

## OpenBurn: Internal Ballistics Simulator

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

## 1. Executive Summary

### Overview

- OpenBurn is an open source, multi-platform solid rocket motor internal ballistics simulator that takes a fuel model, and grain geometry and produces a pressure vs time plot.
- The goal is to replace the outdated and closed source program BurnSim with a newer more versatile program that was more user friendly.

### Highlights

- Our application is what we set it out to be, with the exception with some minor features. We gave our app a user-friendly interface and accurate calculations, and maintained a solid team dynamic, with constant team meetings and an issue board.
- We changed a bit of the requirements around due to time constraint, but it was nothing to do with the main functionality of the app. We broke up the model section of our app, because the backend involved a lot more files than originally anticipated. This happened early on, around 9/22-9/29. There were some changes to the team, but only in the roles. This is just because we didn't want to keep only certain people on certain roles because there were certain points where everyone knows they need to focus on one aspect of the app, and the other parts are good for a while. There was no change to the process.
- The team was not able to implement erosive burning into the model. We needed more time than anticipated making the backend extendible and creating a GUI that would feed it relevant data. As such our program now is in a state where we could implement transient state/erosive burning with one class in the backend and one in the front end.

## 2. Customer Value

- **Target Customer:** University of Arizona IREC Team, rocket enthusiasts.

### Unmet Customer Needs

- Better simulation results than those that are currently available that are also open-source, and accounts for erosive burning. Improve upon graphical user interface.
- Helps design better motors and save on expenses through simulation over physical testing.

### 3. Problem Definition

- Multi-platform, open-source software, introduces multiple unit availability, scalable, improved Graphical User Interface.

#### Proposed Benefit

- Java implementation, open-source equations from NASA.
- OpenBurn provides a visual editor to design and simulate rocket motors via internal ballistics code.
- The major benefit of this software is the ability to simulate motors and export the files to standard formats. The free and open source aspect is new to the community and the code can be extended as needed.

#### User Stories and Use Cases

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}.

#### Changes to the Solution during the Project

1. Date of the change: 11/14  
Motivation for the change: No longer needed  
Description of the change: The ability to show multiple tables on the GUI
2. Multiple grain types no longer available.  
Date of the change: 11/02  
Motivation for change: Not worth the effort because it will not be used often  
Description of change: Stuck to one grain type, instead of having multiple grain types

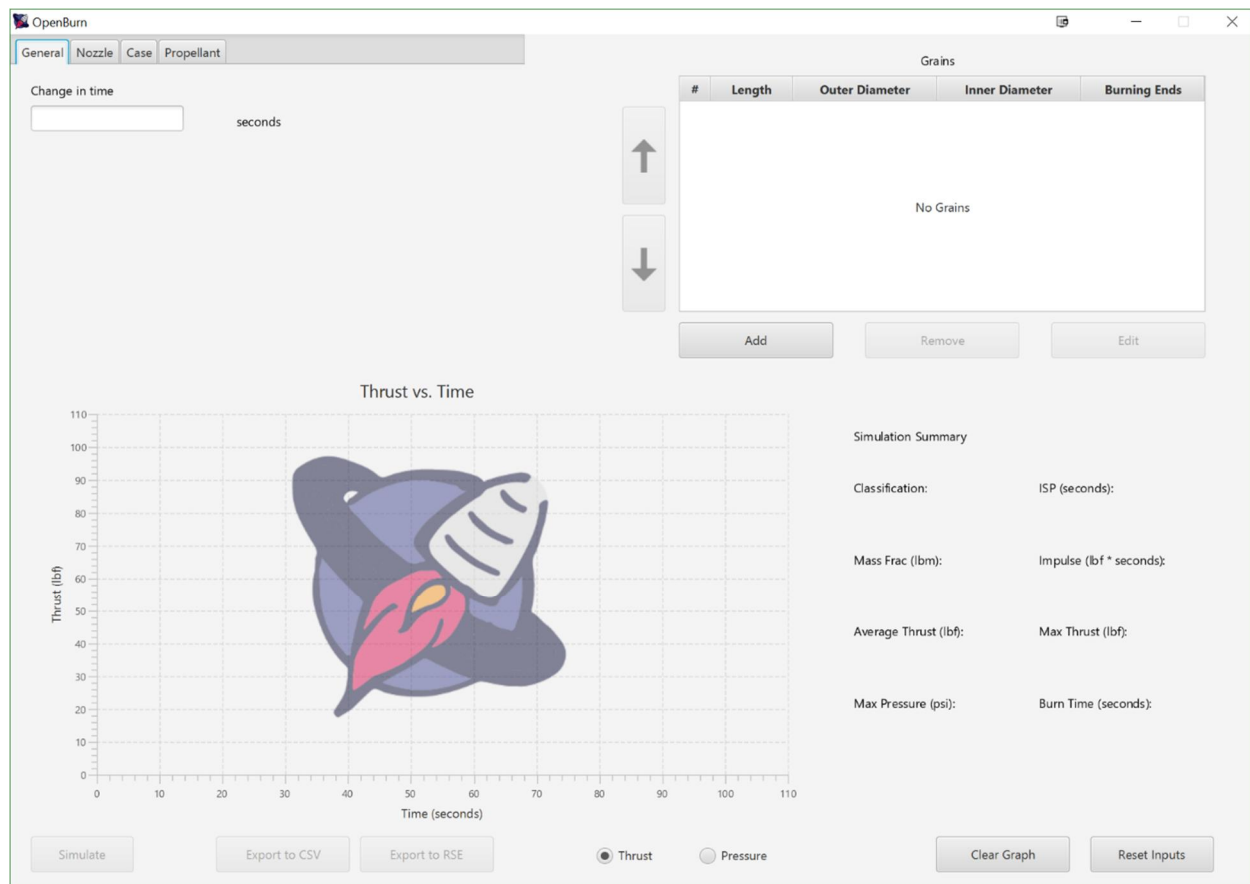
## Measures of Success

- Dr. Mark Langhenry – Current Raytheon propulsion, Former Lockheed Martin Propulsion Engineer, Former AIAA Solids Chair.
- We will test results against actual rocket motor data.
- We will produce a program with an easy to use interface that outputs accurate Thrust vs. Time data for a range of rocket motor designs.

## 4. Technology

### Overview of the System

System overview and acceptance test 1:



The first start image above shows default screen when the GUI jar file is first launched. The screen is divided into a few sections.

