|  |  |  |
| --- | --- | --- |
| CSC 436  Due 10/17/17 | **Project Proposal** |  |

OpenBurn Propulsion Simulator

* *Team #12: Team Rocket*
* *Abhishek Rane, Daniel Tranfaglia, Isaac Plunkett, Andrew Tarr, Vicente Figueroa*

1. Introduction – Viable System

* **Goals for this iteration:** Command-line program that receives input of fuel shape, type, and length. Outputs a CSV file of pressure vs. time. Supports imperial units. Testing for model completed. Original MatLab files with calculations have been translated to Java.
* **What’s working:** Program currently receives input from the CMD line.  Input data is checked and if not acceptable, user is re-prompted for that data.  Rocket is simulated until all fuel has been consumed.  The simulation results are outputted to a csv file.  Spreadsheet editors can take csv file and produce graphs.
* **What tests:** Unit testing for Nozzle and Grain classes completed and passed. Command line interface has been visually tested for ease of use and error checking. RocketMath testing was moved to Feature Release since model methods were not fully complete until the very end, and more value review in the actual results is needed before comparison testing.

1. Customer Value

* Multi-platform, open-source software, introduces multiple unit availability, accounts for erosive burning, improved Graphical User Interface.
* Java implementation, open-source equations from NASA.
* No major changes from original proposal, but RocketMath testing was moved to the Feature Release.
  + Date of Change: October 8th, 2017
  + Motivation for change: Not enough work finished on RocketMath methods to perform testing on the methods. Testing for other model classes and command-line interface were completed instead.
  + RocketMath testing involves in-depth comparisons of values between actual field testing results and simulated results.

1. Use Case – Rocket Engine Simulation
2. User is asked for data.

{User enters data} -------------------------------------------------↓

1. RocketMath runs methods to calculate simulation results

{Methods run with user inputs} ---------------------------------↓

1. Store results in a SimulationResult object

{Create a SimulationResult object from the simulation} ----↓

1. Repeat steps 3 and 4 until all “fuel” is consumed.
2. A csv file is opened.
3. All SimulationResults are written to the csv file.
4. Simulation ends

Alternate Flow:  input data is not acceptable

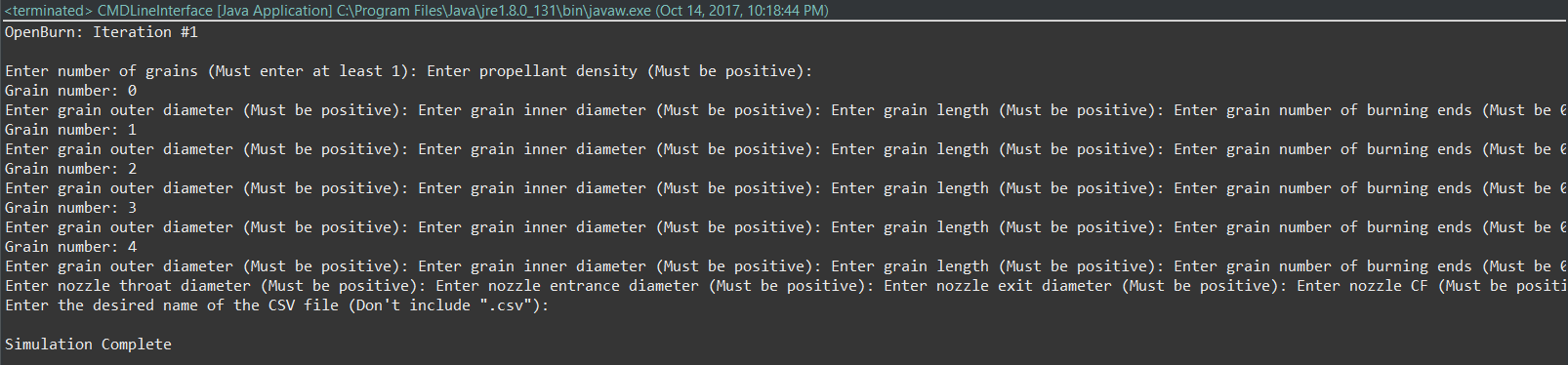
At {User enters data}, if data does not pass sanity check (described to user while in use),

