Like a Boss - Coding Guidelines

The rules stated below should be followed in ALL code to be written for Like a Boss in order to maintain the project organized and easily debuggable. Please follow these rules STRICTLY, without exceptions.

Code Appearance:

- **Variable Naming:** Prefer descriptive names to short names. Variables should be self explanatory.

- **Naming Convention:**
  - For methods and public properties and variables, use UpperCamelCase.
  - For protected and private variables, use _lowerCamelCase.
  - For local variables, use lowerCamelCase.

- **Variable Declaration:** Declare ALL variables and properties at the TOP of the class. If you are using local variables inside a method, declare ALL local variables at the top of the method, even if they are going to be used at smaller scopes inside the method.

- **If/For/While Indentation:** If it only has ONE line of code, place this line of code in the NEXT line after the condition, without {}. If it has more than one line of code, use {} starting in the NEXT line after the condition, and NEVER in the same line as the condition. Never use statements such as continue; and return; on the same line as the condition.

- **Public vs Private:** If a variable or method is NOT going to be used outside of this class, ALWAYS declare it as private. If you need Unity to serialize this variable, simply add the Attribute [SerializeField] to it, to keep it hidden from other scripts but still accessible through the editor.

- **Namespaces:** Your game should have only one namespace. Namespace is used to avoid conflicts not to create taxonomy.

- **Class Declaration:** Never declare classes inside of classes. Due to Unity's editor, all it does is hide the block of code from the Project window, making it harder to find exactly where that class is really being declared.

General Practices:

- **Class vs Struct:** Needs to be evaluated on a case by case, but ALWAYS prefer using structs whenever possible. Beware of the inner workings of structs, but they are usually much faster to work with considering they are always stored in the stack and don't require a heap access.

- **Events:** Never use standard C# events. Instead, use the CallbackCollection class. Standard events are immutable, meaning that whenever a new object register to that event, it is necessary to create a whole new list of "registered objects" for that event, throwing the old one in the garbage. CallbackCollection uses hashtables to avoid this.

- **Anonym Methods:** To avoid Closures, NEVER use Anonym methods. When they do not have closures, their performance is similar and can even be better than standard methods, however, standard methods will never have Closures, making them much safer to use to make sure no performance is wasted.

- **Foreach:** NEVER use foreach. It has terrible performance and causes allocations in the current version of Mono used by Unity.

- **LINQ:** NEVER use LINQ. it is not fully supported by platforms such as iOS and has terrible performance in the version of Mono Unity uses.

- **Casts:** Avoid whenever possible to use casts. When you need to, it's usually better to use "X as Foo" instead of "(Foo)X".

- **Reflection:** As with LINQ, it is not fully supported by some platforms and also usually has terrible performance. Avoid at all costs.

- **Lazy Loading:** Avoid lazy loading at all costs. While they might seem practical at times, we aim to have 0 GC allocations during runtime, and Lazy Loading, even when not allocating GC, requires runtime memory allocations which are an added cost to running the game. Load all assets and scripts needed at load time, using Awake and Start.

- **Folders:** Avoid creating folders with only one script inside. Try to group them logically and making scripts easier to find. Only exception is the root of the _Scripts folder, which should never contain any scripts in it, only subfolders.

Unity Methods:

- **Awake and Start:** Code written in Awake should NEVER access any other object in the game. All initialization that requires a certain object to access another should be done in the 'Start' method. The reason is that Awake does NOT have a guaranteed call order, and there is no way to make sure that one object is initialized before the other, while Start can be easily configured to order script initialization.

- **GetComponent:** Should almost NEVER be called anywhere in the code except for Awake and Start. It is an extremely heavy method and caching results at Start is the best way to avoid it. Can be used if there is no other option.

- **UnityEngine.Mathf:** Never use Unity Engine's math class. Not only it does not offer the best performance, but it also makes the code harder to port to other engines or backend development.

- **Vector3.Distance:** Never use this method. Square Root calculations are heavy and should be avoided where possible. Instead, use **(Vetor1 -**

Vetor2).sqrMagnitude, comparing it to the power of the distance (distance * distance).

- **Raycasts:** If possible, avoid using Raycasts on performance sensitive code (Update or anything that runs too many times). It is not the most costly operation in this document, but if necessary, try and spread its calls into multiple frames or call it sporadically.

- **Resources.Load:** Always use our custom made wrapper to load and unload resources whenever needed.

Written by Fire Horse Studio