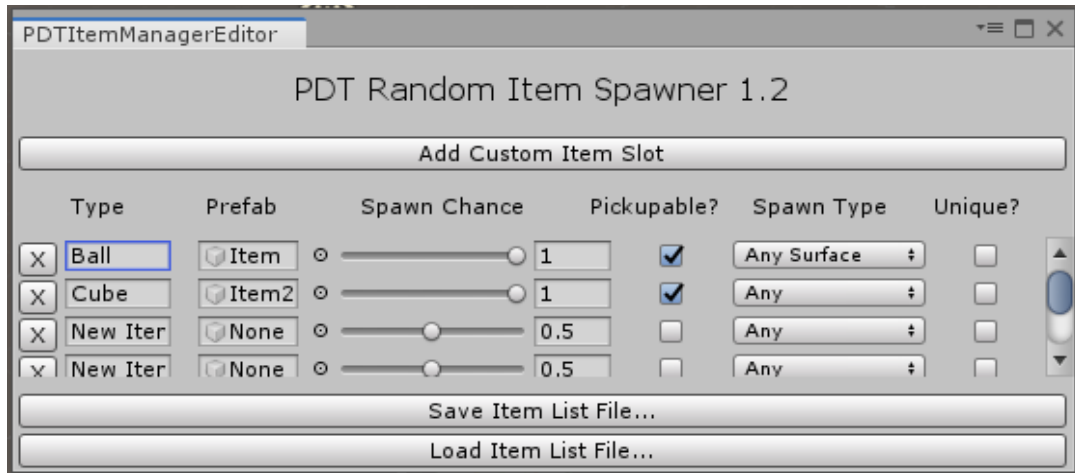


# PDT Item Spawner 1.2

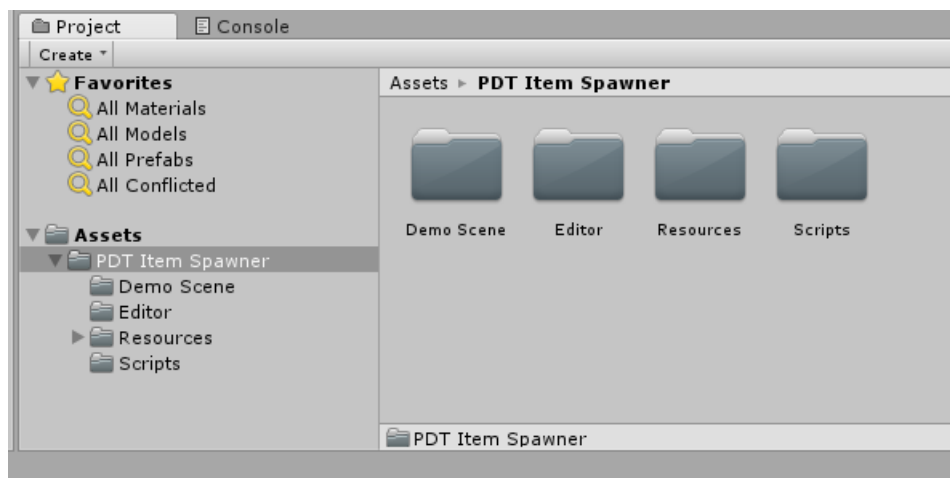
## General Usage Tutorial



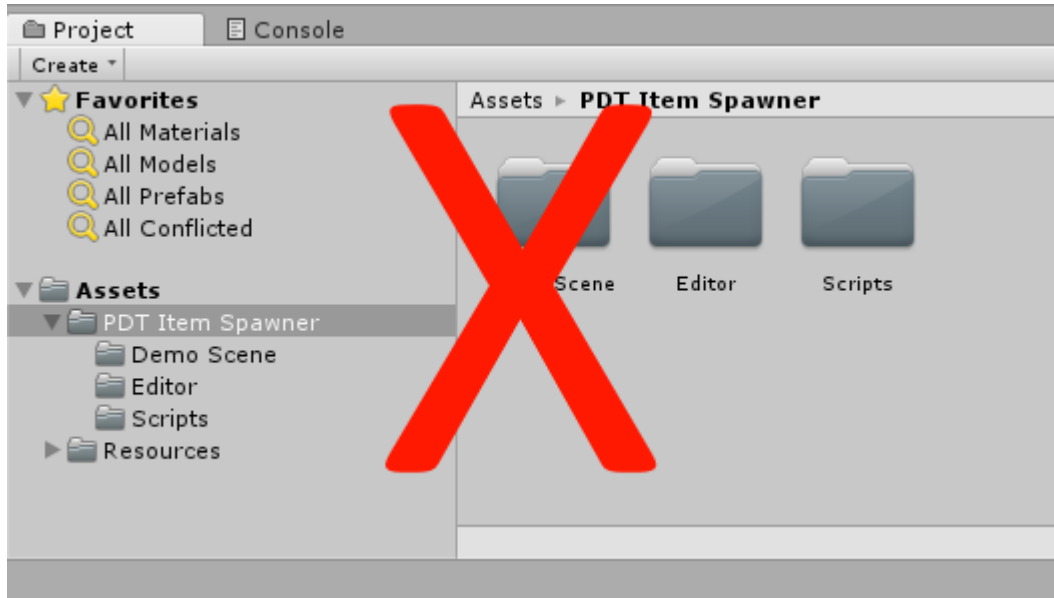
Welcome to PDT Item Spawner! This tutorial will guide you through the initial “From Scratch” setup of the asset, lay down a couple guidelines and help you use this tool to its full potential.

Without any delay, lets get started!

First of all, you want to import the asset into your project. For the sake of this tutorial, I will be using an entirely new unity project.

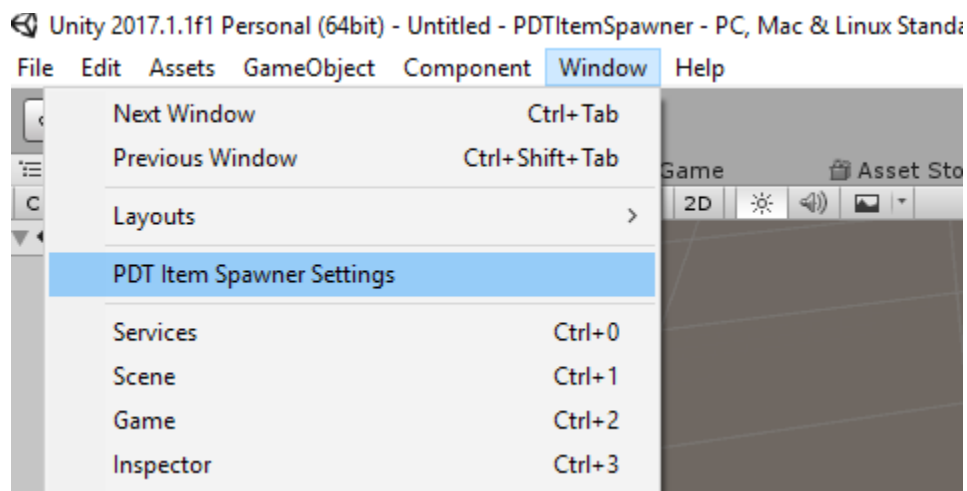


~~From this point you need to drag the “Resources” folder within “PDT Item Spawner” and drop it into your project’s main “Assets” folder. If that folder already exists, simply merge the contents.~~

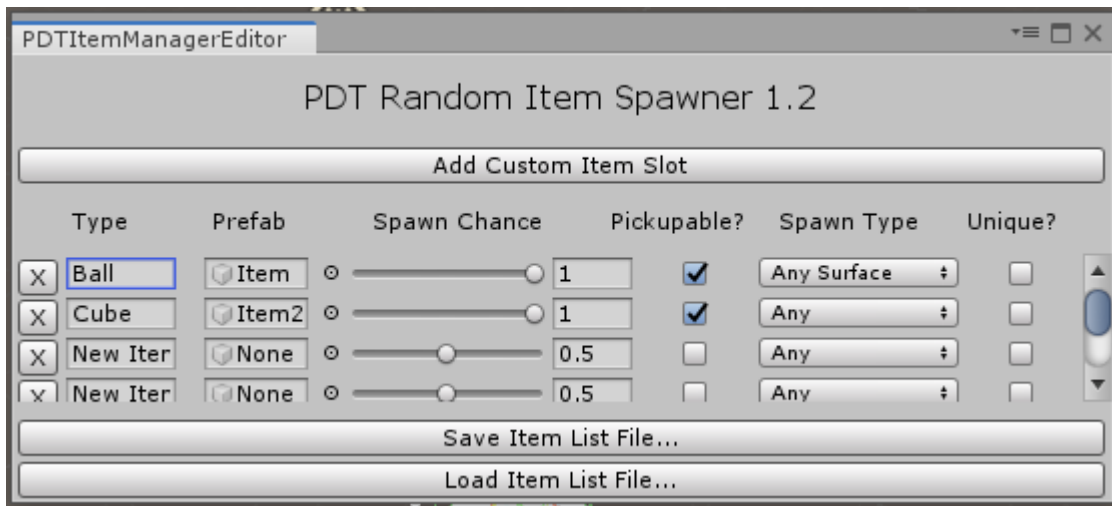


**This step is NO LONGER NEEDED. The Resources folder must be left inside or put back inside of the PDT Item Spawner folder, as it comes by default, to avoid any issues with future versions.**

You can now open the new GUI located under the Window menu.

