Space Wave Shooter

Contents

Project Overview

Thank you for purchasing *Space Wave Shooter*. This is a Unity template pack of a complete, 3D wave shooter game.

You control a spaceship with the goal of destroying all enemy ships throughout the various waves. In-between waves, you can purchase items and upgrades from the shop.

The game is won when you have completed all the waves.

Project Features

- Complete wave shooter game.
- Player ship controller.
- Enemy ships.
- Mobile support.
- Item and upgrade shop.
- Object pooling system.
- Documented code.

Managers

Game Manager

Manages starting and ending waves, player money and game states.

Level Manager

Manages loading level files and setting the current level for the player to play.

Enemy Manager

Manages the spawning of enemies for a specified wave.

Particle Manager

Manages the creation of particle effects.

public void Spawn ()
Spawns in a requested particle at a certain position.

Audio Manager

Manages the playing of sound effects.

public void PlayWithTempAudioSource ()
Creates a temporary empty GameObject with an audio source to play a sfx from. Used for when the object calling the sfx gets destroyed such an explosion upon death.

public void Play ()
Plays a sound effect from an audio source, with an option to randomize the pitch.

Menu

Scripts

The menu is overall managed by the **Menu** script. This manages changing pages.

Each screen has a base class of **MenuScreen** which runs the *OnOpenScreen* when enabled. The play and settings screens each have their own script which inherits from MenuScreen: **PlayScreen** and **SettingsScreen**.

Object Pooling

This project comes with a custom object pooling script (**Pool.cs**) which is used to pool all sorts of objects such as: enemies, projectiles and particle effects. Pooling these objects increases performance, especially with a game like this where things are being spawned constantly.

In the **Game** scene, the *Pool* script is attached to a GameObject prefab called **_Pool**.

public GameObject Spawn ()

The function you call to spawn in an object (alternative to Instantiate).

Parameters:

- *GameObject* prefab
    - The prefab you want to spawn in.
- *Vector3* position
    - World position of the object to spawn.
- *Quaternion* rotation
    - Rotation of the object to spawn.
- *Transform* parent = null
    - (optional) Parent to automatically assign to the object to spawn.

public void Destroy ()

The function you call to destroy a pooled object.

Parameters:

- *GameObject* obj
    - The object you want to destroy.
- *float* time = 0.0f
    - (optional) Time delay before destroying object.

Shop

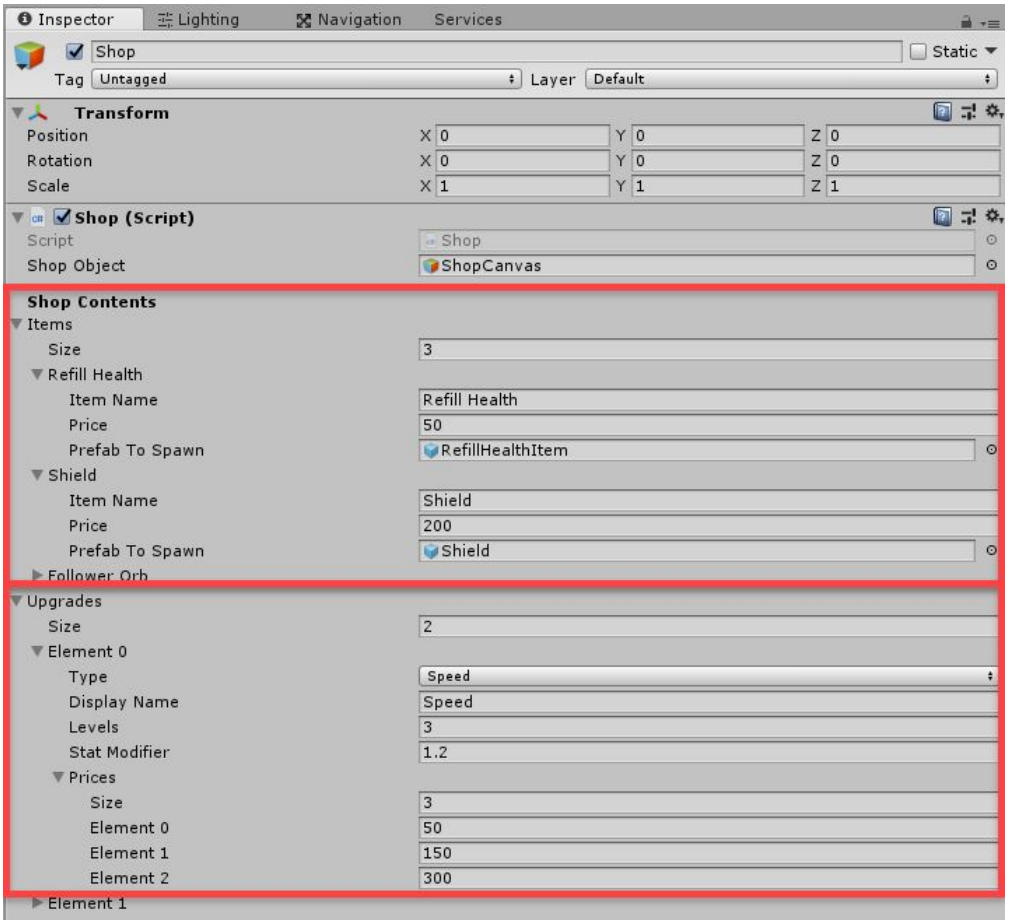
When a wave is not in progress, the player can open up the shop (by pressing the shop button) and choose to buy items and upgrades.

