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. Agregar a los integrantes del grupo a la Organización enviándoles invitación. (Cada integrante debe abrir su correo y aceptar la invitación)
3. Dentro de la página de la Organización debe crear un nuevo repositorio.
4. Dentro de la configuración de la organización ir a “[**Member privileges**](https://github.com/organizations/TallerGIT/settings/member_privileges)” y en “**Organization members**” darles permiso “**Write**” a los miembros.
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.

## Comandos importantes

* Conocer el estado de su repositorio: **git status**
* Clonar localmente un repositorio remoto: **git clone <repositorio\_remoto>**
* Agregar todos los cambios realizados al stage: **git add .**
* Guardar todos los cambios agregados al stage: **git commit -m “Comentarios de los cambios realizados”**
* Enviar al repositorio remoto todos los commit sin enviar: **git push origin master**
* Descargar los nuevos cambios desde el repositorio remoto: **git pull**
* Si hay un usuario grabado en su computadora y desea utilizar otro: **git config --local credential.helper ""**

# 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, pero se puede importar desde Netbeans).
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 |
| GameFactory  GameOverException  GoldModel  IGameFactory  Position  GameModel  SnakeModel | GUIView  GameTile  GameView  RectangularTile  RoundTile  Main | GameController | Constants |

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

**Juan Nebel:**

Sí, los nombres de las clases daban una pista de qué función MVC cumplían las mismas, de igual manera los comentarios. Tal vez si se divide claramente en paquetes sería más fácil diferenciar la función, no obstante es entendible que en ciertas clases el rol no esté estrictamente definido, según la referencia dada en clase (TU Wien. (2013). Motor view controller planner. Best practice software engineering. Recuperado de: <http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html> ), programas con bastante uso de interfaz gráfica (juegos) frecuentan tener clases que desempeñen más de un solo rol.

**Ricardo Vilcacundo:**  
  
Si, puesto que las clases que fueron creadas representan en parte los componentes del MVC, sin embargo como se menciona en comentarios anteriores, hubiera sido preferente separar por paquetes para tener una mayor facilidad para distingir a que partes del MVC corresponde.

**Josè Baño:**

Si, porque cada clase cumplia con una de las funciones del MVC(Model, View, Controller), principalmente de modelos y vista, existiendo solo un controlador que captaba las entradas del usuario.

# Parte 2

Cada integrante debe clonar en su computadora el repositorio remoto y luego todos deben trabajar paralelamente con esta versión. Luego, el orden para enviar los cambios realizados debe ser el siguiente:

1. **Integrante1**(creó el repositorio):
   1. Ventana principal: (GUIView)
      1. Cambiar el texto del botón de “Start Game” a “Let’s Go!!!”.
   2. Juego Gold: (GoldModel)
      1. Por cada ficha recolectada, asigne 2 puntos en lugar de 1.
2. **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).
3. **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.
4. **Integrante1( otra vez )**:
   1. Juego Gold:
      1. Cambie el color de relleno de las fichas a verde (CYAN). (El grupo resolvió hacer fichas color verde para poder diferenciar las fichas del fondo)
      2. Por cada ficha recolectada, asigne 4 puntos en lugar de 2.
   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 1, integrante 2, integrante 3, integrante 1). Esto es para tratar de generar un conflicto de integración de código con cada integrante.

# 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?

**Juan Nebel (Integrante 1):**

Tuve un poco de dificultades porque mi compañero justo había editado la misma linea que me tocaba cambiar a mí, por ende git requería de un merge manual.

**Ricardo Vilcacundo (Integrante 2):**

Tuve un pequeño inconveniente a la hora de subir mi commit puesto que el Compañero que hizo el commit anterior trabajo en la misma linea de codigo que yo, por ende se tuvo que corregir de manera manual ciertos problemitas que existían, pero en fin, si me resultó complicado hacer los cambios, debido a mi falta de experiencia.

**Josè Baño (Integrante 3):**

No hubo problemas al realizar los cambios en el repositorio local, porque fue fácil de encontrar los valores que se pedían cambiar, pero al momento de subir los cambios al repositorio remoto hubo conflicto con los cambios realizados por mis compañeros. No fue complicado arreglar el conflicto, porque mis compañeros me ayudaron a resolverlo.

