CSC 436
Due 10/31/17

Status Report 2

THE UNIVERSITY OF ARIZONA
Computer Science

OpenBurn

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

1. Introduction

- First Iteration: Command-line interface that prompted user for data regarding the propellant density, change in time, grains, and nozzle. Ran simulation calculations to produce results such as thrust vs. time and pressure vs. time, which was written to a CSV file.
- Second Iteration: Developed GUI to handle 1 type of grain, multiple grains, added motor casing to calculations, added unit converter, simulate button on GUI, created rse generator, csv file generator is an optional feature, graph uses JavaFX ScatterChart with wrapper class. GUI is fully functional and produces the same results as the command-line interface.
- Overview of Changes: Changed the graph library originally planned (used JavaFX ScatterChart instead of JFreeChart), some team members worked on the GUI, which was different than what they were originally supposed to work on. Some roles were changed a little to compensate for time, and more work was added on to some members than originally planned.

2. Customer Value

- Problem Definition: Multi-platform, open-source software, introduces multiple unit availability, accounts for erosive burning, improved Graphical User Interface.
- Customer Value: University of Arizona IREC Team, rocket enthusiasts.
- Changes from Proposal – None.

User Stories: We'd like to have all these features working by the Beta Release.

As a user when I enter in values I want to be able to select different units.
As a user I want to be able to select different motor grain types.
As a user I want to be able to enter or select propellant models.
As a user I want to be able to visualize my simulation outputs.
As a user I want to be able to export my simulation output to other programs.
As a user I want my simulations to account for edge case simulations.

Use Case – Rocket Engine Simulation

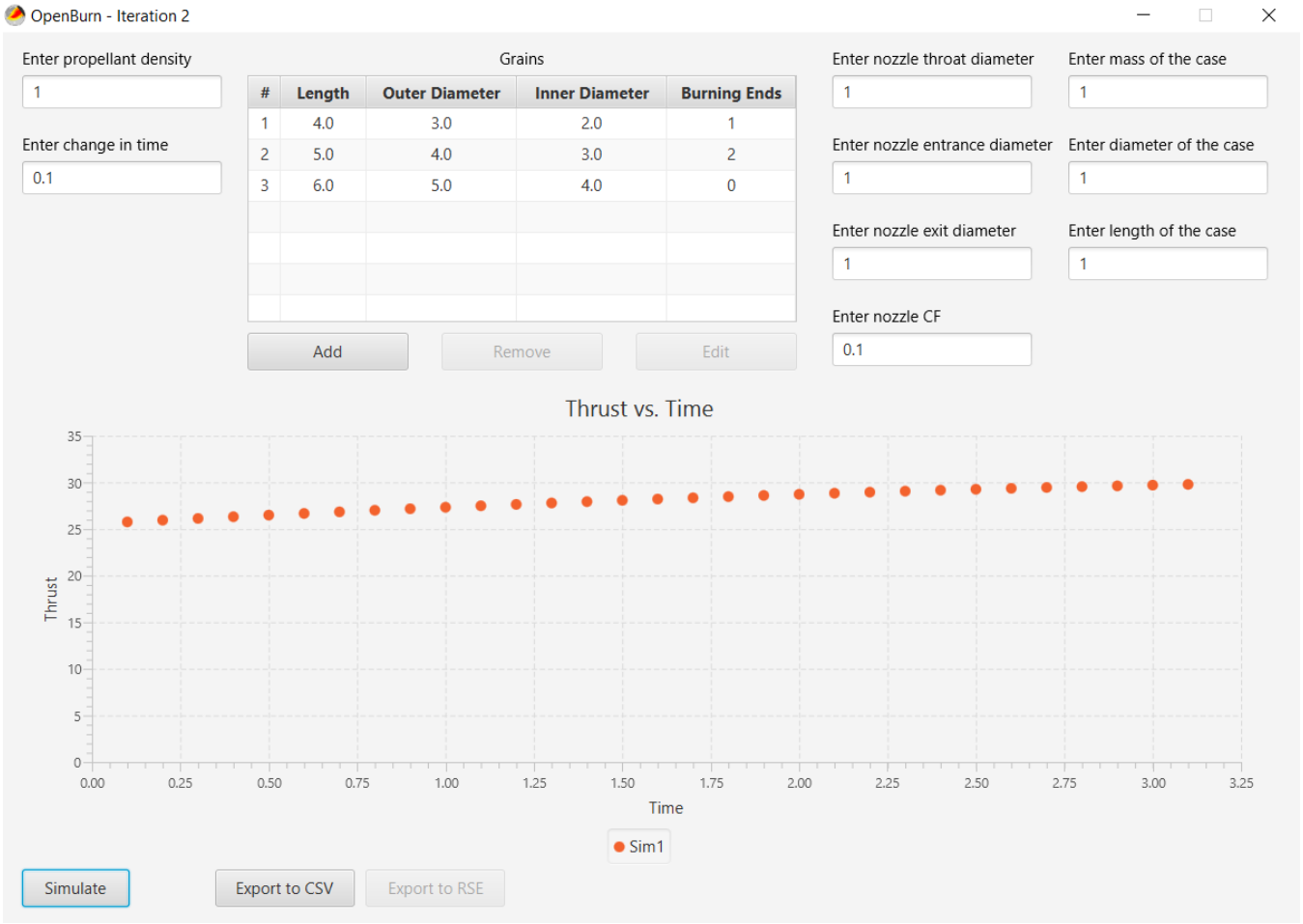
1. User is asked for data.
   {User enters data} ----------↓
2. RocketMath runs methods to calculate simulation results
   {Methods run with user inputs} ----------↓
3. Store results in a SimulationResult object
   {Create a SimulationResult object from the simulation} ----↓
4. Repeat steps 3 and 4 until all "fuel" is consumed.
5. A csv file is opened.
6. All SimulationResults are written to the csv file.
7. 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}.