Items are things that get spawned in such as a shield or a follower orb. The player cannot purchase another one until the one they have is destroyed (only for the shield).

Upgrades are stat increases. Each upgrade can have a few levels where each time a stat is upgraded, the price also goes up for the next level.

The shop is managed by the **Shop** script, where the item and upgrade buttons are initialized and set. When a player clicks on a button, it goes through the script: seeing if they can purchase it and if they have enough money.

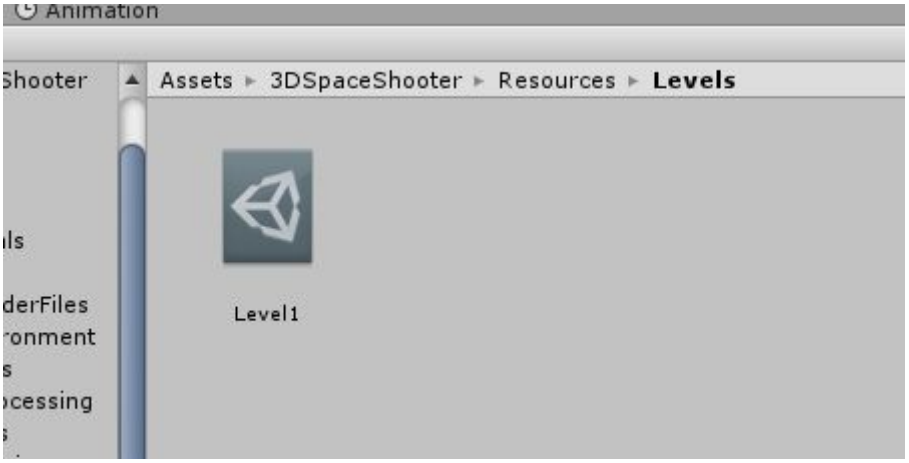
The shop is its own prefab which has the canvas as a child. In the inspector, you can add and edit the shop items and upgrades.



