Week 1:

Describe at least 2 desirable "cutting points" for testing (see lecture topic 2.5) covering main paths and/or boundary condition handling.

1: I want to get the display able to display a little 2px x 2px dot which I want to move around by simply pressing the buttons. I can use some of Lab 7 as a framework to get started. Once I am happy with the display, I want to get the basics of the physics engine working ASAP so I can continue to make the game with that out of the way.

2: Testing and tuning the physics engine will be necessary, and I would like to get it solid as soon as possible. With the display functional, I will test a single slug's trajectory and check that it matches what I'd expect by doing the math myself and comparing. Once the basic physics calculations are there, I can move on to the position data manipulation that most of the other tasks depend on.

Week 2:

Create your unit testing plan, utilizing at least 3 "cutting points". List and summarize 2 conceptual unit tests for each "cutting point" in the summary after the Unit and Functional test sections.

1: Modular drawing functions

a. Being able to draw all sprites needed for the project (slug, platform, satchel, castle, walls) at any position. It should be able to feed in any coordinate and have sprites drawn there.
b. All sprites should be able to be drawn in a sufficiently short time. Once I find how long the set of sprites takes to draw, that time should be more or less constant, and finding if this time is too long will require shortcuts to be found in drawing.

2: Physics task

a. Pause at any point and verify velocity and positional numbers using the time elapsed and constant acceleration. This could be coded into a set of simple kinematics functions that take the elapsed time, returns either velocity or position and use this result to compare against real measured values in an assert.

b. Add asserts for arbitrarily large values that the data should never exceed. Positional data can be checked for values over `~260 (because the real position may not exceed 256). Velocity maximum can be calculated based on unit-to-pixel scaling factor and kinematics equations, adding ~10% above that theoretical max. I want to make sure if my program goes off with crazy data at any point, I can catch that quickly in the debugging process.

3. State machine tests

a. I will be implementing a state machine to store the game's logical states and I will add tests to make sure the game state is what we would expect given certain parameters. For example, we cannot win if the hostages are dead and we cannot win if the platform has blown up. This will make debugging the gameplay more fluid because incorrect cases will be caught quickly.

b. As a part of the important state machine transitions, I will modify global variables to keep the potential of garbage values to a minimum. Testing that this happened properly would require asserting at the beginning of the task loops depending on the current state and data being examined.