Project Proposal: Manim

Name: Nick
Proposed System Under Test (SUT): manim
Link to SUT Source Code: https://github.com/3b1b/manim
SUT Size: 15,568 lines of code (about 200,000 lines of code are provided as examples)

SUT Description:
Manim is an animation engine for explanatory math videos. It's used to create precise animations programmatically (with python), as seen in the videos at 3Blue1Brown.

State of SUT:
Manim is pretty well featured and capable of creating a wide variety of animations to describe math. However, a lack of documentation keeps the project from being widely accessible. And nearly all of the library is untested.

Attributes:

- **Precise:** Animations should describe math functions as exactly as possible
- **Consistent:** Creating animations for the same program should yield the same animation
- **Streamlined:** It should be easier and simpler to create math animations with manim rather than traditional animation software
- **Featured:** It is possible to create animations for any math function / topic
- **Pretty:** Animations should look nice

Components:

- **Build System:** finds the necessary dependencies and compiles the project
- **MObject:** mathematical objects that form the 'actors' of an animation scene
- **Scene:** An animation scene that holds Text, MObjects, and Cameras, and describes how they change
- **Camera:** Describes which MObjects are viewed and renders them

Capabilities:

- **MObject** is **Precise:** MObjects hold the exact mathematical function they describe, as well as instructions to visualize said function
- **MObject** is **Consistent:** the visualization instructions held by an MObject yield a single, unique animation

Unit Tested Capabilities:
Unit tests will be written for the MObject file. By extensively testing this file, we can effectively cover almost all of the most commonly used components of Manim.

Automatically Tested Capabilities:
Property based testing will be used to test MObject transformations. Because it is the root class every MObject inherits from, it is critically important to make sure that the transformations it performs are correct. Additionally, because of the 'mathy' nature of the library, there are a large number of properties to use in a property based approach. To achieve property based testing, I will create a generator that creates an mobject of a maximum depth with a bunch of points. I will then apply some transformation, and check if a property holds.