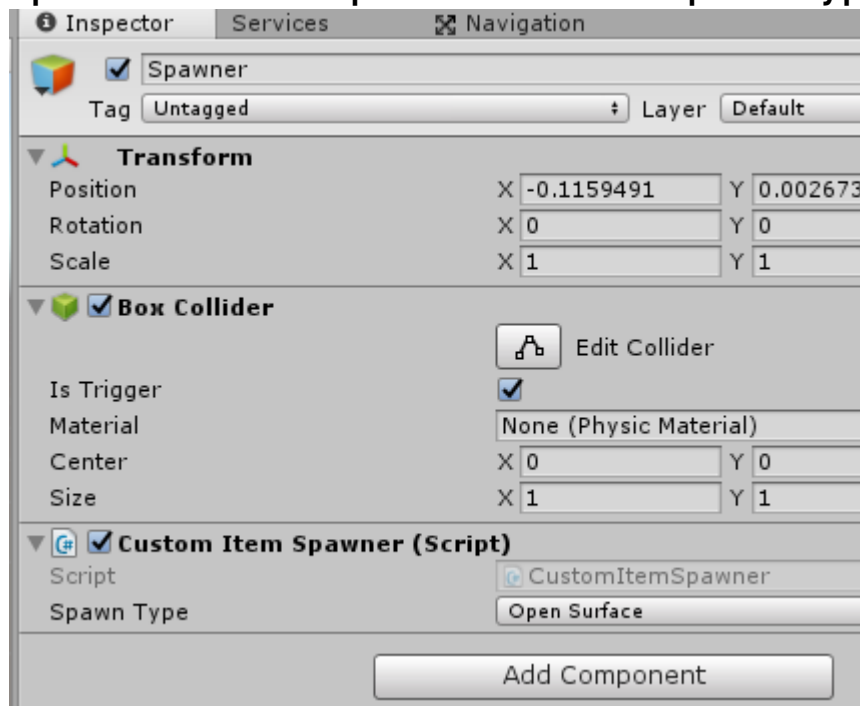


And here we have the working item spawner GUI.



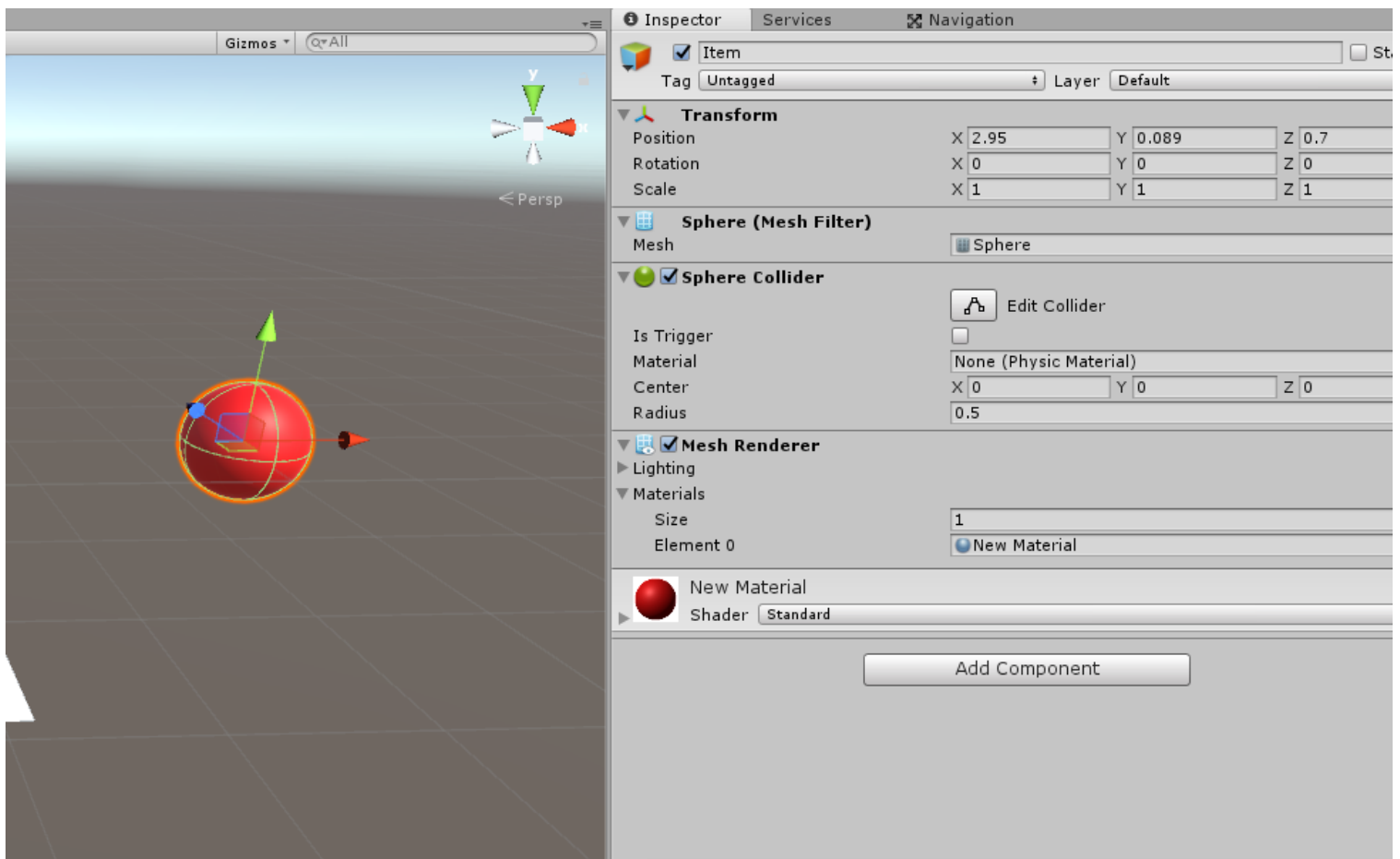
Hit the “X” button next to any existing item slots to delete them since we are starting from scratch. Make sure to always hit Save before closing this GUI to keep changes. We'll come back to this GUI when comes the time to spawn prefabs, but for now we need to set up a couple objects in our scene.

The first object is our item spawner. Create an empty gameobject with a collider set to “IsTrigger” and link the “Custom Item Spawner” script to it. Select “Open Surface” as Spawn Type.



Brief explanation : Box Collider is essentially the “range” which this spawner allows the item to be picked up. The “Spawn Type” variable is used so you can prevent particular items from spawning on some spawns. More on this later.

Right now we need an object to spawn on our new item spawner.  
For this purpose, any object and most components will work.

