EZ ENTERTAINMENT

POOLY

Owner's Manual

v 2.0

Thank you for buying our asset and for supporting its further development. This plugin was created to extend the functionality of Unity's native system. Should you need help, find issues or have any suggestions, don't hesitate to send us a message at support@ezentertainment.eu

Please read the quick setup guide and check out our video tutorials before you start using this asset.

Solutions – Pooly solves the following problems

- Instantiating and destroying a lot of items (such as bullets or enemies) is inefficient and can slow down your projects. A professional way of handling this issue is by using a technique known as object pooling.
- Pooly is a professional pooling system that gives you all the tools you need in order to implement and manage the spawning and despawning of a lot of objects at once.
- The system is easy to understand and quick to integrate, allowing you to configure how objects are pooled, how they are spawned and how they get despawned.
- This All-In-One pool management system allows you to create scene-based pools, comes with a multipurpose object despawner and also with an intelligent object spawning system. All the components have clean custom inspectors that will help you configure them in just a few clicks.
- Pooly works in 2D and 3D space, being optimized for all the mobile platforms.

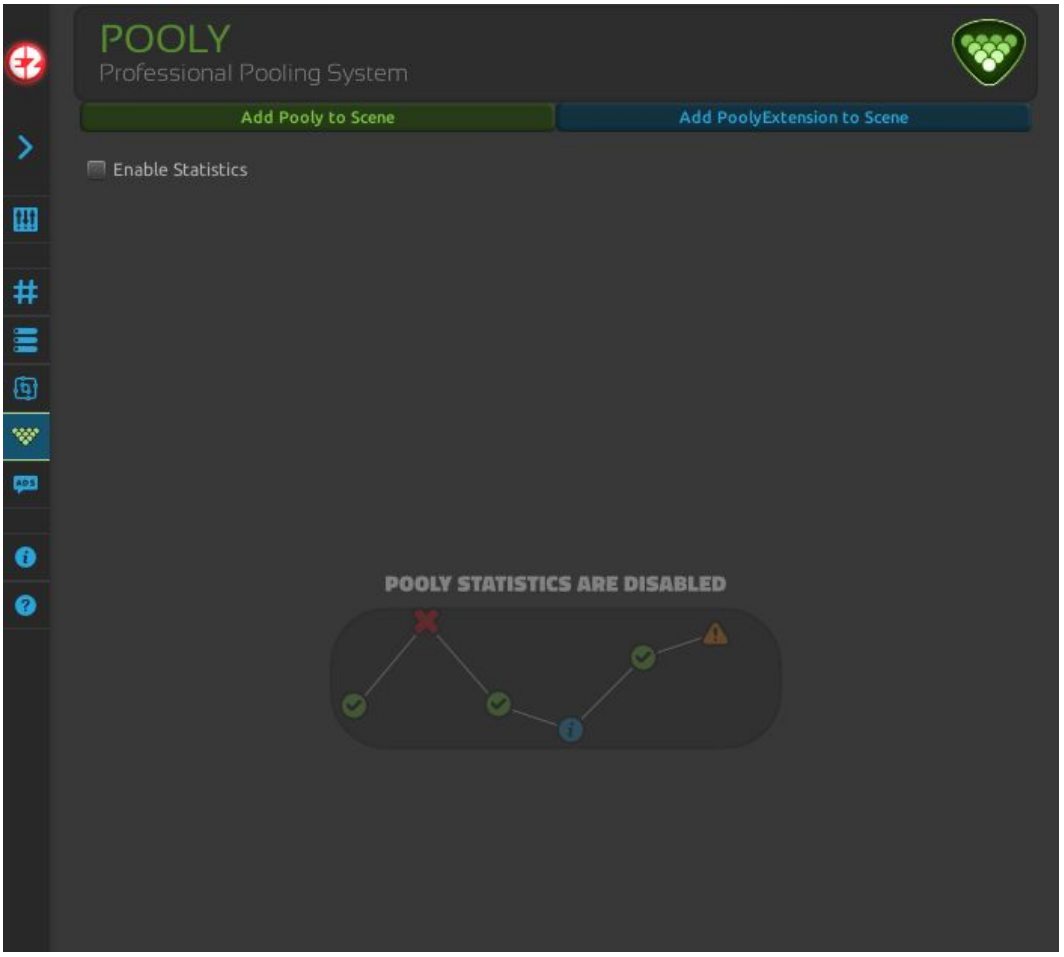
Quick Setup Guide

1. Import Pooly (from @UnityAssetStore).
2. In the top toolbar → Tools → Ez → Control Panel
3. Click on “Pooly”
4. Click on “Add Pooly to Scene”
5. Add the prefabs you want to the pool.
6. Done!

