

# Fog of War

---

*User Guide*

*v1.23*

## Overview

The definitive Fog of War package empowers you to fog up your 2D or 3D game, hiding secret elements from the player. FogOfWar is highly modular, customizable and lightening fast to meet the needs of any RTS, MOBA or adventure game that needs a thick shroud of mist around it.

Features:

- Supports Legacy, URP and HDRP
- Works with 2D and 3D
- Varying map size
- Chunking system for infinite maps in all 3 dimensions
- Different color, texture, filter and blurring options
- Queue-friendly tools
- Great performance (including multithreading)
- Line of Sight (occluding objects)
- View cones (using angles)
- Clear Fog (ie see through to background)
- See-through ground
- Works for both Orthographic and Perspective cameras
- Render to multiple cameras
- Save and load fog between plays
- Option for manual fog updates for turn-based games
- Flexible minimap creation
- Compatible with all devices, including mobile and VR
- All source code provided
- In active development and fast email support

## Updates

### v1.24

- Fixed URP orthographic not working
- Fixed HDRP error in URP sample file
- Documentation improvements

### v1.23

- URP support! (finally)
- Removed major memory allocation in hardware renderer
- Optimised texture retrieval in hardware renderer
- Fixed typo in demo scene
- Fixed some inaccurate rounding errors with fog pixel/cell visibility calculations
- Fixed sample scene having HDRP camera components on them
- Fixed rare shader error about unknown "modepos"
- Added lineOfSightRaycastAngle, lineOfSightRaycastOffset and lineOfSightSeeOutsideRange to FogOfWarUnit to reduce raycast count
- Added CustomFogShader to demonstrate how to make a world-space shader that reacts to fog, as well as the FogOfWarTeam.ApplyToMaterial method
- Added Manual Update button to FogOfWarTeam inspector
- Added FogOfWarTeam.onTextureChanged event
- Added mesh shape type for FogOfWarUnits
- Added sample mesh for Mesh shape type

### v1.22

- Fixed wrong inverse view matrix in legacy and PPSv2 shader
- Made FogOfWar appear in root of volume overrides for HDRP
- Made fog color default to clear for HDRP to work around a unity bug where disabling the volume override doesn't work

### v1.21

- Added support for HDRP 7.1.8+
- Changed multithreading in hardware mode warning to an error
- Improved FogOfWarMinimap performance improvement a bit
- Added Save() and Load() to FogOfWarChunkManager
- Improved FogOfWarChunkManager memory display in editor
- Made FogOfWar shaders automatically included to the always included shader list
- Fixed "Look rotation viewing vector is zero" warning in 3D demo
- Added gizmos for FogOfWarTeam map area
- Added FogOfWarTeam.GetVisibleUnits()
- Improved platform support for demos
- Fixed 2D demo no putting chunks into a parent object
- Put in a hardware accelerated minimap in demo scene

#### v1.20

- Fixed error in shaders not being able to find utils.cginc
- Added a warning that multithreading is not supported in hardware mode
- Fixed snapToPixelCenter not working on all axes
- Added AspectRatioFitter to FogOfWarMinimap
- Fixed render texture release error when using hardware
- Fixed FogOfWarMinimap going transparent when scaled too small
- Added a warning that hardware mode is experimental

#### v1.19

- PPSv2: Made FoW render before the stack
- Added checks for null or unsupported shaders
- Removed old way to do clear fog, and added FogOfWarClearFog component to allow more flexibility
- Fixed antiFlicker not working
- Renamed antiFlicker to snapToPixelCenter in FogOfWarUnit
- Added new FogOfWarMinimap component
- Fixed HideInFog.minFogStrength not working properly
- Split fade duration into fade in and fade out durations
- Added option to multithread raycasts in FogOfWarUnit (utilizing RaycastCommand), required Unity 2018.1
- Removed a lot of the duplicated shader code
- Added FogOfWarTeam.outputToTexture
- Fixed OutOfRange exception when calling FogOfWarTeam.SetFog()

#### v1.18

- Fixed shader build error preventing fog from being seen

#### v1.17

- *This release will break existing setups. Please read the migration section in the user guide!*
- Added support for Postprocessing Stack v2!
- Split FogOfWarTeam from visual side to make new post processing tech easier to migrate to
- Better multi-camera support
- Optimised clear fog
- Improved sample scenes to work across multiple render pipelines

#### v1.16

- Fixed \_CameraWS shader error
- Removed dependency on SV\_VertexID
- Exposed raycasts count for line of sight
- Added gizmos for line of sight raycasts
- Removed all general runtime GC!
- Improved performance, memory for clear fog and test GUI
- Optimised fading when updateUnits is turned off
- Fixed bug where FogOfWar.SetFog() and FogOfWar.SetAll() wouldn't increase the fog

- Fixed some misleading comments in FogOfWar script
- Fixed bug with cell based when map resolution and size are not 1:1
- Fixed screen being upside down on some graphics APIs
- Set min thread count to 2
- Fixed threads not setting count immediately
- Fixed a bunch of warnings that can pop up during play

#### v1.15

- Added an array size check when setting FogOfWar.fogValues
- Fixed flickering when switching chunks with the chunk manager in multithreaded mode
- Fixed flickering when performing a manual update in multithreaded mode
- Made tree materials more diffuse in sample scenes
- Turned off UI by default in sample scenes
- Fade speed now affects fade in and out speeds
- Added boolean to stop unit information from being updated
- Changed manual updates to a updateAutomatically boolean value
- Changed texture shapes to always use the Alpha8 texture format to save memory usage
- Made it so Partial Fog Amount can be changed without destructively affecting the fog map
- Added VR support
- Fixed a bunch of bugs with shape offset
- Added Reinitialize button to FogOfWar component when playing in the editor
- Added rectangular box and texture shapes
- Merged box and texture unit shapes
- Added pixel perfect toggle for screen texture
- Changed filter mode to point filtering toggle
- Fixed major performance issue with clear fog
- Changed fade speed to fade duration (in seconds)
- Made texture shapes not cause errors with multithreading
- Fixed some weird behaviour that can happen when not fogging the far plane

#### v1.14

- Hopefully fixed all of the line ending warnings
- Fixed flickering when switching chunks with the chunk manager in single threaded mode
- Made fog color texture work in world space with options to adjust it
- Made Reinitialize() accessible through the FogOfWar context menu
- Added updateAutomatically and ManualUpdate() to FogOfWar that lets you manually update the fog which is useful for turn-based games or to save on performance when the fog isn't being updated
- Added warning for when there isn't a team for the team specified in HideInFog

