Snog's Simple Wave System

Budget-driven, data-driven wave spawner for Unity 6.

Quick Start — get a working wave in ~5 minutes

1. **Create EnemyDefinition assets**
   - Project window: `Create → Snog → Simple Wave System → Enemy Definition`
   - Assign `prefab`, set `cost` (1 = cheap), set `weight` (relative chance). Set `minWave/maxWave` if needed.
2. **Create a WaveDefinition**
   - Project window: `Create → Snog → Simple Wave System → Wave Definition`
   - Add your `EnemyDefinition` assets to **Allowed Enemies**. Optionally add elite enemies and a boss.
3. **Scene: add a WaveSpawner**
   - Create empty GameObject → attach `WaveSpawner`.
   - Assign the `WaveDefinition` asset.
   - If you want specific spawn locations: set `Spawn Shape` to `PredefinedTransforms` and add child `Transform`s or use the provided `SpawnPoint` component on them.
   - Set `Target` (player) and `Reference Camera` (optional, for OutsideCamera spawn).
4. **(Optional) Pooling**
   - Add a `SimplePool` component (create empty GameObject and add component).

- On `WaveSpawner`, enable `usePooling` and link the `SimplePool`. Set `poolAutoExpand` depending on whether you want it to create new instances when empty.
- **Important:** When pooling, enemies must `SetActive(false)` on death or be returned to the pool. See “Pooling contract” below.

5. **Run**
    - Press Play. The WaveSpawner will generate waves from the `WaveDefinition` and start spawning.

Concepts & Design Overview

- **Budget-based composition** — each wave is generated from a numeric *budget*. Each `EnemyDefinition` has a `cost`. The spawner picks enemies until budget or max-enemy cap is hit.
- **Weight vs Cost** — `weight` affects selection probability; `cost` affects how many/which enemies can be bought by the budget.
- **Scaling by curves** — `WaveDefinition` uses `AnimationCurve`s to define budget, duration, spawn rate, and max enemies across wave numbers.
- **Special wave archetypes** — Rush / Tank / Swarm / Boss: special modifiers change budget, spawn rate, cost bias, etc.
- **Pacing** — spawn pacing curve lets spawn rate vary across wave progress.
- **Pooling (SimplePool)** — optional built-in pooling to reduce runtime allocations.

Editor / Asset Setup

EnemyDefinition

Fields:

- `prefab` (GameObject)
- `cost` (int >=1)
- `weight` (float >=0)
- `minWave` / `maxWave`

Tip: Keep `cost` and `weight` intuitive. Example:

- Grunt: cost=1 weight=10
- Heavy: cost=6 weight=1

`IsAvailableForWave(wave)` returns true only if prefab != null, cost>0, weight>0 and wave within min/max.

WaveDefinition

Key fields:

- `budgetCurve`, `durationCurve`, `spawnRateCurve`, `maxEnemiesCurve`, `maxAliveEnemiesCurve`
- `spawnPacingCurve` — multiply spawn rate across wave progress (0..1).
- `allowedEnemies`, `eliteEnemies`, `bossEnemy`.
- Special wave toggles and multipliers for Rush/Tank/Swarm/Boss.
- Global multipliers for budget/duration/spawn rate.

Design note: the system checks special wave types in a priority order (Boss → Tank → Swarm → Rush). If a wave matches multiple rules, the first matched wins — document that in your store page.

WaveSpawner (inspector highlights)

- `currentWave` — runtime wave index (starts at 1). Designer can set for testing.
- `waveDefinition` — required. If not assigned, the spawner will disable itself.
- `fallbackEnemies` — used if `waveDefinition.allowedEnemies` is empty.
- Spawn selection & shape fields (RoundRobin/Random, PredefinedTransforms / RandomInBox / RingAroundTarget).
- `target` & `referenceCamera` — used by some spawn modes.
- Pooling: `usePooling`, `pool`, `poolAutoExpand`.
- `treatInactiveAsDead` — if true, `spawnedEnemies` treats deactivated objects as dead (recommended when pooling).

API Reference (what you can call / listen to)

SimplePool (component)

- `void SetAutoExpand(bool value)` — sets `autoExpand`.
- `void Warmup(GameObject prefab, int count)` — pre-instantiates `count` copies of `prefab` and stores them.
- `void WarmupAll()` — runs warmup list defined in inspector.
- `GameObject Get(GameObject prefab, Vector3 position, Quaternion rotation)` — obtains an object from the pool (or instantiates if empty & autoExpand).
- `void Return(GameObject prefab, GameObject obj)` — returns a spawned object back to the pool.