3. Technology

System at the end of this iteration

- Goals for this iteration: This iteration, our goals were to make sure that we had a working GUI that showcased the proper functionality of our App. This consisted of making most of the GUI and adding our back-end implementation to the GUI, while keeping the CSV output feature.
- What works: We got the main functionality of the GUI working, by linking the back end to the front end. This allows our app to be more user friendly and puts us one step closer to the final product. CSV output is an optional feature that can be run by clicking a button after a simulation has run.

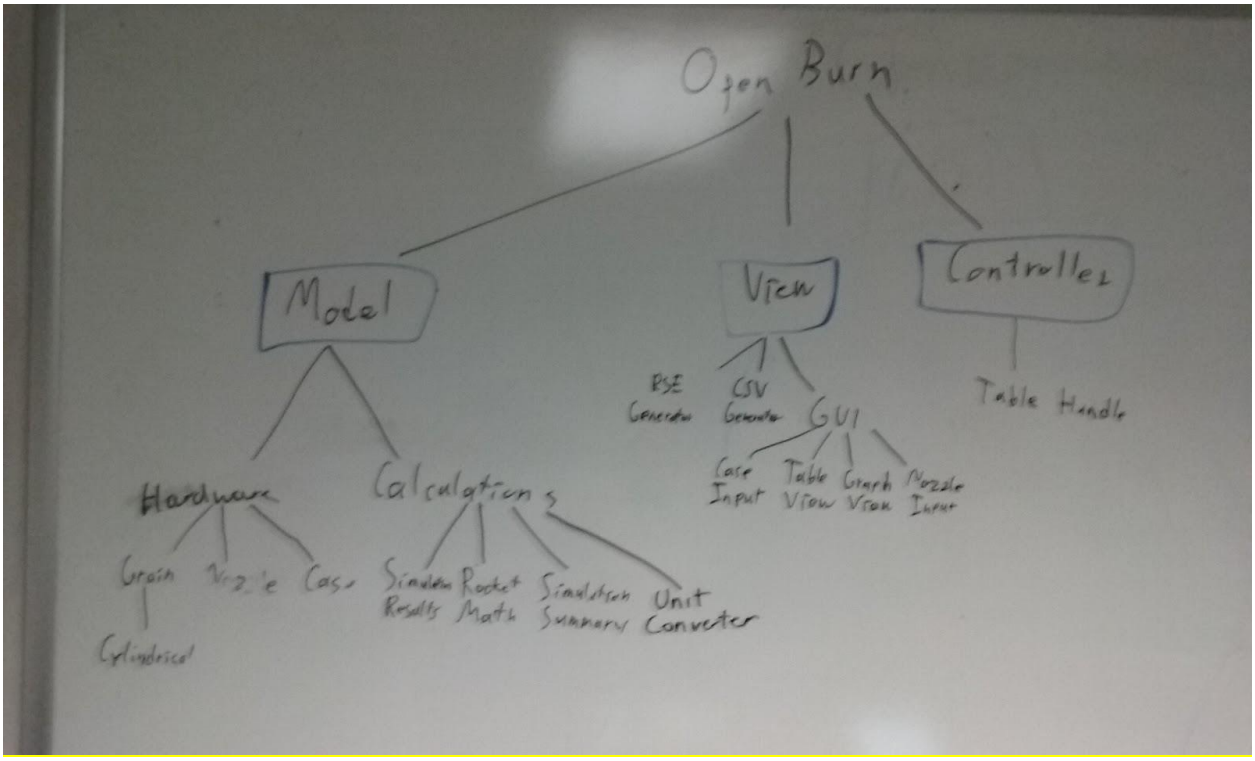


CSV file from the above simulation, named “Sim1.csv”

Time (seconds)	Pressure (psi)	Mass Gene	Mass Gene	Mass Gene	Mass Gene	Port to Thi	Port to Thi	Port to Thi	Mass Flow	Mass Flow	Mass Flow	Mass Flow	Mass Flow	Burn Area	Burn Rate	KN ()
0.1	327.988208	0	0	0	0	0.44914	0.565475	16.25659	0	0	0	0	0	162.5774	0.159729	207
0.2	330.4006971	0	0	0	0	0.45377	0.568417	16.51574	0	0	0	0	0	163.2727	0.160053	207.8853
0.3	332.7678693	0	0	0	0	0.458337	0.571326	16.77747	0	0	0	0	0	163.955	0.160371	208.754