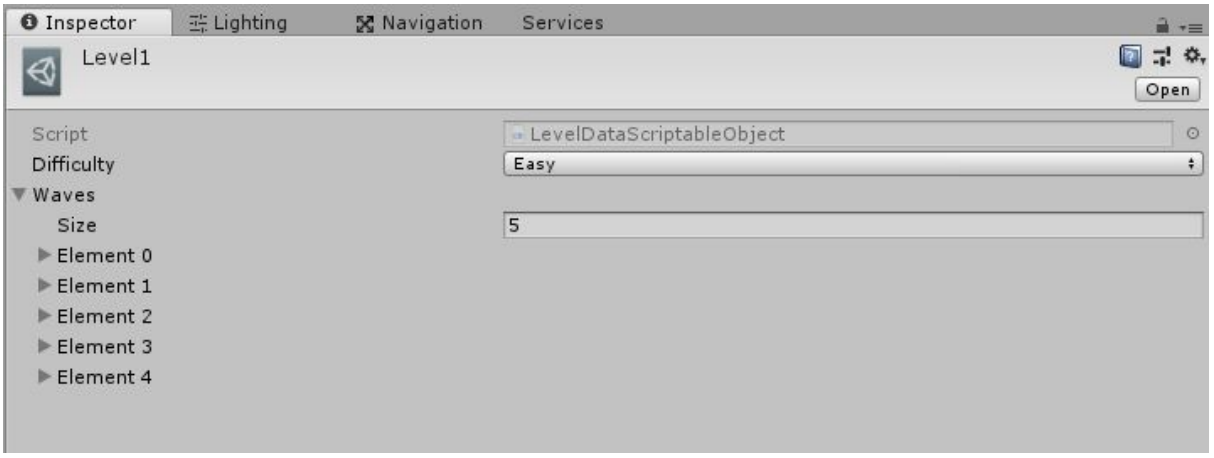
How to Create a Level

Levels in this game are all ran on the same scene (**Game** scene). The only difference between levels, is the waves. Levels are stored as asset files in the *Resources/Levels* folder. These are **LevelDataScriptableObject** scriptable objects.

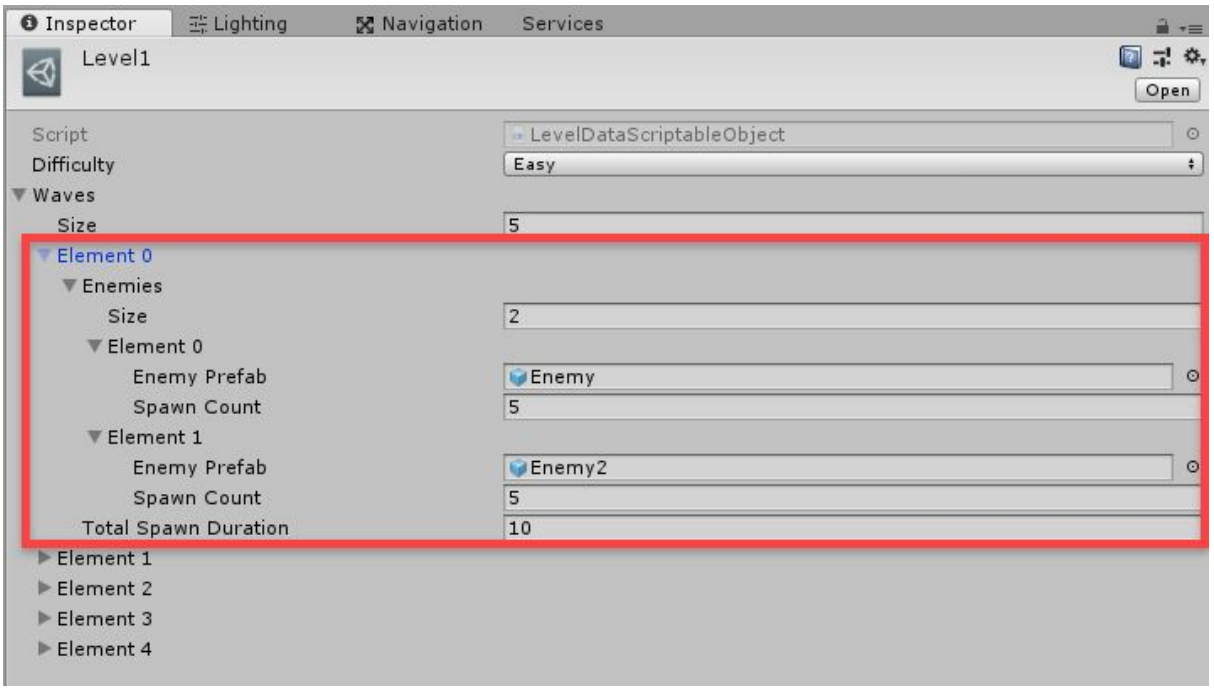
1. Create a new level asset by right clicking in the *Resources/Levels* folder and selecting *Create* > *Level Data*. Rename this to whatever you want, although do note that levels are ordered alphabetically.



