CraftMap: Roguelike Map Engine

Version 1.0.0 | User Manual & Documentation
Ace Horizon Studios

Table of Contents

1. INTRODUCTION

Thank you for purchasing CraftMap!

CraftMap is a complete procedural map generation tool designed for roguelike games. It provides everything needed to create, render, and manage node-based progression maps similar to those found in Slay the Spire, Inscryption, and other roguelike deck-builders.

Key Features

- Fully procedural map generation with **deterministic seeding**
- Configurable node types, distributions, and spacing rules
- Biome-based visual theming with fog of war
- Smooth bezier curve path rendering with shader-based flow animation
- Built-in UI components (tooltip, info panel, legend)
- Zero-allocation tween system for UI animations
- JSON serialization for save/load
- Cross-input support (mouse, touch, new Input System)
- Object pooling for zero GC during regeneration
- Custom editor inspectors with live map preview
- Full localization support

2. REQUIREMENTS

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

Optional:

- New Input System package (automatically detected via assembly define)

3. INSTALLATION

1. Import the CraftMap 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."* — if TMP Essentials are missing, click **Fix** to import them.
4. Open the demo scene:
   `CraftMap/Samples/Demo/Scenes/CraftMap_Demo.unity`
5. Press Play to see the map in action.

4. QUICK START GUIDE

Step 1: Create Node Profiles

Right-click in the Project window and select **Create > CraftMap > Node Profile**. Create at least: Start (NodeType: Start), Enemy (NodeType: Enemy), Boss (NodeType: Boss). Assign an icon sprite and tint color to each.

Step 2: Create a Map Configuration

Right-click and select **Create > CraftMap > Map Configuration**. Set rows (e.g., 15), min/max nodes per row (e.g., 2–4), add Node Weights, and configure Row Overrides (First Row: ForceType = Start, Last Row: ForceType = Boss). Use the Quick Presets bar for common layouts.

Step 3: Create a Biome Profile (Optional)

Right-click and select **Create > CraftMap > Biome Profile**. Configure background color, path colors, and fog of war settings. Use Theme Presets for quick setup and assign it to your Map Configuration's *Visual Theme* field.

Step 4: Set Up the Scene

Use the demo scene as a template, or build manually: create a Canvas (Screen Space – Overlay), add a ScrollRect with a content RectTransform, add MapManager, MapRenderer, MapScrollController, and MapInputHandler. Assign node/path prefabs and container references.

Step 5: Play

Press Play. The map generates automatically if *Generate On Start* is enabled. Click reachable nodes to progress through the map.

5. FOLDER STRUCTURE

All CraftMap files are located under Assets/CraftMap/ after import. The folder layout is as follows:

```
CraftMap/
Documentation/ PDF manual
Editor/ Custom inspectors, editor styles, welcome window
Runtime/
Core/ Colors, debug, localization, input, pooling, seed
Data/ ScriptableObjects and data classes
Generation/ Map generator, validator, weighted random
Rendering/ Visual components (renderer, nodes, paths, fog)
Serialization/ JSON save/load system
Systems/ Manager, audio, input, scroll, pulse
Tween/ Lightweight animation system
UI/ Tooltip, info panel, legend panel
Shaders/ Path flow and node glow shaders
Prefabs/ Node and path prefabs
Samples/Demo/ Example scene, profiles, icons
```

6. CORE CONCEPTS

CraftMap separates **Data** (what a map is) from **View** (how it looks), using ScriptableObjects for all configuration.

6.1 Map Configuration (MapConfiguration)

The central ScriptableObject that defines how maps are generated.

6.1.1 Grid Dimensions:

- **rows: Total number of rows (3–30)**
- **minNodesPerRow / maxNodesPerRow: Node count range per row**

6.1.2 Map Layout:

- **orientation**: Vertical (bottom-to-top) or Horizontal (left-to-right)
- **horizontalSpacing / verticalSpacing**: Distance between nodes
- **positionJitter**: Random offset for organic feel (0–50)

6.1.3 Connection Rules:

- **startingPaths**: Number of paths from the first row
- **convergenceRowsBeforeBoss**: Rows before last to start converging
- **allowCrossingPaths**: Whether paths may visually cross
- **maxConnectionsPerNode**: Max outgoing connections (1–4)

6.1.4 Node Distribution:

- **nodeWeights:** Array of NodeWeight entries — each pairs a NodeProfile with a relative weight. Higher weight = more common.

6.1.5 Row Overrides:

- **firstRowConfig** / **lastRowConfig** / **preLastRowConfig**: Each can be Random, ForceType, or Skip.