Watch the introduction and tutorial videos on our YouTube channel:
https://www.youtube.com/playlist?list=PLRE6VXhDQg2Ma8wVOoYgCf5mxIVvWyxUd

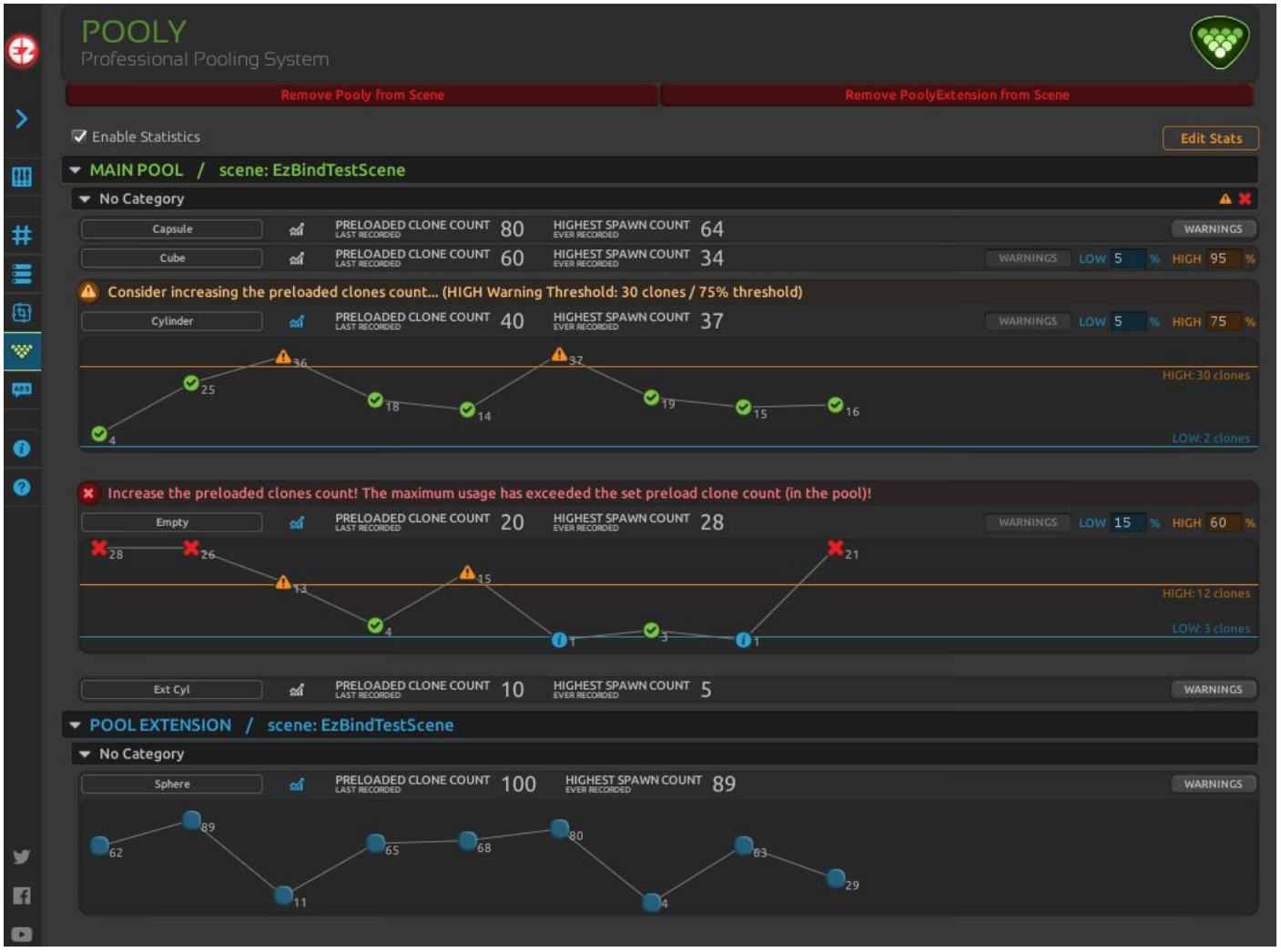
Pooly Window

The Pooly Window can be opened from the Editor toolbar Ez → Control Panel. In the Control Panel window, select “Pooly”. Using it you can add or remove Pooly’s main pool or PoolyExtension to and from your currently opened scene.
Pooly’s main pool GameObject is a singleton and should be added only in your main scene (or start scene) as it will persist across scenes.



This is the Pooly window when the statistics for prefab usage are disabled.

Pooly Statistics



To help figure out the actual usage (spawn count) of prefabs in a project Pooly can collect statistics about the number of spawned clones for each individual prefab.

