User Guide

Ultimate Spawner 2.0

Spawning made easy

Trivial Interactive

Version 2.2.x

Ultimate Spawner 2.0 is a powerful asset that is feature package and makes adding spawning systems to your game easy. Ultimate Spawner 2.0 includes a number of useful spawner systems such as spawn points, spawn areas and more which are suitable for all your spawning needs, whether it be a 2D game or 3D. Our spawner hierarchy system allows you to create complex logical spawning systems with little effort saving you time and work over implementing custom solutions.

Features

- A complete easy to use spawning system
- Quickly setup spawn locations and have your enemies spawning in no time
- Occupied checks determine whether spawn locations are available for spawning
- Using spawn masks you can easily specify which spawnable items are allowed at particular spawners
- Create spawnable item collections where each item has a spawn chance value
- Create hierarchical spawn systems where spawn requests can be delegated to child spawners
- Hierarchical spawn systems allow for complex rule based spawning behaviours
- Spawn triggers allow spawning only when the player is inside a bounding volume
- Spawn targets allow distance based spawning for a particular target
- Despawners allow you to conditionally despawn items based on conditions.
- Easily integrate pooling support
- Highly customizable spawners
- Suitable for 2D and 3D games
- Fully commented C# source code
- Comprehensive .chm API documentation

Add-Ons

An additional waves add on can be purchased separately on the asset store and allows you to create wave based spawning systems via an intuitive node editor. Search for **Ultimate Spawner 2.0 – Waves Add On** on the asset store for more information.

Upgrade Guide

Version 2.1.0

This version contains breaking changes and may cause existing projects to no longer work as expected when imported. Take care when upgrading in existing projects and we would always recommend backing up the project before upgrading an asset.

The breaking changes are listed below:

- **Spawnable Items:** The spawnable items script has been heavily modified causing existing spawnable items assets to no longer work as expected or be compatible as of version 2.1.0.

  Resolution: To fix existing spawnable items assets you will need to select them via the inspector window and remove all entries manually. Once this is done you can use the 'Add Prefab' button to re-add the spawnable items to the asset or drag and drop if preferred. Alternatively, you can delete the spawnable items asset completely and then re-create it via the Ultimate Spawner menu 'Project Window - Right Click -> Create/Ultimate Spawner/Spawnable Items'.

  You can also add a custom spawnable item provide script into the spawnable items asset which can be used for better pooling support as well as allowing support for non-prefab assets such as addressables. See the spawnable items and pooling sections for more details.

Contents

Project Management

Update Ultimate Spawner 2.0

When a new version of Ultimate Spawner is released you may like to update to the newer version as it will usually contain bug fixes and new features. The update process is very simple and is as follows:

1. (Optional) Create a project backup. This step is not essential but is highly recommended to avoid corruption or data loss.
2. Using either your web browser or the built in Asset Store window, find the Ultimate Spawner asset page and select the download button.
3. Simply import the updated package into your project using the asset import popup window that appears.
4. Done! Unity may recompile scripts and update a few things but you should now be updated to the latest version.

Update issues

If you are receiving script errors or other errors after updating then here are a few things to try:

- Try an editor restart. Often this simple step can fix many errors as a result of changes to the project files.
- Try a clean update of Ultimate Spawner:
    1. (Optional) Create a project backup as per normal update procedure.
    2. Locate the root install folder of Ultimate Spawner in the project window. This folder should be named 'UltimateSpawner2.0' if it has not been renamed.
    3. Delete this folder from the project window.
    4. Follow the standard update procedure to download the latest version of Ultimate Spawner from the asset store.
- If you are still having issues then please contact us for further assistance. Contact and support information can be found on the last page of this document.

Uninstall Ultimate Spawner 2.0

To uninstall Ultimate Spawner you will need to focus the 'Project' window in the Unity editor and identify the root install folder of Ultimate Spawner. If the folder has not been modified then it should have the name 'UltimateSpawner2.0' and will be located under the 'Assets' folder by default. Simply delete this folder and all of its contents from the project window and Ultimate Spawner will be removed from the project.

Folder Structure

As already mentioned, Ultimate Spawner imports all of its assets under a root 'UltimateSpawner2.0' folder as shown in figure 1 and the contents are organised into appropriate folders as described below:

Figure 1

- **Demo:** This folder contains only demo/example assets which are used to demonstrate how the various components of Ultimate Spawner could be utilized in a game. This folder can be safely deleted if not required and the functionality of Ultimate Spawner will not be affected.
- **Plugin:** This folder contains a managed assembly used by Ultimate Spawner for displaying GUI elements in the Unity editor such as Inspectors and property drawers. This plugin is required by Ultimate Spawner at edit time only and will not be included in a built game. Full source code for Ultimate Spawner is included under the 'Scripts' folder.
- **Scripts:** The scripts folder contains all of the C# source code for Ultimate Spawner including both runtime and editor types. For more information about the scripts included in the asset take a look at the included scripting reference.

Quick Start

The following section will take your though the basics in order to get Ultimate Spawner up and running as quick as possible. Please note that many of the features will not be discussed in this section but will be covered later in this document.

1. Create a spawnable items asset. With the project window focused, right click and select 'Create -> Ultimate Spawner -> Spawnable Items' in a suitable folder and give the asset a name. A spawnable items asset is required to provide a collection of items that can be spawned.

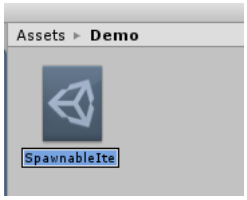

Figure 2

2. Ensure that the spawnable items asset is selected and focus the inspector window. Click the 'Add' button to add a new spawnable item entry. Once you have added a new entry you will need to drag a prefab into the object field named 'Prefab' and we will leave the other options at their default values. This prefab should be something you want to spawn into your scene like an enemy or you could use one of the example prefabs included with Ultimate Spawner for testing purposes.

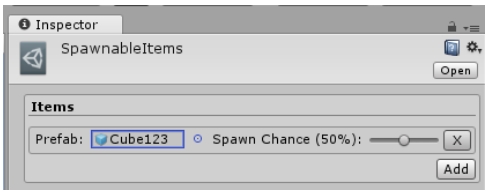

Figure 3

Note that you can also click the small icon to the right of the field to show the asset browser window and an unlimited number of spawnable items may be added to the asset.

3. We will need to turn our attention to the scene to create a spawner which will provide a specific location in 3D space where these items/enemies can be spawned. To do this go to the menu 'GameObject -> Ultimate Spawner -> Spawn Point' to create a simple spawn point game object in the scene. You can move and rotate this spawn point as required using the Unity translate and rotate tools.

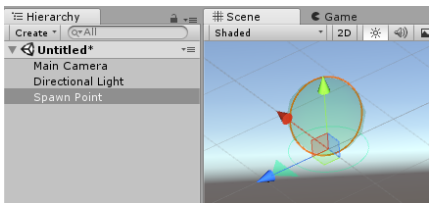

Figure 4

4. Ensure that the newly created spawn point is selected and focus the inspector window. Locate the field on the 'SpawnPoint' script component named 'Spawn Items' and drag the previously created spawnable items asset from the project window into this slot. This will allow the spawner to access these spawnable items.

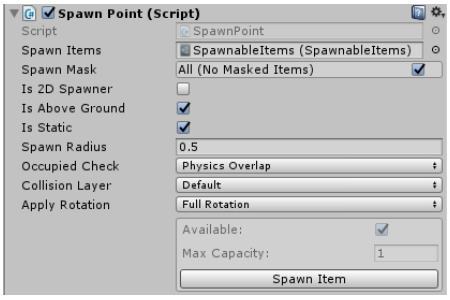

Figure 5

5. Now that we have a simple spawner setup we will need a way to control how and when it will spawn items or enemies. You could do this from a script using the public API but for this example we will use one of the included spawn controller components to control the spawning behaviour. Add an infinite controller to the scene by going to the menu 'GameObject -> Ultimate Spawner -> Infinite Controller'. An infinite controller will issue spawn commands to an assigned spawner continuously based upon its configuration. See the Spawn Controllers section for more information.

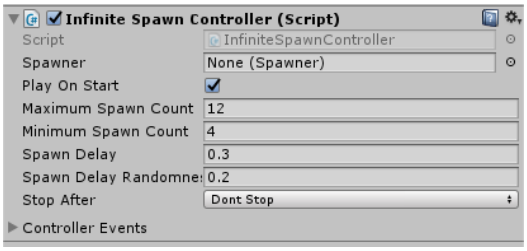

Figure 6

6. We will need to assign the spawn point that we created earlier to the infinite controller. This will mean that all spawn requests issued by the controller will be directed to the spawn point. Simple drag the spawn point game object from the hierarchy window into the 'Spawner' slot of the infinite controller component in the inspector window.

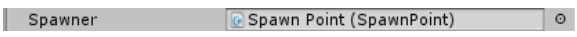

Figure 7

7. You can now run the game by pressing the play button and you should see that the items setup earlier are now being spawned at the spawn point at regular intervals.

Ultimate Spawner 2.0 has many more features to offer and it is recommended that you continue reading to get a better understanding of how everything works. Alternatively you can lookup specific topics as required while you are adding spawning systems to your game.

Spawnable Items

Ultimate Spawner uses a special type of asset that stores a collection of prefabs that can be spawned by a spawner. Each of these prefabs has its own spawn chance value that is used to specify the likelihood of the item being selected for spawning.

Create Menu

You can easily create a spawnable items asset from the menu '**Assets -> Create -> Ultimate Spawner -> Spawnable Items'**. This will cause a new asset to be created in the currently selected project folder as shown in figure 8.

Figure 8

Inspector

You can easily add prefabs that should be spawnable to a spawnable items asset. Simply select the spawnable items asset that you want to add the item to and click the 'Add' button on the inspector window. Figure 9 shows an empty spawnable items asset.