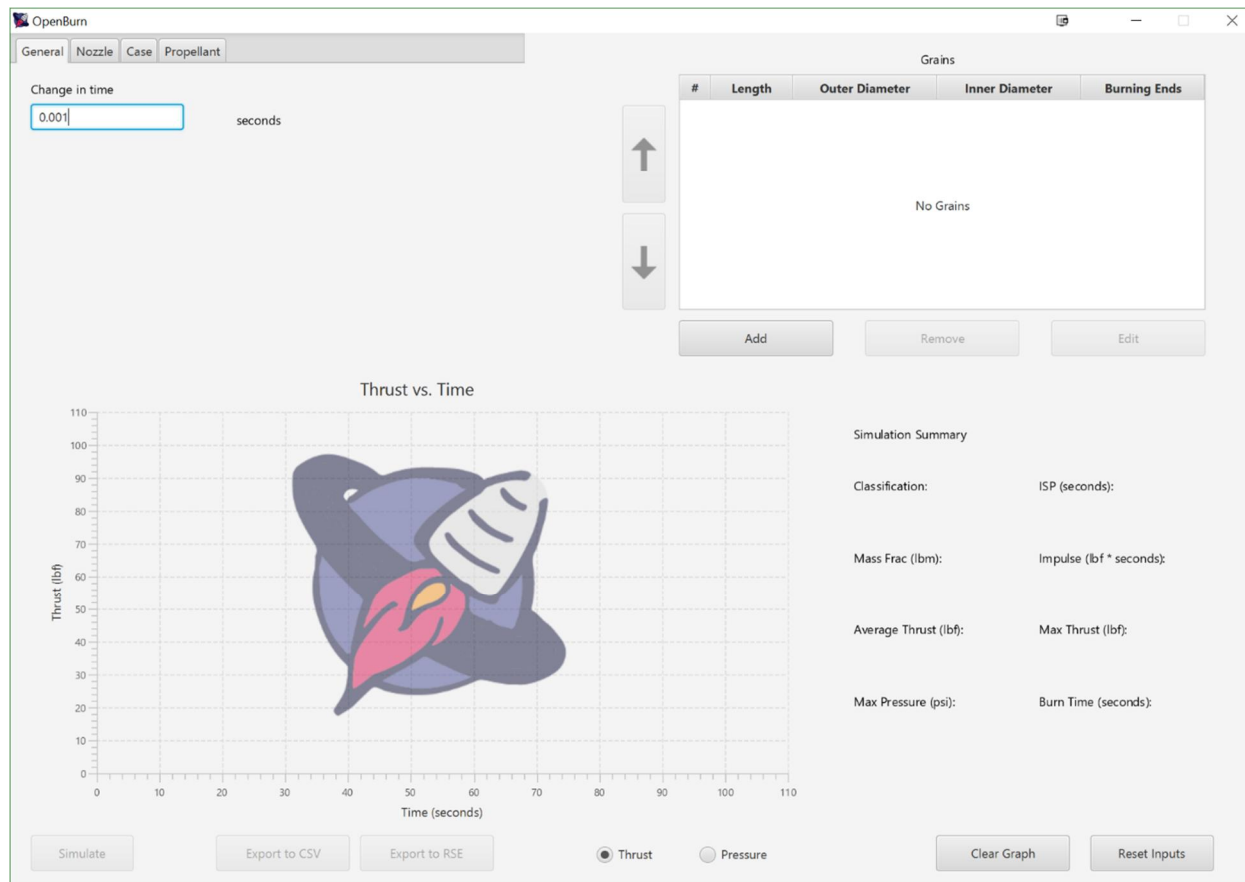
In the top left we have a tab view containing tab panes for all the inputs. Inputs are grouped with the physical part they represent. The general tab that is visible controls internal simulation options. As of now only the timestep can be changed.

On the top right, the Grain selector view is visible. Here new grains can be added or removed. A grain can be selected and its location can also be changed. The Grains on top of the list are closer to the aft end of the motor, while the ones on the bottom are closer to the nozzle.

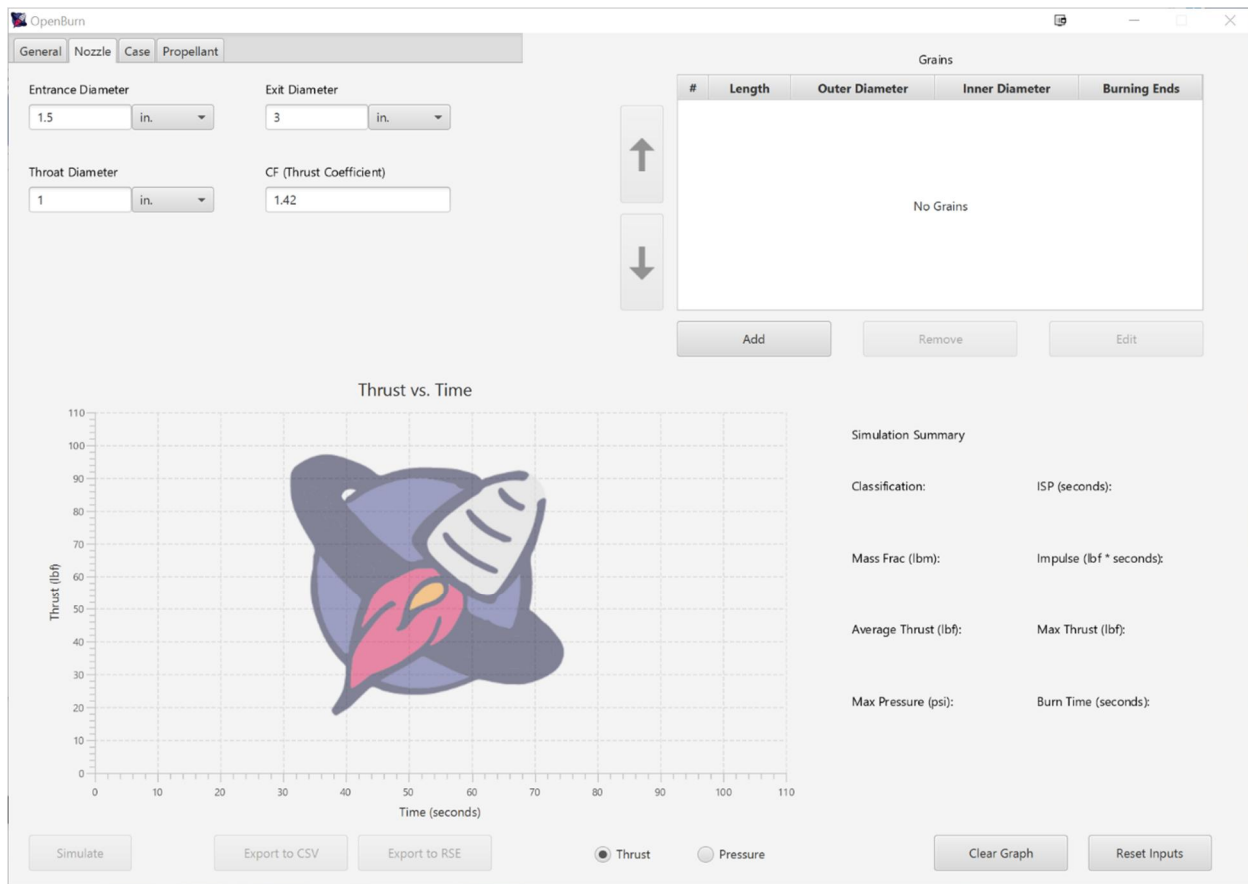
- The bottom left has the graph view. On first start the icon is shown.
- The bottom right has the simulation summary which shows some key states of the motor.
- The bottom bar shows the controls for the GUI.