2. In the Inspector, there are two values. **Difficulty** and **Waves**.
    a. **Difficulty** is an estimation on how difficult the level is (easy, medium, hard).
    b. **Waves** is an array which contains the info for each wave.



3. Inside each wave element, are two values: **Enemies** and **Total Spawn Duration**.
    a. Enemies is an array where you put the type of enemy you want to spawn and how many.
    b. Total Spawn Duration is the amount of time it will take to spawn all the enemies for that wave.



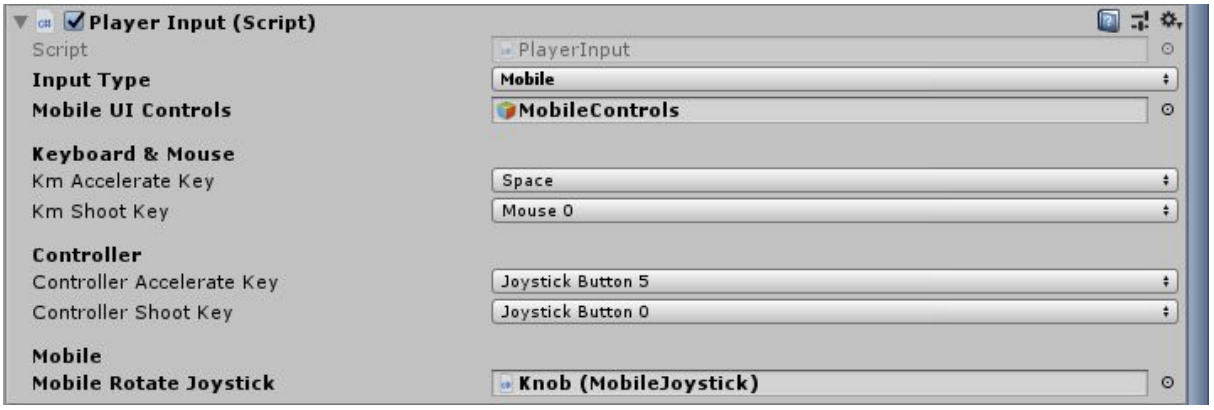
Now you have a new level, which should now automatically appear on the menu.

