Intro:

Entire project uses only these third party assets/libraries:

1. Asteroid and SpaceShip models from asset store. They both are free.
2. Icon for powerups. They are obtained from google image for placeholder purposes.
3. “Log Viewer” to view in-game errors/warnings/exceptions etc. It is completely free.
4. “UnitySerializedReferenceUI” to assign/inject dependencies. https://github.com/TextusGames/UnitySerializedReferenceUI

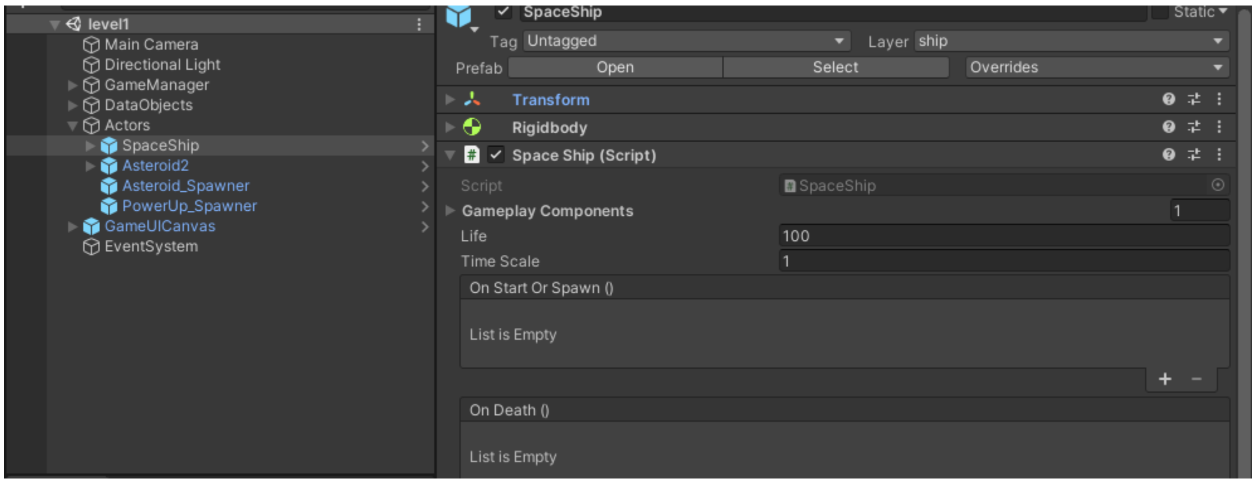
The project uses a gameplay framework I have written for this project, although it can be used in any game. I am fascinated by the “Actor-Gameplay Component” gameplay architecture of Unreal Engine 4. So I took this concept, added my own juice and built something now I call “KGameplayFramework”. I wrote this within the development cycle of this “Asteroid project”.

The project uses a support library called “SaveDataMan”, for saving and loading data without directly operating on PlayerPref.

I should give you a glimpse of my framework since the project is built on top of it. Game designers/artists can choose to skip this portion, but I strongly recommend not to.

KGameplayFramework:

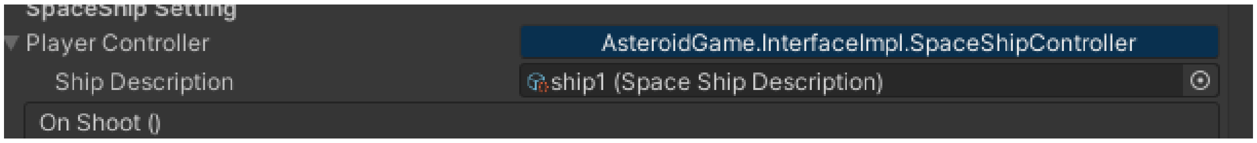
Everything placeable in the scene is “**Actor**”. So actors are actual entities that you can psychologically address as a person, things etc. So a cow is an actor but “grass eating handler” is not! Any class that is a child class or subclass of **GameActor** is technically an Actor.



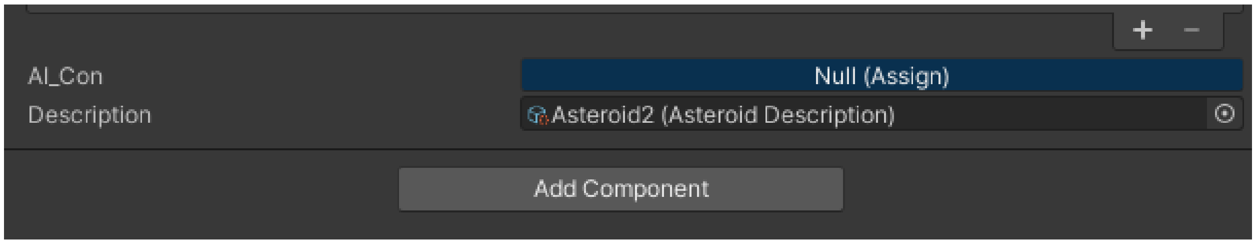
SpaceShip
Tag Untagged
Layer ship
Script SpaceShip
Gameplay Components 1
Life 100
Time Scale 1
On Start Or Spawn ()
List is Empty
On Death ()
List is Empty

Here you can see the spaceship actor.

There are two special actors that are subclassed from GameActor, **PlayerActor** and **AIActor**. They are important for gameplay. PlayerActor handles inputs by using the **new unity input system**. So you can custimize input without coding. How a player should be derived by using inputs is handled by interface “***IPlayerController***”. So you simply create any class and inherit this interface and implement the methods. Then simply in the “PlayerActor”(or any child class of it)’s inspector simply assign the controller by clicking the blue button.


