
# CraftRewards: Loot Generation, Gacha & Draft Reward System

---

Version 1.0.0 | User Manual & Documentation Ace
Horizon Studios

# Table of Contents

# 1. INTRODUCTION

Thank you for purchasing CraftRewards!

CraftRewards is a complete loot generation and reward presentation system designed for roguelike games and gacha mechanics. It provides everything needed to create, configure, roll, and beautifully present item rewards to players.

**Key Features**

- Weighted probability loot tables with nested sub-pools
- Gacha banner system with **soft/hard pity mechanics**
- Roguelike draft mode ("pick 1 of N" selection)
- 50/50 featured item system (Genshin Impact style)
- Full-screen cinematic splash for special reveals
- Card flip animations with rarity-based effects
- Chest opening sequence with Bezier card trajectories
- Custom tween system with **30+ easing curves**
- Object pooling for zero-GC card spawning
- JSON serialization for pity state persistence
- Event-driven architecture for inventory integration
- Custom editor inspectors with live probability preview
- Built-in loot simulator for statistical analysis

# 2. REQUIREMENTS

- Unity 2021.3 LTS or newer
- TextMeshPro package (included with Unity by default)
- TextMeshPro Essentials imported: *Window > TextMeshPro > Import TMP Essential Resources*

# 3. INSTALLATION

1. Import the CraftRewards package from the Unity Asset Store.
2. The Welcome Window will appear automatically on first import.
3. Verify system integrity in the Welcome Window: TextMeshPro should show *"Installed and configured."*
4. Open a demo scene:
   `Samples/01_BasicDraft/`, `02_GachaBanner/`, or `03_AutoLoot/`
5. Press Play to see the system in action.

# 4. QUICK START GUIDE

**Step 1: Create Rarity Profiles**
Right-click in the Project window: **Create > CraftRewards > Rarity**. Create at least: Common (Tier 0), Rare (Tier 1), Epic (Tier 2), Legendary (Tier 3). For each, configure rarityName, glowColor, textColor, pityWeightTier, revealDelay, shakeIntensity, revealSound, and revealVFXPrefab. Use the Theme Presets bar for quick setup.

**Step 2: Create Reward Items**
Right-click: **Create > CraftRewards > Reward Item**. Configure identity (itemID, displayName, description), visuals (icon, rarity, fullArtSprite), drop rules (isUnique), quantity (hasQuantity, min/max range), conditions (requiredPlayerLevel, requiredTags), and rate modification (isRateUpItem, rateUpMultiplier). Use Quick Setup presets for common item types (Currency, Equipment, Consumable, Character).

**Step 3: Create a Loot Table**
Right-click: **Create > CraftRewards > Loot Table**. Add weighted entries, configure guaranteed drops, optional nested tables, and table rules (minDrops, maxDrops, allowDuplicates). Use the Live Map Preview to see probability distribution and the Loot Simulator to verify rates with 10,000+ iterations.

**Step 4: Create a Pity Profile (Optional)**
Right-click: **Create > CraftRewards > Pity Profile**. Configure soft pity (softPityStart, softPityRampRate, softPityMaxBonus), hard pity (hardPityGuarantee), target tier (minimumPityWeightTier), and 50/50 system (useFiftyFifty, guaranteeRateUpAfterLoss). Use Pity Presets for common configurations (Generous, Standard, Strict).

**Step 5: Set Up a Presenter**

For **Gacha Mode**: use `[UI_INTERFACE]-Gacha-Banner.prefab`, drag into your Canvas, and assign your Loot Table to GachaPresenter.LootTable. For **Draft Mode**: use `[UI_INTERFACE]-Basic-Draft.prefab` and call `DraftPresenter.Show(outcome, table)` from your code. For **Headless Mode**: call `RewardEngine.QuickRoll()` and process results.

**Step 6: Test**

Press Play. In Gacha mode, click the pull button. In Draft mode, trigger a reward selection. Verify animations, inventory integration, and pity mechanics all work correctly.

## 5. FOLDER STRUCTURE

All CraftRewards files are located under `Assets/CraftRewards/` after import:

```
CraftRewards/
Documentation/ PDF manual
Editor/
Inspectors/ Custom property drawers and editors
Simulator/ Loot simulation tool
Styles/ CraftRewardsEditorStyles.cs
Windows/ CraftRewardsWelcomeWindow.cs
Runtime/
Animation/ CraftTween system (30+ easing curves)
Core/ RewardEngine, PityTracker, WeightedSelector
Data/ ScriptableObject base classes
Events/ CraftRewardEvents (static event bus)
Pooling/ Object pool for cards/paths
Presentation/ GachaPresenter, DraftPresenter, UI components
Shaders/ Rarity beam and glow shaders
Prefabs/ Ready-to-use UI prefabs
Samples/
01_BasicDraft/ Draft demo scene
02_GachaBanner/ Gacha demo scene
03_AutoLoot/ Headless demo scene
Art/ Sprites, VFX, materials
```

# 6. CORE CONCEPTS

CraftRewards separates **Data** (what a reward is) from **Presentation** (how it looks and animates), using ScriptableObjects for all configuration.

