Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

A Swatches Docker for Krita

Google Summer of Code Project

Motivation

Currently, Krita uses the palette docker to enable users to reuse colors. This docker has the following problems:

1. It is very hard to create and maintain visual patterns in the palettes. When deleting a color, all the positioning of the colors that come after it will change. When adding a color, the user can only add it to the tail of the color list and then switch its position with other colors. It's impossible to leave entries empty. Because the visual pattern is hard to maintain, it's hard to find a specific color to reuse. For example, 30 days ago, I stored 30 colors of a character's skin under different lighting environments. Today, I want to draw the character again, and I want to find exactly the color of his skin under dim blue light I chose last time, but I don't remember the RGB value of the color or its position in the docker, then I need to find it among the 30 colors that are only slightly different and have no pattern in their order.
2. The palettes are not stored with individual paintings. This means every time I create a new palette for one individual painting, it will pollute the collection of palettes for other paintings, making it hard for user to find the palette he wants to use.
3. The UI design is not intuitive. This bug report discusses how the UI design is not functioning well: https://bugs.kde.org/show_bug.cgi?id=369343.

As a painter myself, I want to create a new docker that can work as an alternative for users so that we can have better experience of reusing colors.

Introduction

A swatches docker is like the palette docker, but they differentiate from each other in the following ways:

1. A swatches docker allows users to add a color entry to any position they want in the docker, while in the palette, no empty entries can be left before the last color.
2. A swatches docker allows users to delete a color entry or change its positioning with affecting the position, while in a palette, doing so will affect other color entries.

Compared to a palette, a swatches docker enables users to easily create a group of colors with a visual pattern, so that they can easily pick the color they want to pick up, e. g. a color that's slightly brighter that the currently chosen one which has already been registered into the docker.
The swatch docker I want to create also has the following features:

1. Swatches can stored with the paintings they are created for; at the same time, they can also be shared with other paintings.

Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

2. A better, more intuitive UI design.

Project Goals

To create a swatches docker that comes with the following features:

1. A group/matrix of swatches is associated with a painting. Whenever the user is editing the painting, this matrix will be there, but the matrices created for other paintings will not show up.
2. Multiple matrices of swatches can be associated with a painting. Users can do that by adding new matrices.
3. There are global matrices that can used in all paintings.
4. A matrix of swatches associated with a painting can be exported to be global.
5. Users can add colors to the matrices. Users can also delete, rename and relocate all colors in a matrix.
6. Intuitive UI.

Implementation Details

Plugin

This docker, just as other dockers, will be part of Krita's plugin system. Just like other plugins, a class called SwatchesDockerPlugin along with a class called SwatchesDockerDockerPlugin will be created to tell the system how to create the plugin. SwatchesDockerDock is another class that will be created to define the GUI of the docker. It will also handle user input out of the matrix part that is going to be mentioned below.

MVC Design

The docker will show colors in a matrix that allows empty entries. The matrix is going to follow a MVC pattern.
The view class, KisSwatchesView, is going to be a table view derived from QTableView. It will be accepting user inputs and use KisSwatchesDelegate to handle them. In order to display colors from different color profiles, KoColorDisplayRendererInterface and KisDisplayColorConverter will be used by the view.
The delegate class, KisSwatchesDelegate, will be derived from QAbstractItemDelegate. It be handling the interaction between the GUI and the data. It accepts signals from KisSwatchesView and manipulate KisSwatchesModel based on them.
The model class, KisSwatchesModel will be responsible to hold the data in the form of a list or a table. It is going to use KoColorSet, which holds the data of colors and handles palette files.
I will also modify the KoColorSet, so it can handle palette files that come with position information of the colors.
A new color set chooser will be written. I will call it KisSwatchesSetChooser. It will have a better UI design than KisColorsetChooser, and will be compatible with the new functions of the

swatches docker. Users can use to export a group of swatches so that it can be used in other paintings. It will use KoResourcesItemChooser.

