RTS Fog of War

Documentation

Version: 1.3.1

Online Documentation: https://rts-fogofwar.netlify.app

Krzysztof Bziuk

Table of contents

1. Fog of War RTS

Fog of War RTS is a Unity package that brings the classic "fog of war" mechanic from strategy games like Age of Empires and StarCraft to your project. Unexplored areas remain completely hidden, while previously discovered regions appear dimmed when not actively observed. The system is optimized for real-time strategy (RTS), top-down, and 2D games, efficiently handling thousands of units simultaneously.

1.1 Watch the Demo

See the system in action in the demonstration video below.

YouTube video

1.2 How It Works

The fog of war divides your map into three visibility states:

- **Hidden Areas:** Unexplored regions that have never been discovered by any vision source.
- **Explored Areas:** Previously discovered locations that are not currently in vision range.
- **Visible Areas:** Regions currently within your units' vision range.

As units move, the fog updates automatically in real-time with smooth transitions between states.

1.3 Key Features

- **Customizable Fog Appearance:** Use custom textures and colors for both unexplored and explored fog to match your game's art style.
- **Real-Time Updates:** Fog responds instantly to unit movement with smooth fade transitions between visibility states.
- **High Performance:** Efficiently processes large maps with thousands of units without impacting frame rates.
- **Simple Integration:** Drag-and-drop components make setup straightforward - attach a VisionSource to any unit to reveal fog.
- **UI Map Fog Overlay:** Apply fog of war effect to your custom UI minimaps with a single component.
- **Visibility Events:** Control object behavior based on fog state using UnityEvents or C# events.
- **Vision Layers:** Enable different visibility rules for units on separate floors.
- **Smooth Object Fading:** Built-in Fader component for seamless fade-in and fade-out transitions.
- **Persistent Discovery:** Save and restore explored areas between game sessions.
- **Flexible Refresh Modes:** Choose between automatic real-time updates or manual control for turn-based gameplay.
- **Top-Down and 2D Support:** Works with both XZ orientation (top-down games) and XY orientation (2D side-scrolling games).

1.4 Main Components

- `FogOfWarManager` : Core component managing all fog calculations, map boundaries, and visual configuration.
- `VisionSource` : Reveals fog in circular, square, or triangular patterns.
- `VisibilitySwitch` : Receives visibility events to control object behavior based on fog state.
- `Fader` : Provides smooth fade-in and fade-out transitions for renderers using custom shaders.
- `MapFogOverlayUI` : Applies fog of war overlay to your custom UI minimap.

2. Quick Setup

This guide provides a streamlined workflow for integrating the RTS Fog of War system into your Unity project.

2.1 Setup Steps

1. Import the Fog of War package into your Unity project.
2. Create a new scene or open an existing scene.
3. Create a ground plane (GameObject > 3D Object > Plane) to serve as your playable area.
4. Create an empty GameObject and name it "Fog Manager":
5. Add the `FogOfWarManager` component.
6. Configure the **Bounds** properties to match your ground plane's dimensions.
7. Create a GameObject within the defined bounds area:
8. Add the `VisionSource` component.
9. Adjust the **Scale** property to set the vision radius.
10. Enter Play Mode - the fog should now cover the entire area except for the region around the VisionSource.

3. Components

3.1 FogOfWarManager

The `FogOfWarManager` component is the core component required for the fog of war system to function. It handles all fog of war calculations and serves as the central configuration point for the system.

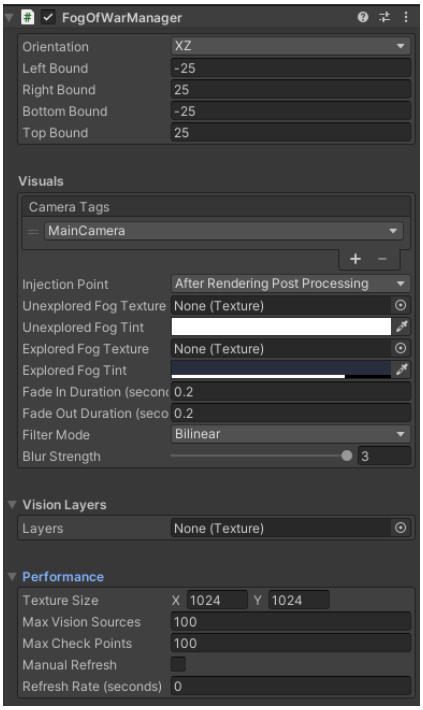
> **Info**
>
> If you disable the `FogOfWarManager` component or the GameObject it's attached to, the fog effect will completely disappear and all VisibilitySwitch components will be visible.

