Game architecture

This is a work in progress, many things can be improved, the UI is just a draft and it should be implemented using TextMesh Pro once that the gameplay is finished.

1.Glossary

Game entity: a physical object that interacts with other entities (Asteroid, Rocket, Spaceship)

Stats: speed, max ammunition, lifetime...

EntityDatas: ScriptableObjects that contain the stats and references to the prefabs needed to create an EntityState. They are similar to the Model in a MVC architecture.

EntityStates: Classes that are created using EntityDatas. They handle their interactions with other EntityStates, they also contain their current stats (that are initialized with the EntityDatas´ stats). They are similar to the Controller in a MVC architecture.

Modules: for complex EntityStates that are not just reacting to other entities, I divided their logic into independent modules (InputModule, AnimationsModule, ThrusterModule...). This approach is similar to a design by composition.

1.Architecture explanation

For the entities and guns, I chose to save their 3D models and VFXs in prefabs. The entities stats and references to these 3D models and VFXs are save in EntityDatas (ScriptableObjects) and their behaviors in EntityStates.

This allows me to have the entities´art separated from their stats values. The heavyweight asset bundles made of prefabs, materials and 3D models are separated from the lightweight asset bundles made just of ScriptableObjects. In this way, I can balance, tweak the game and add more content easily.

Every EntityState is the only responsible for its behavior, they don´t know who has created them or how. This architecture based in States and Datas respects the single responsibility principle creating something similar to a MVC architecture where the logic is separated from the business.

To communicate EntityStates with other systems that are not EntityStates I used Pub/Sub system, the different parts of the game send messages but they don't know who receives them. This Pub/Sub system, is a good way to avoid the use of patterns like Singleton or static classes that couple the code. Every part of the game is totally independent of the others, allowing me to create unit tests easily.

1.1 Spaceship

SpaceshipState is different from other entities like asteroids or rockets, it's not just reacting to other entities, it has animations, input events, movement, guns. These independent parts of the logic are divided into modules, these modules are set up with their own ModuleDatas (ScriptableObjects), exactly like the entities.

These modules are injected by a factory. The references to them in the SpaceshipState are interfaces that allow to the SpaceshipState to update, setup or reset them, but it doesn't know what they are or what they do.

With this architecture, I tried to create a design by composition and followed the Dependency inversion principle, Liskov substitution principle and the Single-responsibility principle.

Best regards,
Michael