0.4	335.0893322	0	0	0	0	0.462841	0.574203	17.04176	0	0	0	0	0	164.6241	0.160683	209.6059
0.5	337.3646998	0	0	0	0	0.467283	0.577048	17.30864	0	0	0	0	0	165.2798	0.160988	210.4408
0.6	339.5935924	0	0	0	0	0.471664	0.579861	17.57808	0	0	0	0	0	165.9222	0.161288	211.2588
0.7	341.775637	0	0	0	0	0.475985	0.582642	17.85011	0	0	0	0	0	166.5511	0.161581	212.0595
0.8	343.9104672	0	0	0	0	0.480247	0.585393	18.12471	0	0	0	0	0	167.1664	0.161868	212.8429
0.9	345.9977236	0	0	0	0	0.48445	0.588114	18.40188	0	0	0	0	0	167.768	0.162148	213.6089
1	348.0370536	0	0	0	0	0.488597	0.590804	18.68164	0	0	0	0	0	168.3558	0.162422	214.3572
1.1	350.0281117	0	0	0	0	0.492687	0.593464	18.96397	0	0	0	0	0	168.9296	0.162689	215.0879
1.2	351.9705598	0	0	0	0	0.496721	0.596095	19.24887	0	0	0	0	0	169.4894	0.16295	215.8007
1.3	353.8640669	0	0	0	0	0.5007	0.598697	19.53635	0	0	0	0	0	170.0352	0.163204	216.4955

- Tests run: We have tested the output to our graph of the results that were produced on Google, and turns out that our app is more accurate in terms of decimals. We also tested the GUI and rocket math to make sure all the calculations are correct.