> **Warning**
>
> Only one `FogOfWarManager` instance can be present in the scene at any time. If multiple instances are detected, the system will automatically destroy the entire GameObject containing any duplicate instances, keeping only the first one that was initialized. An error message will be logged in development builds to help identify the issue.

3.1.1 Usage

1. Create a new GameObject and add the `FogOfWarManager` component to it.
2. Configure the component properties in the Inspector.

3.1.2 Inspector Properties



- **Orientation (enum):** Defines the map's orientation plane.
  - **XZ** - For top-down games (horizontal plane)
  - **XY** - For 2D side-scrolling games (vertical plane)
- **Left Bound:** The left edge coordinate of the fog-covered area.
- **Right Bound:** The right edge coordinate of the fog-covered area.
- **Bottom Bound:** The bottom edge coordinate of the fog-covered area.
- **Top Bound:** The top edge coordinate of the fog-covered area.

Visuals

- **Camera Tags:** A list of camera tags that determine which cameras will render the fog effect. The fog overlay will be applied to any camera containing at least one tag from this list. By default, only the `MainCamera` tag is included.
- **Injection Point:** Specifies when the fog effect is injected into the camera rendering pipeline.
- **Unexplored Fog Texture:** Texture for unexplored (hidden) areas.
- **Unexplored Fog Color:** RGB color tint for the fog texture in unexplored areas.
- **Explored Fog Texture:** Texture for explored but not currently visible areas.
- **Explored Fog Color:** RGBA color tint for the explored fog texture. Alpha controls the opacity level.
- **Fade In Duration (min 0):** The duration in seconds for the fog to fade in when an area becomes hidden.
- **Fade Out Duration (min 0):** The duration in seconds for the fog to fade out when an area becomes visible.
- **Filter Mode (enum):** The texture filtering method applied to the final fog texture.
    - **Point** - No filtering, produces sharp, pixelated edges
    - **Bilinear** - Smooth linear interpolation between pixels
    - **Trilinear** - Highest quality filtering with mipmap blending
- **Blur Strength (0-3):** The intensity of the blur effect applied to fog edges. Higher values produce smoother fog transitions. Set to 0 to disable blur entirely.

Vision Layers

- **Layers:** An optional texture that defines regional vision restrictions on the map. Each color channel (R, G, B, A) corresponds to a vision layer. See Vision Layers for usage.

Performance

- **Texture Size (min 1, max 16384):** The resolution of the fog texture (Vector2Int). For optimal results, use power-of-two values (e.g., 512, 1024, 2048). Higher values increase memory usage and computational cost.
- **Max Vision Sources (min 1):** The maximum number of VisionSource components that can be active simultaneously. Vision sources exceeding this limit will be ignored. Higher values increase computational cost.
- **Max Check Points (min 0):** The maximum number of visibility check points from VisibilitySwitch components. Check points exceeding this limit will be ignored. Higher values increase computational cost.
- **Manual Refresh:** Enabling this option requires fog updates to be triggered manually via the `RefreshThisFrame()` method.
- **Refresh Rate (min 0):** *(This property appears only if Manual Refresh is disabled)* The time interval in seconds between automatic fog refresh cycles. A value of 0 means the fog updates every frame.

3.1.3 Public API

- `static FogOfWarManager Instance` - The singleton instance of the `FogOfWarManager` (read-only). Returns the active `FogOfWarManager` in the current scene.
- `void RefreshThisFrame()` - Forces an immediate fog update on the current frame. Primarily used when Manual Refresh mode is enabled.
- `Vector2Int[] GetDataToSave()` - Returns the current fog discovery state as a compressed array of `Vector2Int` ranges. Use this data to save the fog state.
- `void Load(Vector2Int[] data)` - Restores the fog state from previously saved data. Accepts the data array returned by `GetDataToSave()`.

3.2 VisionSource

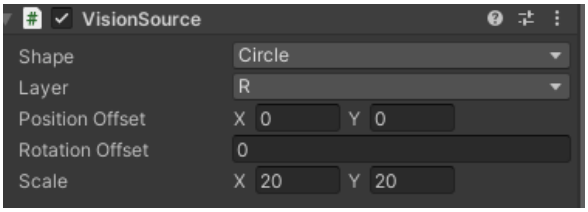
The `VisionSource` component defines an area of vision that reveals fog of war in real-time. It tracks the GameObject's position and rotation from the transform, while allowing additional offsets and scale adjustments.

