Harry Gong, Grace Joseph, Phoebe Lee

Figma Lite

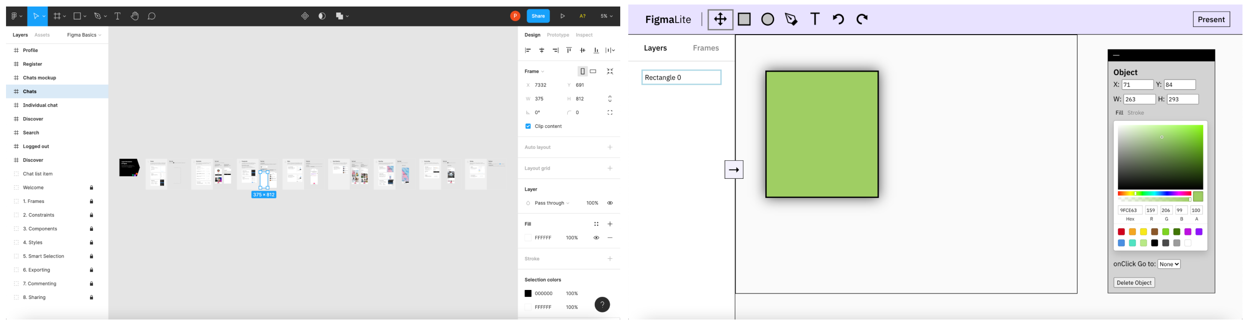
Site URL on Netlify: https://practical-easley-ce87c8.netlify.app/
Github repository: https://github.com/harry8698/SSUI-Final-Project

Introduction

Figma Lite is a prototyping tool that supports different drawing modes along with additional features that facilitate wireframing and presentation. The features and the overall design of Figma Lite are heavily inspired by a widely used online prototyping tool Figma. We utilized code from homework 5 to support basic drawing modes and added text mode, layer/frame panel, separate command palette for editing, and presentation mode to create a more comprehensive tool that resembles Figma. The overall layout and design of Figma Lite is also similar to that of Figma so that the users can effortlessly familiarize themselves with the new prototyping tool.

Related Work

As previously mentioned, we aimed to build a prototyping tool based on features provided in Figma, so all features supported in Figma Lite are also available in Figma. We also made the design and general layout of Figma Lite similar to that of Figma as shown below:



Additional features supported by Figma can be found in the following link:
https://help.figma.com/hc/en-us/categories/360002051613-Get-started#Files-and-projects

Installation Guide

Prerequisite: install git, npm

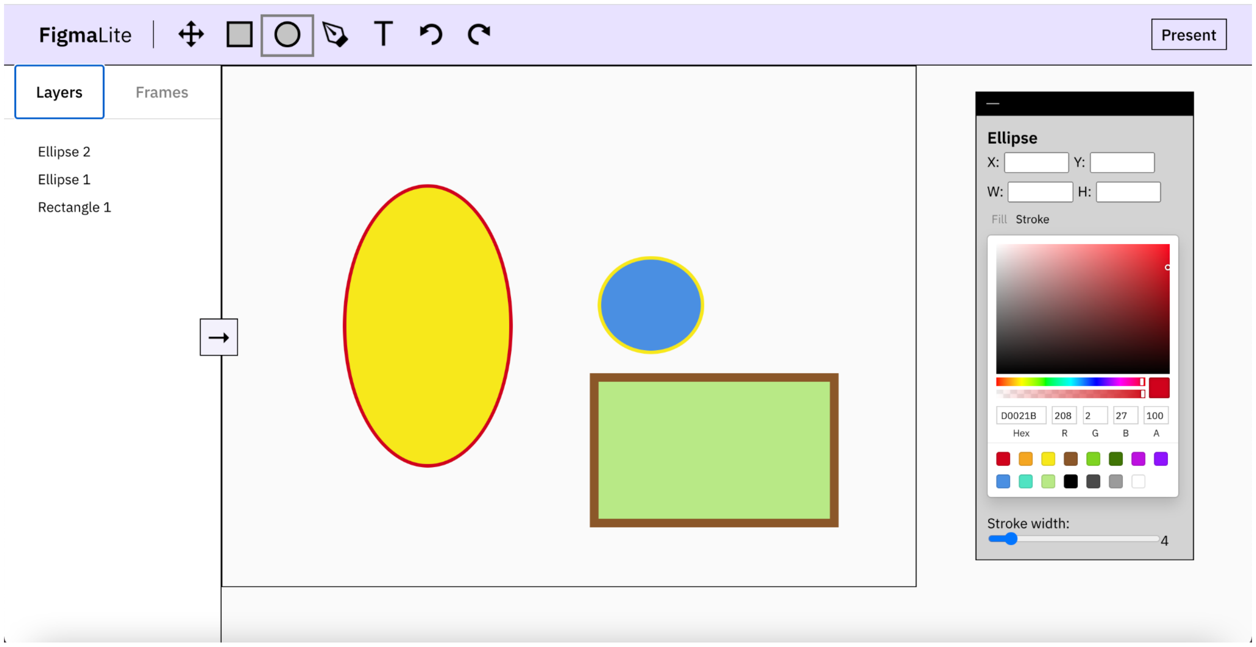
- Pull the source code from GitHub at https://github.com/harry8698/SSUI-Final-Project

