pmjo's Fast Spawn for Unity 3d

Description

Fast Spawn allows the game to spawn game objects without any significant loss in performance. It also gives you an easy way to manage and access your prefabs. The high performance spawning is based on pooling, i.e. keeping a user selected amount of pre-instantiated/deactivated game objects in memory. Pooling will also greatly reduce the amount of required garbage collection which can can cause random lags in games. The spawn objects may be easily divided into groups and the groups can be selectively loaded and unloaded during level/scene changes or even during the gameplay.

Designing your Fast Spawn manager

The first thing to do is to design your Fast Spawn manager. This is done by creating a C# script with a class which inherits from FastSpawnManager. The manager will contain the references to the prefabs which you wish to be able to spawn. The prefab references are defined as public FastSpawnObject items and they can be divided into groups by simply adding them inside to a serializable public class.

Your Fast Spawn manager may contain any number of groups and the groups may contain any number of spawn objects. After you have designed your groups and the items inside them you should expose them. The manager may be modified at any time so you don't have to finish it at once.

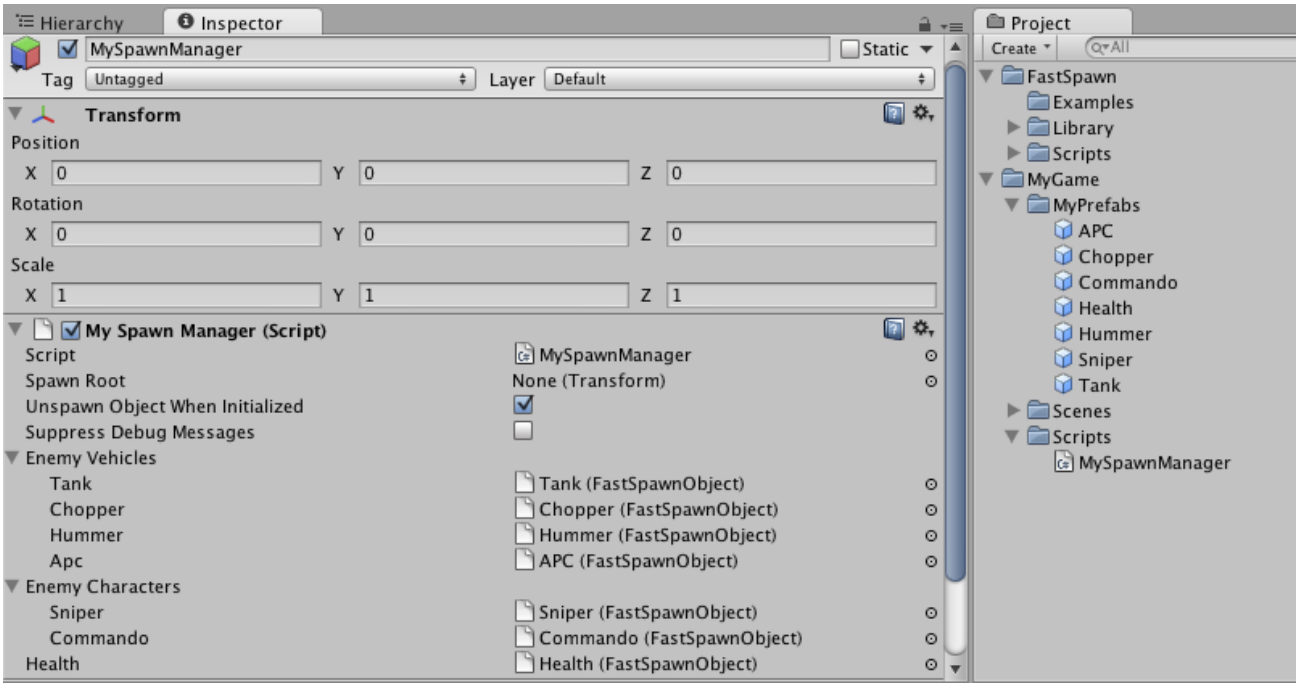
Simple Fast Spawn manager example

```
public class MySpawnManager : FastSpawnManager {

    // Define groups with prefabs you wish to be able to spawn

    [System.Serializable]
    public class EnemyVehicles {
```

```
        public FastSpawnObject tank;
        public FastSpawnObject chopper;
        public FastSpawnObject hummer;
        public FastSpawnObject apc;
    }

    [System.Serializable]
    public class EnemyCharacters {
        public FastSpawnObject sniper;
        public FastSpawnObject commando;
    }

    // Expose the groups

    public EnemyVehicles enemyVehicles;
    public EnemyCharacters enemyCharacters;

    // Groups are not required if you don't want to use them

    public FastSpawnObject health;
}
```

Connecting the prefabs to your Fast Spawn manager

After you have designed your manager you should create an empty game object and drag the C# script created in 'Designing your Fast Spawn manager' on it. You should also drag FastSpawnObject script on all of your prefabs. After that you will be able to connect the prefabs to the manager prefab reference slots.