Player Controller — AsteroidGame.InterfaceImpl.SpaceShipController
Ship Description — ship1 (Space Ship Description)
On Shoot ()

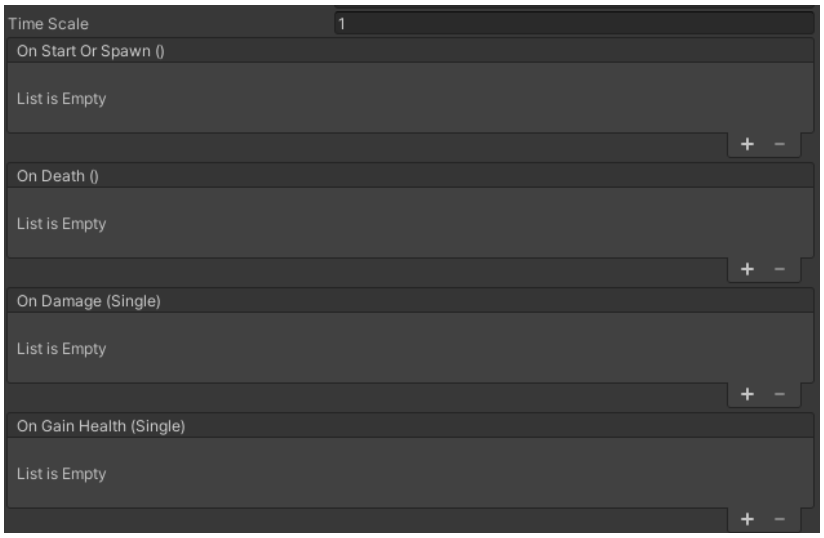
Very similarly you can assign your own AIController too. Our asteroid here is AIActor. Since the asteroid does not have any AI, we currently have null in it.


AI_Con — Null (Assign)
Description — Asteroid2 (Asteroid Description)

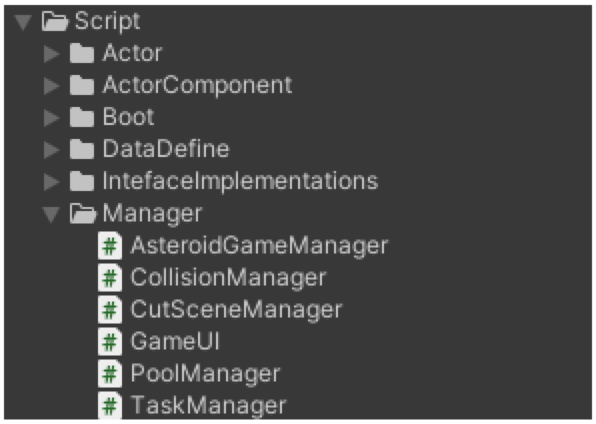
Actors can have **life value**. They **can be damaged**. They **can die**, after death the actor automatically becomes hidden. *The next logical step of framework evolution should be to automatically send the dead actors for object pooling. I have developed a pool manager that is actually being used for projectiles since there can be so many of them in a scene!*
Actors can have their own timescale so you can selectively time manipulate the actors.

Actors can have a list of “**GameplayComponent**”s. So what are they? Any class that is a subclass of GameplayComponent is technically a gameplay component. The component must be attached to the gameobject of the actor or any child gameobject of it. Actor inspector automatically populates the list of gameplay components. You can drag them up/down to change execution order. Generally speaking if a cow should eat grass, dance in the afternoon, age with time then I shall make a cow actor with “EatGrass” and “DanceCow” gameplay components. Aging with time features can be directly tied into the cow actor.

Builtin actors have various unity events you can listen to from code or editor.

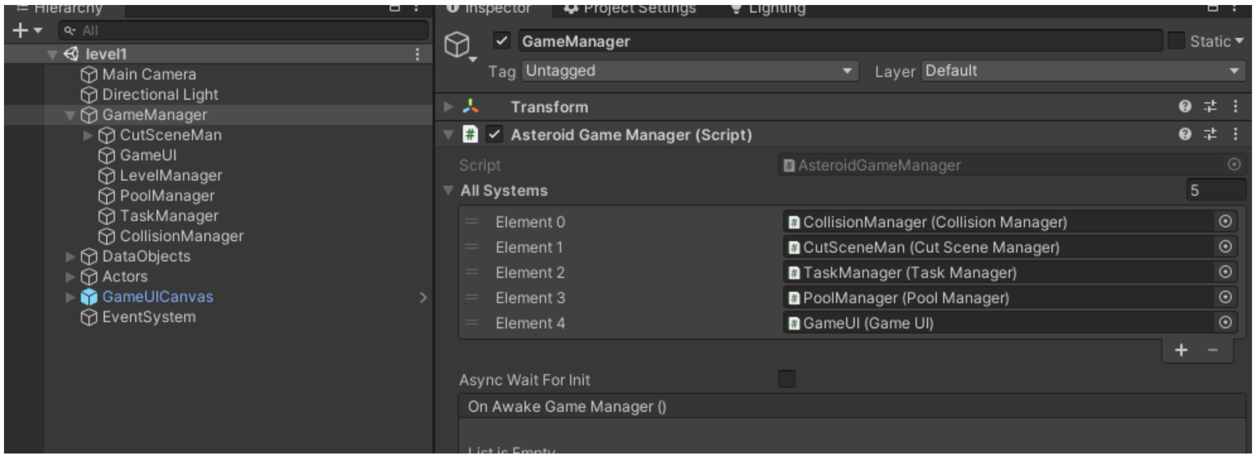

Time Scale — 1
On Start Or Spawn ()
List is Empty
On Death ()
List is Empty
On Damage (Single)
List is Empty
On Gain Health (Single)
List is Empty

Actors update themselves and their linked gameplay components.
So actors and gameplay components, is that it?
No actually. There is a notion of “GameSystem”. Suppose we need to download a set of texture assets from a server then we have to use them into a set of materials. So we need a “TexDownloader” running in the background. Or we have to play a cutscene by hand coding or managing a bunch of timeline assets. So we need a “CutSceneManager”
Those should neither be an actor or gameplay component. They are “GameSystem”. Any class subclassing from “GameSystem” is a game system. Here in this project I used five:



```
Script
    Actor
    ActorComponent
    Boot
    DataDefine
    IntefaceImplementations
    Manager
        AsteroidGameManager
        CollisionManager
        CutSceneManager
        GameUI
        PoolManager
        TaskManager
```

Notice the first one. It is not a game system. Rather it is a “GameManager”. GameManager manages the execution or game systems. GameManager also dictates how the game should be started or ended. A game must have a game manager. So simply have a subclassing class from GameManager(implement how the game starts and ends) and attach it to a gameobject and you are good to go!



```
level1
    Main Camera
    Directional Light
    GameManager
        CutSceneMan
        GameUI
        LevelManager
        PoolManager
        TaskManager
        CollisionManager
    DataObjects
    Actors
    GameUICanvas
    EventSystem
```