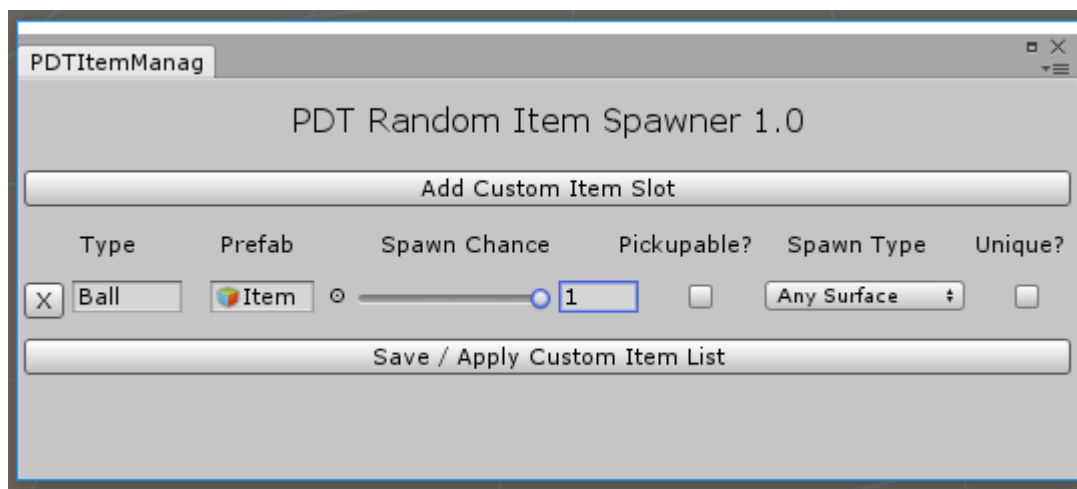


As you can see I have created this simple red sphere as my item.

It is now time to set up this object using the GUI. Again, it can be found under the “Window” context menu.

First, create a new item slot using the “Add Custom Item Slot” button. The “Type” is an arbitrary and unique name you must give each item so that they can be told apart from one another when comes the time to pick the item up and execute the appropriate Pickup actions in scripts.

I named that item “Ball”.



Next, link your created prefab to the slot under the “Prefab” setting.

The “Spawn Chance” slider is used to make rare or common items using a single slider. Since this is a floating point number from 0 to 1, a setting of 1 means your item will spawn 100% of the time.

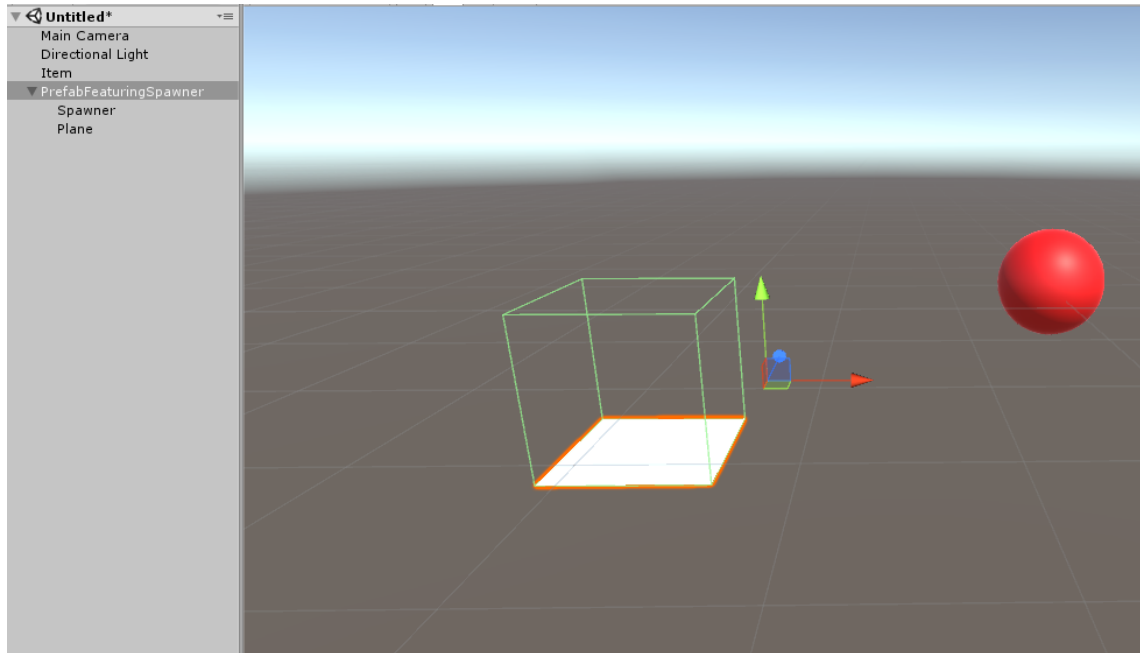
I have also set the spawn type to “Any Surface” to show that this item spawns on our “Open Surface” type spawner.

The “Pickupable?” and “Unique?” toggles will be demonstrated in a little bit. For now lets see how this item will do.

Before closing the UI make sure you hit “Save / Apply Custom Item

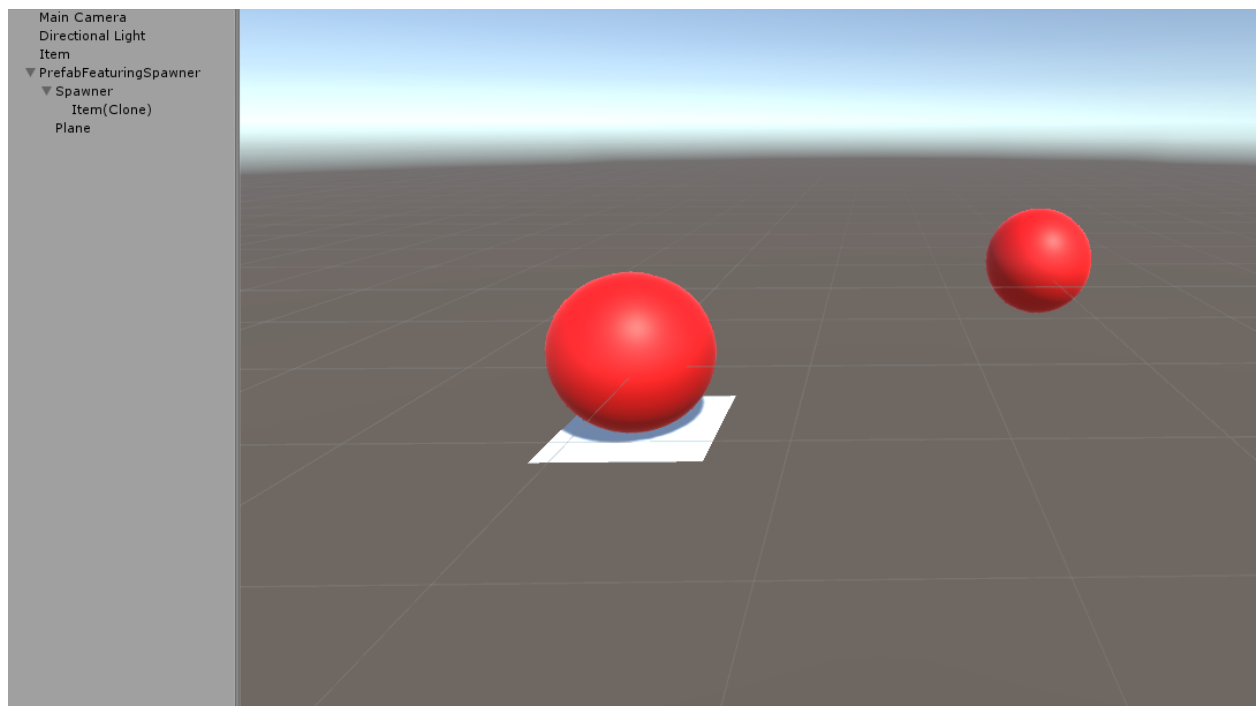
List” to save your current changes to the item list.

Here is our scene before the item spawned. I have placed a plane object underneath the spawner so we can easily see empty spawners.