#### v1.13

- Added missing unfog methods
- Fixed wrong projection with Direct3D 9 renderers
- Made it so you can use just one thread for multithreading
- Made penetration depth work for cell based fog
- Fixed line of sight for 3D physics on XY plane

- Made FogOfWarUnity.rotateToForward work for XY and YZ planes
- Made it so multiple teams can be handled on the same device

#### v1.12

- Added chunking system for infinitely sized maps on all 3 axes
- Made FogOfWar.Reinitialize() much faster
- Added fix for clear fog
- Added Canvas and Graphic hiding in HideUnitInFog script
- Fixed pixel inaccuracy around map borders
- Fixed pixel inaccuracy with line of sight
- Added FogOfWar.VisibilityOfArea()
- Added blurring and antialiasing options
- Added antiflickering option for individual units

#### v1.11

- Added multithreading
- Fog color's alpha will fade the fog in and out
- New fog shapes including box and texture (with sample texture)
- Anti-aliasing on field of view
- Added team identification
- New method for calculating line of sight which greatly reduces the number of raycasts

#### v1.10

- Fixed shader bug in Unity 5.5.0 where depth buffer was inverted for orthographic cameras
- Made fog collider rect optional with a tick box as it seems to be buggy with lower resolution fog maps (a fix will be required for later releases!)
- Fixed bug where fog wasn't clearing on the border edges of the map
- Added slider to specify the strength of fog outside of the map area
- Added HideInFog component to automatically hide object when in the fog
- Added FogOfWarEvents component to detect when an object enters or exits fog
- Optimised fog of war shader by removing all branching
- Placed all components into components menu
- Added example 2D scene
- Fixed bug on mobile devices where depth buffer was flipped

#### v1.9

- Fixed bug where screen would go black when the unity editor loses focus
- Improved performance by converting shader property ids to ints on initialization
- Made fog values public

#### v1.8

- Made fogValues setter public to enable manual setting of fog (for loading games)

#### v1.7

- Improved test scenes with enemies, better trees and smoother movements
- Put all code in FoW namespace
- Added toggle for max fog distance
- Made fog values publicly accessible via property
- Added method to calculate explored percentage
- Split fog planes and physics (ie 2D and 3D)

#### v1.6

- Fixed bug where fog was rendering on camera's far plane
- Fog can have textures applied to them
- Unit vision angles/cones

#### v1.5

- Add Secondary FoW to allow for multiple cameras

#### v1.4

- Added support for 2D (along the X and Y axes)

#### v1.3

- Fog now works with orthographic cameras

#### v1.2

- Added Clear Fog
- Included TransparentShadowCaster (see-through ground) shader

#### v1.1

- Implemented Line of Sight
- Added new assets to demo
- Demo minimap now displays units
- Updated user guide considerably
- Added new method: Unfog(Rect)
- Camera zooming in the demo scene

#### v1.0

- Basic Fog of War implemented
- Basic demo

## Setup

**IMPORTANT:** After importing the package you will probably have compile errors. Please remove the render pipeline assets you won't be using. You can find these in the *FogOfWar/Core/* and *FogOfWar/Samples/* folders. After this step, you should have no compile errors.

Place a **FogOfWarTeam** component anywhere in the scene. This will represent the fog that is visible for an individual team.

The next step is to link a camera to the fog. The steps required to do this depends on which tech you are using:

## Legacy Renderer

Attach a **FogOfWarLegacy** component to the camera you want to use.

Before building, you will need to specify which shaders will go into the build. This should already happen automatically but, if it doesn't, you can follow these steps:

1. Go to **Edit / Project Settings...**
2. Click on **Graphics**
3. Add the following shaders to the *Always Included Shaders* list (under *Built-in Shader Settings*):
  - *FogOfWar/Core/Legacy/FogOfWarLegacy.shader*
  - *FogOfWar/Core/FogOfWarHardware.shader* (if you want to use hardware mode)
  - *FogOfWar/Core/FogOfWarBlurShader.shader* (if you want to use blurring features)
4. Delete the following folder (if you don't want to use the Post Processing Stack v2):
  - *FogOfWar/Core/PPSv2/*

## PPSv2

This is all done through the Post Processing Stack v2 (PPSv2).

1. Ensure that you have downloaded the Post Processing Stack v2 through the Package Manager (see [here](#) for details).
2. Put a **PostProcessVolume** component somewhere in the scene. Make sure you tick *IsGlobal*.
3. Create a **PostProcessProfile** in the project.
4. Add a **FogOfWarPPSv2** effect to the **PostProcessProfile**.
5. Add a **PostProcessLayer** to the camera that will be displaying the fog.
6. The fog will be fully transparent by default, so tick **Fog Color** on the **PostProcessProfile** and increase the alpha.

For more information on the PPSv2, be sure to check out the official documentation for it.

Before building, you will need to specify which shaders will go into the build. This should already happen automatically but, if it doesn't, you can follow these steps:

1. Go to **Edit / Project Settings...**
2. Click on **Graphics**

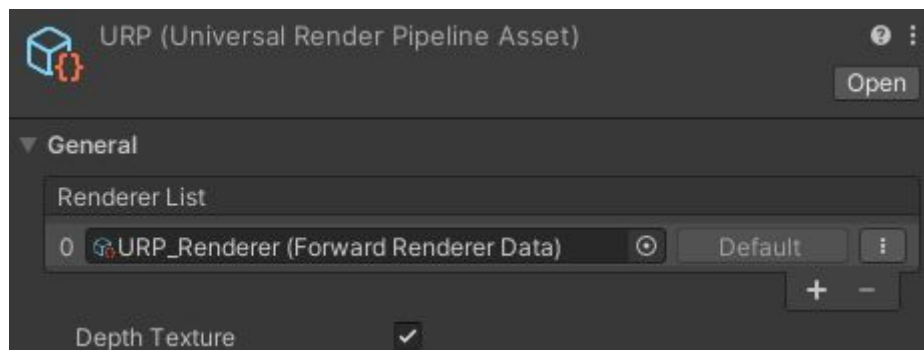


3. Add the following shaders to the *Always Included Shaders* list (under *Built-in Shader Settings*):
  - *FogOfWar/Core/PPSv2/FogOfWarPPSv2.shader*
  - *FogOfWar/Core/FogOfWarHardware.shader* (if you want to use hardware mode)
  - *FogOfWar/Core/FogOfWarBlurShader.shader* (if you want to use blurring features)

## URP

This is all done through the Volume post processing system.

