05-830 HW1: Implement a Node Editor

Hongyi Zhang (AndrewID: hongyiz2)

User Manual

1. Draw a box: press anywhere (not inside a selected box) to start, drag the mouse to draw, and release to finish drawing.
2. Select a line style: click on the line style -- a surrounding box will appear.
3. Select a box: click on the box -- the selection handles will appear.
4. Move a box: select the box if it's not selected, and drag it to where you want it to be. The cursor shape will change to a hand when it is on a selected box and ready to move.
5. Delete a box: select the box if it's not selected, click the Delete button or press backspace on the keyboard to delete it.
6. Select a line color: click the LineColor button, choose a color and click OK.
7. Select a face color: click the FaceColor button, choose a color and click OK.
8. Bring a box to top: select the box and click the BringToTop button.
9. Resize a box: select the box if it's not selected, and drag one of the selection handles to change its size. The cursor shape will change when it is on a handle.

Some design considerations

1. Boxes can only be drawn within the drawing panel. If the user starts from outside to draw, the resulted box will still be restricted within the region if it ends inside.
2. When the box is being moved around or resized, it is always restricted within the drawing panel.
3. The cursor has three possible types of shapes: DEFAULT, MOVE and RESIZE. Somehow system-predefined MOVE doesn't work on MacOS, so I chose HAND shape instead.
4. The window can be resized, and the line that separates the selection panel and the drawing panel will be adjusted accordingly.
5. The box name is written inside the box if the box is large enough to display the name, otherwise it will be written above the box.
6. The box name and the selection handles have the same color as the line style of the box for aesthetic reason.
7. There is some tolerance when the user is trying to select a line style, a box or a handle, which means the target will still be selected even if the user has clicked a few pixels off.

Keep track

1. I used Java Swing.
   1-1. To be specific, it is Java SE 10 and the corresponding Swing/AWT library.
   1-2. These tools are new to me. I had only used PyQt5 in Python to draw two buttons before so I don't really have a clear idea of their similarity.
2. I used Visual Studio Code 1.41.1 as the IDE.
   2-1. I didn't use any plugins.
   2-2. I was previously coding using Eclipse 4.14.0 but found it not as convenient so I switched to VSCode later.

3. The total estimated time spent
   3-1. Time reviewing the Java language: 1hr. I haven't really coded a large program in Java so I took some time to go through what the Java language can offer.
   3-2. Time reading the documentation before coding: 1hr
   3-3. Time thinking about the design: 1hr
   3-4. Time actually typing the code: 7hrs
   3-5. Time looking up and figuring out how to do things while coding: 4.5hrs
   3-6. Time debugging the code: 1.5hrs
   3-7. Time refining design and cleaning code: 2hr
   3-8. Time writing the report: 1hr
4. Main references
   4-1. Java Swing tutorial: https://docs.oracle.com/javase/tutorial/uiswing/index.html
   4-2. Stack Overflow
   4-3. Code references are from various websites and are all cited in the source.
5. The amount of code
   5-1. Actual number of lines of code implemented is 525, as counted by LineCount 0.1.7 for Visual Studio Code.
   5-2. I used the getter/setter generation tool in Eclipse to generate 42 lines of code.

Discussion

Both the programming language and the Swing toolkit are sort of new to me, so I would say this assignment is a real yet interesting challenge. Specific to this assignment, I think the hardest part is not about a particular functionality (except resizing with handles, which took me a really long time to figure out the correct math), but about the interactions of different features. Most of the implementation logic is not difficult, but when they are combined together, it becomes very hard to verify if the interface works as intended due to the infinite possible sequences of these actions. Debugging also becomes difficult as the bug might only appear after a particular series of events, and automated tests might also be hard to realize.

There are also some other difficulties. 1) There are a number of details to take care of. For example, allow drawing a box in different directions, forbid drawing/moving a box out of boundary, deselect the selected box when starting to draw another box, etc.; 2) There can be ambiguities in how the interface should behave. For example, should some of the selection handles be displayed even if another box is on top of them? Should we move the selected box when the mouse clicks within the box region but another box is in the way? Choice of the specifications might have to rise from usability tests, and for this assignment I used PowerPoint as a reference for these behavior; 3) Code logic has to be carefully placed in each listener. For example, a MouseClicked event is always accompanied by a MousePressed event and a MouseReleased event. It is possible that some logic inadvertently be executed more than once if they are not properly placed in the code.

As to learning the toolkit, I started with the tutorials written by Oracle but found they are too long and not as informative. So I decided to directly search for some starter code, and looked

up documentations and StackOverflow as needed as I was adding more functionalities to it. But overall, the logic of how the whole thing works is not elusive. Modification and addition based on the starter code is not that hard either, since the API has pretty consistent signatures and designs. Some weird stuff, however, exists in the inheritance relationship between javax.swing and java.awt, between Graphics and Graphics2D (the former of which doesn't have the setStroke method), etc..

One thing I found annoying is the precise placement of each graphic component. I learned that Java Swing provides a LayoutManager interface, but I ended up not using it, partly because I didn't feel it's worth the time learning it, and partly because I could not directly imagine how it could help display the layout in my mind. I heard that NetBeans is an IDE that supports direct graphical manipulation of the component layout, so that might be able to make life easier.

Finally, may I discuss some issues in the design model. As I was designing my node editor, the boxes appear only as graphics, but not as existent components in the panel. The listener for the MousePressed events is registered by the window rather than the "box components", and inside the listener we determine which box has been clicked on. This doesn't sound natural from an object-oriented perspective, and ideally the listener should be registered by each "box component". I didn't take time to consider this option or redo my code after I completed my design. There may be a neat way to do this in Swing but I'm currently not so sure how.