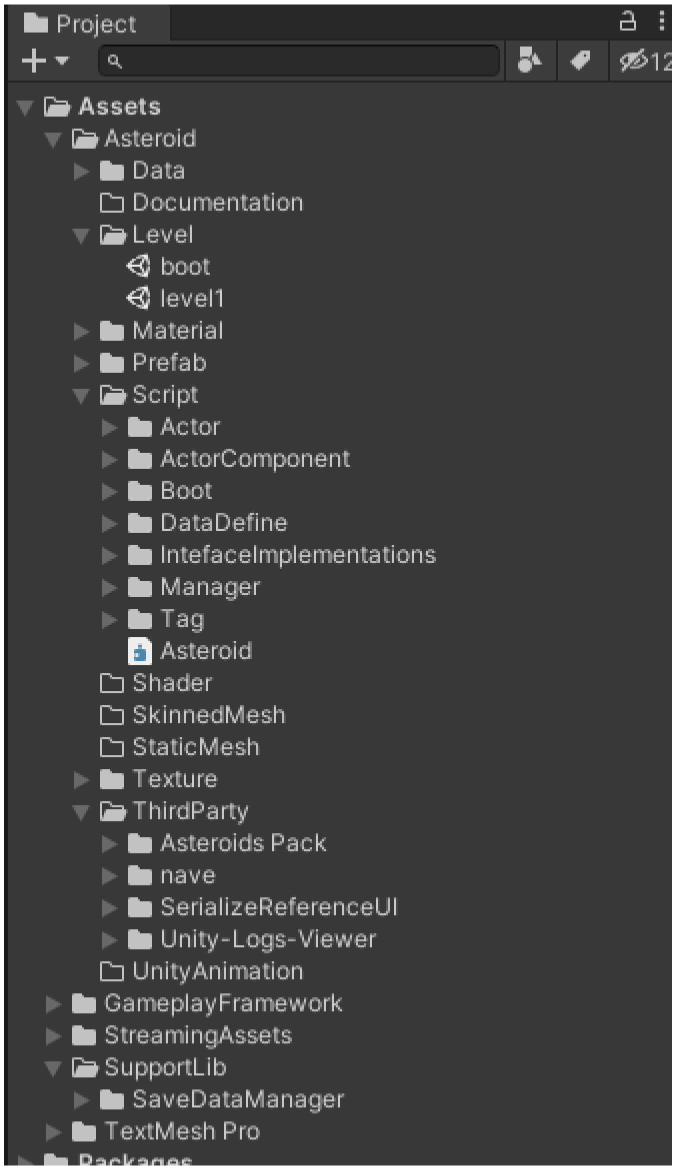
```
GameManager                                   Static
Tag Untagged        Layer Default
Transform
Asteroid Game Manager (Script)
Script              AsteroidGameManager
All Systems                                   5
    Element 0       CollisionManager (Collision Manager)
    Element 1       CutSceneMan (Cut Scene Manager)
    Element 2       TaskManager (Task Manager)
    Element 3       PoolManager (Pool Manager)
    Element 4       GameUI (Game UI)
Async Wait For Init
On Awake Game Manager ()
```

<u>Here I have an Asteroid Game Manager!</u>

Notice, no actor or gameplay component actually updates if your game has not been started.

Designer guide:

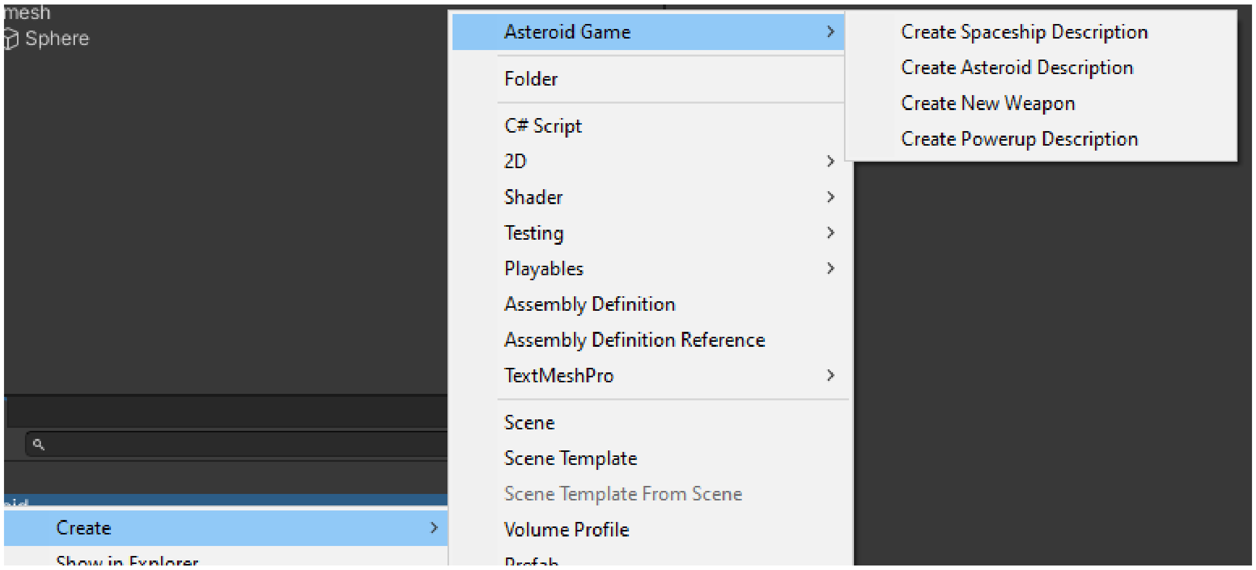
Project folder organization:



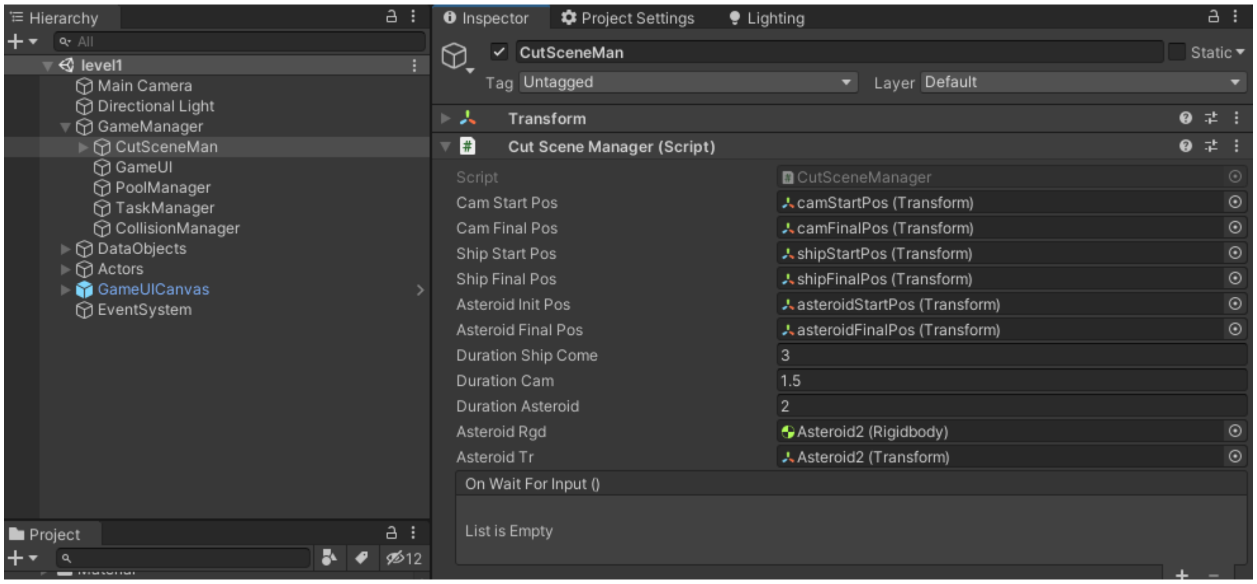
```
Project
Assets
  Asteroid
    Data
    Documentation
    Level
      boot
      level1
    Material
    Prefab
    Script
      Actor
      ActorComponent
      Boot
      DataDefine
      IntefaceImplementations
      Manager
      Tag
      Asteroid
    Shader
    SkinnedMesh
    StaticMesh
    Texture
    ThirdParty
      Asteroids Pack
      nave
      SerializeReferenceUI
      Unity-Logs-Viewer
    UnityAnimation
  GameplayFramework
  StreamingAssets
  SupportLib
    SaveDataManager
  TextMesh Pro
Packages
```

The boot scene contains game startup codes such as third party SDKs(Facebook, Analytics etc). Here I simply used a simple scene loader since I do not have any other startup codes.

To create various kinds of game data in form scriptable object, use context menu as such:



For startup cutscene, you can change the timing and positions of camera, spaceship and asteroid:



CutSceneMan

Tag Untagged Layer Default

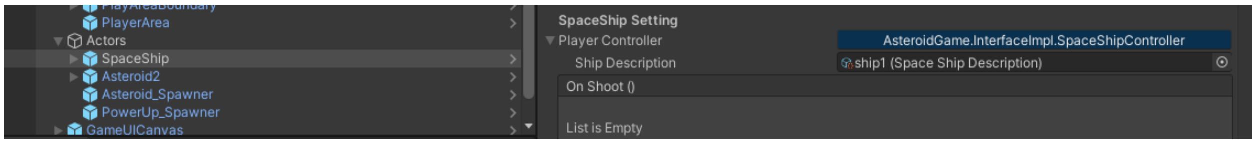
Cut Scene Manager (Script)

Script	CutSceneManager
Cam Start Pos	camStartPos (Transform)
Cam Final Pos	camFinalPos (Transform)
Ship Start Pos	shipStartPos (Transform)
Ship Final Pos	shipFinalPos (Transform)
Asteroid Init Pos	asteroidStartPos (Transform)
Asteroid Final Pos	asteroidFinalPos (Transform)
Duration Ship Come	3
Duration Cam	1.5
Duration Asteroid	2
Asteroid Rgd	Asteroid2 (Rigidbody)
Asteroid Tr	Asteroid2 (Transform)

On Wait For Input ()

List is Empty

You can customize spaceship by assigning its description scriptable object in the player controller section.

