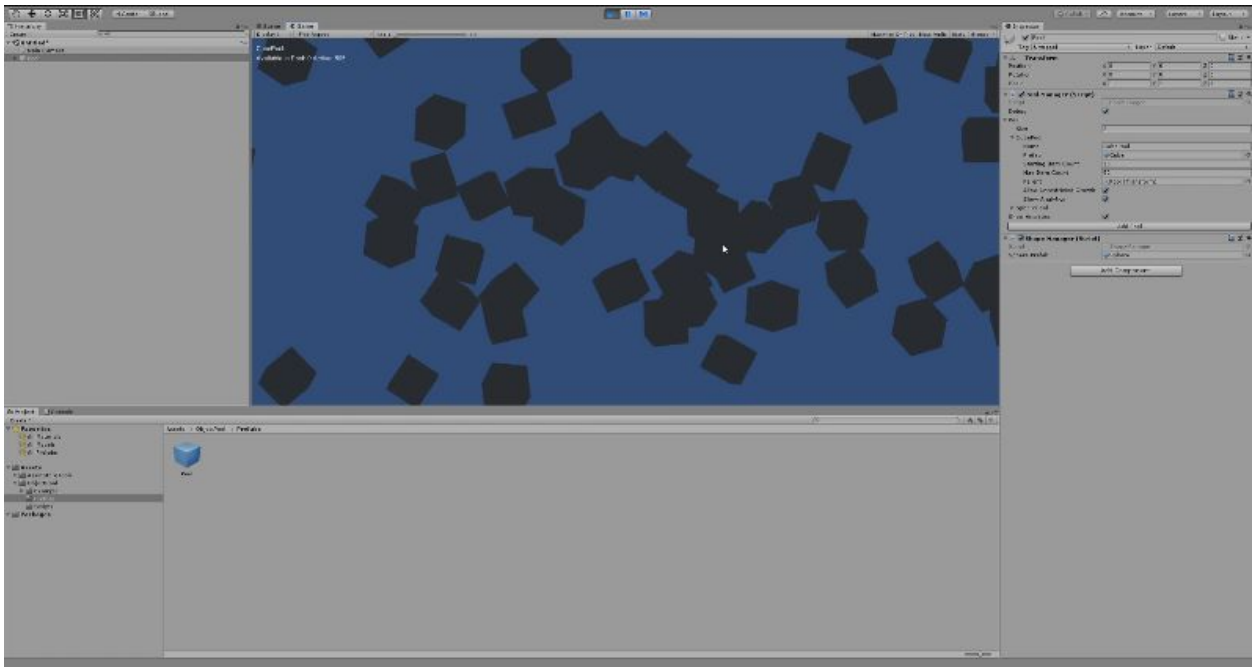
GameObject Pool Manager Documentation



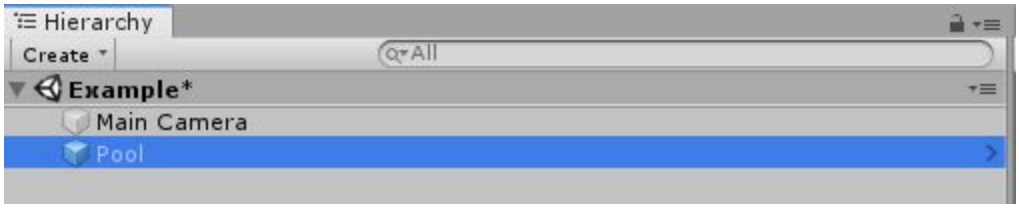
GameObjectPool.Pool extends *System.Collections.Generic.Queue* and takes instructions in the form of PoolSettings objects in order to minimize performance issues when allocating and deallocating GameObjects.

Pools can be predefined by modifying the settings of a PoolManager object, or can be defined within scripts.

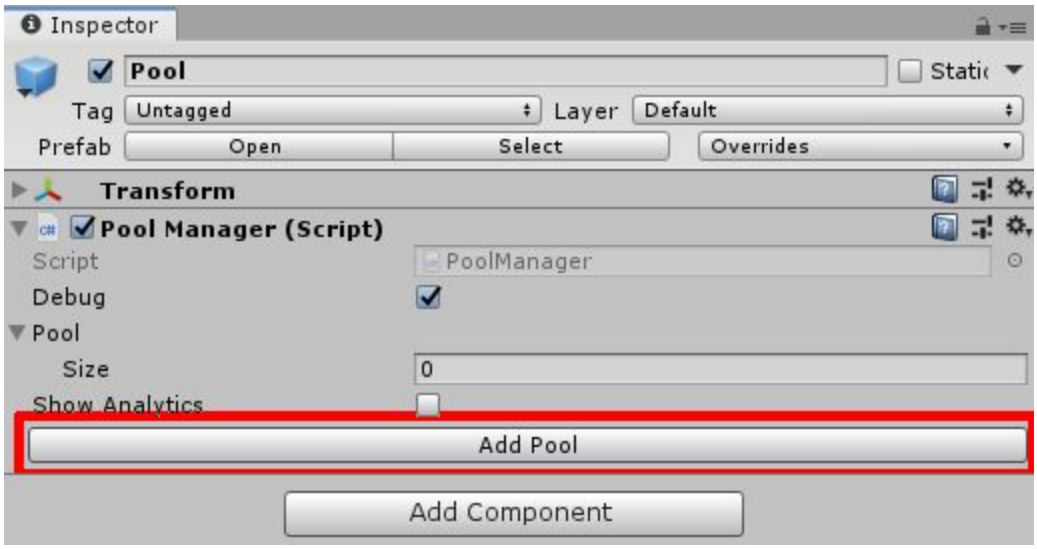
Defining pools in a Pool Manager's settings