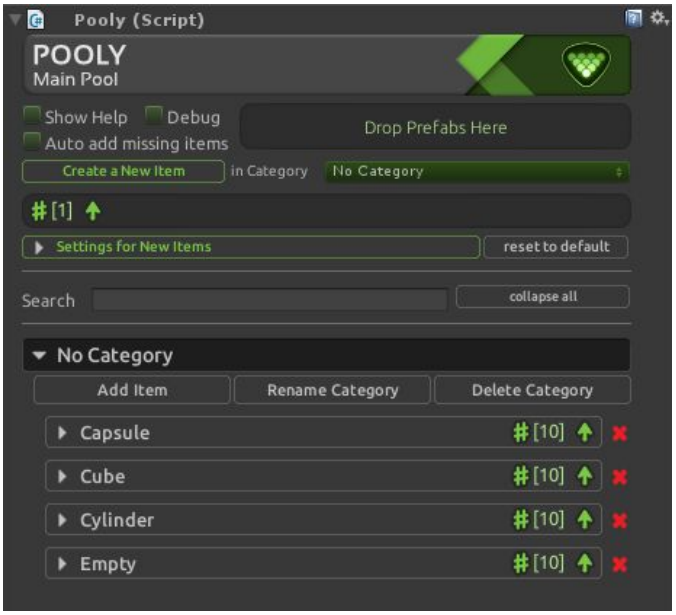
Note: The statistics will only be collected when running the game in the Unity Editor, they will never be included or collected in a build.

Statistics are collected per pool type and per scene:

- Pooly Main Pool:
    - all prefabs in the main pool are grouped together in statistics
    - one record for the main pool is generated per play session, regardless of scene changes
- Pooly Extension:
    - all prefabs in extensions are grouped together based on the scene they have been spawned in.
    - every time an extension is loaded (via scene change), a new statistics record is recorded
    - usage of the same prefab under different extensions in different scenes is recorded separately
- Maximum history size is of 12 records / pool; when the limit is reached, the oldest record is deleted to accommodate the new record
- Maximum usage for a prefab is recorded separately; even if the record for maximum usage is overwritten, the value is not lost
- Warnings for high or low usage can be enabled and configured for each individual prefab; based on these settings, the system makes it easy to find prefabs that are either over-provisioned or under-provisioned
- Statistics can be reset per pool, per category or per item and, if needed, individual records can be deleted

Pooly Main Pool Component

This is the Pooly Main Pool GameObject. Here you can add the prefabs that you need to have available at all times. Pooly's Main Pool is a singleton. This means that it will not be destroyed and will be available for use even if you change scenes.



Editor inspector overview:

- Show help - when enabled shows detailed help in the inspector for all elements.
- Debug - when enabled, information and debug messages are printed in the console at runtime.
- Auto add missing items - when checked, Pooly will automatically instantiate clones of prefabs that were not added to the pool via inspector. This happens only when trying to spawn a clone of a prefab referenced by a Transform (not by its name string).
- Create a New Item - add a prefab to the pool, in the specified category and with the currently configured settings.
- Drop Prefabs here - drop zone for quickly adding prefabs to the pool in bulk. These items will use the selected category and settings.
- Settings for New Items - the settings applied when new prefabs are added to the pool
- Search - search bar that facilitates finding prefabs in the pool. Accepts regular expressions (regex).
- Below the search bar the inspector will show the configured categories, their items and configurations.

API and Code Examples

To be able to use Pooly in your scripts, remember to add:

```
using Ez.Pooly;
```

Spawning

Pooly offers several methods that can be used to spawn a clone of a prefab. All Pooly.Spawn() methods return a Transform reference of the spawned clone.

The following methods spawn a clone inside the pool (the parent of the spawned GameObject will be Pooly) and return the Transform's reference:

- Spawns a clone of the prefab referenced via its Transform, in the desired position and with the rotation specified via Quaternion:

```
Pooly.Spawn(Transform prefab, Vector3 position, Quaternion rotation)
```

- Spawns a clone of the prefab referenced via its Transform, in the desired position, with the rotation specified via Quaternion and with the specified localScale:

```
Pooly.Spawn(Transform prefab, Vector3 position, Vector3 rotation, Vector3 localScale)
```

- Spawns a clone of the prefab referenced via its Transform, in the desired position and with the rotation specified via Vector3 Euler Angles:

```
Pooly.Spawn(Transform prefab, Vector3 position, Vector3 rotation)
```

- Spawns a clone of the prefab referenced via its Transform, in the desired position, with the rotation specified via Vector3 Euler Angles and with the specified localScale:

```
Pooly.Spawn(Transform prefab, Vector3 position, Vector3 rotation, Vector3 localScale)
```

- Spawns a clone of the prefab specified by name, in the desired position and with the rotation specified via Quaternion:

```
Pooly.Spawn(string itemName, Vector3 position, Quaternion rotation)
```

- Spawns a clone of the prefab specified by name, in the desired position, with the rotation specified via Quaternion and with the specified localScale:

```
Pooly.Spawn(string itemName, Vector3 position, Quaternion rotation, Vector3 localScale)
```

- Spawns a clone of the prefab specified by name, in the desired position and with the rotation specified via Vector3 Euler Angles:

```
Pooly.Spawn(string itemName, Vector3 position, Vector3 rotation)
```

- Spawns a clone of the prefab specified by name, in the desired position, with the rotation specified via Vector3 Euler Angles and with the specified localScale:

```
Pooly.Spawn(string itemName, Vector3 position, Vector3 rotation, Vector3 localScale)
```