6.1.6 Editor Features:

- **Quick Presets:** Compact, Standard, Epic, Linear, Branching
- Live Map Preview with type distribution bar
- Real-time validation with error/warning display

6.2 Node Profiles (NodeProfile)

Defines the visual appearance and metadata of a node type.

- **profileId**: Auto-generated unique identifier (used for serialization)
- **displayName**: Name shown in UI
- **nodeType**: Enum value (Start, Enemy, EliteEnemy, RestSite, etc.)
- **description**: Text shown in the info panel
- **icon**: Sprite displayed on the node
- **tintColor**: Primary color applied to the node frame
- **scaleMultiplier**: Size multiplier (0.5–2.0)
- **isKeyNode**: Marks nodes with special importance

Editor Features: Live node preview with icon, tint, and scale. Profile ID shown as read-only.

6.3 Biome Profiles (BiomeProfile)

Defines the visual theme applied to the entire map.

6.3.1 Background:

- **backgroundSprite**: Optional tileable background
- **backgroundColor**: Fallback color or tint

6.3.2 Paths:

- **pathColor / visitedPathColor**: Unvisited and traveled path colors
- **pathWidth**: Bezier curve thickness (1–10)
- **pathFlowSpeed / pathFlowColor**: Dashed-line flow animation speed and color

6.3.3 Fog of War:

- **revealedAlpha**: Alpha for visited/reachable nodes (0–1)
- **adjacentAlpha**: Alpha for nearby visible nodes (0–1)
- **hiddenAlpha**: Alpha for distant nodes (0–1)
- **fogFadeSpeed**: Transition speed when fog changes

6.3.4 Editor Features:

- **Theme Presets:** Dark Dungeon, Forest, Ice Cavern, Volcanic, Mystic Void
- Fog gradient visualization bars

7. SCENE SETUP

The recommended hierarchy for your scene is:

```
[CraftMap_Manager] MapManager component
MapCanvas Canvas (Screen Space – Overlay recommended)
MapViewport ScrollRect + MapInputHandler + MapScrollController
MapContent RectTransform (ScrollRect.content)
MapBackground Image (auto-created by MapRenderer if missing)
PathContainer RectTransform (holds path instances)
NodeContainer RectTransform (holds node instances)
MapOverlay MapOverlay component (optional)
UIOverlay RectTransform
NodeTooltip NodeTooltip component (optional)
NodeInfoPanel NodeInfoPanel component (optional)
NodeLegendPanel NodeLegendPanel component (optional)
[CraftMap_Audio] MapAudioManager component (optional)
[CraftMap_FogOfWar] FogOfWar component (optional)
```

MapManager automatically finds child components via `GetComponentInChildren` if references are not manually assigned. Explicit assignment is recommended for clarity and performance.

Prefab Requirements:

- Node Prefab: Must have NodeView, CanvasGroup, RectTransform components
- Path Prefab: Must have PathRenderer, CanvasRenderer components

8. MAP MANAGER

MapManager is the central orchestrator. It coordinates generation, rendering, player progression, and all subsystems.

Key Methods

- GenerateAndRenderMap() — Generate with random or custom seed
- GenerateAndRenderWithSeed(int seed) — Generate with specific seed
- MoveToNode(int nodeId) — Attempt player movement
- RefreshVisuals() — Update all visual states
- SetConfiguration(MapConfiguration) — Change config at runtime

C# Events

- MapGenerated(MapGraph) — New map created
- NodeSelected(NodeData) — Node clicked
- PlayerMoved(NodeData) — Player moved to node
- MapCompleted(NodeData) — Final node reached
- NodeHoverStarted / NodeHoverEnded(NodeData) — Hover state changes

9. GENERATION SYSTEM

9.1 Default Generator

DefaultMapGenerator creates a Directed Acyclic Graph (DAG) through these steps:

1. **Grid Allocation:** Determine node count per row
2. **Node Creation:** Instantiate NodeData objects
3. **Connection Building:** Create forward-only edges
4. **Type Assignment:** Weighted random with spacing rules
5. **Row Overrides:** Apply forced types
6. **Position Calculation:** World positions with jitter
7. **Visibility Init:** Set initial fog of war state

All generation is 100% **deterministic** for a given seed. No Unity API calls during generation (pure C#).

