Team Lead 3: Quality Assurance

By: Kristie Olds, Nathaniel Palmer, Jeremy Wisecarver, Garrett Wells and Scott Martin

The Unity Test Framework (UTF)

- How Unity developers and users test code
  - Edit-mode testing
    - Test that can be done before initializing a scene
    - Test are more simple, typically much faster
  - Play-mode testing
    - Test that check scripts at runtime (Update, loops, etc.)
    - Test can check more complexity, typically much slower
- UTF uses Unity's integration of the NUnit library
  - Unit-testing framework for all .NET languages (C#, F, PowerShell)
  - Based off JUnit (java testing)
    - JUnit is based on the principle of A TRIP

A TRIP

- Automatic
- Thorough
- Repeatable
- Independant
- Professional

Automatic

- All test should run “automatically”
  - Every test should run with one click
- Test suites
  - A collection of test cases
  - Typically all test are in a test suite
    - Occasionally test will be grouped by similar test (smoke test)
- Your team should be able to access test easily
  - Ensures that your teammates code can pass all test before they commit

Thorough

- Good test will test as much as possible
  - Cost-to-benefit ratio
    - Too much testing can be expensive
- Code coverage
  - Are all execution paths and methods tested?
    - All public methods should be tested
    - Private methods can be debated
- Scenario coverage
  - Are all situations being tested?
- Specification coverage
  - Are all requirements being tested?

Repeatable

- Test should produce the same results every time
  - Test should not need modification between uses
- Do not test parameters that are uncontrollable
  - Test would render useless
    - Time tends to be unexpected
    - Testing objects that may be deleted earlier in the code

Independent

- Test should test one thing at a time
  - Multiple assertions are okay
    - They should all test one feature
    - Test should pinpoint problem location
- Test should not rely on each other
  - There should be no order on how test should be run
    - A test suite should be able to pass all test in any order

Professional

- There should be equal amounts of test code and production code
  - Code should be well documented
  - Names should be intention-revealing
  - There should be no duplication
- Readability
  - Anyone should quickly be able to know what test do
  - Test should be symmetric

Additional Standards of Testing

- Test should be fast
  - Longer test = Less testing
  - Less testing = More errors
  - More errors = Longer debugging
- Don’t test code you don’t need to
  - Code you don’t own
  - Getters and Setters

Unit Test Definition

- Also known as component testing where individual units or components are tested.
- This is to make sure that all units of software perform as they are suppose to
- A unit is the smallest testable part of any software
    - Ex: Function, Procedure
        - In OOP a method is the smallest unit
    - Fake objects are used to help unit testing
    - You are essentially using small bits of code to check your code

Why should I use unit tests?

- It’s an easy way to make better code faster
- It helps instill confidence in the coder
- It promotes consistency
- It will just make life easier

Unit Tests

- Unit tests should follow as you saw earlier A TRIP
  - Automatic
  - Thorough
  - Repeatable
  - Independent
  - Professional

Automatic

- You shouldn't need to know how the rest of your group's code works
- The unity test framework is there to make sure that everything is working
  - It will automatically tell you if you broke any of the code and can tell you if everything works as intended
- The unity test framework also allows you to quickly and easily run all of your tests with the press of a button

Thorough

- Your tests should cover all of your boundary cases
- Boundary case Def:
  - The behavior of a system when one of its inputs is at or just beyond its maximum or minimum limits
- Your tests should also cover anything you think might be liable to break
- How thorough your testing is in many cases is a pure judgement call
  - Do not test “Getters” or “Setters”

Repeatable

- Your tests should not have variables that are out of your control
- Your tests will be rendered useless if the results of the test change regularly
- Exceptions might be random variables in your stress test
  - Do not make the failure condition directly dependent on the random variable

Independent

- Your tests should only be testing one thing at a time.
- If you don't only test one thing at a time you will run into a specific problem
  - You won't know precisely where your code failed a test
- Unit-testing being independent tells you what you broke
  - Either in someone else's code or your own

Professional

- Unit-test code should be written professionally
  - Code we write for unit tests is just as real as the code in the final product
- If you find yourself reusing code over and over you should use methods
  - Fight the urge to treat test-code as a second class type of code
- The test-plan part is a part of the oral exam
  - Dr. BC will be looking for professional looking code

Other requirements of your Unit-tests

- Unit-tests should run very fast
  - The longer unit-tests take to run the less often they will be run
- The Unit-tests should be readable
  - The less readable a unit test is the more difficult it is to fix issues when they arise.

