Motivation

- The simplest approach is to actually calculate the force of each star on every other star.
- If you have n stars, this will take roughly n^2 calculations to finish.
- existing software:
  - AMUSE combines many implementations.
  - AMUSE is very complex.

Overview

Astrophysical Multipurpose Software Environment (AMUSE)

- Here is our architecture diagram alongside AMUSE's architecture.
- AMUSE uses a library called MPI to gather physics code written in many languages under one python interface.
  - Codes include gravity and stellar evolution to name a few.
- Our software provides a framework that builds on AMUSE to generate and run experiments.
- The interface lets the user put the experiment together.
- The experiment generator lets advanced users customize details.
- The network layer gives the user a view of how the cluster is being used.
- We provide a storage API to share experiments.

State of AMUSE

- Partnership between Drexel and the Leiden Observatory in the Netherlands, sponsored by NOVA.
- NOVA = Netherlands Research School for Astronomy
- Mention large scale again
- Written by hand = hard to share
- Waste of work to replicate someone else's diagnostics to fit your exact circumstances.
- Code to the right is FORTRAN from AMUSE's community codebase.

Purpose of GPUnit

- Ease the creation, execution, and analysis of experiments with AMUSE
- Create experiments with minimal to no programming
- Repeatability
- Sharing Experiments
- API for results / diagnostics

Target Audiences

- Physics Student
  - Minimal to no programming experience and minimal knowledge of astronomy
  - Has an interest in learning and performing experiments
- Observational Astrophysicists
  - Not much programming experience
  - Good understanding of astronomy
  - Interested in reproducing and analyzing observed phenomena
- Theoretical Astrophysicists
  - Significant programming experience
  - Good understanding of astronomy
  - Interested in creating their own experiments with custom analysis code

Features

- Explain how features satisfy requirements.
- Configurable experiments -> less programming.
- Diagnostics -> common API for metrics
- Code is generated to run actual experiment -> advanced users can tweak it
- Storage of state -> repeat experiment if it crashes

Design

- We settled on Python because AMUSE is a Python library, interaction is streamlined.
- If we had used C++, AMUSE would run in a separate process, introduces unnecessary disconnect between our code and AMUSE.
- Challenges:
  - Figuring out how AMUSE works.
  - Making a useful tool that simplified experiment creation without taking away any of AMUSE's power/features.
  - Allow future developers to expand on this work:
    - Modular diagnostics w/API to do the work
    - Experiment storage abstraction: allows for remote backup, sharing of results

Tests

- Table of tests that pass.

User Testing

- Tested with customers (Steve/Tim)

Project Plan

- AMUSE codebase is large and complex (as we have mentioned)
- Before we could plan our project we needed to figure out how AMUSE worked.
- Learning continued throughout the project.

Team Management

- Bi-weekly team meetings helped get a lot of work done
- Able to code and discuss at the same time in person (useful)

Project Impact

- Researchers can discover important things much faster when they don't have to fuss with experiment boilerplate.
- Students can learn about what astrophysicists really do first-hand without going too deep into complicated issues.

Demo

- Demonstration of a simulation.

Questions

- Questions?