Harry Gong, Grace Joseph, Phoebe Lee

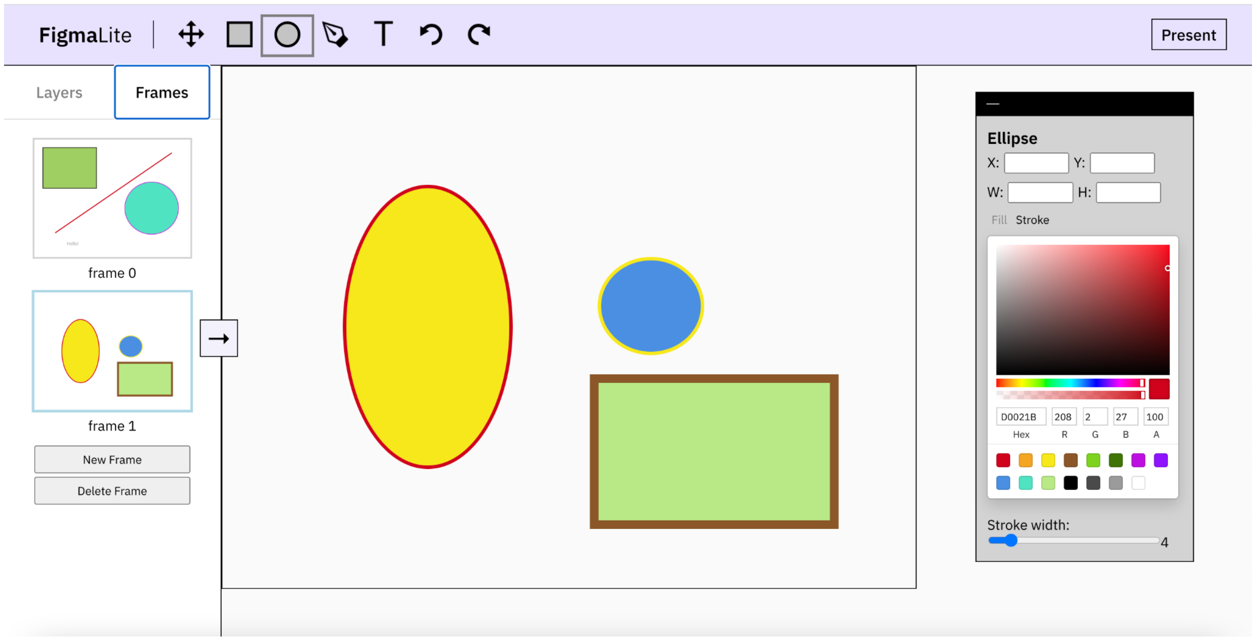
- Go under directory figma-lite/
- Run npm install to install packages
- Run npm start to start the app

Example

An example screen with layers tab open:



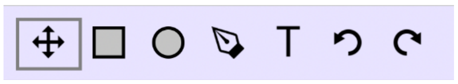
An example screen with frames tab open:



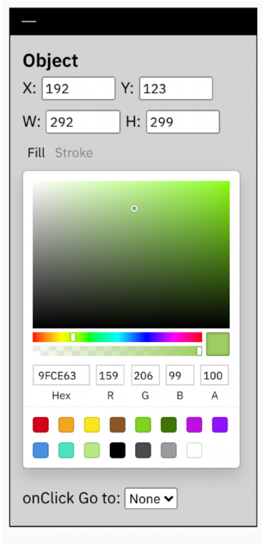
Harry Gong, Grace Joseph, Phoebe Lee

User's Manual

1. Mode/Undo/Redo Bar: Located in the navigation bar, users can switch between different modes (rectangle, ellipse, line, and text) to create different objects. In addition to switching between modes, users can undo and redo their actions using the arrows to the right of the 4 modes regardless of which mode they are in. Depending on the mode selected, the Command Palette will change contents.