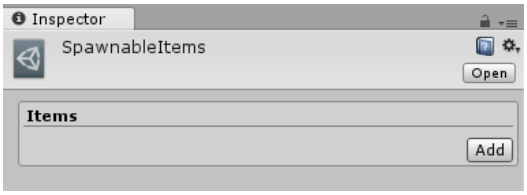

Figure 9

Once you have clicked the add button you will see that an empty slot is added to the asset. This slot has a prefab field where you should drag the prefab of your choice as well as a spawn chance slider which can be used to modify the chance value of the item being selected for spawning. A chance of 0 will mean that the item to never be spawned.

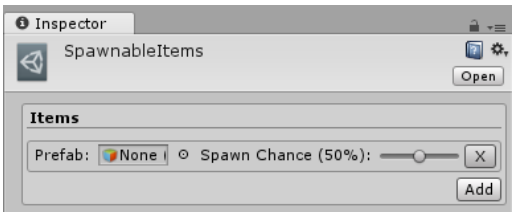

Figure 10

Each entry can easily be removed from the spawnable items asset by clicking the 'X' button to the right of the spawn chance slider.

Custom Spawnable Item providers

As of version 2.1, Ultimate Spawner now supports adding custom spawnable item providers to the spawnable items asset. A spawnable asset provider is simply a script which implements the 'UltimateSpawner.Spawning.SpawnableItemProvider' abstract class which allows you to handle the creation and destruction of spawned items manually. The spawnable item provide script inherits from Unity' ScirptableObject type meaning that you will need to create assets from your custom spawnable item provider script which you can then drag and drop into the spawnable items inspector.

Implementing a custom spawnable item provider could be useful if you want to have much more control over the creation and destruction process of spawnable items. An ideal example would be to integrate a pooling system.

The following section will demonstrate how a custom spawnable item provider can be created and added:

1. Create a new script in the project and implement the 'UltimateSpawner.Spawning.SpawnableItemProvider' abstract class. You can take a look at the default provider script 'Assets/UltimateSpawner2.0/Scripts/Spawning/PrefabSpawnableItemProvider.cs' included with Ultimate Spawner 2.0 to see how this could be done.

C# Code

```
using UnityEngine;
using UltimateSpawner.Spawning;

[CreateAssetMenu(fileName = "Custom Spawnable Asset Provider", menuName =
"Ultimate Spawner Demo/Custom Spawnable Asset Provider")]
public class CustomSpawnableAssetProvider : SpawnableItemProvider
{
    // Public
    public GameObject spawnablePrefab;

    // Properties
    public override bool IsAssigned
    {
        get { return spawnablePrefab != null; }
    }

    public override string ItemName
    {
        get { return spawnablePrefab.name; }
    }

    // Methods
    public override Object CreateSpawnableInstance(SpawnLocation spawnLocation,
SpawnRotationApplyMode applyRotation)
    {
        return Instantiate(spawnablePrefab, spawnLocation.SpawnPosition,
spawnLocation.SpawnRotation);
    }

    public override void DestroySpawnableInstance(Object spawnableInstance)
    {
        Destroy(spawnableInstance);
    }
}
```

2. Once you have implemented the abstract class then it is recommended that you add a 'CreateAssetMenu' attribute to the type declaration because we will need to create an asset from the script. This attribute will essentially add a menu item to Unity's context menu in the project window allowing you to create an asset from your provider script.
3. Create an asset from your provider script using the menu item that you added in step 2. Navigate to an appropriate folder in the project window and right click to bring up the context menu. Follow your specified menu path to create the asset and rename as desired.
4. You can now add your custom spawnable item provider to a spawnable items asset so that it can be used by the spawning system. To do this you can simply select the spawnable items asset and hit the 'Add provider' button to create a new item slot. After doing this you can then drag the spawnable item provider asset you created in step 3 to this empty slot. Alternatively you can drag your spawnable item provider asset into the drop section of the spawnable items inspector window to add to the items collection.

Spawner Nodes

Ultimate Spawner has many different types of spawners which have different use cases and configurations however all spawners are either classed as end point spawners or group based spawners. You can use these two types of spawner nodes to create complex spawning systems to suit your game's needs:

End Point Spawners

An end point spawner is a spawner node which is able to handle spawn requests directly. What this means is that the spawner will take a spawn request input either directly or from a parent spawner and handle the spawn request without delegating the work. As a result, an end point spawner as indicated by its name cannot manage child spawners and will be the last node in a spawning chain.

The **Spawn Point Spawn Area,** and **Spawn Nav Mesh** components are both end point spawners and you will typically place them around the scene where items should be spawned. You can then use other spawn nodes to create more complex spawning systems.

Group Spawners

Group spawners are the alternative spawn node type and are considered as managing elements as they will not handle spawn requests directly. Instead a group spawner will take a spawn request input either directly or from a parent spawner and will then delegate the actual spawn request to one of its child spawners based upon the configuration values of the spawner. This could either be via random selection or many other more logical decisions such as distance-based calculations.

You can use as many group spawners as needed in a chain in order to create complex spawning systems using logic and rule in order to spawn the items. The **Spawner Group** and **Spawn Trigger Volume** components are both group based spawners.

Combining Spawner Nodes

Once you are familiar with the different types of spawner nodes and how they work you can start to combine them into spawner chains in order to create complex spawning systems. In order to combine and end point spawner and a group spawner you will need to make the end point spawner a child game object via the hierarchy window:

1. Create a Spawner group game object by going to 'GameObject -> Ultimate Spawner -> Spawn Group'. A new game object will be added to the scene:

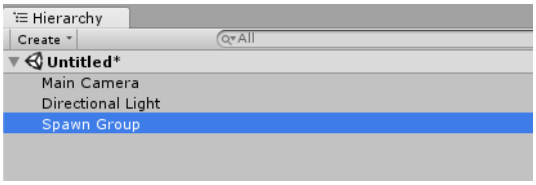

Figure 11

2. Note that the spawner group component will give us a warning in the inspector window that there are no child spawners associated. This means that the group spawner will not function until we fix that.

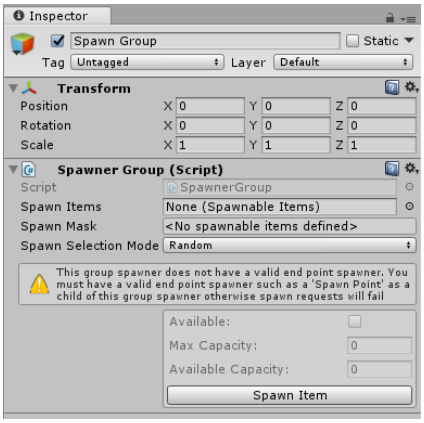

Figure 12

3. Create a Spawn Point game object as a child of the Spawner Group object by right clicking on the game object named ‘Spawn Group’ in the hierarchy window and selecting ‘Ultimate Spawner -> SpawnPoint’.

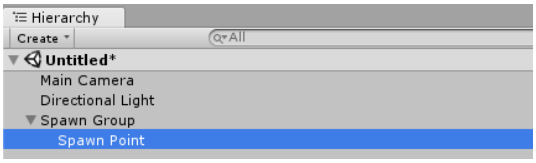

Figure 13

4. If you take a look at the inspector window for the newly created Spawn Point game object you should see that the ‘Spawn Items’ and ‘Spawn Mask’ fields are now linked to the parent group spawner game object.

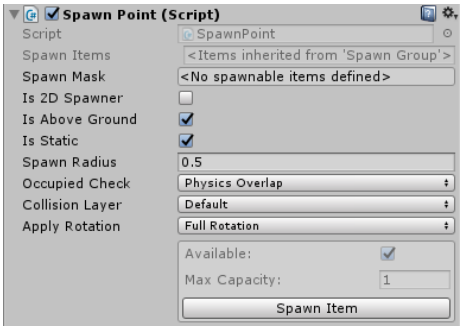

Figure 14

5. The spawn items field is now inherited from the parent spawner component so we should assign a spawnable items asset to the parent spawner to finish the setup of this basic spawn system. To do this simply select the game object named ‘Spawn Group’ from the hierarchy window and drag a suitable spawnable items asset into the ‘Spawn Items’ field. You can use a previously created asset for this or you can use the included demo asset located at ‘Assets/Ultimate Spawner/Demo/SpawnableItems.asset’.

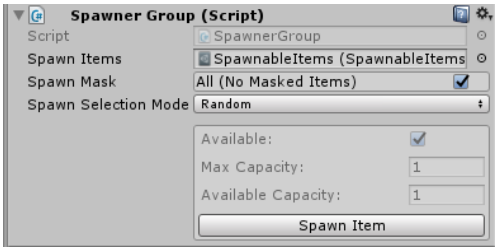

Figure 15

6. The spawn system is now ready to spawn an item so we should test it to make sure it works as expected. With the spawn group game object still selected click the 'Spawn Item' button located on the inspector window for the spawner component and see that an item is spawned into the scene at the location of the Spawn Point. We can see that the spawn request has been processed by the spawn group game object and then distributed to the spawn point game object to spawn the final item.

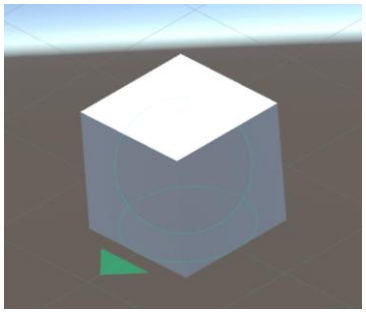

Figure 16

7. We now have a basic spawn system setup using each type of spawn node however we are not really getting the benefit out of our group spawner because we have only setup 1 child spawner. Group spawners are designed to manage many child spawners distributing the spawn requests based upon the selection mod property associated with the group spawner. We can make this spawn system more useful by adding more child spawners. To do this select the game object named 'Spawn Point' which should be a child object of the spawn group. Right click on this object in the hierarchy window and select 'Duplicate'. Alternativley you can use the shortcut keys 'Ctrl + D' which will also duplicate the object. Move this new spawn point along the X axis so that the 2 spawn points are not overlapping.