Important: `Return` requires the original `prefab` reference. If your enemy cannot trivially know that, add a small `PooledObject` component (see

Integration Patterns).

WaveSpawner (component)

- Inspector events:
  - `onWaveStarted` (UnityEvent) and C# event `OnWaveStarted(int)`
  - `onWaveEnded` (UnityEvent) and C# event `OnWaveEnded(int)`
  - `onEnemySpawned` (UnityEvent) and C# event `OnEnemySpawned(GameObject)`
  - `onAllEnemiesDefeated` (UnityEvent) and C# event `OnAllEnemiesDefeated()`
- Read-only properties (getters):
  - `WaveDefinition WaveDefinition`
  - `IReadOnlyList<EnemyDefinition> FallbackEnemies`
  - `SpawnShape CurrentSpawnShape`
  - `SpawnMode CurrentSpawnMode`
  - `IReadOnlyList<Transform> SpawnPoints`
  - `Bounds SpawnBox`
  - `float RingInnerRadius`
  - `float RingOuterRadius`
  - `Transform Target`
  - `Camera ReferenceCamera`
  - `bool UsePooling`
  - `SimplePool Pool`

WaveSpawner handles:

- Generating the wave queue from budget & enemy pools.
- Pacing spawns via `spawnTimer` and `spawnRate`.

- Tracking `spawnedEnemies` list and cleaning null/inactive depending on `treatInactiveAsDead`.
- Raising events for wave/enemy lifecycle.

Integration Patterns & Examples

Recommended: PooledObject helper

To avoid passing `prefab` every time on return, add a tiny component to pooled instances:

`public class PooledObject : MonoBehaviour { public GameObject prefab; }`

Set `po.prefab = prefab;` in `SimplePool.CreateInstance()` when instantiating. Then provide an overload:

```
public void Return(GameObject obj)
{
    var po = obj.GetComponent<PooledObject>();
    if (po != null)
        Return(po.prefab, obj);
    else
        Destroy(obj); // fallback
}
```

(You can add this helper locally in your project — it's a small and safe quality-of-life change.)

Enemy example that returns itself to pool on death

SimpleEnemy.cs

```
using UnityEngine;
using System;

public class SimpleEnemy : MonoBehaviour
{
    public event Action<GameObject> OnDeath;
    [Header("Optional: pool prefab reference used when
returning")]

    public GameObject poolPrefab; // set by pool when creating
    public void Die()
    {
    // cleanup logic (particles, stop AI, etc)
    OnDeath?.Invoke(gameObject);
    // If pooling is used, the owner should return this to
pool.
    // For small projects you can deactivate:
    gameObject.SetActive(false);
    }
}
```

Option A — Enemy returns itself to pool (less recommended):

- `SimpleEnemy` can find a pool instance and call `pool.Return(poolPrefab, gameObject)` in `Die()`.

Option B — WaveSpawner manages lifetime (recommended):

- When WaveSpawner spawns an enemy it subscribes to `SimpleEnemy.OnDeath`.

- On death callback, spawner unsubscribes, removes enemy from `spawnedEnemies`, and calls `pool.Return(prefab, enemy)` if pooling is enabled.

This makes ownership explicit and avoids leaking pooled objects.

Example: WaveSpawner subscribing to enemy death

Add to `SpawnEnemy()` after `spawnedEnemies.Add(enemy);`:

```
var simple = enemy.GetComponent<SimpleEnemy>();
if (simple != null)
{
    void OnEnemyDeath(GameObject e)
    {
        spawnedEnemies.Remove(enemy);
        if (UsePooling && Pool != null)
            Pool.Return(prefab, enemy);
        simple.OnDeath -= OnEnemyDeath;
    }
    simple.OnDeath += OnEnemyDeath;
}
```

This ensures WaveSpawner controls removal and returning.

Best Practices & Tips

- **Designer-friendly assets**: Provide several example `EnemyDefinition`s and `WaveDefinition`s covering early-game, mid-game, swarm, and boss templates.

- **Cache wave computations**: Call `GetBudget/GetDuration/GetSpawnRate/GetMaxEnemies/GetMaxAliveEnemies` once in `GenerateWave()` and store them. The system currently recomputes some values during `Update()`—not expensive but neat optimization.
- **SpawnPoints weights**: `SpawnPoint.weight` exists but isn't used in the current `WaveSpawner`. Consider adding weighted spawn selection to create "hotter" spawn areas.
- **Test with `currentWave`**: Set `currentWave` in the inspector for quick wave preview without needing to progress in-game.
- **Consistent prefab references**: Use the exact same prefab asset reference when calling `pool.Get(prefab, ...)` and `pool.Return(prefab, obj)`. If you use Addressables or AssetBundles, confirm references remain identical at runtime.
- **Pooling and NavMeshAgents**: If an enemy has a `NavMeshAgent`, make sure you reset relevant agent state when reactivating pooled objects (stopping velocity, re-enable obstacle avoidance, set destination, etc). Consider an `IPoolable` interface with `OnSpawned()` / `OnDespawned()` hooks.
- **MaxAlive = 0** means "disabled" (internally returns `int.MaxValue`).