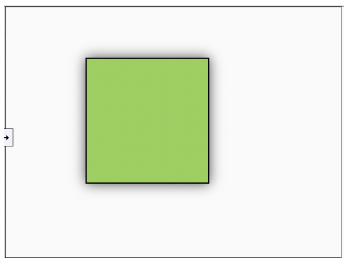
2. Command Palette: This component allows the user to customize the properties of the object they are creating. Depending on the mode selected the contents of the palette will change. All objects can have their position changed via the X and Y coordinate input fields at the top of the palette and specifically the rectangles and ellipses can change width and height depending on the corresponding width and height input fields. When an object is initially created in the workspace, the aforementioned input fields will have default values corresponding to the initial position and size of the object created. When a user changes the values in these input fields and presses "Enter", the object will automatically reposition and/or resize in the workspace based on the inputs. For text objects, below these input fields, there are additional dropdowns that allow the user to determine the font and font size of the text object being created or modify the text object selected. If the user is in the rectangle or ellipse mode, then below the input fields there will be tabs for setting the Fill and Stroke properties with a corresponding color picker. For text objects, only the Fill property can be set and for line objects, only the Stroke property can be set. When an object is selected or about to be created, users can use the slider on the color picker to change colors and use the hue selector to change the tint/shade of the color selected in the slider. They can also choose among the preset colors below the sliders or input hex / rgba values to specify a color. When an object is being created, its fill and/or stroke will be set to the default color shown on the picker and similarly when an object is selected the color picker values will reflect the colors of the object selected. For rectangle, ellipse, and line objects, below the color picker on the Stroke tab, there is a slider so that users can specify the stroke width for the object. Similar to the other fields, if an object is selected, by default the stroke width slider will be set to the value of the object's stroke width and any changes made to the slider will be reflected in the stroke width of the

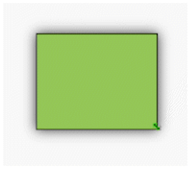


Object
X: 192 Y: 123
W: 292 H: 299
Fill Stroke
9FCE63 159 206 99 100
Hex R G B A
onClick Go to: None

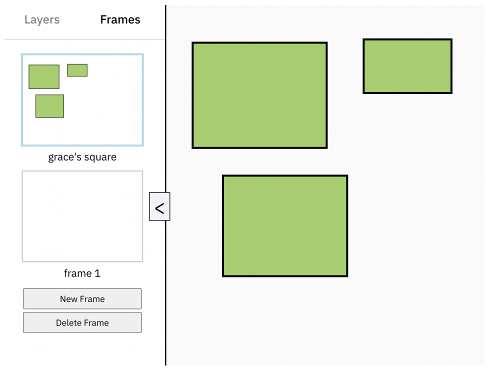
selected object. When the user is in the selection mode and has selected any type of object, at the bottom of the command palette is the onclick interaction dropdown which allows users to add an onclick interaction to the object selected. The dropdown's contents include all the frames created by the user. By default the dropdown is set to "None" indicating that there is no interaction associated with the object, but if a user selects one of the frames in the dropdown then an onclick interaction is set for the object so that in presentation mode, clicking on that object will send the user to the frame indicated in the dropdown. If the user is in selection mode, below the onclick interaction dropdown is a button allowing for the user to delete the object selected. It should be noted that the command palette is both collapsible and moveable, meaning that the user can toggle whether or not the palette's contents are on display using the +/- button in the black bar on top of the palette and that the user can move the palette around the window.

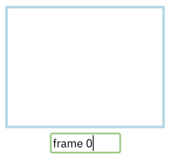
3. Workspace: This is where the user can create and edit objects/layers. The workspace is composed of frames that the user can create and delete. The user can create different objects/layers on each frame. Objects cannot be created outside of the workspace and the user can customize the contents of each frame in the workspace through the mode/undo/redo bar, the command palette, and the layer/frame panel. The contents of the workspace are presented in presentation mode. In select mode, any object selected will have a drop shadow and can be resized by hovering over the edges and then dragging.