## 6.1 Reward Items (RewardItem)

The base unit of the loot system. Each RewardItem is a ScriptableObject defining:

- **Identity**: itemID (auto-sanitized from asset name), displayName, description
- **Visuals**: icon, rarity, fullArtSprite (512×1024 recommended)
- **Drop Rules**: isUnique (cannot appear twice in same roll)
- **Quantity**: hasQuantity, quantityMin / quantityMax for stackable items
- **Conditions**: requiredPlayerLevel, requiredTags (matches RollRequest.activeTags)
- **Rate Modification**: isRateUpItem, rateUpMultiplier (2.0 = double chance)
- **Custom Data**: Free-form JSON string (not parsed by CraftRewards)

RewardItems are currency-agnostic — they represent anything: weapons, consumables, characters, currency, unlocks.

## 6.2 Rarity Profiles (RewardRarity)

Defines the visual identity of a reward tier:

### 6.2.1 Identity:

- **rarityName**: Display name (Common, Rare, Epic, etc.)
- **sortOrder**: Priority for sorting (higher = rarer)

### 6.2.2 Visuals:

- **glowColor**: Primary color for beams and accents
- **textColor**: Item name color in UI
- **frameOverlay**: Optional sprite for card border
- **revealVFXPrefab**: Particle effect on reveal

### 6.2.3 Audio:

- **revealSound**: Played when card flips
- **loopAmbience**: Background loop during reveal

### 6.2.4 Pity System:

- **pityWeightTier**: Integer tier (0 = common, 3+ = legendary). Items with tier >= PityProfile.minimumPityWeightTier reset the pity counter.

#### 6.2.5 Animation:

- **revealDelay**: Extra pause before flip (builds anticipation)
- **shakeIntensity**: Screen shake multiplier
- **revealPunchScale**: Scale punch after flip (1.1 = 110%)

### 6.3 Loot Tables (LootTable)

Central data structure defining a pool of weighted item drops. Components:

- **Weighted Entries**: Each pairs a RewardItem with a weight (float). Probability = (entry weight / total weight) × 100%. Supports conditional entries.
- **Guaranteed Drops**: Items that drop independently of weighted selection, each with a chance gate (0–1). Processed before weighted rolls.
- **Nested Tables**: Sub-tables that participate in weighted selection — when selected, rolls are made inside the sub-table. Useful for category-based loot.
- **Table Rules**: defaultMinDrops / defaultMaxDrops, allowDuplicatesDefault.
- **Pity System**: defaultPityProfile applied when RollRequest doesn't override.

### 6.4 Pity Profiles (PityProfile)

Defines mathematical rules for bad luck protection:

#### 6.4.1 Soft Pity:

- **softPityStart**: Number of pulls before boost begins
- **softPityRampRate**: Bonus weight added per pull after soft pity
- **softPityMaxBonus**: Cap on cumulative bonus
- Example: Start=75, RampRate=0.05, MaxBonus=1.0 → Pull 76: +5%, Pull 77: +10%, ..., Pull 95: +100% (capped)

#### 6.4.2 Hard Pity:

- **hardPityGuarantee**: Absolute pull count for forced drop
- All weighted selection is bypassed — a high-tier item is guaranteed

#### 6.4.3 50/50 System:

- **useFiftyFifty**: Enable featured item mechanic
- **guaranteeRateUpAfterLoss**: Next drop is featured after off-banner loss

# 7. REWARD ENGINE

RewardEngine is the core calculation system — instance-based with deterministic seed support and a static convenience API.

## 7.1 Roll Requests (RollRequest)

Encapsulates all parameters for a single roll. Required: **table** and **amount**. Optional:

- **allowDuplicates**: Same item multiple times? (default: true)
- **pityTracker**: PityTracker instance for pity mechanics
- **luckMultiplier**: Multiplier for rare+ item weights (default: 1.0)
- **playerLevel**: Filter items by requiredPlayerLevel
- **activeTags**: Set of condition tags
- **excludeItemIDs**: Items to exclude from pool
- **forcedSeed**: Override RNG seed
- **maxNestedDepth**: Prevent infinite recursion (default: 8)

## 7.2 Roll Outcomes (RollOutcome)

Contains the complete result of a roll:

- **results**: List<RewardResult> of all selected items
- **totalPullCount**: PityTracker.TotalPulls after roll (0 if no tracker)
- **hardPityActivated**: True if hard pity was triggered
- **fiftyFiftyWon / fiftyFiftyLost**: 50/50 outcome flags
- **seed**: RNG seed used (-1 if no explicit seed)
- **sourceTableName**: Name of the source LootTable

Each **RewardResult** contains: item, quantity, wasPityTriggered, wasRateUpActive, wasGuaranteed, wasFromNestedTable, rollIndex, effectiveWeight.