The following methods spawn a prefab clone, reparent it under the specified parent Transform and return the Transform's reference:

- Spawns a clone of the prefab referenced via its Transform, in the desired position and with the rotation specified via Quaternion then reparents it under parent's Transform:

```
Pooly.Spawn(Transform prefab, Vector3 position, Quaternion rotation, Transform parent)
```

- Spawns a clone of the prefab referenced via its Transform, in the desired position, with the rotation specified via Quaternion and with the specified localScale and reparents it under the parent Transform:

```
Pooly.Spawn(Transform prefab, Vector3 position, Quaternion rotation, Vector3 localScale, Transform
parent)
```

- Spawns a clone of the prefab referenced via its Transform, in the desired position and with the rotation specified via Vector3 Euler Angles then reparents it under parent's Transform:

```
Pooly.Spawn(Transform prefab, Vector3 position, Vector3 rotation, Transform parent)
```

- Spawns a clone of the prefab referenced via its Transform, in the desired position, with the rotation specified via Vector3 Euler Angles and with the specified localScale and reparents it under the parent Transform:

```
Pooly.Spawn(Transform prefab, Vector3 position, Vector3 rotation, Vector3 localScale, Transform parent)
```

- Spawns a clone of the prefab specified by name, in the desired position and with the rotation specified via Quaternion then reparents it under parent's Transform:

```
Pooly.Spawn(string itemName, Vector3 position, Quaternion rotation, Transform parent)
```

- Spawns a clone of the prefab specified by name, in the desired position and with the rotation specified via Quaternion and with the specified localScale and reparents it under the parent Transform:

```
Pooly.Spawn(string itemName, Vector3 position, Quaternion rotation, Vector3 localScale, Transform
parent
```

- Spawns a clone of the prefab specified by name, in the desired position and with the rotation specified via Vector3 Euler Angles then reparents it under parent's Transform:

```
Pooly.Spawn(string itemName, Vector3 position, Vector3 rotation, Transform parent)
```

- Spawns a clone of the prefab specified by name, in the desired position and with the rotation specified via Vector3 Euler Angles and with the specified localScale and reparents it under the parent Transform:

```
Pooly.Spawn(string itemName, Vector3 position, Vector3 rotation, Vector3 localScale, Transform parent)
```

Despawning

Pooly offers the following methods for despawning clones:

- Despawns the referenced clone and returns it to the pool:

```
Pooly.Despawn(Transform clone)
```

- Despawns all the clones of the prefab referenced via Transform:

```
Pooly.DespawnAllClonesOfPrefab(Transform prefab)
```

- Despawns all the clones of the prefab with the specified name:

```
Pooly.DespawnAllClonesOfPrefab(string prefabName)
```

- Despawns all the clones of all the prefabs in the specified category:

```
Pooly.DespawnAllClonesInCategory(string category)
```

- Despawns all the clones of all the prefabs from the referenced Pooly Extension in the specified category:

```
Pooly.DespawnAllClonesInCategory(PoolyExtension poolExtension, string category)
```

- Despawns all the clones of all the prefabs from the referenced Pooly Extension:

```
Pooly.DespawnAllClonesFromPoolExtension(PoolyExtension poolExtension)
```

- Despawns all the clones of all the prefabs from all pools:

```
Pooly.DespawnAllClones()
```

OnSpawned() and OnDespawned()

To allow simple clone management, when pooled clones are spawned and despawned, Pooly automatically invokes the OnSpawned() and respectively OnDespawned() methods on the clone's components.

In order for a script to handle the spawning/despawning of its GameObject, just add the following methods to it:

```
    void OnSpawned()
    {
        // Invoked when the clone is spawned
        // Your code here
    }
```

```
    void OnDespawned()
    {
        // Invoked when the clone is despawned
        // Your code here
    }
```

Clone Information

If needed, there are several helper methods that can be used to find information regarding the pooled clones at runtime:

- To verify if a prefab has any spawned clones:

```
Pooly.HasActiveClones(Transform prefab)
Pooly.HasActiveClones(string itemName)

// Returns true is the prefab has active (spawned) clones, false otherwise
```

- To find the number of spawned clones of a prefab:

```
Pooly.GetActiveCloneCount(Transform prefab)
Pooly.GetActiveCloneCount(string itemName)

// Returns the number of active (spawned) clones for the prefab.
```

- To verify if a prefab has disabled (despawned) clones available:

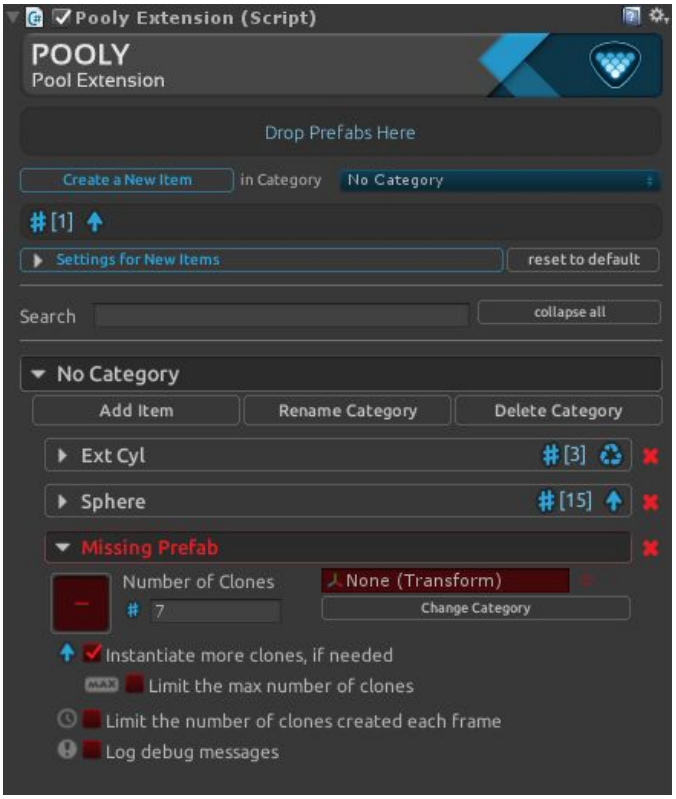
```
Pooly.HasDisabledClones(Transform prefab)
Pooly.HasDisabledClones(string itemName)

// Returns true is the prefab has disabled (despawned) clones available.
```

- To find the number of despawned clones in the pool for a certain prefab:

```
Pooly.GetDisabledCloneCount(Transform prefab)
Pooly.GetDisabledCloneCount(string itemName)

// Returns the number of disabled (despawned) clones available for the prefab.
```

Pooly Extension Component

Pooly extension allows you to extend pooling by adding temporary, scene-based object pools. The items pooled in an extension are available as long as the extension itself is active and enabled.

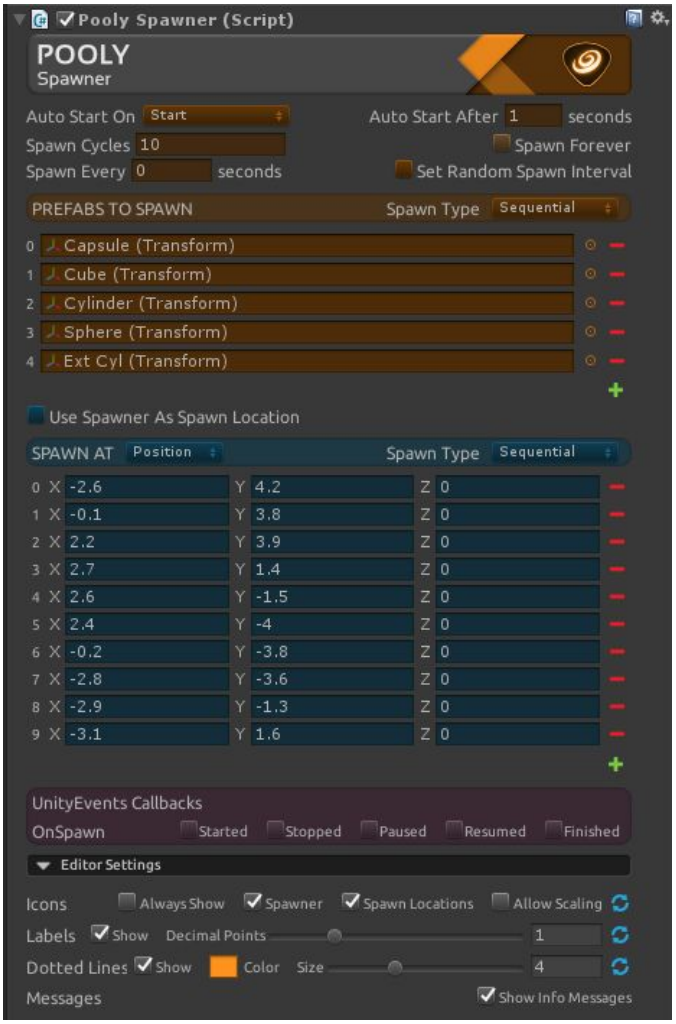


Editor inspector overview:

- Show help - when enabled shows detailed help in the inspector for all elements.
- Create a New Item - add a prefab to this pool extension, in the specified category and with the currently configured settings.
- Drop Prefabs here - drop zone for quickly adding prefabs in bulk. These items will use the selected category and settings.
- Settings for New Items - the settings applied when new prefabs are added to the pool
- Search - search bar that facilitates finding prefabs in this pool extension. Accepts regular expressions (regex).
- Below the search bar the inspector will show the configured categories, their items and configurations for this pool extension only.

Pooly Spawner

The Pooly Spawner is an automated spawning solution that is meant to make the spawning of clones a simple and automated process.



