipal ventanaPrincipal1 = new VentanaPrincipal(1,controlador, ALTURA_TABLERO,
41              //VentanaPrincipal ventanaPrincipal2 = new VentanaPrincipal(2,controlador, ALTURA_TABLERO,
42              modelo.addObserver(ventanaPuntuacion1);
43              //modelo.addObserver(ventanaPuntuacion2);
44              modelo.addObserver(ventanaPrincipal1);
45              //modelo.addObserver(ventanaPrincipal2);
46              ventanaPuntuacion1.setVisible(true);
47              //ventanaPuntuacion2.setVisible(true);
48              ventanaPrincipal1.setVisible(true);
49              //ventanaPrincipal2.setVisible(true);
```

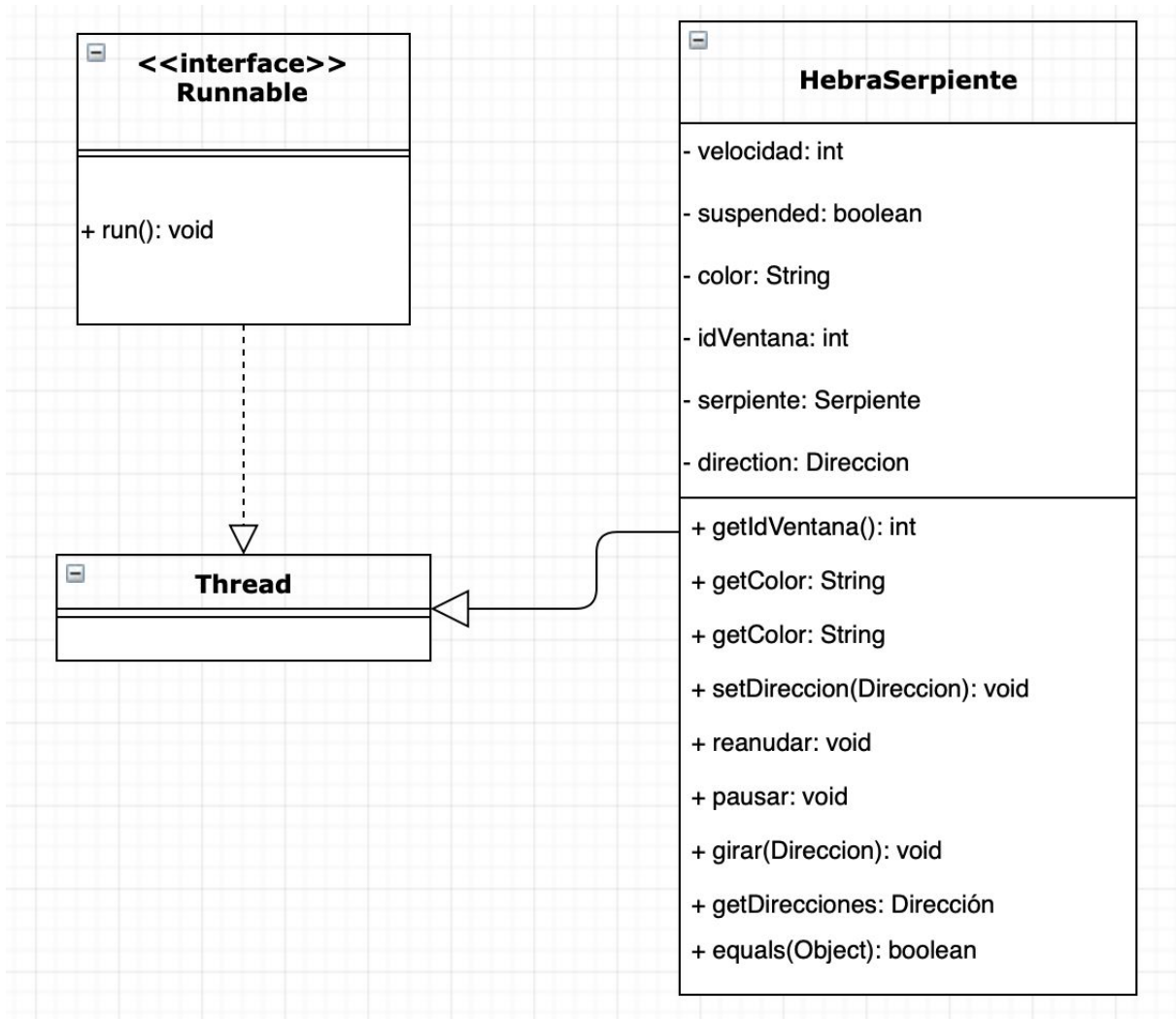
## DOS JUGADORES



```
30 //EJ1 (Event dispatch thread, es la hebra de despacho de swin, es la hebra que se encarga de
31 //SwingUtilities clase que contiene el metodo static 'invokeLater'
32 //Runnable: interfaz que solo tiene el método run.