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

**Juan Nebel (Integrante 1):**

Yo considero que todos los archivos cumplen un rol importante, sin embargo, si debo elegir un rol indispensable (y en consecuencia los archivos que cumplen dicha función) sería el rol del modelo, porque uno puede elegir varias maneras de presentar una interfaz gráfica (JavaFx, Swing, etc.) y de esta decisión derivan las clases controladoras y las clases de vistas. Incluso se puede realizar un programa sin interfaz gráfica (scripts, sistemas embebidos, etc.), pero un modelo que almacene información y defina la lógica de la misma es indispensable.

**Ricardo Vilcacundo(Integrante 2):**

Todas las clases Cumplen un papel importante en el programa, no obstante, las clases que considero de mayor importancia son aquellas que tienen los mecanismos para poder acceder a información (Modelo), esto es debido a que se pueden trabajar en sistemas que no tengan interfaz gráfica.

**Josè Baño(Integrante 3):**

Las clases más importantes del programa son: GameModel y GameView. Porque son el Modelo y Vista principales del juego. El componente más importante del MVC es el modelo, porque este se encarga de manejar todos los datos, es el que maneja todo lo que está sucediendo en la aplicación.

1. ¿Qué aspectos ayudaron a realizar los cambios?

**Juan Nebel (Integrante 1):**

Git me proporcionó un buen desglose de las líneas que necesitaban ser revisadas por un programador, sin esto, el trabajo se tornaría muy tedioso y largo. También me ayudó la explicación en clase y tener conocimiento de qué era importante preservar al momento de definir el merge manual.

R**icardo Vilcacundo(Integrante 2):**

Básicamente Git proporcionaba los pasos a seguir para poder aplicar los cambios, y gracias a investigaciones y parte del contenido de la clase el proceso resultó menos tedioso de lo que parecía,cabe mencionar que el trabajo grupal fue de mucho importancia porque los otros miembros del grupo ayudaban a despejar aquellas dudas que se tenía.

**Josè Baño(Integrante 3):**

Los comentarios de cada clase me ayudaron a comprender cada función, facilitando encontrar los valores que se pidieron cambiar, al momento de unir las versiones de cada integrante, se produjo un conflicto con los cambios que realizaron mis compañeros, el cual se pudo arreglar con facilidad utilizando la plataforma de Git y los comandos aprendidos en clases.

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

**Juan Nebel (Integrante 1):**

Creo que se intentó seguir el paradigma MVC como guía general al momento de diseño, no obstante ciertas clases, que incluso tenían modelo en el nombre, parecían - a ratos - realizar funciones de controlador(SnakeModel, GoldModel, GameModel, etc.).

**Ricardo Vilcacundo(Integrante 2):**

Luego de revisar el codigo puedo notar que en gran parte si se intentó seguir la lógica del MVC puesto que se puede diferenciar a simple vista que clases cumplen propiamente las funciones establecidas por el MVC, sin embargo existen clases que cumplen más de un rol que no necesariamente es el que tiene especificado, pero es mínimo.

**Josè Baño(Integrante 3):**

No, porque existían clases que trabajaban como modelo y controladores al mismo tiempo. Ademas la unica clase con Controller en el nombre cumplia con la función de recibir las entradas del usuario y enviarla al modelo, pero la única interacción que tenía con la vista era el comando de refrescar la imagen cada 150 milisegundos o cuando se presionaba una tecla.

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

**Juan Nebel (Integrante 1):**

Carezco de experiencia haciendo juegos, entonces no tengo muchos puntos de comparación, pero yo trataría de separar un poco más las funciones de modelo y controlador, específicamente en las clase SnakeModel. Se me ocurre también separar las clases en paquetes para brindar una visualización más ordenada a otro ingeniero que tenga que trabajar sobre la aplicación.

**Ricardo Vilcacundo( Integrante 2):**

Como ya se mencionó existen clases que cumplen más de un rol, entonces lo que haría para mejorar la arquitectura es separar y dejar que cada clase cumpla el rol establecido para que de esta manera pueda poder separar en paquetes y sea un programa fácil de entender.

**Josè Baño(Integrante 3):**

Separa las funciones de las clases que cumplen de modelo y controladores para que cumplan un papel más definido, y separar las clases en paquetes según que tipo de componente 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 último commit del repositorio remoto.