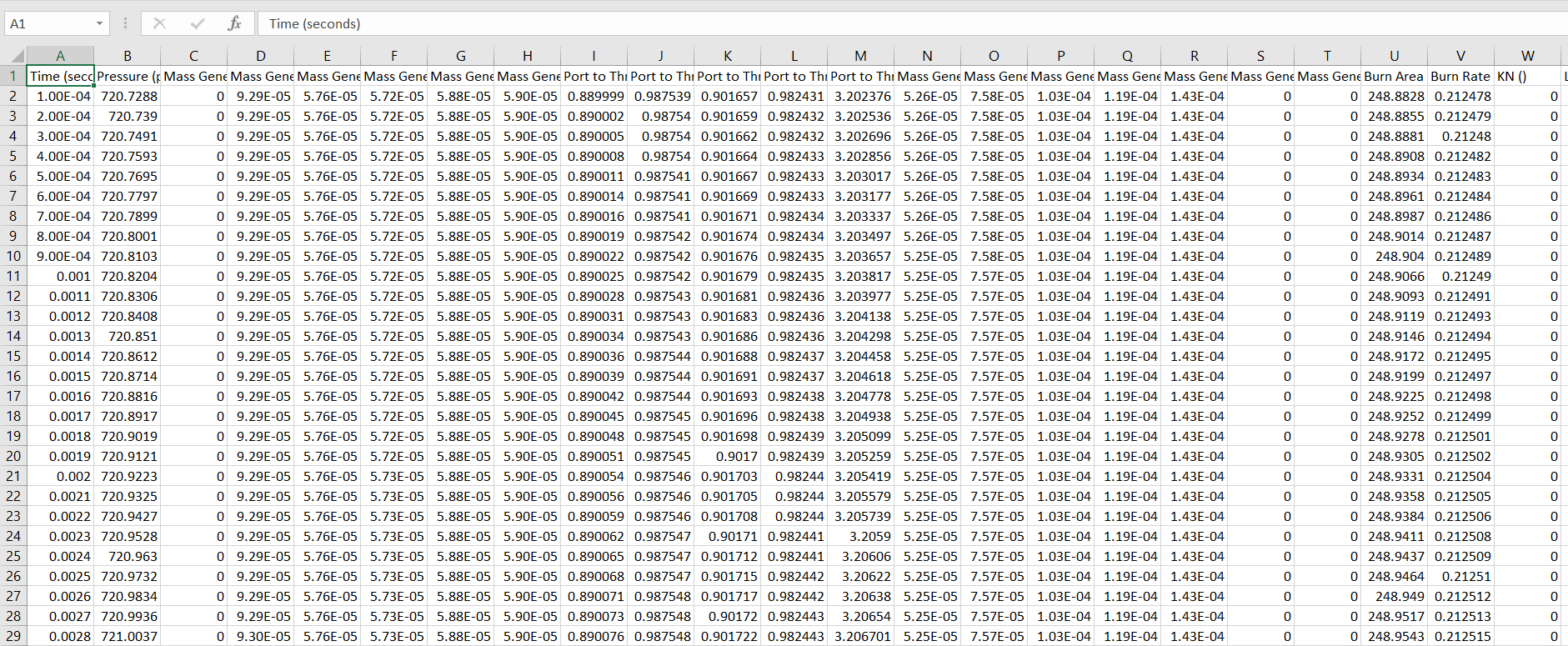
resume the basic flow at {User enters data}.

1. Technology

System at the end of this iteration

* **Goals for this iteration:** Functioning CMD interface, output to CSV, model completed.
* **What works:** Input prompting and collection from the user, output to CSV file.





Module Hierarchy and Module Guide

We use a MVC hierarchy, with our main math and simulation result being produced in the modal. The controller and the view working together to get the desired simulation data from the user. The view as of right now is in the format of a csv file, and produces the results, exports them to that file, and then can be made into a graph.

