10 Things I Hate About U^{nity}

Grievances regarding our favorite game engine

Why Unity?

Before we complain about Unity, I want to emphasize a subset of the many reasons why we do in fact like the engine:

- it's $free.99
  - unless you make lots of money, in which case you get a paid license
- it's remarkably cross-platform (like 25+): iOS/Android/Windows, Windows/OSX/Linux, PS4/XBone/Switch, WebGL, 3DS, etc.
- actually user-friendly GUI development of games
- easy, sane scripting (C# or, ugh, JS; also, Boo???)

The pros here mostly point to the fact that it's easy for a beginner to get started, and that indie studios can roll things out without having to deal with fancier stuff.

Why, Unity?

On the other hand, Unity has some, um, frustrating characteristics.

Chances are extremely high that you'll encounter one (or two, or three…) of these as you work on your projects.

Set jimmies to maximum over rustle.

- Dr. Robert Zubek, Lead Programmer, SomaSim, LLC.

Public Variable Serialization

If you have a public field on a `MonoBehaviour` which you modify in-editor, future changes to that field's default value *won't manifest* in previously instantiated copies of that component.

- Unity serializes out the fields of the `GameObject` in question and deserializes them whenever it needs to repopulate those fields (e.g., whenever you stop playing a scene)
- So, new default values in the public fields of the `MonoBehaviour` will not be seen, since they *aren't the serialized values.*

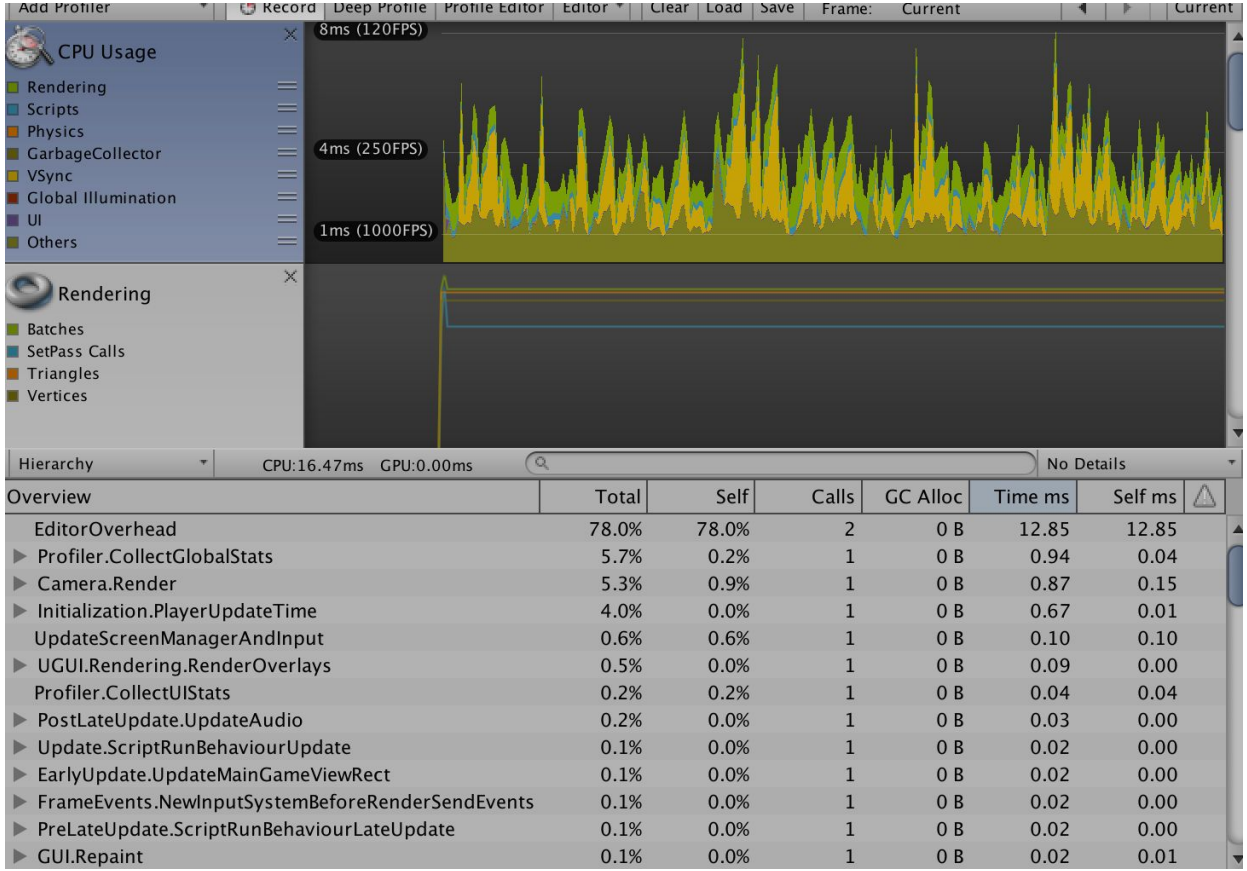
Frustration Level: 5

Profiling

It’s alright, I suppose.

