Hey guys,
Here some decisions I made:

Objects pool
Some objects are frequency in use, and therefore we require to instantiate a large amount of game object, in a relative short time.
Therefore, I decided to create an object pool, to allow of tracking unactivated game objects, so instead to creating a new one, we can call to the pool and reuse an unactive one.
To manage these kind of game objects, I created the interface "Spawnable". This interface is use to manage these items creating, by supporting the following functions:

- Init – setting the game object setting on spawn, this allow to reset the object, so it would be ready to use.
- `GetSpawnableObjetsType – a function that allow us to get an object identifier, so when we can ask the pool if there is an unactive object of a certain type, that can be in use.`
- Destroy – used to added functionality at the destruction moment (like dropping an item, doing an explosion etc'), and adding the gameobject to the unactive game object pool.
- Duplicate, used by the game bounties (explanation later), to duplicate the object, to create a mirror object, with the same properties.

Cyclic world
To create the cyclic world effect I decided to spawn a mirror object each time that a game pass the end of the screen. This way I could create a realistic effect when half object passed (showing the two half).
The game consists of two things:

1. Game bound area – the area that the player see on camera, were the game happen.
   This use a collider to destroy (mainly by using the spawnable destroy function), to destroy the objects that leave the screen.
2. Game borders – use to track object that pass the screen, and creating a mirror object corresponding to them. If the object is a spawnable type, it would use the duplicate function to preserve the object properties (moving direction, etc).

The code architecture:

GameCore
This part is the core of the game, it includes:

- Spaceship logic – giving the spaceship it main functionality: moving and shooting.
- Game Manager – managing the score, ship health. Can be assign with the relevant event: on health change, on healthzero, on score change, so it easy to future expand the game by adding features like: graphic effects, sound effects, etc. in addition it is more easy to maintain the game rather than directly active the UI / sound.
- Spawnable – (see above)
- Spawn Pool Manager – (see about)

- Calculation tool – used to prevent code duplication for calculation that are often in use by multitype scripts.

World Bound
Doesn't depend on other component rather than the spawnable interface, that it use.
includes a script for automatically setting the Game Bound, and the game borders positions and dimensions.
(see above)

SpawnableObjects
include all the objects that implement spawnable, and the code that spawn the asteroids in the game:

- Asteroid Logic
- Bullet Logic
- Explosion
- SpawnableObject – implements the basic functionality of the spawnable interface.
- Buffs – adding functionality to prefabs: adding health / score when clicked.

UI
Doesn't depend on other components.

- UI manager – responsible for updating score / health, or showing end game screen by listening to the game manager.

Features
Some independent functionality that can be added to game objects, like auto rotation, or ability to get damage, and calling a callback.

Includes: Health Logic, Rotate

GameInit
Sets the GameUI to listen to the gameManager.
Sets the GameManager to listen to the spaceship healthLogic script.

I decided to save the health on the game manager and to track it with a health logic, since the world is cyclic, and I implemented the cyclic feature with duplication. Therefore, there need to be a single source that track the game data.

- ⇨ **I set the UI border to auto fit to the screen, , instead of a fixed size of 200 px**
  , since it let the game to be more adjustable to other screen sizes, and its look better.
- ⇨ **I attached a code architecture diagram.**

Hope you enjoyed from the game,
Ilan.