1. Locate your **UniversalRenderPipelineAsset** (if you are unsure, you can find it in **Edit / Project Settings / Graphics / Scriptable Render Pipeline Settings**)
2. Tick **Depth Texture**
3. Ensure there is a **RenderData** in the **Renderer List** (at the top)



4. In the **RenderData**, click on **Add Renderer Feature / Fog Of War URP Feature**
5. On your camera, go to the **Rendering** section and ensure that **Depth Texture** is set to either **On** or **Use Pipeline Settings**
6. The fog will be fully transparent by default, so tick **Fog Color** on the volume profile, and increase the alpha.

Before building, you will need to specify which shaders will go into the build. This should already happen automatically but, if it doesn't, you can follow these steps:

4. Go to **Edit / Project Settings...**
5. Click on **Graphics**
6. Add the following shaders to the *Always Included Shaders* list (under *Built-in Shader Settings*):
  - *FogOfWar/Core/URP/FogOfWarURP.shader*
  - *FogOfWar/Core/FogOfWarHardware.shader* (if you want to use hardware mode)
  - *FogOfWar/Core/FogOfWarBlurShader.shader* (if you want to use blurring features)

## HDRP

This is all done through the Volume post processing system.

1. Go to the **Project Settings**.
2. In the **HDRP Default Settings** tab, go to the **Custom Post Process Orders**.
3. Under **Before Post Process**, add **FogOfWarHDRP** to the list.
4. The fog will be fully transparent by default, so tick **Fog Color** on the volume profile, and increase the alpha.

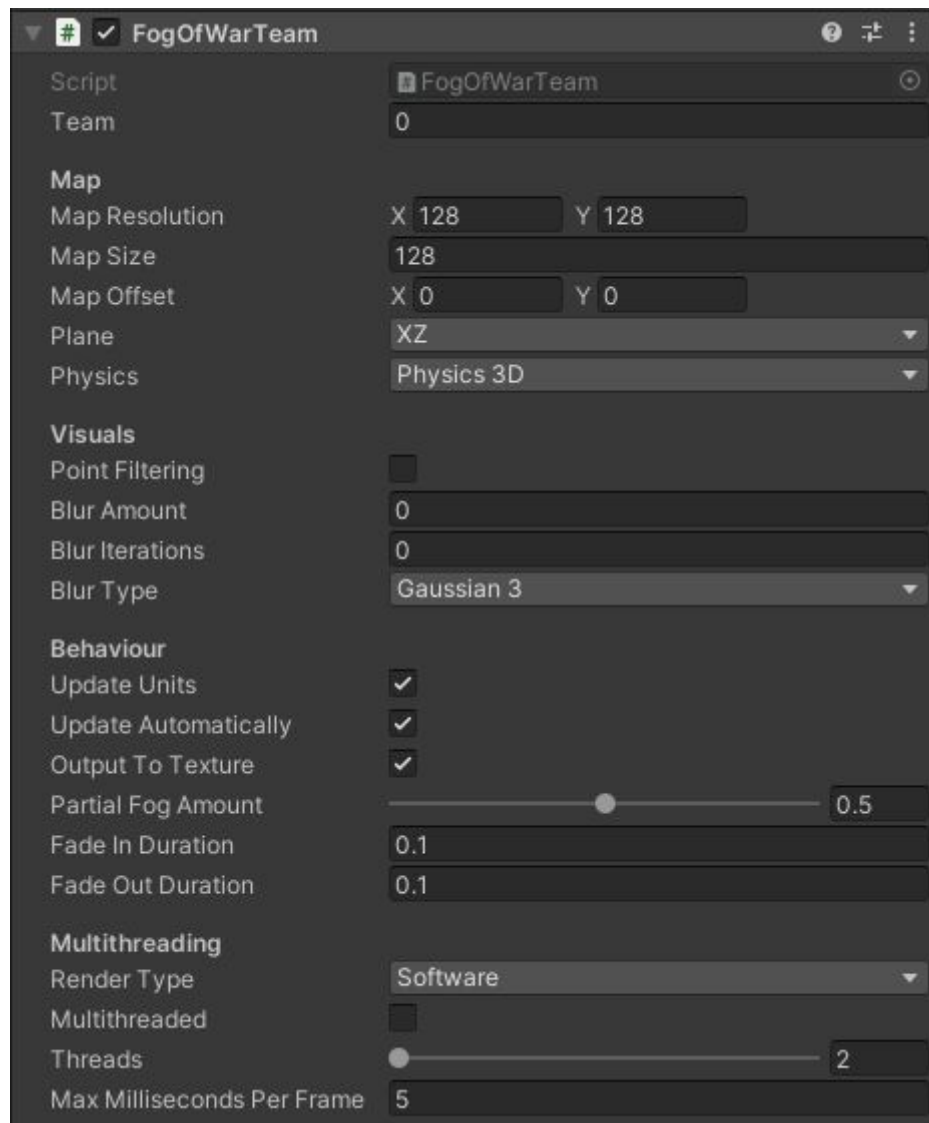
For more information on the HDRP volume system, be sure to check out the official documentation for it.

Before building, you will need to specify which shaders will go into the build. This should already happen automatically but, if it doesn't, you can follow these steps:

1. Go to **Edit / Project Settings...**
2. Click on **Graphics**
3. Add the following shaders to the *Always Included Shaders* list (under *Built-in Shader Settings*):
  - *FogOfWar/Core/HDRP/FogOfWarHDRP.shader*
  - *FogOfWar/Core/FogOfWarHardware.shader* (if you want to use hardware mode)
  - *FogOfWar/Core/FogOfWarBlurShader.shader* (if you want to use blurring features)

## Components

### FogOfWarTeam



*FogOfWarTeam* represents everything that a single team sees. There must be one of these for each team. At least one needs to be in the scene.

You can see what area the map will cover by looking for the blue box that will cover the scene in the scene viewport.

To save and load the fog values, there is a **GetCurrentFogValues(ref byte[])** method that stores the fog values. You can set it with **SetCurrentFogValues(byte[])** at any time after the fog has been initialized. There are also the **GetTotalFogValues(ref byte[])** and **SetTotalFogValues(byte[])** that can get/set the fog including what's visible to current units.

When in play mode, a Reinitialize button will appear. Clicking this will reset the fog and completely start the fog process again, which can help to debug issues.

When **Update Units** and **Update Automatically** are turned off, a **Manual Update** button will appear. Pressing this will let you simulate **ManualUpdate()** being called.