If you inspect the FastSpawnComponent on your prefabs you notice that it contains two configurable options, spawn count and spawn type. The spawn count is the maximum count of prefab instances you wish to be able to exist at the same time. The spawn type defines how the next activated instance will be selected. The Always type will always activate the oldest spawn object, even if it is still in use. It is suitable for things like bullets trails and it performs slightly better than the IfAvailable type. The IfAvailable type is suitable objects such as characters, vehicles and things which may never be reused before they are really unspawned. The actual unspawn should happen for example when the bullet gets too far away or when a vehicle gets destroyed. You should never actually destroy a Fast Spawn object, you should just unspawn it instead.

Loading and unloading

Before you can actually spawn objects you will have to load the prefabs. Prefabs may be loaded or unloaded one by one or in groups. To save some memory you should only load what you need in the current level and unload all unneeded items when the level is changed. Loading and unloading may be called inside the manager script or in any other C# script. Incase you destroy the spawn manager, all loaded items will be unloaded automatically.

In a very simple game you could do the loading in the Start function inside the manager script. You should never load objects inside the Awake function since the Fast Spawn manager might not yet be initialized.

```
void Start() {
    // Load two groups at the same time
    LoadObjects(enemyVehicles, enemyCharacters);

    // You may also load items one by one
    LoadObjects(health);
}
```

You could also make your Fast Spawn manager persistent (not to be destroyed when the scene is changed) by calling DontDestroyOnLoad(gameObject) inside the Start function and then load the required items when the level is changed on OnLevelWasLoaded function.

```
public class MySpawnManager : FastSpawnManager {

    // Define groups with prefabs you wish to be able to spawn

    [System.Serializable]
    public class Level1 {
        public FastSpawnObject soldier;
        public FastSpawnObject commando;
    }
```

```
    [System.Serializable]
    public class Level2 {
        public FastSpawnObject tank;
        public FastSpawnObject hummer;
        public FastSpawnObject apc;
    }

    [System.Serializable]
    public class Level3 {
        public FastSpawnObject mammothTank;
        public FastSpawnObject chopper;
    }

    [System.Serializable]
    public class Common {
        public FastSpawnObject smallExplosion;
        public FastSpawnObject bigExplosion;
        public FastSpawnObject bulletTrail;
        public FastSpawnObject repairKit;
    }

    // Expose the groups

    public Level1 level1;
    public Level2 level2;
    public Level3 level3;
    public Common common;

    // Make this manager persistent

    void Start() {
        DontDestroyOnLoad(gameObject);
    }

    // You can use OnLevelWasLoaded from MonoBehaviour to load and unload needed items when level
is changed

    void OnLevelWasLoaded(int level) {

        // Unload previous items (you could also unload only the unneeded items)

        UnloadAllObjects();

        // Always load common items

        LoadObjects(common);

        // Load items based on the loaded scene

        switch(level) {
```

```
            case 1:
                LoadObjects(level1);
                break;

            case 2:
                LoadObjects(level2);
                break;

            case 3:
                LoadObjects(level3);
                break;
        }
    }
}
```

Another way of handling multiple levels is to create multiple Fast Spawn managers, one for each level. Incase you don't want to create the prefab instances to the root of your hierarchy you may set the spawn root game object on your spawn manager instance.

Spawning and unspawning objects

The actual spawning is done by creating a reference to your Fast Spawn manager and calling the SpawnObject function. The SpawnObject takes 3 parameters, the prefab you wish spawn, the target position and rotation. To unspawn an object you should call UnspawnObject with the FastSpawnObject instance you wish to unspawn. You should never actually destroy a Fast Spawn object, you should just unspawn it instead. If you want to totally remove it from memory, use the Unload function.