How to Change Input Type

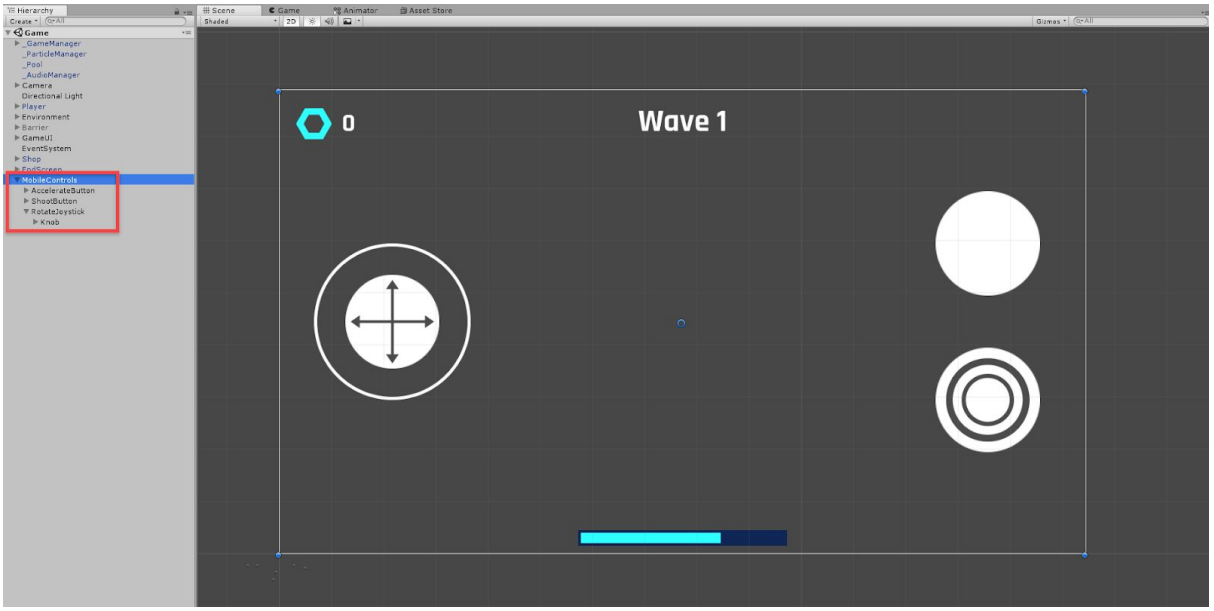
In this game, there is support for **three** different types of control schemes. Keyboard/mouse, mobile and gamepad controller.

All of the inputs are ran through the **PlayerInput.cs** script (located on the **Player** object). Here, you can…

- Change the input type (Keyboard Mouse, Controller, Mobile).
- Change the keys for keyboard and mouse.
- Change the keys for controller.



The mobile controls are found on the **MobileControls** canvas object. By default this is disabled, but enabled during play if the input type is set to Mobile.



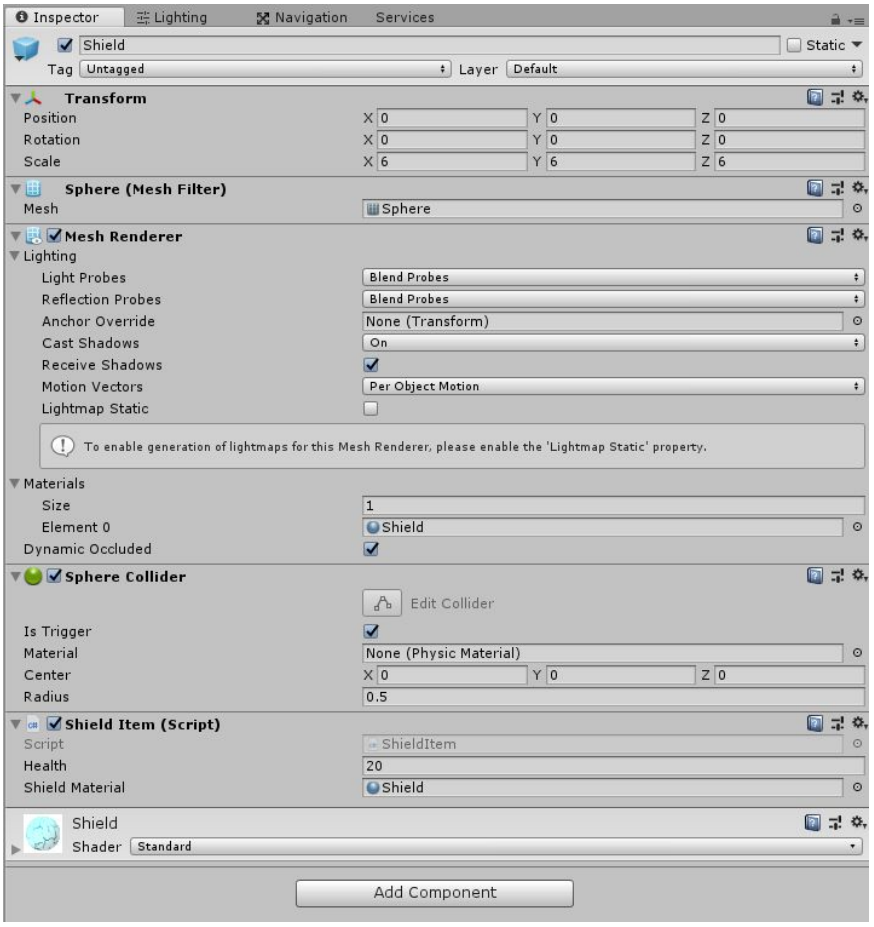
How to Create a New Shop Item

In the **Shop**, you can purchase an item. When this is purchased, the linked prefab is instantiated. This prefab will have a script on it which inherits from the base **Item** class.

