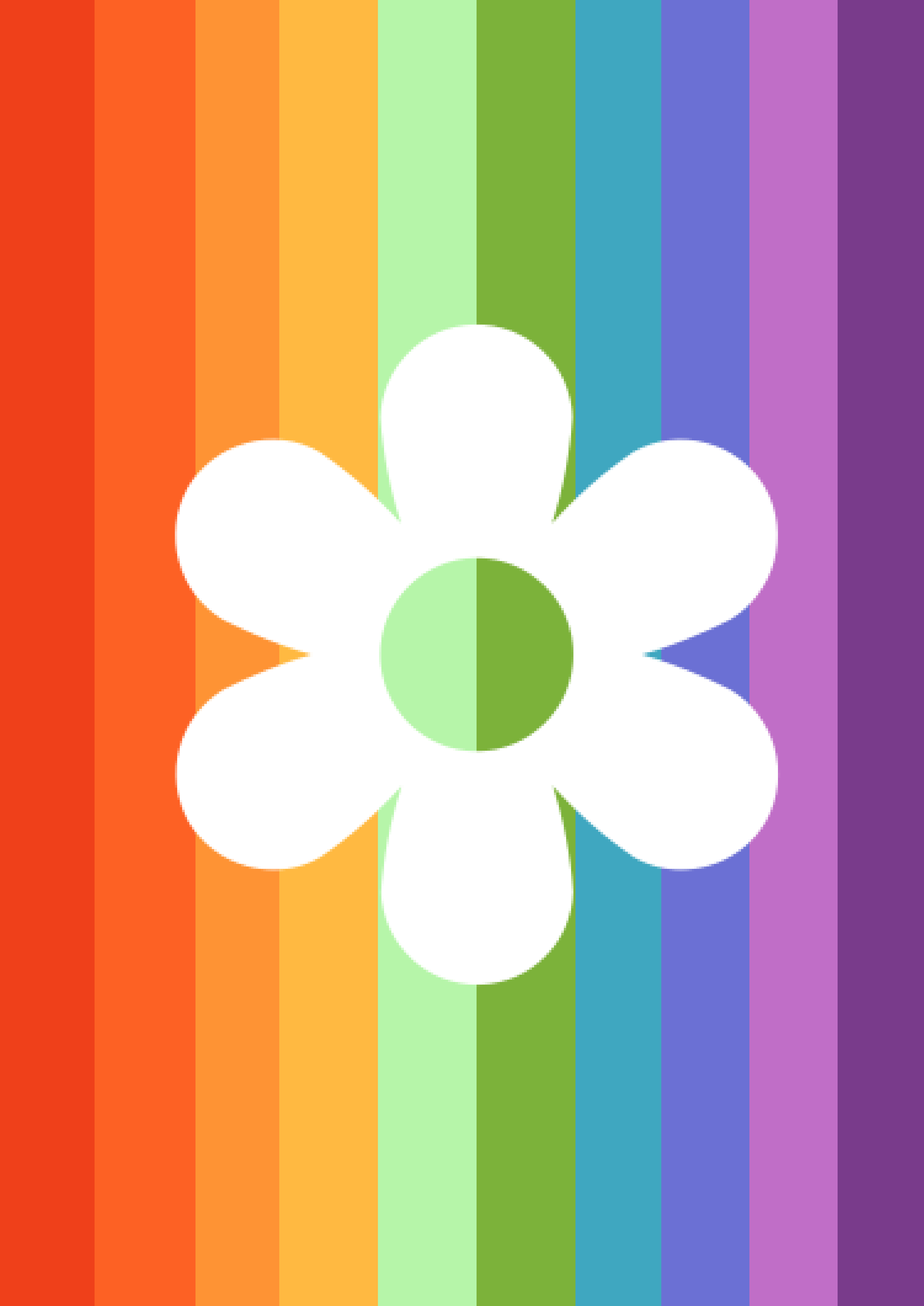


Project Report - **Drippy Flower Field**

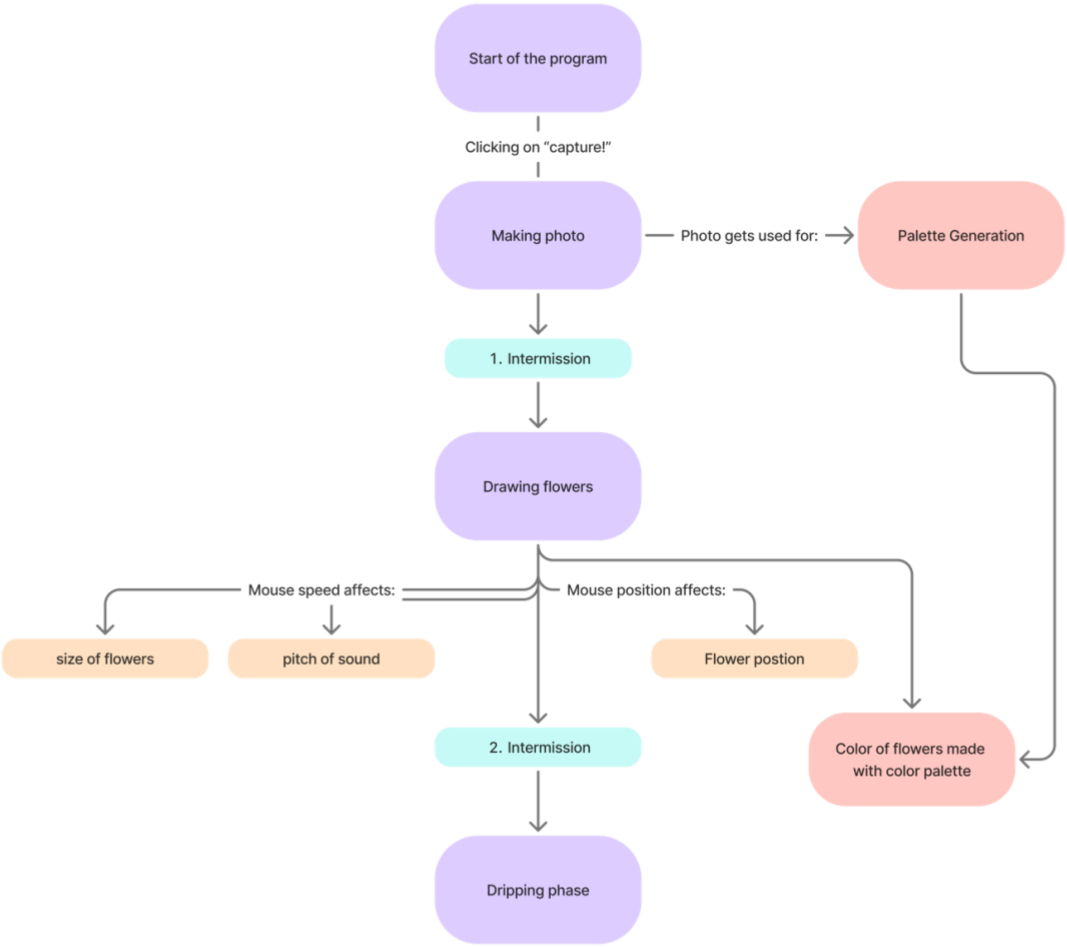
Introduction

The motivation for this project was to create a program that inspires users to create simple but pretty art. We combined two ideas for an interactive art project, one of drawing a flower field and another of drip art. The result is a program where users create a quick field of flowers and then enhance/destroy it through dripping and smearing. The work was divided among the three of us as follows: Sidra worked on the fundamental drawing function, camera input and flower designs, Kathi worked on the dripping, color palette generation and assembly of the code pieces, and Sophie worked on the flower drawing function and sound implementation. The tools required were Processing and three additional libraries for the camera, sound and sorting objects in a class. The project is interactive because the user is the creative force behind the program and there are multiple input methods that influence the outcome. Our inspiration for this project comes from interactive digital installations using flowers, drip paintings where through destruction a painting can be reinterpreted and finally, algorithms like Adobe Color, that extracts color palettes from images much like what we were able to implement in our project.

Interaction Concept

The focus of this project is user lead creation. Different types of interaction, video and mouse movement, play a big role in the output of the program. The system responds to the user in real time. We designed this program for the user to feel like they are creating art through interacting with the screen. This should be a fun and enjoyable experience, that is reflected by the bright color palettes.