There are two modes for rendering the fog maps, each with different capabilities:

	Software	Hardware
Per-pixel Accuracy	✓	✗
Blurring	✓	✓
Multithreading	✓	✗
Cell Based	✓	✗
Unit Textures	Not in Multithreaded Mode	✓
Non-Readable Textures	✗	✓
Lots of Units	Slow	✓
High Resolution	Slow	✓

<b>Team</b>	The index of which team this fog will clear for. All units that have the same team index will have the fog cleared.
<b><u>Map</u></b>	
<b>Map Resolution</b>	The resolution of the texture used to render the fog. Setting this to a high value can have a significant performance impact, so try to keep it as low as possible. It is recommended to always be a power of 2. If using chunking, this must always be square.
<b>Map Size</b>	The size of the map that the fog will cover in world coordinates. If using chunking, this should be the smallest maximum area that the player can see if the player is in the center of the map.
<b>Map Offset</b>	If the map is not centered at the origin (0, 0) you can offset it with this. This should be left to zero when using chunking.
<b>Plane</b>	Which plane the fog will be rendered to. 3D is usually XZ and 2D is usually XY
<b>Physics</b>	What physics will be used for raycasting and collision detection. This is Unity's built-in 2D or 3D.
<b><u>Visuals</u></b>	
<b>Point Filtering</b>	If unticked, bilinear filtering will be used, and point filtering will be used if ticked. Bilinear Filtering will have a smoother look, Point Filtering will have a pixelated feel to it.
<b>Blur Amount</b>	How far the blur will spread. This is measured in pixels of the fog texture. If this is zero, no blurring will be applied.
<b>Blur Iterations</b>	How many time the blur will be applied. If this is zero, not blurring will be applied.
<b>Blur Type</b>	The blurring algorithm that will be used. Gaussian 3 is slightly faster but usually doesn't give as good results as Gaussian 5.
<b><u>Behaviour</u></b>	

<b>Update Units</b>	If turned off, units will not update. This saves a lot of performance if doing a turn-based game.
<b>Update Automatically</b>	When set to true, the fog will be updated automatically every frame. If false, you will need to call <b>FoW.FogOfWarTeam.GetTeam(0).ManualUpdate()</b> from code to update the fog. This is useful for turn-based games.
<b>Output To Texture</b>	When set to true, fog will be rendered to the texture. Set this to false if you want to update the fog but the fog is not being rendered to the screen. This can give a significant performance boost when dealing with multiple teams and there is only one team's fog being rendered.
<b>Fade In Duration</b>	How long it will take for fog to fade in (in seconds).
<b>Fade Out Duration</b>	How long it will take for fog to face out (in seconds).
<b>Multithreading</b>	
<b>Render Type</b>	Which technique will be used to render the fog. Software is currently the most stable and has the most features supported. It will give the best performance for low resolution maps (128 and lower) with a low number of units (less than 8). Hardware is currently experimental and is lacking some features. It will give the best performance when rendering a high resolution map and/or with lots of units.
<b>Multithreading</b>	If true, multithreading will be enabled.
<b>Threads</b>	The total number of threads to be used on Fog of War. If Multithreading is not ticked, this will do nothing.
<b>Max Milliseconds Per Frame</b>	The maximum number of milliseconds that Fog Of War will spend on each frame. This applies to both single threaded and multithreaded Fog Of War. A higher value will slow performance but reduce stuttering.

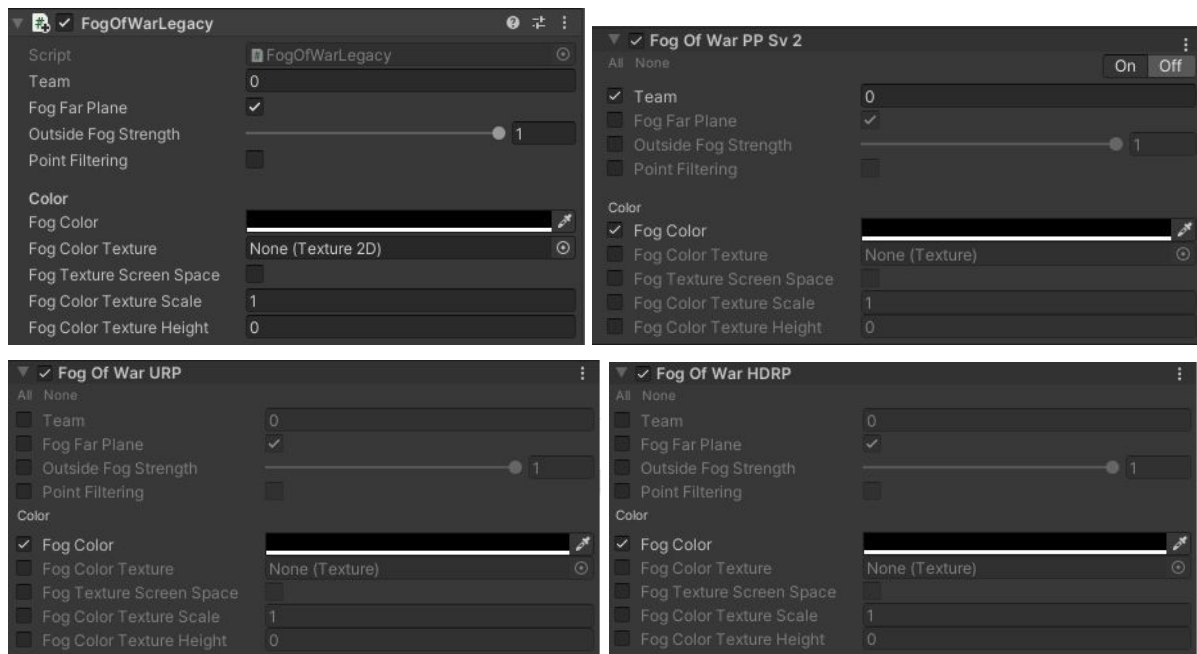
## Useful Functions

<b>static FogOfWarTeam GetTeam(int teamID)</b>	Returns the FogOfWarTeam for a particular team. If no team exists, null will be returned.
<b>void Reinitialize()</b>	Reinitializes fog texture. Call this if you have changed the mapSize, mapResolution or mapOffset during runtime. This will also reset the fog. You can manually call this from the editor by right-clicking the FogOfWar component.
<b>int mapByteSize</b>	This property is how large an array must be when calling GetCurrentFogValues() or GetTotalFogValues(). This will always be <b>mapResolution.x * mapResolution.y</b> .
<b>void GetCurrentFogValues(ref byte[] values)</b>	Copies the current fog values into the specified array. The current fog values is the fog that was cleared from the last frame. If you want the fog as it is displayed on-screen, use GetTotalFogValues(). Returned values are 0 for completely unfogged, and 255 for completely fogged. The size of the array should be mapResolution * mapResolution.