Saving and File Format

Currently, Krita uses the KPL as its default format to store palettes. I am going to either the file format, adding an attribute “position” to the it. To handle such changes, KoColorSet will be modified.

The instance of KisDocument of the current painting will be modified so that it tracks the groups of colors associated with the painting. KisKraSaver will also be modified so that the groups can be saved.

Testing

There will be a unit test for each non-GUI class. The tests are going to be written in the Qt Test Framework.

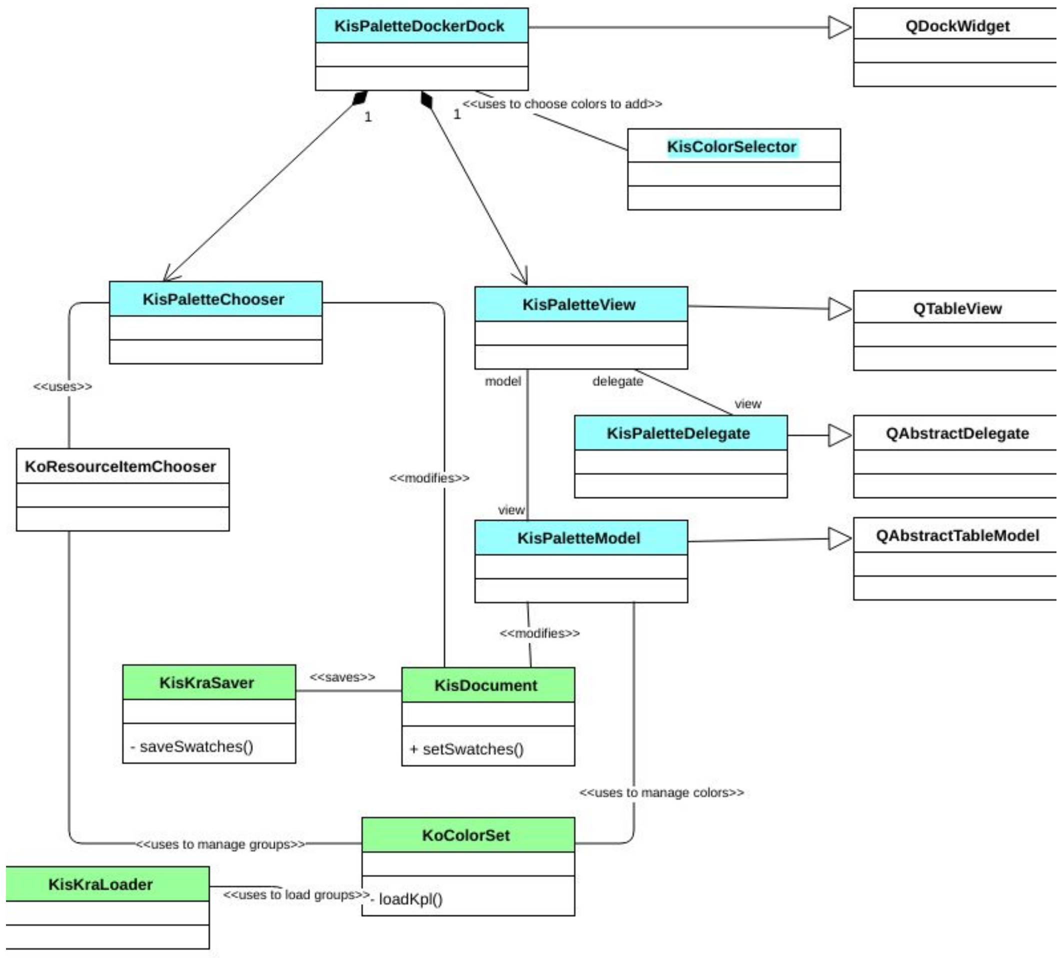
Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

UX Design

The docker is going to contain a matrix of colors and other GUI elements. Users can click an empty spot in the matrix to add the current foreground color to it, drag and drop to change its location or to switch it with another color. Users can right click an empty spot to choose a color from a color selector to add to it. The color selector is going to be a popup or a dialog. It will be using KisColorSelector, which is a class that currently belongs to the plugin “advanced color selector.” I want to export it so that it can be used but other plugins. Users can right click a spot with a color to choose to delete the color, to change it, to switch it with another color, or to rename it. Below the matrix, there is going to be a combobox from with users can choose from groups of colors available to the painting. There are button beside the combobox to add, delete, rename or export groups.

