Project Description

Name: SimpleSheets

Description: A simple spreadsheet application, built on a custom, modular UI framework atop 112 graphics, that implements core spreadsheet features (namely, tabular editing, formulae, and simple data visualization) as well as several advanced capabilities such as scraping online tabular data and linear programming.

Competitive Analysis

The foremost competitors in the spreadsheet space are Microsoft Excel, Google Sheets, Numbers (Apple's spreadsheet application), and LibreOffice Calc. Excel, Sheets, and Calc are all based on a traditional spreadsheet design, where the entire canvas is filled with editable cells from the outset. Numbers, conversely, offers a more modular interface, wherein users are greeted with a blank canvas and can populate it with various entities, including tables (which are akin to the full-page sheets in other apps), graphics, text boxes, images, and other items. Since my goal is to focus on the core spreadsheet functionality of SimpleSheets, I've decided to adopt the Excel model instead of Apple's. This decision to focus on function over form is evident in several other design choices, such as opting not to include advanced styling and customization options (which are likewise a prominent feature of Numbers, though also present to a more limited extent in other spreadsheet apps) in order to focus on enhancing the application's core algorithmic feature set.

Like all major spreadsheet apps, SimpleSheets will give users the ability to enter, manipulate, and delete tabular data, adopting familiar interaction patterns already established by the major spreadsheet apps. It will allow users to visualize data, though with a less robust set of visualizations than is available in the likes of Excel or Numbers. It will allow users to formulaically fill cells using a formula language, though one obviously more limited in scope than the languages developed over years in other programs. SimpleSheets differentiates itself by offering the ability to natively import tabular data directly from the web (many spreadsheets recognize tabular data when pasted but do not allow the user to fetch web data directly from the app) and by building in linear programming functionality (which is available as an add-in in Excel but not supported by default).

Structural Plan

My project is built atop a custom modular UI system, meaning that I can easily compartmentalize UI components into their own self-contained classes. In general, each major UI element will be its own class, with core UI classes being housed in the `modular_graphics` module and major app-specific ones in `ui_components`. (The main spreadsheet app UI scene is contained in the main file, `SpreadsheetApp.py`.) I will also have separate modules for primary functional aspects of my app, including `formulae`, `data_visualization`, and the like. Within each module,

primary functionality will be contained within `__init__.py`, with supplemental files housing meaningful supporting functionality as needed (e.g., `data_structures.py` in `formulae` contains my implementations of stacks and "dependency graphs").

Algorithmic Plan

Custom UI Framework

Each UI element subclasses a `UIElement` base class, which provides the basic infrastructure for UI components (drawing, managing children, handling input events, etc.). Every element is initialized with an identifier, an (x, y) position relative to its parent, and arbitrary properties that define its appearance or behavior and may be updatable. Every UI element can populate itself by adding *child* UI elements. Parent elements add children relative to their own position (ensuring modularity); the `UIElement.appendChild` method then computes the child's absolute position on screen by adding to its relative position the position of its parent (since the root node starts at (0,0), this transitively ensures every element will be assigned the correct position). Note that, because the UI framework is modular, there is no need for any child or its parent to access these absolute positions; they are merely necessary for the top-level `App` class to process and draw children correctly. Indeed, `App` draws the UI by calling the `draw` method of all child `UIElement`s, which each in turn call the `draw` method of their own children. Each time, `draw` passes to the child elements a `RelativeCanvas` that maps local coordinates to absolute ones, so that children can draw relative to their own position rather than the canvas as a whole. Most children, however, will simply render their own children; `modular_graphics` exposes a set of *atomic* elements (e.g., `Rectangle`, `Text`) that override the `draw` method to make calls to the `RelativeCanvas` directly. This allows for complex UI elements to be built up simply by composing existing ones.

Formulae

- Formula Parsing: I strip all spaces and capitalize the input, then (using a generator) continually find the next "token" in the string, which is one of `(`, `)`, or `,`. I also maintain a stack, the top of which is the innermost operator/function currently being evaluated. If the token is `(`, I add the new operator to the top of the stack; if the token is `,`, I add the parameter (which is either a literal or a reference to a cell) as an operand to the topmost operator on the stack; if the token is `)`, I pop the current operator off the stack and pass it to the next operator as an operand. This elegantly constructs a representation of the formula that can be quickly evaluated.
- Formula Evaluation: I iterate through the operands of the given `Formula` object and evaluate each (either directly using a literal, retrieving the cell value, or recursively evaluating a nested formula, as the case may be), then evaluate the formula itself. Because of the recursive data structure we generate when parsing, this is relatively straightforward.