9.2 Row Overrides

Row configurations control specific rows using RowMode:

- **RowMode.Random**: Normal weighted random (default)
- **RowMode.ForceType**: All nodes become a specific type
- **RowMode.Skip**: No nodes in this row (advanced)

9.3 Spacing Rules

NodeSpacingRule controls when and how often a type can appear:

- **targetType**: Which NodeType this rule applies to
- **minRow**: Cannot appear before this row index
- **minRowGap**: Minimum rows between occurrences

Example: EliteEnemy with minRow=6, minRowGap=3 — no elites before row 6, and after one appears, at least 2 empty rows before the next.

9.4 Custom Generators

Implement the IMapGenerator interface:

```
public interface IMapGenerator
{
MapGraph Generate(MapConfiguration config, int seed);
}
```

Requirements: must be 100% deterministic, pure (no side effects, no Unity API calls), and return a valid MapGraph with forward-only connections.

10. VISUAL SYSTEM

10.1 Map Renderer

MapRenderer manages the visual lifecycle: creates node and path instances from object pools, handles spawn animation with row-by-row staggering, manages background image from BiomeProfile, and provides `GetNodeView(int nodeId)` for lookups.

- **nodePrefab / pathPrefab:** Prefabs to instantiate
- **nodeContainer / pathContainer:** Parent transforms
- **animateSpawn:** Enable/disable entrance animation
- **rowDelay / nodeStagger / spawnDuration:** Animation timing

10.2 Node View

NodeView is the visual representation of a single node. Handles hover, click, pulse animation, and visual state.

- **_visualTarget:** Transform for scale animations
- **_iconImage:** Node type icon
- **_frameImage:** Colored frame ring
- **_glowObject:** Glow effect (active when reachable)
- **_visitedMarker:** Indicator for visited nodes

10.3 Path Renderer

PathRenderer draws bezier curves between nodes using Unity UI mesh generation. Features cubic bezier curves, state-based coloring (Visited, Available, Locked, Hidden), animated reveal effect, flow animation, and optional shader-based rendering.

- **CraftMap_FlowingPath:** GPU-based flow animation and glow
- **CraftMap_NodeGlow:** Radial gradient glow for nodes

10.4 Fog of War

FogOfWar manages node visibility based on player position:

- **Visited:** Full alpha (player has been here)
- **Reachable:** Full alpha (player can go here next)
- **Visible:** Partial alpha (nearby but not reachable)
- **Hidden:** Low alpha (far away — uses hiddenAlpha from BiomeProfile)

10.5 Map Overlay

MapOverlay draws a golden line through all visited nodes, showing the player's journey. Also manages an optional player marker with smooth movement and auto-clears on new map generation.

11. UI COMPONENTS

11.1 Node Tooltip

Floating tooltip that appears above the hovered node.

- Supports both TMP and Legacy Text
- Auto-flips when near screen edges
- Updates position every frame (follows scroll/zoom)

- Smooth fade and scale entrance animation
- Shows icon, name, and type description

11.2 Node Info Panel

Detailed panel showing node information with an action button.

- Icon with tinted background
- Title, description, and type label
- Status indicators (visited, reachable, locked)
- Colored status bar
- "Travel" action button (auto-disables for locked/visited)
- Slide animation with configurable easing

11.3 Node Legend Panel

Sidebar panel showing all node types present in the current map.

- Auto-populates from current MapGraph
- Supports custom prefabs via ILegendItemView interface
- Dynamic creation fallback (no prefab required)
- Slide toggle animation

12. NAVIGATION AND INPUT

12.1 Scroll Controller (MapScrollController)

Handles content sizing, centering, and smooth panning.

- Auto-calculates content size based on map bounds
- Centers map within viewport
- Smooth pan-to-node animation (SmoothStep easing)
- **_panSpeed**: Animation speed (1–20)
- **_panTargetAlignment**: Where node appears in viewport (0.2–0.8)
- **_initialScrollPosition**: Starting scroll position (0–1)

12.2 Input Handler (MapInputHandler)

Handles zoom via mouse scroll and touch pinch.

- Mouse scroll wheel zoom
- Touch pinch-to-zoom

- Zoom-to-cursor (focal point preserved)
- Smooth zoom interpolation
- Cross-input via InputHelper (supports Legacy and New Input System)

13. ANIMATION SYSTEM (CRAFTTWEEN)

CraftTween is a lightweight, zero-dependency tween system built specifically for CraftMap. It produces minimal GC allocation through object pooling.

