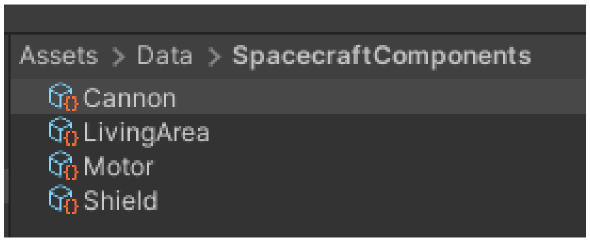
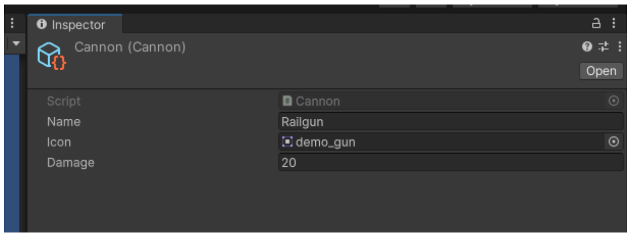
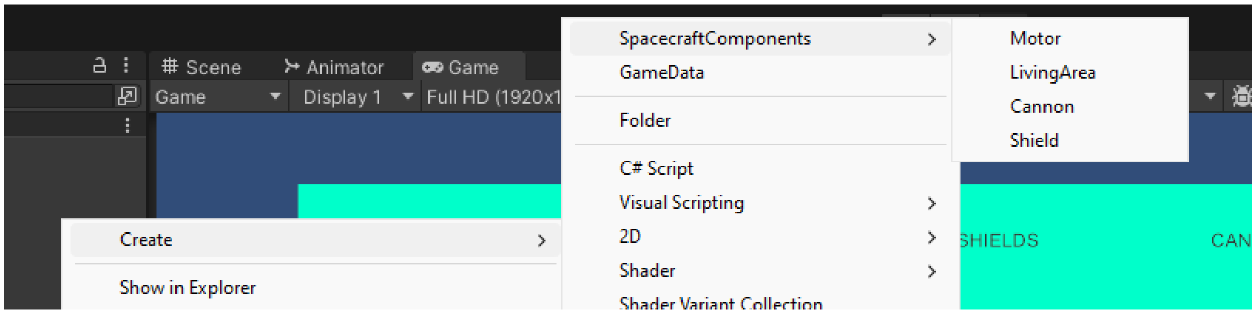
I designed this demo system to be easily modifiable, allowing designers to add new spacecraft components and modify the attributes of existing ones simply by editing Scriptable Objects without changing any code.





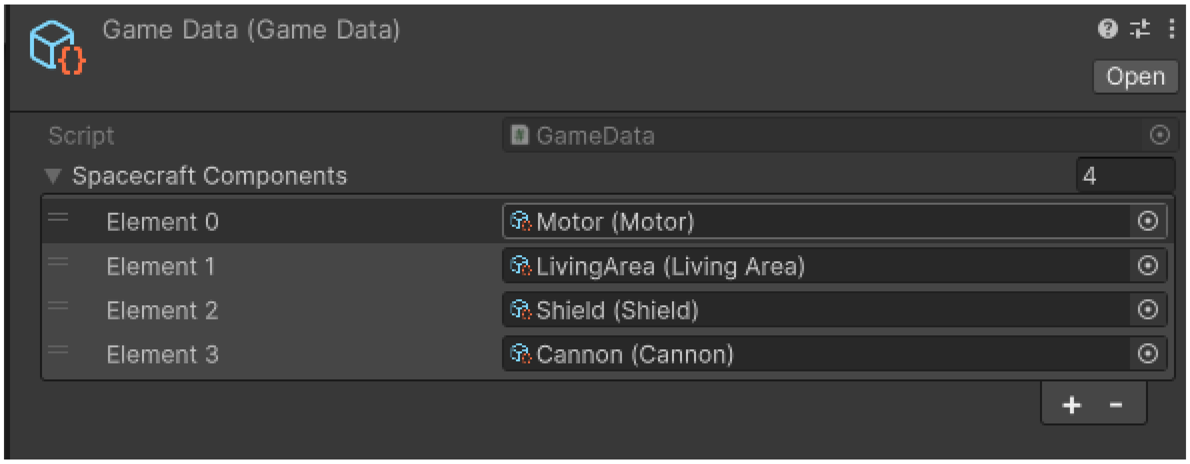
Script: Cannon
Name: Railgun
Icon: demo_gun
Damage: 20

As you can see in the screenshots above, I created 4 demo components, each with its own easily modifiable attributes. Designers can easily create new components from the context menu, as shown in the screenshot below.



