Lesson 7

Unit testing

Nicolas Serf
serf.nicolas@gmail.com

Unit testing

Unit testing

Why

Unit testing

ENcapsulation

Reliability

Evolution

Budget

Definition

Encapsulation is an important concept in **development**, and it is even **more true** when it comes to **unit testing** because it is kind of **mandatory**.

Isolate

Indeed, the idea of a **unit test** is to test a **single functionality**, without any dependency on something else. Thing of an **addition** for a math library for example.

API

When developing, it is always good to have an **API** idea over a **feature**. WHich mean you develop some **internal functionalities**, and some **externals**, that will be available to be **used**… and **tested**.

Unit testing

Reliability

Encapsulation

Evolution

Budget

Definition

Reliability is something you want when it comes to a software. And that **reliability** is achieve by **ensuring** you can **track down** as **much bugs** as possible, in a reliable way.

Scenario

A **unit test** is a piece of **software** that run the **exact same logic** you've written. This is typically why you have **reliability**, because the scenario you write will **not change** even if you change the **feature**.

Error-prone

If you ask a **human** to ensure that everything is **working**, it is **error-prone**, he could **miss** a use case, miss the test, etc... That's the **reliability** you have when it comes to automatic unit testing.

Unit testing

Evolution

Encapsulation

Reliability

Budget

Definition

Unit test allows you to **develop** and **evolve** a feature without being **worried** about **breaking** something. That part is ensure by the tests that will run after you've done with the **modifications**

API

That's also why **API** principe is **important**. If you have a **clear view** about what a feature needs to **offers**, you are then free to **modify** as much as you want inside the feature, because the **API** will **remain** the **same**

Unit testing

Budget

Encapsulation

Reliability

Evolution

Definition

Maintaining, **debugging**, **evolving**, is actually **80%** of the job when you **develop** a feature. If you **propagate** that idea to the budget of a game, you realize how important the **process** is.

SHort run

On the **short run**, you'll be **spending** more time to **develops** the **unit tests**, the **encapsulation** and **independence** of your feature. You may think that you're too **slow** compared to someone **not doing** that

Long run

If you look on the **long run**, you'll be speeding way more resources (**human**, **money**) on a feature **poorly developed** without unit testing, without a clear API definition, because of the **maintenance**.

Unit testing

Who

Unit testing

SOurce code

Writer

Tester

Definition

In essence, a **unit test** is code written by someone that has **access** to the **source code**. It would not be impossible to have that in a **visual scripting language**, but that's not the way it is does **usually**

Written code

As stated above, **writing unit testing** is basically **writing code**, most likely in the **language** you've been **developing** your game. That's why you also need **knowledge** about general **programming**

Module

We've spoke about **modules** a bit; **unit testing** are most likely inside a specific **modules** or even a **container**. It allows to **not ship** the tests on **builds**, not have **dependencies** and clear **distinctions**

Unit testing

Writer

Source code

Tester

Definition

The person which is **writing** the **unit testing** code must be **aware** of the **constraints**, the **language**, the **features**, etc... It is basically the owner and is responsible for ensure that tests are correct

Designers

Designers are kind of **link** to **unit test**. They will **not write** them, but they'll give from the **GDD**, what they **want** from a **feature**. That's the **entry point** on which **programmers** will be **writing** the **tests**.

Programmers

Not much to say about that, what all we've seen already, you understand that's a **programmer's job** to **write** code for **unit testing**, or even a **QA engineers** in some cases.

Unit testing

Tester

Source code

Writer

Definition

On the other hand, **tester**, or more precisely, the person in charge to **running** the **test** will depends on the **scenario**.

QA

There is also some **tests** that needs to happens in a **gameplay scenario**. In that case, **QA** can follow the **rules** from the test, and ensure that everything is **working fine**.

Automatic

In most cases, **unit tests** are **statics**, which means they **don't require** to **play** the game. As a dev, you'll most likely **trigger** this **test**, or you could even have a **dedicated pipeline** on the **CICD** to have that.