> **Rotation Behavior**
>
> The rotation behavior depends on the **Orientation** setting in the FogOfWarManager:
>
> - **XZ orientation** (top-down games): Uses the GameObject's Y-axis rotation
> - **XY orientation** (2D side-scrolling games): Uses the GameObject's Z-axis rotation

3.2.1 Usage

1. Add the `VisionSource` component to any GameObject that should reveal fog.
2. Configure the component properties in the Inspector.

3.2.2 Inspector Properties



- **Shape (enum):** The geometric shape used for fog revelation.
  - **Circle** - Circular vision area
  - **Square** - Square vision area
  - **Triangle** - Triangular vision area
- **Layer (flags):** Defines on which layers fog is revealed. See Vision Layers.
- **Position Offset:** Position offset relative to the GameObject's position.
- **Rotation Offset (0-360):** Rotation offset in degrees added to the GameObject's transform rotation.
- **Scale (min 0):** Size multiplier of the vision shape. Independent of the GameObject's transform scale.

3.2.3 Public API

- `VisionShape Shape` - Gets or sets the geometric shape of the vision area (`Circle`, `Square`, or `Triangle`).
- `VisionLayer Layer` - Gets or sets the vision layer flags.
- `Vector2 PositionOffset` - Gets or sets the position offset relative to the GameObject's position.
- `float RotationOffset` - Gets or sets the rotation offset in degrees. Values are automatically normalized to the 0-360 range.
- `Vector2 Scale` - Gets or sets the size multiplier. Minimum value is `Vector2.zero` (values below zero are clamped). Independent of the GameObject's transform scale.

3.3 VisibilitySwitch

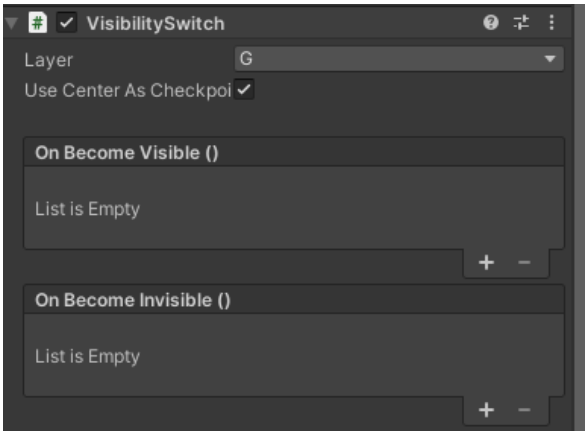
The `VisibilitySwitch` component enables GameObjects to receive events about visibility state changes in the fog of war system. The object is considered visible when at least one of its checkpoint positions is within range of any VisionSource with a matching layer. Checkpoint positions are offsets relative to the GameObject's position and rotation.

> **Rotation Behavior**
>
> The rotation behavior depends on the **Orientation** setting in the FogOfWarManager:
>
> - **XZ orientation** (top-down games): Uses the GameObject's Y-axis rotation
> - **XY orientation** (2D side-scrolling games): Uses the GameObject's Z-axis rotation

3.3.1 Usage

1. Add the `VisibilitySwitch` component to any GameObject that should receive events about fog visibility changes.
2. Configure the component properties in the Inspector.

3.3.2 Inspector Properties



- **Layer (flags):** Defines on which layers the object is visible. See Vision Layers.
- **Use Center As Checkpoint:** When enabled, uses only the GameObject's center position `(0, 0)` as a single checkpoint. When disabled, allows you to define custom checkpoint positions in the Check Points list.
- **Check Points:** *(This property appears only if "Use Center As Checkpoint" is unchecked)* A list of Vector2 positions relative to the GameObject's position and rotation. The object is considered visible if any checkpoint position is within range of a VisionSource with matching layers. At least one checkpoint is required.
- **On Become Visible:** A UnityEvent triggered when the object transitions from hidden to visible state.
- **On Become Invisible:** A UnityEvent triggered when the object transitions from visible to hidden state.

3.3.3 Public API

- `bool IsVisible` - Returns `true` if the object is currently visible, `false` if hidden (read-only).
- `VisionLayer Layer` - Gets or sets the vision layer flags.
- `event Action<bool> OnVisibilityChanged` - A C# event triggered when visibility state changes. The boolean parameter is `true` when becoming visible, `false` when becoming hidden.

3.4 Fader

The `Fader` component provides smooth fade-in and fade-out transitions for renderers based on fog of war visibility changes.

> **Custom Shader Required**
>
> The `Fader` requires a custom shader with a float property (default: `_FadePercent`) that controls transparency. The shader must use this property to fade the material's alpha channel. Sample shaders demonstrating this functionality are available in the **Samples** section of the package.