SpawnObject example

```
using UnityEngine;
using System.Collections;

public class MyGameLogic : MonoBehaviour {
    private FastSpawnManager mySpawnManager;
    private bool shouldSpawnGroupOfVehicles = true;

    void Awake() {
        mySpawnManager = (FastSpawnManager) FindObjectOfType(typeof(FastSpawnManager));
    }

    void Update() {
        if(shouldSpawnGroupOfVehicles) {
            // Spawn 5 tanks to a distant location
            for(int i=0; i<5; i++) {
                mySpawnManager.SpawnObject(mySpawnManager.vehicles.tank, new Vector3(Random.Range
(-10.0f, 10.0f), 0.0f, 50.0f), Quaternion.identity);
            }
            shouldSpawnGroupOfVehicles = false;
        }
```

```
    }
}
```

UnspawnObject example

```
using UnityEngine;
using System.Collections;

public class Tank : MonoBehaviour {
    private FastSpawnManager mySpawnManager;
    private FastSpawnObject thisSpawnObject;

    void Awake() {
        mySpawnManager = (FastSpawnManager) FindObjectOfType(typeof(FastSpawnManager));
        thisSpawnObject = GetComponent<FastSpawnObject>();
    }

    void OnCollisionEnter(Collision collision) {
        // Unspawn this spawn object when it get hit by a missile and spawn an explosion to the s
ame location
        if(collision.transform.gameObject.tag.Equals("missile")) {
            mySpawnManager.SpawnObject(mySpawnManager.effects.bigExplosion, transform.position, Q
uaternion.identity);
            mySpawnManager.UnspawnObject(thisSpawnObject);
        }
    }
}
```

Receiving spawn events inside the spawn objects

When your characters, vehicles and other game objects have complicated AI, state handling etc. you probably want to know when the object is spawned and when it is unspawned. This can be easily achieved by subscribing to OnSpawn and OnUnspawn events. Note: Incase you are still running Unity 3.5 you can use the OnSpawn and OnUnspawn events to enable and disable your spawn object's children.

```
using UnityEngine;
using System.Collections;

public class SomeGameObject : MonoBehaviour {
    private FastSpawnObject thisSpawnObject;

    void Awake() {
        thisSpawnObject = GetComponent<FastSpawnObject>();

        // Add spawn and unspawn event listeners

        if(thisSpawnObject != null) {
```

```
            thisSpawnObject.OnSpawn += OnSpawn;
            thisSpawnObject.OnUnspawn += OnUnspawn;
        }
    }

    private void OnSpawn() {
        // Reset your AI etc. here or at OnUnspawn
        // Incase you are running Unity 3.5 you can enable your child game objects here with SetA
ctiveRecursively or preferably via cached child objects
        Debug.Log(gameObject.name + " just got spawned!");
    }

    private void OnUnspawn() {
        // Incase you are running Unity 3.5 you can disable your child game objects here
        Debug.Log(gameObject.name + " just got unspawned!");
    }

    void OnDestroy() {
        // Remove spawn and unspawn event listeners
        if(thisSpawnObject != null) {
            thisSpawnObject.OnSpawn -= OnSpawn;
            thisSpawnObject.OnUnspawn -= OnUnspawn;
        }
    }
}
```

By default the spawn manager also calls unspawn on your spawn objects when they are initialized but this feature can be disabled by unchecking the 'Unspawn Object When Initialized' from your spawn manager instance.

All available functions

```
// Load a single object or a group or multiple objects/groups (separated by comma)
LoadObjects(params object[] prefabs)

// Unload a single object or a group or multiple objects/groups (separated by comma)
UnloadObjects(params object[] prefabs)

// Unload all objects
UnloadAllObjects()

// Spawn instance of a prefab to position with rotation
SpawnObject(FastSpawnObject prefab, Vector3 position, Quaternion rotation)

// Unspawn instance of a prefab
UnspawnObject(FastSpawnObject prefabInstance)
```

Singleton for easier access

Instead of retrieving the reference to your FastSpawnManager you can create a singleton for easier access. By adding the following code to your spawn manager you can access it from any script with just typing MySpawnManager.SharedInstance.

```
// Singleton

private static MySpawnManager sharedInstance;

public static MySpawnManager SharedInstance {
    get {
        if(sharedInstance == null) {
            sharedInstance = (MySpawnManager)FindObjectOfType(typeof(MySpawnManager));
        }

        return sharedInstance;
    }
}

public void OnApplicationQuit() {
    sharedInstance = null;
}
```

Accessing the spawn manager through a singleton

```
// After creating the singleton you can easily access your spawn manager from any C# script

if(MySpawnManager.SharedInstance != null) {
    MySpawnManager.SharedInstance.SpawnObject(MySpawnManager.SharedInstance.vehicles.tank, Vector
3.zero, Quaternion.identity);
}
```