- Simulate runs the simulation, it is greyed out until all fields are filled.
- Export to CSV generates a CSV file of all the data generated. It is greyed out till a simulation is run.
- Export to RSE generates a Rocksim readable engine file. It is greyed out till a simulation is run.
- Thrust and pressure toggle between graphs. This will be expanded on post release.
- Clear graph clears the graph but leaves the values in the input fields alone.
- Reset inputs clears the input fields.

Example walkthrough:

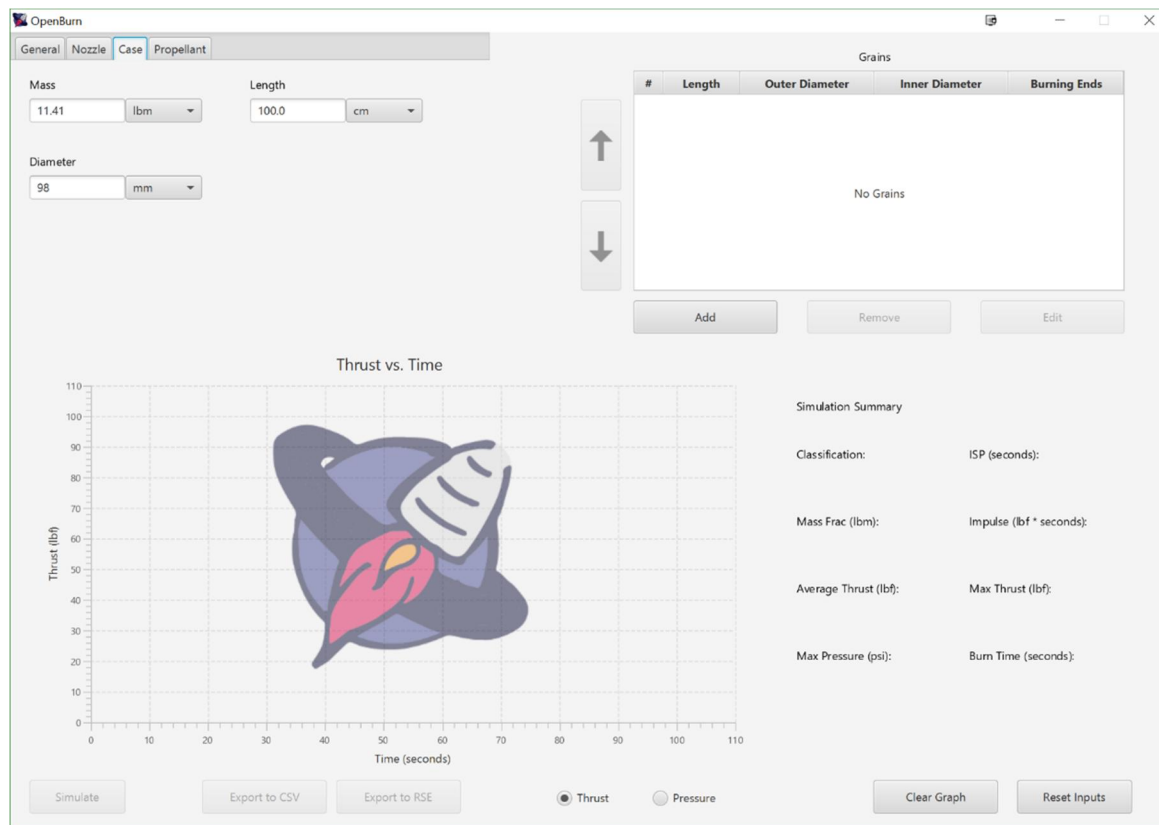


We can set the timestep as shown in the image above.

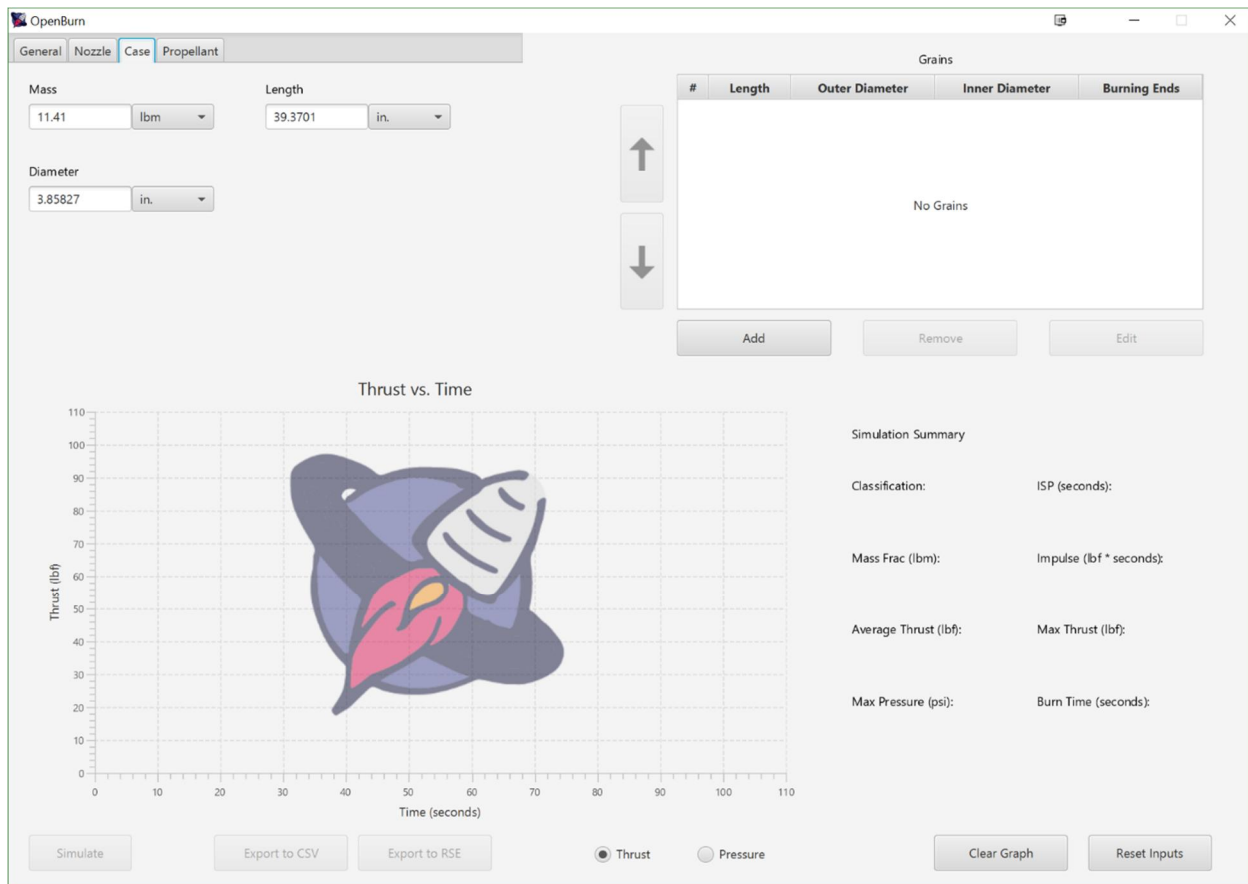


The nozzle view above lets us select some nozzle characteristics. Entrance and exit diameters don't do anything in this build but will be used for mass flow and automatic CF calculations. We can manually set the values of throat diameter and CF.

