Informe TPE Programación Orientada a Objetos

Integrantes:

- Kim, Azul Candelaria (Legajo: 60264)
- Lombardi, Matías Federico (Legajo: 60527)
- Rosati, Santos Matías (Legajo: 60052)

Funcionalidades agregadas:

1. **Cuadrado, Elipse y Línea:** Fueron añadidas nuevas formas geométricas, todas extendiendo de Figure. Cuadrado es una subclase de Rectángulo y Círculo delegó la mayor parte de su comportamiento a Elipse. Además fueron agregados sus respectivos botones para crear dichas figuras. Estas nuevas figuras también se pueden mover en el espacio.

2. **Borde y Relleno:** Ahora las figuras pueden modificar tanto el ancho del borde como el color del mismo y del relleno (excepto para la línea ya que consta únicamente de un borde). Luego fueron añadidos un slider y dos ColorPickers en el menú para permitir el cambio de grosor y color de borde, y relleno de una figura.

3. **Borrado y Selección Múltiple:** Se modificó la implementación para que permitiera el seleccionado de múltiples figuras mediante el uso de una colección. Esta se realiza mediante el dibujo de un rectángulo imaginario. Además, se agregó un botón que permite el borrado de las figuras seleccionadas.

4. **Profundidad:** Se agregaron dos nuevos botones para traer al frente o mandar al fondo la figura seleccionada. En caso de tener varias figuras seleccionadas, todas pasan a estar al fondo o al frente del resto (sólo una queda efectivamente al fondo o al frente).

Modificaciones realizadas:

- La implementación original utilizaba el operador instanceof al momento de mover o dibujar una figura, o para saber si un punto pertenece a ella. Al no aprovechar las ventajas del paradigma orientada a objetos (por ejemplo, la herencia de algún método en *Figure*), se los removió y en su lugar se realizaron las siguientes modificaciones:
    - Con respecto al movimiento de figuras: se creó una interfaz *Movable*, de la cual extienden tanto *Point* (en esta clase además se cambió el modificador de acceso de sus variables de instancia de public a private, ya que éstas deberían permanecer ocultas) como *Figure*. De esta manera, una figura para cumplir con el contrato de la interfaz, obtiene un vector de sus puntos e invoca al método correspondiente para moverlos en el espacio.

    - En cuanto a la creación de figuras nuevas: con respecto al front-end, se creó una clase *FigureButton* a la cual pertenecen los botones de agregado de figuras para que sepan crear la figura correspondiente (esto se hizo en pos de mejorar el estilo del código ya que preguntar reiteradas veces qué botón de figura estaba seleccionada era más bien imperativo). Con respecto al back-end, se implementó el patrón de diseño de observadores el cual consiste en que la clase observada mantiene una lista de sus dependientes y

al momento en que ocurre un cambio en su estado, los notifica para que realicen la acción correspondiente. De esta forma, la manera en la que se actualizan los cambios en la vista se mantiene independiente de la implementación del modelo, quien únicamente debe avisar cuándo ocurrió un cambio.

- En relación con la pertenencia de un punto a una figura: se quitó el método *figureBelongs* dentro de *PaintPane* y se agregó este comportamiento dentro de *Figure*.

- Se cambió el movimiento de las figuras ya que la implementación anterior no era precisa (no se actualizaba nunca el punto de inicio, por lo que, por ejemplo, si se movía el cursor inicialmente mucho hacia la derecha y luego un poco hacia la izquierda, la figura continuaba moviéndose hacia la derecha).

- Debido a que el *setOnMouseClicked* causaba un comportamiento indeseado ya que el mismo se llama una vez soltado el mouse luego de arrastrarlo en lugar de llamarse solamente al inicio del click, se decidió migrar su comportamiento al evento de *mouseReleased*, verificando entonces que no se haya movido el mouse luego del click inicial.

- En la implementación original al tener varias figuras solapadas, y seleccionar mediante un click la de más arriba, aparecían en el *StatusPane* como seleccionadas todas estas. Debido a que este comportamiento era incorrecto, se lo modificó de forma que apareciera únicamente la figura seleccionada.

- *CanvasState* guardaba en un *ArrayList* las figuras que se encontraban en pantalla. Esto se decidió cambiar a una *LinkedList* para aprovechar el iterador descendiente que esta provee al momento de seleccionar una única figura con un click y para hacer más eficiente el reordenamiento al traer al frente o enviar al fondo las figuras seleccionadas. Asimismo, se le agregó un set de figuras para realizar operaciones con las que estuvieran seleccionadas; para esto, se le asignó un id único para cada figura (siguiendo los lineamientos vistos en clase) para que puedan ser usadas correctamente dentro de un *HashSet*.

Dificultades encontradas:

1. **Herencia de línea:** se presentaron dificultades decidiendo cómo aplicar la herencia de la línea desde figura, ya que se necesitaba poder acceder al color de relleno de cualquier figura pero no sería una buena herencia que una línea tenga esta propiedad, dado que esta no es utilizada. La decisión tomada fue crear un enum de propiedades de colores (*ColorProperty*) y tener en *Figure* como propiedad, un mapa de *ColorProperty* a *Color*. Junto con esto, se agregó el método *isFillable* para determinar si una figura tiene relleno o no. De esta manera, a cada figura se le

agrega la clave correspondiente al color del relleno (y su correspondiente valor indicando el color) si y sólo si es rellenable.

2. **Separación del front-end y back-end:** se generaron dudas respecto a si una figura debería saber dibujarse a sí misma o no, debido a que la misma pertenece al back-end. Al ser algo muy propio de la interfaz gráfica utilizada, se optó por el patrón de diseño mencionado en las modificaciones realizadas debido a las ventajas que provee: el objeto observado no necesita saber nada sobre el observador, son totalmente independientes, y ante el agregado de otro observador, no hay necesidad de modificar al objeto (lo que da como resultado un bajo acoplamiento). Cabe aclarar que se agregó el método *getData* aprovechando que los métodos utilizados de *GraphicsContext* para dibujar necesitan 4 datos de cada figura.

3. **Modularización del front-end:** otro de los inconvenientes encontrados fue la decisión sobre qué hacer con los botones en *PaintPane.* Una de las posibles soluciones que se pensaron fue hacer una clase *LeftButtonsBar* (similar a la del menú), la cual tendría la botonera izquierda y todas sus funcionalidades. Sin embargo, se decidió dejarlo como estaba debido a que al encontrarse tan acoplada la relación entre cada una de las componentes de *PaintPane*, el código resultante no sería tan claro como lo esperado.

4. **Rectángulo de selección:** surgieron dificultades en cómo crear el rectángulo utilizado para la selección múltiple. En un principio, se aplicó un método que recibía una instancia de *Rectangle*; no obstante, este proveía propiedades innecesarias (borde, relleno, entre otros), además que no proporcionaba una solución genérica (por ejemplo, si se quiere saber si una elipse contiene a otra figura). Una de las propuestas fue crear nuevas clases de figuras que sean movibles (dejando a las originales inmutables en este aspecto) que sirvan como planilla para las figuras que se pueden dibujar; sin embargo, se le estaría agregando más complejidad al diseño que la deseada. Por lo tanto, se determinó que sería más apropiado crear una interfaz funcional *Selector,* la cual cada figura recibe en el método *isContained*, cuyo único método abstracto define si un punto pertenece o no al área deseada. De esta forma, en *PaintPane* simplemente se utiliza una expresión lambda para indicar el método de selección.

Apéndice 1: diagrama de clases de las figuras