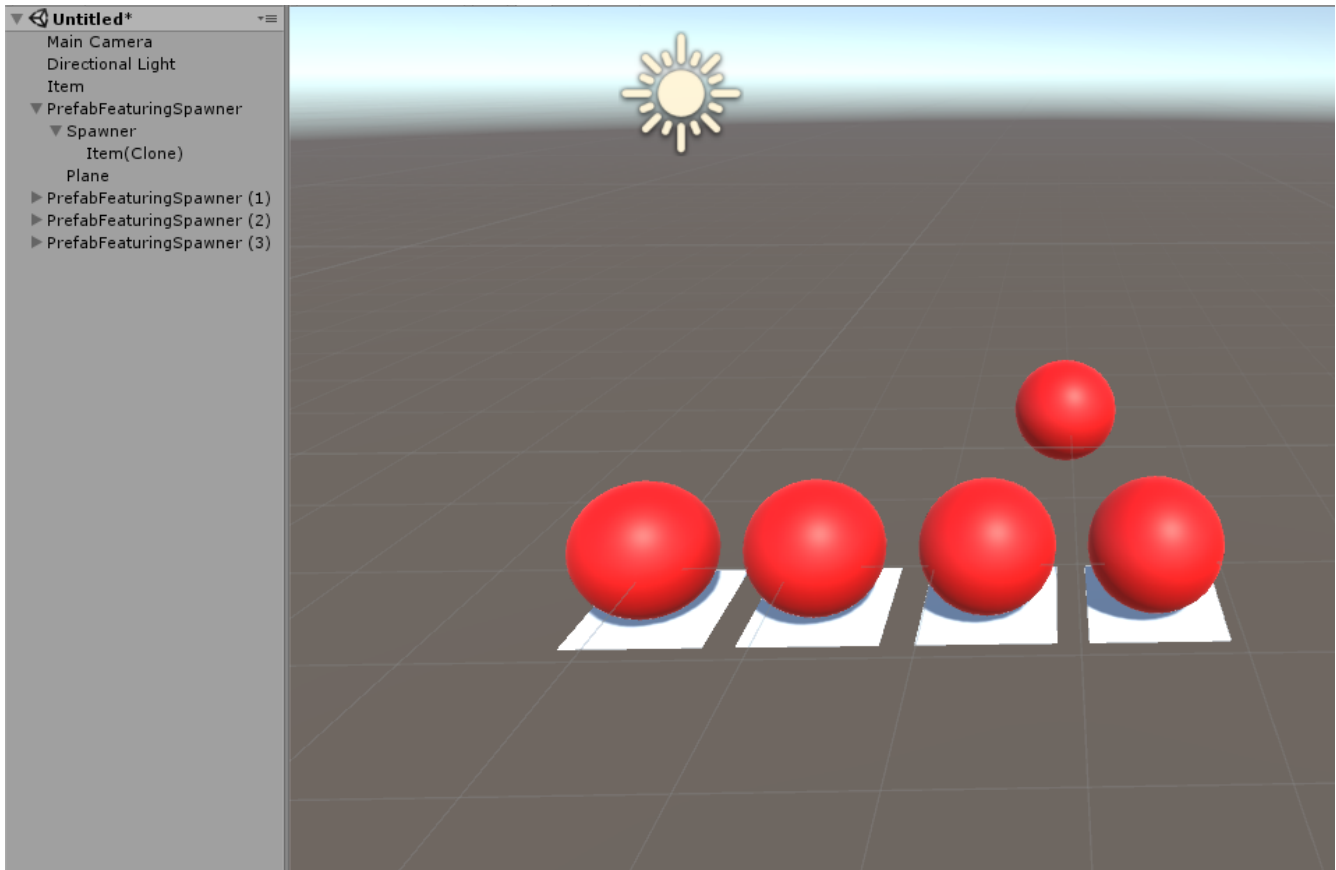
**Important : Do not attach any game object to the Spawner object. They will be destroyed upon entering play mode and may cause other weird behavior.**

And here is our scene after the item has spawned!



As you can see in the scene object tree, our item has successfully been cloned and placed where the item spawner was!

Lets take this to another level and duplicate our spawners a couple times.

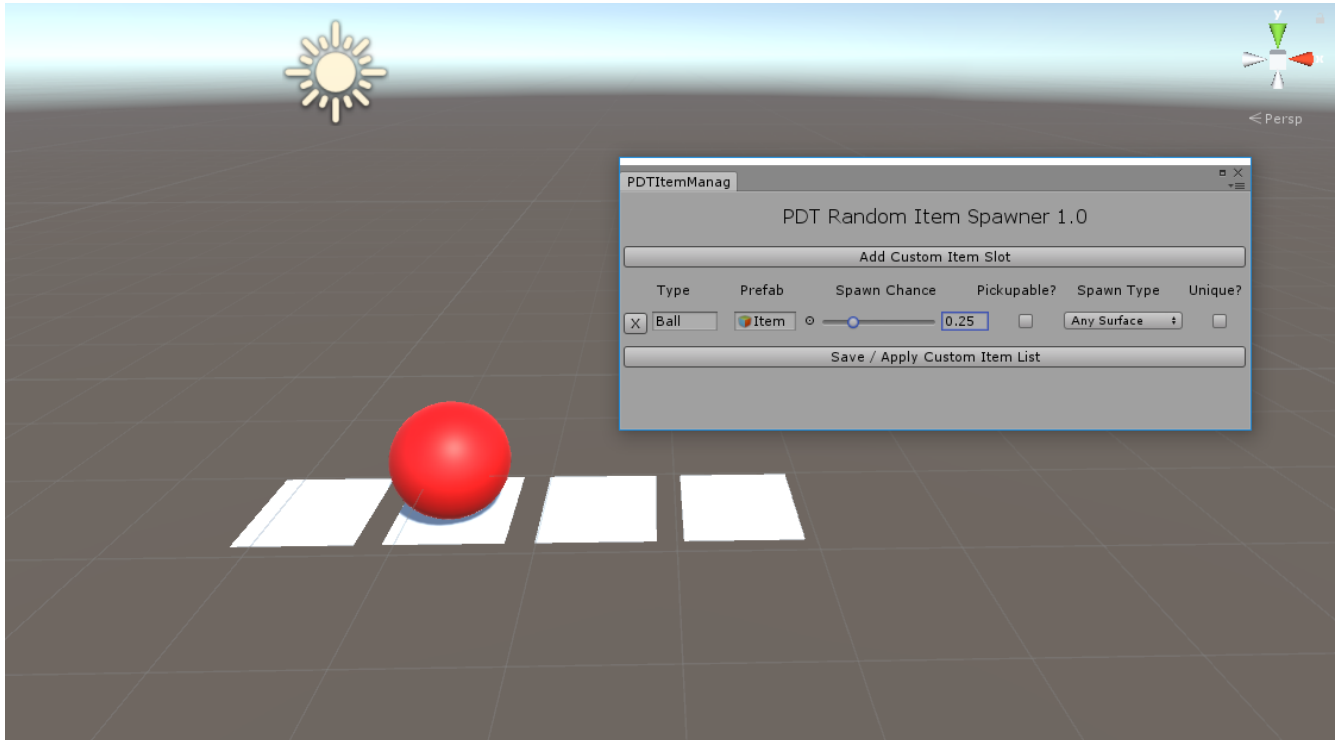


Since this object has a 100% spawn chance, it generates everywhere!