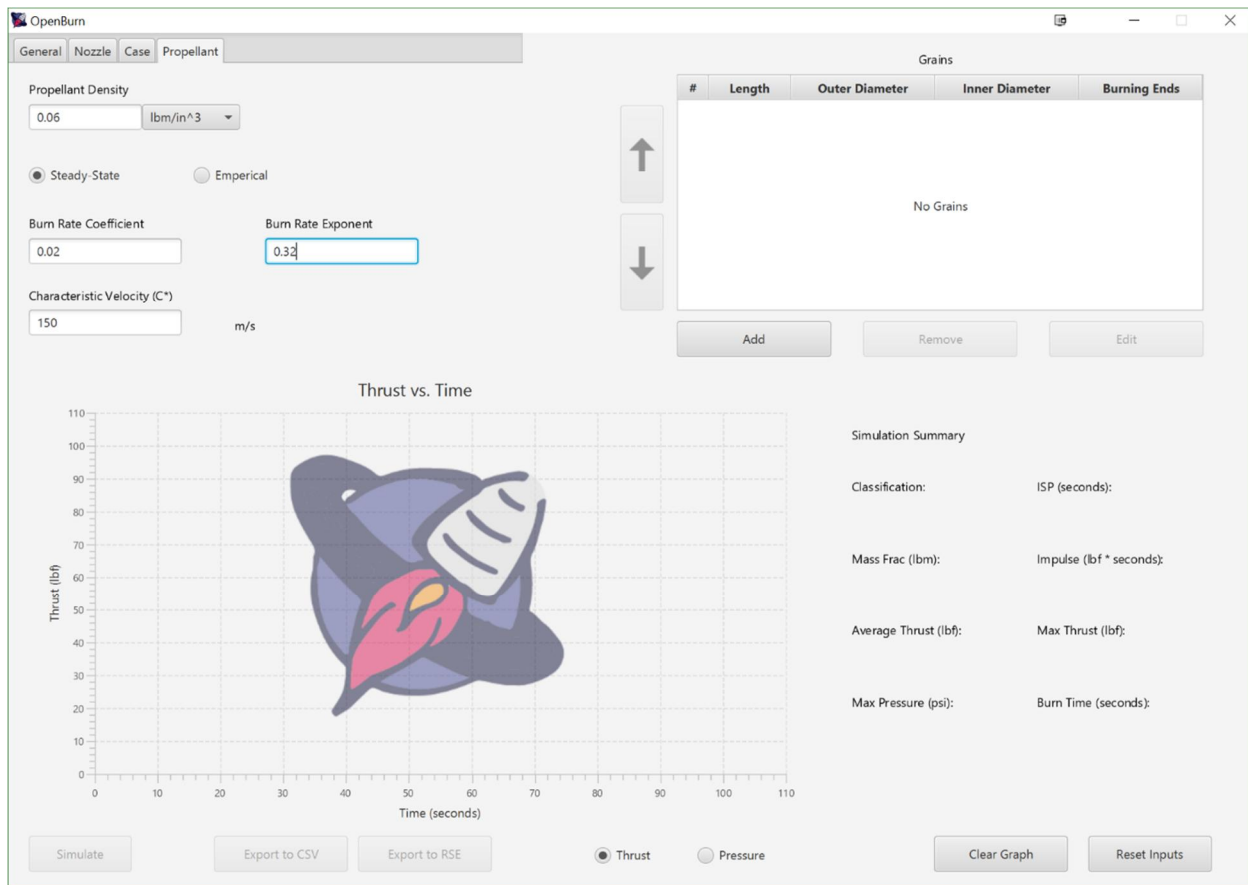
Case:



The case view above allows one to enter the case geometry. This is only used when generating a Rocksim engine file, however the simulate won't be available unless they are filled out. The case being described here is the Cessaroni ProX 98mm 6-G case, an off the shelf case common in large rockets. This view also demonstrates the mixed unit input functionality available.

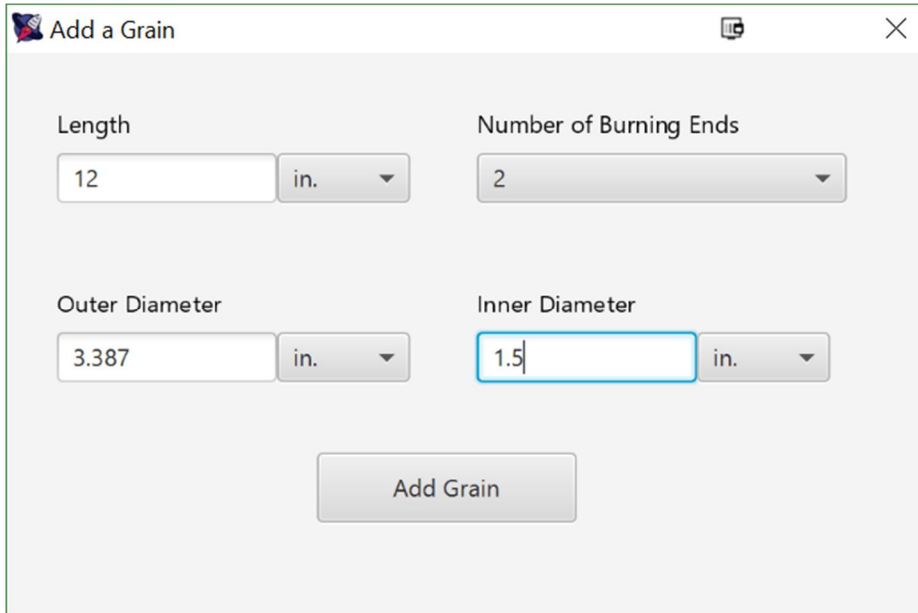


The view above is of the case view. The lengths are the same as the last image, but the units are changed. If there is data in a field and a new unit is selected, the data is automatically converted to the new unit.