Supported Operations

- `Scale(Transform, to, duration)`
- `ScalePunch(Transform, punchScale, duration)`
- `Move(RectTransform, to, duration)`
- `Fade(CanvasGroup, to, duration)`
- `ColorTo(Graphic, to, duration)`
- `Value(from, to, duration, callback)`
- `Delay(seconds, callback)`

Chaining

```
CraftTween.Scale(myTransform, Vector3.one * 1.2f, 0.3f)
.SetDelay(0.1f)
.SetEasing(Easing.OutBack)
.OnComplete(() => Debug.Log("Done"));
```

Easing Options

Linear, InQuad, OutQuad, InOutQuad, InCubic, OutCubic, InOutCubic, OutBack, OutElastic, OutBounce, InExpo, OutExpo

Kill Methods

```
CraftTween.KillAll(targetObject)
```

```
CraftTween.KillEverything()
```

```
handle.Kill(complete: false)
```

14. SERIALIZATION AND SAVE/LOAD

MapSerializer handles JSON serialization of MapGraph.

Save / Load (PlayerPrefs)

```
MapSerializer.SaveToPlayerPrefs(mapGraph, "MySave");
MapGraph graph = MapSerializer.LoadFromPlayerPrefs("MySave");
```

JSON String

```
string json = MapSerializer.ToJson(graph, prettyPrint: true);
MapGraph graph = MapSerializer.FromJson(json);
```

Utilities

```
bool exists = MapSerializer.HasSave("MySave");
MapSerializer.DeleteSave("MySave");
SaveInfo? info = MapSerializer.GetSaveInfo("MySave");
```

SaveInfo provides quick metadata: seed, nodeCount, totalRows, visitedCount, Progress (0–1), and isComplete. Version migration is handled automatically.

15. LOCALIZATION

CraftMapLocalization provides a simple key-value string system.

Get a string:

```
string label = CraftMapLocalization.Get("ui.button.travel");
```

Override a string:

```
CraftMapLocalization.Set("ui.button.travel", "Go");
```

Clear overrides:

```
CraftMapLocalization.ClearOverrides();
```

Default String Key Groups

```
nodetype.* — Node type labels (uppercase)
node.*.desc — Node descriptions
legend.* — Legend panel descriptions
tooltip.* — Tooltip descriptions
ui.button.* — Button labels (travel, visited, locked)
status.* — Status labels (reachable, visited, locked)
```

16. AUDIO

MapAudioManager handles all sound effects. Assign AudioClips in the Inspector:

- Node Hover Sound
- Node Click Sound
- Node Blocked Sound
- Path Reveal Sound
- Player Move Sound
- Map Generated Sound
- Map Complete Sound

Configuration

- Master Volume (0–1)
- Hover Volume / Click Volume (0–1)
- Hover Cooldown (prevents audio spam)

MapManager calls audio methods automatically. No manual wiring needed if MapAudioManager is assigned or exists as a child.

17. API REFERENCE

17.1 MapManager

```
void GenerateAndRenderMap()
void GenerateAndRenderWithSeed(int seed)
bool MoveToNode(int nodeId)
NodeData GetCurrentPlayerNode()
void SetConfiguration(MapConfiguration config)
void SetCustomSeed(string seed)
void RefreshVisuals()
MapGraph CurrentGraph { get; }
int CurrentSeed { get; }
bool IsCompleted { get; }
event Action MapGenerated;
event Action NodeSelected;
event Action PlayerMoved;
```

17.2 MapGraph

```
NodeData GetNode(int id) // O(1)
bool TryGetNode(int id, out NodeData node)
IReadOnlyList GetNodesInRow(int row)
IReadOnlyList GetReachableNodes() // Cached
bool IsNodeReachable(int nodeId)
MapStatistics GetStatistics()
```

17.3 MapSerializer

```
static string ToJson(MapGraph graph, bool prettyPrint = false)
static MapGraph FromJson(string json)
static void SaveToPlayerPrefs(MapGraph graph, string key)
static MapGraph LoadFromPlayerPrefs(string key)
static bool HasSave(string key)
static void DeleteSave(string key)
static SaveInfo? GetSaveInfo(string key)
```

17.4 CraftMapLocalization

```
static string Get(string key)
static void Set(string key, string value)
static void SetBulk(Dictionary strings)
static void ClearOverrides()
static string GetTypeLabel(NodeType type)
static string GetNodeDescription(NodeType type)
```

17.5 CraftTween

```
static TweenHandle Scale(Transform, Vector3 to, float duration, Easing)
static TweenHandle Move(RectTransform, Vector2 to, float duration, Easing)
static TweenHandle Fade(CanvasGroup, float to, float duration, Easing)
static TweenHandle ColorTo(Graphic, Color to, float duration, Easing)
static TweenHandle Delay(float delay, Action callback)
static void KillAll(Object target, bool complete = false)
TweenHandle SetDelay(float) / SetEasing(Easing) / OnComplete(Action)
```