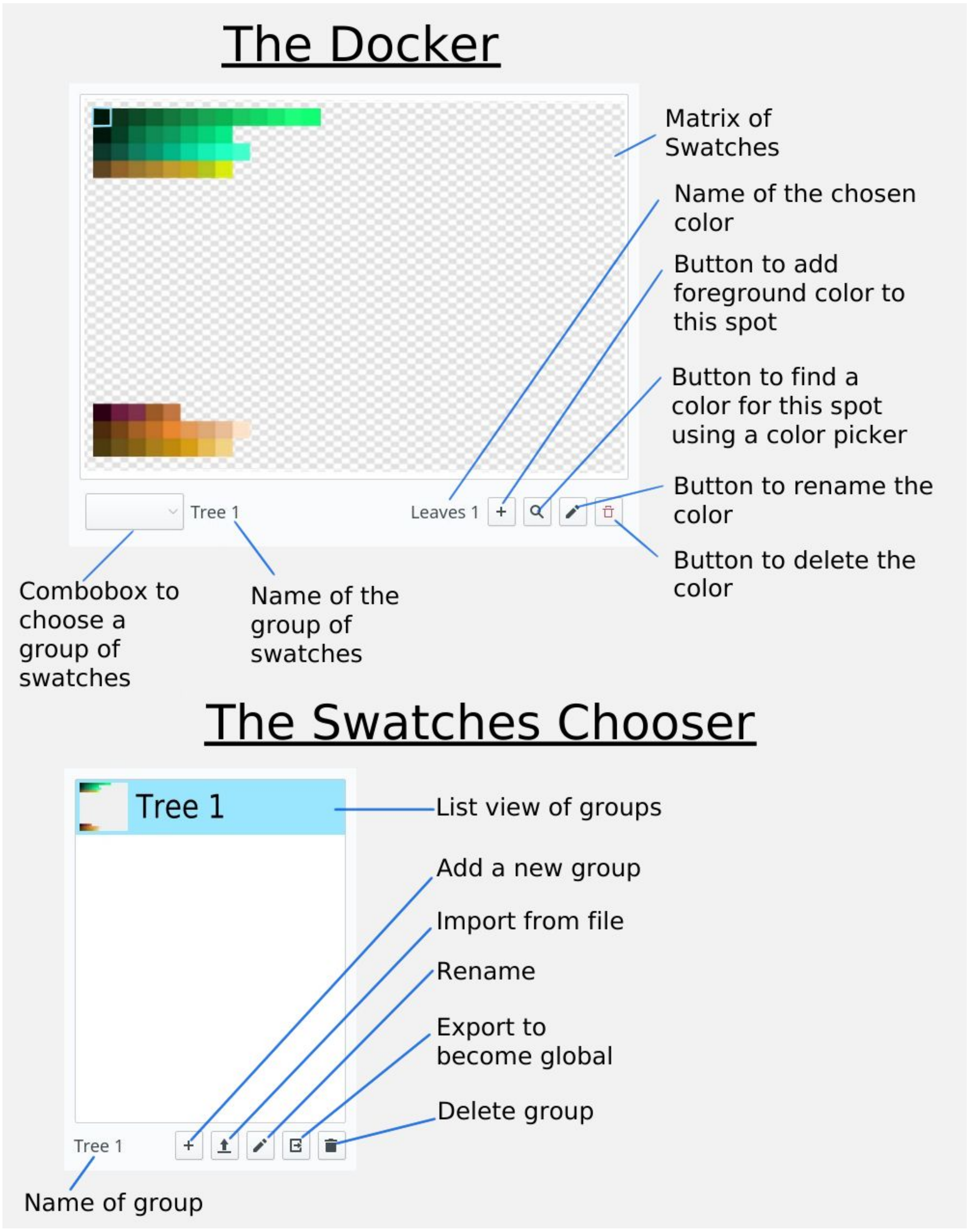
Class Diagram

Below is a simplified class diagram showing only how important relevant classes that belong to the plugin are related to each other. The methods that are mentioned are the methods that I believe will be added/modified for now. Green classes are classes that currently exist but will be modified, and blue classes are new classes that are going to be added.