Creating helper functions

You can extend the spawn manager functionality by creating helper functions inside the manager.

```
public class MySpawnManager : FastSpawnManager {

    // Define groups with prefabs you wish to be able to spawn

    [System.Serializable]
    public class Zombies {
        public FastSpawnObject maleZombie1;
        public FastSpawnObject maleZombie2;
        public FastSpawnObject maleZombie3;
        public FastSpawnObject femaleZombie1;
        public FastSpawnObject femaleZombie2;
        public FastSpawnObject femaleZombie3;
```

```
    }

    [System.Serializable]
    public class AdvancedZombies {
        public FastSpawnObject mutantZombie1;
        public FastSpawnObject mutantZombie2;
    }

    [System.Serializable]
    public class Survivors {
        public FastSpawnObject male1;
        public FastSpawnObject male2;
        public FastSpawnObject male3;
        public FastSpawnObject female1;
        public FastSpawnObject female2;
        public FastSpawnObject female3;
    }

    [System.Serializable]
    public class PlayerOptions {
        public FastSpawnObject gunMan;
        public FastSpawnObject giant;
        public FastSpawnObject commando;
    }

    // Expose the groups defined above

    public Zombies zombies;
    public AdvancedZombies advancedZombies;
    public Survivors survivors;
    public PlayerOptions playerOptions;

    // You can create custom helper methods for spawning

    public void SpawnRandomMaleOrFemaleZombie(Vector3 position) {
        FastSpawnObject[] zombieOptions = { zombies.femaleZombie1, zombies.femaleZombie2, zombies
.femaleZombie3, zombies.maleZombie1, zombies.maleZombie2, zombies.maleZombie3 };
        FastSpawnObject randomZombie = zombieOptions[Random.Range(0, zombieOptions.Length + 1)];
        SpawnObject(randomZombie, position, Quaternion.identity);
    }

    // And also custom helper methods for loading and unloading

    public void OnPlayerSelected() {
        // You don't need to load the full group of items, you can load single items too
        LoadObjects(playerOptions.commando);
    }

    public void LoadBasicZombies() {
        LoadObjects(zombies);
```

```
    }

    public void OnCheckpoint1() {
        // You can unload single items
        UnloadObjects(zombies.femaleZombie3);

        // You can unload full groups too
        UnloadObjects(survivors);

        LoadObjects(advancedZombies);
    }
}
```

Using Fast Spawn with Shuriken particle systems

You might be wondering how do make a spawnable particle system. It's actually quite easy. You just have to reset the particle system when it's unspawned and play it when it gets spawned. The following example shows how to do it. Note: The following example requires Unity 4 but you may use the same logic for legacy particle effects too.

```
using UnityEngine;
using System.Collections;

public class SpawnableParticleSystem : MonoBehaviour {
    private FastSpawnObject cachedFastSpawnObject;
    public ParticleSystem cachedParticleSystem;

    void Awake() {
        cachedFastSpawnObject = GetComponent<FastSpawnObject>();
        cachedParticleSystem = GetComponent<ParticleSystem>();

        if(cachedParticleSystem == null) {
            cachedParticleSystem = GetComponentInChildren<ParticleSystem>();
        }

        if(cachedFastSpawnObject != null) {
            cachedFastSpawnObject.OnSpawn += OnSpawn;
            cachedFastSpawnObject.OnUnspawn += OnUnspawn;
        }
    }

    void OnDestroy() {
        if(cachedFastSpawnObject != null) {
            cachedFastSpawnObject.OnSpawn -= OnSpawn;
            cachedFastSpawnObject.OnUnspawn -= OnUnspawn;
        }
    }
```

```
    void OnSpawn() {
        if(cachedParticleSystem != null) {
            cachedParticleSystem.Play();
        }
    }

    void OnUnspawn() {
        if(cachedParticleSystem != null) {
            if(cachedParticleSystem.IsAlive()) {
                cachedParticleSystem.Stop();
            }

            cachedParticleSystem.Clear();
        }
    }

    void LateUpdate() {
        if(cachedParticleSystem != null) {
            if(!cachedParticleSystem.IsAlive()) {
                // Use singleton reference for easier access
                if(MySpawnManager.SharedInstance != null && cachedFastSpawnObject != null) {
                    MySpawnManager.SharedInstance.Unspawn(cachedFastSpawnObject);
                }
            }
        }
    }
}
```

The particle system component may be a child of the spawn object or directly attached to it. The particle system gets automatically unspawned when the particle system is no longer alive.