Unit testing weakness

- How do you test your unit-tests?
- You do not unit-test your unit-tests
- Deal with unit-tests when you fix bugs
  - If a bug isn't caught by the unit-test this bug went through a hole in your unit-test safety net
- Try introducing bugs yourself to find weaknesses in your unit-tests

Without unit tests

- You are building a “house of cards”
- The time to complete a project becomes unpredictable
- You are required to know all of your code forward and backwards
- We don’t know if the code is doing what is required of it
- Boundary conditions might not be functioning properly

Install Unity Test Framework

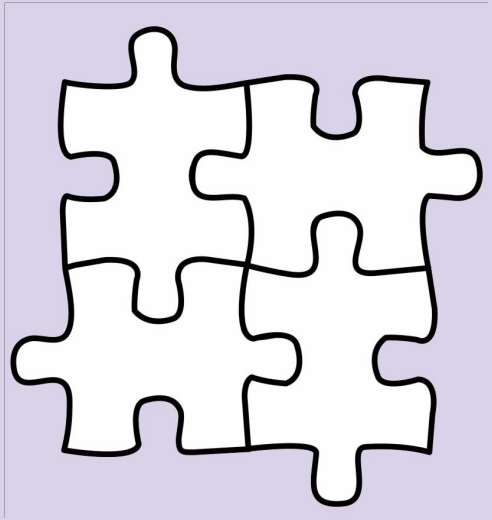
1. From open project, select Window → Package Manager
2. From sources select “Packages: Unity Registry”
3. Search “Test Framework”
4. Install

Unit Test Example

- Set up a Unit Test directory
- Use assemblies to setup references
- Create an edit mode test script
- Create a play mode test script
    - Ways to determine test success and failure

What Is Integration Testing?

Combining the **Unit Test Modules** into a interface to verify all modules work together correctly



3 Important Parts of Integration Testing

- Verifies all modules work together
- Reveals underlying errors in the interfacing
- Verifies any newly added components aren’t affected

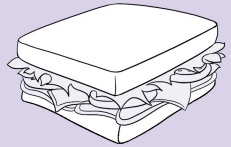
The Four Approaches

Top-Down

Bottom-Up

Big-Bang

Sandwich



Top - Down Integration Test

Testing starts from a high level module to Low Level

- Pros
  - Very consistent
  - Takes less time
  - Location of fault is easier
  - Major flaw detection
- Cons
  - May require several stubs
  - Early release support is not good
  - Testing for basic functionally occur late

Bottom - Up Integration Test

Testing starts from a low level module to high Level

- Pros
  - Application efficient
  - Less time required
  - Easier test conditions for the creation process
- Cons
  - Necessary to have several drivers
  - Testing for data flow is late
  - Early release is poorly supported
  - Detection late of key interface defects

Big - Bang Integration Test

All modules are tested as a whole

- Pros
  - Everything tested at once
  - Easy for small systems
  - Testing time is saved
- Cons
  - Takes a while to set up
  - The root causes of failures is harder to track
  - Chances of interface links missing
  - Important modules aren’t prioritized

Sandwich Integration Approach

Make use of both Top-Down and Bottom-Up testing with a middle layer as the target