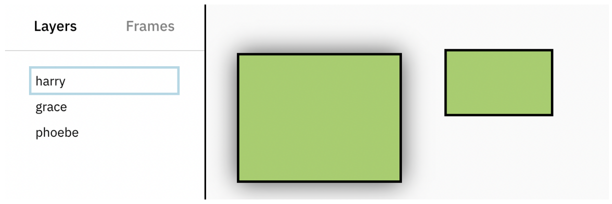




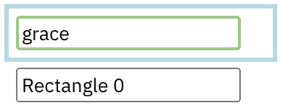
4. Layer/Frame Panel: This component consists of tabs that allow the user to switch between the layers and frames palette. In the frames palette, users can see all their frames listed and rename them by double clicking on each frame's title. When a user selects a frame, it populates the workspace and they can make edits to the contents of the frame. Below the frames are buttons that allow the user to create new frames and delete the selected frame. Frames can also be reordered by selecting and dragging a frame to its intended position among all the frames. The layers palette operates similarly, showcasing all the layers (objects) created on the current frame in order of creation (with the most recently created at the top). Layers can be reordered through the select and drag to the desired



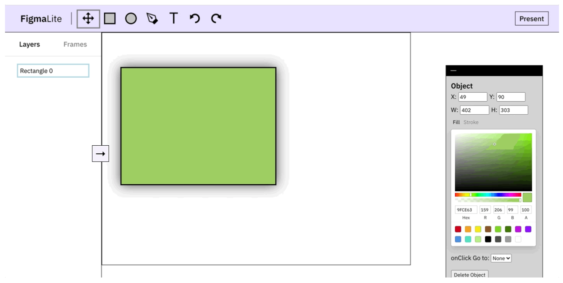




position, updating the order of the objects in the workspace. Additionally, layers can also be renamed by double clicking the title of the layer. As objects are created and deleted via the mode/undo/redo bar and command palette, the layers palette updates for that frame. It should be noted that this component is collapsible via clicking the arrow button on the right edge of the panel.



5. Presentation Mode: This mode is triggered by clicking on the "Present" button in the upper right corner. Once clicked a popup will appear with the current frame and its contents. The user can then use the arrows keys to shift between the frames, press on "Enter" to make the presentation fullscreen, and click on any of the objects in the frames with onclick interactions to go to the frames the interaction was set for. The user can press the "Esc" key to exit presentation mode and remove the popup.



Design Rationale

The design of our platform was heavily influenced by Figma *(hence the name Figma Lite)* in its layout of a navigation bar where the user can select the mode (rectangle, ellipse, line, text, and present), a layers palette on the left hand side, and a command palette where the user can customize objects (layers) based on the mode they are in. In contrast to Figma, we opted for a collapsible layers/frames palette and command palette and a moveable command palette because those were features we had found useful in other design tools such as Adobe Photoshop, XD, and Indesign. We wanted our users to be able to somewhat customize their layout and limit their cognitive load by being able to collapse the palettes and move the command palette around the screen.

We admired how Figma by default considered each object on its own layer, so we implemented that design pattern in our layers palette and allowed for reordering and renaming similar to the way Figma allows it for its layers. Figma also has the notion of frames, which are foundational elements that are composed of objects/layers. While Figma allows for these frames to all display simultaneously in one workspace, we opted to mimic Powerpoint in our implementation of frames – making them individual containers that a user can switch between. We made this design choice because it was much more straightforward to implement and would allow us to dedicate more time on allowing users to customize the objects that composed their frames rather than the frames themselves. That said, similar to Figma we decided to support reordering and renaming of frames. Additionally while Figma does not have explicit undo, redo, and delete buttons, we opted to include these buttons to limit the gulf of execution and make it more explicit to the user that they can perform these actions.

Figma also supports prototyping, allowing for users to add interactions to different objects in frames that navigate them to other frames. While Figma's implementation of this feature is far more advanced than ours, enabling users to create and drag arrows between frames to set an interaction, we wanted to add this prototyping feature and specifically allow users to create onclick interactions. In our implementation, we decided to add a section to the command palette when an object is selected that allows them to specify which frame the onclick would navigate them to in presentation mode.

Implementation

1. Text mode: As we utilized homework 5 for implementation of basic drawing modes and command objects, the implementation of text mode was done in a similar manner to the base code of homework 5. We defined a text object in 'Text.js' in the shapes folder so that it could be added to the svg layer and stored in the shapes array along with other objects. The parameters 'oldText', 'newtext', 'fontsize', and 'font family' were added to the shapes data to store additional information of the created text object and were utilized in the new command objects for text editing and updating shape. When the workspace is clicked in text mode, the 'AddShapeCommandObject' immediately gets triggered to create an empty text object with both oldText and newText values set to empty string. While the text object is selected, all keyboard inputs get stored in the newText value of the selected object and gets displayed on screen. Deselecting the text object will then finalize the text editing and trigger a creation of new 'AddShapeCommandObject'. The oldText value of the previously selected object will then be set to its newText value. Undoing the command will revoke the text value of the object back to what it was like before the editing. Selecting the text object in select mode and editing its text value will trigger the same command object and follow the same process. Updating other attributes - font size, font family, fill color - triggers creation of 'UpdateShapeCommandObject' and follows the same process as updating other objects on the workspace.