## 7.3 Weighted Selection (WeightedSelector)

High-performance weighted random using cumulative weight arrays and binary search. O(N) to build, O(log N) per selection. Algorithm:

1. Build cumulative weight array: [w1, w1+w2, w1+w2+w3, ...]
2. Roll random float in [0, totalWeight)
3. Binary search to find the first cumulative weight >= roll

## 7.4 Pity System Mechanics

The pity system operates in 5 phases:

- Phase 1: **Check Hard Pity**: If tracker.PullsSinceHighTier >= hardPityGuarantee → force high-tier drop, bypass weighted selection, set wasPityTriggered = true
- Phase 2: **Apply Soft Pity**: Calculate bonus = (pullsSince - softPityStart) × rampRate, clamped to maxBonus. Multiply high-tier item weights by (1 + bonus)
- Phase 3: **Weighted Selection**: Normal selection from modified pool
- Phase 4: **50/50 Check**: If high-tier selected and 50/50 enabled — 50% chance for rate-up item, or guarantee if player lost last time
- Phase 5: **Update Tracker**: If item.rarity.pityWeightTier >= minimumPityWeightTier → reset counter, else increment

All pity logic is **deterministic** for a given seed and tracker state.

# 8. PRESENTATION LAYER

## 8.1 Gacha Presenter (GachaPresenter)

Manages the full gacha sequence:

- Chest Entrance: Drops from above with bounce
- Wait for Click: Idle pulse animation
- Shake: Position and rotation shake
- Burst: Chest opens, particle effect
- Cards Shoot Out: Bezier curves from chest to grid positions
- Cards Land Face-Down: Flip animation on click
- Full Art Splash (optional): For items with showFullArtSplash
- Claim All: Summary screen

**Key Configuration:** cardPrefab, cardParent, chestAnimator, cardSlots, cardSpacingX/Y, maxCardsPerRow, cardShootStagger, cardFlightDuration, bezierArcHeight, allowSkipAnimation, showSummaryScreen, fullArtSplashPanel.
**Events:** OnPullCompleted(RollOutcome), OnAllRevealed(), OnAllClaimed(). **API:** `void Show(RollOutcome, LootTable)` / `void Hide()`

## 8.2 Draft Presenter (DraftPresenter)

Manages roguelike draft selection: cards appear face-up, player selects, selected card scales up, others fade out. Supports optional confirm button, reroll, and timer.

**Configuration:** selectCount, allowReroll, maxRerolls, timeLimit, confirmButton.
**Events:** OnRewardSelected(RewardResult), OnDraftCompleted(List<RewardResult>), OnRerollUsed(), OnTimerExpired().

### 8.3 Reward Card View (RewardCardView)

Visual representation of a single card. Flip animation uses a 2.5D illusion via scaleX compression:

- Phase 1: Compress to 0 (InQuad easing)
- Midpoint: Swap back → front, activate effects
- Phase 2: Expand to 1 (OutBack easing)
- Finale: Punch scale, screen shake, sound

**States:** Revealed, Interactable, Selected (draft mode), Dimmed (non-selectable).

### 8.4 Chest Animator (ChestAnimator)

Manages the chest lifecycle in gacha mode. Uses three sprites: Closed (idle), Shake (bulging), Open (burst). Methods: PlayEntrance, PlayShake, PlayBurst, PlayExit, PlayFullSequence. Provides `CenterPosition` for Bezier origin calculation.

### 8.5 Full Art Splash Panel (FullArtSplashPanel)

Cinematic full-screen reveal for special items. Sequence: background dim + white flash → rarity beam scales up →full art slides in from right → info panel slides from left → name underline extends → "Tap to continue" pulses.

**API:** `void Show(RewardResult, Action onDismiss)` / `void Hide()` / `void HideImmediate()`

## 9. UI COMPONENTS

### 9.1 Summary Panel (SummaryPanel)

Displays a summary of all obtained rewards after a gacha session.

- Groups results by rarity (highest first)
- Shows icon, name, quantity, and special badges
- Pity badge for pity-triggered items; Rate-up badge for featured items
- Staggered entrance animation
- Close button with callback — API: void Show(RollOutcome, Action onClose)

### 9.2 Rate Display Panel (RateDisplayPanel)

Displays drop rate probabilities for a LootTable. Required by law in several regions for gacha mechanics.

- Groups entries by rarity tier with combined and individual percentages
- Color-coded rarity bars
- Toggle show/hide button with fade animation
- API: void Populate(LootTable) / void Show() / void Hide() / void Toggle()

### 9.3 Node Tooltip (Optional)

Simple floating tooltip for showing item info on hover.

- Supports TMP and Legacy Text
- Auto-flips when near screen edges
- Smooth fade-in/out — API: void Show(RewardItem, Vector2) / void Hide()

### 9.4 Node Info Panel (Optional)

Detailed side panel for showing full item information.

- Icon with tinted background, title, description, and rarity label
- Status indicators (obtained, new, locked)
- Optional action button — API: void Show(RewardItem) / void SetActionButton(string, Action)

## 10. ANIMATION SYSTEM (CRAFTTWEEN)

---

CraftTween is a lightweight, zero-dependency tween system built specifically for CraftRewards. It produces minimal GC allocation through object pooling and avoids virtual calls.