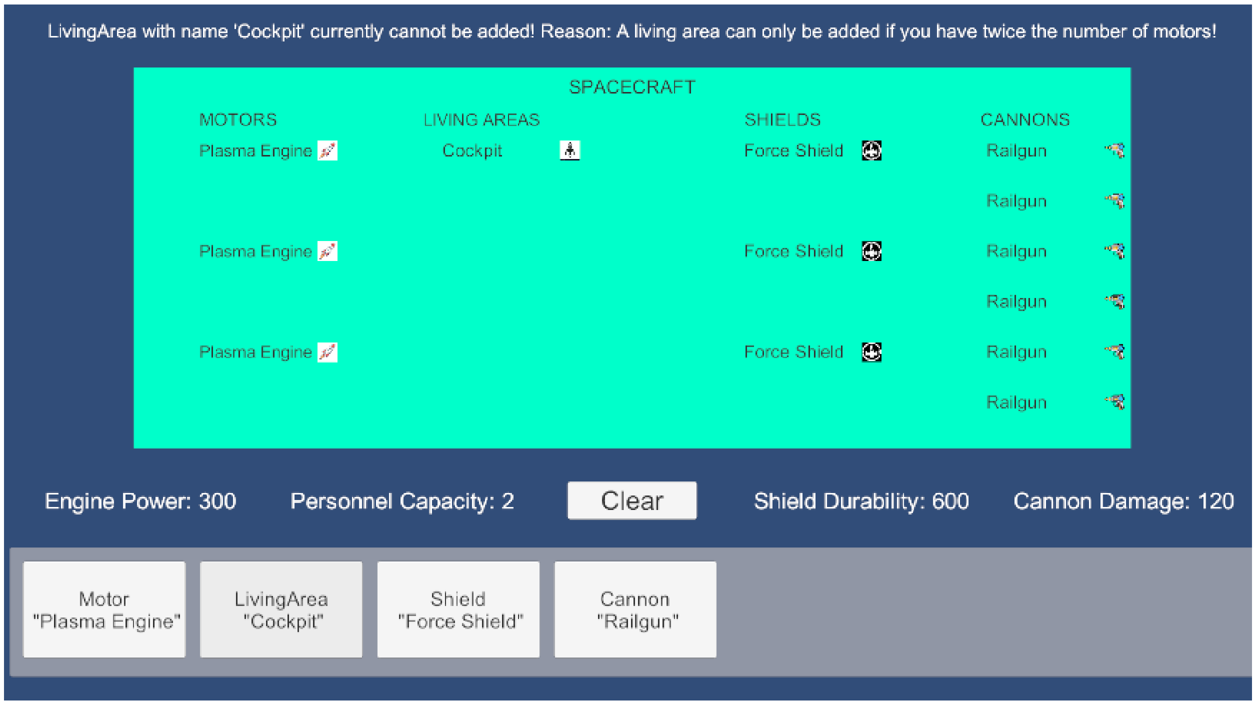
So, if graphic or game designers want to create a new spacecraft component, they need to:

1. Create it through context menu.
2. Edit attributes (name, sprite etc)
3. Add it to “GameData” Scriptable Object (see below)

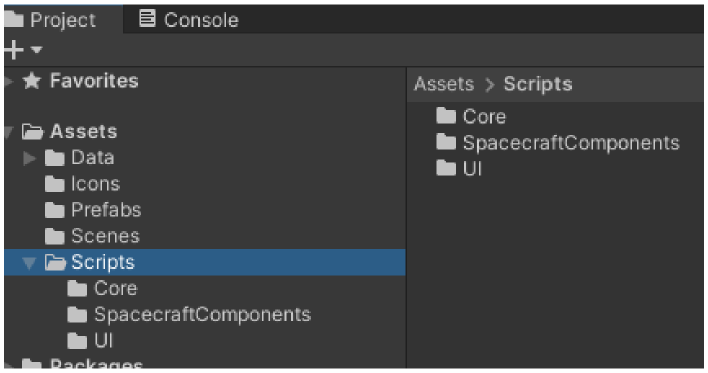


Script: GameData
Spacecraft Components: 4
Element 0: Motor (Motor)
Element 1: LivingArea (Living Area)
Element 2: Shield (Shield)
Element 3: Cannon (Cannon)

All components added to this object are available for use in the game. This allows designers to easily test different configurations and visuals without needing to change any code. I decided to use Scriptable Objects to store data because they are not only easy to create and edit but also compatible with Addressables in Unity. Addressables enhance memory management by allowing dynamic loading and unloading of assets, thereby reducing memory usage. You can also host Addressables on a remote server, which facilitates flexible content updates by enabling remote changes to object attributes without requiring full app releases. Additionally, it reduces the initial app size by downloading only the necessary assets initially.


LivingArea with name 'Cockpit' currently cannot be added! Reason: A living area can only be added if you have twice the number of motors!

Now, let's discuss functionality. As seen in the screenshot above, components added to the 'Game Data' are displayed as buttons. Clicking on these buttons adds the corresponding components to the spacecraft. Additionally, there are overall spacecraft stats that are updated with every added module. Furthermore, there's text at the top of the window that informs whether a component was successfully added or not. In the latter case, it also provides a reason why the component couldn't be added. You can clear the build and start from the ground up by clicking the 'Clear' button.



The code is pretty self-explanatory without complications. We have the GameController, which drives the flow of the game and connects different components. For instance, it controls the UI with the help of ManagerUI and changes the spacecraft state based on events sent from that manager. All components inherit from an abstract class with some abstract methods like 'CanBeAdded'. Thanks to this structure, the spacecraft has only one list of all components it currently has installed. There are also some comments around, including some possible questions—check 'Cannon.cs' for an example.