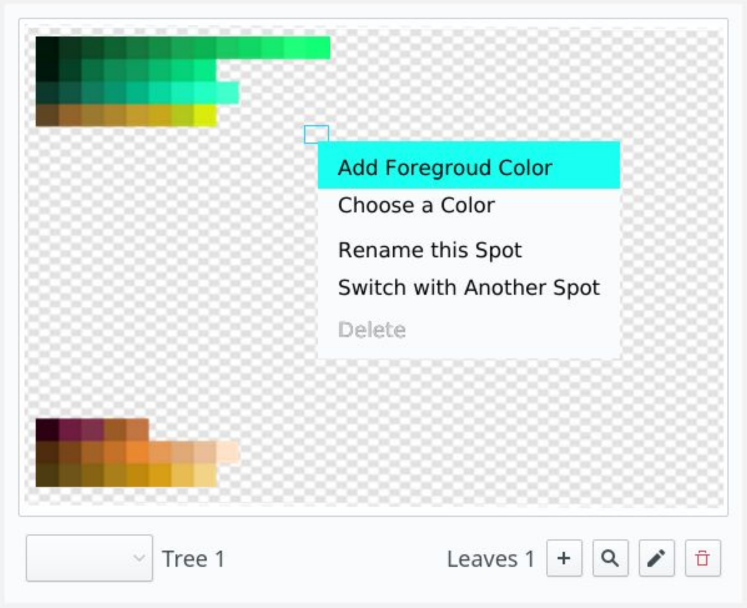
Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

Mock-ups

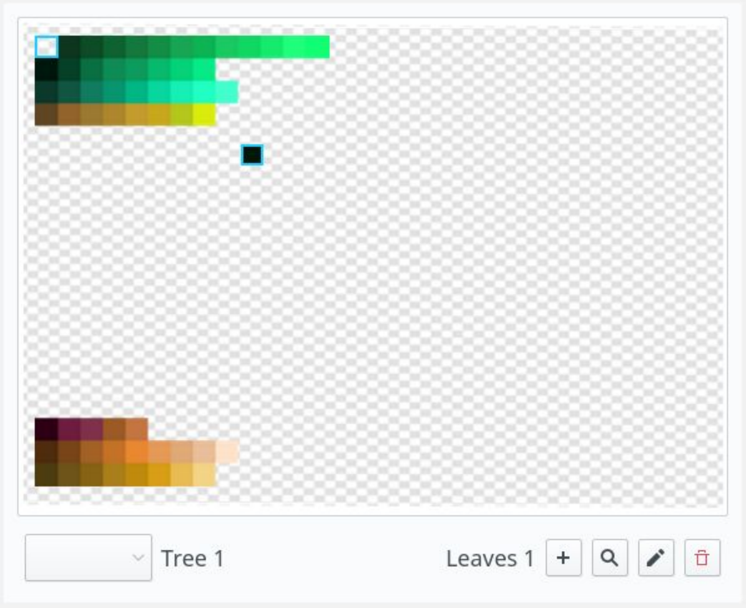

The Docker
Matrix of Swatches
Name of the chosen color
Button to add foreground color to this spot
Button to find a color for this spot using a color picker
Button to rename the color
Button to delete the color
Tree 1
Leaves 1
Combobox to choose a group of swatches
Name of the group of swatches
The Swatches Chooser
Tree 1
List view of groups
Add a new group
Import from file
Rename
Export to become global
Delete group
Tree 1
Name of group

Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

Right Click an Empty Spot



Drag and Drop to Arrange Colors



Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

Timeline

Week 1 (May 14 - May 20)
Finalize the design of the GUI.
Have non-animate GUI ready.
Have the GUI of the chooser for groups of swatches ready.
Have GUI of right click menu ready.
Write documentation to indicate the usage of each GUI element.
Start to implement the model.
Write unit tests for the model.