> **Required Component**
>
> The `Fader` component requires a `VisibilitySwitch` component on the same GameObject. It will automatically retrieve the visibility state from the attached `VisibilitySwitch`.

> **Automatic Renderer Management**
>
> At the end of fade out, all renderers in the list are automatically disabled by the `Fader` component. You do not need to manually disable renderers in your `VisibilitySwitch` event handlers (`On Become Invisible`). The `Fader` handles this automatically.

3.4.1 Usage

1. Ensure your GameObject has a `VisibilitySwitch` component attached.
2. Add the `Fader` component to the same GameObject.
3. Configure the fade properties in the Inspector.

3.4.2 Inspector Properties

- **Fade Duration (min 0):** The duration in seconds for fade in/out transitions.
- **Material Fade Percent Name:** The name of the material shader property used to control the fade effect. Default is `_FadePercent`. This property should be a float value between 0 (fully transparent/hidden) and 1 (fully visible) in your shader.
- **Renderers:** An array of Renderer components to which the fade effect will be applied.

3.5 MapFogOverlayUI

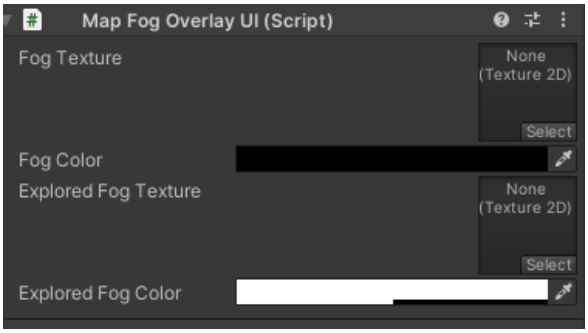
The `MapFogOverlayUI` component applies fog of war overlay to a UI image, allowing you to add the fog effect to your custom map.

> **Required Component**
>
> The `MapFogOverlayUI` component requires either an `Image` or `RawImage` component on the same GameObject.

3.5.1 Usage

1. Create a GameObject under your Canvas.
2. Add an `Image` or `RawImage` component to the GameObject and assign your map texture.
3. Add the `MapFogOverlayUI` component to the same GameObject.
4. Configure the fog properties in the Inspector.

3.5.2 Inspector Properties



- **Unexplored Fog Texture:** Texture for unexplored (hidden) areas on the minimap.
- **Unexplored Fog Color:** RGB color tint for the fog texture in unexplored areas.
- **Explored Fog Texture:** Texture for explored but not currently visible areas on the minimap.
- **Explored Fog Color:** RGBA color tint for the explored fog texture. Alpha controls the opacity level.

4. Vision Layers

Vision layers are an optional feature that allows you to control which layers reveal fog of war and which objects can be detected.

4.1 Demo Video

See the vision layers system in action in the demonstration video below.

Vision Layers Demo

4.2 What Are Vision Layers?

The system provides four layer flags (R, G, B, A) that can be configured in the FogOfWarManager. Each layer can represent different floors. An object is only visible when its VisibilitySwitch layer matches at least one layer flag of a VisionSource.

4.3 How to Set Up

4.3.1 1. Configure Layers Texture

In the FogOfWarManager Inspector, you can assign a **Layers** texture where each color channel (R, G, B, A) defines a separate vision layer on the map:

- **R channel** = Red layer
- **G channel** = Green layer
- **B channel** = Blue layer
- **A channel** = Alpha layer

This texture allows you to create map regions with different visibility rules. Leave empty if you only want to use component-based layers.

4.3.2 2. Set VisionSource Layer

On each VisionSource component, set the **Layer** property to define which layers this vision source reveals fog on.

4.3.3 3. Set VisibilitySwitch Layer

On each VisibilitySwitch component, set the **Layer** property to define on which layers this object can be detected.

4.4 Common Use Cases

Use layers to represent different floors in multi-level buildings:

- **R Layer** = Ground floor (Floor 0)
- **G Layer** = First floor (Floor 1)
- **A Layer** = Second floor (Floor 2)
- **B Layer** = Flying units (Floor 3)

Example:

- Ground units: VisionSource reveals fog only on R layer
- First floor units: VisionSource reveals fog on R and G layers
- Flying units: VisionSource reveals fog on all layers

This creates realistic line-of-sight where ground units can't reveal fog on upper floors, but units on higher floors can reveal fog downward.