Unless you need to go deep into the bowels of a particular call, in which case you’ll spend minutes waiting for the UI to update.

Frustration Level: 4



Collaboration

You should be using a Version Control System.
This sounds like a suggestion. It’s really not.

However, there are some difficulties when Unity meets VCS:

- Version control is generally not designed to work with binary files (images, executables, etc.)
- Lots of Unity’s generated files are binary, so you’ll get merge conflicts if two people touch basically anything (scenes, prefabs, etc.)

Frustration Level: 4 - 7

(Partial) Solution: Learn You a Git for Great Good

```
# making a repo
git init
git clone $url
git remote add origin $url     # set up a remote for your repo
#remote stuff
git push -u origin $branchname # push a branch to origin that does not already exist
git push                       # push tracked changes to origin
git pull                       # pull changes from origin to a given local branch
# branches
git checkout -b $branchname    # make a new branch
git checkout $branchname       # go to an already made branch
git merge $branchname          # merge a given branch's changes into your current branch
# general stuff
git add $filename              # add file of name $filename
git add .                      # add all of . (the current directory)
git checkout $filename         # untrack a previously added file
git commit -m "Your Message"   # commit something with a given message
# changing history
git reset --hard $commit       # go back to $commit, clobbering everything (be careful with this!)
git revert $commit             # reverts $commit with a new commit that undoes all of the changes from $commit
```

(Partial) Solution: Project Organization

First of all: *Talk to each other.*

Habits:

- *Never push directly to master.* Thank me later.
- Make your commit messages useful.
  - “Fixed <type of bug>” is fine (and a lot better than “fuck”)
- Merge other people’s stuff into your branch first, *then* push changes.

Practical Tips:

- Make Different Scenes for Different People
  - Name them things like “dev-ethan” and “dev-rob” and “demo” so that it’s *clear* what’s what
- Use prefabs! It’s nice to have a “canonical” example.

Prefabs

As a consequence of broken serialization:

- Consider a prefab ***ExamplePrefab***
- Drop an instance of ***ExamplePrefab*** into the scene as the child of some `GameObject`
- Save this new `GameObject` as a new prefab
- Changes to ***ExamplePrefab*** will no longer affect the copy of ***ExamplePrefab*** that we dropped into the scene earlier.

Frustration Level: 4 - 6, depending on how you feel that prefabs ought to work

The Manual

- Sparsely and unevenly populated: here's hoping you're not looking for something obscure, 'cause Lord knows if you'll find anything useful on it
- Things like: "Legacy Documentation: Version **5.6** (Go to current version)"
  - clicking on "current version" takes you to the current version of *the manual*, not of the page you were looking at 🙄
- Shit like this ➔

Grid.cellLayout

public GridLayout.CellLayout **cellLayout**;

Description

The layout of the cells in the Grid.

Frustration Level: 6

Fucking .meta Files

This is not a contentful slide, I just hate .meta files.

Frustration Level: ???

Serialization (Redux)

- “The serializer does not support null.”
  - Seriously?
- “If it serializes an object and a field is null, we just instantiate a new object of that type and serialize that.”
  - Wtf?
- “Obviously this could lead to infinite cycles, so we have a relatively magical depth limit of 7 levels.”
  - what. the. fuck.
- “**No support for polymorphism**”
  - see above

Frustration Level: 8

Non-Deterministic Breakpoint Failures in Mono

Before *Rider* and *Visual Studios for Mac* (both of which are quite recent), there was really only one option for OSX folks: *MonoDevelop*.

MonoDevelop is a flaming pile of shit.

Among other things, it occasionally has non-deterministic breakpoint failures. Meaning that the breakpoints that you set while debugging *don't do anything, but you can't predict when this is going to happen.*

Frustration Level: 10

Physics

Unity physics is... frustrating.

In particular, if you're working on a platformer or other collision-heavy game, you might have noticed that your `GameObjects` have a tendency towards:

- lethargy and floatiness (things like forces)
- snappiness and rigidity (things like `Rigidbody2D.MovePosition`)
- incorrectness and frustration (things like `Transform.position`)

It's often difficult to figure out exactly which pieces of Unity physics you need to use - and how you need to tweak them - to get the results that you want.
Frustration Level: 6

(Partial) Solution: P-Controllers

For movement at least, there is a broad and often overlooked solution: P-Controllers.

Proportional controllers are a dirt-cheap and dead-simple approach to feedback control. The shtick is that they apply a correction proportional to the difference between the current value and the desired value. As a concrete example:

```
var RB = GetComponent<Rigidbody>();
var force = Acceleration * (movement - RB.velocity);
RB.AddForce(force);
```

That’s pretty much all there is to it.