TPE POO

Hernando, Lautaro (62329)
Ippolito, Martin (62510)
Scilipoti, Marco (62512)

2021

Enumeración y breve descripción de las funcionalidades agregadas

1.Cuadrado, Elipse y Línea

Para la implementación de las 3 nuevas figuras decidimos crear en el back end 3 nuevas clases abstractas una para cada una de ellas. Las 3 clases se extienden de la clase abstracta llamada Figure la cual modela el comportamiento por default de una figura. En particular el cuadrado extiende de la clase abstracta rectángulo y el círculo implementado por la cátedra extiende de la clase elipse creada para este punto. También cabe destacar que dentro de las clases de front end, se crearon sus respectivas clases las cuales implementan el método abstracto draw el cual indica cómo dibujar dichas figuras.

2.Personalización de figuras: Borde y relleno

Para la implementación de dicho punto lo primero que hicimos fue agregar al back end una clase denominada FigureStyle la cual contiene el estilo de una figura. Mediante la composición, hacemos que la clase figura contenga un FigureStryle y es mediante los métodos de la interfaz Colorable que cambiamos el estilo de la figura en sí.
En cuanto al frontend se implementaron los dos colores picker uno para el color del borde y otro para el color del relleno de la figura. Los cuales tienen un default que decidimos que desde el back end les diga cual es dicho default. También para la implementación del ancho del borde decidimos usar el Slider, cuyo valor por default lo posee el Figure (back-end) pero su rango depende de la implementación del front-end.

3.Borrado y Selección Múltiple

Para la implementación de estas dos funcionalidades fue muy importante pensar que el método de selección lo debemos procesar desde el backend, y que lo debíamos procesar en el CanvasState. En particular para el seleccionado lo que hicimos es que una Figura pueda decidir si está dentro de un rectángulo o no, pasando únicamente su diagonal. Entonces cuando desde el front llamamos al método de selección que se ubica en el back end con los puntos de la diagonal, lo que se hace es buscar por los elementos guardados en una lista y ver si alguno de ellos están dentro del rectángulo, si alguno de los mismos está lo que se hace es cambiar el color del borde y luego se los agrega a una lista de elementos seleccionados.
Para el borrado de los elementos seleccionados lo que se hace es un deleteAll a la lista que contiene todos los elementos almacenados mandado como parámetro la lista con los elementos seleccionados, y luego lo único que basta es hacer es desde el frontend un redraw de los elementos que están almacenados.
Cabe aclarar que para almacenar los elementos seleccionados, se utiliza un ArrayList paralelo al LinkedList que almacena la totalidad de figuras.

4. Profundidad

Para la implementación de la profundidad nos dimos cuenta que los primeros elementos de la lista eran aquellos que se dibujan primero, quedando entonces en “el fondo”. Por lo tanto cuando se invoque a los métodos moveToFront/moveToBack, simplemente se los mueve al final o al principio de la lista. En particular el sendToFront lo que hace es mover a las figuras seleccionadas al final de la lista y el sendToBack lo que hace es mandar las figuras al frente de la lista.
Vale destacar que se eligió guardar las figuras dentro de una LinkedList ya que cuenta con métodos que permiten agregar un objeto al principio o al final, según como se desee.

Modificaciones realizadas.

Back-end:

-**[Model.Components.*]** Jerarquía de clases de figuras, se ha mejorado la herencia de funcionalidad. Agregamos figuras como elipse, línea, etc. Point quedó como su propia clase con cambios menores (equals, move, etc). Las figuras tienen métodos final, de modo que el front no pueda cambiar su comportamiento innecesariamente. El front sólo debe modificar el método display, para que sepa dibujarse bajo el contexto del front.

-**[Model.Interfaces.*]** Encapsulamos funcionalidades como interfaces que implementamos en figure. Estas interfaces son movable, drawable, selectable y colorable.

-**[Model.Exceptions.*]** Creamos una excepción propia para el back.

-**[CanvasState]** Ahora puede manejar figuras seleccionadas (incluyendo métodos que agregan utilidad) y puede enviar al fondo/frente una figura. Para todo esto, modificamos la lista original que contenía todas las figuras y la convertimos en una linkedList. Las figuras seleccionadas quedan en un ArrayList separado.

-**[FigureStyle]** Simplemente encapsula los colores de una figura y el grosor de su contorno.

Front-end:

-**[Components**.*] Ahora el front extiende las clases figura, círculo, rectángulo, etc. Con el objetivo de simplemente agregarle la funcionalidad de que sepan dibujarse bajo el contexto del front que se use

-**[Engines.Ovals/Polygons Engine]** El dibujo de las figuras está modularizado por clases con métodos de clase, que reciben una figura y la dibujan según el contexto del front (JavaFX)

-**[Engines.Hex String Engine]** Debido a que el back almacena los colores con un string que tiene el codigo en hexadecimal, tenemos una clase en el front que se encarga de convertir una instancia de Color (Propia de JavaFX) al string que requiere el back.

