Matrix: Progress Report

Stanley: [strategy refactor, new equation types (exponentiation/LCM/GCD), random seed, input persistence, automated testing]
As the user_package was mostly finished by Phase 1, Stanley worked on worksheet generation. He also added some features in the UI, and implemented a GitHub feature for automated testing. Most notably, he refactored the equation_builders package, in order to reduce redundancy in code. Previously, operand creation was done using a Builder design pattern, where subclasses of abstract Builder classes would implement often redundant methods. The main difference was that each subclass had one specific method to generate operands uniquely, which is why he believed the Strategy design pattern is more apt for the scenario. The PR for the refactor is located here. He also implemented new whole number equation types: exponentiation, LCM and GCD. He also added the random seed feature embedded in worksheets then down to equations. The random seed allows users to regenerate the exact same worksheet previously made. The PR for the random seed is found here. In the UI, he helped debug the issue of the button display appearing different on Mac/Windows. He also allowed for temporary persistence of user input. Specifically, it allows users, who fill in their topic details and head to the next page, the ability to automatically have their past input shown when going back one page. Lastly, he set up Apache Ant for the GitHub repository to automate testing whenever a commit or pull request is merged into main.

Sean: [parameter classes, equations can convert into hashmaps for PDF generation, latex representations, mixed fractions, new equation types (fractions multiplication/division)]
With the facade design pattern setup for PDF generation was now set up, Sean primarily worked on extending its functionality. Firstly Sean refactored hashmaps into new classes to store the parameters for equation generation details (ex. number of questions, operand ranges, negative values allowed ...) all of which are in the equation_parameters package. The PR for this refactor is here. This improved the readability of the code and allowed for creating different subclasses of equation details which would be useful later on as different equations (ex. fraction vs whole numbers) had slightly different parameters. From there Sean made another refactor to have Worksheet classes send up equation information to the PDF generator as a hashmap rather than a list, see here. While it was a small refactor, this drastically improved readability. Finally with these refactors in place, Sean could easily add new latex representations such as vertical format and division bracket format by manipulating the equation information hashmap. Similarly he added mixed fractions vs improper fractions which required splitting fraction classes into two subclasses, all of which is seen here. On the side, Sean also implemented algorithms for fraction multiplication and division questions by utilizing the utility classes made by Will to make sure the questions were reasonable, shown here.

Will: [new equation types (fractions addition/subtraction), constants, utilities package, new symbols (fractions, exponents), refactoring]
Will worked primarily on creating new equation types and refactoring older code to remove various code smells. First, he created the fraction class and its associated calculations. This required him to implement smaller features as well regarding fraction reduction and improper versus proper representation. Because his expression tree adhered to the open/closed principle, his fraction class was a subclass of Symbol and therefore easily implemented. He then implemented the exponentiation operator in a similar manner. Next, Will completed the algorithms for generating fractional addition and subtraction. However, a purely random generation of fraction problems would have created an undesirably high number of difficult problems, commonly seen in other worksheet generation programs (ex. most denominators would be primes or share

no common factors, so trying to make the two the same when solving the problem would result in excessively large denominators). Given this wasn't the goal of the program, Will decided to create two utilities: FactorFinder would find the various factors of a number (in various representations including all the possible factors, prime factorization, and exponentiated prime factorization), and DenominatorDistribution would use this to increase the probability of selecting easily divisible numbers based on the number of factors. This, along with other optimizations, made the equations realistic and offered the user a high degree of control. The PR for this feature can be found here. These changes were important as they helped the group adhere to one of the major requirements of the specification, that being the existence of Values beyond WholeNum. Will also performed a series of refactors to remove duplicate code and reduce the chance of bugs. For one, Will had used the extract to superclass refactor to prevent overlap between the largely similar algorithms (such as fraction multiplication/division), as shown here. Will also added constants rather than strings for the HashMap keys. These classes were imported statically, making them easier to use (see here for this PR). In general, these refactors were important as they helped improve the readability and reusability of the code, decreasing the occurrence of bugs.