Module Hierarchy



Module Guide

Grain (Model.hardware.grain)

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

Cylindrical Grain (Model.hardware.grains)

- Subclass of grain that implements the equations needed for a cylindrical grain
- This module provides the available burn surface area, inner volume, burn status of a grain and also performs grain regression.
- The module hides the equations that are specific to cylindrical grains
- 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.

Case (Model.hardware)
- Stores the geometry of the case hardware
- Provided the storage for various data representing the state case
- Hides any checks for the case
- Negative numbers will produce incorrect results.

Nozzle (Model.hardware)
- Stores geometry of a rocket nozzle.
- Provides diameter and area when prompted.
- Converts input diameter to area.
- Negative numbers will produce incorrect results.

Rocket Math (Model.calculations)
- Does the main calculations of the simulation to the most accurate degree.
- 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.
- The secret is the formulas and calculations being done in the back end.
- Errors - calculations with negative numbers.

Unit Converter (Model.calculations)
- Provides conversions between imperial and metric.
- Provides static functions which accepts a double and returns a double.
- Hides the mathematics of unit conversion.
- Due to every number being a valid measurement, there is no meaningful error checking.

Simulation Results (Model.calculations)
- Stores the simulation data for one point in time.
- Provides storage for various data representing the state of the rocket at one point in time.
- Hides the implementation for converting itself to a string.
- Negative numbers will produce incorrect results.

Simulation Summary (Model.calculations)
- Performs analysis of the results generated by a results generator.
- The service is it provides the impulse of the motor, its classification, mins and max values. Also provides a getter for all these data points.
- The secret is the code and equations for the analysis.
- Error checking is the equations break down with negative numbers and this is checked for.

CSV Converter(View)
- This takes the output from what was calculated from the back end. This includes thrust, pressure, and time columns
- 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.
- The secret is handling the creation of the csv file as well as outputting the data to the file.
- The error checking in this module is making sure that the given file from the user does not have a .csv extension

Graph View (View)
- This takes the output of the backend calculations. This includes time and pressure.
- The service provides is creating a visual graph of our data.
- The secret is handling the JavaFX ScatterChart interface.
- The error checking is ensuring the backend output exists.

Case Input View (View)
- Contains all input fields to create a case to use during simulations.
- The service is providing fields for the user to enter specifications about a case.
- The secret is that data in the fields is taken to the back-end for simulation calculations.
- The error checking (for now) is throwing exceptions for negative inputs.

Nozzle Input View (View)
- Contains all input fields to create a nozzle to use during simulations.
- The service is providing fields for the user to enter specifications about a nozzle.
- The secret is that data in the fields is taken to the back-end for simulation calculations.
- The error checking (for now) is throwing exceptions for negative inputs.

Grain Input View (View.grain.input)
- Contains a table view and buttons to add, remove, and edit grains.
- The service is providing a compact and easy to use interface for adding any number of grains for a simulation.
- The secret is that the list of grains is taken to the back-end for simulation calculations.
- The error checking (for now) is throwing exceptions for negative input, or if inner diameter is greater than outer diameter, or burning ends is not 0, 1, or 2.

Add Grain Window (View.grain.input)
- This opens a new window where the user can enter data specific to the grain such as the length, number of burning ends, outer diameter and inner diameter.
- This takes the grain data from the user and applies that to the grain which will be used to run the simulation.
- The secret is that it is the data is applied to the grain and taken to to the back end to help run the simulation.
- The error checking is to make sure the number of burning ends is 0, 1, or 2. Also make sure that outer diameter is greater than the inner diameter.