<b>void SetCurrentFogValues(byte[] currentvalues)</b>	<p>Copies the specified array into the current fog values.</p> <p>The fog values is the fog that was cleared from the last frame.</p> <p>If you want the fog as it is displayed on-screen, use SetTotalFogValues().</p> <p>Returned values are 0 for completely unfogged, and 255 for completely fogged.</p> <p>The size of the array should be mapResolution * mapResolution.</p>
<b>void GetTotalFogValues(ref byte[] totalvalues)</b>	<p>Copies the total fog values into the specified array.</p> <p>The total fog values is the fog as it appears on screen from the last frame.</p> <p>If you want the fog.</p> <p>Returned values are 0 for completely unfogged, and 255 for completely fogged.</p> <p>The size of the array should be mapResolution * mapResolution.</p>
<b>void SetTotalFogValues(byte[] totalvalues)</b>	<p>Copies the specified array into the total fog values.</p> <p>The total fog values is the fog as it appears on screen from the last frame.</p> <p>If you want the fog.</p> <p>Returned values are 0 for completely unfogged, and 255 for completely fogged.</p> <p>The size of the array should be mapResolution * mapResolution.</p>
<b>float ExploredArea(int skip = 1)</b>	<p>Returns how much of the map has been explored/unfogged, where 0 is 0% and 1 is 100%.</p> <p>Increase the skip value to improve performance but sacrifice accuracy.</p>
<b>Vector2i WorldPositionToFogPosition(Vector3 position)</b>	<p>Converts a world position to a fog pixel position. Values will be between 0 and mapResolution.</p>
<b>byte GetFogValue(Vector3 position)</b>	<p>Returns the fog amount at a particular world position. 0 is fully unfogged and 255 if fully fogged.</p>
<b>void SetFog(Bounds bounds, byte value)</b>	<p>Set the fog for a square area of the map. Positions are all in world coordinates. 0 is fully unfogged and 255 if fully fogged.</p>
<b>void SetAll(byte value)</b>	<p>Sets the fog value for the entire map. Set to 0 for completely unfogged, to 255 for completely fogged.</p>
<b>float VisibilityOfArea(Bounds worldbounds, byte value)</b>	<p>Checks the average visibility of an area. 0 is fully unfogged and 1 if fully fogged.</p>
<b>void GetVisibleUnits(int teamIndex, byte maxGog, List&lt;FogOfWarUnit&gt; units)</b>	<p>Returns a list of all of the units in a specific team that are visible to the current team. The threshold of visibility is specified with the maxFog value, where 0 is fully unfogged and</p>

	255 is fully fogged. The list of units will not be cleared when called.
<b>ApplyToMaterial(Material material, float outsidefogstrength = 1)</b>	Applies all properties to the specified material that are required to detect fog for this team.
<b>void ManualUpdate(float timeSinceLastUpdate)</b>	Forces the fog to update. This should only be called when updateAutomatically is true. You can manually call this from the editor by right-clicking the FogOfWar component.
<b>onTextureChanged</b>	Callback for whenever the fog has been changed.

## FogOfWar Display



Depending on which post processing technology you are using, you will need to set this up differently.

	FogOfWarLegacy	PPSv2	Volume System
<b>Legacy</b>	✓	✓	✗
<b>URP</b>	✗	✓	✓
<b>HDRP</b>	✗	✗	✓

*FogOfWarLegacy* must be placed on the cameras you want to display the fog of war. This will use Unity's **OnRenderImage()** calls.

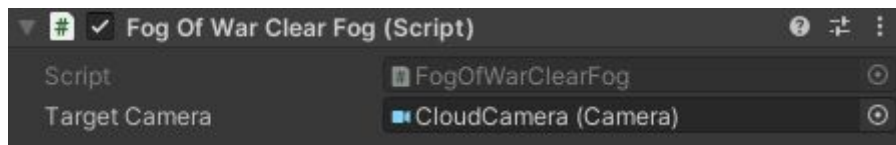
Unity's *Postprocessing Stack v2* is required for LWRP and HDRP. You may need to [install](#) it in your project if it isn't already. On your post processing volume, click on *Add Effect...* and pick *FogOfWar*.

HDRP in Unity 2019.1 (using their volume system) doesn't yet allow for custom post processing effects. FogOfWar will support this soon.

<b><u>Map</u></b>	
<b>Team</b>	The index of which team will be displayed. This will be the same index as is set on the <i>FogOfWarTeam</i> component.
<b>Fog Far Plane</b>	If true, anything at the camera's far plane will have fog rendered over it. If you want to see the skybox, set this to false. It is slightly better for performance to have this turned on.
<b>Outside Fog Strength</b>	Specifies how strong the fog will be outside of the fog map area. 1 = fully fogged, 0 = no fog.
<b>Point Filtering</b>	If unticked, bilinear filter will be used, and point filtering will be used if ticked. Bilinear Filtering will have a smoother look, Point Filtering will have a pixelated feel to it.
<b><u>Color</u></b>	
<b>Fog Color</b>	The color of the fog. When using clear fog, the alpha value will determine how transparent the fogged area will be (you usually want the alpha to be zero).
<b>Fog Color Texture</b>	A texture that can be applied over the top of the fog.
<b>Fog Texture Screen Space</b>	If true, the displayed texture will be in screen space. <i>fogColorTextureScale</i> and <i>fogColorTextureHeight</i> will not be used.
<b>Fog Color Texture Scale</b>	The size that the fog color texture appears to be in world space.
<b>Fog Color Texture Height</b>	The illusion of height that the fog color texture is at.



## FogOfWarClearFog



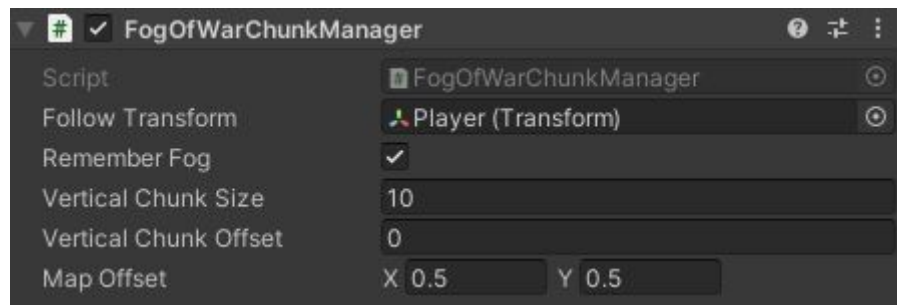
Clear Fog enables you to render the contents of a camera as the fog instead of just a plain color. For example, you may want to render clouds of a starry-space backdrop.

This component must be placed on the same GameObject as the FogOfWarTeam component!