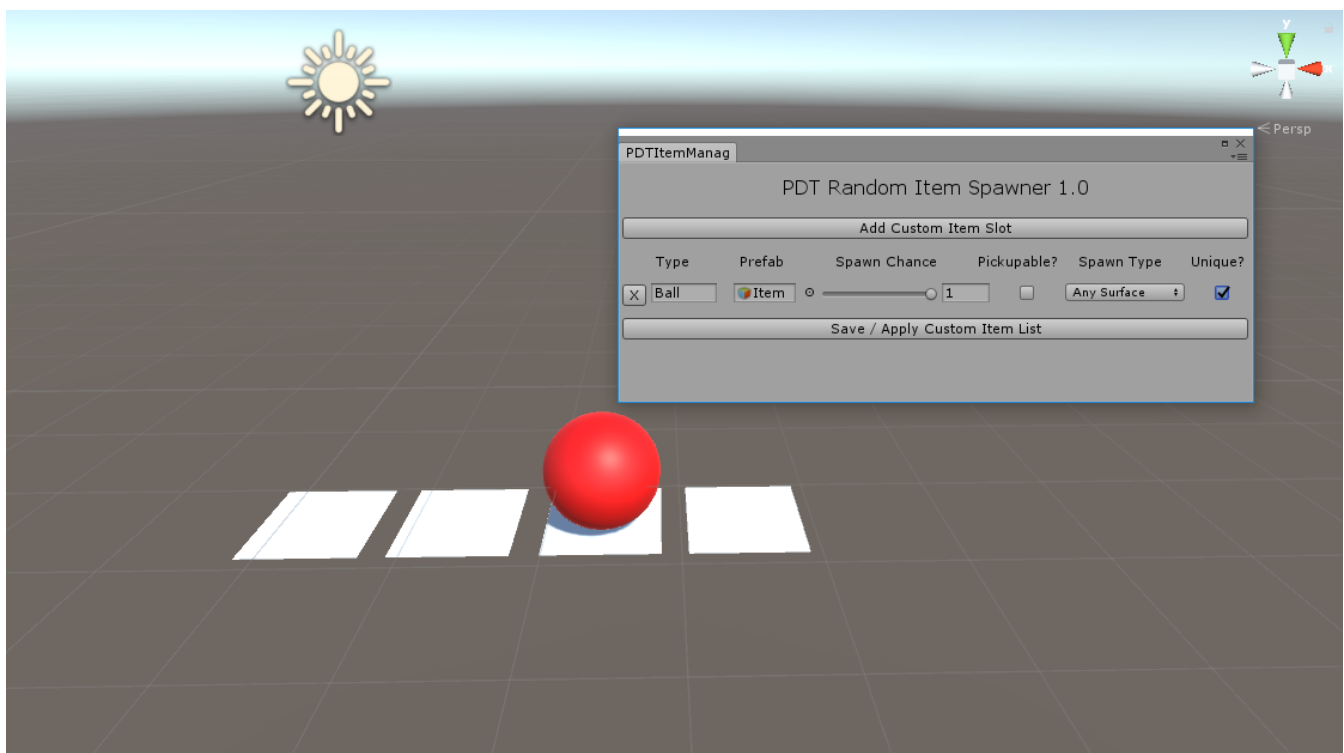
If you have gotten to this point in the tutorial, you should now be able to use this asset to spawn a particular item all over your level, but this does not make for much interesting or eventful gameplay. We need to randomize the generation a little bit.

To do this, lets start editing some parameters within the Item Spawner GUI and observe the unfolding changes.

Here, a spawn chance of 25% caused 1 ball out of 4 to appear. While the randomness of RNG can be statistically satisfying from time to time, true randomness does not guarantee this will happen.



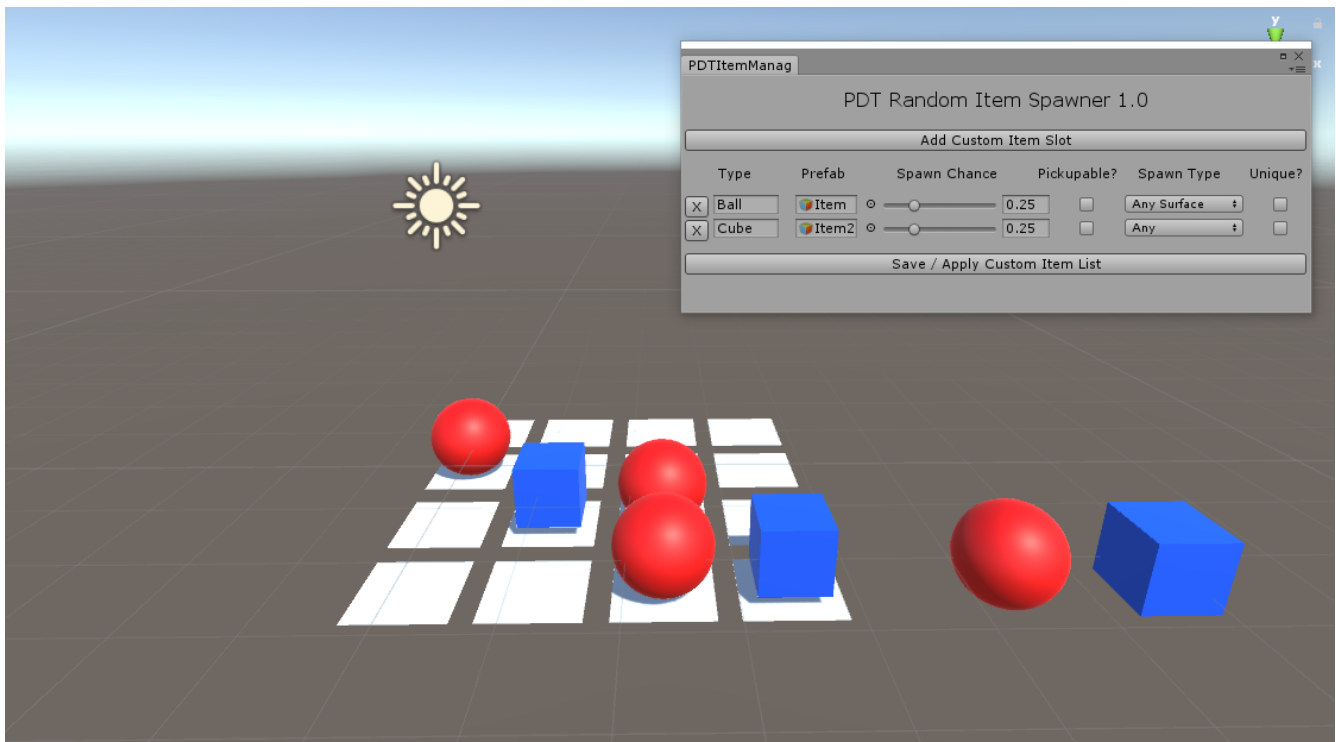
Here, an item with 100% spawn chance only generated once since it is marked as being “Unique”.





The “Unique?” toggle allows every item to become a one time spawning item. This is extremely convenient for gameplay relevant items that are not just random loot.

To show off a little more setups of the item spawners, I have duplicated the spawners a bunch of times and added a new “Cube” item to the GUI.

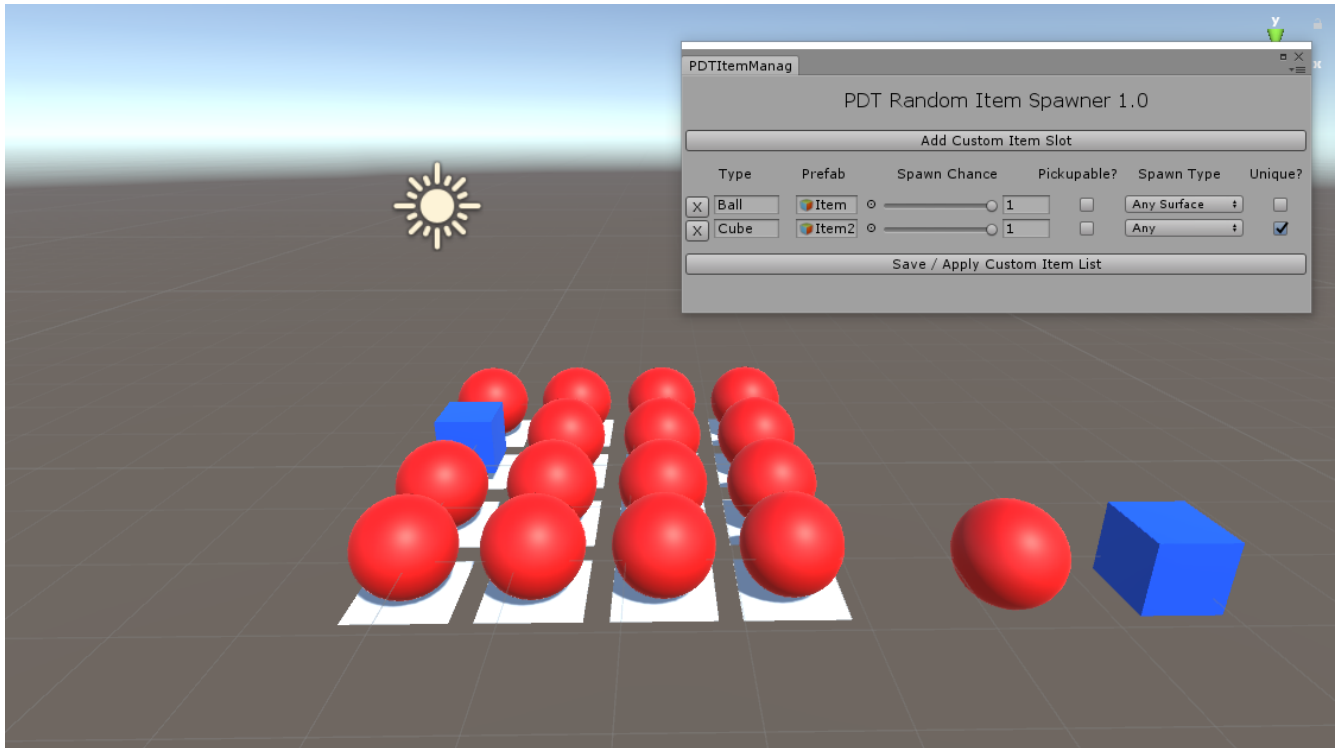


As you can see, the Cube item also spawns on the “Open Surface” type spawners since it can spawn on “Any” spawn type.