A few things happen when the item is created.

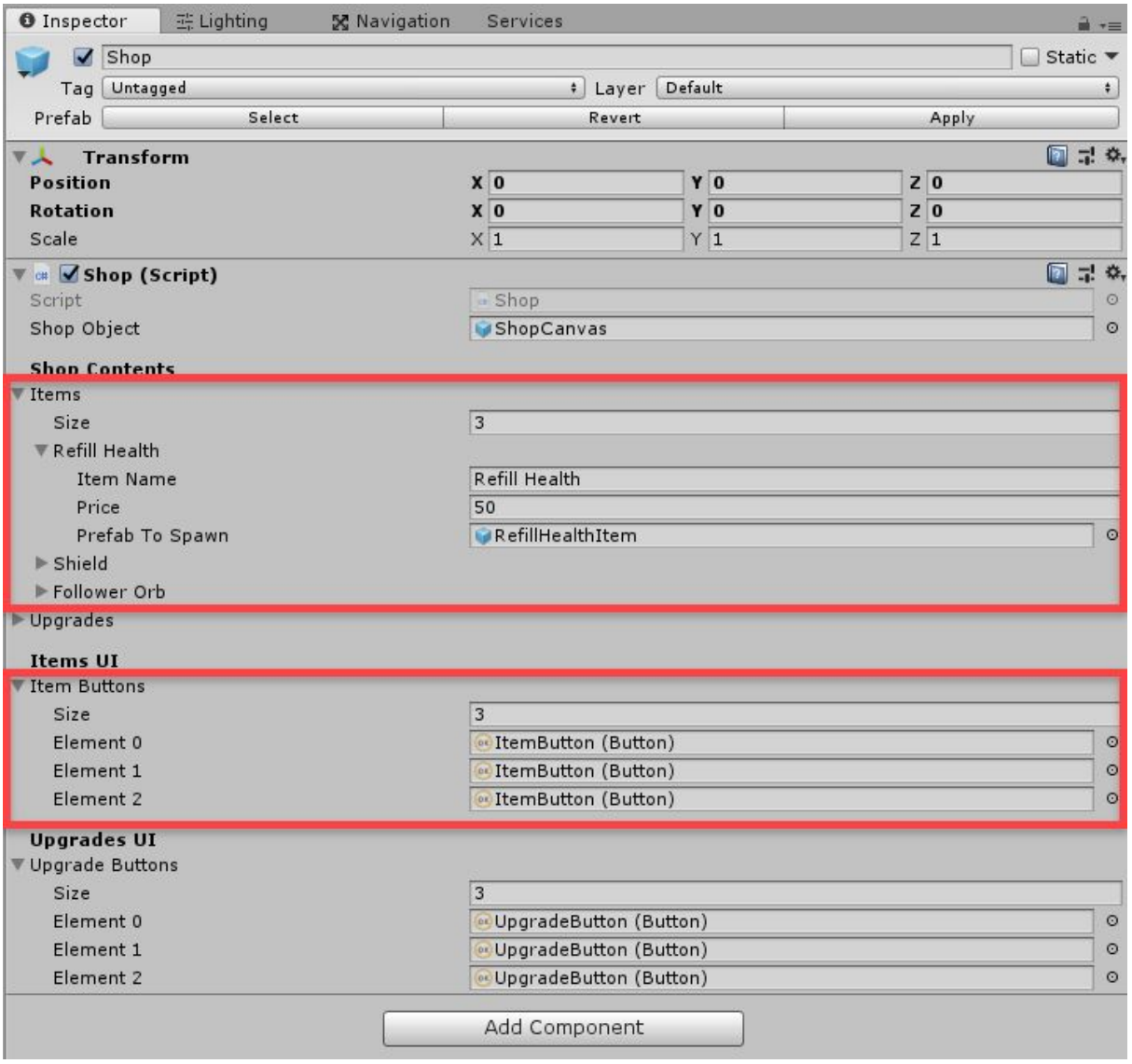
1. The **Initialize** function is called (this can be overridden with stuff you want to setup).
2. The **SetProperties** function is called (this just links the item).