33 // el objetivo de lo anterior es crear una cola de eventos para ir almacenando los eventos
34 public void run() {
35     GameModel modelo = new GameModel(ALTURA_TABLERO, ANCHURA_TABLERO);
36     Controlador controlador = new Controlador(modelo);
37     VentanaPuntuacion ventanaPuntuacion1 = new VentanaPuntuacion(1);
38     //
39     VentanaPuntuacion ventanaPuntuacion2 = new VentanaPuntuacion(2);
40     VentanaPrincipal ventanaPrincipal1 = new VentanaPrincipal(1, controlador, ALTURA_TABLERO,
41     //
42     VentanaPrincipal ventanaPrincipal2 = new VentanaPrincipal(2, controlador, ALTURA_TABLERO,
43     modelo.addObserver(ventanaPuntuacion1);
44     //
45     modelo.addObserver(ventanaPuntuacion2);
46     modelo.addObserver(ventanaPrincipal1);
47     //
48     modelo.addObserver(ventanaPrincipal2);
49     ventanaPuntuacion1.setVisible(true);
50     //
51     ventanaPuntuacion2.setVisible(true);
52     ventanaPrincipal1.setVisible(true);
53     //
54     ventanaPrincipal2.setVisible(true);
55
56 }
57 });
58 }
59
60
```

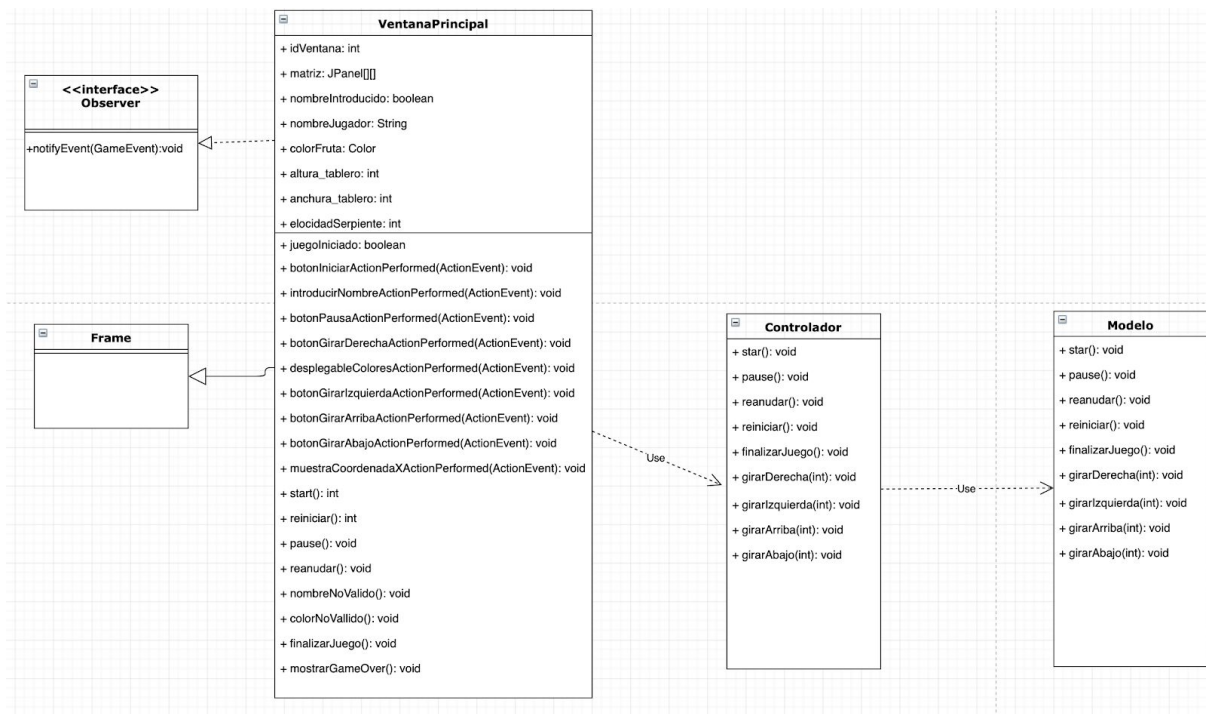
# Hebras

Para desarrollar la funcionalidad de la serpiente hemos hecho uso de hebras.