CMD Interface (Controller)

1. Prompts user for simulation result data.
2. The service is that it takes the data and matches it to the proper places on the back end.
3. The secret for the CMD Interface is converting the data to a format that can be handled by the back end.
4. The error checking done by this module is making sure the user enters the correct type of data. For example, it is an error if the user enters characters instead of digits

CSV Generator (View)

1. This takes the output from what was calculated from the back end. This includes thrust, pressure, and time columns
2. The service provided is a creation of a csv file that can be used by other modules to create graphs, for the user to view the data.
3. The secret is handling the creation of the csv file as well as outputting the data to the file.
4. The error checking in this module is making sure that the given file from the user does not have a .csv extension

Rocket Math (Model)

1. Does the main calculations of the simulation to the most accurate degree.
2. Does all the calculations and passes the data to the SimulationResults object based on user preference, and keeps some of the data for later calculation.
3. The secret is the formulas and calculations being done in the back end.
4. Errors - calculations with negative numbers.

Grain (Model)

1. Superclass for all grains and provides the interface for grains
2. The service this module provides is the public interface for all grains.
3. This modules secret is the type of its subclasses. Hides the calculations performed by its subclasses.
4. This module is an abstract class. Any errors will would be handled by its subclasses.

Cylindrical Grain (Model)

1. Subclass of grain that implements the equations needed for a cylindrical grain
2. This module provides the available burn surface area, inner volume, burn status of a grain and also performs grain regression.
3. The module hides the equations that are specific to cylindrical grains
4. The module ensures that the geometry is physically possible (i.e. no negative numbers or volumes) and throws an error. The error must be caught, higher up in the module hierarchy.

Nozzle (Model)

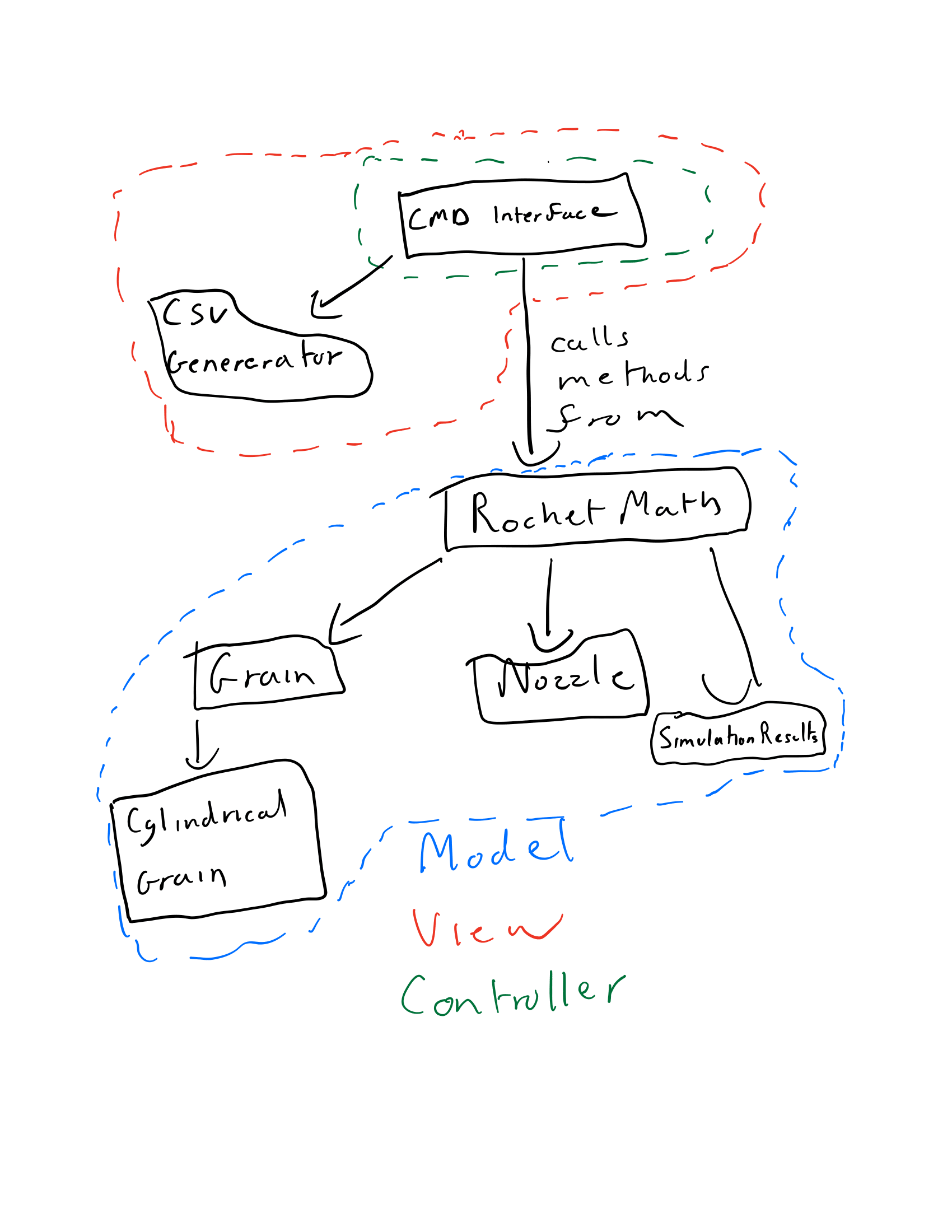
1. Stores geometry of a rocket nozzle.
2. Provides diameter and area when prompted.
3. Converts input diameter to area.
4. Negative numbers will produce incorrect results.