Unit testing

When

Unit testing

Pre-production

Production

Development

Definition

Pre-production is a specific **phase** where all **ideas** are **tested**, and a lot of things changes based on **feedbacks given** by **designer** when they try a feature.

Prototypes

This is the **phase** where all **features** are in **prototyping** phase, it means you do a **quick** and **dirty implementation** in order for them to test.

Implication

During this **phase**, you **SHOULD NOT** write any **unit test**, because you are not even sure that the feature **will stay** in the **same** way, or even **stay at all**

Unit testing

Production

Pre-production

Development

Definition

When you are in **production**, you could either be in **maintenance** or development, but the same logic applies.

No excuses

This is the time where **unit** test **benefits** will kicks in… or where you'll **regrets** not having done anything about **testing** and **debugging** because each modification may break something.

Worth in the long run

This is why **unit testing** is **worth** in the **long run**, because you'll be able to **modify** as much things as you want **implementation** wise, test will ensure that your **public API** is still working as intended.

Unit testing

Development

Pre-production

Production

Definition

Development refers to the time where you'll be **developing** a **new feature**. This is the moment where you **should write unit tests** if the system **allows** it. It is part of the **development process**

As much as possible

Not every features offers a possibility to be **unit tested**. That's also why unit testing is way less used in **video games**. But there is still a lot of things to be **tested** if you properly **decouple** things

Unit testing

How

Unit testing

Basis

Cycle

Type

Editor

Runtime

Definition

Writing unit tests is quite **straightforward** when you have a **clear idea** about what is the **public API** of your **feature**. About running the test, it will also be quite simple because most engine provide **tools** for it

CICD

CICD stands for **continuous integration** and **continuous delivery**. It is a **pipeline** to be **built outside** the **game**, but which will be used by **programmers** in order to **fasten** the **production process**.

Automatic

When you have a proper **CICD pipeline**, **running tests** becomes **automatically**, and could also make **builds fails** etc...

Unit testing

Cycle

Basis

Type

Editor

Runtime

Definition

When you want to **approach unit test** as **central** to your development process, you dive into a **TDD approach**. It stands for **Test Driven development**. It is **not fully possible** in video games though

Write test first

For **features** where you can do **unit tests**, always **starts** by **writing** the **tests**. In that situation, you'll not **over-engineer** the **process**, you'll do **implementation** that **answers** the tests and nothing more.

Unit testing

Type

Basis

Cycle

Editor

Runtime

Whitebox

Also referred to as **glass box** or **transparent testing**. The **tester** is aware of the **application's internal functionality** and can test it against the **design** and **requirements**

Blackbox

In this type of **unit testing**, **testers** validate the software **application's user interface**, along with its **input** and **output**.

Greybox

It is a blend of **whitebox** and **black-box testing** In this type of testing, the **testers** are not **completely aware** of the **application internals**, **functionality**, and **design requirements**.

Unit testing

Editor

Basis

Cycle

Type

Runtime

Definition

When it comes to **game development**, **editor testing** are **static tests**, that do **not require** a **context** or **playing**. It is a list of tests that needs to pass in order for the feature to be **considered valid**

Manually

If you **don't have CICD**, or that you want to **ensure** that the **tests pass before commiting**, you'll be able to **manually run** the tests from the editor

Implication

Editor tests are useful for **encapsulated feature**, that have **no dependencies** and **don't require** other **system** to be tested. They are also most likely **easier** to **write**.

Unit testing

Runtime

Basis

Cycle

Type

Editor

Definition

On the other hand, **runtime tests** will need a **context**, they'll be **trigger** by **tester** that are playing the game. They are more **complex** to **write** and will implies **multiples systems** more likely

React

Basically, the **tests** will **not be runned** in **editor**, but during **playtime**. The **testers** will ensure to **enable** some **events**, and the tests will be **triggered** at that **time** and **runned**.