# Patrón MVC

Hemos utilizado el modelo MVC (Modelo-Vista-Controlador) que nos permite aislar las modificaciones realizadas en las vistas con respecto al modelo y viceversa.

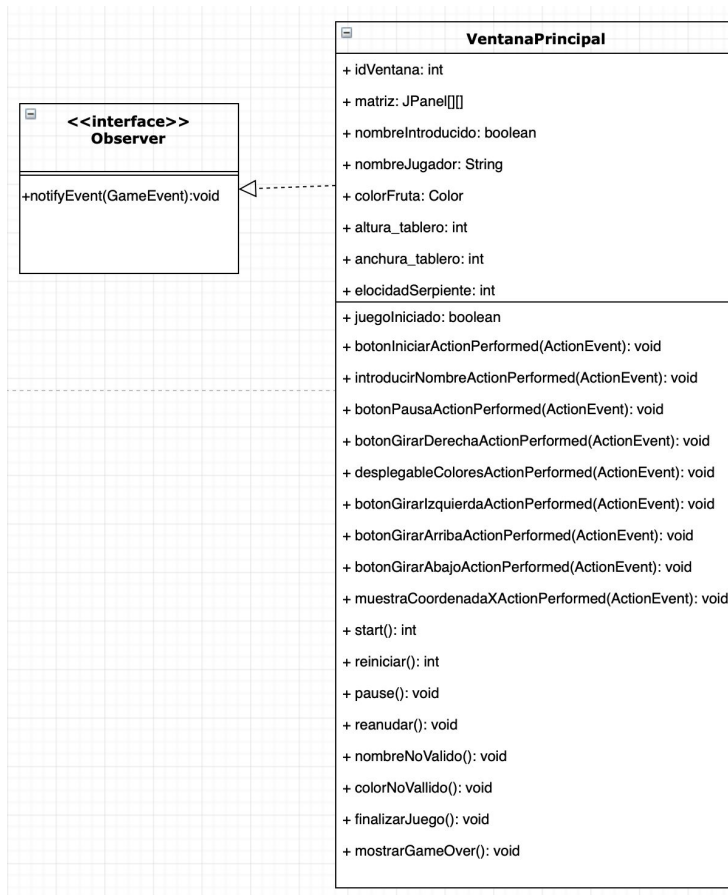


## Vista

The screenshot shows the main window of the game. It features a large empty rectangular area at the top for the game board. Below this, there is a text input field with the placeholder text "Introduzca nombre del jugador, el color y pulsar iniciar". Underneath the input field, there are labels for "Nombre", "Eje Y", and "Eje X", each followed by a small input field. To the right of these fields are three directional buttons: a left arrow, a down arrow, and a right arrow. Below these, there is a "Color" label and a dropdown menu currently showing "Green". At the bottom of the window, there are two buttons: "Iniciar" and "Pause".

The screenshot shows a section of the game interface for displaying the score. It has a light gray background. At the top, the word "Puntuación" is centered. Below it, there are two empty rectangular boxes, one above the other, intended for displaying the score.

