AutoPool 1.0 - Script Reference

Get the most up to date information and tutorials at:
Mobfarmgames.weebly.com

Script Reference

In most cases, these scripts will be added to the appropriate objects automatically, but some efficiency can be realized by placing them manually ahead of time.

- AP_Pool
  This script provides the settings for individual object pools.
- AP_Manager
  Provides references and will interface with the various pools, and controls how other scripts may create and modify object pools. Placed as a parent object of the object pools it will manage.
- AP_Reference
  Will be automatically placed on objects in the pool. It allows an object to remember when it was spawned and to which pool it belongs.
- MF_StaticAutoPool
  A class that provides the static functions used to interface with the object pool manager. This is kept in the scripts folder and does not need to be attached to an object.
  MF_AutoPool:
  InitializeSpawn(), **Spawn**(), **Despawn**(), **GetActiveCount**(), **GetAvailableCount**(), **DespawnPool**(), **DespawnAll**(), **RemovePool**(), **RemoveAll**()

AP_Pool

The script attached to individual pools.

Public Variables

poolBlock : PoolBlock
This is a wrapper containing the object pool fields. Using this wrapper allows multi-editing of object pools in the inspector.

Public Class PoolBlock

prefab : GameObject
The prefab that this pool contains.

size : int
The starting size of the object pool.

emptyBehaviour : AP_enum.EmptyBehaviour enum { Grow, Fail, ReuseOldest }
How the pool behaves when the pool is empty and a spawn is requested.

Grow: Will instantiate a new object in the pool if the pool is less than *maxSize*. This will be slowest if the pool has to grow by many items, but it is the most flexible.

Fail: No spawn will be created at this time. Best for performance, but the least flexible.

ReuseOldest: Will find the oldest active object, and reuse it as a new spawn. Useful for certain applications, but can be quite slow with large pool sizes, as it has to check every item to find the oldest one. If the pool is empty only occasionally, consider using Grow, or starting with a larger pool.

maxSize : int
The maximum size to which EmptyBehaviour.Grow can enlarge the pool.
This field is hidden when EmptyBehaviour.Grow is not selected.

maxEmptyBehaviour : AP_enum.MaxEmptyBehaviour enum { Fail, ReuseOldest }
How the pool behaves when the pool is at *maxSize*, is empty, and a spawn is requested.
This field is hidden when EmptyBehaviour.Grow is not selected.

Fail: No spawn will be created at this time. Best for performance, but the least flexible.

ReuseOldest: Will find the oldest active object, and reuse it as a new spawn. Useful for certain applications, but can be quite slow with large pool sizes, as it has to check every item to find the oldest one. If the pools is empty only occasionally, consider using Grow, or starting with a larger pool.

printLogOnQuit : bool
When exiting the player, prints a pool usage summary in the console for development and debugging.
Start Size: Size of pool when beginning the scene.

Init Added: Number of objects added by InitializeSpawn() calls at runtime.
Grow Objects: Number of objects added with EmptyBehaviour.Grow.
End Size: Total objects of this pool, active and inactive, at the time of the log report.
Failed Spawns: Number of Spawn() requests that didn't return a spawn.
Reused Objects: Number of times an object was reused before despawning normally.
Most objects active at once: The most items for this pool active at once.

Inspector Buttons

Some tools for development and debugging.

Print Log
Prints the current usage log to the console.

Spawn
Request a spawn from this pool.

Public Methods

Normally, you would call the spawn method using MF_AutoPool.Spawn(). However, you can also access a specific pool directly using the following methods:
Spawn : GameObject ( )
Spawn : GameObject (*child* : int?)
Spawn : GameObject (*pos* : Vector3, *rot* : Quaternion)
Spawn : GameObject (*child* : int?, *pos* : Vector3, *rot* : Quaternion)
Spawns an object from this pool at *pos* position and *rot* rotation.
if *pos* and *rot* are unused, the position and rotation will be that of the pool object.
Optionally, a *child* index can be specified for prefabs containing children. That child will be enabled, and the others will remain disabled. This feature can be used to simulate different types of objects.

Hidden Variables

pool : Stack { PoolItem }
Items in the pool available for spawning.

masterPool : List { PoolItem }
Used to track all items belonging to the pool, both active and inactive.

Public Class PoolItem

obj : GameObject
The root object of a pool item.

refScript : AP_Reference
A cached reference to the AP_Reference script on *obj*.

AP_Manager

Attached to the parent object of the object pools.

Public Variables

allowCreate : bool
Will allow scripts using MF_AutoPool.InitializePool() to create new object pools.

allowModify : bool
Will allow scripts using MF_AutoPool.InitializePool() to modify existing object pools. Note, that InitializePool() will always be allowed to modify a pool it is in the process of creating.

printAllLogsOnQuit : bool
When exiting the player, prints an usage summary for all pools in the console for development and debugging.
Start Size: Size of pool when beginning the scene.
Init Added: Number of objects added by InitializeSpawn() calls at runtime.
Grow Objects: Number of objects added with EmptyBehaviour.Grow.
End Size: Total objects of this pool, active and inactive, at the time of the log report.
Failed Spawns: Number of Spawn() requests that didn't return a spawn.
Reused Objects: Number of times an object was reused before despawning normally.
Most objects active at once: The most items for this pool active at once.