Also, while both of those items have a 25% chance of spawning, none of them are statistically “truthful” to this 25%. Keep this in mind when adding new items.

You may need to do some play testing and balancing regarding the amount of items that SHOULD generate versus the amount of items that ACTUALLY generate.

Here is another configuration of the item spawner. Here using 100% spawn chance for both items, except the “Cube” item is now set as an Unique item.



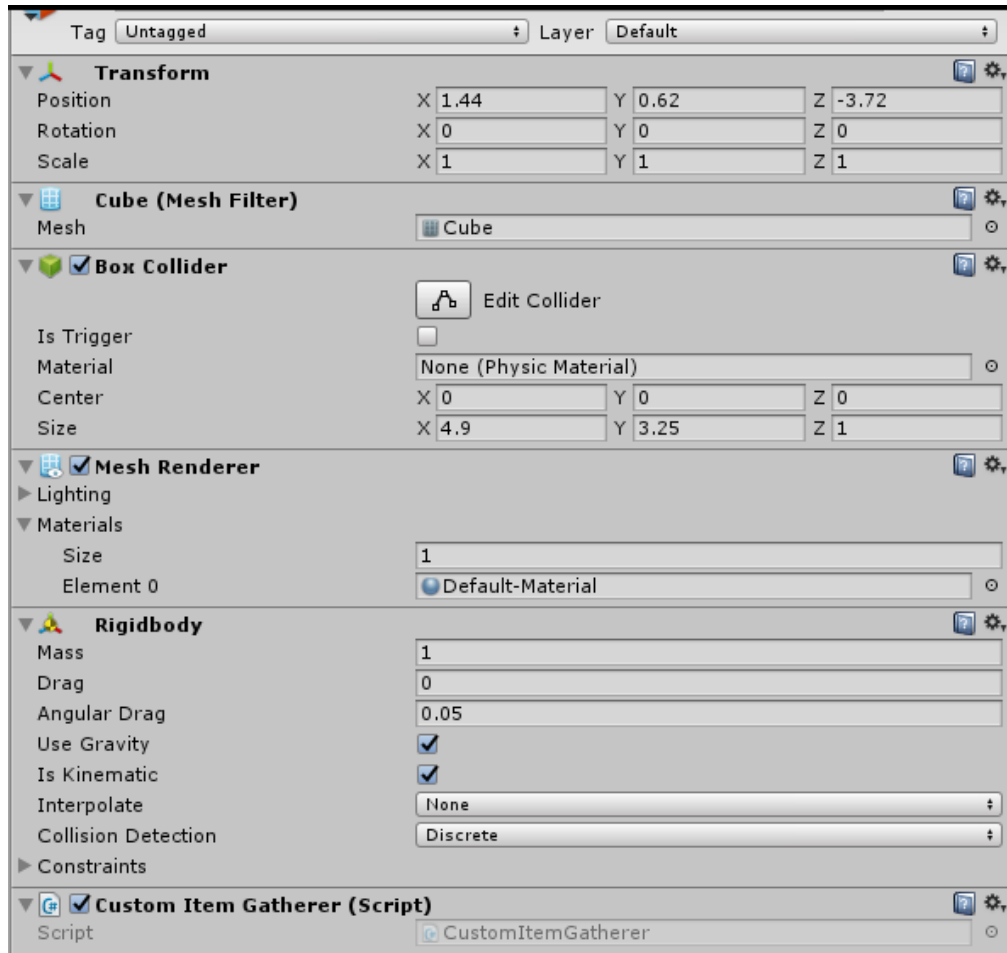
As you can see, the Cube item spawned once while the Ball item spawned everywhere else.

At this point, it is extremely easy to simply add more items to the GUI and see them instantly added to the spawners we have set up in this scene.

Indeed, the demo scene is more than just a quick set up scene : it also allows you to quickly add and test items without having to edit your actual game project.

Now you may notice we have yet to use one of those toggles. The “Pickupable?” toggle is rather self explanatory : it allows items to be picked up by GameObjects that are allowed to do so.

To set up a game object able to pick up your items, it needs a couple components. First, you need a non trigger collider. Next, you need a rigidbody component set to kinematic to disable physics.

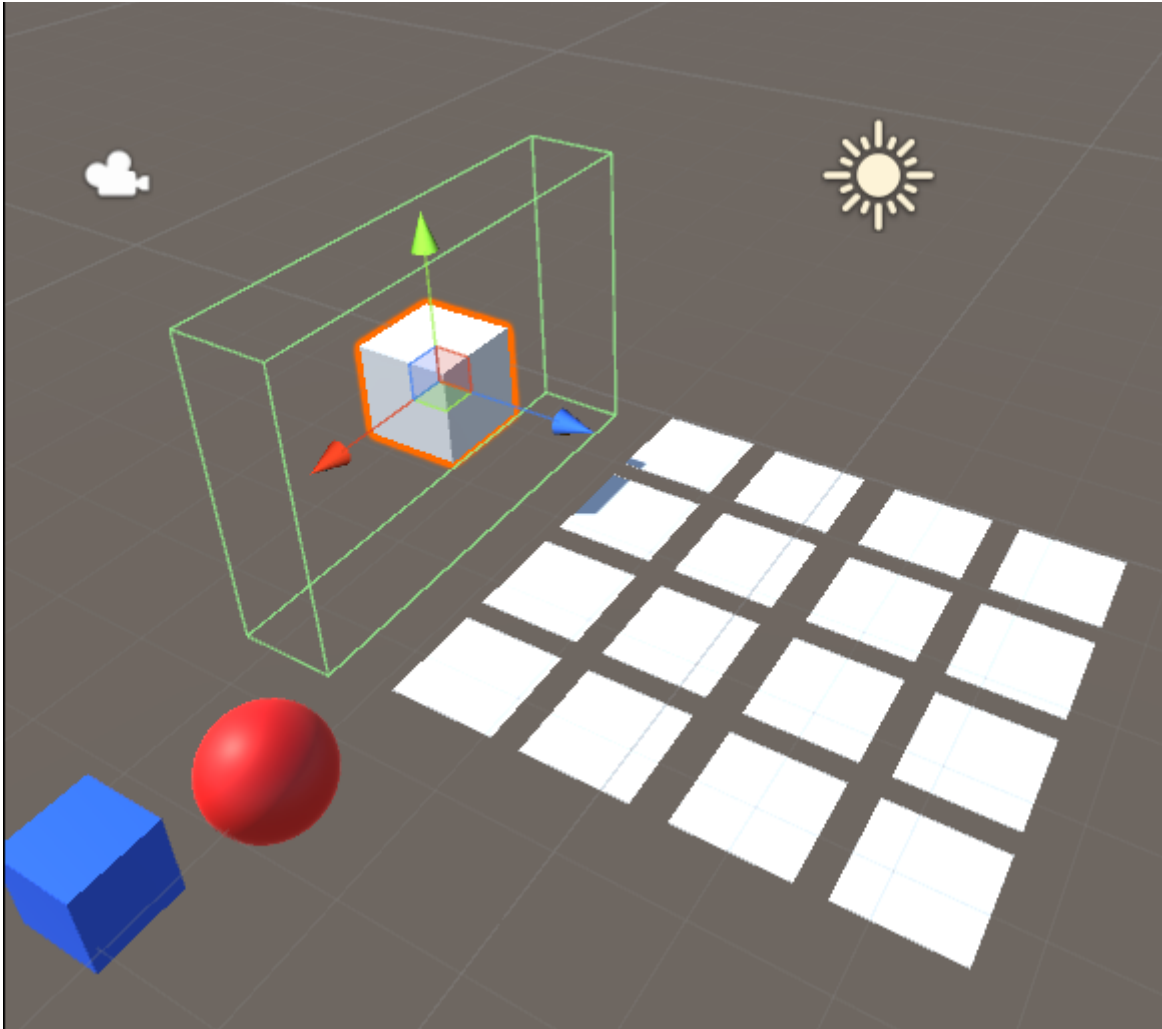


Finally you need to add the “Custom Item Gatherer” script to this game object. This is the script that allows the item to be picked up.

