System and Unit Test Report

System Test Scenarios

Sprint 1

Sprint 1 User Story 1:
As a general user, I want an application with a simple yet visually pleasing UI.

Scenario:

1. Open the app
2. User should see a simple yet visually pleasing UI

Sprint 1 User Story 2:
As a traditional artist, I want to be able to use my camera and gallery to observe colors of what I am drawing, and have easier reference of what colors I need.

Scenario:

1. Open the app
2. Click on the add image button at the bottom right of the current image
    - Select “camera”
        - Take a picture
        - Approve the picture
    - Select “gallery”
        - Select a picture

Sprint 2

Sprint 2 User Story 1:
As a general user, I want to be able to view the photo I've taken on the main color picking page to select colors from

Scenario:

1. Open the app
2. Click on the add image button at the bottom right of the current image
    - Select "camera"
        - Take a picture, approve the picture
    - Select "gallery"
        - Select a picture
3. View image on the color picker fragment

Sprint 2 User Story 2:
As a general user, I want to be able to tap on the color I want to pick anywhere the image is on the screen, whether using the live camera or a static image.

Scenario: (Note: Live camera not implemented)

1. Start app. On the main screen, tap or drag on the image.
2. Check debug log for coordinates to verify the top left is 0, etc.
3. Compare displayed hex with online sites to see that the color displayed looks the same.

Sprint 2 User Story 3:
As a digital artist, I want to see the RGB/HEX/HSV values of a color that I pick.

Scenario:

1. Open the app
2. On the color picker fragment, select a color from the image
3. Click on the info button

4. View the RGB/HEX/HSV values
    - Compare to values on an online converter to confirm they are the correct values

Sprint 3

Sprint 3 User Story 1:
As a long-term user, I want to be able to save colors for later usage in a library directly on the app.

Scenario:

1. Open the app
2. Have a color to save
    - Select a color on the color picker fragment
        - Click on the save color button in the bottom right corner
        - Select “save color” from the dialog menu
    - Create a new color on the edit color activity
        - Click on the save color button in the bottom right corner
3. Click on the Saved Colors fragment to view color you just saved

Sprint 3 User Story 2:
As an artist, I want to be able to create and save palettes for the colors I chose, for easy drawing reference.

See (Sprint 4 User Story 1)

Sprint 3 User Story 3:
As a frontend designer, I want the ability to export RGB, Hex, and HSV values from the app to more easily use these colors in my personal projects.

Scenario:

1. Open the app
2. Select a color from the image or select a color from saved colors/a palette
3. Click on the copy button next to the desired value: RGB, HEX, HSV
    - A notification will pop up to notify you that the value has been copied
4. Exit the app

5. Paste the copied value in desired location

Added From Backlog:
As an artist, I want to be able to edit my chosen color and save it for further use.

Scenario:

1. Open the app
2. Select a color from the image or select a color from saved colors/a palette
3. Click on the edit color button (shaped like a pencil)
4. Select a color editing mode
    - Click on the button in the top right color. It displays the current mode: RGB or HSV
5. Change the color
    - Drag the sliders or click on a value name to input a value
    - View the edited color on the top right next to the original color
    - If you dislike the new color, select the reset button on the bottom left to return the new color to the original color
6. Click the save button on the bottom right of the screen to save the new color

Sprint 3 User Story 4:
As a user of the app, I want to keep my chosen photo and recent picked color info on the color picker fragment when switching fragments.

Scenario:

1. Open the app
2. Select an image
3. Pick a color from the image
4. Change fragments
5. Return to the color picker fragment
6. View that the same image and color are displayed

Sprint 3 User Story 5:
As a user of the app, I want to know the name of the color I've selected

Scenario:

1. Open the app. Tap or drag on the image.
2. Verify that the TextView changes from the previous text to the name of the selected color.

Sprint 4

Sprint 4 User Story 1:
As an artist, I want to be able to create and save palettes for the colors I chose, for easy drawing reference.

Scenario:

1. Open the app
2. Pick a color from the image
3. Click on the save button in the bottom right
4. Select either “create new palette” or “save to palette”

Sprint 4 User Story 2:
As our teacher/TA, I want a complete and clear documentation of the project code so I can get a clear understanding of the project code and what was worked on.

Scenario:

1. Open the git hub, view the documentation

Sprint 4 User Story 3:
As a user, I want to see the palettes that contain the selected color from the Color Info Activity. (User Story scrapped)

Scenario (If user story had been completed):

1. Open the app
2. Pick a color from the image
3. Click on the color info button
4. View the palettes that the color is in

Sprint 4 User Story 4:
As an artist, I want to be able to generate color harmonies from a color to see what other colors would look good with it.

Scenario:

1. Open the app. Select the Color Info button. Select the View Harmonies button.

2. Verify that the palettes include the selected color.
3. Verify that the palettes have colors in both directions around the selected color.
4. Check the debug log to see that the exact values being calculated are as expected.
5. Select a palette, and see that some info about the colors in it are displayed.

Sprint 4 User Story 5:
As a user of the app, I want an aesthetically pleasing loading screen.

Scenario:

1. Open the app
2. View the loading screen

Unit Tests

HarmonyGenerator.java Equivalence Tests:

A harmony can be generated for any color. Colors have three components: Hue, Saturation, and Value. The valid hue range of a color is 0 through 360, inclusive. Saturation and value can be 0.0 through 1.0, inclusive.