Unit testing

Limitations

Unit testing

UI

Detection

Time

Definition

Testing UIs is far from being **straightforward**. The reason is quite simple, you **cannot think** about **everything**, and **UI** is **directed** to **human**, so a **human** needs to tells if **everything** is fine.

Localization

Localization is also **problematic** because localization is mostly done **off-shore** by **companies internal** to the **country** you want the **localization** on. They ensure that text are **readable**, **no error**, etc...

Resolution

Another topic that cannot be tested by code is the **resolution appearance** of a **game**. Only a **human** can tell if the **UI** is **looking good** on a **specific resolution**, if some **texts** are not **readable**, etc...

Unit testing

Detection

UI

Time

Definition

In reality, **detecting every bug** is not that **simple**, you'll always **forgot** a specific case. Moreover, in **game development**, players like to **break** game.

System-wide / Integration

Static testing is not the **vast majority** of tests needed in **video games**. There is a **player** behind the **controller**, and the **majority** of **tests** are actually **integration** or **systems**.

Unit testing

Time

UI

Detection

Definition

Even if **proper unit testing** can save a lot of times in the long run, it is time consuming at first to first the code for testing.

Video game

In **every industry**, it is quite hard to **explains** to a **manager** that you are taking **more time** because you are thinking the **project** in the **long run**, **depending** on **management** etc… but it is even harder in **video game** industry when **visuals** are everything and **results** are needed to show for **investors** etc…

Unity

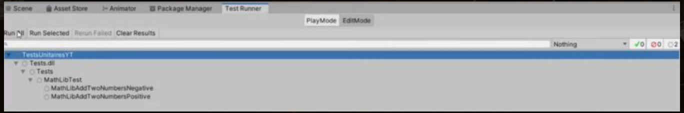
Unit testing

[Test] - Nunit - Useful for static tests
[UnityTest] - Unity Test avec IEnumerator - Useful to skip frames, wait time, etc...

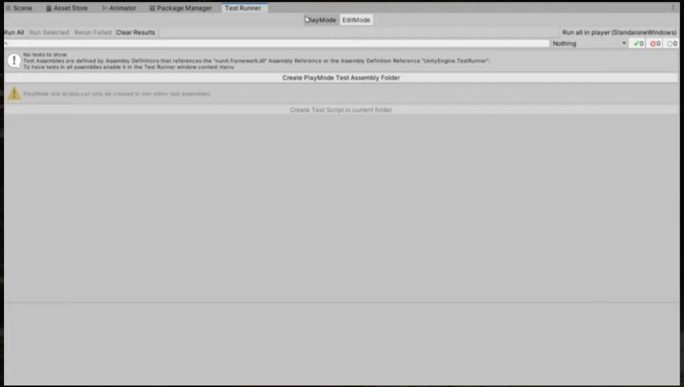
[TestCase(X, Y)] - Can be added above the function multiple times
[NUnit.Framework.Range(X, Y)] - Can be added in front of attributs

[OneTimeSetUp] - Called 1 time before all test begins
[SetUp] - Called 1 time before a test begins
[OneTimeTearDown] - Called 1 time after all tests are been executed
[TearDown] - Called 1 time after a test

Assert.XXX
Assert.AreEqual(X, Y)



```
▼ TestsUnitairesYT
  ▼ ○ Tests.dll
    ▼ ○ Tests
      ▼ ○ MathLibTest
          ○ MathLibAddTwoNumbersNegative
          ○ MathLibAddTwoNumbersPositive
```



No tests to show
Test Assemblies are defined by Assembly Definitions that references the "nunit.framework.dll" Assembly Reference or the Assembly Definition Reference "UnityEngine.TestRunner".
To have tests in all assemblies enable it in the Test Runner window context menu

Create PlayMode Test Assembly Folder

PlayMode test scripts can only be created in non editor test assemblies.

Create Test Script in current folder

Live demonstration

? Questions ?