I have also kept the original cube mesh renderer components and extended the collider box beyond the size of the actual cube. This bigger bounding box is simply for demonstration purpose as it will allow me to slide the item gatherer above our spawners and pick up every item in one go.

In a real game, you would set this collider to whichever size you feel is comfortable for players to pick up items as they play.

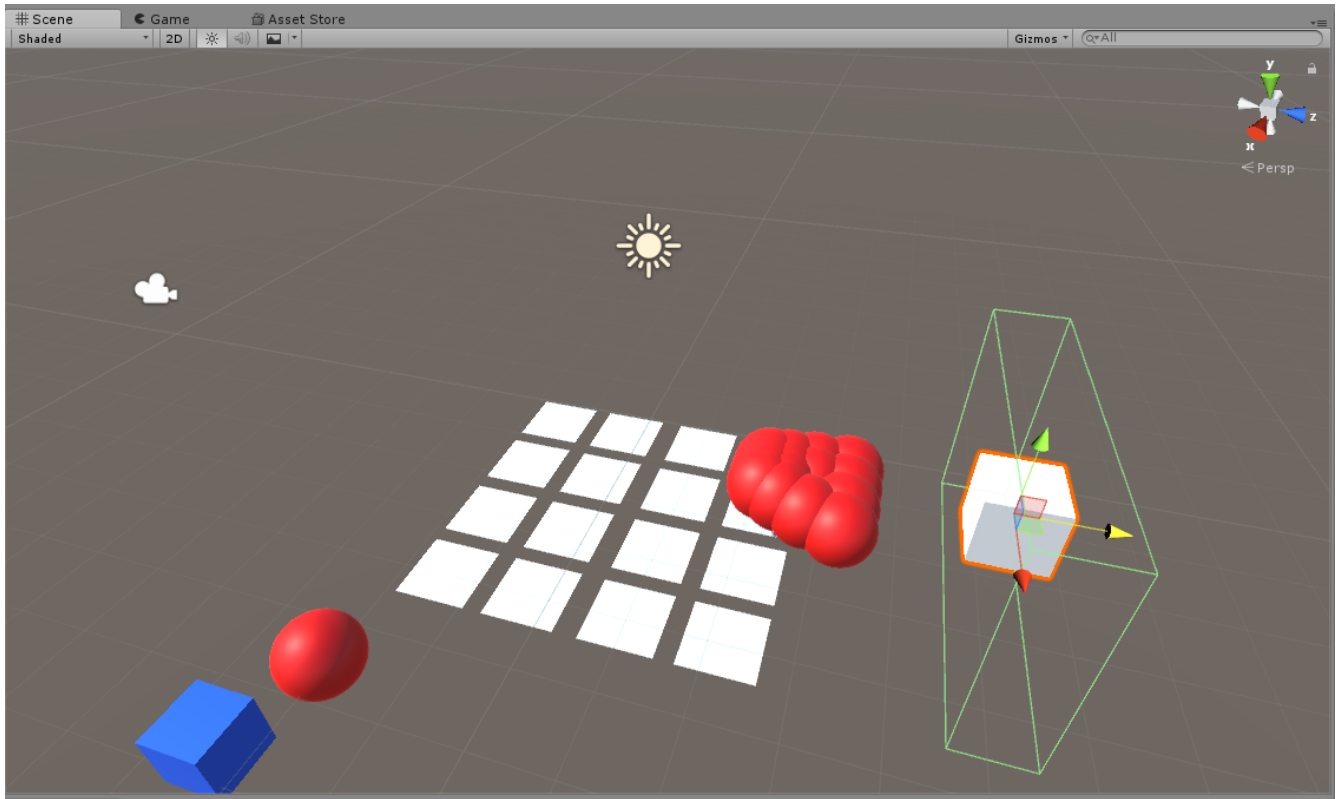
Here we have this item gatherer in scene view.



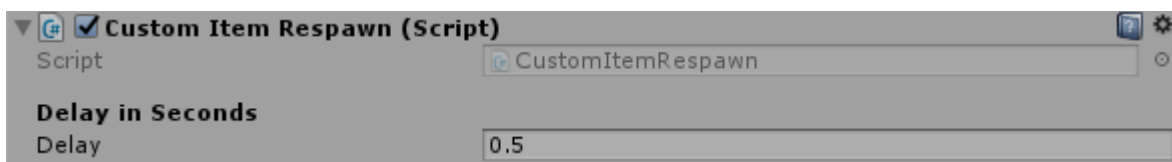
You can see the bigger bounding box covering the width of our spawner lot.

Before you hit play and test this, make sure you have marked whichever items you want to pick up as Pickupable within the GUI.

Finally, you can hit play and drag the item gatherer across our spawners to see that items follow and are “picked up” by our item gatherer when they are close enough.



Version 1.1 includes a new script allowing you to set a respawn timer to certain item spawners. Simply add the CustomItemRespawn script to the spawner on which you wish to allow items to respawn.

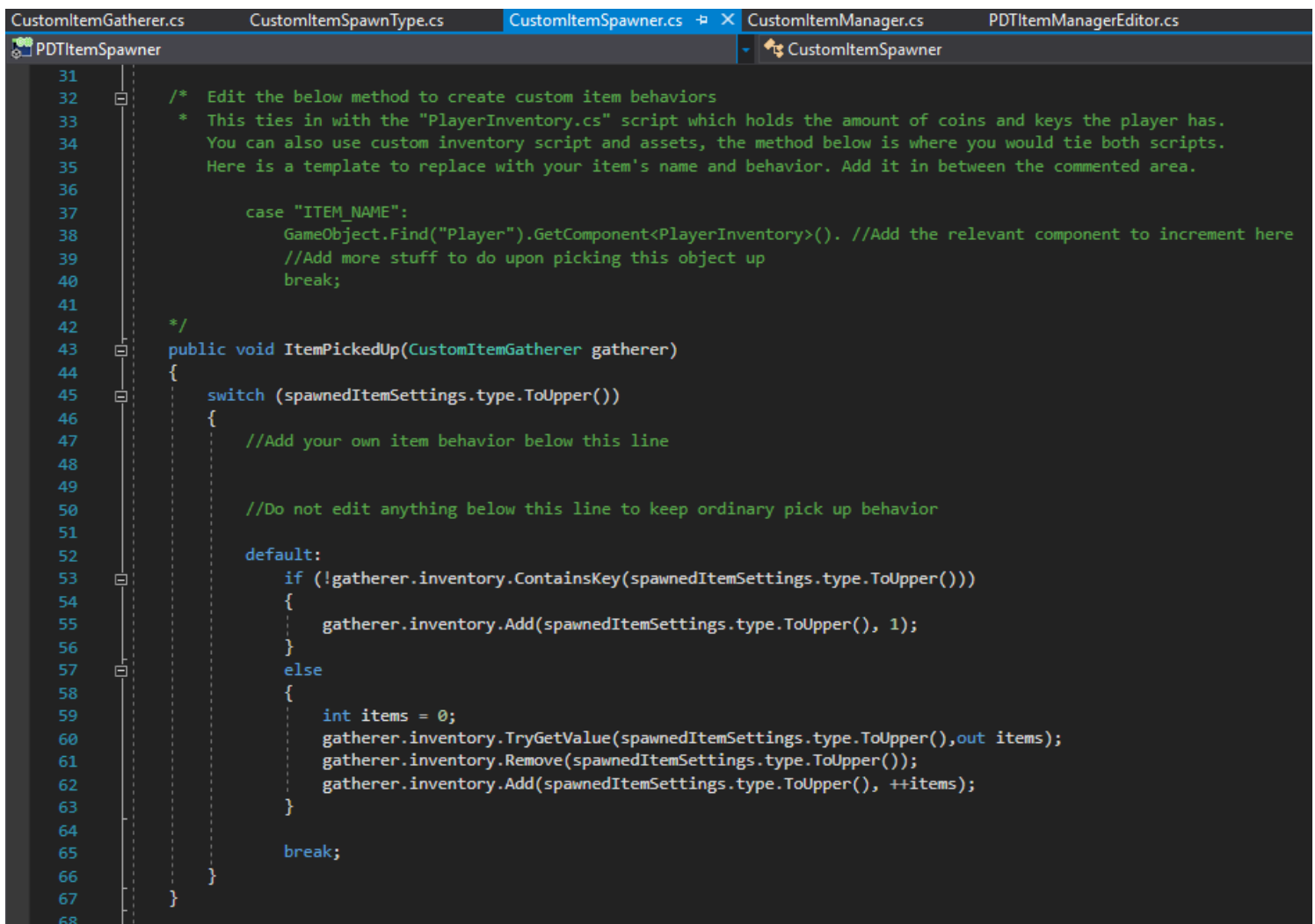


It is now easy to simply set a respawn delay representing the amount of seconds that pass in between each attempt to spawn a new item. Keep in mind that the “Spawn Chance” parameter of each item that is lower than 100% might cause this delay to take longer. Also, note that this countdown will only begin after the previous item is picked up.