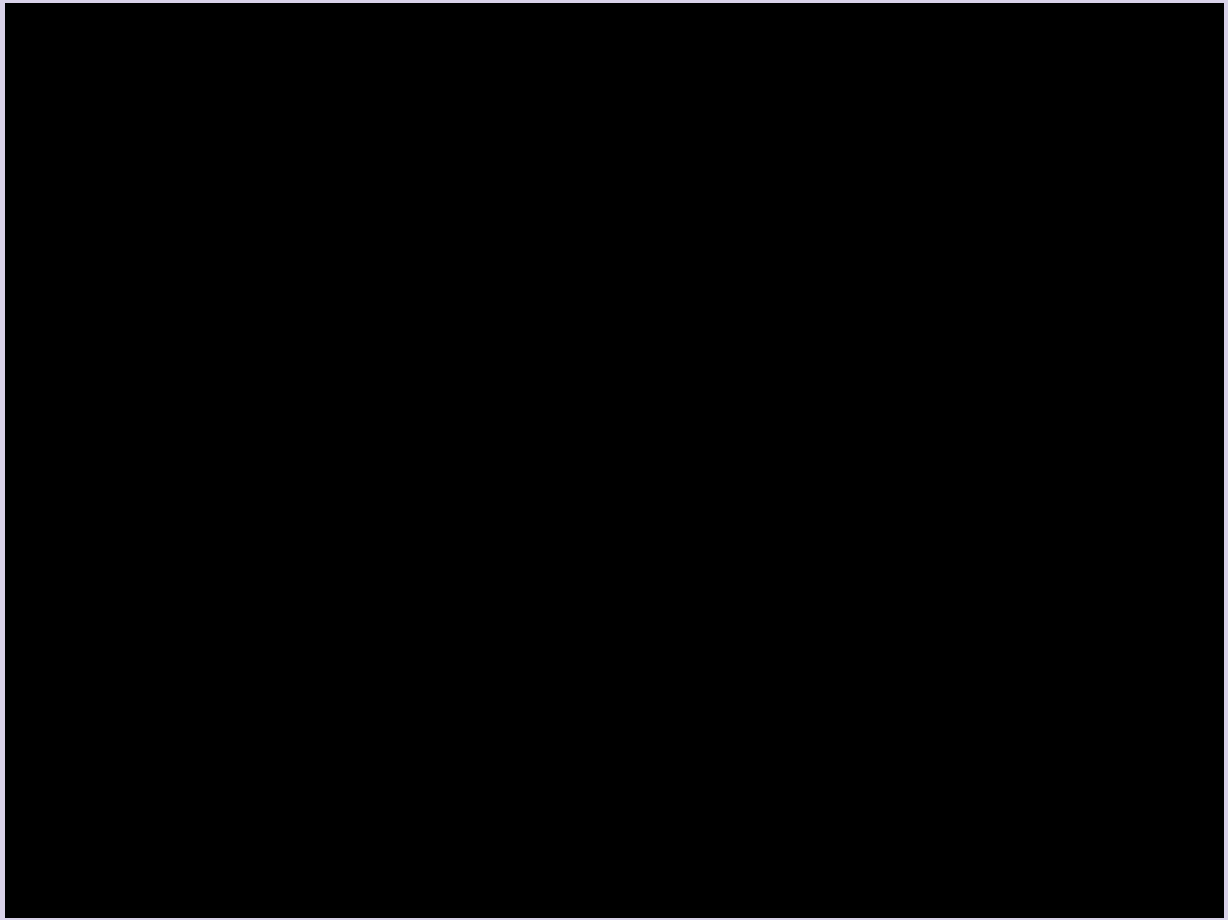
- Pros
  - Layers can be tested simultaneously
- Cons
  - It's spendy
  - Requires higher skill level
  - Not as extensive testing

What makes a good Integration Test

The idea behind Integration Testing is to combine parts of the application and test as a group to see that they work together.

In doing this, you expose interaction faults between components and improve the program as a whole.

Making an integration test



What is a Stress Test

The goal of a stress test is to find the limit at which the program breaks and how it handles crash errors. It also gives valuable information about how the program works when the situation is less than ideal or unexpected.

1. Plan what element you want to test
    a. Where do you think your biggest bottleneck is?
2. Write a script to test it
3. Run the script and get your results
4. Did it meet your expectations? If not, optimize and re-run the test

How to make a stress test