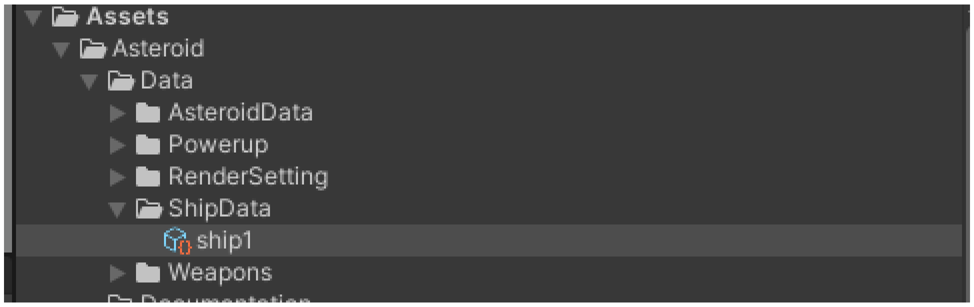


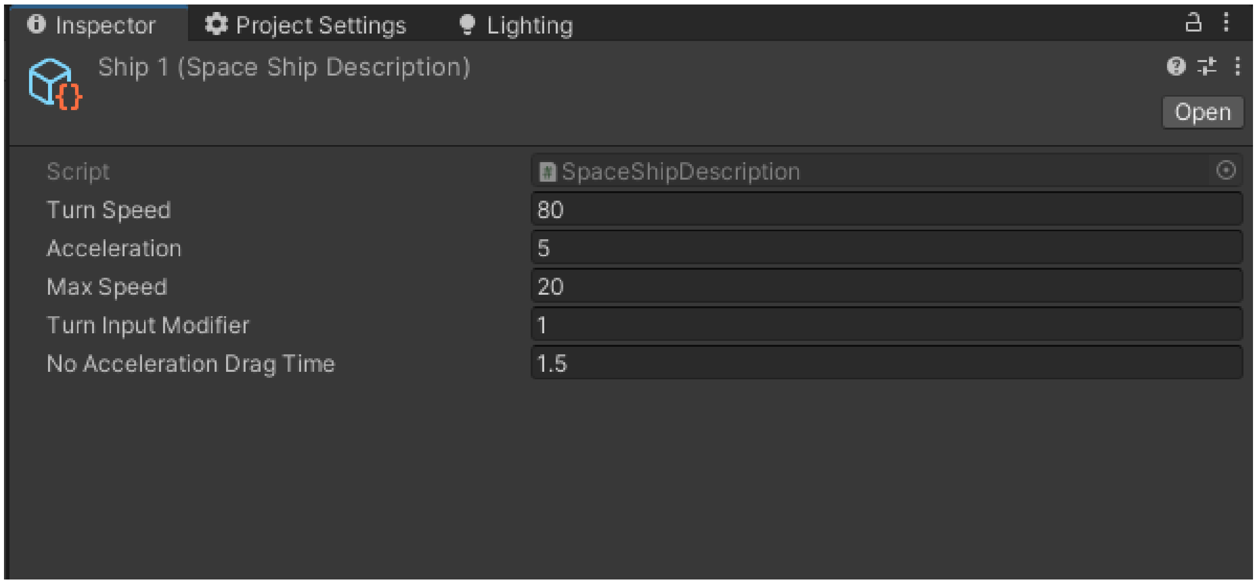
SpaceShip Setting

Player Controller	AsteroidGame.InterfaceImpl.SpaceShipController
Ship Description	ship1 (Space Ship Description)

On Shoot ()

List is Empty

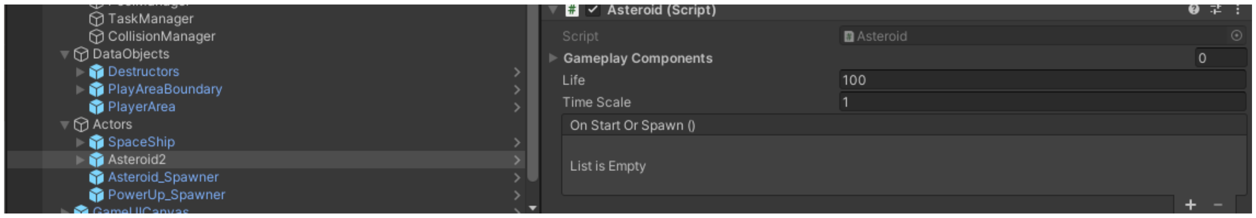




Script	SpaceShipDescription
Turn Speed	80
Acceleration	5
Max Speed	20
Turn Input Modifier	1
No Acceleration Drag Time	1.5

Turn input modifier -1 will rotate in opposite fashion with input. If you raise the drag time, you will have more dragging effect.

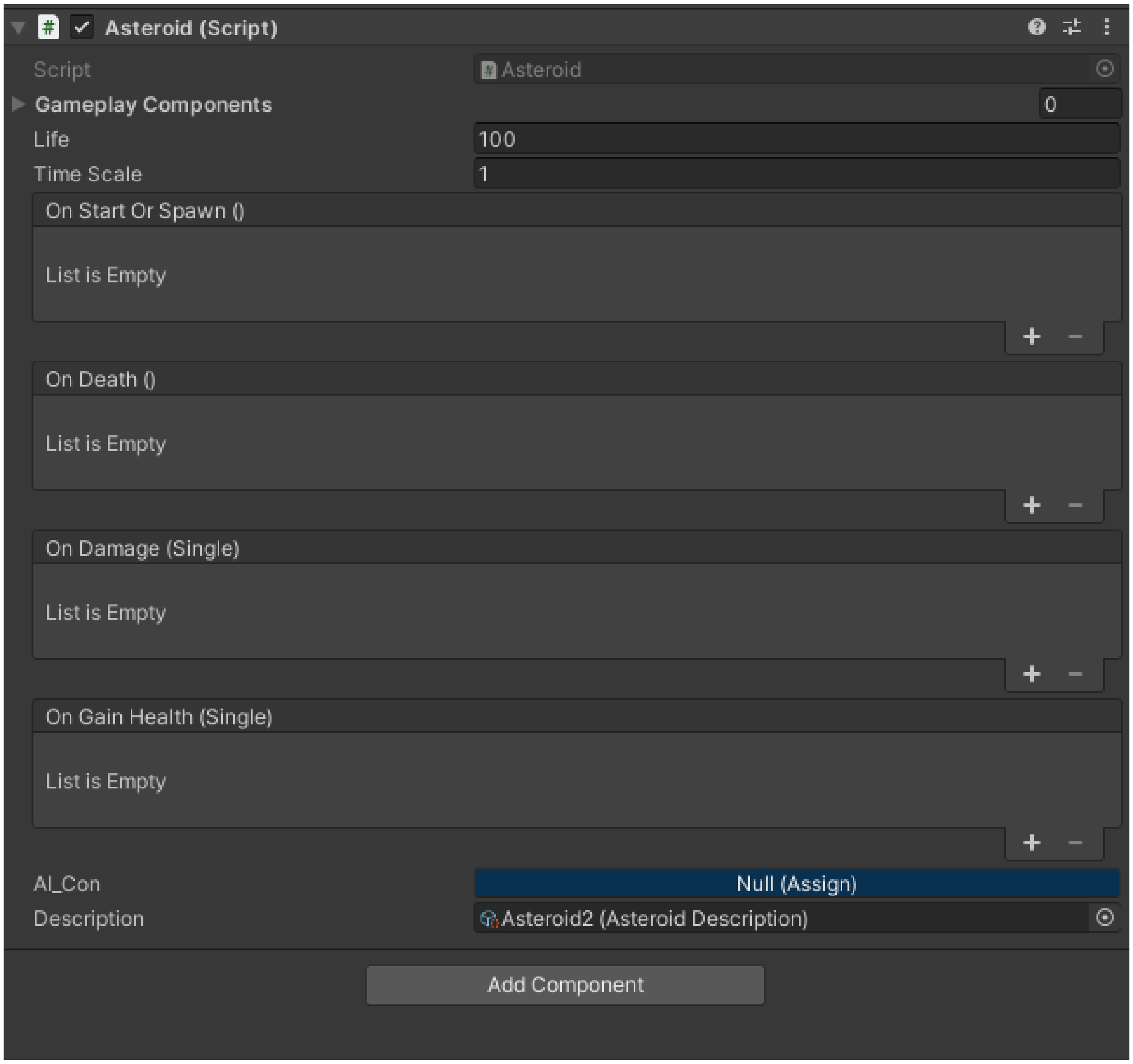
You can customize asteroid too:



Script	Asteroid
Gameplay Components	0
Life	100
Time Scale	1

On Start Or Spawn ()

List is Empty


Asteroid (Script)

Script	Asteroid
Gameplay Components	0
Life	100
Time Scale	1

On Start Or Spawn ()

List is Empty

On Death ()

List is Empty

On Damage (Single)

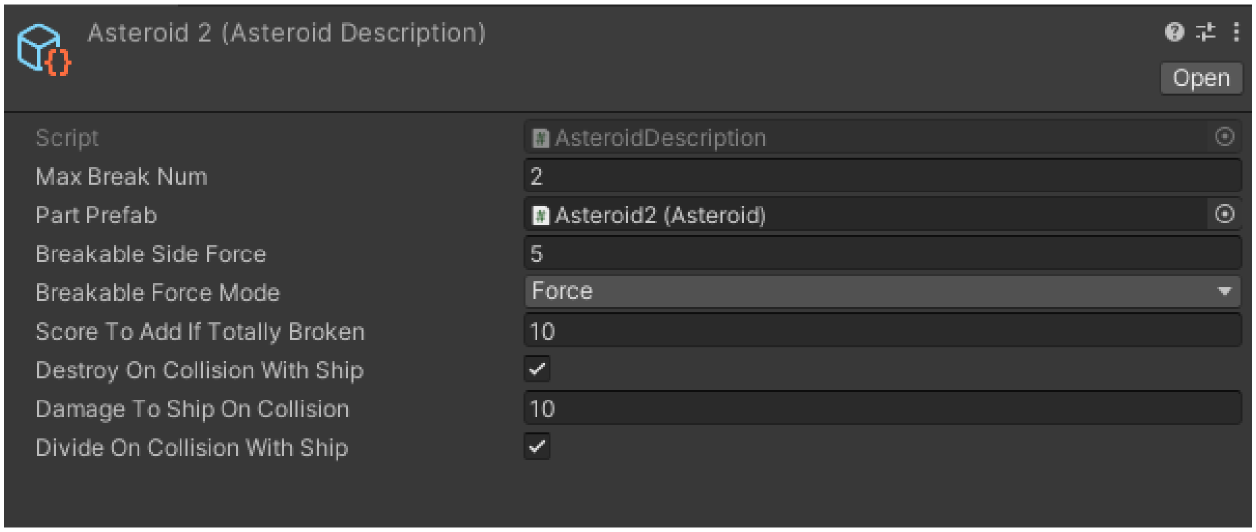
List is Empty

On Gain Health (Single)

List is Empty

AI_Con	Null (Assign)
Description	Asteroid2 (Asteroid Description)

Add Component


Asteroid 2 (Asteroid Description)

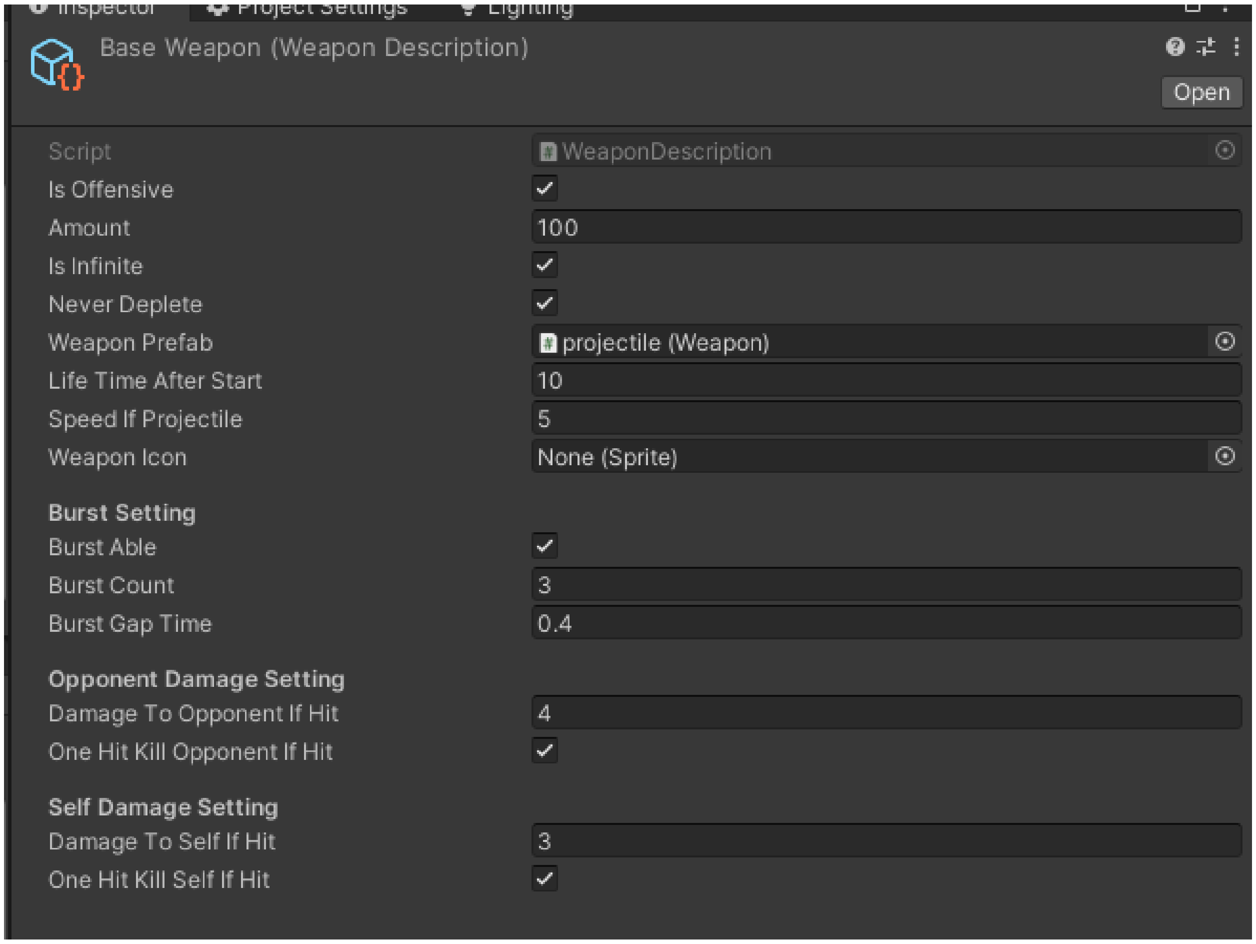
Open

Script	AsteroidDescription
Max Break Num	2
Part Prefab	Asteroid2 (Asteroid)
Breakable Side Force	5
Breakable Force Mode	Force
Score To Add If Totally Broken	10
Destroy On Collision With Ship	✓
Damage To Ship On Collision	10
Divide On Collision With Ship	✓

