PRÁCTICA FINAL PROGRAMACIÓN II

“BALLS SIMULATION”

Profesor: Miquel Mascaró Oliver
Asignatura: Programació II (21707)
Curso: 2018-2019
Autores: Adrián Bennasar Polzin, Felix Lluís Aguilar Ferrer

ENLACE VIDEO:

Introducción

La solución consiste en una interfaz gráfica en la cual hay un menú y un panel donde se visualizan una serie de círculos, los cuales se mueve por este. El menú se compone de dos "CheckBox" que sirven para determinar el comportamiento de los círculos y de un "JTextField" para elegir la cantidad de círculos que se quieren visualizar en el panel.

El movimiento de los círculos se puede dar de dos formas:

- Los círculos se mueven con una aceleración constante.
- Los círculos se mueven con una aceleración que depende de la posición del ratón.

La interacción con los límites del panel se puede dar de dos maneras:

- Los círculos pueden atravesar las paredes y aparecer en el otro extremo del panel.
- Los círculos pueden chocar con los límites del panel invirtiendo así su velocidad.

Diseño

El diseño que hemos utilizado para montar la solución ha sido dividir el problema en diferentes conceptos. Cada uno contiene sus propios atributos y métodos que realizan las acciones dentro de su ámbito. Para ello se han creado 4 conceptos:

- Window (ventana).
- CirclePanel (Panel de círculos).
- Circle (Círculo).
- Vector.

Window:

Esta clase es la principal, su objetivo principal es invocar las funciones de otras clases en las que se ejecutan operaciones a más bajo nivel. Además en esta se definen los componentes que forman el menú lateral ("TextField", "CheckBox", etc.) y se construye el comportamiento de la ventana. Finalmente se definen los escuchadores de los eventos del menú.

Todos los "Strings" del menú y el nombre de la ventana se han declarado como constantes al principio de la clase. Además, se obtienen la resolución de la pantalla y se guarda en dos constantes llamadas screenHeight y screenWidth.

El flujo del programa comienza en esta clase mediante los métodos **main** y **start**.

Métodos:

- **Constructor:** establece las características básicas de la ventana (título, tamaño y localización). Además se establece la operación por defecto de cierre de la ventana. Por último se invoca al método initContents().

- **InitContents:** construye los elementos del menú, se monta el "layout" y se añaden los elementos correspondientes en cada zona.
- **Start:** inicia el array de círculos y se invoca al bucle infinito.
- **Main:** hace una instanciación de Window, la cual se hace visible y se llama al método start.

CirclePanel:

Aquí se define el concepto de panel de círculos. Está definido por 4 atributos que consisten en una colección de círculos, un vector para la posición del ratón y dos variables booleanas para controlar la interacción con los límites y el tipo de aceleración.

Esta clase contiene todo lo relacionado con el dibujado del panel y su contenido junto al bucle de actualización de los círculos.

Métodos:

- **Constructor:** inicializa el vector posición a (0,0) y se añade el escuchador de ratón ya que esta es la clase oyente.
- **PaintComponent:** llama al paintComponent de JAVA para ejecutar las instrucciones por defecto de este método, a continuación se realizan tantas iteraciones del método paint como círculos haya en la colección de círculos. Finalmente, se ejecuta la instrucción repaint que llama al paintComponent de nuevo.
- **infiniteLoop:** contiene un bucle que no finaliza hasta que se cierra el programa, en este se actualizan los círculos y se tienen en cuenta los dos booleanos para saber si se tiene que seguir un hilo u otro en cada uno de ellos.
- **resizeBallsCollection:** monta una nueva colección de círculos sobreescribiendo el actual.
- **mouseDragged:** actualiza el vector obteniendo la posición del ratón.
- **mouseMoved:** actualiza el vector obteniendo la posición del ratón.

Circle:

En esta clase se define el concepto de un círculo, el cual está definido por 5 atributos y 4 constantes. Los atributos consisten en la forma, el color, el vector posición, velocidad y aceleración. Las contantes son el diámetro(50.0), velocidad máxima(5), la aceleración en caída(0, 0.05) y finalmente aceleración del ratón(0.12).

Aquí se encuentran todos los métodos que interactúan con los atributos del círculo.

Métodos:

- **Constructor:** impone las propiedades iniciales del círculo de manera aleatoria exceptuando la aceleración que empieza en (0,0).
- **Paint:** realiza el dibujado del círculo. Incluye la clase RenderingHints para obtener una mejor calidad de renderizado.
- **Movement:** actualiza la velocidad y realiza el movimiento del círculo mediante la suma de la velocidad y la posición actual. Para evitar que el círculo vaya demasiado rápido e impone un límite de velocidad (la constante MAXSPEED). Por último actualiza la forma del círculo.

- **FallingAcceleration:** impone la aceleración constante de caída a la aceleración del círculo.
- **MouseAcceleration:** calcula la aceleración que debe tener el círculo dependiendo de su posición respecto al ratón y actualiza esta.
- **InteractionWithWalls:** si se produce alguna interacción con los límites del panel de círculos, invierte la velocidad en el eje correspondiente.
- **InteractionWithoutWalls:** si se produce alguna interacción con los límites del panel de círculos, permite al círculo salir del marco del panel y una vez completamente fuera de este, lo traslada al lateral contrario, oculta a la vista hasta que entre de nuevo en el panel.

Vector:

Contiene el concepto Vector junto a sus dos atributos X e Y. Además contiene las operaciones más utilizadas para operar con vectores.

Métodos:

- **Add:** suma componente a componente de dos vectores.
- **Sub:** resta componente a componente de dos vectores.
- **Mult:** multiplica el vector por un escalar pasado por parámetro.
- **Div:** divide el vector por un escalar pasado por parámetro. Si el escalar introducido es 0, saltará una excepción.
- **Mod:** obtiene la magnitud del vector.
- **Uni:** obtiene el vector unitario del vector con el que se llama al método.
- **Lim:** comprueba si el escalar introducido es menor que el módulo del vector y si es así, se obtiene el vector unitario y se multiplica por el escalar.

Conclusiones

Experiencia obtenida:

- Capacidad de análisis del enunciado que lleva a una correcta planificación lo cual permite una implementación fluida de la solución.
- Programar en equipo, coordinándonos con herramientas como GitHub.
- Programación orientada a eventos e interfaz gráfica de usuario.
- Aplicación de los vectores a la programación.

Lecciones aprendidas:

- Al sobrescribir un método de Java, considerar la invocación a este método para que se incluyan las operaciones por defecto en nuestro método, evitando así posibles alteraciones indeseadas. Por ejemplo: incluir en nuestro paintComponent de la práctica: super.paintComponent (Graphics g)
- Al incluir todos los métodos que interactúan con los atributos de una clase en esta misma, evitamos un uso excesivo de los get y set. Además, proporcionamos mayor independencia entre las diferentes clases.

- Pensar en las diferentes clases que vamos a necesitar antes de ponernos a programar, para tener los conceptos bien separados.
- Analizar los aspectos a tener en cuenta en la práctica y como implementarlos en la solución utilizando pseudocódigo y diagramas o bocetos.
- Al utilizar GitHub, hemos podido llevar un exhaustivo control de las diferentes versiones del código y nos hemos motivado viendo como avanzamos poco a poco.

Problemas a los que nos hemos enfrentado:

- Programar con eventos y entender como se relacionan con el resto del código, ya que hasta ahora hemos programado de forma secuencial (recorridos, búsquedas, etc.).
- Adaptarnos a las restricciones del enunciado.
- Comprender como implementar el movimiento de los círculos (vectores).