Groupe 9

Project follow-up and conception choices.

For this third iteration we had three main tasks.

- Finish story 4, with the addition of thickness and a label on the shapes.
- Implement syntax coloring and highlighting from story 11.
- Refactor the project to have a more consistent and clean code.

Translate from TikZ to canvas view

Since this is the second part of this story, we already have the “bridge” linking the two representations so it wasn’t too complicated.

Conception

The idea is that when we detect changes in the TextArea with the listener *handleCodeChange*, we check if all the lines are correct using regular expressions. When all the code is correct, we delete all the shapes in the canvas and we create a new java representation of each line (each line is a different shape). From the java representation, then we create the JavaFX shapes and add them to the canvas.
Also we have to keep track of the shapes that are selected, and have them correctly selected after delete and redraw all the shapes. In order to do so, we detect the lines that have changed and adjust the IDs of the shapes that are selected so we can select them once we have redrawn all the shapes.

Adding the thickness wasn’t complicated, since it’s only a parameter to add in the java representation.

In order to add labels to the shapes in the TikZ code, previously existing instructions were appended with
“node [align=center, above= *offset_to_center* cm, right= *offset_to_center* cm] { *label_text* }”.
We wanted to keep the one-to-one relationship between a shape on the canvas and one line of TikZ code.

Syntax coloring

In this section, we take the selected shape ID and take the shape in the canvas with it.
Then we get the line position in the canvas with the findWordUpgrade method.
This method creates an array with the start and end position of each word in the given string.
Finally, we highlight the position of the line with RichText.

For highlighting words, it’s exactly the same, we have created an array with all Tikz words and find all those words in textArea. For each word, we return their position and change their color.

Groupe 9

The refactoring

In this iteration we have refactored mostly four functionalities:

1. the import/export
2. the saving and loading of a project
3. the view_controller of the edition of a shape
4. the class utility

Conception

The two first weren't properly integrated in the structure and not MVC and lacked some test cases since the first iteration. All of this was added.

In the third case, the class was growing too large, thus to add new functionalities it was preferable to split it into "EditorController" and "ShapeHandler". Even though both classes are interdependent, the readability is improved.

Finally the utility class was a mix of methods used by the view, as well as the controller and the model making the purpose of the class undefined. It has thus been split into three utility files, one by layer.

Conclusion

In conclusion we consider that we have reached our goals. This refactor allows us to be more confident in the future.