Editor inspector overview:

- Auto Start - allows the spawner to automatically start spawning
    - Automatic spawning can be based on the spawner's Start() or OnSpawned() methods or can be started manually.
    - Allows setting an initial delay to spawning (when starting automatically)
- Spawn cycles - how many clones the spawner should spawn. Can be set to spawn forever
- Spawn interval - the delay between spawning two clones. Can be a set value, or a random interval based on a min-max range
- Extra Spawn Settings - available when using the Spawner or a list of Transforms as spawn target(s):
    - Match Target Transform Rotation - the spawned clone is rotated to match the target
    - Match Target Transform Scale - the spawned clone is scaled to match the target
    - Reparent Under Target Transform - the spawned clone is reparented under the target Transform
- Spawn prefabs - the objects whose clones the spawner should spawn. When multiple prefabs are defined, the next clone to be spawned can be selected either sequentially or randomly (based on configurable weights).
- Spawn Location - location(s) where the clones should be spawned. Can be set to:
    - Spawner's own location.
    - Specified points in the scene defined as Vector3s
    - Specified Transforms in the scene
    - When multiple spawn locations are defined (either Vector3s or Transforms), the next spawn location can be selected either sequentially or randomly (based on configurable weights).
- UnityEvent Callbacks - callbacks based on UnityEvents available when:
    - Spawning starts
    - Spawning is stopped
    - Spawning is paused
    - Spawning is resumed
    - Spawning is finished (the last spawn cycle is completed)

- Editor settings - persistent settings that affect how all spawners are displayed in the Editor's Scene View.

API and Code Examples

To be able to use the Pooly Spawner in your scripts, remember to add:

```
using Ez.Pooly;
```

In order to be able to access the methods of a Pooly Spawner it is necessary to have a reference to it:

```
PoolySpawner spawner;
```

To dynamically add or remove prefabs to/from the spawner:

```
spawner.AddPrefabToSpawner(Transform prefab, int weight = 100);
// Adds a new prefab to the spawner
// With an optionally defined weight (used if spawning random clones of the defined prefabs)

spawner.RemovePrefabFromSpawner(Transform prefab);
// Removes a prefab from the spawner. If the prefab is not found, nothing happens.

spawner.ReplacePrefabFromSpawner(Transform oldPrefab, Transform newPrefab)
// Replaces a prefab that is referenced in the spawner with a another one.
// If either prefab is null or the oldPrefab is not found, nothing happens.
```

To change the spawn chance (weight) of a prefab:

```
spawner.UpdateSpawnChanceForPrefab(Transform prefab, int newWeight)
```

To update the spawn chance (weight) for a spawn position or a spawn location (Transform):

```
spawner.UpdateSpawnChanceForSpawnPosition(int positionIndex, int newWeight)

spawner.UpdateSpawnChanceForSpawnTransform(Transform spawnPoint, int newWeight)
```

To manually start the spawning process of the spawner:

```
spawner.StartSpawn();

spawner.StartSpawn(float spawnStartDelay);
// spawnStartDelay specifies the initial delay before spawning starts
```

To stop the spawning cycle:

```
spawner.StopSpawn(bool resetSpawnedCount = true);
// Stops the spawn cycle. If resetSpawnedCount is set to false, this function will act as PauseSpawn
and won't not reset the spawnedCount counter.
```

To pause the spawning cycle:

```
spawner.PauseSpawn();
```

To resume the spawning cycle:

```
spawner.ResumeSpawn();
```

To spawn the next prefab in the cycle immediately, use the SpawnNext() methods:

```
spawner.SpawnNext();
// Spawns a prefab at one of the spawn locations if spawnForever is enabled or the spawning had not finished yet.

spawner.SpawnNext(Transform prefab);
// Spawns the specified prefab at one of the spawn locations if spawnForever is enabled or the spawning had not finished yet.
```

To spawn the next prefab in the cycle immediately and get a reference to the spawned clone, use SpawnNextAndGetReference():

```
spawner.SpawnNextAndGetReference();
// Spawns a prefab at one of the spawn locations if spawnForever is enabled or the spawning had not finished yet. Returns the Transform reference of the spawned clone.

spawner.SpawnNextAndGetReference(Transform prefab);
// Spawns the specified prefab at one of the spawn locations if spawnForever is enabled or the spawning had not finished yet. Returns the Transform reference of the spawned clone.
```

To check if the spawner can spawn another clone:

```
bool spawner.CanSpawn;
// Read-only property - returns true if spawnForever is enabled or if the spawning hasn't finished.
```

Callbacks are available via UnityEvents when spawning is started, stopped, paused, resumed or finished. To manually add or remove spawner callback listeners:

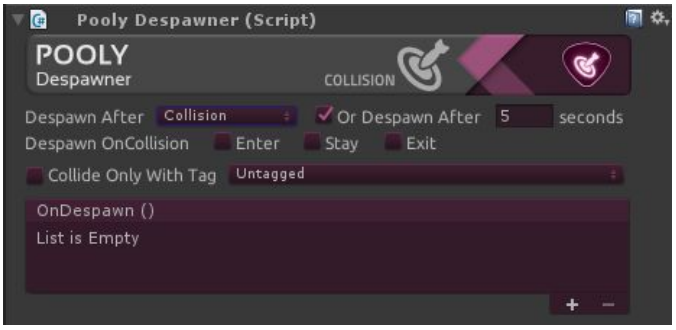
```
spawner.AddOnSpawnCallbackListener(OnSpawnEvent onSpawnEvent, UnityAction listener)

//Example:
// public void MyListener() { /* code here */ }
// spawner.AddOnSpawnCallbackListener(PoolySpawner.OnSpawnEvent.OnSpawnStarted, MyListener);

spawner.RemoveOnSpawnCallbackListener(OnSpawnEvent onSpawnEvent, UnityAction listener)

//Example:
// public void MyListener() { /* code here */ }
// spawner.AddOnSpawnCallbackListener(PoolySpawner.OnSpawnEvent.OnSpawnStarted, MyListener);
```

Pooly Despawner

The Pooly Despawner is a simple to use solution that automates the despawning of prefab clones, based on specific situations.

Despawn After Collision

With this setting, the despawner script will automatically despawn its GameObject when the Collision conditions are met.

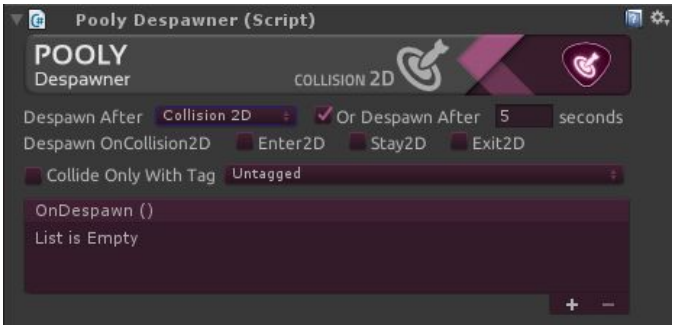


Editor inspector overview:

- Despawn After - Collision:
  - Sets a Collision event as the despawning trigger
  - Optionally, a maximum lifetime can be defined
- Despawn OnCollision - select the type of Collision:
  - Enter
  - Stay
  - Exit
- Collide Only with Tag - despawning will only be triggered by Collision events with a GameObject having the specified Tag.
- OnDespawn - UnityEvent invoked immediately before despawning.

Despawn After Collision2D

With this setting, the despawner script will automatically despawn its GameObject when the Collision2D conditions are met.

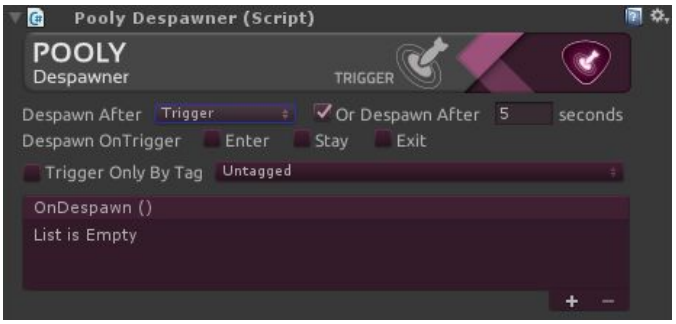


Editor inspector overview:

- Despawn After - Collision2D:
  - Sets a Collision2D event as the despawning trigger
  - Optionally, a maximum lifetime can be defined
- Despawn OnCollision - select the type of Collision2D:
  - Enter2D
  - Stay2D
  - Exit2D
- Collide Only with Tag - despawning will only be triggered by Collision events with a GameObject having the specified Tag.
- OnDespawn - UnityEvent invoked immediately before despawning.

Despawn After Trigger

With this setting, the despawner script will automatically despawn its GameObject when the Trigger conditions are met.

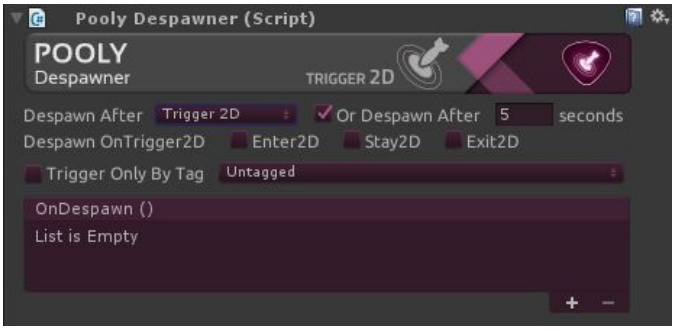


Editor inspector overview:

- Despawn After - Trigger:
  - Sets a Trigger event as the despawning trigger
  - Optionally, a maximum lifetime can be defined
- Despawn OnTrigger - select the type of Trigger event:
  - Enter
  - Stay
  - Exit
- Collide Only with Tag - despawning will only be triggered by Trigger events with a GameObject having the specified Tag.
- OnDespawn - UnityEvent invoked immediately before despawning.