### 10.1 Core Operations

**Transform Tweens:**

```
Scale(Transform, to, duration, Easing)
ScaleUniform(Transform, to, duration, Easing)
ScalePunch(Transform, punchScale, duration)
```

**RectTransform Tweens:**

```
Move(RectTransform, to, duration, Easing)
MoveBezier(RectTransform, start, control, end, duration, Easing)
MoveBezierCubic(RectTransform, start, c1, c2, end, duration, Easing)
Rotate(RectTransform, to, duration, Easing)
PunchPosition(RectTransform, punch, duration)
ShakePosition(RectTransform, intensity, duration, vibrato)
ShakeRotation(RectTransform, intensity, duration, vibrato)
```

**UI Tweens:**

```
Fade(CanvasGroup, to, duration, Easing)
ColorTo(Graphic,to, duration, Easing)
```

**Utility:**

```
Value(from, to, duration, Action, Easing)
Delay(float delay, Action callback)
```

## 10.2 Chaining and Configuration

```
CraftTween.Scale(myTransform, Vector3.one * 1.2f, 0.3f)
.SetDelay(0.1f)
.SetEasing(Easing.OutBack)
.SetLoops(3, LoopType.PingPong)
.SetTimeScale(useUnscaledTime: false)
.OnComplete(() => Debug.Log("Done"))
.OnLoopComplete(loopIndex => Debug.Log($"Loop {loopIndex} done"));
```

## 10.3 Easing Curves — 30 built-in options

- **Linear:** Linear
- **Quadratic:** InQuad, OutQuad, InOutQuad
- **Cubic:** InCubic, OutCubic, InOutCubic
- **Quartic:** InQuart, OutQuart, InOutQuart
- **Quintic:** InQuint, OutQuint, InOutQuint
- **Sinusoidal:** InSine, OutSine, InOutSine
- **Exponential:** InExpo, OutExpo, InOutExpo
- **Circular:** InCirc, OutCirc, InOutCirc
- **Back (Overshoot):** InBack, OutBack, InOutBack
- **Elastic (Spring):** InElastic, OutElastic, InOutElastic
- **Bounce:** InBounce, OutBounce, InOutBounce

## 10.4 Loop Types

- **LoopType.None**: Play once and complete
- **LoopType.Restart**: Restart from beginning each cycle
- **LoopType.PingPong**: Reverse direction each cycle

## 10.5 Sequences

TweenSequence choreographs multiple tweens in order:

```
Var seq = CraftTween.Sequence()
.Append(() => CraftTween.Scale(card1, Vector3.one * 1.2f, 0.3f))
.AppendInterval(0.5f)
.AppendCallback(() => Debug.Log("Halfway"))
.Append(() => CraftTween.Move(rect, Vector2.zero, 0.4f))
.OnComplete(() => Debug.Log("All done"))
.Play();
```

### 10.6 Kill Methods

```
CraftTween.KillAll(targetObject, complete: false)
CraftTween.KillEverything()
handle.Kill(complete: true)
```

If `complete=true`, the tween snaps to its final value and fires OnComplete before being killed.

### 10.7 Performance Notes

- Internal object pool: max 128 TweenInstance objects — exceeding causes new allocations but no failure
- All tweens run in CraftTweenRunner.Update() — no coroutines
- Target object alive-checking uses Unity's implicit bool conversion

### 10.8 Bezier Curves

- Quadratic (3 control points): $B(t) = (1-t)^2P + 2(1-t)tP + t^2P$
- Cubic (4 control points): $B(t) = (1-t)^3P + 3(1-t)^2tP + 3(1-t)t^2P + t^3P$
- Used for card trajectories in GachaPresenter.

### 10.9 Shake Animations

ShakePosition and ShakeRotation use deterministic pseudo-Perlin noise for smooth oscillation. Parameters: **intensity** (max pixels/degrees), **duration** (total time), **vibrato** (oscillations). Shake automatically returns to the original position on completion.

## 11. EVENT SYSTEM

CraftRewards provides a static event bus for loose coupling between systems. All events are fired on the main thread.

### 11.1 Available Events

`CraftRewardEvents.OnRollCompleted(RollOutcome)`

- Fired immediately after RewardEngine.Roll() completes, before any UI presentation. Useful for analytics and server validation.

`CraftRewardEvents.OnRewardClaimed(RewardResult)`

- Fired when a single reward is claimed. Primary event for inventory integration.

`CraftRewardEvents.OnAllRewardsClaimed(RollOutcome)`

- Fired when all rewards in a session are claimed (Claim All or Draft confirmed).

`CraftRewardEvents.OnDraftSelectionMade(RewardResult, RewardResult[])`

- Fired in draft mode with both selected and rejected items.

`CraftRewardEvents.OnHardPityTriggered(RewardResult)`

- Fired when hard pity guarantee activates. Useful for special UI effects or analytics.

`CraftRewardEvents.OnSoftPityStarted(int pullCount)`

- Fired when player crosses the soft pity threshold.

`CraftRewardEvents.OnCardRevealed(RewardResult)`

- Fired when a card flip animation completes.

`CraftRewardEvents.OnPresentationClosed()`

- Fired when gacha/draft UI is fully closed.

### 11.2 Usage Example

```
void OnEnable()
{
CraftRewardEvents.OnRewardClaimed += HandleRewardClaimed;
CraftRewardEvents.OnHardPityTriggered += HandlePityTrigger;
}
void HandleRewardClaimed(RewardResult result)
{
AddItem(result.item.itemID, result.quantity);
if (result.wasPityTriggered) ShowSpecialNotification("Pity!");
}
```

### 11.3 Clearing Listeners

To prevent stale references during scene changes:

```
CraftRewardEvents.ClearAllListeners();
```

This is automatically called during domain reload (entering/exiting play mode).