- Formula Dependency Management: First, I retrieve the dependencies for a formula by finding all cell references in the formula's own operands, then recursively getting the dependencies of all formula operands it takes. I then maintain a two-way dependency "graph" (which is essentially two adjacency lists with only the methods required for dependency tracking), adding all dependency cells as dependencies of the given cell as well as adding the given cell as a dependent of all dependency cells. This bidirectional storage ensures that it is easy to determine which cells to recompute when a cell is edited (look at the edited cell's dependents) as well as to modify the dependencies of an existing cell (remove the cell as a dependent from any prior dependencies that no longer are such).

Linear Programming

I'm still a bit unclear about the mathematics behind LP, but the general approach I'll take is as follows:

- Parse the formula for the objective cell to obtain a linear objective function (evaluating all non-variable cells—i.e., constants), converting to minimization/maximization (whichever I decide upon) as needed
- For every constraint, convert to a minimization/maximization (consistent with above)
- Replace any lower- (i.e., > 0) bounded variables with zero-bounded temp variables
- Convert to a matrix (tableau) composed of the linear inequalities as equations in slack-variable form, and then perform the Simplex algorithm (which I still need to research further)

Timeline Plan

- TP0: custom UI framework, basic spreadsheet cells UI
- TP1: spreadsheet cells UI with full interaction, formula infrastructure, web scraping
- TP2: sheets & file read/write, data visualization, more robust formula options
- TP3: linear programming, data summarization (e.g., Google Sheets explore/Numbers preview bar, or possibly expanding to a pivot tables-like feature if feasible)

Version Control Plan

I will be using Git as my version control system with a private GitHub repository for cloud storage. I am familiar with the version control system and have used it in a number of past projects. Below is a formatted log of my Git repo as of writing:

```
~/PycharmProjects/15112_term_project (master *%=) ❯ git graph
* 5db20bc - (HEAD -> master, origin/master) Only rerender dependents if they are visible (28 minutes ago) <aaplmath>
* 6e14f64 - Fix formula evaluation and dependency bugs (15 hours ago) <aaplmath>
* 38e0c08 - Rewrite DependencyGraph with two-way relationships (16 hours ago) <aaplmath>
* 5b62bf6 - Add partially working dependency refresh (can't remove deps yet) (17 hours ago) <aaplmath>
* b7b1cdd - Ensure formula cleared on delete when cell selected (2 days ago) <aaplmath>
* fd58980 - Improve handling of formula text in cells and preview (2 days ago) <aaplmath>
* 71f8092 - Fix text clipping on scroll and cell save on select; use CellRef (2 days ago) <aaplmath>
* 14ca416 - Fix issues with selection and preview (2 days ago) <aaplmath>
* 1d5ccbc - Continue working on cell selection/editing behavior (2 days ago) <aaplmath>
* 92f74a9 - Add semi-functional cell selection (2 days ago) <aaplmath>
* b1d8498 - Integrate formulas into grid (currently only compute on scroll) (3 days ago) <aaplmath>
* 864d523 - Rewrite formula parsing using stacks (3 days ago) <aaplmath>
* d8065b8 - Begin experimenting with (not fully functional) formulae (3 days ago) <aaplmath>
* 52c9a7b - Create modular_graphics package; begin modal experimentation (3 days ago) <aaplmath>
* f334424 - (tag: tp0) Pre-TP0 changes (expand tech demo, remove unused method) (5 days ago) <aaplmath>
* c0f2080 - Add scrolling behavior to spreadsheet (6 days ago) <aaplmath>
* 941be58 - Separate core modular graphics classes; add simple spreadsheet UI demo (6 days ago) <aaplmath>
* 3d5d432 - Make App more modular by loading scenes (7 days ago) <aaplmath>
* 837f35a - Move most child appends to init, add text field (7 days ago) <aaplmath>
* 0418201 - Initial commit (7 days ago) <aaplmath>
~/PycharmProjects/15112_term_project (master *%=) ❯
```

Module List

- Requests
- Beautiful Soup 4

TP2 Update

Due to the implementation of data visualization taking slightly longer than expected, I was unable to implement multiple spreadsheets or more robust formulae for TP2. I now expect to complete those features by TP3; however, this may mean that one of either linear programming or data summarization may not be feasible within this timeframe.

TP 3 Update

As expected, I did not have time to take on linear programming between TP2 and TP3. Instead, I implemented multiple spreadsheets and data transposition, which makes it considerably easier to ensure data is in the right format for plotting, as well as a better graphical UI, support for entering "cell blocks" in formulas, improved keyboard navigation, and auto-detection of the presence of series titles in charts, among other features.