When setting up the camera, you will usually want to set its Layer Mask to something different.

<b>Target Camera</b>	The camera that will be used to render over the fog.
----------------------	--

## FogOfWarChunkManager



*FogOfWarChunkManager* will automatically move the map to center on the player. This allows for an infinite map that can run infinitely in all directions without a huge performance impact. The chunk manager also allows for vertical chunks. This is useful for games that cover multiple floors.

This component must be attached to the same *GameObject* as the *FogOfWarTeam* component!

If you are making an RTS or a game where the whole scene must be updated at once, you should **not** use the chunk manager. The chunk manager works best with games where you control only one character through a large environment.

There are some important things to note when using the chunk manager:

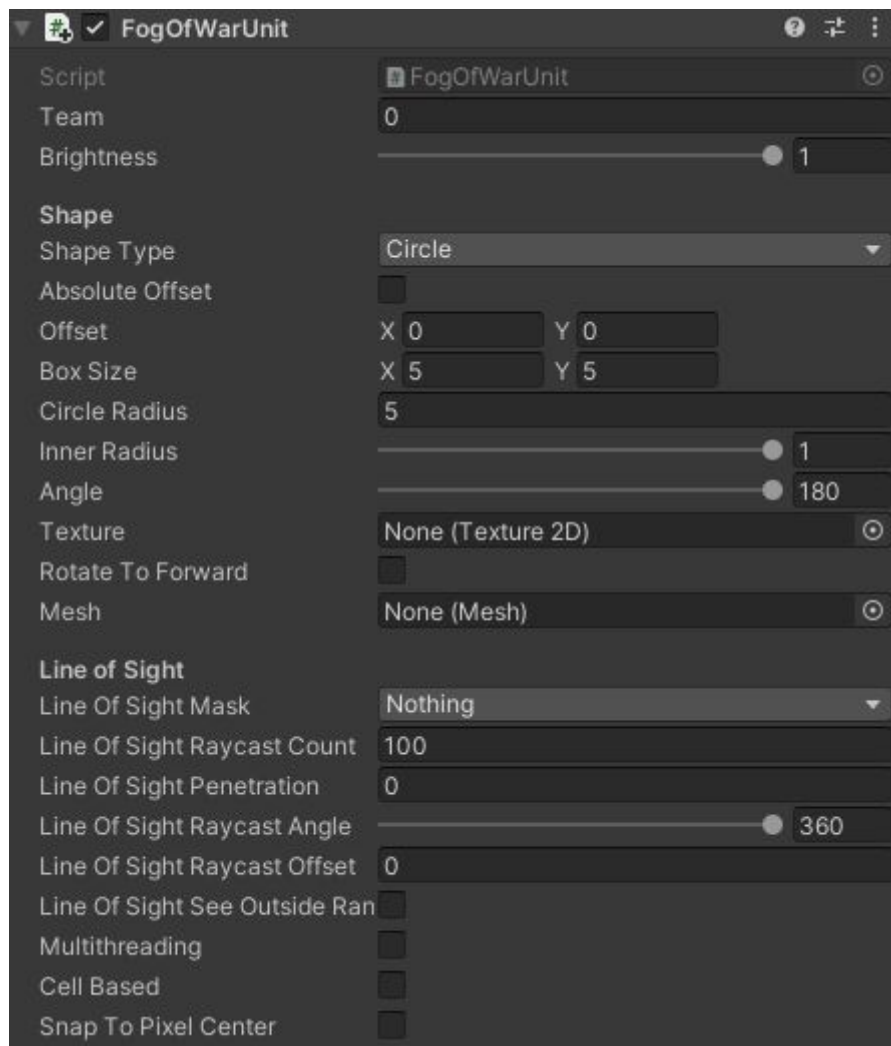
- The map resolution needs to be square and divisible by 2. It is recommended to use a power of 2.
- The map size is the size of the map that will be visible at any one time. This means that it should not cover the entire scene. It should be the maximum distance that the player can see at any time.
- The map offset has no effect when the chunk manager is enabled

<b>Follow Transform</b>	The game object that the fog should follow. This is usually the player.
<b>Remember Fog</b>	If false, the chunk manager will completely discard all fog information once a chunk is left.
<b>Vertical Chunk Size</b>	The size of a floor before the next vertical chunk is loaded. This should be the distance from one floor to another.
<b>Vertical Chunk Offset</b>	The offset from zero where the first vertical chunk is.
<b>Map Offset</b>	The offset that the chunks will change at. By default (with this set to 0, 0), the border will lie over the world origin.

### Useful Functions

<b>byte[] Save()</b>	Saves all of the chunk data to a byte array.
<b>bool Load(byte[] data)</b>	Loads a previously saved byte array. For this to work, the <b>mapResolution</b> on the <b>FogOfWarTeam</b> component must be the same as it was when it was saved.

## FogOfWarUnit



*FogOfWarUnit* should be placed on every GameObject that will reveal areas in the fog.

When using textures on Box shapes, the textures **must** be a single channel texture with the Alpha8 format.

When using Mesh shapes, the mesh must be readable for software mode, and must be on the same plane as the fog plane (as set on the **FogOfWarTeam** component). If the mesh has changed at runtime, call **RefreshMesh()**.

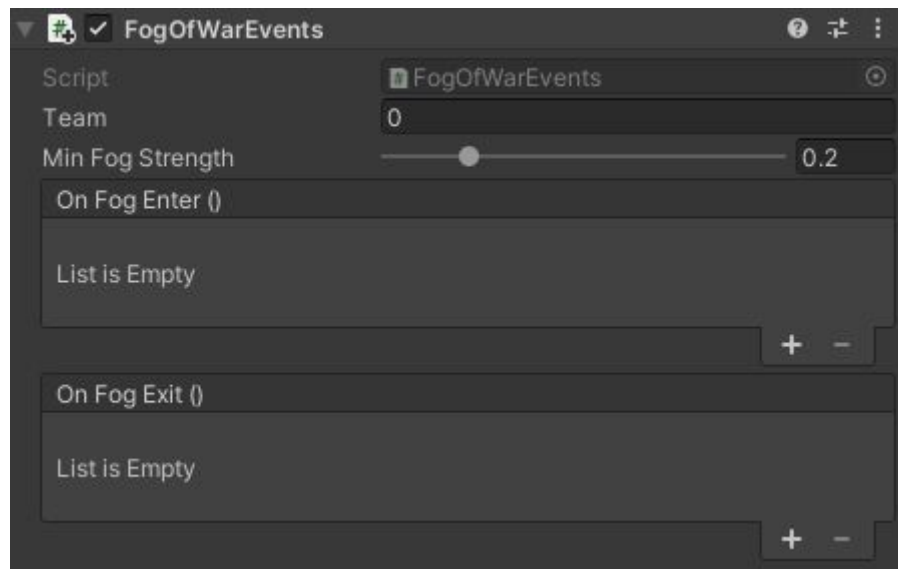
When using Software mode (on the FogOfWarTeam component), textures must have **Read/Write Enabled** in the texture import settings.