Navigate to **Assets > ObjectPool > Prefabs** and locate the Pool prefab

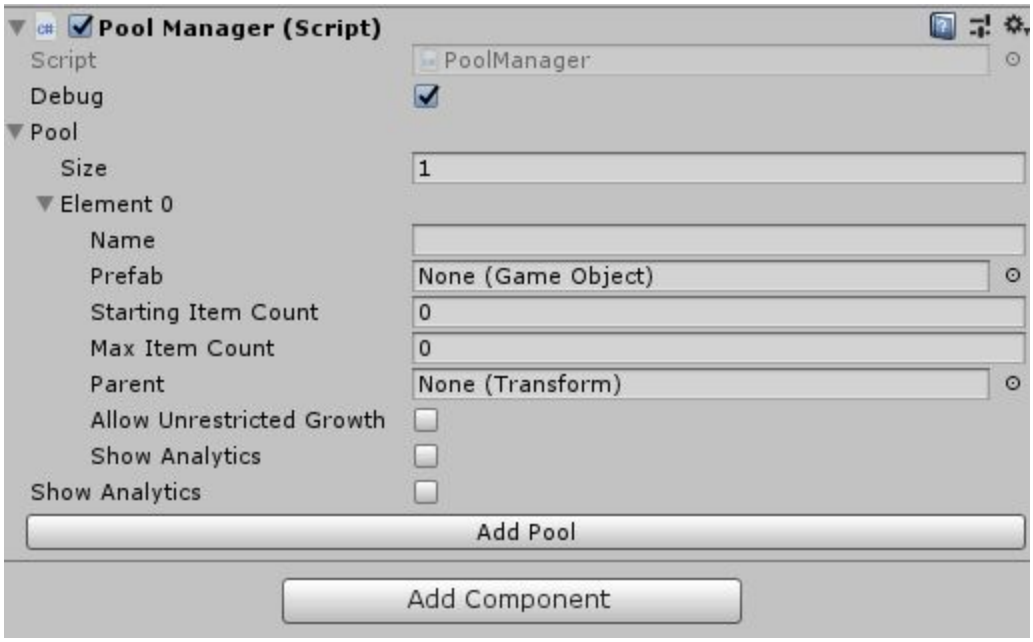
Drag the Pool prefab into your scene's Hierarchy and open the inspector tab with Pool highlighted



In the inspector tab click Add Pool to add a Pool Settings object to the available Game Object pools



A new pool element will appear with the following settings



Pool Settings attributes

Name	The name of the pool - this will be used to reference the pool within your scripts.
Prefab	The prefab item that will fill the pool.
Starting Item Count	The number of items to initially spawn at run-time.
Max Item Count	The maximum number of items to add to the pool.
Parent	Objects spawned for the pool will be parented to this Transform.
Allow Unrestricted Growth	Check this to allow the pool to continue populating itself with objects after max item count is reached.
Show analytics	Display information about the pool in the GUI.

Defining pools in code

Pools can also be defined within code and instantiated at runtime.

Example:

```
PoolManager.AddPool(new PoolSettings
{
        name = "SpherePool",
        prefab = spherePrefab,
        startingItemCount = 10,
        maxItemCount = 50,
        parent = this.transform,
        allowUnrestrictedGrowth = true
});
```

Managing pool items

Retrieve an item from a Pool

In order to retrieve an object from a pool, Use the **PoolManager.Get** static method. There are four ways you can call this method:

> **PoolManager.Get**(string name);
> **PoolManager.Get**(string name, Vector3 position);
> **PoolManager.Get**(string name, Vector3 position, Quaternion rotation);
> **PoolManager.Get**(string name, Vector3 position, Quaternion rotation,Transform parent);

Each of these methods returns a GameObject from the requested pool (defined by string **name**)

Return an item to the Pool

In order to return an item to the pool, call SetActive(false) on the item:

```
transform.gameObject.SetActive(false);
```

Example Scene

Navigate to **ObjectPool > Scenes** and open the scene labeled **Example** and press play.

The initial Cube pool is instantiated with 10 deactivated Cube prefabs which are parented to the Pool object.

Left click within the Game window and the initial 10 Cubes will spawn, emptying the pool. Max Item Count is set to 50, so 40 more deactivated Cube prefabs will be Instantiated at run-time, added to the scene and then retrieved from the pool and activated.

Check “unrestricted growth” to the pool grow in size.

After a short amount of time the Cubes will deactivate and be added back to the pool. You will see this reported in the GUI.

Similarly Right clicking in the Game window will spawn spheres with unrestricted growth. Note that the sphere settings are added at run-time.

Run-time pool instantiation can be seen in the **Start** method within
ObjectPool > Scripts > ShapeManager.cs

```
private void Start()
{
	PoolManager.AddPool(new PoolSettings
	{
	name = "SpherePool",
	prefab = spherePrefab,
	startingItemCount = 10,
	maxItemCount = 50,
	parent = this.transform,
	allowUnrestrictedGrowth = true
	});
}
```

Pool object retrieval can be seen in the **Update** method within
ObjectPool > Scripts > ShapeManager.cs

```
void Update()
{
	Vector3 location = new Vector3(
	Camera.main.ScreenToWorldPoint(Input.mousePosition).x,
	Camera.main.ScreenToWorldPoint(Input.mousePosition).y,
	0
	);

	// On left mouse click, spawn a cube object from the cube pool.
	if (Input.GetMouseButton(0)) PoolManager.Get(cubePool, location, Random.rotation);

	// On right mouse click, spawn a sphere object from the sphere pool.
	if (Input.GetMouseButton(1))
	{
	if (PoolManager.HasPool(spherePool))
	{
		PoolManager.Get(spherePool, location, Random.rotation);
	}
	}
}
```