Inspector Buttons

Create Pool
This is just a shortcut for editor usage. It simply creates a child object with the AP_Pool script.

Print Log
Prints the current usage log for all pools to the console.

Hidden Variables

poolRef : Dictionary { *key* : GameObject, *value* AP_Pool }
Stores references to the various pool scripts by the prefab they contain.

AP_Reference

Automatically placed at the root level on items in an object pool. Alternately, may be placed by hand ahead of time, allowing the use of the *delay* variable, and slightly quicker execution during runtime.

Public Variables

delay : float
When this unit despawns, there will be a *delay* before it returns to the pool. This can be useful to give other scripts time to adjust in case the item were to immediately respawn.

Hidden Variables

poolScript : AP_Pool
The pool that this item belongs to.

timeSpawned : float
When this item was spawned.

MF_StaticAutoPool

The main interface for using the object pools. In all cases, object pools are referenced by the prefab they contain.

Example usage from anywhere in the project:
MF_AutoPool.Spawn(*prefab*);
MF_AutoPool.Despawn(*object*);
prefab is a reference to the prefab to spawn, and *object* is a reference to a spawned object.

Public Class MF_AutoPool

Public Static Methods

InitializeSpawn : bool (*prefab* : GameObject)
Initializes the pool containing *prefab* and can add items to that pool's size.
This method is optional, and is used to ensure all pool reference links are created at the time this method is called. With heavy pool usage, this may reduce lag spikes by creating all references early in the scene.
InitializeSpawn : bool (*prefab* : GameObject, *addPool* : float, *minPool* : int)
InitializeSpawn : bool (*prefab* : GameObject, *addPool* : float, *minPool* : int, *emptyBehavior* AP_enum.EmptyBehavior, *maxEmptyBehavior* : AP_enum.MaxEmptyBehavior)
Additionally, this method can also be used to dynamically construct pools at runtime.
If the pool manager object AP_Manager in the scene is marked *allowCreate* and *allowModify* as true: Each time this method is called a pool will be constructed with a minimum of *minPool* objects. If *minPool* has already been reached, it will add *addPool* objects. If *minPool* was not 0, then the pool size will be increased to *minPool* + *addPool* items. If addPool is a float < 1, *addPool* will be that percentage of items already in the pool.
emptyBehavior and *maxEmptyBehavior* set the respective properties of the pool being created or modified.

If the pool manager *allowCreate* is false, pools cannot be created by InitializeSpawn().
If the pool manager *allowModify* is false, pools cannot be modified by InitializeSpawn(), however InitializeSpawn() is always allowed to modify a pool it is in the process of creating.

Spawn : GameObject (*prefab* : GameObject)
Spawn : GameObject (*prefab* : GameObject, *child* : int?)
Spawn : GameObject (*prefab* : GameObject, *pos* : Vector3, *rot* : Quaternion)
Spawn : GameObject (*prefab* : GameObject, *child* : int?, *pos* : Vector3, *rot* : Quaternion)
Spawns an object from the pool containing *prefab* at *pos* position and *rot* rotation.
if *pos* and *rot* are unused, the position and rotation will be that of the pool object.
Optionally, a *child* index can be specified for prefabs containing children. That child will be enabled, and the others will remain disabled. This feature can be used to simulate different types of objects.

Despawn : bool (*obj* : GameObject)
Despawn : bool (*obj* : GameObject, *time* : float)
Deactivates *obj* but waits the *delay* on its AP_Reference script before returning to the pool.
If *time* is specified and > 0, *time* will be used instead.
Returns false if the item could not be despawned.
These Despawn() methods are slightly slower than the methods below, because these must use *obj*.GetComponent<AP_Reference>() to find the script, whereas the ones below reference it directly.

Despawn : bool (*script* : AP_Reference)
Despawn : bool (*script* : AP_Reference, *time* : float)
Deactivates the object attached to *script* but waits the *delay* on AP_Reference script before returning to the pool.
If *time* is specified and > 0, *time* will be used instead.
Returns false if the item could not be despawned.
These methods are slightly faster than using the above Despawn() methods with a GameObject as a parameter.

GetActiveCount : int (*prefab* : GameObject)
Returns the number of active spawns in the scene from the pool containing *prefab*.

GetAvailableCount : int (*prefab* : GameObject)
Returns the number of unspawned items from the pool containing *prefab*.

DespawnPool : bool (*prefab* : GameObject)
Despawn all objects belonging to the pool containing *prefab*.
Will return false if *prefab* is null or the pool couldn't be found.

DespawnAll : bool ( )
Despawn all objects belonging to all pools, including currently active spawns.
Will return false if *prefab* is null or one of the pools couldn't be found.

RemovePool : bool (*prefab* : GameObject)
Destroy the pool containing *prefab* and all its items, including currently active spawns.
Will return false if *prefab* is null or the pool couldn't be found.

RemoveAll : bool ( )
Destroy all pools and all their items, including currently active spawns. The pool manager will remain intact.
Will return false if one of the pools couldn't be found.