## 12. SERIALIZATION AND SAVE/LOAD

---

CraftRewards uses JSON serialization via Unity's JsonUtility for pity state persistence. The PityTracker class is fully serializable.

### 12.1 PityTracker Serialization

```
string json = myTracker.ToJson();
```

```
PityTracker tracker = PityTracker.FromJson(json);
```

### 12.2 PlayerPrefs Integration

```
string json = PlayerPrefs.GetString("pity_banner_celestial", "");
PlayerPrefs.SetString("pity_banner_celestial", myTracker.ToJson());
PityTracker tracker = PityTracker.FromJson(json);
```

### 12.3 PityTracker Serialized Fields

- **_pullsSinceHighTier**: Pulls since last high-tier drop
- **_totalPulls**: Total lifetime pulls
- **_lostFiftyFiftyLast**: Lost the last 50/50?
- **_bannerID**: Optional banner identifier
- **_history**: List<PullRecord> (last 200 pulls — pullNumber, rarityTier, timestamp)

### 12.4 History Analytics

```
int longest = tracker.GetLongestDryStreak(minimumHighTier: 2);
int count = tracker.GetHighTierCount(minimumHighTier: 2);
```

### 12.5 Multi-Banner Tracking

Use separate PityTracker instances per banner, saved with unique PlayerPrefs keys (`pity_{bannerID}`). Set `separatePityPerBanner = true` in the PityProfile.

## 13. SHADERS

CraftRewards includes 5 custom shaders for visual effects. All shaders are UI-compatible (support masking and clipping). All use additive or standard blending as appropriate.

### 13.1 CraftRewards/RarityBeam

Vertical light pillar behind reward cards.

- _Color: Tint color (from RewardRarity.glowColor)
- _Intensity: Brightness multiplier (0–5)
- _ScrollSpeed: Vertical UV scroll speed (0–5)
- _PulseSpeed / _PulseMin / _PulseMax: Alpha pulse parameters
- _FadeBottom / _FadeTop: Vertical fade range (0–1)

### 13.2 CraftRewards/UIGlow

Additive glow effect for card highlights and rarity auras.

- _Color: Glow color
- _Intensity: Brightness (0–10)
- _PulseSpeed / _PulseAmplitude: Pulse parameters

### 13.3 CraftRewards/ParticleBurst

Additive blending shader for particle systems. Supports vertex color and soft edge fade.

- _Intensity: Brightness (0–10)
- _SoftFade: Edge softness (0–1)

### 13.4 CraftRewards/RadialFill

Radial countdown timer for draft mode. Ring-shaped with configurable thickness.

- _Fill: Fill amount (0–1, animated via script)
- _Thickness: Ring thickness (0–0.5)
- _Smoothing: Edge anti-aliasing (0–0.05)

### 13.5 CraftRewards/UIColorOverlay

Standard UI shader with optional desaturation — used for dimmed/rejected cards in draft mode.

- _Desaturation: Grayscale amount (0–1, animated on card reject)
- _OverlayStrength: Overlay blend amount (0–1)
- _Brightness: Brightness multiplier (0–3)

### 13.6 Shader Configuration

To ensure shaders are included in builds:

- Open **Project Settings > Graphics**
- Scroll to *Always Included Shaders*
- Add all 5 CraftRewards shaders

If shaders are missing, UI elements will appear pink/magenta.

## 14. API REFERENCE

### 14.1 RewardEngine

```
RewardEngine() // Random seed
RewardEngine(int seed) // Deterministic seed
static RewardEngine Default { get; } // Shared instance
static RollOutcome QuickRoll(LootTable, int amount = 1)
static RollOutcome QuickRollUnique(LootTable, int amount = 1)
RollOutcome Roll(RollRequest request)
int Seed { get; }
```

### 14.2 PityTracker

```
int PullsSinceHighTier { get; set; }
int TotalPulls { get; set; }
bool LostFiftyFiftyLast { get; set; }
IReadOnlyList History { get; }
void RecordPull(int rarityTier, int minimumHighTier)
void ResetPityCounter()
void ResetAll()
string ToJson() / string ToJsonPretty()
static PityTracker FromJson(string json)
int GetLongestDryStreak(int minimumHighTier)
int GetHighTierCount(int minimumHighTier)
```

### 14.3 RollRequest

```
RollRequest(LootTable table, int amount = 1)
bool allowDuplicates // default: true
PityTracker pityTracker // default: null
float luckMultiplier // default: 1.0
int playerLevel // default: 0
HashSet activeTags
HashSet excludeItemIDs
int? forcedSeed
int maxNestedDepth // default: 8
RollRequest WithAmount(int newAmount)
```

### 14.4 RollOutcome

```
List results
int totalPullCount
bool hardPityActivated
bool fiftyFiftyWon / fiftyFiftyLost
int seed
string sourceTableName
List GetResultsByRarityTier(int minimumTier)
int GetHighestRarityTier()
void SortByRarityDescending()
static RollOutcome Empty
```

