**Logical Splitwise**

Contributors: Ethan Oro, Gunguk Kim, Faraz Abbasi

1. **Motivation, Inputs, Outputs**

The problem we are trying to solve is circumventing the complex process of splitting bills from vacations, gifts, restaurants, etc, amongst friends. We all know the hassle when a group of friends go out and has to figure out the specific amount each person needs to pay. With logic programming and interactive, collaborative worksheets, we aim to solve this program. The inputs are the amount of money each user spent, the amount of tip and tax, the users themselves and whether they are paying the bill, paying later or not paying at all. The outputs are how much each individual owes to whoever fronted the bill. In addition, there are two requirements that must be satisfied: there must be exactly one person fronting the bill, and at least 2 people on the bill. The program outputs whether these requirements are met.

1. **Solutions, What Didn’t Work, Optimizations**

As described above, our program allows users to split tabs efficiently. There are a number of options and features. The program begins by “adding” participants to the display. They are then either determined to be someone who will pay later, “Needs to Pay,” or someone that will not pay, “Not Paying,” or theperson who is paying for the bill now, “Fronter.” Then each user inputs the amount they spent and the tip and tax applied. Then the amount each user owes appears on the right-hand side. The amount owed is determined by adding the individual’s amount spent and averaging the amount of cost from everyone not paying across all individuals who do pay. For example, if person A is the fronter (spent $50), person B (spent $100) is not paying and person C (spent $200) is paying later, person C would owe person A, $250 (200 + 100/2). One feature that we included, but then chose to get rid of because it didn’t work with our design was a feature to evenly split what is owed, or stagger the amount owed. We realized that this feature is somewhat redundant, as you can easily slide who owes what, and made our design less appealing. In terms of optimizations, there were several made during this process. For example, originally, the code was built to individually handle several different scenarios, (one fronter and two people paying later, one fronter and one not paying and one paying, etc) but this quickly became complex. To optimize, we made the code much more general, allowing it to handle most cases with minimal need for extra cases. Lastly, we noticed in the collaborative version, there were buggy instances where a third-party user (the person that doesn’t own the sheet) could not change the roles. After spending significant time debugging, we could not come to a conclusion as to why this happens and think it may be the internal workings of how Lily stores datasets.

1. **Alternatives**

As discussed in the previous section, one of the alternatives that we were deciding between was whether or not to include a “split evenly” button. Which we ultimately decided against to make the UI more appealing to the user. Another general design we thought about was whether or not to have more than one fronter. We decided that it is much more of a hassle to have to payback multiple people, and thus we required exactly one fronter.

1. **Logic Programming Component**

Logic programming was very convenient for this solution. Many aspects of this project heavily relied on processes and ideas that were simplified by logic programming, such as having one fronter, ensuring the total amount spent added to the bill, determining how the money was split and making sure anyone not paying didn’t owe any amount. With all of these constraints, having Logic programming as a tool was extremely effective and circumvented many tedious lines of code. It is true that a program of this sort could be programmed in Java or Python, however it would have required much more time and complexity. In addition, the use of collaborative worksheets with logic programming within those was very useful. As a group, we do not have extensive experience in HTML, so having the ability to use widgets to create a UI, and use logic programming with that, made the process much faster and pain free. Generally, for the logic of the program, having Logic Programming’s’ transitions and its automatic, real time updates made the program much more efficient. Most importantly, however, was the ability to make the worksheet collaborative where each person can access the sheet and edit the amount they spent.

1. **Future Work**

With more time into the project, we would like to make this collaborate sheet available on the public web for anyone to use with dashboard to store multiple past bills as reference. Also, we want to add a feature in which one person can choose to fully cover another. This way, if this option is selected, someone not paying isn’t distributed over the mass, but rather it can be allocated to one individual person.

**Acknowledgements and Feedback:**

We would like to thank Mike, Vinay and Abhijeet for the great instruction and for all of the help and guidance over the quarter. Our group really enjoyed this quarter’s work and knowledge we gained. As for feedback, we wished throughout the quarter, especially in the beginning, to get a thorough walk-through of Sierra. Abhijeet’s office hours helped a ton but would be very useful before the first assignment. We also would have liked more office hours spread across the week/closer to the due date to assist with problems we encountered on the assignments. Lastly, it would be very helpful if in lectures there were more examples and live demos of using Sierra to solve example problems similar to the process of assignments.