Progress Report: Matrix

Stanley Hua, Kerim Saltoglu, Ethan Ing, Piotr Pralat, Will Jeong, Sean Jeong

Specification: Ethan

After we confirmed that we would develop a math worksheet app, we discussed the main functions of the app. Essentially, we decided that we want to provide users with the ability to create customizable math worksheets at the grade school level. From there, we decided exactly what the user could customize; such as topic, question format, difficulty, number of questions, title, and font size. Once the user generates their customized worksheet, a single local PDF (which can be saved and downloaded) will be created with an answer sheet on a separate page. Along with that, we will allow each user to access previous worksheets, user details, and user scores (which they can input). After completing a detailed specification, we imagined how the app would be used from a user perspective, and created a diagram of the user interface.

CRC model: Sean

For this part we divided into 3 subgroups, and each of us generated a CRC model based on the same specification. Once completed we regrouped and merged all the models together, discussing over details and edited our specification accordingly.

Our CRC model is organized into the 4 levels as designated by clean architecture. We also have two main branches to the project: the 'user' branch, which focuses on the user information and worksheet history (amongst others) and the worksheet generator' branch, which is about actually generating the worksheet. The 'user' related cards are on the right and 'worksheet generator' related cards are on the left for clarity.

Starting with the 'worksheet generator' branch's frameworks and drivers, UserInterface takes in various inputs and sends that information to the interface adapters. The interface adapters, WorksheetGenerator is responsible for creating the equations for a Worksheet (but is not presentable), while PDFPresenter is responsible for displaying the Worksheet (as a PDF). Within the application business rules, there is only EquationGenerator, which will be responsible for most of the computational work, as it creates and solves pseudo-randomized equations with the given parameters. Finally, the entities themselves are Equations (contain both question and answers, hence their name).

For 'user', the UserInterface can send various method calls to UserController (interface adapter) to either store or retrieve information about Users (the entity). UserController can store information via the UserManagerUpdater (application business rule) or retrieve information via the UserManagerViewer (application business rule).

Scenario walkthrough: Stan

We decided to split it into two distinct parts: User-related classes and Worksheet-related classes. Following the subgroups used in creating the CRC models, Kerim and Stan provided details about the relationships of User-related classes, while Sean and Will did the same for Worksheet generation-related functions. In particular, the User-related portion discusses how a user can log in (or register an account), view their profile (including past scores), and display their past actions of generating worksheets, as well record scores on past worksheets or remove past records. All of which are handled by the UserController, which creates a UserManager class that creates and delegates tasks to the UserManagerViewer and UserManagerUpdater classes. The Worksheet-related portion goes over how the user interacts with the UI to specify Worksheet parameters (topic, format, difficulty, number of questions, etc.), which are used by WorksheetGenerator to generate questions and package it in a way that is easy for the PDFPresenter to generate their PDF versions.

Skeleton program: Will + Sean

So far, we've implemented some basic functionality for all classes listed on the CRC model, except for a few enterprise business rules related to equation generation (LcmGcfEquation, BinaryExpTree, Fraction). These were omitted as only one type of equation was needed to demonstrate basic equation generation functionality. Currently, our user interface simply prompts the user for inputs in the console. This accesses and stores relevant information into the User class (ex. name) and can also generate a random standard addition Worksheet within the console (all the formatting details are not used, but are kept track of).

What has each group member been working on and plans to work on next?: Will

We've divided our team into 3 subgroups to address 3 general responsibilities. Ethan and Piotr are working on the user interface and other frameworks and drivers. Kerim and Stan are focusing on User related methods/classes such as logging in, recording scores, etc. Finally, Will and Sean are implementing classes related to equation generation. Regarding our next steps, we will continue to focus on improving these three aspects of our project (though the subgroups assigned to these tasks may change). For example, the user interface will need to be adapted into an app interface. Additionally, the user information needs to be stored in some file between sessions. One last suggestion would be to adapt the current "PDF representation" into an actual PDF.

What has worked well so far with your design as you have started implementing the code?: Piotr

We used a factory design pattern which helped us create the objects that we wanted for our code without having to stick to exact classes. This gave us some freedoms while we implemented our code while also making sure we did not stray away from our larger goals. Additionally, our design (factory design pattern) also made it easy for us to implement code that expanded our project\add new features. Furthermore, the design worked well when we needed to test the code we implemented. We are able to test each class individually and independently without unnecessary hassles.

One open question your group is struggling with (TA will address it in their feedback for phase 0) - Will and Sean

What's the cleanest way to implement the question types we want? Currently, we're thinking of passing strings that represent the question type ("standard add") and difficulty ("easy") into an if statement that uses these strings to determine what algorithm should be called. However, this approach is very messy. For example, if we have standard addition, subtraction, multiplication, division, and their fractional variants, we get 8 basic equation types. Combined with 3 difficulty levels (easy, medium hard) will mean an if statement that deals with 24 cases. One way we could reduce the number of cases is by adding parameters instead of having them choose easy/medium/hard; for example, addition problems could also ask for the range of numbers you want to see in the question (ex. 0-100), and this would directly be used in the calculation. This would give the user more options, but it would also create an inconsistent interface between different types of questions. Another option might be to use a map that maps a question type + equation to a certain range of parameters, but this would be an extremely lengthy map. Furthermore, we're unsure whether these separate question types should be their own classes or if we should just use a helper method for each equation and store the output in a general BedmasEquation.