Kerim: [Interface Adapter Layer, User package, Model View Presenter Architectural Pattern]
As the user_package was almost finished by Phase 1, Kerim mostly worked on the UserController and UserManager classes. The UserController class handles most of the requests for the User such as registering a new user or logging in a new User. UserManager handles User entities such as deleting a User or creating a new User.
In order to follow the dependency inversion principle Kerim decided that creating an Interface named DataAccessInterface would be the best course of action. This interface can be implemented by any DataAccess vessel such as the database in the FrameWorks and Drivers Layer. Any class in FrameWorks and Drivers Layer that properly implements the DataAccessInterface can be instantiated as a DataAccessInterface object without breaking the Dependency inversion rule. In addition, Kerim thought that this decision would bring flexibility in the future in the case that his team decided to change where they stored the data. As a matter of fact, by the end of Phase 2 the database connection was implemented which could be connected to the Interface Adapters Layer by using the DataAccessInterface without breaking the dependency inversion principle. Currently the data is stored locally with the help of a class named LocalDataAccess which implements DataAccessInterface.
In order to implement the Model View Presenter Architectural Pattern, Kerim had to modify the Screen classes in the user_interface package and create the UserPresenter class. The pull request for the Model View Presenter Architectural Pattern can be found here. Anytime the user requests something from the screen classes, they immediately call the UserPresenter. UserPresenter takes these requests and delivers them to the model which consists of UserManager and HistoryManager. The UserPresenter sends the output it receives from the model back to the view (screen classes) so that it can be displayed to the end user.

Ethan: [User Interface design, GUI, tracking user input, catching invalid input, displaying user information]
Ethan primarily worked on the User Interface; including the design and display. In phase 1, Ethan designed the UI to include one superclass (screen) and multiple subclasses for each screen. For example, the topic screen and equation details pages are 2 different subclasses. This particular design was clean and simple. Ethan was also able to design a simple GUI by the end of Phase 1. This GUI had multiple screens; such as the login screen, topic screen, equation details screen, and preview screen. The display prompted the user for their input and customized a worksheet accordingly. Before progressing, the inputted information was checked to be valid. If the input was invalid (e.g. invalid username), the user would not be allowed to progress to the next screen and a warning message would appear. After all the inputs that the user entered were checked to be

valid, the user would be able to customize their worksheet and see a preview of the worksheet they generated. In addition, the user could download the worksheet to a particular file path they input. The PR including these particular features can be seen here. The final updates to the UI for phase 1 were included in this PR; including storing the user's worksheet information, catching invalid inputs, and downloading the worksheet. In phase 2, Ethan focused more on improving the UI aesthetics (adhering to the 7 principles of universal design), while also improving many features implemented in phase 1. This ranged from changing the color scheme of the UI, to displaying multiple pages of the worksheet on the preview screen. He also added a logout feature and "delete user" feature. Finally, he added the options of LCM, GCD, fractions, and exponents to customize worksheets. Along with that, there were several invalid operator and question type/format combinations which were accounted for. For instance, the user cannot create a LCM fractions worksheet. Thus, this combination was caught and a warning is displayed to the user.

Piotr: [GUI, Worksheet History Screen, Regenerate Old Worksheets (Similar vs Exact), more accessible UI (appearance of the GUI and Enter Key), screens for fraction questions, warning messages for invalid inputs]
Piotr was tasked with implementing the User Interface and the tasks affiliated with it. In phase one, Piotr worked on and completed the initial rough design for the worksheet history screen (which he would later updated in phase two), the initial GUI (which was also updated in phase two), and the methods and design of collecting the user inputs (textboxes, comboboxes, buttons and the use of multiple screens for clarity).
In the next phase, Piotr focused on improving the design of the worksheet history screen, displaying and implementing the new question types in the GUI, creating warning messages for these new question types (for invalid user inputs), making the User Interface more accessible by implementing the ability to use the Enter Key instead of buttons, and the regenerate worksheet function that allows the user to decide between regenerating an old worksheet to be exactly the same or similar (same amount of questions and format just potentially different values). Piotr improved the design of the worksheet history screen by changing into a table which was cleaner aesthetically and made more sense from a design standpoint. Additionally, he implemented the remove worksheet and store worksheet score functions for this screen. Piotr worked on displaying the new question types in the GUI which included creating new screens. For example, equation details for addition/subtraction fractions and multiplication/division fractions are different from one another and different from the whole number equation details. The pull request that includes many of these changes (worksheet history, fraction equation details, invalid input messages, and more) can be seen here (Ethan and Piotr worked on the same branch).