Let's have a look at the **Shield** item. This prefab has a mesh renderer (rendering the shield) and a sphere collider (detecting bullets). The **ShieldItem.cs** script inherits from **Item**.



When this item is purchased, the shield prefab is instantiated. The script overrides the **Initialize** function, where it becomes a child of the player and sets its local position to 0. When enough bullets have hit the shield, it calls the **DestroyItem** function (on the base Item class), which destroys the item and lets the shop know that the player can buy it again.

To add this item to the shop, select the **Shop** object and add a new element to the **Items** array. Fill in the details and link the item prefab. Then add a new element to the **Item Buttons** array (make sure to duplicate one of the item buttons so it can fit all of the items).



Scripts

Data

LevelDataScriptableObject.cs
Scriptable object which we use to create levels and waves. Contains data for levels and waves which the game then can read. Levels are stored in the *Resources/Levels* folder.

Enemies

Enemy.cs
Logic for the base enemy in the game. Chasing player, shooting at them, etc. Inherits from *Ship*.

BomberEnemy.cs
Logic for the bomber enemy. Chasing player and blowing them up. Inherits from *Ship*.

Interfaces

IDamageable.cs
Interface for all objects that can be damaged (player, enemies, projectiles).

Items

Item.cs
Base class for all items.

OrbItem.cs
Moves the orb around the player, damaging enemies it hits. Inherits from *Item*.

RefillHealthItem.cs
Refills the player's health then goes away. Inherits from *Item*.

ShieldItem.cs
Creates a protective shield around the player. Inherits from *Item*.

Managers

AudioManager.cs
Manages the playing of sound effects.

EnemyManager.cs
Manages the spawning of enemies for a given wave.

GameManager.cs
Manages the game state, player money and starting/ending waves.

LevelManager.cs
Manages the loading of level files and setting the current level.

ParticleManager.cs
Manages the creation of particle effects.

Menu

Menu.cs
Manages the changing of menu screens.

MenuScreen.cs
Base class for a menu screen.

PlayScreen.cs
Manages the loading of level files and playing a level when the corresponding button is pressed. Inherits from *MenuScreen*.

SettingsScreen.cs
Manages the changing and saving of settings. Inherits from *MenuScreen*.

Mobile

MobileButton.cs
Mobile UI button for shooting or accelerating.

MobileJoystick.cs
Mobile joystick for rotating the ship.

Player

CameraController.cs
Makes the camera follow the player.

PlayerController.cs
Controls the movement and general actions of the player. Inherits from *Ship*.

PlayerInput.cs
Detects input based on type requested and calls corresponding events.

Projectile

Projectile.cs
Detects collisions with *IDamagable* objects, damaging them.

TrackingProjectile.cs
Projectile which follows a target. Inherits from *Projectile*.

Shop

Shop.cs
Manages the overall shop actions: initializing buttons, buying upgrades and items.

ShopItem.cs
Data class for shop items.

ShopUpgrade.cs
Data class for shop upgrades.

UI

GameUI.cs
Manages the general game UI. Health bar, wave text, money text.

UIButton.cs
Script all in-game buttons have. Adds more feedback such as scaling and sound effects.

Other

Pool.cs
Object pooling script.

Ship.cs
Base class for all ship entities. Manages acceleration and rotation.

ShipShooting.cs
Base class for all ship shooting. Spawning projectiles.