### 14.5 RewardResult (Struct)

```
RewardItem item
int quantity
bool wasPityTriggered
bool wasRateUpActive
bool wasGuaranteed
bool wasFromNestedTable
int rollIndex
float effectiveWeight
string ItemID { get; }
int RaritySortOrder { get; }
```

## 14.6 CraftTween — Static API

```
static TweenHandle Scale(Transform, Vector3 to, float, Easing)
static TweenHandle Move(RectTransform, Vector2 to, float, Easing)
static TweenHandle MoveBezier(RectTransform, Vector2 s, Vector2 c, Vector2 e,
float, Easing)
static TweenHandle Fade(CanvasGroup, float to, float, Easing)
static TweenHandle ColorTo(Graphic, Color to, float, Easing)
static TweenHandle ShakePosition(RectTransform, float, float, int)
static TweenHandle Delay(float delay, Action callback)
static TweenSequence Sequence()
static void KillAll(UnityEngine.Object, bool complete = false)
static void KillEverything()
static int ActiveCount { get; }
```

## 14.7 TweenHandle

```
bool IsAlive / IsComplete / IsKilled
float Progress
TweenHandle SetDelay(float)
TweenHandle SetEasing(Easing)
TweenHandle SetLoops(int, LoopType)
TweenHandle SetTimeScale(bool useUnscaledTime)
TweenHandle OnComplete(Action)
TweenHandle OnLoopComplete(Action)
void Kill(bool complete = false)
```

## 14.8 CraftRewardEvents

```
static event Action OnRollCompleted
static event Action OnRewardClaimed
static event Action OnAllRewardsClaimed
static event  Action OnDraftSelectionMade
static event Action OnHardPityTriggered
static event Action OnSoftPityStarted
static event Action OnCardRevealed
static event  Action OnPresentationClosed
static void ClearAllListeners()
```

# 15. EDITOR TOOLS

CraftRewards includes extensive custom editor inspectors and tools to streamline asset creation and configuration.

## 15.1 Loot Table Inspector

- Overview: Total Weight, Unique Items, Guaranteed Drops, Nested Tables, Drop Range, Rarity Breakdown
- Presets: Boss Drop, Gacha x1, Gacha x10, Daily Login, Draft (3 choices), Event Pool
- Weighted Entries: Custom inline editor with icon preview, color-coded rarity bars, weight/percentage columns
- Probability Preview: Live percentage calculation and rarity distribution summary
- Validation: Real-time error detection, missing references, circular reference detection
- Loot Simulator: Run 100 to 1,000,000 iterations with distribution histogram, deviation analysis, CSV export

## 15.2 Reward Item Inspector

- Card Preview: Live 150×200px card rendering with icon, rarity colors, frame overlay
- Quick Setup Presets: Currency, Equipment, Consumable, Character
- Full Art Splash Section: Large preview (140px), checkerboard transparency background
- Validation summary at bottom: missing icon, missing rarity, empty name, min>max errors

## 15.3 Rarity Inspector

- Color Preview: Live glow color bar and text color sample
- Theme Presets: Common, Uncommon, Rare, Epic, Legendary, Mythic
- Audio Section: Reveal Sound and Loop Ambience with in-editor Play/Stop buttons
- Pity System Section: Tier slider with explanation text

## 15.4 Pity Profile Inspector

- Pity Curve Preview: Live graph (120px) with soft pity (yellow) and hard pity (red) lines
- Pity Presets: Generous (40/60), Standard (75/90), Strict (80/100), Linear, No Pity
- Interactive Test: Slider to test at Pull #X — shows bonus weight, multiplier, and status
- 50/50 System Section: Toggle with status message based on config

## 15.5 Welcome Window

- Access: Tools > CraftRewards > Welcome
- System Status: Unity version, TMP, demo scenes checks
- Quick Start: Create Rarity, Reward Item, Loot Table, Pity Profile buttons
- Demo Scenes: One-click launch for all three demos
- Documentation: Open PDF manual and Visit Website buttons

### 15.6 Loot Simulator

- Runs headless loot generation for statistical analysis
- Iterations: 100 to 1,000,000 — Items Per Roll: 1 to 20
- Results: Total Rolls, Execution Time, Pity Triggers, Worst/Best Streaks
- Histogram: Per-item bars sorted by rarity with Actual%, Expected%, Deviation%
- Deviation Summary: Green (<5%), Yellow (5–15%), Red (>15%)
- CSV Export for external analysis

### 15.7 Custom Inspectors — General Features

- Foldable sections for compact view
- Color-coded status indicators: [OK], [MISSING], [Optional]
- Live preview where applicable
- Validation with contextual help boxes
- Preset buttons for common configurations
- Consistent color palette with dark/light skin support

### 15.8 Menu Items

Access via **Tools > CraftRewards >**: Welcome, Create Rarity, Create Reward Item, Create Loot Table, Create Pity Profile.

Also available via **Assets > Create > CraftRewards >**: Rarity, Reward Item, Loot Table, Pity Profile.