The screenshot shows a close-up of the color selection dropdown menu. The label "Color" is to the left of the dropdown. The dropdown menu is open, showing a list of color options: "Green" (which is selected and highlighted with a blue bar and a checkmark), "Red", "Black", and "Blue".



Las clases **VentanaPrincipal** y **VentanaSecundaria** tendrán un objeto de tipo **Controlador**.

Las clases “**Ventana Principal**” y “**Ventana Secundaria**” serán dos clases que implementen a la interfaz **observer**.

Dentro de la vista encontramos dos pantallas de usuario: la “**Pantalla Principal**” que es donde se desarrolla el juego y el “**Pantalla Secundaria**” que es donde aparecen las puntuaciones.

## Controlador

El controlador es la clase encargada en poner el contacto a las vistas con el modelo. Dentro de la clase **Controlador** existirá un objeto de tipo **modelo**.

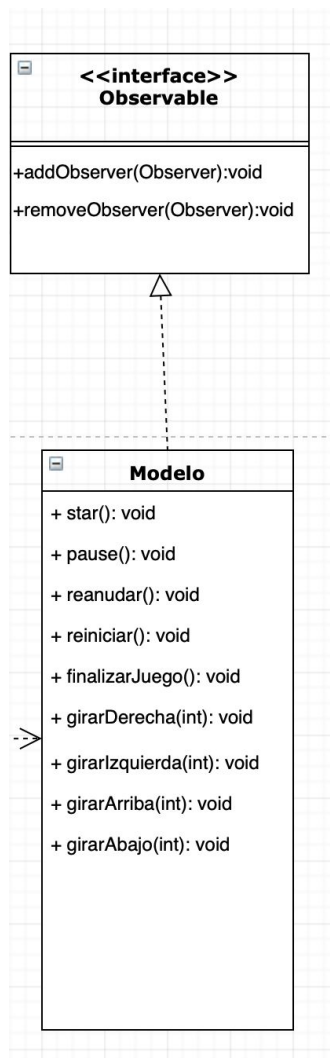
Notar que el sentido de la comunicación será desde las vistas hacia el modelo pero no en sentido contrario.

# Modelo

El modelo será la clase encargada de implementar toda la lógica del juego, es decir, todas las normas del juego y la gestión de la hebras así como la opción multijugador correrá por cuenta del modelo.

La comunicación entre modelo y las vistas se realizará mediante el uso del patrón observer. En este caso los observadores serán las vistas y el observado será el modelo.

La clase GameModel será una clase que implemente a la interfaz Observable.



# Patrón Observer

Este patrón nos permitirá poner en contacto a la clase modelo con cada una de las vistas. De esta forma tendremos una independencia total ante cambios realizados en las vistas con respecto al modelo y viceversa

Este patrón hace uso de eventos para establecer la comunicación, es decir la clase modelo genera eventos que “notificará” a cada una de los observadores (vistas).

## Eventos

Tendremos eventos de diferente tipo:

START,  
PAUSE,  
STOP,  
MOVER\_SERPIENTE,  
NUEVA\_FRUTA,  
REANUDAR,  
REINICIAR,  
FINALIZAR\_JUEGO,  
NOMBRE\_NO\_VALIDO,  
COLOR\_NO\_VALIDO

Cada uno de los eventos contendrá la información necesaria para informar a las vistas de los cambios que debe realizar.



# Funcionalidades extraordinarias

- El juego no podrá ser inicializado antes de introducir el nombre de los jugadores.
- En el modo multijugador cada jugador deberá escoger un nombre diferente al del otro jugador, si un color ya está elegido y el otro jugador intenta seleccionar el mismo color la interfaz de usuario mostrará un mensaje de error y además se eliminarán los colores ya usados en el desplegable.
- Al finalizar el juego se mostrarán en ambas pantallas de los jugadores el mensaje de Game Over.
- El juego se podrá pausar desde cualquiera de los terminales de los jugadores