Remove Grain Window (View.grain.input)
- This opens a window that allows the user to remove a grain. The user selects the grain then clicks yes if they want it deleted.
- This pulls the grain data from the back end then allows the user remove it, then no longer applies that grain when doing the simulation.
- The secret is that the grain data is pulled from the back end then deleted.
- There is no error checking, however if the user changes their mind and ends up wanting to keep the grain, they can select “no” and it will close the window not affecting any other grains.

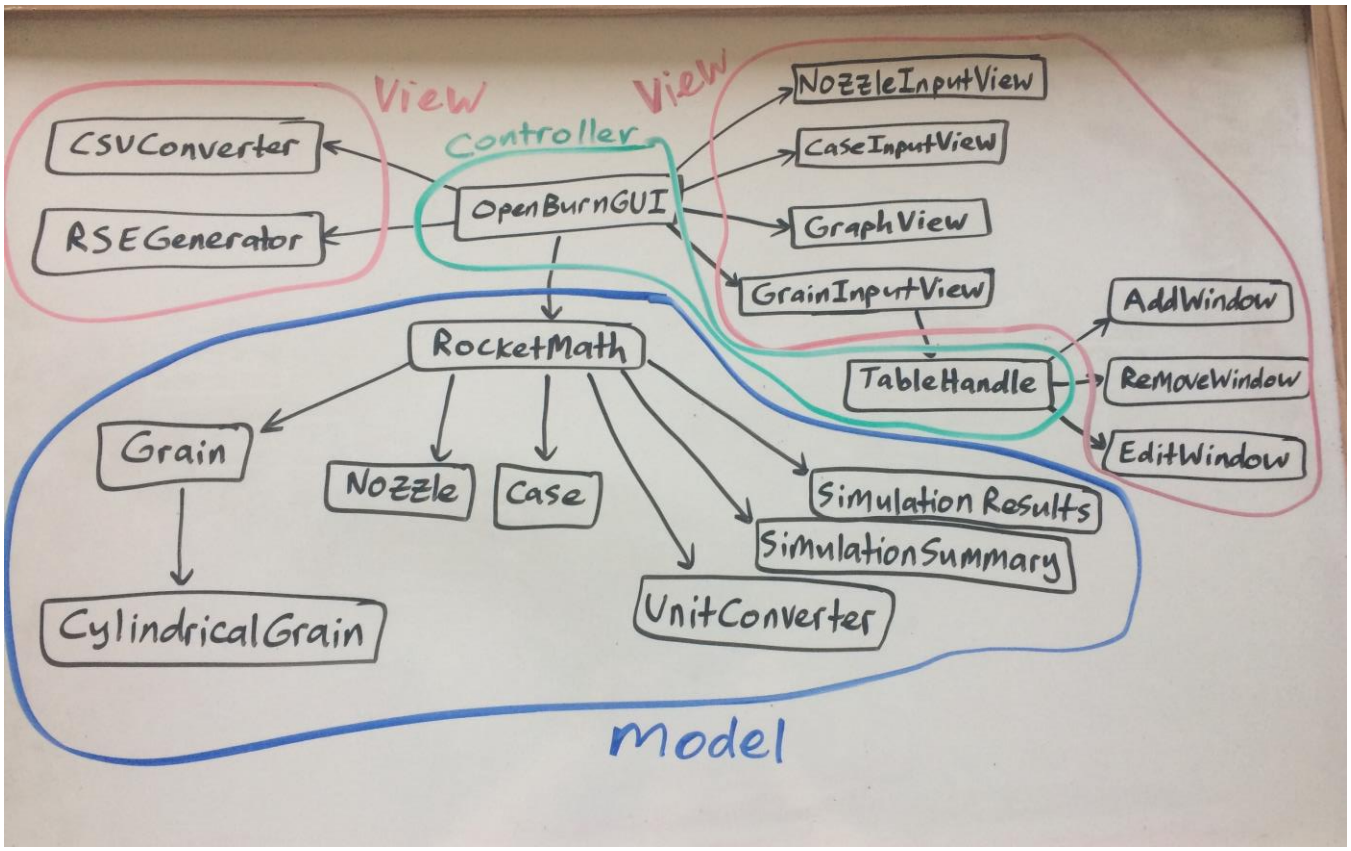
Edit Grain Window (View.grain.input)