2. Command Palette: The code provided in homework 5 were wired into our implementation of draggable/collapsible command palette. To support more colors than we did in homework 5, we used React color library to add color palette for selecting and updating fill color and stroke color. When the object is selected, changing its position/width/height in the command palette triggers 'updateShape' function to update

the values in shapeData accordingly and creates an 'UpdateShapeCommandObject' to support undo and redo of the corresponding change.

3. Layers: Similar to Figma, the term 'layer' in Figma-lite means an individual shape like a rectangle or a text box, and the ordering of the layers is the ordering of the z-index of the shapes. In homework 5, the ids of the shapes drawn in the workspace are stored in an array, and the ids of newly drawn shapes are appended to the end of the array. When rendering the svg layer, we are iterating through this array so that newer shapes are displayed above older ones. Therefore, if we change the order of the elements in this array, the ordering of the shapes will also change. To implement the drag & drop feature in the left layers panel, we utilized the html drag events. First, the shapes array is filtered and reversed so that only the names of visible shapes are displayed and the order of the layers list reflects the order of the layers (the first layer is the layer at the top). We used a table to store these layers by using a tbody for each. Then we attach a callback function to the 'onDragStart' event on the tbody elements to store the id of the layer being dragged in a state. We are also attaching a callback function for the 'onDragOver' event for each layer that swaps its id with the id of the layer being dragged in the shapes array. Since the shapes array is a component state, when the swap happens, React will re-render the page, and the order of the layers will be modified accordingly. [note: due to unknown reasons, the color picker palette can sometimes cause reordering to lag]

4. Frames: We added a property to each shape object that stores the id of the frame it's in. When rendering the svg layer and the thumbnails in the frames palette, we are using this information to filter the shapes array and use it to draw the svg layers. Reordering in the frames palette is implemented the same way as the layers palette.

5. Resizing: Due to the limited timeframe, we weren't able to implement the selection handles for shape resizing. However, we found a library called 'interact.js' that could modify the cursor shape when it nears the boundary of the shape. The library provides resize move, start and end events and functions that allow us to attach callback functions to the listeners of these events. All we had to do was to integrate this library into SVGLayer.js and add a resizing state to avoid conflicts with the original shape moving functionalities.

6. Presentation Mode: The presentation mode displays all frames in the current order and also supports onclick interactions set by the user. When the user clicks on the 'present' button, a new svg container gets created and the current frame number gets initialized to 1, rendering the objects in the first frame. Moving to a different frame updates the current frame number and re-renders the objects accordingly. The object selection and editing

doesn't get triggered even if the objects are clicked as mouse click event handlers are tied to the 'workspace-svg' container. Triggering the onclick interaction updates the current frame number to the destination frame number, so the shapes on the destination frame get rendered into the svg container. Fullscreen and Popup modules provided in React full-screen library and React popup library were used to support the pop up screen and full screen mode.

Reference to external libraries:

- React color: https://casesandberg.github.io/react-color/
- Interact.js: https://interactjs.io/
- React popup: https://react-popup.elazizi.com/react-modal/
- React fullscreen: https://www.npmjs.com/package/react-full-screen

Future Work

With significantly more time, our system could be improved to include user authentication as well as support collaboration with real-time synchronous updates for users sharing a workspace. While difficult to implement, in the future a backend could be added to our system to support both of these features, making it more akin to Figma's current model. Additionally, once collaboration is supported, our platform could also include a commenting mode where users can post and reply to comments on specific frames.

More shorter-term updates to our current platform could include supporting toggleable alignment guides and gridlines, adding a polygon mode where users could indicate the amount of vertices for the polygon in the command palette, adding a dark mode, enabling users to customize the size of their workspace and frames, allowing users to move shapes between frames, and allowing for users to group objects/layers or multiselect. We could also use Bootstrap to better style the components. Some of these ideas would be heavily facilitated by the inclusion of more libraries and could be implemented in a shorter amount of time than the aforementioned backend related updates.