## 16. BEST PRACTICES

### 16.1 Loot Table Design

**Recommended weight distribution:**

- Common tier (70–85%): Currency, basic materials
- Uncommon tier (10–20%): Useful consumables, common equipment
- Rare tier (5–10%): Good equipment, rare materials
- Epic tier (1–3%): Premium equipment, special items
- Legendary tier (0.5–1%): Best equipment, characters
- Mythic tier (<0.5%): Ultra-rare, prestige items

Run the Loot Simulator with 10,000+ iterations and verify actual rates match expected within 5%. Avoid extremely small weights (<0.001) due to precision loss.

## 16.2 Pity Configuration

- **Standard Gacha (Genshin style)**: Soft@75, RampRate=0.05, Hard@90, 50/50 enabled with guarantee after loss
- **Generous Gacha (casual)**: Soft@40, RampRate=0.08, Hard@60, 50/50 disabled
- **No Pity (pure RNG)**: Don't assign PityProfile, or set hardPityGuarantee to 999
- **Multi-Banner**: Use separate PityTracker per banner with unique save keys

## 16.3 UI/UX Recommendations

- Show pity counter in UI (X/90 pulls) and highlight when soft pity is active
- Show "Guarantee Active" badge after 50/50 loss
- Allow skip animation for repeat pullers
- Always show drop rates — legally required in many regions (use RateDisplayPanel)

## 16.4 Performance Optimization

- Reuse RewardEngine instance — avoid creating per roll
- Keep loot table entries under 100 for fast selection
- Avoid deep nesting (max 3 levels)
- Limit max card count per gacha pull (10 max recommended)
- Reduce particle effects on low-end mobile devices

## 16.5 Integration Patterns

### Inventory Integration:

```
CraftRewardEvents.OnRewardClaimed += (result) => {
MyInventory.AddItem(result.item.itemID, result.quantity);
};
```

### Currency Deduction:

```
if (MyCurrency.Spend("gems", 160)) {
var outcome = RewardEngine.QuickRoll(table, 1);
presenter.Show(outcome);
}
```

**Server Validation (anti-cheat):** Use forcedSeed from server. Same seed + same table = same results, allowing server-side validation of client rolls.

### 16.6 Content Creation Workflow

- **Step 1:** Define 4–6 Rarity ScriptableObjects with distinct colors and pity tiers (0, 0, 1, 2, 3, 4)
- **Step 2:** Create item database organized by folders (Items/Currency, Items/Equipment, Items/Characters)

- **Step 3:** Build Loot Tables (EnemyDrops, BossDrops, ShopStock, GachaBanner) — verify with simulator
- **Step 4:** Configure Pity Profiles per banner — use Standard preset for gacha
- **Step 5:** Set up Presenters using provided prefabs — customize to match your game style
- **Step 6:** Test in Play Mode — verify animations, inventory, pity, skip button

# 17. PERFORMANCE NOTES

## 17.1 Memory Allocation

**Zero-GC Operations:**

- RewardEngine.Roll() (pure math, no allocations)
- WeightedSelector (reuses float array)
- CraftTween (object pooled, max 128 instances)
- ComponentPool (reuses card instances)
- Struct-based data (RewardResult, LootEntry, GuaranteedDrop)

**Necessary Allocations:** RollOutcome.results (List allocation), PityTracker.History, TweenSequence, JSON serialization strings.

## 17.2 CPU Performance

**RewardEngine.Roll() Complexity:** O(N) entry processing + O(N) build + O(log N) select + O(1) pity.

**Typical Performance:** 100-entry table, 10 rolls: <1ms (desktop), 1–2ms (mobile). Loot Simulator: 10,000 iterations ≈ 40–60ms (desktop).

## 17.3 GPU Performance

- All 5 shaders are simple — no complex math or multi-pass rendering
- UIGlow uses additive blending — fill-rate dependent on mobile
- Limit particle count to 50–100 max per VFX
- Use atlased sprites for batching

## 17.4Scalability — Maximum Recommended Limits

- Loot Table Entries: 200
- Gacha Pull Count: 10
- Active Tweens: 200
- Nested Table Depth: 3
- Pity History Size: 200 (auto-trimmed)

### 17.5Build Size

- Package Size: ~8 MB uncompressed
- Shaders: 5 files, ~15 KB compiled; Scripts: 50+ files, ~200 KB DLL
- Demo Assets: ~6 MB (sprites, audio, scenes) — can be excluded from final build

## 18. TROUBLESHOOTING AND FAQ

### 18.1 Common Issues

**Q: Cards don't appear / are invisible**

A: Check that RewardCardView prefab has CanvasGroup and Image components. Verify sprites are assigned to RewardItem.icon. Check Canvas render mode is Screen Space.

**Q: Shaders appear pink/magenta**

A: Add all 5 shaders to Project Settings > Graphics > Always Included Shaders.

**Q: Gacha sequence doesn't start**

A: Verify GachaPresenter has all references assigned. Check ChestAnimator and CardPrefab are set. Ensure RollOutcome has results (not empty).

**Q: Draft selection doesn't work**

A: Check DraftPresenter.selectCount (must be ≥1). Verify cards are interactable (not dimmed). Ensure RollOutcome has enough results.

**Q: Pity system not working**

A: Verify PityProfile is assigned to LootTable or RollRequest. Check PityTracker is passed in RollRequest. Ensure RewardRarity.pityWeightTier is set correctly.

