**7 Ways to Keep Your Unity Project Organized**

I saw a person on Quora the other day, asking how programmers are able to write projects that consist of over 10,000 lines of code. When software gets bigger, it is more difficult to maintain and that’s a fact. So here’s the thing – if you don’t keep your project organized, you’re going to have a hard time to keep the pace. Later on, you will find yourself wasting time with a messy project instead of adding new features. This is also true regarding any Unity Project. Here are (in my opinion) the most important tips that will help you with keeping your project organized.

**1. Directory Structure**

We cannot talk about organization without mentioning organizing project directory structure. Unity gives you a total freedom in that matter, but because of that, it can frequently get really messy. This is the directory structure I personally use:

* 3rd-Party
* Animations
* Audio
  + Music
  + SFX
* Materials
* Models
* Plugins
* Prefabs
* Resources
* Textures
* Sandbox
* Scenes
  + Levels
  + Other
* Scripts
  + Editor
* Shaders

1. **Do not store any asset files in the root directory**. Use subdirectories whenever possible.
2. **Do not create any additional directories in the root directory**, unless you really need to.
3. **Be consistent with naming**. If you decide to use camel case for directory names and low letters for assets, stick to that convention.
4. **Don’t try to move context-specific assets to the general directories.** For instance, if there are materials generated from the model, don’t move them to Materials directory because later you won’t know where these come from.
5. Use 3rd-Party to store assets imported from the Asset Store. They usually have their own structure that shouldn’t be altered.
6. Use *Sandbox* directory for any experiments you’re not entirely sure about. While working on this kind of things, the last thing that you want to care about is a proper organization. Do what you want, then remove it or organize when you’re certain that you want to include it in your project. When you’re working on a project with other people, create your personal Sandbox subdirectory like: *Sandbox/JohnyC*.

**2. Scene hierarchy structure**

Next to the project’s hierarchy there’s also scene hierarchy. As before, I will present you a template. You can adjust it to your needs.

* Management
* GUI
* Cameras
* Lights
* World
  + Terrain
  + Props
* \_Dynamic

There are several rules you should follow:

1. All empty objects should be located at 0,0,0 with default rotation and scale.
2. When you’re instantiating an object in runtime, make sure to put it in \_Dynamic – do not pollute the root of your hierarchy or you will find it difficult to navigate through it.
3. For empty objects that are only containers for scripts, use “@” as prefix – e.g. @Cheats

**3. Use prefabs for everything**

Prefabs in Unity are not perfect, but they are the best thing you will find to share pre-configured hierarchies of objects. Generally speaking, **try to prefab everything that you put on your scenes**. You should be able to create a new level from an empty scene just by adding one or more prefabs to it.

The reason why you should use prefabs is that when a prefab changes, all the instances change too. Have 100 levels and want to add a camera effect on all of them? Not a problem! If your camera is a prefab, just add a camera effect to the camera prefab!

Be aware that **you cannot have a prefab in another prefab**. Use links instead – have a field that requires a prefab to be assigned and make sure to assign it when instance is created. Consider auto-connecting prefab instances in *Awake()* or *OnEnable()* when it makes sense.

**4. Learn how to use version control system (VCS)**

You may already know something about GIT, Subversion or any other VCS out there. As a matter of fact, “knowing something” is only a small piece of what you may learn. You should focus on learning about important but infrequently used features of VCS of your choice. Why? Mostly because VCS systems are much more powerful that you think, and unfortunately many users are using these as nothing more than a backup and synchronized solutions. For example, did you know that GIT allows you to stash your changes, so you can work on them later without committing anything to your master branch?

Programmers tend to comment out blocks of code in case it’s needed later. Don’t do that! If you’re using VCS learn how to quickly browse previous versions of a file. When you are familiar with it, your code looks a lot nicer without unnecessary block of commented code.

Here’s a nice resource of tips for GIT users: <http://gitready.com/>

**5. Learn to write editor scripts**

Unity is a great game engine in the matter of extensibility (see Asset Store). Learn how to write editor scripts and utilize this knowledge. You don’t necessary need to create fancy GUI for your scripts, it can be something simple, as menu entries that are doing something useful. Here are some examples of editor scripts that I have created not so long ago:

* Google Sheets .csv download – I had a translation spreadsheet saved on Google Drive. It automatically downloaded the newest version as .csv file, so I never had to do it manually.
* Randomize the position, rotation and size of trees – I had a lot of trees and wanted it to look more like a forest than a grid.
* Create distribution – Built for specified target, zips all the files and copy to the right place.
* String replace in the sources – I had several files that contained the application version.

You can learn how to create editor scripts from the [official documentation](http://docs.unity3d.com/Manual/ExtendingTheEditor.html).

**6. Learn to program defensively**

Have you heard about [defensive programming](https://en.wikipedia.org/wiki/Defensive_programming)? Wikipedia defines it as follows:

**Defensive programming** is a form of [defensive design](https://en.wikipedia.org/wiki/Defensive_design) intended to ensure the continuing function of a piece of [software](https://en.wikipedia.org/wiki/Software) under unforeseen circumstances. Defensive programming techniques are used especially when a piece of software could be misused.

Generally when you’re writing MonoBehaviours, you should make sure that:

* All needed references are set
* All required components are present
* If you’re using singletons, make sure that they exists
* If you’re searching for objects and expect to find something, do it as fast as possible
* Mix-in editor code ([ExecuteInEditMode](https://docs.unity3d.com/ScriptReference/ExecuteInEditMode.html) and [#if UNITY\_EDITOR](http://docs.unity3d.com/Manual/PlatformDependentCompilation.html)) to do as many checks as possible before you run the scene

For many of these checks you can use [asserts](http://blog.theknightsofunity.com/unity-5-1-assertion-library/). You should also read A Story of [NullPointerException Part 1](http://blog.theknightsofunity.com/story-nullreferenceexception-part-12/) and [2](http://blog.theknightsofunity.com/story-nullreferenceexception-part-22/).

**7. Implement in-editor and/or in-game cheats**

After you learn how to write editor script, you should be able to write a set of in-editor cheats. It can work as menu entry that unlocks something (all levels for instance). It’s really easy to create:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class Cheats  {      [MenuItem("My Game/Cheats/Unlock All Levels")]      public static void UnlockAllLevels()      {          if (Application.isPlaying)          {              // unlock code here...          } else {              Debug.LogError("Not in play mode.");          }      }  } |

Generally you should write cheats that will allow you to:

* Unlock all levels, characters, items etc
* Give you immortality
* Add/subtract values like time, money, coins etc
* Allow you to see things not meant to be seen by players
* Anything else that will help you with testing your game

Of course more practical (but harder to write) are in-game cheats. These type of cheats can be executed outside Unity editor, but you have to think how you would like to execute it. See our other article about [implementing cheats subsystem](http://blog.theknightsofunity.com/implement-game-cheats-subsystem-within-unity/) controlled by mouse.