18. EDITOR TOOLS

18.1 Map Configuration Inspector

- Quick Presets bar (Compact, Standard, Epic, Linear, Branching)
- Live Map Preview showing generated nodes and connections
- Preview controls: seed input, Random, Regenerate, toggle paths/labels
- Node type distribution bar with percentages

- Statistics panel (rows, avg nodes, connections, type counts)
- Real-time configuration validation

18.2 Node Profile Inspector

- Live node preview with icon, tint color, and scale
- Read-only profile ID display
- Organized sections: Identity, Visuals, Advanced

18.3 Biome Profile Inspector

- Theme preview with path colors, fog levels, and flow indicator
- Theme Presets: Dark Dungeon, Forest, Ice Cavern, Volcanic, Mystic Void
- Color palette swatches and fog gradient visualization

18.4 Map Manager Inspector

- System Status panel (shows which references are assigned)
- Runtime State display (seed, nodes, progress, visited count)
- Quick Actions (Regenerate button in play mode)
- Organized sections: Definition, Core Systems, UI, Events

18.5 Welcome Window

- Access via: Tools > CraftMap > Welcome
- System integrity checks (TMP, resources)
- Quick Start buttons for creating assets
- Demo scene launcher and documentation links
- Auto-shows on first import (can be disabled)

19. PERFORMANCE NOTES

Object Pooling

- Nodes and paths are pooled via ObjectPool
- Zero Instantiate/Destroy calls during map regeneration
- Configurable pool sizes in MapRenderer inspector

GC Allocation

- CraftTween uses internal object pooling (max 128 instances)
- MapGraph caches reachable nodes — invalidated only on player movement
- MapPulseManager uses struct-based targets
- Debug logs stripped from production builds via [Conditional]

Recommendations for Mobile

- Keep total nodes under 200
- Use 10–15 curve segments for paths (default: 20)
- Disable flow animation shader on low-end devices
- Consider reducing fog visible range

20. TROUBLESHOOTING AND FAQ

Q: Nodes are not visible / text is invisible

A: Ensure TMP Essentials are imported: Window > TextMeshPro > Import TMP Essential Resources.

Q: Map does not generate

A: Check the configuration validation messages in the Inspector. Ensure at least one valid NodeWeight entry exists.

Q: Paths look wrong / no animation

A: Verify the CraftMap_FlowingPath shader is included in your build. Add it to Project Settings > Graphics > Always Included Shaders.

Q: Touch zoom does not work

A: Call InputHelper.InitializeTouchSupport() at startup. MapInputHandler does this automatically in Awake().

Q: Save/Load produces errors

A: Ensure the MapGraph was fully generated before saving. Check that JSON is not empty or corrupted.

Q: Custom generator maps are invalid

A: Run GraphValidator.Validate(graph) to find issues. Ensure all connections are forward-only and every non-terminal node has incoming and outgoing connections.

Q: Nodes overlap visually

A: Reduce positionJitter or increase spacing values. Use the live preview to check layout before runtime.

Q: How do I add new node types?

A: Add a new value to the NodeType enum in Enums.cs, create a corresponding NodeProfile asset, add default colors in CraftMapColors.cs, and add localization strings.

Q: How do I change the map at runtime?

A: Call MapManager.SetConfiguration(newConfig) then MapManager.GenerateAndRenderMap().

Q: How do I integrate with my game logic?

A: Subscribe to MapManager events: MapManager.Instance.NodeSelected += OnNodeSelected; then read NodeData.nodeType in your handler to determine the encounter.

21. CHANGELOG

Version 1.0.0 — Initial Release

- Procedural DAG map generation with deterministic seeding
- 8 built-in node types: Start, Enemy, EliteEnemy, RestSite, Treasure, Shop, Mystery, Boss
- Configurable row overrides and spacing rules
- Biome-based visual theming
- Fog of war with configurable visibility
- Bezier curve path rendering with flow shader
- Built-in UI: tooltip, info panel, legend panel
- CraftTween animation system
- JSON serialization with version migration
- Localization support
- Cross-input support (mouse, touch, new Input System)
- Object pooling for zero-GC regeneration
- Custom editor inspectors with live preview
- Welcome window with system checks
- Demo scene with full setup

22. LICENSE AND SUPPORT

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

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

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

Thank you for using CraftMap!

CraftMap © 2026 — Made with ❤️ for Unity Developers.