4.5 Gameplay Benefits

Multi-level environments:

- Tactical advantage from high ground
- Flying units valuable for reconnaissance

Stealth gameplay:

- Hidden units requiring special detection
- Asymmetric visibility mechanics

5. Examples

The package includes demo scenes that demonstrate the features of the fog of war system.

5.1 Simple Example

This demo scene demonstrates all core features of the fog of war system in action.

5.1.1 Controls

- Use the arrow keys to move the camera.
- Press 'M' to toggle the minimap between minimap and fullscreen map modes.
- Click on player units, then click on the map to move them.

5.1.2 What the demo shows

- `VisionSource` components on player units that dynamically reveal fog.
- `VisibilitySwitch` and `Fader` components demonstrating visibility events:
    - Enemies and large objects (like buildings) appear and disappear based on fog coverage.
    - Custom Shader Graph transparent shader used for fading enemies smoothly.
    - Enemy map markers are enabled/disabled based on visibility.
- `MapFogOverlayUI` component:
    - Demonstrates how to create a custom minimap with map markers.
    - Minimap displays fog coverage in real-time.
    - Toggle between minimap and fullscreen map views.
- Sample quest scenario:
    - UI directs the player to a marked location.
    - Discovering the area updates the quest and reveals part of the map.

5.2 Layers Example

This demo scene demonstrates the Vision Layers system for elevation-based visibility.

5.2.1 What the demo shows

- Multi-floor building environment with different elevation levels.
- Units on different floors with layer-based vision:
    - Ground units can only see ground level.
    - Units on higher floors can see their floor and below.
    - Flying units can see all elevations.
- Practical application of the vision layer system for multi-level gameplay.

6. Performance

6.1 Performance Benchmarks

Performance benchmarks were conducted to analyze the relationship between Vision Source count, refresh intervals, and frame rate performance. Testing parameters included a 4096x4096 resolution fog texture with uncapped frame rate.

Vision Sources \ Refresh Interval	0	0.0167	0.02
0	540 FPS	1100 FPS	1200 FPS
1000	530 FPS	1100 FPS	1200 FPS
3000	520 FPS	1095 FPS	1200 FPS
5000	500 FPS	1095 FPS	1190 FPS
10000	480 FPS	1090 FPS	1185 FPS

6.1.1 Benchmark Analysis

The benchmark results show excellent performance characteristics:

Low Unit Count Performance: Adding the first 1000 vision sources causes almost no performance drop - the system handles them with minimal overhead. This is great news for typical RTS games, which usually have a few hundred units at most.

Test Conditions: These benchmarks used an extremely large 4096x4096 fog texture, which is much larger than most games need. This texture size would only be necessary for very large maps. In most real-world scenarios, a 1024x1024 texture will provide good visual quality while running significantly faster than the results shown above.

7. Advanced

This page covers advanced technical details including URP rendering requirements, automatic configuration validation and global shader properties.

7.1 URP Depth Texture Requirement

Requires the **Depth Texture** feature to be enabled in the Universal Render Pipeline (URP) asset for proper functionality.

7.1.1 Automatic Validation

An automatic validator checks and enables the Depth Texture setting when the project loads in the Unity Editor. If disabled, the validator will:

1. Automatically enable `Depth Texture` in your active URP asset
2. Save the changes to the asset
3. Display a warning message in the Console with a reference to the modified asset

> **Warning**
>
> Disabling the Depth Texture after it has been automatically enabled may cause the fog effect to not render correctly.

7.2 Global Shader Properties

Global shader properties are automatically set by the `FogOfWarManager` and can be accessed in custom shaders.

Property Name	Type	Description
`_FogOfWar_Enabled`	`half`	Fog enabled state (0 = disabled, 1 = enabled)
`_FogOfWar_OrientationXZ`	`float`	Map orientation (0 = XY plane, 1 = XZ plane)
`_FogOfWar_WorldToFogParams`	`float4`	World-to-fog UV transformation - **xy**: scale values (inverse of map size) - **zw**: UV offset values
`_FogOfWar_MaskTexture`	`Texture2D`	Mask texture containing visibility data - **Red channel (x)**: discovered areas - **Green channel (y)**: inverse of visible areas
`_FogOfWar_UnexploredFogTexture`	`Texture2D`	Texture for unexplored (hidden) fog areas
`_FogOfWar_UnexploredFogColor`	`half3`	RGB color tint for unexplored fog (linear color space)
`_FogOfWar_ExploredFogTexture`	`Texture2D`	Texture for explored but not currently visible areas
`_FogOfWar_ExploredFogColor`	`half4`	RGBA color and alpha for explored fog (linear color space)