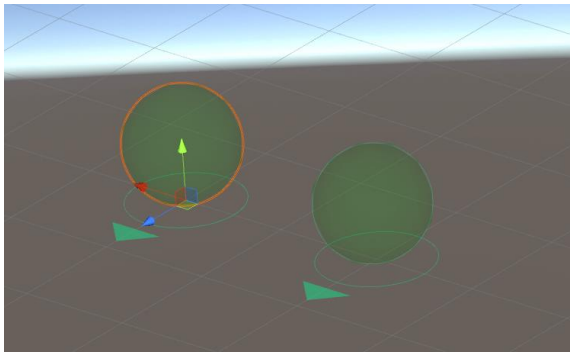

Figure 17

8. Spawn a new item as before from the spawn group inspector window and see that the spawned item is now spawned at a random point. If you spawn another item you should see that the other available spawn point is used.

As you can see this process of building spawn systems out of spawn nodes can be quite powerful and there is no limit to the nesting of nodes. You can make a spawn group object a child of another spawn group and so on to create very complex systems. You can also have both a spawn group and an end point spawner as child nodes of a spawn group. The possibilities are endless especially as every spawner has many properties that can affect the spawning logic such as distance-based spawner delegation or trigger volume interactions. Take a look at the included demo scenes to see some more complex spawning systems and how they are setup.

Spawners

The following section will cover the various spawner components included with Ultimate Spawner and the properties that you can change to configure these spawners.

Spawner Essentials

Before detailing the spawner components that are included, there are a couple of other essential concepts used by Ultimate Spawner which are quite important and these are spawn items and spawn masks:

Spawn Items

Every spawner must have access to a spawnable items asset which is used to define a collection of prefab assets that can be spawned by that spawner. These spawnable items can either be directly assigned to a spawner or they can be inherited from a parent spawner. This will be determined by the hierarchical setup of your spawners and any child spawners will inherit from the parent. See the Spawner Nodes section for more information. This means that you could potentially have one parent spawner that you assign spawnable items to and all child spawners will inherit their items making it easy to tweak the spawnable items for all spawners.

For information on creating a spawnable items asset see the section ‘Spawnable Items’.

Spawn Mask

A spawn mask is used in conjunction with a spawn items reference and allows you to specify which spawnable items can be spawned at the spawner whose mask you are editing. The mask is essentially used to disable certain spawnable items from the collection meaning that the spawner will not be able to spawn that particular item. This is useful for setting up specialized spawn points that should not be able to spawn all items listed in the spawnable items asset. An example case could be a spawn point for a boss enemy. This method of masking spawnable items means that you do not need to create a new spawnable items asset for each spawner that has customized item spawning rules.

Spawn Point

A spawn point is a type of spawner that represents a world position and rotation where a spawnable item can be spawned. A spawn point is ideal for marking the spawn/respawn location of a game character, AI or collectable item.

Create Menu

A new spawn point can be created in the scene via the menu '**GameObject -> Ultimate Spawner -> Spawn Point'**. A new spawn point will be created in the active scene which consists of a game object and an 'UltimateSpawner.Spawning.SpawnPoint' script component. Figures 18 and 19 show a spawn point in the scene view in both 3d and 2d mode.

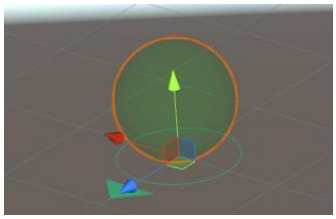

Figure 18

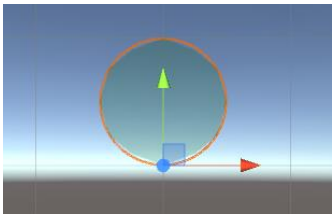

Figure 19

Gizmos

As you can see the spawn point has a number of visual elements (or gizmos) to help you set up the spawn point correctly in the scene. These gizmos vary slightly between the 2d and 3d representations.

These gizmos also provide some feedback about the state of the spawner. When the spawner is available for spawning you will see all gizmos are drawn in a shade of green to indicate that it is available. When the spawner is unable to spawn its gizmos will be displayed in a shade of red. This can be useful for debugging in game (Make sure Gizmos are enabled in the scene view window to be able to see these graphics in game).

- **Sphere volume:** A spawn point also has a spherical volume which is used to define the bounds of the spawner. If the spawn point has occupied checks enabled then it will use this bounding sphere to determine whether any physical objects are inside the bounds of the spawn point to determine whether it is able to spawn a new item or not. The spherical size of the spawn point can be modified using the 'Spawn Radius' inspector value.
- **Direction Indicator (3D only):** You will notice a triangle gizmo at the base of the spawn point which is used to indicate the facing direction. If you rotate the spawn point then this arrow will always face in the objects forward direction. Depending upon the spawner settings, spawnable items may inherit the rotation of the spawn point when that are spawned into the scene so this arrow can be used to ensure that they face the desired direction.
- **Floor Circle (3D only):** A circle outline is drawn at floor level of the spawn point to mark the ground location. Depending upon the spawn point settings the spawn volume may appear above or in line with the floor level. This is useful for making sure that you position spawn points at the correct height.

Inspector

A spawn point has a number of configurable settings that can be modified via the inspector window. Figure 20 shows all the default settings for a spawn point.

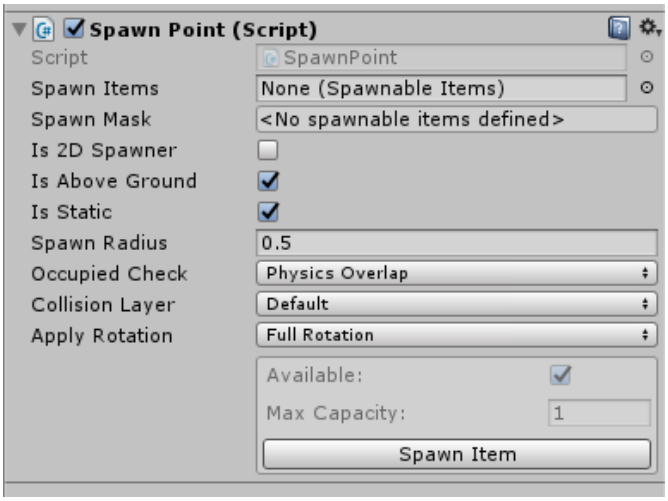

Figure 20

- **Spawn Items:** See the previous section 'Spawn Items'.
- **Spawn Mask:** See the previous section 'Spawn Mask'.
- **Is 2D Spawner:** If your game is 2D then you should enable this value which will setup the spawn point for 2D games.
- **Is Above Ground:** When enabled, the spawn location of the spawn point will be moved above ground (Y axis) by the amount of 'Spawn Radius' so that spawning inside the ground mesh can be avoided.
- **Is Static:** You should enable this when your spawn point will not be moved during gameplay. By enabling this value you will have a slight improvement in performance.
- **Spawn Radius:** The size in units of the spawn radius required to spawn items. This radius value is used to define the bounds of the spawn point which may be used to physics occupied checks.
- **Occupied Check:** The type of occupied check to use to determine whether the spawn point is capable of spawning. Options are as follows:
    - **None:** No occupied check will be performed and the spawn point will always be able to spawn items.
    - **Physics Overlap (Default):** Use the physics API to overlap a bounding volume using the spawn point settings to check for physics objects within the spawner bounds.
    - **Physics Trigger:** Use an attached trigger collider (or collider 2D for 2d games) to detect objects entering and existing the bounds of the spawner. If no valid trigger collider is attached when the game starts the spawn point will automatically create a trigger sphere collider (or circle collider for 2d games) to act as the trigger.
- **Collision Layer:** The layer that all physics occupied checks should be performed on. This value is hidden if 'Occupied Check' is set to none.
- **Apply Rotation:** Which axes of rotation should be applied to spawned items. Options are as follows:
    - **None:** Do not apply any rotation to spawned items.
    - **Full Rotation (3D Default):** Apply all axes of rotation to the spawned item

- **Z Rotation (2D Default):** Only apply rotation on the Z axis.

Spawn Info

The inspector also has some useful spawner information which is displayed in a box at the bottom of the component inspector. These values are:

- **Available:** This toggle will be checked if the spawn point is considered available and is able to spawn an item.
- **Max Capacity:** This field displays the maximum potential spawn locations that the spawn point has. A spawn point will always have a max spawn capacity of 1.
- **Spawn Item (Button):** This button will attempt to spawn an item into the scene using the current settings of the spawner. This will also work in edit mode.

Physics Triggers

As mentioned previously, when the 'Occupied Check' is set to 'Physics Trigger' the spawn point will use an attached physics trigger collider to detect physical objects inside the spawner bounds. By default, a spawn point will create a sphere collider (circle collider for 2d games) as the trigger but you are able to use other collider types also.

In order to use a trigger collider other than the default sphere (or circle) collider you will need to attach the desired collider at edit time. To do this you will simply need to attach a physics collider component such as a box collider to the same object as the spawn point script. This collider will be detected and used at runtime and you will now be able to control the shape of the spawn point volume by modifying the collider's properties.

> **Note:** *You will need to ensure that you attach the correct collider type to suit your game. For example, if you game is setup as 2D then you will need to ensure that only 2d physics colliders are used otherwise the spawn point will fall back and create an appropriate collider.*

Spawn Area

A spawn area is used to define an area of the scene where many spawnable items could be spawned. The capacity of the spawn area is decided by the size of the spawn plane along with the required spacing of each spawnable item but a spawn area is generally used to define a large number of potential spawn locations. Potential uses could be a shooting team game where each team of players has a spawn area that they are respawned from.