Despawn After Trigger2D

With this setting, the despawner script will automatically despawn its GameObject when the Trigger2D conditions are met.

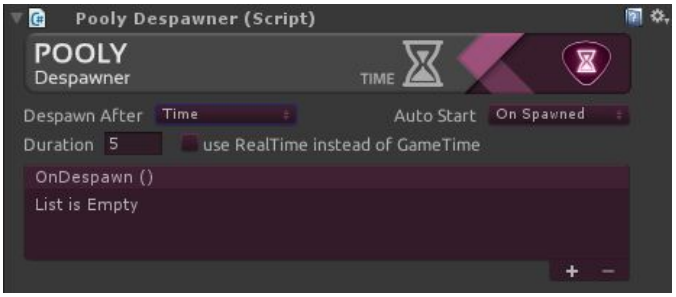


Editor inspector overview:

- Despawn After - Trigger2D:
  - Sets a Trigger2D event as the despawning trigger
  - Optionally, a maximum lifetime can be defined
- Despawn OnTrigger - select the type of Trigger2D event:
  - Enter2D
  - Stay2D
  - Exit2D
- Collide Only with Tag - despawning will only be triggered by Trigger2D events with a GameObject having the specified Tag.
- OnDespawn - UnityEvent invoked immediately before despawning.

Despawn After Time

With this setting, the despawner script will automatically despawn its GameObject after the specified duration.



Editor inspector overview:

- Despawn After - Time
- Auto Start:
    - OnSpawned - the countdown for despawning is automatically started when this GameObject is spawned
    - Never - the countdown must be started manually by an external script
- Duration - lifetime of the GameObject, in seconds
- Use RealTime instead of GameTime - can be used to make the despawning independent of timescale.
- OnDespawn - UnityEvent invoked immediately before despawning.

API and Code Examples

To be able to use the Pooly Despawner in your scripts, remember to add:

```
using Ez.Pooly;
```

In order to be able to access the methods of a Pooly Despawner, it is necessary to have a reference to it:

```
PoolyDespaner despawner;
```

To manually start the despawning timer:

```
despawner.StartTimer(float duration = 0);
// Starts a despawn timer with a set duration.
// If the duration is less than or equal to zero, it will trigger an instant despawn.

despawner.StartTimer(float durationMinimum, float durationMaximum);
// Starts a despawn timer with a random duration.
// If durationMinimum and durationMaximum are less than or equal to zero, or if durationMinimum is
greater than durationMaximum, it will trigger an instant despawn.
```

Despawn After Sound Played

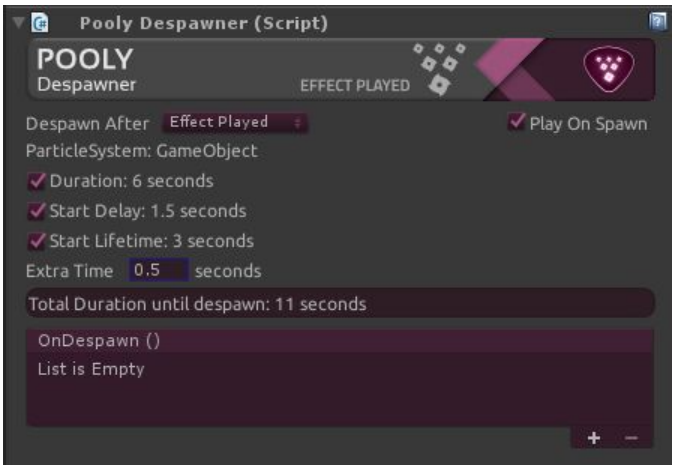
With this setting, the despawner script will automatically despawn its GameObject after its associated sound has played. Useful when you want to automatically spawn, play then despawn sound clips.

Editor inspector overview:

- Despawn After - Sound Played
- Play On Spawn - automatically play the AudioClip when this GameObject is spawned.
- Audio Source - information
    - AudioSource - the owner
    - AudioClip - source audio clip
    - Duration - the audio clip's duration
- OnDespawn - UnityEvent invoked immediately before despawning.

Despawn After Effect Played

With this setting, the despawner script will automatically despawn its GameObject after its associated effect has played. Useful when you want to automatically spawn, play then despawn particle effects.



Editor inspector overview:

- Despawn After - Effect Played.
- Play On Spawn - automatically play this effect.
- Particle System - information:
    - Particle System - the owner
    - Duration - Total Duration takes into account the ParticleSystem Duration
    - Start Delay - Total Duration takes into account the ParticleSystem Start Delay
    - Start Lifetime - Total Duration takes into account the ParticleSystem Start Lifetime
- Extra time - increase the Total Duration by this amount.
- Total Duration - duration, in seconds, before the GameObject is despawned.
- OnDespawn - UnityEvent invoked immediately before despawning.