1. The analogous scheme only modifies the hue. The first test can be a color with a hue value of 180. Analogous colors should be generated both above and below the chosen color's hue.
2. The second test should be a color near the top of the hue range, such as 350, so that some of the generated analogous colors wrap around from 360 to 0.
3. The third test should be a color near the bottom of the hue range, such as 10, so that some of the generated analogous colors wrap around from 0 to 360.
4. Split-Complementary and the two Evenly Spaced schemes also only modify a color's hue, so the same tests have covered them.
5. The monochromatic scheme modifies only the value. Wrapping around wouldn't make sense, but we can still verify that the program handles large and small values well by displaying valid colors.
    - Repeat steps 1 to 3 but instead of hues and checking the analogous scheme, use values of 0.5, 0.9, and 0.1 and verify results in the monochromatic scheme.

ColorPickerFragment.java

- Possible inputs: non-transparent colors, partially transparent colors, fully transparent colors

Equivalence Class	Description	Representative	Observed Output
$E_{Non-transparent}$	A non-transparent color is picked from the image	Clicking on the default dropper in the yellow/orange section	The color Freesia is picked and displayed with its HEX value and name

$E_{Partially\ transparent}$	A partially transparent “color” is picked from the image	Clicking on the borders of the image where colors converge or on a different image	The color without transparency is picked and displayed with its HEX value

$E_{Fully\ transparent}$	A fully transparent “color” is picked from the image	Clicking on the checkerboard “transparency” around the dropper	The color Black is picked and displayed with its HEX value

EditColorActivity.java Equivalence Tests:

- updateSeekbarsHSV(int hue, int saturation, int value) and
    - Hue should be in the range (0,360)
    - Saturation and value should be in the range (0,100)
    - Equivalence classes and test case representatives:

Equivalence Class	Description	Representative	Observed Output
$E_{HSV\ in\ desired\ range}$	All of the inputs are greater than or equal to 0 and less than or equal to the max desired input	Hue = 100, Saturation = 50, Value = 50	Seekbars update to the input
$E_{HSV < 0}$	One or more of the inputs are less than 0	Hue = -1, Saturation = 50, Value = 50	Seekbars update to Hue = 0, saturation = 50, value = 50 (The value < 0 will be set as 0)
$E_{HSV > desired\ input}$	One or more of the inputs are greater than the max expected value	Hue = 100, Saturation = 200, Value = 50	Seekbars update to Hue = 100, Saturation = 100, Value = 50 (The value > max will be set to max)

- updateSeekbarsRGB(int red, int green, int blue)
    - Red, green, and blue should be in the range (0, 255)

Equivalence Class	Description	Representative	Observed Output
$E_{RGB\ \ in\ desired\ range}$	All of the inputs are greater than or equal to 0 and less than or equal to 255	Red = 100, Green = 100, Blue = 100	Seekbars update to the input
$E_{RGB < 0}$	One or more of the inputs are less than 0	Red = -1. Green = 100, Blue = 100	Seekbars update to Red = 0, Green = 100, Blue = 100 (The value < 0 will be set as 0)
$E_{RGB\ > desired\ input}$	One or more of the inputs are greater than 255	Red = 300. Green = 100, Blue = 100	Seekbars update to Red = 255, Green = 100, Blue = 100 (The value > max will be set to max)

Color Info Equivalence Tests:

- Possible scenarios: Calling color info from color picker fragment, from a saved color, from a color in a platte, or from a color in a harmony

Equivalence Class	Description	Representative	Observed Output
$E_{color\ picker\ fragment}$	Color Info Activity was started from the color picker fragment	Sotek Green selected from the default image	Color info correctly gets the color, its name, and its values which are copyable
$E_{saved\ colors}$	Color Info Activity was started from the saved colors fragment	Figue selected from saved colors	Color info correctly gets the color, its name, and its values which are copyable
$E_{palette}$	Color Info Activity	The dummy	Color info retrieves

	was started by clicking on a color in a palette	palette “Three Colors” was selected, then highlighter	the wrong color (gets a color from saved colors). This is a bug
$E_{harmony}$	Color Info Activity was started by clicking on a color in a harmony	Color harmonies generated by Sotek Green. Analogous harmony selected, color 3 is then clicked	Color info retrieves the wrong color (gets a color from saved colors). This is a bug

ColorDatabase.java

- Possible scenarios: Called from color picker fragment when saving to saved color or palette depending on choice in the save dialog, from SavedColors where we get the info from the database to populate the list, from ColorInfoActivity where a color’s name, HEX, RGB, HSV values are displayed, from EditColorActivity when a new color is saved from the new slider values.

Equivalence Class	Description	Representative	Observed Output
$E_{color\ picker\ fragment}$	ColorDatabase called when saving color from image	Vivid Imagination is picked, and saved to save color	ColorDatabase correctly saves the color Vivid Imagination into the SavedColor fragment
$E_{SavedColors}$	ColorDatabase called from saved colors to display saved colors	Vivid Imagination from color picker fragment was saved	ColorDatabase correctly displays Vivid Imagination from the color picker fragment and also new colors made from the EditColorActivity
$E_{palette}$	ColorDatabase called when saving to palette from the	Dummy palettes no real call to database	Dummy palettes so no real call to database

	saved dialog that appears in color picker fragment		
$E_{ColorInfoActivity}$	ColorDatabase called when viewing a color, details are given like NAME, HEX, RGB, HSV	Vivid Imagination from the SavedColor fragment is clicked	ColorDatabase correctly displays name, HEX, RGB, HSV of Vivid Imagination
$E_{EditColorActivity}$	ColorDatabase called when saving the new color created from the sliders	Dead Flesh is created from the slider, original color VIvid Imagination, saved button is then clicked	ColorDatabase correctly saves and displays the new color Dead Flesh in SavedColor.