-**[Engines.CanvasEngine]** Posee un método de clase que, dado un CanvasState(back-end) y un Canvas (front-end), se encarga de redibujar el canvas.

-**[Engines.ButtonsEngine]** Se encarga de configurar, inicializar, asignar, etc. Todos los botones que irán al menú vertical de la izquierda. Por lo que Paint panel nada más debe llamar a métodos de Buttons Engine para el manejo de los botones. Esta clase hace uso de ColotControlsEngine, y de controllers.

-**[Engines.ColorControlsEngine]** Se encarga de crear los ColorPicker/Slider para las modificaciones de colores y grosor del contorno. Además se encarga de su funcionalidad.

-**[Controllers.FigureButtons]** Es un enum que encapsula la funcionalidad de cada botón de creación de figuras. Se encarga de crear el botón (JavaFX) y la figura en el front (La cual la crea en el back a su vez)

-**[Controllers.FunctionButtons]** Es un enum que encapsula la funcionalidad de cada botón de funcionalidad sobre una figura o un grupo de figuras. Se encarga de crearlo y de la funcionalidad propia del botón.

-**[Controllers.MouseEvent]** Simplemente encapsula un evento del Mouse, en el cual se tiene un punto inicial y un punto final.

-**[AppLauncher]** No se han hecho modificaciones significantes, simplemente comentarios.

-**[AppMenuBar]** No se han hecho modificaciones significantes, simplemente comentarios.

-**[MainFrame]** No se han hecho modificaciones significantes, simplemente comentarios.

-**[PaintPanel]** Se ha simplificado/modularizado con la inclusión de las clases mencionadas anteriormente. Haciendo así, una mejor modularización de funcionamiento.

-**[StatusPanel]** No se han hecho modificaciones significantes, simplemente comentarios.

Justificación sobre modificaciones más importantes hechas sobre la implementación original:

Back-end:

-**[model]** Primero tomamos la funcionalidad en común de las diferentes clases y las pasamos a la clase Figure. Además de implementar las diferentes interfaces que definen funcionalidades de las Figures, y de implementar los componentes del enunciado, declaramos los componentes como clases abstractas. De esta manera pudimos declarar el método *display* como método abstracto, y así sea definido por el front-end. De esta manera pudimos mantener el *canvasState* en el backend, y diseñar un back-end que funcione para diferentes front-ends.

-**[canvasState]** Consideramos oportuno mantener la lista con las figuras del canvas, y agregamos la funcionalidad de selección, consideramos que si podía ser implementado en el back-end era mejor que sea implementado en el front-end.

Front-end:

-**[controllers]** Con el objetivo de eliminar la cadena de ifs para detectar si un botón para crear figuras estaba seleccionado, decidimos agrupar todos los botones de figuras en un enum y asi poder manejar

la funcionalidad de crear un boton en si mismo. De la misma manera, creamos un enum de los botones de función que requieren re-dibujar el canvas una vez que son clickeados por el usuario.

-**[engines] La mayor parte de los cambios fue pasar toda la funcionalidad de *paintPanel* al resto de los proyectos, principalmente a la clase *ButtonsEngine* y *ColorControlsEngine***

Problemas encontrados durante el desarrollo.

Principalmente, nos hemos encontrado con el desafio de utilizar JavaFX para el desarrollo de la interfaz gráfica del proyecto. Para lo cual debimos aprender y leer su documentación para conocer la mejor forma de resolver algunas cuestiones (que métodos utilizar, como funciona, etc).
Una vez que hemos leído e interpretado el código original, comenzamos a realizar las modificaciones necesarias. Primero nos propusimos corregir todo lo necesario de la implementación original para luego agregar funcionalidad.
El primer problema fue realizar una correcta separación front y back, para lo cual hemos tenido dudas/problemas (Por ejemplo, ¿Cómo debemos dibujar las figuras?). Ante todo, lo hemos solucionado preguntándonos, ¿Si cambio el front, el back sigue funcionando correctamente? ¿Y viceversa?.

Por otro lado, un problema que encontramos fue como simplificar, modularizar, etc, la clase PaintPane. Se nos hizo muy difícil comprender qué hacía esta clase en cada método. Para lo cual la solución fue separar las funcionalidades en distintas clases. Por ejemplo, la creación y manejo de los botones de JavaFX , la hemos separada en otras clases, entre ellas ButtonsEngine (la cual también se modulariza de distintas formas).

También uno de los problemas más decisivos fue la implementación correcta del canvas state en donde desde ahi podemos hacer el funcionamiento correcto de seleccionar y las funcionalidades del borrado, cambio de color y el send to front/back.