Create Menu

A new spawn area can be created in the scene via the menu '**GameObject -> Ultimate Spawner -> Spawn Area**. A new spawn area will be created in the active scene which consists of a game object and an 'UltimateSpawner.Spawning.SpawnArea' script component. Figures 21 and 22 show a spawn point in the scene view in both 3d and 2d mode.

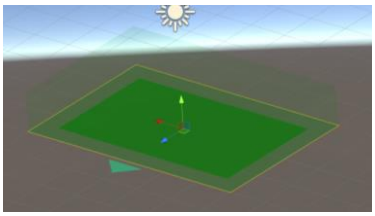

Figure 21

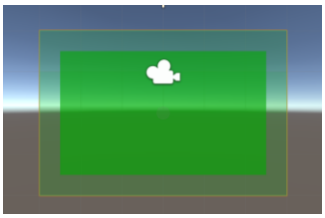

Figure 22

Gizmos

As you can see the spawn area has a number of visual elements (or gizmos) to help you set up the spawn area correctly in the scene. These gizmos vary slightly between the 2d and 3d representations.

These gizmos also provide some feedback about the state of the spawner. When the spawner is available for spawning you will see all gizmos are drawn in a shade of green to indicate that it is available. When the spawner is unable to spawn its gizmos will be displayed in a shade of red. Spawn areas can be partially able to spawn in which case the color will be somewhere between green and red to indicate the amount of availability as figure 23 shows. This can be useful for debugging in game (Make sure Gizmos are enabled in the scene view window to be able to see these graphics in game).

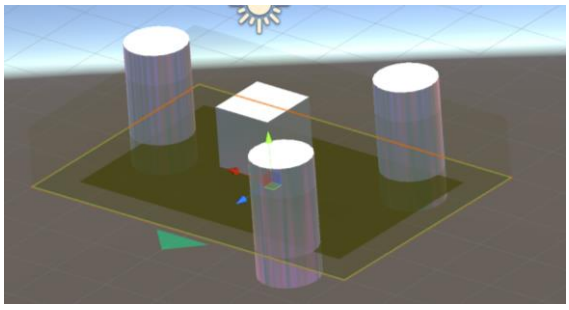

Figure 23

- **Inner Spawn Plane:** The inner spawn plane is used to show the extents that spawnable items will be spawned on the spawn area and is shown as the centre green rectangle. All spawnable items will be spawned inside this plane not taking into account the size of the item. This means that the items transform position will be inside this plane but visual elements may lie outside.
- **Outer Spawn Plane:** The outer spawn plane is used to define the extreme extents of the spawn area and is shown as the less intense green area surrounding the inner spawn plane. The distance between the inner and outer plane is used to indicate the required spacing radius of the spawnable items. If you modify the spawn radius for the area you will observe that the distance between the inner and outer plane edges are modified.
- **Enclosing Volume (3D only):** It may be difficult to see from the image but there is a faint solid box drawn around the spawn area. This box is used to mark the ceiling height of the spawn area when in 3d mode. The ceiling height determines how tall the spawn area is which is used in physics occupied checks to determine whether certain points on the spawn area are available for spawning.
- **Direction Indicator (3D only):** You will notice a triangle gizmo at the base of the spawn point which is used to indicate the facing direction. If you rotate the spawn point then this arrow will always face in the objects forward direction. Depending upon the spawner settings, spawnable items may inherit the rotation of the spawn point when that are spawned into the scene so this arrow can be used to ensure that they face the desired direction.

Inspector

A spawn area has a number of configurable settings that can be modified via the inspector window. Figure 24 shows all the default settings for a spawn area.

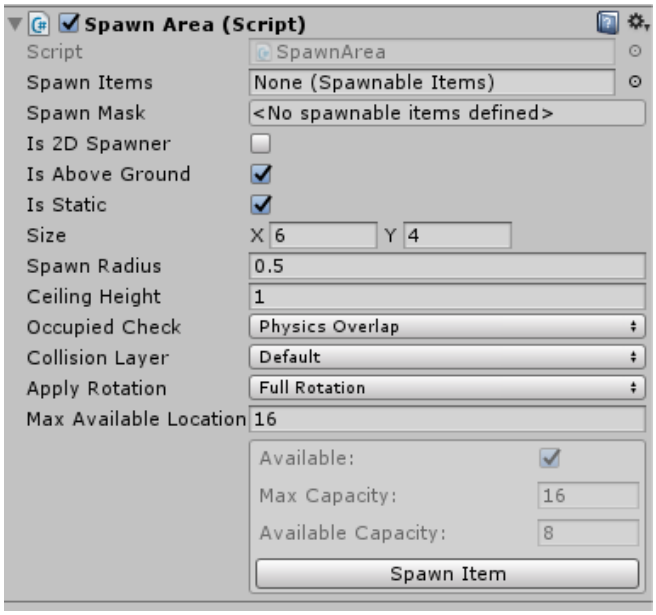

Figure 24

- **Spawn Items:** See the previous section 'Spawn Items'.
- **Spawn Mask:** See the previous section 'Spawn Mask'.
- **Is 2D Spawner:** If your game is 2D then you should enable this value which will setup the spawn area for 2D games.

- **Is Above Ground:** When enabled, the spawn locations of the spawn area will be moved above ground (Y axis) by the amount of 'Ceiling Height *div* 2' so that spawning inside the ground mesh can be avoided.
- **Is Static:** You should enable this when your spawn area will not be moved during gameplay. By enabling this value you will have a slight improvement in performance.
- **Size:** The size in units of the spawn area plane. Use in conjunction with 'Ceiling Height' (when in 3d mode) to define the overall bounds of the spawn area.
- **Spawn Radius:** The size in units of the spawn radius required to spawn items. This radius value is used to define the bounds of the spawn area which may be used to physics occupied checks.
- **Ceiling Height:** The overall height of the spawn area as used by physics occupied checks. Usually a ceiling height of 'Spawn Radius *mul* 2' will be sufficient.
- **Occupied Check:** The type of occupied check to use to determine whether the spawn area is capable of spawning. Options are as follows:
    - **None:** No occupied check will be performed and the spawn area will always be able to spawn items.
    - **Physics Overlap (Default):** Use the physics API to overlap a bounding volume using the spawn area settings to check for physics objects within the spawner bounds.
    - **Physics Trigger:** Use an attached trigger collider (or collider 2D for 2d games) to detect objects entering and existing the bounds of the spawner. If no valid trigger collider is attached when the game starts the spawn area will automatically create a trigger box collider to act as the trigger.
- **Collision Layer:** The layer that all physics occupied checks should be performed on. This value is hidden if 'Occupied Check' is set to none.
- **Apply Rotation:** Which axes of rotation should be applied to spawned items. Options are as follows:
    - **None:** Do not apply any rotation to spawned items.
    - **Full Rotation (3D Default):** Apply all axis of rotation to the spawned item.
    - **Y Rotation:** Only apply rotation on the Y axis.
    - **Z Rotation (2D Default):** Only apply rotation on the Z axis.
- **Max Available Locations:** Use this value to limit the amount of spawnable items that the spawn area can hold at one time. The physical available locations will depend upon the size of the spawn area along with the spawn radius but you can set a hard upper limit to restrict the maximum capacity of larger areas. You can set this value to -1 to set the max capacity the max physical capacity.

Spawn Info

The inspector also has some useful spawner information which is displayed in a box at the bottom of the component inspector. These values are:

- **Available:** This toggle will be checked if the spawn area is considered available and is able to spawn an item.
- **Max Capacity:** This field displays the maximum potential spawn locations that the spawn area has. This value will depend upon the size of the spawn area as well as the spawn radius used. This value will also be influenced by 'Max Available Locations'
- **Available Capacity:** This field displays the total number of available spawn locations within the spawn area. This value is calculated by using the specified 'Occupied Check' to detect physical objects within the bounds of the spawn area.

- **Spawn Item (Button):** This button will attempt to spawn an item into the scene using the current settings of the spawner. This will also work in edit mode.

Spawn Nav Mesh

A Spawn Nav Mesh is a special end point spawner that is able to work in conjunction with a navigation mesh baked into the scene. The component will use the navigation mesh to select a random spawn location in the scene which can be useful for spawning pickup items etc which should be able to spawn at any walkable location in the scene.

One important thing to note when working with Spawn Nav Mesh components is that they do not support occupied checks and as a result they will always return true when 'IsAvailable' is queried. There will also be an infinite number of possible spawn locations (represented as 999 in the inspector) due to the lack of occupied checks. You should be aware that it is possible for multiple items to spawn in close proximity causing them to overlap however this is generally a rare case, especially in larger scenes.

Create Menu

A new spawn Nav Mesh object can be created in the scene via the menu '**GameObject -> Ultimate Spawner -> Spawn Nav Mesh'**. A new spawn nav mesh will be created in the active scene which consists of a game object and an 'UltimateSpawner.Spawning.SpawnNavMesh' script component. Note that there will be gizmos rendered in the scene for this component.

Inspector

A Spawn Nav Mesh component is mostly fixed function however there are a few configurable settings which can be changed via the inspector window. Figure 25 shows all settings for the component.

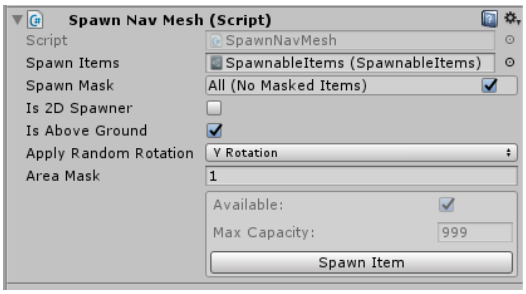

Figure 25

- **Spawn Items:** See the previous section 'Spawn Items'.
- **Spawn Mask:** See the previous section 'Spawn Mask'
- **Is 2D Spawner:** This option is inherited and is not currently used by the Spawn Nav Mesh component.
- **Is Above Ground:** When true, all spawn locations will be moved above ground height by a value of '0.5' units. This can be useful to avoid spawning items in the floor.
- **Apply Random Rotation:** Which axes or rotation should be randomly set. Options are as follows:
    - **None:** Do not apply any rotation to the spawned item. Items will have their rotation set to (0,0,0).
    - **Full:** All rotation axis will be randomly set.
    - **Y Rotation (Default):** Only the Y axis rotation will be randomly set.