Common Pitfalls & Troubleshooting

Problem: Nothing spawns.

- Check `waveDefinition` is assigned.
- Ensure `allowedEnemies` or `fallbackEnemies` has valid `EnemyDefinition`s with `prefab` set and `cost > 0`, `weight > 0`.

- Check budget: `GetBudget()` might be 0 if curves/multipliers are wrong.

Problem: Wave never ends.

- If pooling is used and `treatInactiveAsDead == false`, spawner expects objects to be destroyed or set to null. If enemies are deactivated but not returned/nullified, `spawnedEnemies` may never be empty. Recommended: enable `treatInactiveAsDead` for pooling or ensure enemies explicitly call `pool.Return()`.

Problem: Pool returns wrong object or fails.

- `SimplePool.Return(prefab, obj)` requires correct `prefab`. Consider PooledObject helper to store prefab on instance.

Problem: Unexpected boss duplicates.

- Check boss exists in `allowedEnemies` or `eliteEnemies` as well as `bossEnemy`. If so, you can accidentally spawn boss twice.

Problem: Designer thinks wave is weaker than expected.

- Budget left over likely caused by no available enemy fitting remaining budget. Log `budget` leftover in `GenerateEnemiesFromBudget()` if this happens frequently.

Suggested Improvements (for future updates / Pro version)

- Add `IPoolable` interface and call `OnSpawned` / `OnDespawned`.
- Let `SpawnPoint.weight` be used for weighted spawn point selection.

- Add editor tools: wave preview/timeline, spawn-visualization gizmos, live debug overlay (current queue, budget left).
- Provide an Editor window for building wave templates / waves-by-level.
- Expose spawn selection strategies fully (Closest/Farthest implemented).
- Add Addressables compatibility sample.
- Add example integration with a simple Score/Time UI for demo.

Packaging for the Asset Store (what to include)

Folders & files:

- `Assets/SimpleWaveSystem/Runtime/` — all runtime scripts.
- `Assets/SimpleWaveSystem/Editor/` — any custom inspectors or editor utilities (optional).
- `Assets/SimpleWaveSystem/Demo/` — at least one scene showcasing:
  - Basic wave using `PredefinedTransforms`
  - Random-in-box spawning
  - Pooling example (with `SimplePool` warmup)
  - A boss wave demo
- `Documentation/` — this README as a markdown or PDF.
- `Examples/EnemyDefinitions/` and `Examples/WaveDefinitions/` — example assets.
- `CHANGELOG.md` and `LICENSE.txt` (Unity Asset Store EULA + your contact).
- `README.txt` — brief install and Unity version compatibility (Unity 6 URP).

Suggested store page contents:

- Short pitch bullets (budget-driven, data-driven, special waves, pooling).
- Screenshots of the demo scene and inspector.
- Brief Quick Start steps (copy 3–5 lines).
- Version & Unity compatibility (explicitly state Unity 6 URP).

FAQ (short)

Q: Can I use this with Addressables?
A: Yes, but ensure the prefab reference used with the pool is the same runtime object reference. Addressables can complicate identity — test pooling carefully.

Q: Do enemies auto-return to pool?
A: Not by default. Either enemy must `SetActive(false)` on death (if using `treatInactiveAsDead`), or you must call `pool.Return(prefab, obj)` explicitly. Use the PooledObject helper to simplify.

Q: Can wave types overlap?
A: The code checks special wave types in a fixed priority order; only the first match applies. Document this behavior or change code to support multiple simultaneous modifiers if desired.

Example Asset Values (starter tuning)

- Budget curve: Wave 1 = 8, Wave 5 = 16, Wave 20 = 100.
- Duration: Wave 1 = 20s, Wave 10 = 45s.

- SpawnRate: Wave 1 = 0.6 spawns/s, Wave 20 = 2 spawns/s.
- MaxEnemies: Wave 1 = 6, Wave 10 = 24.
- Grunt: cost=1 weight=10
- Elite grunt: cost=3 weight=2 (add to elite pool)
- Brute: cost=6 weight=1

These numbers are starting points: iterate visually.

Support

For further support email to snogdev@gmail.com