- Opens a window that allows the user to edit grain specific data (length, number of burning ends, outer diameter, and inner diameter).
- This allows the user to edit the grain data, then is saved so it can be applied when the simulation is run.
- The secret is that the original grain data is pulled from the back end, then allows the user to edit the data, then is saved and sent to the back end.
- This has the same error checking as the add grain window.

Table Handle (Controller)

- Provides an interface for communication between the grain input view, and any of the grain windows (add, remove, and edit).
- The service is allowing the grain input view and grain windows to transfer data between each other.
- The secret is that input data is transferred from the grain windows to the grain input view into the table.
- Error checking is done by checking the input data in add and edit for grain exceptions.

Architecture of the system



Description:

Model - As in the previous iteration, our model still has a Grain Hierarchy to potentially allow for multiple types of grains, and a nozzle class, as well as the RocketMath class to run simulation calculations and produce a set of SimulationResults. We now have a Case class to represent the case of a motor, which goes into calculations. We also have a UnitConverter class and SimulationSummary to eventually provide details of a simulation on the GUI.

View - With the implementation of a GUI, multiple view classes were needed, including a GrainInputView which handles the grain table with the help of AddWindow, RemoveWindow, and EditWindow. We also have input views for Nozzle and Case, as well as GraphView, which is a wrapper class for the JavaFX ScatterChart. CSVConverter is still implemented as an optional feature, and RSEGenerator will be used to produce RSE files for simulation data in future iterations.

Controller - The command-line interface has been replaced with a GUI, which serves as the main controller for interaction between the user and the back-end. TableHandle was created as a means of communicating and transferring data between

4. Project Management

- Plan for the next iteration and the rest of the semester: For the next iteration, we plan to clean up the GUI, and have more features by adding ways to export and import data. We also have a logo that we plan to add for better presentation, for now we are using a first draft of the logo. We are also planning on implementing more types of grains, and allowing the user to switch unit systems.
- Track changes to design:
    - 10/23/17 - We added sub-packages and moved modules around to help organize our system, and added more files to break up the work. More of the changes to our code was refactoring and commenting.

5. Team

- Abhishek (Product Manager): Implemented Simulation Summary code. Demoed project to advisor and got feedback. Got new equations for an updated physical model. Fixed bugs in main sim loop.
- Isaac (Front-End): Implemented Unit Converter. Wrote tests to verify to loss of precision when converting metric to imperial and back to metric. Added functionality to the csv file button on the GUI.
- Daniel (QA Engineer): Implemented graph feature, as well as the GUI mapping of where components should be placed. Produced testing for RocketMath, which completes testing of individual back-end modules. Monitored GitHub repository.
- Andrew (Back-End): Developed the components on the GUI to create grains, and make GUI easier to use.
- Vicente (Team Coordinator): Ran the Trello boards, kept up with tasks, and set up weekly meeting for the group. Also transposed the RSEGenerator file, and tested the main functionality for the GUI.

6. Reflection

- What went well: We once again had very good communication among our teammates, and never had an instance where people had to do unnecessary work or two people worked on the same thing.
- What didn’t go well: We had to be a bit rushed in our 2nd iteration because we weren’t given much time from the first iteration. But we were able to stay on track by redistributing work as needed.
- Features that were not implemented: All 2nd iteration goals were met.
- How we plan to overcome the issues we encountered: We planned to shift the focus of certain tasks to the more important/difficult ones, allowing for more collaboration on certain issues.
- What we will do differently for the next iteration: Next iteration we plan to stay more on track when it comes to the due dates, and not make the crunch time at the end.