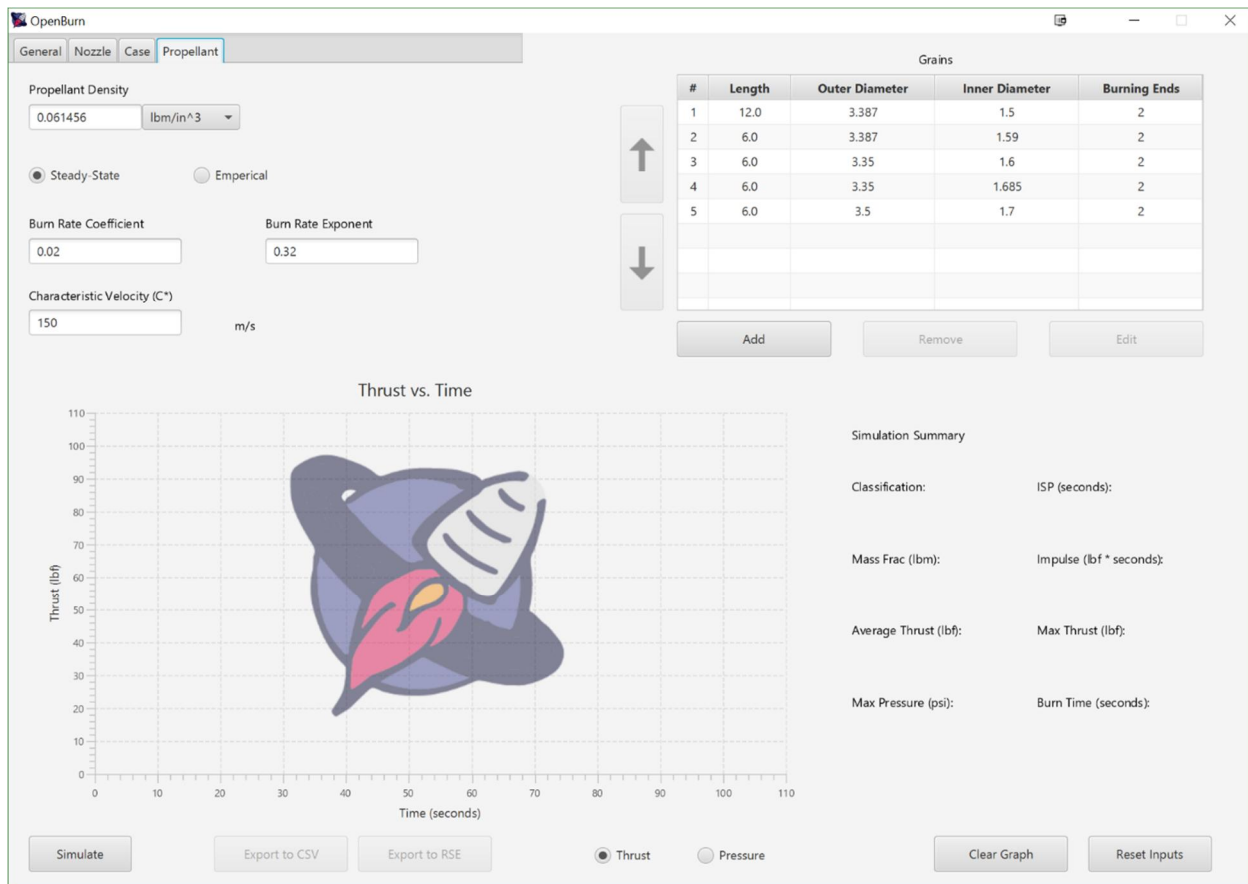
The propellant view allows one to enter propellant model data. The standard view shows the steady state view, which shows the most common ways of reporting data. The other method is to use empirical data. The empirical data view is mainly for data generated by a past UA team and is only included for legacy purposes.



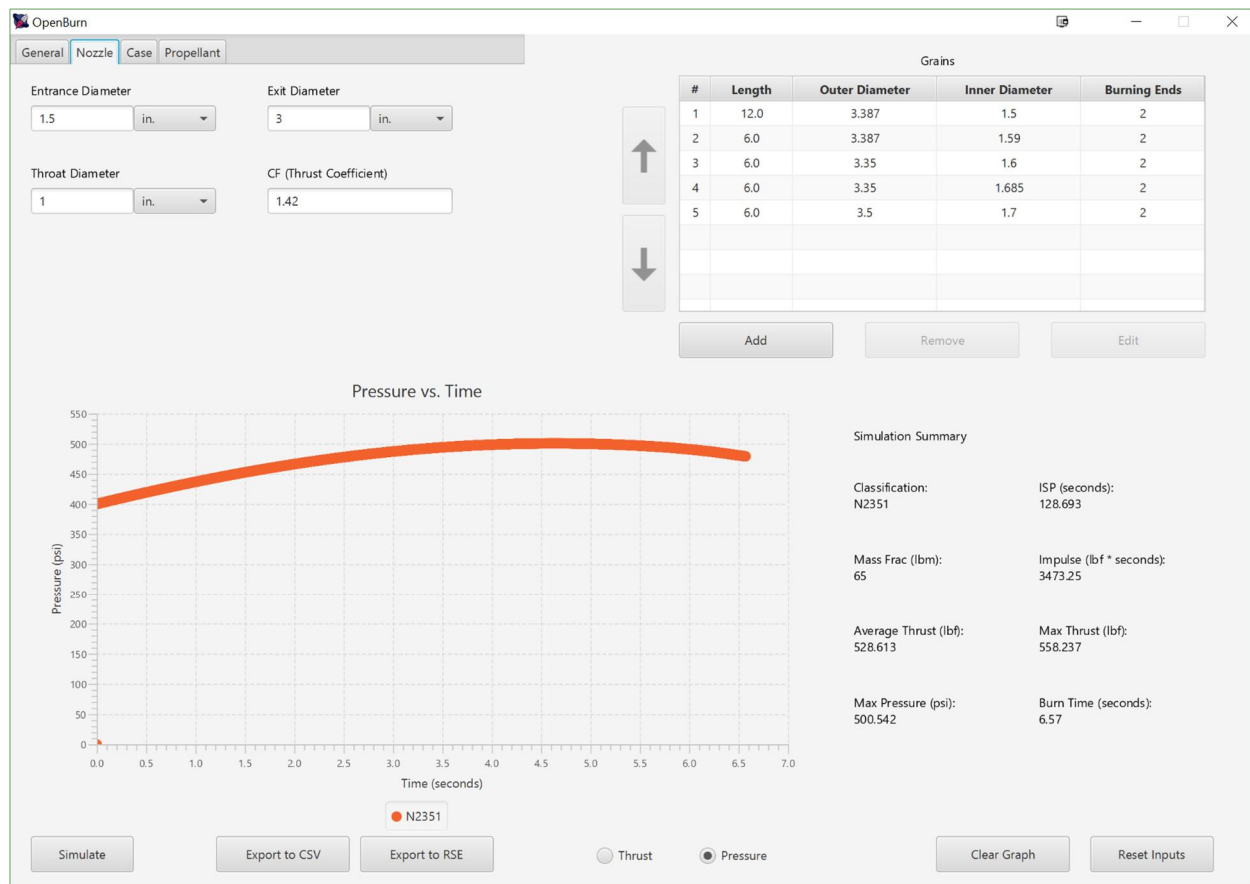


The screenshot shows a window titled "Add a Grain" with a close button in the top right corner. Inside the window, there are four input fields arranged in a 2x2 grid. The top-left field is labeled "Length" and contains the value "12" with a unit dropdown menu set to "in.". The top-right field is labeled "Number of Burning Ends" and contains the value "2" with a dropdown menu. The bottom-left field is labeled "Outer Diameter" and contains the value "3.387" with a unit dropdown menu set to "in.". The bottom-right field is labeled "Inner Diameter" and contains the value "1.5" with a unit dropdown menu set to "in.". Below these fields is a single button labeled "Add Grain".