Ways of interaction in DFF



Algorithmic Art Generation

When the user drags the mouse across the screen, one of five flower types get generated. The flowers are saved as <Flower>Objects in an Arraylist that gets redrawn each frame. This ensures that the flowers don't change color or disappear, between the drawing and the dripping steps for example. The color palette is assembled from the five most dominant colors in the image that the user takes at the beginning of each round. For every pixel, an object of type <Pixelcolor> is created with the parameters, hue, saturation, brightness and frequency, which is saved in the Arraylist "pixelcolor." If the color or a similar color, calculated in the variable "dist", is already in the Arraylist "pixelcolor", it doesn't get added, rather the frequency is increased by one. After all the pixels in the image have been processed, the five colors with the highest frequencies are added to three different arrays that represent the hue, saturation and brightness of each color and are referenced in the flower class as fill colors. The drips in the third phase are made up of multiple overlapping circles. They are objects of type <Drip> saved in the Arraylist "drips". This is necessary to create a more naturally and flowing drip. For a more realistic look, the drips shift slightly to each side during the flow. Each drip has a maximum y-position and as they near this maximum, the

circles increase in size. The audio element is a prefabricated sound that during flower generation changes pitch depending on the speed of the mouse.
Our program supports endless variety through randomness and response to user input. The position of flowers is based on mouse position with slight randomness in the x and y directions. The type of flower generated is totally random, however the size depends on the speed of the mouse. In the dripping phase, the amount of dripping is also determined through user interaction. Finally, the color palette is changed each round depending on the image that the user takes. These features make it impossible for an outcome to be repeated.

Implementation of User Interaction & User Input

Our program uses the input types of mouse, keyboard and video. In the first phase of the program, users see video camera input and can click the capture button to trigger the takePhoto() function. The current camera display will then be saved and cropped to the 400x400 pixel square that was highlighted through the UI as the space in which users should hold up their chosen colors. This cropped image is then used for the color palette generation. In the second phase, through clicking and dragging across the canvas, the user can create flowers. As stated previously, the position of the flowers is based on the mouse position. The speed of the mouse influences the size of the flowers and pitch of the sound. Dripping in phase three is also triggered by mouse clicks and dragging. Finally, at any time, the user can start a new round by pressing ‘s’ on the keyboard. This saves the current canvas as a .jpg image named using the time and date. All Arraylists are cleared, the counter is set back to 0 and the camera is restarted to set up for the new round.

All code was written in Processing and we used three libraries. java.util.Comparator is used for comparing and sorting <Pixelcolor> objects in the Arraylist “pixelcolor” by frequency. processing.video.* is used for camera input and processing.sound.* for playing sound.

Technical Challenges & Solutions

One technical barrier that we encountered is that the initial loading of the camera takes a couple seconds. We did not come up with a solution for this because there are no issues loading the camera between each round. After the user saves their image, the camera appears immediately for the next round to start. In an exhibit, the code would be running endlessly so the initial lag of the camera wouldn’t be an issue. Additionally, the bad quality of the camera was creating color palettes that were bland and dark. We changed the colorMode from RGB to HSB so we could increase the saturation by 1.5 and the brightness by 1.2. This ensures that our program always produces colorful and lively color palettes.

During the creation of this program, we learned many lessons. For example, the using different libraries expands the possibilities of what you can do and also help with difficult tasks, such as comparing objects in a class. We also learned to not be afraid to have high ambitions because there will be a way to realize it. Initially, the idea of palette generation through camera input seemed daunting but in the end, it worked really well and added a new level of interactivity.

Finally, Arraylists are extremely useful and versatile when working with many objects, especially when the number of objects changes each round.

Conclusion

We imagine our program to be displayed in an art exhibition on a screen, with either mouse or touchscreen interaction. One button would save the picture and start a new round. The camera would be beside or above the screen. Nearby, would be a box with colorful items, such as children's toys, plastic flowers, books, magazines etc to inspire people when they are creating their color palette. A separate screen would display previously saved images that visitors can browse.
Some things we would like to enhance is the sound, for example different sounds depending on the dominant hue of the photo. Also, potentially incorporating hand tracking through the camera instead of moving the mouse for a mor engaging experience.