So this asteroid will only be divided into two parts, no more! Upon division, the smaller asteroid's shape is dictated by the "Part Prefab" in the prefab folder.

Now let us look at how to create a weapon and subsequently use them to create a new power up!
A gameobject that has a "Weapon"(or child class) actor, non kinematic rigidbody, and a set of colliders which are marked as triggers is technically a weapon actor!

Weapon actors alone can not do anything. We need weapon description data.

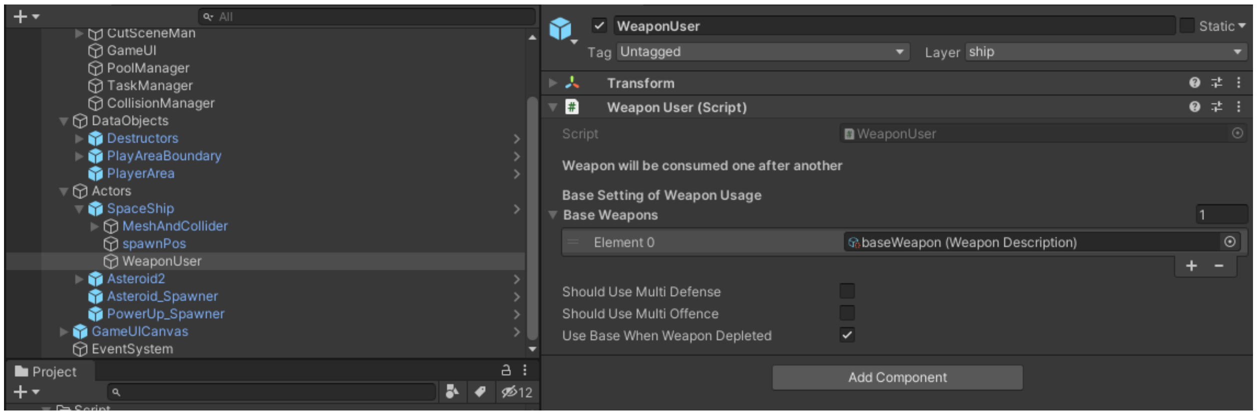


Base Weapon (Weapon Description)

Script	WeaponDescription
Is Offensive	✓
Amount	100
Is Infinite	✓
Never Deplete	✓
Weapon Prefab	projectile (Weapon)
Life Time After Start	10
Speed If Projectile	5
Weapon Icon	None (Sprite)
Burst Setting	
Burst Able	✓
Burst Count	3
Burst Gap Time	0.4
Opponent Damage Setting	
Damage To Opponent If Hit	4
One Hit Kill Opponent If Hit	✓
Self Damage Setting	
Damage To Self If Hit	3
One Hit Kill Self If Hit	✓

Projectiles are offensive weapons while shields are a defensive weapon. Amount is how many the weapon can be used before it is totally depleted. Or you can choose to have an infinite amount of weapons. If you choose that your weapon should never be depleted then only a custom advanced code can deplete the weapon. Weapon prefab are the weapon actor prefab you created ealier. If a weapon is infinite and can be depleted then the lifetime of such a weapon is the "Life time after start". So how can this weapon data be used?

Then this data must be used by the "WeaponUser" gameplay component that is linked by SpaceShip actor.