    - **Z Rotation:** Only the Z axis of rotation will be randomly set
- **Area Mask:** The mask value for all Nav Mesh queries. This can be used for filtering Navigation meshes.

Spawn Group

A spawn group is a special type of spawner that is able to delegate spawn requests to child spawners which means you can create hierarchical spawner structures. The main benefit of a spawn group is that it allows you to have a single point of access for issuing spawn requests but allows for any number of child spawners to handle that request based upon the rules you define via the inspector.

You are able to register child spawners with a spawner group by simply creating another spawner and making it a child of the spawn group. The spawn group will then automatically detect the child spawner in the hierarchy and use it for spawn requests and inspector information.

Note: *You are able to chain any number of spawn groups in the hierarchy to create deep complex spawner chains. Simply attach a spawn group object as a child to another spawn group object.*

Create Menu

A new spawn group can be created in the scene via the menu '**GameObject -> Ultimate Spawner -> Spawn Group**. A new spawn group will be created in the active scene which consists of a game object and an 'UltimateSpawner.Spawning.SpawnGroup' script component. A spawn group has no scene representation.

Inspector

A spawn group has a number of configurable settings that can be modified via the inspector window. Figure 26 shows all the default settings for a spawn group.

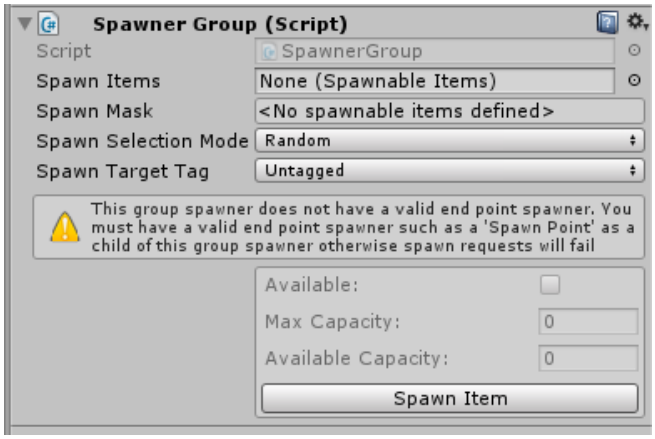

Figure 26

- **Spawn Items:** See the previous section 'Spawn Items'.
- **Spawn Mask:** See the previous section 'Spawn Mask'.
- **Spawn Selection Mode:** The mode used to select the next child spawner to handle a spawn request. Options are as follows:
    - **Random:** Select a random child spawner
    - **Sequential**: Use each spawner in turn
    - **Reverse Sequential**: Use each spawner in turn in reverse direction
    - **At Index:** Use the spawner at the specified index
    - **Nearest Target:** Use the spawner that is nearest to a Spawner Target script

    - **Nearest Target With Tag:** Use the spawner that is nearest to a Spawner Target script which has the specified tag
    - **Farthest Target:** Use the spawner that is farthest from a Spawner Target script
    - **Farthest Target With Tag:** Use the spawner that is farthest from a Spawner Target script which has the specified tag
- **Spawn Target Tag:** Used with distance based spawn selection modes. The tag of the Spawner Target to use for distance checks.
- **Spawn Selection Index:** Used with index based spawn selection mode. The index of the child spawner to use for spawning.

Spawn Info

The inspector also has some useful spawner information which is displayed in a box at the bottom of the component inspector. These values are:

- **Available:** This toggle will be checked if the spawn group is considered available and is able to spawn an item.
- **Max Capacity:** This field displays the maximum potential spawn locations that the spawn group has. This value will depend upon the number of child spawners that the group has.
- **Available Capacity:** This field displays the total number of available spawn locations within the spawn group. This value is calculated by checking the availability of child spawners.
- **Spawn Item (Button):** This button will attempt to spawn an item into the scene using the current settings of the spawner. This will also work in edit mode.

Spawn Trigger Volume

A spawn trigger volume is a type of group spawner which has a bounding physics trigger volume associated with it. This trigger volume is used to detect the presence of game objects, particularly ones with a specified tag which will then cause the spawner to be available while the object is inside the boundaries. Spawn trigger volumes are useful for breaking up areas by requiring the player of another game object to be inside the volume before it is able to spawn via its child spawners.

One important thing to mention is that a spawn trigger volume needs to have a physics collider attached the same game object. Any type of collider can be used but it must be a trigger and should be used to define the bounds of the volume. If you game is 2D then you will need to use physics 2d collider components.

Create Menu

A new spawn group can be created in the scene via the menu '**GameObject -> Ultimate Spawner -> Spawn Trigger Volume'**. A new spawn trigger volume will be created in the active scene which consists of a game object and an 'UltimateSpawner.Spawning.SpawnTriggerVolume' script component. A spawn trigger volume has no scene representation.

Inspector

A spawn trigger volume has a number of configurable settings that can be modified via the inspector window. Figure 27 shows all the default settings for a spawn trigger volume.

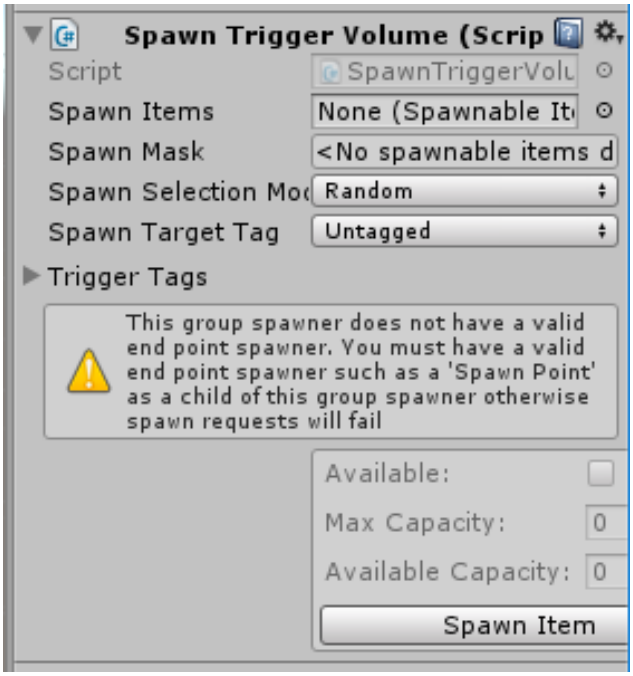

Figure 27

- **Spawn Items:** See the previous section 'Spawn Items'.
- **Spawn Mask:** See the previous section 'Spawn Mask'.
- **Spawn Selection Mode:** The mode used to select the next child spawner to handle a spawn request. Options are as follows:
    - **Random:** Select a random child spawner

    - **Sequential**: Use each spawner in turn
    - **Reverse Sequential**: Use each spawner in turn in reverse direction
    - **At Index:** Use the spawner at the specified index
    - **Nearest Target:** Use the spawner that is nearest to a Spawner Target script
    - **Nearest Target With Tag:** Use the spawner that is nearest to a Spawner Target script which has the specified tag
    - **Farthest Target:** Use the spawner that is farthest from a Spawner Target script
    - **Farthest Target With Tag:** Use the spawner that is farthest from a Spawner Target script which has the specified tag
- **Spawn Target Tag:** Used with distance based spawn selection modes. The tag of the Spawner Target to use for distance checks.
- **Target Proximity Deadzone:** Used with nearest distance based spawn selection modes. When set to a value greater than '0', this option is used to indicate a deadzone distance where any spawn locations located within this distance value will be ignored for the spawn request. IE. The spawn target entity is too close to the spawner. This deadzone can be useful for games where you want to spawn enemies near the player but not right on top of them which could make the game too difficult.
- **Spawn Selection Index:** Used with index based spawn selection mode. The index of the child spawner to use for spawning.
- **Trigger Tags:** An array of tags that will activate the trigger when a physical game object with that tag enters the volume.

Spawn Info

The inspector also has some useful spawner information which is displayed in a box at the bottom of the component inspector. These values are:

- **Available:** This toggle will be checked if the spawn trigger volume is considered available and is able to spawn an item.
- **Max Capacity:** This field displays the maximum potential spawn locations that the spawn trigger volume has. This value will depend upon the number of child spawners that the group has.
- **Available Capacity:** This field displays the total number of available spawn locations within the spawn trigger volume. This value is calculated by checking the availability of child spawners.
- **Spawn Item (Button):** This button will attempt to spawn an item into the scene using the current settings of the spawner. This will also work in edit mode.

Despawners

Ultimate Spawner version 2.2 adds the ability to despawn items in a number of different ways. Despawning simply means that the item will be removed from the game in some way or another and the actual way in which this is achieved is delegated to the spawn provider that created the item. The default spawn provider for prefabs will first check for any pooling handlers added via the global pooling events and will let the pooling handler the despawning of the item if found. If there is no pooling handler registered then Ultimate Spawner will simply destroy the item.

Ultimate Spawner has a number of despawner components which can be used to despawn an item based on different game play conditions such as collisions or triggers, as well as components for generic cases like delayed despawning.

Using Despawners

Despawners are implemented as components and can simply be added to a game object to have control over its destruction. There are a few caveats with this approach though:

1. Certain despawners such as 'DespawnOfterAmount' can only despawn items which have been created by a spawner. This is due to the fact that the alive item count needs to be tracked which is only possible by tracing spawned items.
2. Despawners are not able to despawn 'Spawner' components. This is due to the fact that spawners are treated specially as they can contain a number of despawner components which will be cloned and applied to all items created by that particular spawner. As a result, despawners that are attached to a game object with a spawner component will remain inactive until cloned onto a spawned item.