Next you might be wondering what if you have multiple particle systems inside a Fast Spawn object. The following example helps you to solve that problem. Just define the parent particle systems to the parentParticleSystems field. Incase you use sub emitters, leave them out since the parent particle systems will control their visibility.

```
using UnityEngine;
using System.Collections;

public class SpawnableParticleSystems : MonoBehaviour {
    private FastSpawnObject cachedFastSpawnObject;
    public ParticleSystem[] parentParticleSystems;

    void Awake() {
        cachedFastSpawnObject = GetComponent<FastSpawnObject>();

        if(cachedFastSpawnObject != null) {
            cachedFastSpawnObject.OnSpawn += OnSpawn;
            cachedFastSpawnObject.OnUnspawn += OnUnspawn;
```

```
        }
    }

    void OnDestroy() {
        if(cachedFastSpawnObject != null) {
            cachedFastSpawnObject.OnSpawn -= OnSpawn;
            cachedFastSpawnObject.OnUnspawn -= OnUnspawn;
        }
    }

    void OnSpawn() {
        foreach(ParticleSystem particleSystem in parentParticleSystems) {
            particleSystem.Play();
        }
    }

    void OnUnspawn() {
        foreach(ParticleSystem particleSystem in parentParticleSystems) {
            if(particleSystem.IsAlive()) {
                particleSystem.Stop();
            }

            particleSystem.Clear();
        }
    }

    void LateUpdate() {
        foreach(ParticleSystem particleSystem in parentParticleSystems) {
            if(particleSystem.IsAlive()) {
                return;
            }
        }

        // If all the particle systems are stopped, unspawn this spawn object
        if(MySpawnManager.SharedInstance != null && cachedFastSpawnObject != null) {
            MySpawnManager.SharedInstance.Unspawn(cachedFastSpawnObject);
        }
    }
}
```

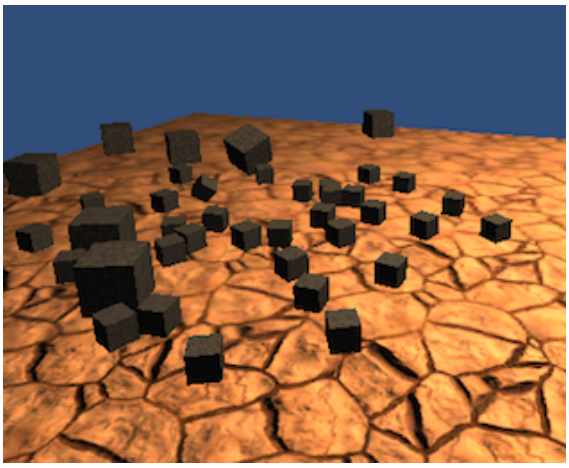
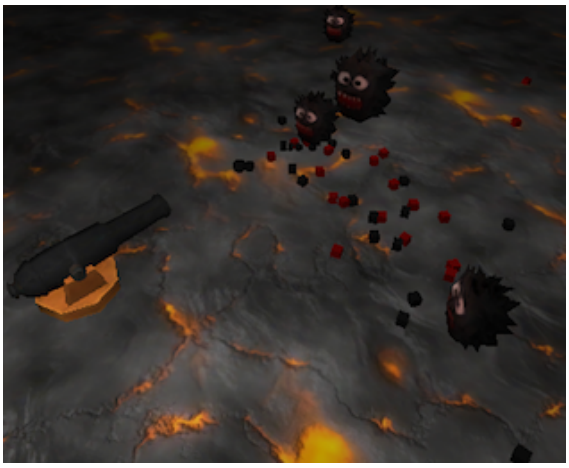
Tips

- Eventhough the Fast Spawn manager prefab container was designed to be used for spawning you can use the same container for none spawn objects too, for example for holding references to different player characters etc. Instead of defining FastSpawnObject playerCharacterOption1 you can define it as Transform playerCharacterOption1 and then instantiate it manually with the Instantiate function. It helps you to keep your prefabs organized.
- Always use the same name for the prefab and for the corresponding Fast Spawn manager item to keep things organized.
- Create helper methods like the 'SpawnRandomMaleOrFemaleZombie' above to make the spawning

even easier.

Provided example scenes

There are two different example scenes inside the Fast Spawn package, CubeFighter and BogeyManCannon.

Support

Incase you are having problems with Fast Spawn, don't hesitate to contact **support@pmjo.org**