When line of sight is enabled, red lines are drawn in the viewport to visualize what the line of sight system is doing.

<b>Team</b>	The index of the team that this unit is a part of. This must match the Team value on the Fog of War component for the fog to be cleared.
<b>Brightness</b>	The maximum amount that this unit can reveal the fog. This should usually be set to 1, unless you want to do some sort of lighting effects with it.
<b>Shape</b>	

<b>Shape Type</b>	Can be Circle, Box or Mesh.
<b>Absolute Offset</b>	If false, <b>Offset</b> will be relative to the unit's rotation. If true, the value will be in world space (along the fog plane).
<b>Offset</b>	The offset from the unit's position where the shape's center will be.
<b>Box Size</b>	The size of the box. Only applicable for Box shapes.
<b>Circle Radius</b>	How far the unit can see. This is used to define the circle shape.
<b>Inner Radius</b>	How smooth the edge of the circle is. Only applicable to Circle shapes.
<b>Angle</b>	View cone angle. 180 degrees is full vision, 0 degrees is no vision. The forward direction will always be Y+ for 2D and Z+ for 3D.
<b>Texture</b>	The texture that will be used to generate the shape. This only applies to Box shapes. This must be a black and white texture where white is visible, and black is fogged. Make sure you set the texture to readable in the import settings!
<b>Rotate To Forward</b>	This only applies to texture shapes. If true, the texture will always face the same direction as the unit. If false, the texture will always be oriented to the world.
<b>Mesh</b>	This only applied to mesh shapes. This sets the mesh to be used. The mesh must be on the same plane as the FogOfWarTeam component's plane. Call RefreshMesh() whenever changing the mesh at runtime. If using software mode, ensure that the mesh is readable.
<b><u>Line of Sight</u></b>	
<b>Line Of Sight Mask</b>	A layer mask that specifies which objects will occlude the unit's vision. Be sure that the occluding objects have a collider on it and that they are at the same vertical height as the unit to be properly occluded. Enabling this can have a significant impact on performance with large radii.
<b>Line Of Sight Raycast Count</b>	The number of rays cast from the unit's center to calculate the visible area of the unit. Keep this number as low as possible for best performance.
<b>Line Of Sight Penetration</b>	How far through objects that the unit's vision will penetrate through.
<b>Line Of Sight Raycast Angle</b>	The maximum angle that raycasts are used.
<b>Line Of Sight Raycast Offset</b>	The offset of the raycast angle start point. Use this to rotate the raycast angles.
<b>Line Of Sight See Outside Range</b>	If true, anything beyond the range specified in LineOfSightRaycastAngle will be visible.
<b>Multithreading</b>	Enables multithreaded raycasting using the job system. This option is only available in Unity 2018.1 and higher.
<b>Cell Based</b>	Enable this if you are doing a tile-based game where one tile is the same size as one pixel on the fog map. This will make line of sight check collision properly. You would generally want to leave <b>Line Of Sight Penetration</b> to zero if this is enabled.
<b>Snap To Pixel Center</b>	Enable this if flickering is occurring when using line of sight. This will snap the unit's position to the nearest fog pixel. The downside to this is that movements will become more jarring with slower movements with lower resolution maps. You can minimize this by applying a blur/antialiasing.

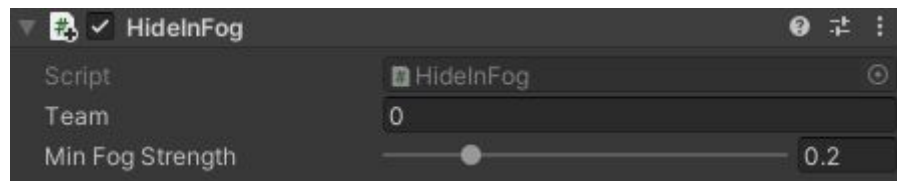
## FogOfWarEvents



*FogOfWarEvents* allows you to set events of what should happen when this GameObject enters or exits the fog. The position is based off the GameObject's Transform.

<b>Team</b>	Which FogOfWar team will be tracked.
<b>Min Fog Strength</b>	The minimum fog strength until the OnFogEnter is registered.
<b>On Fog Enter</b>	An event that is called when the object enters fog that is a greater strength than MinFogStrength.
<b>On FogExit</b>	An event that is called when the object exits fog that is a greater strength than MinFogStrength.

## HideInFog



*HideInFog* will hide the object when it enters fog. The object will be made invisible by enabling and disabling the Renderer, UI Canvas or UI Graphic attached to this GameObject.

<b>Team</b>	Which FogOfWarTeam will be tracked.
<b>Min Fog Strength</b>	The minimum fog strength until the object is made invisible.

## FogOfWarMinimap



**WARNING: It is not recommended to use this due to performance issues and may be removed in future releases. See the FAQ for a better solution.**

A simple minimap display that can be attached to a RawImage UI. This must be on the same GameObject as a RawImage component.

This may not do everything that you may want. Feel free to use the code as a template to make it do more of what you want. If you want even more flexibility, consider using a separate camera with a RenderTexture instead.

<b>Team</b>	Which FogOfWarTeam will be tracked.
<b>Fog Color</b>	The color of fully fogged areas.
<b>Non Fog Color</b>	The color of non-fogged areas.
<b>Unit Color</b>	The color for the selected team's units.
<b>Enemy Color</b>	The color for units that are not on the selected team.
<b>Camera Color</b>	The color of the camera. If no camera is specified, this will not appear on the minimap.
<b>Opponent Min Fog Strength</b>	The minimum amount of fog for an enemy unit to be visible, where 0 is fully unfogged, and 1 is fully fogged.
<b>Camera</b>	The camera to track (optional).
<b>Icon Size</b>	The size (in pixels) of units and the camera.
<b>Aspect Ratio Fitter</b>	An aspect ratio fitter component that will set the aspect ratio to the same as FogOfWarTeam.mapResolution (optional).