As mentioned previously, despawners are simply components which can be attached to game objects, however some despawners require certain components such as SpawnableIdentity to be present which is only added when an item is spawned. This means that they will only work if the item has been spawned by Ultimate Spawner in the first place. There are a few different ways to use despawner components in your game:

- **Despawer attached to Spawner:** It may sound strange, but one or more despawners can be added to a spawner component such as 'Spawn Point' and will not affect the spawner in any way. Instead, when the spawner creates an item via a spawn request, each attached despawner will be cloned onto the spawnable item. This means that you can create multiple spawners throughout your game level and set them up with different despawn behaviours so that items can be despawned based upon their spawn location.

  To add a despawner component to a spawner object, click the 'Add Component' button in the inspector window and navigate to the 'Ultimate Spawner' menu item where you will find a number of despawner components to add. Select a depawner component such as 'Despawn After Time' and you should find that the component does not appear to be added. This is to be expected because despawner components as displayed via the spawner instector for ease of use and organisational purposes. You should see the spawner inspector update with a 'Despawners' foldout added.

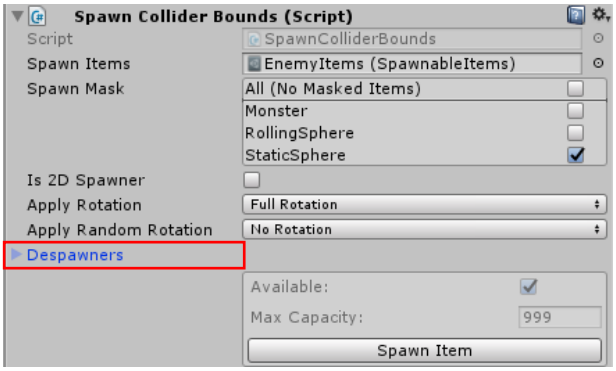

Figure 28

After expanding this foldout, you will see a list of all added despawner components where you can edit their properties, enabled or disable cloning onto spawned items and remove the despawner.

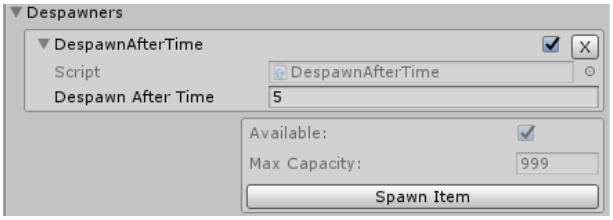

Figure 29

- All properties related to the spawner will be displayed once the particular despawner foldout is expanded.
- The checkbox at the top right of the window will enable or disable cloning. When enabled (The default option), the despawner component and all of its properties will be copied onto the spawned item. Unticking the check box will cause that particular despawner to not be cloned onto spawned items.
- You can use the 'X' button at the top right of a despawner to remove the component completely.

- **Despawner Attached to Spawnable Item:** You can also add despawners directly to prefab assets that have been registered with a spawnable items asset. This is useful for settings the despawn behaviour on a per item basis instead of a per spawner basis. Once the item has been spawned into the scene, the despawner will be activated and will despawn the item once its despawn condition has been met.
- **Despawner Attached to Non-Spawnable Item:** You can also attach some despawners to normal scene objects which are in no way related to Ultimate Spawner and they will work just as expected. There are some exceptions to this. The 'Despawn After Amount' despawner will only work with items spawned by Ultimate Spawner as it needs information about the number of alive items in order to work correctly. Take a look at the documentation section for a particular despawner to find out whether it can work on non-spawnable item objects.

Despawn After Time

The despawn after time despawner does exactly what you would expect. It will cause an item to be despawner after the specified amount of time has elapsed. You can add this despawner via the menu

Create Menu

A new despawn after time component can be added to the selected game object via the menu '**Component -> Ultimate Spawner -> Despawn After Time'.** Alternatively, you can also add this despawner via '**Add Component -> Ultimate Spawner -> Despawn After Time** from the inspector window.

Inspector

The despawn after time component only has a single property which is editiable via the inspector window:

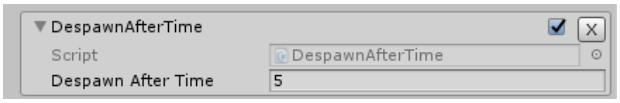

Figure 30

- **Despawn After Time:** The amount of time in seconds that the component should wait before it despawns the parent game object.

Note: *This despawner can be used with non-spawnable item objects. Ie game objects do not need to be spawned by Ultimate Spawner in order for this component to work correctly.*

Despawn After Amount

The despawn after amount despawner is intended to limit the number of a particular item in the scene at any given time. Once the number of a particular item in the scene exceeds the maximum, this despawner will cause the items that were created the longest time ago to be despawned. This component is very useful for creating automatic item recycling pools for a particular game item or effect such as bullet hole decals. By setting the maximum number of bullet holes that can be spawned, old bullet holes will be despawned to make room for new ones and you don't need to worry about too many resources in the scene at once. Couple this with pooling and you have a very complete and efficient way of creating bullet hole effect. Of course, the component can be used for many other scenarios other than bullet hole effects.

Create menu

A new despawn after amount component can be added to the selected game object via the menu '**Component -> Ultimate Spawner -> Despawn After Amount'**. Alternatively, you can also add this despawner via '**Add Component -> Ultimate Spawner -> Despawn After Amount'** from the inspector window.

Inspector

The despawn after amount component only defines a single inspector property which is modifiable:

Figure 31

- **Max Allowed Count:** The maximum number of items which can exist in the scene at any given time. Note that this limit only applies to items of the same origin. Ie They were created from the same spawnable item parent and they are essentially instances of the same prefab.

Note: *This despawner can only be used with items that were spawned by Ultimate Spawner. This is due to the need for the component to track the number of alive items of a particular type which is only available by spawning via the spawnable items system.*

Despawn On Collision

This component requires a 2D or 3D collider component to be attached to the same game object.

The despawn on collision component can be used to desapwn an item when a physics collision occurs. The component supports both 2D and 3D physics colliders but must be set to either 2D or 3D mode depending upon your game. The Unity 'CollisionEnter' event is used to trigger the despawn of either the attached object or the other colliding object depending upon the value of 'Despawn Target'.

Create menu

A new despawn on collision component can be added to the selected game object via the menu '**Component -> Ultimate Spawner -> Despawn On Collision'**. Alternatively, you can also add this despawner via '**Add Component -> Ultimate Spawner -> Despawn On Collision'** from the inspector window.

Inspector

The despawn on collision component has a number of properties that can be edited via the inspector window:

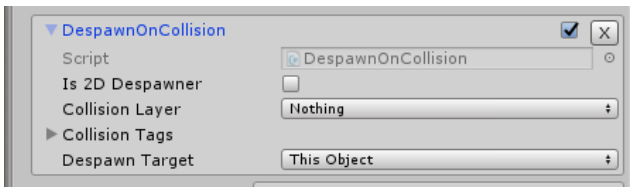

Figure 32

- **Is 2D Despawner:** Should the collision event detect 2D or 3D collision events. This value will auto detect the current active scene mode but you can also change it manually if required.
- **Collision Layer:** The layer that colliding objects should exist on in order for them to trigger a collision despawn.

- **Collision Tags:** A collection of tags which can be used to filter out game objects which can and cannot cause collision despawns. If one or more tags are specified, the colliding object must be tagged with one of the collision tags in order to cause a collision despawn.
- **Despawn Target:** You can use this property to specify whether the game object with the despawn script should be despawned when the condition is met or whether the other colliding object should be despawned. Options are:
    - **This Object:** The game object parent of the despawner on collision script will be despawned when a collision is detected.
    - **Other Object:** The other game object that collided with the despawner game object will be despwned.

Note: *This despawner can be used with non-spawnable item objects. Ie game objects do not need to be spawned by Ultimate Spawner in order for this component to work correctly.*

Despawn On Trigger

This component requires a 2D or 3D trigger collider component to be attached to the same game object.

The despawn on trigger component is much like the despawn on collision component but will detect trigger events as opposed to collision events. The despawner uses the Unity 'TriggerEnter' events to detect when a physical object enters the attached trigger collider.

Create menu

A new despawn on trigger component can be added to the selected game object via the menu '**Component -> Ultimate Spawner -> Despawn On Trigger'**. Alternatively, you can also add this despawner via '**Add Component -> Ultimate Spawner -> Despawn On Trigger'** from the inspector window.

Inspector

The despawn on trigger component has a number of properties which can be editied via the inspector window:

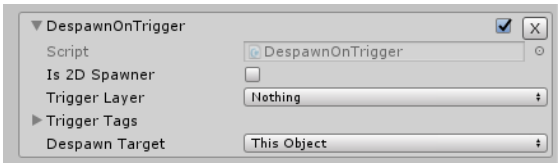

Figure 33

- **Is 2D Despawner:** Should the trigger event detect 2D or 3D collision events. This value will auto detect the current active scene mode but you can also change it manually if required.
- **Collision Layer:** The layer that triggering objects should exist on in order for them to cause a trigger despawn.
- **Collision Tags:** A collection of tags which can be used to filter out game objects which can and cannot cause trigger despawns. If one or more tags are specified, the colliding object must be tagged with one of the collision tags in order to cause a trigger despawn.

- **Despawn Target:** You can use this property to specify whether the game object with the despawn script should be despawned when the condition is met or whether the other triggering object should be despawned. Options are:
    - **This Object:** The game object parent of the despawner on collision script will be despawned when a collision is detected.
    - **Other Object:** The other game object that collided with the despawner game object will be despwned.