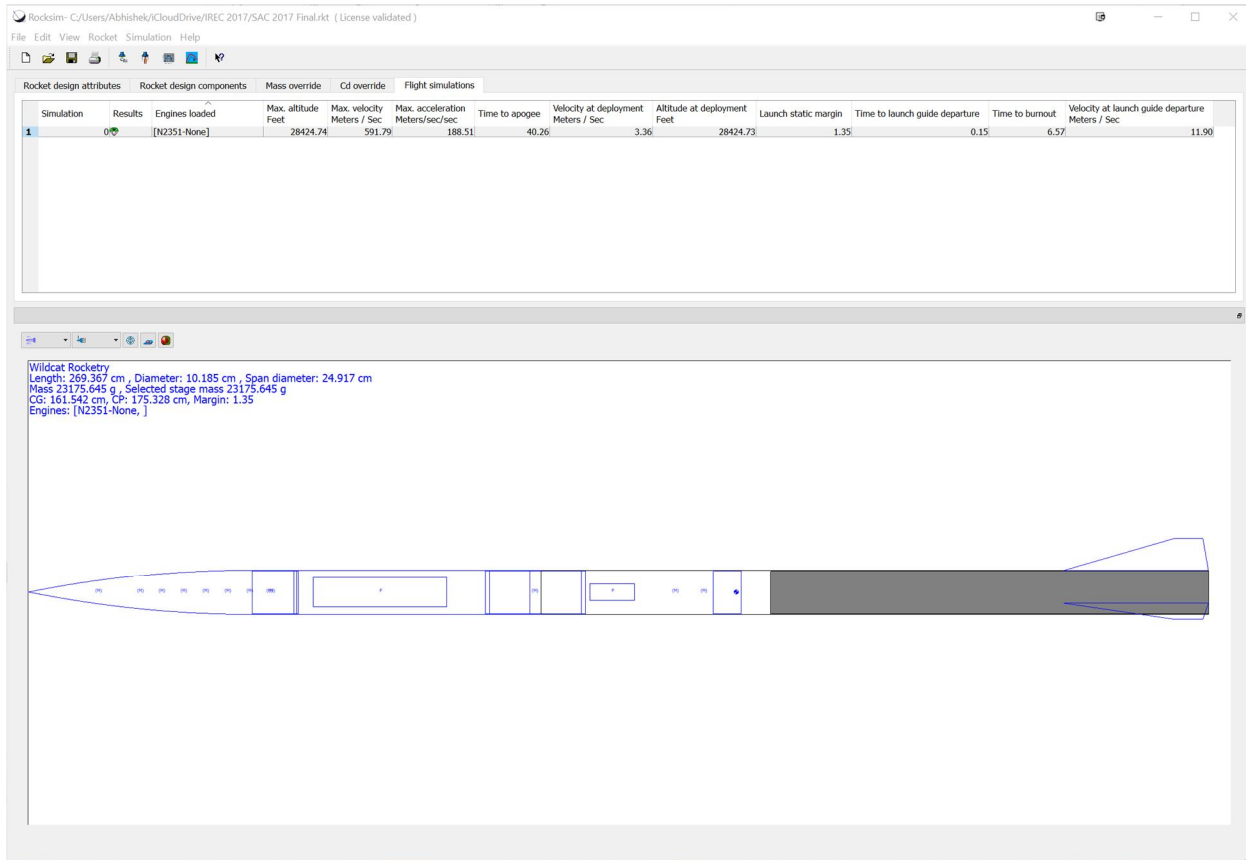
The grain input view above shows the fields available for a motor. The only currently implemented inputs are for a cylindrical grain. The burning ends indicates the number of burning ends available. The other 3 govern the shape of the grain. A sanity check is done when add grain is pressed. For example the outer diameter cannot be smaller than the inner diameter.



The above image shows the GUI right before the simulation is run. The simulate button is no longer greyed out and grain table is populated. The configuration described above shows the motor built by the University of Arizona AIAA team for 1<sup>st</sup> annual Spaceport America Cup.