**Q: Animations are choppy**

A: Reduce active tween count (use CraftTween.ActiveCount to check). Check for excessive GC allocations causing frame spikes.

**Q: JSON deserialization fails**

A: Ensure JSON string is valid. Check for version mismatch. Verify PityTracker.FromJson() handles empty/null gracefully.

**Q: Duplicate items appear despite allowDuplicates=false**

A: Expected behavior when pool size < requested amount. Reduce request.amount or increase pool size.

## 18.2 Integration FAQ

**Q: How do I add items to my inventory?**

A: Subscribe to CraftRewardEvents.OnRewardClaimed and call your inventory system with result.item.itemID and result.quantity.

**Q: How do I save pity state?**

A: Use tracker.ToJson() and save to PlayerPrefs.SetString("pity_banner", json).

**Q: Can I use CraftRewards with Addressables?**

A: Yes. RewardItem, LootTable, etc. are ScriptableObjects — load with Addressables.LoadAssetAsync("key").

**Q: Can I modify loot tables at runtime?**

A: Not recommended (ScriptableObjects are shared). Instead use RollRequest parameters: activeTags, excludeItemIDs, luckMultiplier. For dynamic tables, create via ScriptableObject.CreateInstance().

**Q: Can I animate custom properties?**

A: Yes, use CraftTween.Value(0f, 1f, 1f, (v) => myMaterial.SetFloat("_Prop", v)).

## 18.3 Design FAQ

**Q: What's the difference between Draft and Gacha?**

A: Draft: Cards face-up, player selects 1 of N, instant reveal. Gacha: Cards face-down, animated reveal, all cards kept.

**Q: When should I use guaranteed drops vs weighted entries?**

A: Guaranteed: Fixed rewards (tutorial chest, milestones). Weighted: Random loot (enemies, chests, gacha).

**Q: Should I use pity for non-gacha games?**

A: Yes, optional but helpful. Example: Guarantee a healing item after 5 combat rooms with no heal drop.

**Q: What's the ideal weight distribution?**

A: Follow the 70/20/8/2 rule: 70% common, 20% uncommon, 8% rare, 2% epic+.

**Q: Should I show drop rates to players?**

A: Yes — legally required in many regions (China, Japan, South Korea, EU). Use RateDisplayPanel or a custom implementation.

### 18.4 Technical FAQ

**Q: Is CraftRewards deterministic?**

A: Yes — given the same seed and RollRequest, results are identical every time.

**Q: Does CraftRewards work with multiplayer?**

A: Yes. Use server-authoritative rolls: server generates seed, client rolls with that seed, server validates results match.

**Q: Can I use CraftRewards in WebGL?**

A: Yes, fully compatible. JSON serialization works in WebGL.

**Q: Does CraftRewards work with URP/HDRP?**

A: Shaders are built-in RP. For URP/HDRP, use Unity's shader upgrader or disable shaders and use Sprite Renderer-based effects instead.

## 19. CHANGELOG

### Version 1.0.0 — Initial Release

**Core Systems:**

- RewardEngine: Weighted probability loot generation with deterministic seed support
- PityTracker: Session-persistent pity tracking with JSON serialization
- WeightedSelector: High-performance binary search-based selection
- PityProfile: Configurable soft/hard pity with 50/50 system
- RollRequest / RollOutcome: Comprehensive roll configuration and result API

**Data Structures:**

- RewardItem, RewardRarity, LootTable, PityProfile ScriptableObjects
- GuaranteedDrop and NestedTableEntry sub-table support

**Presentation Layer:**

- GachaPresenter: Full lootbox sequence with chest animation
- DraftPresenter: Roguelike draft with selection and reroll
- RewardCardView: Flippable card with rarity effects
- ChestAnimator: Drop/shake/burst sequence
- FullArtSplashPanel: Cinematic full-screen reveal
- SummaryPanel and RateDisplayPanel

**Animation System (CraftTween):**

- 30+ easing curves, Transform/RectTransform/UI tweens
- TweenSequence for choreography
- Object pooling (max 128 instances), zero-allocation updates
- Loop support (None, Restart, PingPong)

**Shaders:**

- CraftRewards/RarityBeam, UIGlow, ParticleBurst, RadialFill, UIColorOverlay

**Editor Tools:**

- Loot Table Inspector: Live preview, simulator, validation
- Reward Item Inspector: Card preview, quick setup presets
- Rarity Inspector: Color preview, theme presets, audio preview
- Pity Profile Inspector: Curve preview, interactive test
- Welcome Window and CraftRewardsEditorStyles

**Demo Scenes:**

- 01_BasicDraft: Roguelike card selection demo
- 02_GachaBanner: Full gacha system demo with pity UI
- 03_AutoLoot: Headless simulation demo

## 20. LICENSE AND SUPPORT

CraftRewards is licensed under the **Unity Asset Store EULA**. **Support:**

- Website: https://www.acehorizonstudio.com
- Email: support@acehorizonstudio.com

If you enjoy CraftRewards, please consider leaving a review on the Unity Asset Store. Your feedback helps us improve the tool.

**Thank you for using CraftRewards!**

CraftRewards © 2026 — Made with ❤️ for Unity Developers.