Note: *This despawner can be used with non-spawnable item objects. Ie game objects do not need to be spawned by Ultimate Spawner in order for this component to work correctly.*

Despawn On Event

A despawn on event component can be linked up with Unity events or animation events in order to cause an item to e despawned when a certain game play event or animation event occurs. The component has a single public method 'DespawnItem' which can be hooked up to Unity events or animation events via the inspector window or animation window.

Create menu

A new despawn on event component can be added to the selected game object via the menu '**Component -> Ultimate Spawner -> Despawn On Event'**. Alternatively, you can also add this despawner via '**Add Component -> Ultimate Spawner -> Despawn On Event'** from the inspector window.

Despawn Distance

A despawn distance component is used to depsawn an item when certain distances conditions are met. Much like distance based spawners, the distance despawner used SpawnerTargets as objects that are capable or being used for distance calculations. You should make sure that atleast one object in your scene has a SpawnerTarget component attached otherwise the despawner will not work as expected. Generally, this script will be attached to player objects or similar to base the spawning systems around their movements.

Create menu

A new despawn distance component can be added to the selected game object via the menu '**Component -> Ultimate Spawner -> Despawn Distance**. Alternatively, you can also add this despawner via '**Add Component -> Ultimate Spawner -> Despawn Distance** from the inspector window.

Inspector

The despawn distance component has a number of properties which can be editied via the inspector window:

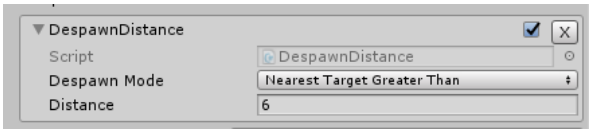

Figure 34

- **Despawn Mode:** The distance based condition used to despawn the item. Possible options are:
    - **NearestTargetGreaterThan:** The despawner will measure distance to the nearest available SpawerTarget and despawn if that distance is greater than the specified distance value.
    - **NearestTargetLessThan:** The despawner will measure distance to the nearest available SpawnerTarget and despawn if that distance is less that the specified distance value.
    - **NearestTargetWithTagGreaterThan:** The despawner will measure distance to the nearest available SpawnerTarget with the specified tag value. If the distance is greater than the specified distance, then the item will despawn.
    - **NearestTargetWithTagLessThan:** The despawner will measure distance to the nearest available SpawnerTarget with the specified tag value. If the distance is less than the specified distance, then the item will despawn.
    - **FurthestTargetGreaterThan:** The despawner will measure distance to the furthest available SpawnerTarget and despawn if that distance is greater than the specified distance value.
    - **FurthestTargetLessThan:** The despawner will measure distance to the furthest available SpawnerTarget and despawn if that distance is less than the specified distance value.
    - **FurthestTargetWithTagGreaterThan:** The despawner will measure distance to the furthest available SpawnerTarget with the specified tag value. If the distance is greater than the specified distance, then the item will despawn.
    - **FurthestTargetWithTagLessThan:** The despawner will measure distance to the furthest SpawnerTarget with the specified tag value. If the distance is less than the specified distance, then the item will despawn.
- **Distance:** A distance value in units used in conjunction with the DespawnMode to create a distance based despawn condition.
- **Despawner Target Tag (Visible only when a tag based Depsawn Mode is selected):** A tag value used to find specific SpawnerTarget objects in the scene such as players or similar. This value is only used in conjunction with Despawn Modes that have 'WithTag'.

> **Note:** *This despawner can be used with non-spawnable item objects. Ie game objects do not need to be spawned by Ultimate Spawner in order for this component to work correctly.*

Despawn From Code

Ultimate Spawner also provides despawn functionality from code so that you can despawn an item immediately or when a condition is met. The following static methods can be used to despawn items:

- **UltimateSpawner.Despawn(Object):** You can call this method on any Unity object to attempt to despawn it immediately. The object should be a game object or a component otherwise the despawn request will do nothing. If the object was spawned via Ultimate Spawner, the correct spawnable item provider will be used to despawn the item so that the object can be correctly destroyed or pooled depending upon the provider.
- **UltimateSpawner.Despawn(SpawnableIdentity):** You can call this method if you have a reference to a 'SpawnableIddentity' component and want to destroy its parent game object.
- **UltimateSpawner.DespawnAfterTime(GameObject, float):** You can call this method to despawn the target object once the specified amount of time in seconds has passed. Internally this method will add and configure a despawn after time component to handle the object despawning.
- **UltimateSpawner.DespawnAfterAmount(GameObject, int):** You can call this method to despawn the target object or objects of the same item type when the specified number of alive items in the scene is exceeded. Internalls this method will add and configure a despawn after amount component to handle the despawn behaviour.
- **UltimateSpawner.DespawnOnCollision(GameObject, DespawnTarget, bool, LayerMask, string[]):** You can use this method to cause the target object to be despawned when a collision event occurs. The target object must have a collider component attached in order for the despawer to have any effect. Internally this method will add and configure a despawn on collision component to handle the despawning of the object.
- **UltimateSpawner.DespawnOnTrigger(GameObject, DespawnTarget, bool, LayerMask, string[]):** Much like the above method, you can use this method to cause the target object to be despawned when a trigger events occurs. The target object must have a collider component attached and configured as a trigger in order to work as expected. Internally this method will add and configure a despawn on trigger component to handle the despawning of the object.
- **UltimateSpawner.DespawnOnEvent(GameObject, UnityEvent):** You can use this method to add a despawn event listener for the specified event on the target object. When the specified Unity event is invoked, the target object will be despawned. Internally this method will add and configure a despawn on event component to handle the despawning when the event occurs.
- **UltimateSpawner.DespawnDistance(GameObject, DespawnDistanceMode, float, string):** You can use this method to cause an item or object to be despawned when the specified distance based condition is met. Internally this method will add and configure a despawn distance component to handle the despawning of the object. Note that there must be atleast one SpawnerTarget object in the scene for distance based despawning to work correctly.

Spawner Targets

If you are using a Spawn Group spawner to distribute spawn request to child spawners then you will be able to select a number of distance based modes from the 'Spawn Selection Mode' inspector option. These distance-based checks allow the spawn group to select the next child spawner by performing distance calculations between the child spawners position and a particular targets position.

Spawner Targets can be used to mark a particular object as a target for these distance based checks. A Spawner Target is just a script component that can be attached to a game object which will be detected automatically by Ultimate Replay once the object has been added to the scene. This can be useful for games that have large scenes where enemies should always be spawned as near as possible to increase the challenge level.

In order for an object to become a target for these distance check you will need to attach an 'UltimateSpawner.Spawning.SpawnerTarget' script to the game object which will be detected by the spawning system at runtime.

Note: *It is OK to attach a Spawner Target to a prefab that will be instantiated when the game starts. You will receive a warning on the Spawner Group component if a distance based check is selected but no Spawner Targets could be found in the scene. If you know a prefab with the Spawner Target script will be instantiated into the scene then you can simply ignore this warning.*

Spawn Controllers

Spawn controllers are used to create spawning patterns and behaviours for your game so that items are spawned at the assigned spawner(s). Controllers are intended to issue spawn requests to spawners in a controlled fashion and each controller will achieve this using different rules or logic to offer a wide variety of behaviours.

Infinite Controller

An infinite controller is probably the most useful built in spawn controller which is highly customizable and able to create a variety of spawning behaviours. The aim of the infinite controller is to issue spawn requests at regular intervals until certain conditions are met, such as total number of items spawned or time elapsed etc. This makes it perfect for endless battle scenarios or fixed battle scenes where the player must defeat a certain number of enemies to continue. It is also useful for continuously spawning collectable items throughout the scene, especially with its minimum and maximum concurrent spawned items limits.

Create Menu

A new infinite spawn controller can be created in the scene via the menu **'GameObject -> Ultimate Spawner -> Infinite Controller'**. A new event controller object will be created in the active scene which consists of a game object and an 'UltimateSpawner.InfiniteSpawnController script component. An infinite controller has no scene representation.

Inspector

The infinite controller component has many configurable properties as already mentioned and they can be edited via the inspector window in order to customize the spawning behaviour.

- **Spawner:** The root spawner reference that all spawn requests will be issued to. Any type of spawner can be assigned to this slot in order to handle the spawn requests including group based spawners.
- **Play On Start:** When enabled, the controller will automatically begin its spawning sequence when the game starts. The spawning sequence must be running in order for the physics events to be registered and handled by the controller.
- **Maximum Spawn Count:** The maximum number of concurrent items that can exists in the scene at any time. If the number of alive spawned items equals this value then subsequent trigger events will be queued until on or more items are removed from the scene. You can set this value to '-1' in order to remove the upper limit checks.
- **Minimum Spawn Count:** The minimum number of concurrent items that should be spawned into the scene at any time. Note that this value is only a target and it may not be practically feasible to spawn the minimum number of items due to spawn points being occupied or low spawning intervals.
- **Spawn Delay:** The amount of time in seconds that must pass before another spawn request can be sent.
- **Spawn Delay Randomness:** Used in conjunction with 'Spawn Delay'. The amount of time in seconds used to randomize the spawning delay. This time value is used as an upper limit where a random value will be generated between 0 and this value. The resulting value will

then be added to the 'Spawn Delay' value to generate a random delay value before the next spawn request can be sent.

- **Stop After:** A condition used to automatically end the infinite controllers spawning sequence when it is met.
    - **Don't Stop:** The infinite controller will not stop spawning unless manually instructed via the scripting API.
    - **Time Elapsed:** The infinite controller will stop spawning after the specified amount of time has passed. The time value can be specified via the 'Stop After Time' property.
    - **Spawnable Count:** The infinite controller will stop spawning when the desired number of items have been spawned. The number of items can be specified via the 'Stop After Spawn Count' property.
- **Stop After Time:** Used in conjunction with 'Time Elapsed' stop condition. The amount of time in seconds that the controller should continue spawning for.
- **Stop After Spawn Count:** Used in conjunction with 'Spawnable Count' stop condition. The number of enemies that should be spawned before the infinite controller stops spawning.