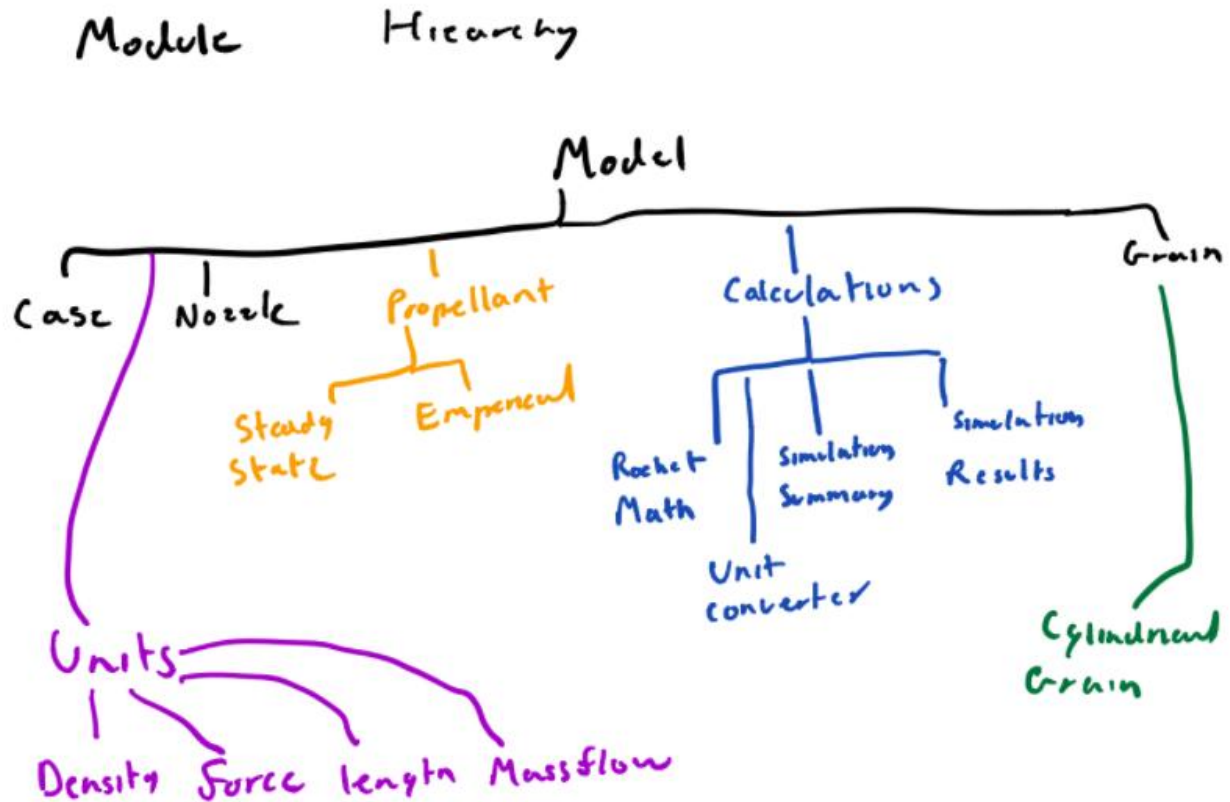


Running the simulation we get the above view. The graph view is populated with the right data and the simulation summary gives a quick view of the results. When building a motor, a designer generally has a few key parameters to meet and the simulation summary provides them. The mass fraction output is currently not functional. The export buttons are also functional and present the default file explorer for the OS that is being used.



This view is of Rocksim. The engine file was imported into its database and was selected for the simulation.

## Module Hierarchy



## Module Guide

### Case (Model)

- Stores the geometry of the case hardware
- Provides the storage for various data representing the state case
- Hides any checks for the case
- Negative numbers will produce incorrect results.

### Nozzle (Model)

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

### Propellant (model.propellant)

- The superclass for propellant. Provides the interface to be used by the simulation loop.
- Handles the data for each propellant
- Hides any checks for the case
- Negative numbers will produce incorrect results.

**EmpericalPropellant (model.propellant)**

- Implements the equations for getting chamber pressure and burn rate via empirical data point inputs. This primarily functions as a legacy input for some data that the UA team has.

**SteadyStatePropellant (model.propellant)**

- Contains the inputs in a standardized format for a propellant model. This is how most propellant is reported in industry and as such this is the default model used.

**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 calculations.
- The secret is the formulas and calculations being done in the back end.
- Errors - calculations with negative numbers.

**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.

**Cylindrical Grain (Model.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.

**Grain (Model.grain)**

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

**DensityUnits (model.unitConversion)**

- Enum for the density units that can be used in the GUI.

**ForceUnits (model.unitConversion)**

- Enum for the force units that can be used in the GUI.

**LengthUnits (model.unitConversion)**

- Enum for the length units that can be used in the GUI.

**MassFlowRateUnits (model.unitConversion)**

- Enum for the mass flow rate units that can be used in the GUI.

**PressureUnits (model.unitConversion)**

- Enum for the pressure units that can be used in the GUI.

**Unit Converter (Model.calculations)**

- Provides conversions between imperial and metric.
- Provides static functions which accept a double and return a double.
- Hides the mathematics of unit conversion.
- Due to every number being a valid measurement, there is no meaningful error checking.

**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.

**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 provided 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.

**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.

**PropellantInputView (View)**

- Contains input fields for propellant in a tab in the OpenBurnGUI.
- The service this provides is allowing the user to change the type of propellant they want.
- The secret is the math behind the propellant model.
- This class checks for the proper inputs for each propellant.

**RSEGenerator (View)**

- This file layout and formats the RSE file.
- It generates the file based on the data provided in the simulation.
- The secret about it is the formatting that is needed for an RSE file.
- This handles error that come from improper data.

**SimulationSummaryView (View)**

- This displays a summary of the simulation results on the GUI.
- This takes the data and condenses it to a more readable format.
- The secret is where the data is held.
- There is no error handling for this.

**Graph View (View)**

- This takes the output of the backend calculations. This includes time and pressure.
- The service provided 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.

**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 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.

**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.

**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.

**OpenBurnGUI (Controller)**

- When the calculate button is pressed, gets data from input text fields and constructs motor and grain objects.
- Passes them to static functions from the RocketMath class. Passes the output into the GraphView class.
- The secret is where we store the data and process it
- This does a lot of error checking, such as making sure every input field is completed.

**CMDLineInterface (Controller)**

- Used to produce accurate results that output to csv file during iteration 1 before the main gui was developed. No longer used in the final release.

**DensityUnitsSelector (Controller)**

- Changes the type of density units to the user's preference.

**GrainTableHandle (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.

**LengthUnitsSelector (Controller)**

- Changes the type of length units to the user's preference.

**MassUnitsSelector (Controller)**

- Changes the type of mass units to the user's preference.

