Organization

Folder Structure

- All assets exist in their own folders containing all other assets of those types.
    - Ex: All material assets exist in the Materials folder.
- Any packages should contain all their necessary assets needed to function within a single folder.
    - Ex: TextMeshPro exists solely in the TextMesh Pro folder, there is no other place it should be using assets from.

Asset Naming

- All assets (except scripts & scenes) begin with 3-5 characters that represent what type of asset it is, following with an underscore (_) and finally the asset name.
    - Ex: a sprite for an apple would be named “spr_apple”, a prefab of an enemy would be named “pre_enemy”, a material for stone would be named “mat_stone”

Code Conventions

Naming

- Classes:
    - The names of classes should clearly represent what said class is being used for or what it represents.
    - All class names use CamelCase format
- Methods:
    - The names of methods should be able to explain what said method calculates or returns or does in general. (If a single name won’t be enough to explain what the method does, a comment should be on the line directly above the method to provide a more thorough explanation)
- Variables:
    - The names of variables are highly requested to be fairly verbose. A verbose enough name should be able to explain what the variable represents in just the name.
    - Private static variables begin with a p_ to declare that they should only be useable within strictly that class.

Structure

- All “managers” should inherit the BaseSingleton<T> class for ease of access to them.
- Some systems work with various class structure, such as every “Effect” class has a version of an Activate method that basically serves to update the effect as a new effect, then return itself in this new form. This works with the “effect pooling” system.
- A single script should have one goal (unless it’s a manager) and contain anything that it, and only it, needs to complete said goal (structs, very simple classes/objects **NOT** inherited classes, abstracts/interfaces)

High Level Systems

- **Shot Handling System:** Handles all shots & their raycasts in the game. [Singleton Manager]
- **Effect Pooling System:** Handles all types of effects in the game using pre-instantiated object queues. [Singleton Manager]
- **First-Person Character Controller:** The character controller for any entity that uses a gun & moves around. Uses a byte ID with static methods to get anywhere in code. [IDed Singleton]

Class Topics

Singletons

Used with all managers to allow access to them from anywhere in the code. This greatly cleans up code and allows any script to reference a core system within the game such as UI elements, level components, or general game elements through the game manager. More components will be integrated into these existing singletons and more might also be added.

ScriptableObjects

ScriptableObjects were used to create the guns and hold information & stats within them such as fire rate, shot count, max shot ricochets, mesh, etc. They were very helpful for expandability in adding more guns and also switching between guns during gameplay. They might also be helpful in the future if we implement things like power ups or other things along those lines.