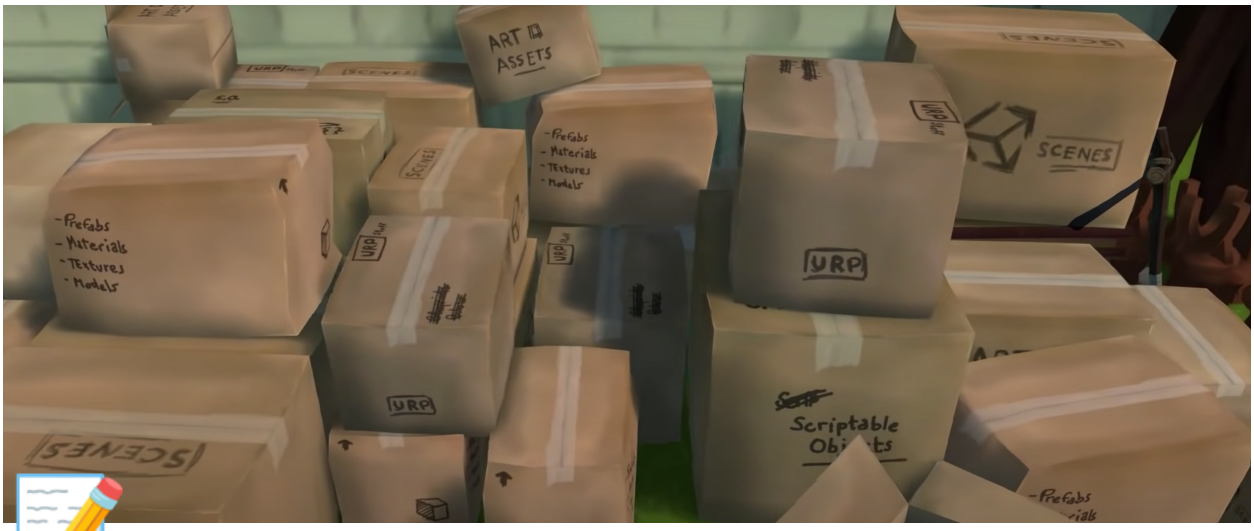

Unity Guideline

In this guideline every aspect related to nomenclature, assets/folder organization and programming dogmas will be explained.

These are guidelines for keeping your project organized and allow your team to quickly find the assets they need. Games are large projects that span several months, thus having standardized conventions that make sense will avoid headaches in the long run.

This is just a guide for organization and nomenclature, every programmer should do whatever he/she thinks is a proper solution for every problem applying ***S.O.L.I.D***. principles.

🗂️ Folders Structure

Folders can get really messy as the project gets bigger. To avoid this issue we will present a simple, yet effective way to organize and name asset folders to easily access to any piece of asset that the programmer or the designer needs.

```
ASSETS

+---Art
|   +---Audio
    |   +---Music
    |   +---Sounds            # Samples and sound effects

|   +---Graphics
    |   +---Animations        # Animations and animations controllers
    |   +---Materials
    |   |   +---PhysicsMat    # Physics Materials
    |   +---Models            # FBX, Blend
    |   |   +---Static Mesh   # Boneless static meshes
    |   |   +---Skinned Mesh  # Skinned bone meshes
    |   +---Textures          # PNG, JPG... images
    |   +---Sprites           # 2D Sprites
    |   |   +---Icons         # Game icons
    |   +---Fonts             # Fonts (tff...)
    |   *---Placeholders      # Placeholder Assets


+---Data                  # Game user data (ex: Scriptable object instances)
|   +---Channels          # Communication Channels
|   +---Localization      # Localization files (xml, csv...)

+---Scripts
|   +---Managers          # Generic abstract managers (singletons)
|   +---Interfaces        # Interfaces scripts
|   +---Enums             # Isolated enum scripts
|   +---ScriptableObjects # Scriptable Objects
|   +---Shaders           # Shader files and shader graphs
|   +---Presets           # Scrips presets
|   +---Utils             # Generic and common with all projects (Singleton.cs)
|   *---Editor            # Editor scripts

*---Plugins               # Every related assets of externals plugins

+---Prefabs
|   +---UI                # UI related prefabs (canvas, huds...)

+---Scenes
|   +---Test Scenes       # Scenes used as testing
```

```
+---Settings               # Settings according to rendering and pipelines
|   /---AudioConfig        # Audio Managers...

*---Resources              # Resources special folder

+---LabCave                # Settings and assets used by LabCave packages


================================================================================

"+": Folders to be added by the programmer
"/": Optional, 3rd parties folders, can be multiple
"*": Special folder: Unity will treat them differently when built.
```

🖌️ Assets Nomenclature

Prefixes_

Every assets should have a proper prefix that makes the Unity Inspector searching as easy as possible.

- Animations: **A**_*AnimationName*
- Animation Controller (Animator): **AC**_*AnimatorName*

- Materials: **M**_*MaterialName*
- Physics Materials: **PM**_*PhysicsMaterialName*

- Static Meshes: **SM**_*StaticMeshName*
- Skinned Meshes: **SK**_*SkinnedMeshName*

- Textures: **T**_*TextureName*
- Sprites: **SP**_*SpriteName*
- SpriteAtlas: **SA**_*SpriteAtlasName*

- Audio: **AU**_*AudioName*

- Interfaces: **I***interfaceName (without underscore)*
- Scriptable Objects: **SO**_*ScriptableObjectName*
- Editor Scripts: **CE**_*CustomEditorName (without underscore)*
- Enums: **E***EnumName*
- Presets: **PS**_*PresetName*
- Shaders: **SH**_*ShaderName*
- ShaderGraphs: **SG**_*ShaderGraphName*

- Prefabs: **PF**_*PrefabName*
- Scenes: **SC**_*SceneName*

_Suffixes

Suffixes specified an asset attribute that makes it different from others.