Trigger Controller

A trigger controller can be used to issue spawn requests to spawners when a physics collision event occurs. Trigger controllers must have a suitable 2D or 3D physics collider component attached to the game object and can be configured to issue spawn requests when collision enter or exit events take place. A typical use case for this controller would be spawning a boss enemy when the player enters a certain area designated by a trigger collider.

Create Menu

A new trigger controller can be created in the scene via the menu **'GameObject -> Ultimate Spawner -> Trigger Controller'**. A new trigger controller object will be created in the active scene which consists of a game object and an 'UltimateSpawner.TriggerSpawnController' script component. A trigger controller has no scene representation.

Inspector

The trigger controller component has a number of configurable properties which can be modified via the inspector window in order to change the spawning behaviour.

- **Spawner:** The root spawner reference that all spawn requests will be issued to. Any type of spawner can be assigned to this slot in order to handle the spawn requests including group based spawners.
- **Play On Start:** When enabled, the controller will automatically begin its spawning sequence when the game starts. The spawning sequence must be running in order for the physics events to be registered and handled by the controller.
- **Maximum Spawn Count:** The maximum number of concurrent items that can exists in the scene at any time. If the number of alive spawned items equals this value then subsequent trigger events will be queued until on or more items are removed from the scene. You can set this value to '-1' in order to remove the upper limit checks.
- **Trigger Once:** When enabled, the trigger controller will only be able to accept a single trigger event meaning that only 1 item can ever be spawned by the controller. After 1 item is spawned, all subsequent trigger events will be ignored.

- **Trigger Mode:** The mode used to determine when the spawn request should be sent to the assigned spawner. Options are:
    - **On Enter:** Send the spawn request when a physical object enters the trigger collider bounds.
    - **On Exit:** Send the spawn request when a physical object leaves the trigger collider bounds.
- **Trigger Tags:** A collection of tags which will activate the trigger controller. When one or more tags are specified the trigger controller will require that the colliding object has one of these tags associated with it otherwise the collision event will be ignored. This can be useful for ensuring that only certain game objects can activate the trigger controller.
- **Controller Events:** A number of Unity events which are triggered at certain points in the controllers spawning sequence.
    - **On Item Spawned (Transform):** Called when a new item has been spawned as a result of a trigger event. This event accepts a Transform argument of the spawned item.
    - **On Item Despawned (Transform):** Called when an item previously spawned by the controller has been destroyed. This event accepts a Transform argument of the despawned item.
    - **On Start:** Called when the spawn controller has started its spawn routine. When 'Play On Start' is enabled, this event will be invoked when the game starts.
    - **On Pause:** Called when the spawn controller has been paused.
    - **On Resume:** Called when the spawn controller has resumed from a paused state.
    - **On End:** Called when the spawn controller has ended its spawning sequence. This could be as a result of manually stopping the controller via script or by the controller reaching an automatic ending condition.
    - **On Spawn Trigger:** Called when the trigger controller has received a valid physical trigger event and will send an item spawn request to the assigned spawner.

Troubleshooting

SYMPTOMS	POSSIBLE SOLUTIONS
NO ITEMS SPAWNING	Ensure that the controller is active. You can enable the 'Play On Start' option to force the controller to run on start-up.
	If 'Trigger Once' is enabled, ensure that the controller has not already spawned an item meaning that any subsequent spawn request will be ignored.
	If 'Maximum Spawn Count' is not set to '-1', ensure that the controller has not already reached the spawn limit.
	Check that the triggering object has a 2D or 3D collider and corresponding rigid body attached.
	Check that the triggering object is assigned a tag that is registered in the controller's trigger tags.

Event Controller

An event controller is a useful spawn controller that will issue spawn requests when a specified Unity behaviour event such as 'Awake' is called. This can be useful for pre-spawning a number of items when the game starts of when other game objects are destroyed.

Create Menu

A new event spawn controller can be created in the scene via the menu **'GameObject -> Ultimate Spawner -> Event Controller'**. A new event controller object will be created in the active scene which consists of a game object and an 'UltimateSpawner.EventSpawnController script component. An event controller has no scene representation.

Inspector

The event controller component has a number of configurable properties which can be modified via the inspector window in order to change the spawning behaviour.

- **Spawner:** The root spawner reference that all spawn requests will be issued to. Any type of spawner can be assigned to this slot in order to handle the spawn requests including group based spawners.
- **Play On Start:** When enabled, the controller will automatically begin its spawning sequence when the game starts. The spawning sequence must be running in order for the Unity events to be registered and handled by the controller.
- **Maximum Spawn Count:** The maximum number of concurrent items that can exists in the scene at any time. If the number of alive spawned items equals this value then subsequent trigger events will be queued until on or more items are removed from the scene. You can set this value to '-1' in order to remove the upper limit checks.
- **Spawn Event:** The Unity behaviour event used to trigger the spawn requests.
    - **Awake:** The controller will issue spawn requests when the Unity 'Awake' method is called on the event controller script.
    - **Start:** The controller will issue spawn requests when the Unity 'Start' method is called on the event controller script.
    - **On Enable:** The controller will issue spawn requests when the Unity 'OnEnable' method is called on the event controller script.
    - **On Disable:** The controller will issue spawn requests when the Unity 'OnDisable' method is called on the event controller script.
    - **On Destroy:** The controller will issue spawn requests when the Unity 'OnDestroy' method is called on the event controller script.
- **Spawn Amount:** The number of spawn requests that will be sent when the select spawn event fires. All spawn requests will be dispatched within a single frame unless one ore more associated spawners are not available in which case the requests will be queued.
- **Spawn Delay:** The amount of time in seconds that the controller will wait before dispatching all spawn requests to the handling spawner.
- **Controller Events:** A number of Unity events which are triggered at certain points in the controllers spawning sequence.
    - **On Item Spawned (Transform):** Called when a new item has been spawned as a result of a behaviour event. This event accepts a Transform argument of the spawned item.

- **On Item Despawned (Transform):** Called when an item previously spawned by the controller has been destroyed. This event accepts a Transform argument of the despawned item.
- **On Start:** Called when the spawn controller has started its spawn routine. When 'Play On Start' is enabled, this event will be invoked when the game starts.
- **On Pause:** Called when the spawn controller has been paused.
- **On Resume:** Called when the spawn controller has resumed from a paused state.
- **On End:** Called when the spawn controller has ended its spawning sequence. This could be as a result of manually stopping the controller via script or by the controller reaching an automatic ending condition.
- **On Spawn Trigger:** Called when the event controller has received the configured Unity behaviour event and will send an item spawn request to the assigned spawner.

Pooling Support

If you need to spawn lots of the same items which will continuously be destroyed then you will likely want to use some sort of pooling system to avoid the performance impact of instantiation calls. Ultimate Spawner makes it very easy to take over the instantiation calls so that pooling support can be added using your favourite pooling solution.

Every instantiate call that Ultimate Spawner makes will be handled by the static method 'UltimateSpawner.UltimateSpawnerInstantiate(Object, Vector3, Quaternion)' by default. You can easily change the instantiate method used by Ultimate Replay by assigning a custom instantiate method to the static variable 'UltimateSpawner.OnUltimateSpawnerInstantiate'. This variable is of type 'Func<Object, Vector3, Quaternion, Object>' meaning that you can assign a method to it like so:

C# Code

```
using UnityEngine;
using UltimateSpawner;

class Example : MonoBehaviour
{
    void Awake()
    {
        // Hook up the handler
        UltimateSpawner.OnUltimateSpawnerInstantiate =
HandleSpawnerInstanitate;
    }

    Object HandleSpawnerInstanitate(Object prefab, Vector3 position,
Quaternion rotation)
    {
        // Implement pooling here using your favourite pooling
system
        // return SomePoolingSystem.GetPooledInstance(prefab, ...);

        // Default instantiate example
        return Object.Instantiate(prefab, position, rotation);
    }
```

Once you have assigned a method to this handler then Ultimate Spawner will attempt to use it for all instantiate calls. Note that if the assigned handler throws an exception or returns an invalid object the system will fall back to the default instantiate handler to avoid error.

Ultimate Spawner will only ever create instances of a prefab and it will never destroy any instances so an 'OnDestroy' pooling handler is not required.

Need more control?

As of version 2.1, Ultimate Spawner 2.0 now allows custom spawnable item providers to be created which will allow you to register a script with a spawnable items asset which is then responsible for the creation and destruction of that particular item. Since you will have full control over item creation, you could integrate any pooling solution required or even create item from other sources

such as addressable assets. Take a look at the previous **Spawnable Items - Custom Spawnable item provider** section for more details.

Report a Bug

At Trivial Interactive we test our assets thoroughly to ensure that they are fit for purpose and ready for use in games but it is often inevitable that a bug may sneak into a release version and only expose its self under a strict set of conditions.

If you feel you have exposed a bug within the asset and want to get it fixed then please let us know and we will do our best to resolve it. We would ask that you describe the scenario in which the bug occurs along with instructions on how to reproduce the bug so that we have the best possible chance of resolving the issue and releasing a patch update.

http://trivialinteractive.co.uk/bug-report/

Request a feature

If you feel that Ultimate Spawner should contain a feature that is not currently incorporated then you can request to have it added into the next release. If there is enough demand for a specific feature then we will do our best to add it into a future version. Please note, there are some features that are not possible to implement due some limitations of Unity but a workaround may be possible.

http://trivialinteractive.co.uk/feature-request/

Contact Us

Feel free to contact us if you are having trouble with the asset and need assistance. Contact can either be made by the contact options on the asset store or buy the link below.

Please attempt to describe the problem as best you can so we can fully understand the issue you are facing and help you come to a resolution. Help us to help you :-)

http://trivialinteractive.co.uk/contact-us/