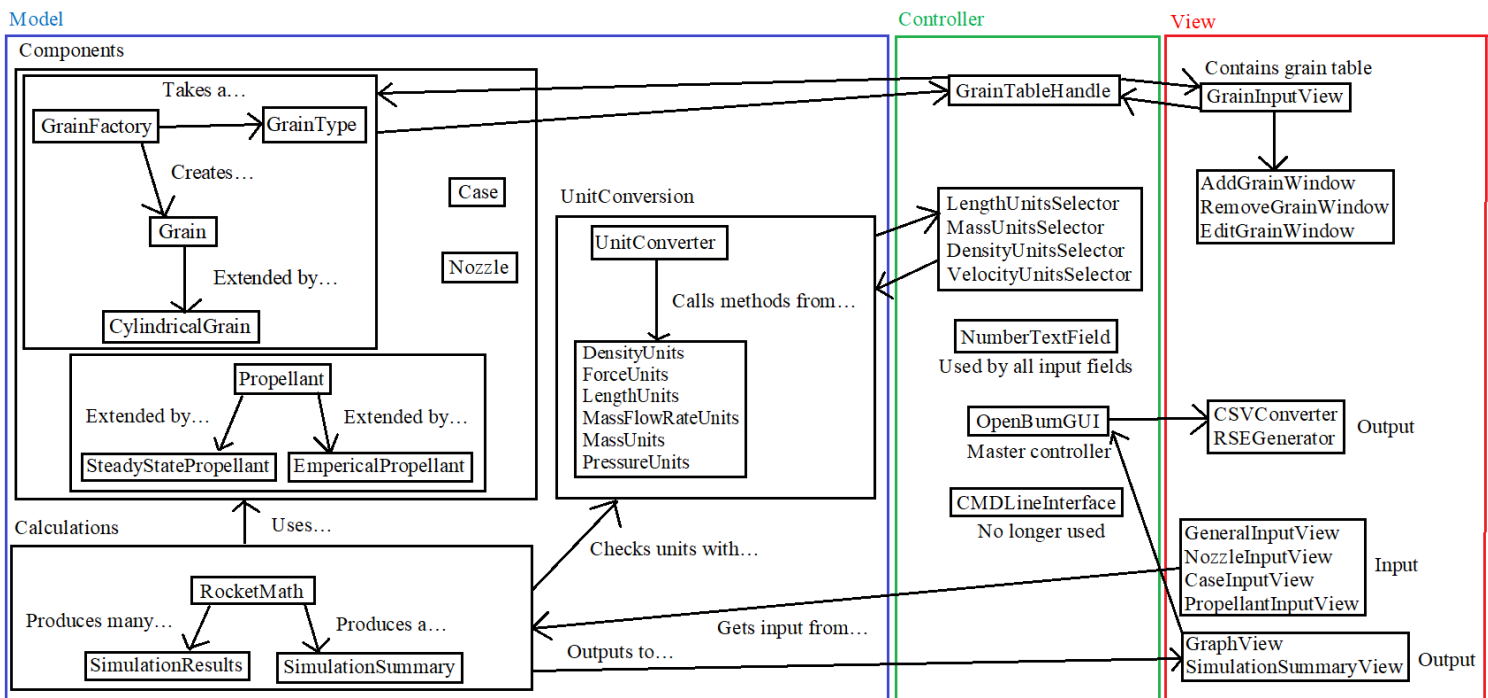
**VelocityUnitsSelector (Controller)**

- Changes the type of velocity units to the user's preference.

### NumberTextField (Controller)

- A text field that only allows numbers and one decimal point to be input
- This provides the service of handling the numbers and data input to certain parts
- The secret is how the data is handled
- This checks for any inputs that are not numbers

### Architecture of the system



**Model:** The model is split up into 3 parts: Components, Calculations, and UnitConversion. The components include Grains, Case, Nozzle, and Propellant, all of which are created using inputs from the input views (See View) during a simulation. The Calculations represent the output of the application. When a simulation runs, methods from RocketMath are called to produce a list of Simulation Results and a Simulation Summary to be displayed on the GraphView and SimulationSummaryView. UnitConversion is used to convert units in the text fields.

**View:** Input is given using the many input views – GeneralInputView, NozzleInputView, CaseInputView, and PropellantInputView – all of which are encased in a TabPane (JavaFX). The GrainInputView contains the grain table and communicates with the Add, Remove, and Edit Windows via the GrainTableHandle (See Controller). When a simulation is run, the results are displayed on the GraphView and the SimulationSummaryView, then the master controller (OpenBurnGUI) can optionally save data using the CSVConverter and RSEGenerator modules.

**Controller:** The master controller of the entire application is OpenBurnGUI, which contains main(). The main GUI contains all input and output views, along with export buttons for CSV and RSE. NumberTextField was created to only allow numbers in all inputs. All the unit selectors extend ComboBox (JavaFX) to allow the user to convert between units in real-time. The GrainTableHandle provides an interface for the grain table to communicate with the Add, Remove, and Edit Windows. Lastly, the CMDLineInterface module is no longer active in the final release, but was included since it was active in the Viable system release.

## Tools to Build the System

- Eclipse, JUnit, JavaFX, CSV/Excel files, RSE files, GitHub, JRE.

## 5. Team

### Backgrounds

- Backgrounds

**Abhishek:** Aerospace engineering and computer science experience. I was a project lead for the intercollegiate rocket team and I have experience designing and building solid rocket motors. I was responsible for the simulations related to the motor. I also worked at an autonomous drone technology startup in Tucson. I have used Java, C, MATLAB and have participated in several hackathons.

**Vicente:** I have majored in computer science my whole time in college and have some experience with GUIs and simulations. I am experienced with C, Java, and Javascript. I currently work at an internship with HealthTrio so I have a good amount of experience with working as a team and experience with Agile, Git and JIRA.

**Isaac:** Has participated in multiple game jams and hackathons. Experience with C, Java, C#, and Unity3D.

**Daniel:** Experience in C, Python, and Java, along with 12 months of internship experience with Metropia Inc. (6 months) and currently at HealthTrio LLC (6 months). Has worked with Agile teams that utilize software such as Git and JIRA. An aspiring Full-Stack developer who can design systems and databases. Has also participated in two hackathons, and has some experience with drones.

**Andrew:** Industry experience with testing and releasing an application, experience with Java, C, HTML, CSS, and JavaScript.

### Skills / Roles

**Abhishek (Product Manager)** – Knowledgeable of the product requirements and constraints. Worked on a lot of the back end calculations and main implementation.

**Vicente (Team Coordinator)** – Tracks progress for the team. Somewhat of a Scrum Master. Worked on files when needed and took time to plan out meeting and team plans

**Isaac (Front-End Developer)** – Also worked on a lot of the back end and main implementation, also took care of a lot of bug fixes and other needed code.

**Andrew (Back-End Developer)** – worked on some back end files and front end files, also linked them to work with each other. Worked a lot on controller

**Daniel (Quality Assurance Engineer)** – Main designer of the GUI and graph view. Also helped refactor code and found bugs to be fixed a lot of the time.

**All members** – Development of certain components, and monitoring the work and progress of the entire team.

## 6. Project Management

### Process

- Our team used Agile development to stay on track with releases and make it so the whole team was assigned a task. We will use Trello and GitHub to drive our Agile process. We typically tested modules as we wrote them similar to a V-Process.

### Schedule

- **Viable** - CMD program that receives inputs of fuel shape, type and length and outputs a CSV of Pressure vs. Time. Supports Imperial units. Calculation assume steady state.
- **Feature** - GUI that produces a visual graph of Pressure vs Time. Can save data to a CSV. Imperial and metric units.
- **Beta** - GUI that produces a visual graph of Thrust vs Time along with Pressure vs Time. Can save data to a RSE file. Also can clear the graph and reset all the inputs for all the current data. This will also account for different propellant types. (This got deleted)
- **Final** - Calculations support transient state calculations (same here), can display a summary of simulation results.

### Team Meetings

- Our team has meet 2-3 times a week in person. Also we will use the GroupMe app to stay in contact, along with Facebook messenger and calls to stay on the same page.
- Additionally we will use Trello to keep track of tasks and their completion. This helps break down each iteration and clearly state what needed to be done for everyone
- We discussed the status of our individual parts. How close we are to implementing the next iteration, and if we need help to implement our individual part.
- We had all our team members give updates on the progress of the current tasks. Also plan what tasks need to be done next.

## 7. Reflection

- Reflection is an important section. Include lessons learned from doing the project. This is the place to look back over the entire project.

### 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.
- **Best Practices:** Weekly team meetings, Git Branching, Trello ticket system and instant messaging.

### 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.
- Our team seemed to have a low number of people working on features at a time, which caused some implementations to get done slower than expected. We would plan emergency meetings to re-discuss iteration goals and get the team back on track.
- Erosive burning and multiple grain types were not implemented due to lack of time and some slow implementation completion times.

### Recommendations?

- **What we would do differently:** We would be more realistic with our iteration goals and not overestimate the time it takes to implement a feature or debug.
- **Advice for others:** Be realistic with your team's experience levels and abilities, and try to find a project that fits those specifications.