Now that we have gone through all of the GUI features of this asset, lets see how we can take it to the next level and customize our item spawners even further.

For example, the item gatherer script is merely a “Holder” for the amount of items of a particular type you have picked up. If you are using another asset for inventory management, you may need to replace my script and add references to this inventory manager within the existing code.

Lets look at a couple scripts and what they mean for the item spawning process so that you can edit them if you need to.



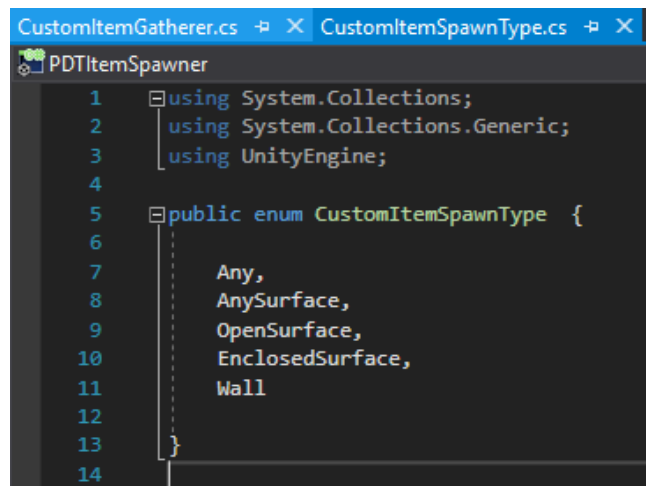
```
31
32  /* Edit the below method to create custom item behaviors
33  * This ties in with the "PlayerInventory.cs" script which holds the amount of coins and keys the player has.
34  You can also use custom inventory script and assets, the method below is where you would tie both scripts.
35  Here is a template to replace with your item's name and behavior. Add it in between the commented area.
36
37      case "ITEM_NAME":
38          GameObject.Find("Player").GetComponent<PlayerInventory>(). //Add the relevant component to increment here
39          //Add more stuff to do upon picking this object up
40          break;
41
42  */
43  public void ItemPickedUp(CustomItemGatherer gatherer)
44  {
45      switch (spawnedItemSettings.type.ToUpper())
46      {
47          //Add your own item behavior below this line
48
49          //Do not edit anything below this line to keep ordinary pick up behavior
50
51          default:
52              if (!gatherer.inventory.ContainsKey(spawnedItemSettings.type.ToUpper()))
53              {
54                  gatherer.inventory.Add(spawnedItemSettings.type.ToUpper(), 1);
55              }
56              else
57              {
58                  int items = 0;
59                  gatherer.inventory.TryGetValue(spawnedItemSettings.type.ToUpper(),out items);
60                  gatherer.inventory.Remove(spawnedItemSettings.type.ToUpper());
61                  gatherer.inventory.Add(spawnedItemSettings.type.ToUpper(), ++items);
62              }
63
64              break;
65      }
66  }
67
68
```

First of all, if you want to create custom behaviours that happen when an item is picked up, such as toggling on a light after a flashlight item is picked up, “ItemPickedUp” within “CustomItemSpawner.cs” is where you make edits. Items are found using their “Type” parameter, which is the name you gave the item. Within this function, item names are interpreted as case insensitive meaning they will be CAPITALIZED before being tested.

Simply copy and paste the commented template, replace the ITEM NAME with your capitalized item's name and add your custom code in there.

Note that if you handle your item in here, the original “default” behaviour of adding the item to a “Holder” variable doesn't happen.

Next up, Spawn Types. To add more of those, simply open up the “CustomItemSpawnType.cs” script and add more types to the Enum.



```
CustomItemGatherer.cs x CustomItemSpawnType.cs x
PDTItemSpawner
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public enum CustomItemSpawnType {
6
7     Any,
8     AnySurface,
9     OpenSurface,
10    EnclosedSurface,
11    Wall
12 }
13
14
```

However, an added type will only spawn items featuring this exact spawn type or the “Any” type. To create general usage types like my “Any Surface” type which allows the object to spawn on any spawn of a “Surface” type, you need to edit the following method, named “spawnTypeCheck” within “CustomItemSpawner.cs”

```

//Edit the following method to add more general spawn types
//which need to be handled and return true if the item type
//is equal to the types included by your general spawn type
bool spawnTypeCheck(CustomItemSpawnType item)
{
    if (item == spawnType) return true;

    else if(item == CustomItemSpawnType.AnySurface)
    {
        return (spawnType == CustomItemSpawnType.EnclosedSurface || spawnType == CustomItemSpawnType.OpenSurface);
    }
    else if(item == CustomItemSpawnType.Any)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

The logic behind making general spawn types should be quite easy to understand for anyone who has dealt with programming before.

For those who are not familiar with programming let me try and explain. The first “if” clause will return TRUE ( and allow the item to spawn ) if the item's spawn type is exactly the same as our spawner's spawn type.

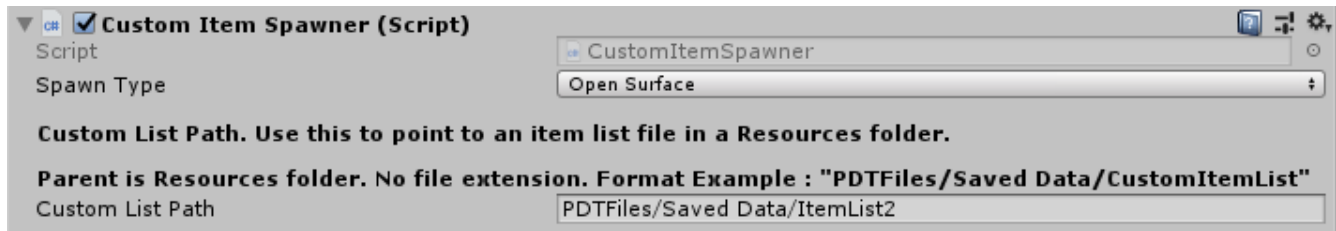
The first “else if” is a general usage spawn type. If the item's type is equal to AnySurface, this algorithm will return TRUE if the spawner's type is EnclosedSurface or OpenSurface.

In Procedural Dungeon Toolkit, this feature is used to make sure candles are not placed where their flame would “burn” through the prefab on which it spawned. Any spawn that should be able to spawn that candle is set to “Open Surface” and others “Enclosed Surface”.

The next “else if” checks if the item's type is equal to “Any” in which case the algorithm will always return TRUE. And finally the “else” clause is what happens if nothing matched, which returns FALSE meaning the object cannot spawn there.



Added in version 1.2 is the ability to use custom list files per spawner / scene by pointing to a list file inside of the Resources folder using a string parameter on the spawner component.



Follow the format example to point to custom folder structures that you can use to sort item list files for use in different scenes. Do not add the .txt extension to this parameter. Do not start with a slash.

As long as the file exists in Resources and that the objects it uses exist in the current scene, this feature allows you to create anything from different difficulty levels to entirely different item sets depending on the scene you are playing.

With that, you should now be able to fully customize this asset and start using it to create awesome random item spawners in your games. Again, the asset is fully compatible with standalone builds!

This concludes the General Usage Tutorial for PDT Item Spawner.

For more information, question or doubts, please send me an email.

Revision3 - [r3eckon@gmail.com](mailto:r3eckon@gmail.com)