Taller Git - MVC

# Objetivos:

* Analizar la estructura de un proyecto implementado a partir del patrón arquitectónico MVC.
* Realizar modificaciones al proyecto de forma distribuida utilizando GitHub.
* Resolver conflictos de integración de código utilizando diferentes estrategias.

# Antecedentes

Cada grupo debe crear una organización dentro de Github y crear un nuevo repositorio en donde todos los integrantes puedan tener permisos para realizar cambios (rw). La primera parte de este taller la deben realizar en conjunto, mientras que la segunda parte debe ser dividida entre los integrantes y deben subir sus avances al repositorio remoto.

## Pasos previos

1. Un integrante debe crear una organización en GitHub con un nombre cualquiera para este taller.
2. Dentro de la página de la Organización debe crear un nuevo repositorio.
3. Dentro de este repositorio debe agregar a los demás integrantes del grupo (**Collaborators**).
4. A cada integrante debe darle permiso para poder modificar(**write**) el código del repositorio.
5. Después, debe clonar el repositorio en un directorio de la computadora local.
6. Descomprimir y copiar el código del proyecto Snake dentro del repositorio local.
7. Finalmente, debe agregar los nuevos archivos al repositorio remoto.

# Parte 1

1. Abrir su correo asociado a Github y aceptar la invitación para colaborar en el proyecto.
2. Clone en un directorio de su computadora el repositorio remoto. (Esto deben realizarlo todos los integrantes en cada computadora)
3. Importe el proyecto al IDE de su preferencia (Se recomienda usar Eclipse).
4. Observe y analice la estructura del proyecto, complete lo siguiente.

* Según su criterio, ¿cómo clasificaría a las clases/archivos que conforman el proyecto?:

|  |  |  |  |
| --- | --- | --- | --- |
| Modelos | Vistas | Controladores | Auxiliares |
| GameModel  GoldModel  SnakeModel  Position | GameView  GUIView  RectangularTile  RoundTile  GameTile | GameController  GameFactory  IgameFactory  Main | Constants  GameOverException |

* ¿Considera usted que el proyecto fue implementado siguiendo los principios del patrón MVC?

Si, ya que separa los datos de la aplicación en las clases como SnakeModel y GoldModel los cuales representan los elementos que usan los modelos de objetos especificos, por ejemplo el RectangularTile se usa en el SnakeModel y RoundTile se usa en la clase GoldModel, esto se diferencia de la interfaz del usuario la cual se ve representada en las clases GameView y GUIView, estas clases usan los diferentes elementos graficos que se presentan directamente al usuario como el comboBox para seleccionar que juego quiere.

La logica de control se ve representada en la clase GameController el cual actua como conexión entre la vista y el modelo, gestionando a traves de los diferentes metodos las acciones correspondiente de los modelos y la interaccione del usuario en la vista, por ejemplo actua para establecer los metodos que mueven el snake (en el juego Snake) usando las flechas de direccion del teclado, la clase SnakeModel tiene la estructura de los objetos que crean el juego Snake, la clase GameView presenta esos objetos al usuario y la clase GameController usa los metodos para interactuar la clase SnakeModel y presentar esas interacciones al usuario directamete a traves de la clase GameView.

Tambien se presentan diferentes clases extras y necesarias como la clase CONSTANTS o la clase que maneja las excepciones GameOverException.

# Parte 2

Realicen las siguientes modificaciones al código de acuerdo, tomando uno solo rol:

1. **Integrante2**:
   1. Ventana principal:
      1. Cambiar las dimensiones de la ventana a 15x15.
      2. Cambiar el texto del botón “Start Game” a “Let’s Play”
      3. Cambiar el color de fondo de la pantalla principal de blanco a gris (lightGray)
   2. Juego Snake:
      1. Para el juego ‘Snake’ cambie el color de la serpiente a verde (GREEN).
2. **Integrante3**:
   1. Ventana principal:
      1. Cambiar el texto del botón “Start Game” a “Empezar”
      2. Cambiar el color de fondo de la pantalla principal de blanco a celeste.
   2. Juego Gold:
      1. Cambie el color de relleno de las fichas a verde (green) y el color del borde a azul (blue).
      2. Por cada ficha recolectada, asigne 3 puntos en lugar de 1.
3. **Integrante1**(creó el repositorio):
   1. Juego Gold:
      1. Cambie el color de relleno de las fichas a verde (green).
      2. Por cada ficha recolectada, asigne 2 puntos en lugar de 1.
   2. Juego Snake:
      1. Aumente el número de frutas de 1 a 3.

Nota: Tengan en cuenta que cada integrante debe realizar los cambios que le corresponde, pero al finalizar deben subir en el orden (integrante 2, integrante 3, integrante 1). Esto es para tratar de generar un conflicto de integración de código.

# Parte 3

Responda a las siguientes interrogantes (Solo uno del grupo debe subir este archivo al GitHub con las respuestas grupales):

1. ¿Le resultó complicado realizar los cambios solicitados?

Fue sencillo, ya que las clases estaban bien organizadas y tenian los comentarios correspondientes al funcionamiento, de manera que para cambiar un aspecto especifico de un objeto, por ejemplo “coin” buscamos su modelo y realizamos los cambios a los atributos o metodos correspondientes.

1. ¿Cuáles considera usted que son los archivos/clases ‘más importantes?

La clase GameController, ya que sin ella no se pudiera relacionar las clases Modelo con las clase Vista y los cambios no se verian presentados al usuario, ya que no hay relacion entre estos

1. ¿Qué aspectos ayudaron a realizar los cambios?

La organización de las clases en Modelos, Vistas y el Controlador, ya que separa los datos de los objetos de su presentacion al usuario y como interactuan estas dos, este estilo de arquitectura y los comentarios correspondientes, nos permite ubicar facilmente la clase y el metodo que realizan las opciones especificas de los cambios pedidos

1. Luego de haber explorado el código, ¿considera usted que se respetan los principios de MVC?

Si, porque separa la logica del programa, la intefaz que se le presenta al usuario y los componentes intermediarios entre estos ,es decir los que tienen la logica entre ambos.

1. ¿Qué cambios haría para mejorar la arquitectura de la aplicación?

Cambiar el nombre de las clases para clasificarlas como Modelos y Vistas a aquellas cuyo nombre de clase, no lo expresa.Al indagar en el codigo notamos el comportamiento de este con respecto a las demas clases, por ejemplo Position describe una posicion en dos dimensiones entonces representa un modelo al igual que GameModel, de igual forma GameFactory se lo considera un controlador, ya que interactura directamente con la vista ( comboBox) y la logica de su interacion con el modelo en este caso, retorna un objeto del modelo indicado en al interaccion.

Los nombres de las clases deben represtar este comportamiento, por ejemplo GameFactory cambia a GameFactoryController para mejorar en entendimiento rapido de estas clases

Tambien se pueden crear clases abstractas o interfaces para generlizar de mayor forma en Modelos, Vistas y Controladores y que cada clase herede o implemente dependiendo que elemento es.

# Entregables

1. En el repositorio de GitHub debe estar el código con los cambios requeridos.
2. En el repositorio de GitHub debe estar subido este archivo con las respuestas de todo el grupo.
3. En Sidweb debe subir el enlace del ultimo commit del repositorio remoto.