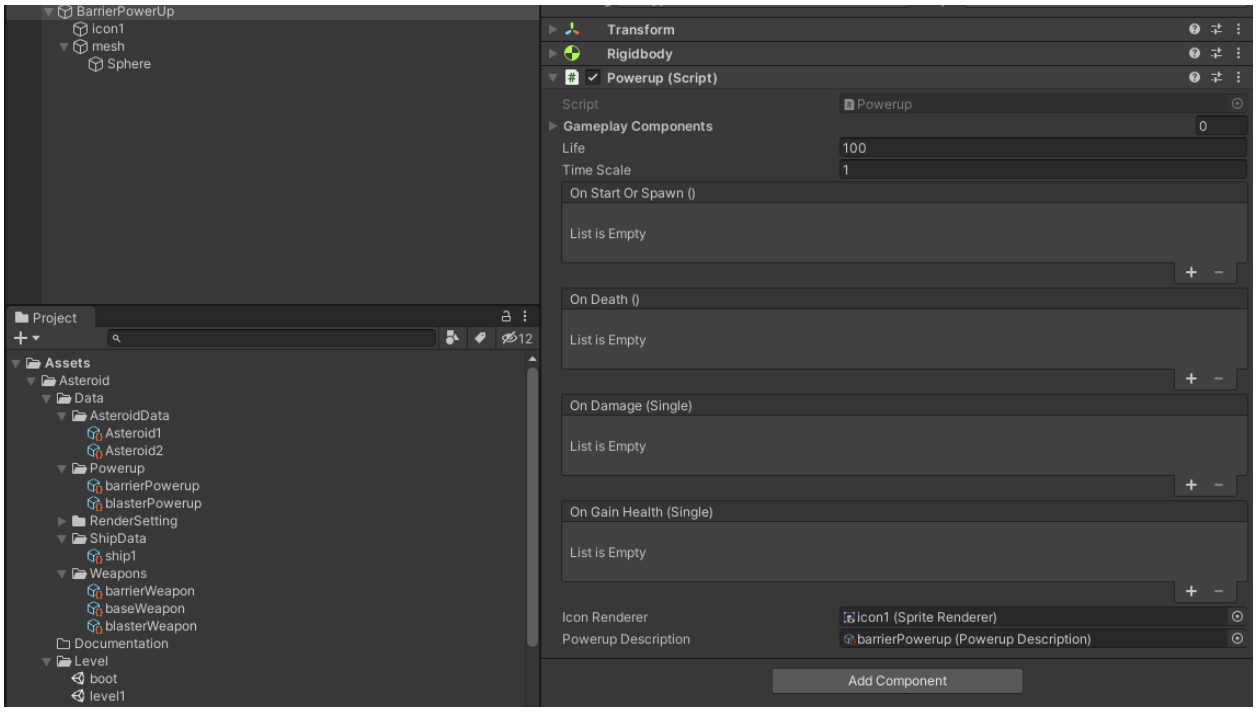

CutSceneMan
GameUI
PoolManager
TaskManager
CollisionManager
DataObjects
Destructors
PlayAreaBoundary
PlayerArea
Actors
SpaceShip
MeshAndCollider
spawnPos
WeaponUser
Asteroid2
Asteroid_Spawner
PowerUp_Spawner
GameUICanvas
EventSystem

WeaponUser — Tag: Untagged — Layer: ship

Weapon User (Script)
Script: WeaponUser
Weapon will be consumed one after another
Base Setting of Weapon Usage
Base Weapons: 1
Element 0: baseWeapon (Weapon Description)
Should Use Multi Defense: ☐
Should Use Multi Offence: ☐
Use Base When Weapon Depleted: ✓

When the shoot input is invoked, if there is no active weapon added then the base weapons will be fired. You can turn off this behaviour, thus without any equipped weapon the player won't fire anything! You can choose to use multiple offensive or defensive weapons upon fire. "Multi Defense", "Multi Offense" options are for this purpose.

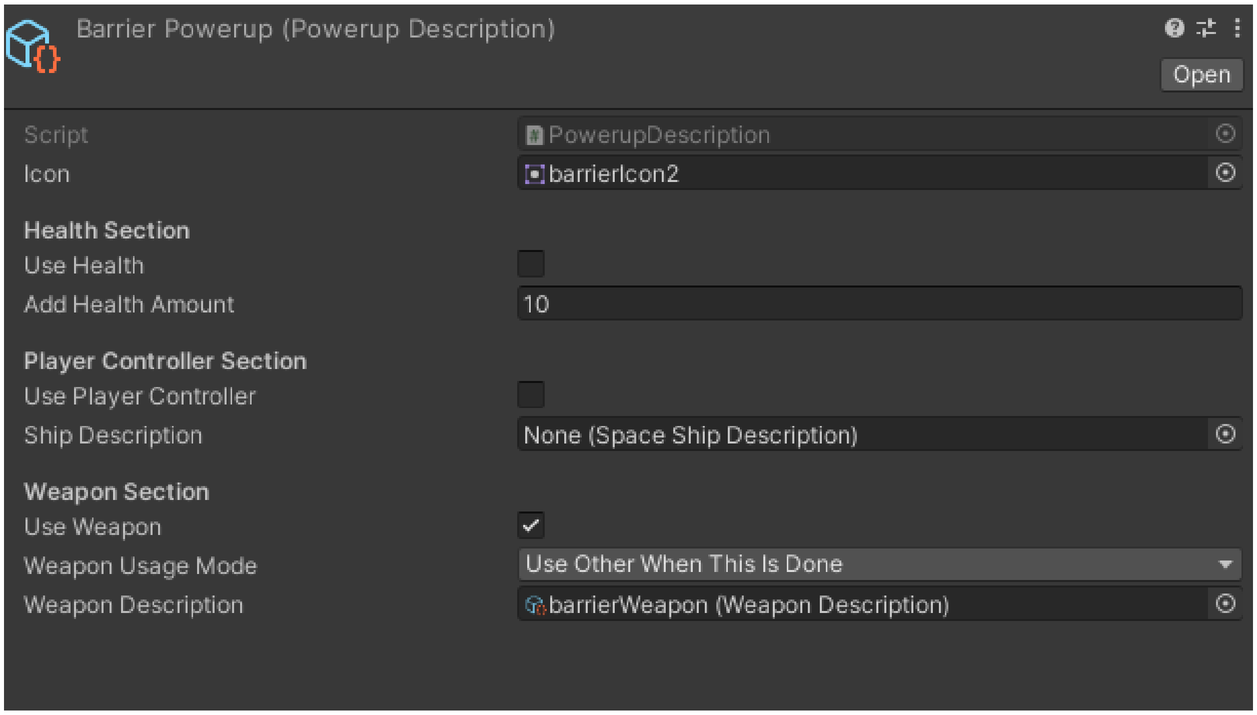
There is another way you can use weapons. Via powerups!
Let us create a powerup!


BarrierPowerUp
icon1
mesh
Sphere

Assets
Asteroid
Data
AsteroidData
Asteroid1
Asteroid2
Powerup
barrierPowerup
blasterPowerup
RenderSetting
ShipData
ship1
Weapons
barrierWeapon
baseWeapon
blasterWeapon
Documentation
Level
boot
level1

Powerup (Script)
Script: Powerup
Gameplay Components: 0
Life: 100
Time Scale: 1
On Start Or Spawn (): List is Empty
On Death (): List is Empty
On Damage (Single): List is Empty
On Gain Health (Single): List is Empty
Icon Renderer: icon1 (Sprite Renderer)
Powerup Description: barrierPowerup (Powerup Description)

Yes, similar to weapons, we use the "Powerup" class instead. This expects a powerup description data.


Barrier Powerup (Powerup Description)

Open

Script	PowerupDescription
Icon	barrierIcon2
Health Section	
Use Health	☐
Add Health Amount	10
Player Controller Section	
Use Player Controller	☐
Ship Description	None (Space Ship Description)
Weapon Section	
Use Weapon	✓
Weapon Usage Mode	Use Other When This Is Done
Weapon Description	barrierWeapon (Weapon Description)

And powerup data expects weapon data.
So usage chain is this:
Powerup Script-->Powerup data--> weapon data --> weapon script prefab --> weapon user

Weapon user's responsibility is to manage the weapon collection and release the proper weapon into the game world for consumption. Weapon scripts consume the weapon if it is defensive. Otherwise a state container class handles the consumption.