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.

Week 3:

Review your described unit tests, and speculate whether each would Pass or Fail based on your current implementation. Add a section to your test plan for "functional tests", with 10 tests briefly described, and speculate whether they would pass or not as well. (Pass/Fail: likely 80+% will be "Fail" at this point. The P/F-ness does not affect your grade at this point--only describing the (hypothetical) unit tests, describing your functional tests, and indicating which tests would pass or fail will affect your grade at this point.).

60% pass

1. Modular drawing functions
    a. Is the input coordinate greater than the screen boundaries for any function? (Platform, slugs, satchels)
        i. *Pass* - My position in physics tasks has boundaries for each implemented subsystem.
    b. Is the display task being executed in a sufficiently small period of time?
        i. *Pass - the* task is under 20ms / 100ms and has sufficient CPU time for other tasks to run.
2. Physics task

    a. Are any velocity values running away from a reasonable range (EFM_ASSERT(xVel < 1000))
        i. *Pass* - For currently implemented features, velocities are reset consistently, and acceleration calculation is happening correctly.
    b. Are any position values running away from a reasonable range?
        i. *Pass* - Sprites “bounce” off of walls and ceiling of the display and stay within the allowable range of 0-128
    c. Is the physics task taking too long to run?
        i. *Pass* - The physics task is under 5ms, leaving plenty of runtime for other subsystems.
3. State machine tests (STATE MACHINE HASN’T BEEN IMPLEMENTED YET)
    a. Are game structs initialized before the game has started?
        i. *Fail* - No main menu has been implemented
    b. Is the game running during the correct “runGame” state?
        i. *Fail* - No sm implemented.
    c. Does the game correctly change to “gameFail” state once one of the failing end conditions is met?
        i. *Fail* - No game logic has been implemented.
    d. Does the game correctly change to “gameWin” state once one of the victory conditions is met?
        i. *Fail* - No game logic has been implemented.

Week 4:

Carefully specify your functional tests, such that another person could understand how to execute them. Summarize hypothetical-unit and real functional test results. ("NotRun" will make sense for many functional tests that are not possible until you get further)

Functional Tests:

1. To test the slider affecting platform position, place your figner on the far-right side of the capsense slider. You will notice the platform accelerate towards the right and bounce off the right side of the screen. You can move your finger to the far-left side of the capsense and notice the platform will accelerate to the left, causing the platform to slow down, stop briefly, and then start moving to the left. If the platform moves too fast and hits the wall it will explode and the game will end.
2. To test the railgun press and hold BTN1. You will notice the left-hand status bar increasing until you either let go of the button or hit the maximum allowed charge.

When you let go of the bar, you will also see that the generator's total charge will decrease and quickly charge back up.

3. To test the satchel charges, you can either wait for a satchel to randomly hit the platform or intentionally put yourself in its path. When the satchel hits you, you will notice that the game ends because you lost.
4. To test the game state, shoot the foundation of the castle enough to have LED1 start to blink. After it blinks for a moment it will be solid amber, indicating the evacuation is complete. Once the LED is solid amber that indicates you have won and the game will end.