Texture Suffixes

- Albedo: _**AL**

- Specular: _**SP**
- Roughness: _**R**
- Metallic: _**MT**
- Glossiness: _**GL**
- Normal: _**N**
- Height: _**H**
- Displacement: _**DP**
- Emission: _**EM**
- Ambient Occlusion: _**AO**
- Mask: _**M**

Debug Assets Naming

Every asset that is temporary, for debug purposes or placeholder should be named between brackets:

[DebugAssetName]

📀 Scripts Style Guide

This guide will develop the fundamentals of naming and conventions of the C# programming language oriented to Unity. As the first fundamental is:

TAKE YOUR TIME NAMING YOUR ASSETS, SCRIPTS, FUNCTIONS AND VARIABLES!!

Names are the most important part of every element in programming as they must show the functionality and the use of that programming part. Do not hesitate in taking your time and use as much letters as you need.

PascalCase

Pascal Case is an upper case naming format for several parts of the program.

It consists in separating the names by uppercase beginning with an upper case too.

Ex: ***MySuperClassName***

camelCase

Camel case is similar to the pascal case but the word starts with a low case.

Ex: ***thisIsMyVariableName***

lowercase

This is a case that uses lower case letters for the full name. (***Really unreadable***)

Ex: ***thisismyvariablename***

Snake_Case

Snake case starts with either lower or upper case but the parts are separated by underscore.

This type of case will not be used as much as the previous ones

Ex: ***this_Is_A_Script, My_Object_Name***

Classes

Classes will mostly be *Public* and will use the PascalCase as well as the script name that holds the classes. The names should have a substantive convention

Ex: ***Dog.cs, MainPlayer.cs, FlyingEnemy.cs***

```
public class ObjectPoolController : MonoBehaviour{}
```

Functions

Functions should have a verb name convention and have to be written in PascalCase. At least *public* functions should have a summary indicating the functionality, input arguments definition and return value definition.

The name of the functions should always be as most descriptive as possible to avoid over commenting code.

Ex:

```
/// <summary>
/// Fires the weapon
/// </summary>
/// <param name="weaponName">Name of the weapon</param>
/// <returns>Returns true if the weapon hits</returns>
public bool Fire(string weaponName)
{
  ....
    if(projectil.hits == true)
      return true;

    else
      return false;
}
```

Variables

Private variables are the ones that are only visible by the class they are declared into. As other variables they should be named in camelCase.

The default type of scope for a variable is private but we should always indicate it.

Public variables should be in camelCase too, although some programmers use PascalCase. We will be attached to Unity criteria, for example transform.position is a public variable and Unity uses camelCase.

Ex:

```
private int myXPosition;

private void SaveXPosition()
{
  myXPosition = transform.position.x;
}
```

Fields

Fields are variables that define a *set* and *get* functions where you can write any lambda or common type functions in response to writing or reading this variable. They are written in PascalCase. To make a Field from a variable you just have to go the Visual Studio IDE, right click on the variable then click in ***Quick Actions and Refactorings*** and then ***Encapsulate Field: “name of the variable”***

Ex:

```
private int playerLife = 5;

//Field related to a private variable playerLife
public int PlayerLife =
{
  get => playerLife; //Lambda function

  set
  {
```

```
    playerLife -= value; //Value is the numeric value "assigned"

    if(playerLife <= 0)
        Debug.Log("player is dead");
  }
}
```

Attributes

Attributes are useful tools for every experimented programmer, it helps the user or the designer by exposing parameters or variables in the inspector. There is a huge amount of them, but we will discuss the commonly used ones and why we use them frequently.

[SerializeField]

This attribute is used to expose a private variable in the inspector. *Public* variables are exposed in the inspector by default. To avoid that feature use *[HideOnInspector]* attribute.

Ex:

```
[SerializeField]
private int playerLife = 5;

[HideOnInspector]
public string playerName;
```

[Tooltip(”Tooltip Text”)]

This special attribute shows a text in the inspector when you put the cursor on top of the variable where this attribute is located. It is commonly use to specify units for some kind of physic variables.

Ex:

```
[SerializeField]
[Tooltip("In seconds")]
private int fadeDelay = 5;
```

[Header(”Header Text”)]

This one groups of exposed or serialize fields until the program finds another heather. A title is described so these groups are separated by space and the header name. It is really useful for organization when a script scales in exposed variables.

Ex:

```
[Header("Player Variables")]
[SerializeField]
private int playerLife = 5;

[SerializeField]
private int playerMana = 20;

[Header("Weapon Variables")]
[SerializeField]
private int ammunition = 10;

[SerializeField]
private float fireStrength = 20.5f;
```

For further information about C# attributes, check the references below:

- https://github.com/teebarjunk/Unity-Built-In-Attributes
- https://docs.unity3d.com/Manual/Attributes.html

References

- https://github.com/stillwwater/UnityStyleGuide#asset-naming

- https://github.com/justinwasilenko/Unity-Style-Guide
- **Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)**
- **Unity Documentation:** https://docs.unity3d.com/Manual/index.html