Week 2 (May 21 - May 27)
Finish the model. Comment the code.
Test and debug the model.

Week 3 (May 28 - Jun 3)
Start to implement user input handling of the parts out of the matrix, such as the combo box for choosing a group of swatches, buttons for adding a group, renaming a group, exporting a group to be global, etc.
Update documentation to reflect any usage changes during implementation.
Write unit tests for handling of the list of groups.

Week 4 (June 4 - June 10)
Continue to implement user input handling of the parts out of the matrix.
Test and debug the handling the list of groups.

Week 5 (June 11 - June 17)
Finish the user input handling of the parts out of the matrix.

Week 6 (June 18 - June 24)
Start to implement input handling of the matrix, including adding a color by clicking an empty place, drag and drop the color, drag and drop the color to an icon of trash bin to delete it, right click menu and dialogs used to add, delete, switch and rename colors, tooltips of colors, etc.

Week 7 (June 25 - July 1)
Finish implementing input handling of the matrix.
Document the input handling of the matrix.

Week 8 (July 2 - July 8)
Start to implement file handling, including handling individual palette files and global configuration files.
Start to write unit test for file handling.

Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

Week 9 (July 9 - July 15)
Finish file handling unit tests.
Finish the file handling. Comment the code.
Prepare documentation section for file handling.
Test and debug the file handling.
Add the docker to Krita's plugin system.
Design the GUI.

Week 10 (July 16 - July 22)
Start to test and debug the docker as a whole.
Start to put all documentation sections together.
Update documentation to reflect any usage changes during debugging.

Week 11 (July 23 - July 29)
Finish testing and debugging.
Put all documentation sections together and finalize it.

Week 12 (July 20 - Aug 6)
Buffer week in case any unexpected event happens.

Michael Zhou
simeixh@gmail.com
+1 (805) 293-7240

About me

IRC nick: simeir
Email: simeirxh@gmail.com
Phone: +1 (805)293-7240
Prior contributions:
https://phabricator.kde.org/D10061
https://phabricator.kde.org/D9999
https://phabricator.kde.org/D10572
https://phabricator.kde.org/D10157
https://phabricator.kde.org/D10103

I am a student from China who is currently a 2nd year undergraduate Computer Science student in University of California, Santa Barbara. I started digital painting in high school and Krita is the tool that I like best. Currently, I am practicing anime-style illustrating.

In China, Japan and South Korea, where anime is more popular, lots of digital artist are using this style. Currently, Painter Tool SAI is the most popular one, and it does have some features that are more handy for the anime-style. This proposal is inspired by the swatches of SAI. At the same time, Krita also comes with functions that are better than SAI.

As a digital painter myself, I hope I can have a better software to use when painting. As a user of the anime-style, I think I can bring the ideas from the East to improve Krita's development. As a supporter of freeware, I hope I can provide digital artists who like the anime-style a better free alternative. Therefore, I started contributing to Krita, and became a developer in February.

The swatches docker is a relatively large one in all the changes I want to make to Krita, and I am still familiarize myself with Krita. Therefore, I hope to utilize the opportunity of GSoC to get help from the mentors so that I can have a better chance to realize my idea.

After GSoC, I'll continue to contribute to Krita. There are still a lot of things I want to do. For example, how Krita adds a layer to a painting with a lot of group layers is kind of inhumane :). Changing it will be the first thing I am going to do after GSoC, or maybe before that.

I'm not submitting proposals to other organizations.

I don't plan on working on anything this summer other than GSoC. If I couldn't get into it, I will still try to implement the idea myself.