Simulation Results (Model)

1. Stores the simulation data for one point in time.
2. Provides storage for various data representing the state of the rocket at one point in time.
3. Hides the implementation for converting itself to a string.
4. Negative numbers will produce incorrect results.

Architecture of the system

* As things stand, the system has a working command-line interface (CMD Interface) that prompts the user for input regarding the Nozzle, and Grain(s). The methods from the RocketMath class are called to create a list of SimulatioResults objects. Lastly, the list of SimulationResult objects is written to a CSV file using the CSV Generator, and the simulation ends.



1. Project Management

* **Plan for next iteration:** Feature Release - GUI that produce visual graph of Pressure vs. Time, Imperial and metric units, finish testing RocketMath methods. Save graph view as an image (PNG or JPEG).
* **Tracking changes:** To track changes, we will continue using a GitHub repository with the OpenBurn project, communicate via weekly meetings, and use Trello with a new board for the Feature Release. All worked out well during the previous iteration. We are also constantly communicating when changes or commits are made through
* **Plan for the rest of the semester:**
  + **Feature -** GUI that produces a visual graph of Pressure vs Time.  Can save data to a CSV.  Imperial and metric units. RocketMath testing completed.
  + **Beta -**GUI that produces a visual graph of Thrust vs Time.  Can save data to a RSE file.  Calculations account for Erosive burning.  Can display multiple plots.
  + **Final -** Calculations support transient state calculations, can display side-by-side graphs for comparison.

1. Team

* **Abhishek (Product Manager):**  Wrote majority of the RocketMath class, which contains all simulation logic. Provided insight to product requirements and answered questions.
* **Daniel (Quality Assurance Engineer)**: Wrote unit testing for most model classes (RocketMath testing moved to iteration 2), refactored all code at the end of the previous iteration, set standards for future development practices, monitored GitHub repository commits and merges.
* **Isaac (Main Front-End Developer):** Created CMD Line interface that contains the main method.  Wrote basic Grain and SimulationResult classes.
* **Vicente (Team Coordinator / Scrum Master):** Ran Trello board, which included making tickets, assigning task, and breaking down what needs to be done for each sprint/ iteration. Led team meetings. Also coordinated regular meet ups, and practiced keeping the team in communication with each other. Also transposed some matlab files to Java.
* **Andrew (Main Back-End Developer):** Wrote a csv file converter which took the input data and exported it to a csv file.

1. Reflection

* **What went well:** Communicated well, met on a frequent basis, created tickets for what needed to be done, completed the iteration early. All major features were implemented in this iteration.
* **What didn’t go well:** Updating trello tickets late, creating tests for RocketMath.
* **Remediation plan:** Update tickets to communicate to the team when the task is completed and create more tests for the system.