## Third-Party Support

### RTS Engine

Created by SOUMIDELRIO

Asset Store Page: <https://www.assetstore.unity3d.com/en/#!/content/79732>

SOUMIDELRIO has kindly created a package to allow you to easily integrate Fog Of War into RTS Engine. You can find information on it and download it here:

<http://soumidelrio.com/docs/unity-rts-engine/third-party-support/fog-of-war/>



## Using Custom Shaders

FogOfWar is implemented as a post process, but it is possible to apply it only in specific shaders. This can have various benefits, including improved performance (assuming there is minimal overdraw) and transparency support. Unfortunately, it is not possible for FogOfWar to support every type of shader out of the box, so if you would like to take this approach, you will need to set it up manually.

You can find a sample shader at **FogOfWar/Samples/CustomFogShader.shader**. This is an unlit shader which works in all render pipelines, and applies fog to just this material. You should modify it to support your specific shaders.

In order for it to work, you must pass in the correct information into the shader. This can be done by calling **FoW.FogOfWarTeam.GetTeam(0).ApplyToMaterial(material)**. This should only be called when first initializing, and when **FogOfWarTeam.onTextureChanged** is called.

## Migrating From 1.16

FogOfWar 1.17 introduced big changes to how everything works. This is a checklist to ensure proper migration:

- A *FogOfWarTeam* component must be placed in the scene. There should be one for each team.
- Set up your camera depending on which post processing tech you are using:
  - Legacy Pipeline: Add the *FogOfWarLegacy* component to your camera.
  - Post Processing Stack v2: Add the *FogOfWar* effect to your post processing volume and ensure that your camera has a *PostProcessingLayer* component on it.
- FogOfWarSecondary no longer exists.
- Any code that used **FogOfWar.GetFogOfWarTeam()** must be changed to **FogOfWarTeam.GetTeam()**.

## FAQs

### Why does the fog appear to update at a slow frame rate?

You can easily change the update frequency in the **FogOfWarTeam** component.

Updating the fog every frame for every unit can have a massive performance impact. To combat this, units update the fog individually. After a set amount of time, the fog is rendered and then the loop starts again.

If you only have a few units, updating the fog each frame may be viable. But if you have hundreds of units, the frame rate will become unbearable.

### What is Clear Fog and how do I get it to work?

Clear Fog enables you to replace the fog with the rendering of another camera. This can be used to create weird effects where the fog won't be just a single color (such as a space backdrop or animated clouds).

To get it to work, there are a few steps you must take:

1. Setup a camera that you want to be rendered to (make sure you set the layer masks to what you want them to be!)
2. Add a **FogOfWarClearFog** component to the same **GameObject** as the **FogOfWarTeam** component.
3. Set the target camera on **FogOfWarClearFog** component to be the camera that you made earlier.

### How can I have a see-through ground (ie a space scene)?

You need to have a surface for the fog to project on. But it is possible to make that surface transparent. There is a shader in the **FogOfWar** folder called **TransparentShadowCaster** which will allow you to do this. When applying it to a material, make sure that the alpha on the color is set to 0, and that the *Render Queue* is set to 2500 or less.

One downside to this method is that objects in the background may not receive shadows.

An alternative is to use the **FogOfWarClearFog** component (see above).

### How do I get Line of Sight working?

There are a few things you must do:

1. On each unit, make sure the Line of Sight Mask is set.
2. Make sure all occluding objects are of the same mask as set in step 1.
3. Make sure all occluding objects have a collider.
4. Make sure all occluding objects are level with the unit (ie they must be at the same height, or the unit will see right through it).
5. (Optional) Tweak the Field of View Penetration variable on the Fog of War component so that units can see the objects they are looking at.

6. (Optional) If your game is tile-based, you should tick the Cell Based toggle.

## What can I do to improve performance?

There are a number of things:

- If you have a team that is not being displayed, disable the FogOfWarTeam component.
- Reduce the Map Resolution, the number of units, or each Unit's vision radius. This should always be the first port of call!
- Try multithreading! There are many things that dictate performance with threads, so your best bet is to open up Unity's profiler and see what works best for you.
- Blurring of large map resolutions can also have a significant impact on performance. To turn it off, set the Blur Iterations to zero. Less iterations will reduce the performance impact. A higher resolution map will also slow down the blurring process greatly.
- If you're using Line of Sight:
  - Make sure the ground/terrain is not on the same layer as the unit's Line of Sight Mask.
  - Make the Line Of Sight occlusion object's colliders as small as possible and space out the objects as much as possible. The more objects a single unit can see, the worse performance will get.
  - Consider setting the Field of View Penetration value to zero. An alternative to the penetration is to let the occlusion objects clear the fog themselves as opposed to letting the units do it.

## How can I render things on top of the fog?

Fog of War renders the fog as an image effect. This means that anything that the camera renders will be hidden by the fog. If you want to render anything on top, you can use Unity's UI (both including pre-4.6 UI).

For anything more complicated (such as 3D objects that are not affected by the fog) you can achieve this by using an additional camera:

1. Put the objects you want to be affected on a different layer
2. Create a new camera and put it as a child of the main camera (make sure the transform is set to zero and the camera component is identical to the main camera)
3. Set the camera's depth to a higher value than the main camera.
4. Set the camera's culling mask to render only the new layer
5. The objects should now be drawn on top of the fog!

## The minimap component doesn't do everything I want it to!

Minimaps can be complicated and vary greatly from game-to-game. The FogOfWarMinimap component demonstrates how to generate a custom texture from the team data (feel free to modify the code if you can).

If this isn't enough, you should set up a separate camera that will render the whole map at once, but with a different layer mask, showing only a simplified version of the map.

## How do I save/load the fog?

You can access all of the fog values by `FoW.FogOfWarTeam.GetTeam(0).GetTotalFogValues()`. This will retrieve a byte array and the size will depend on the `mapResolution` variable. You can read/write to this at any time. See the above documentation for FogOfWarTeam.

## Why don't my textures work on units?

Textures only work in single threaded mode and on Box-shaped units. To set the texture up, make sure that you set the texture to be **Single Channel** with the **Alpha8** format in the texture import settings for that texture.

## Can I mix Unity's 3D physics with 2D art?

Yes! Set the FogOfWarTeam plane to XY and physics to Physics3D.

## Does FogOfWar work with tile-based games?

Yes! Just make sure to tick **Cell Based** on all units that use line of sight.

## Why can't I see the fog in a build?

Make sure you have included the shaders into the *Always Included Shaders* list in the Project Settings. Set the Setup section for more details.

## How can I make a minimap using the fog?

You can use the FogOfWarMinimap component if you have a small map resolution and only want very basic visuals. For anything else, you will need to do some manual work for it. You can find an example of this in the 3D sample.

Here is a list of steps to get a minimap up and running:

1. Add an extra camera to screen.
  - a. Make sure it is orthographic.
  - b. Point it downward toward the fog/map plane.
  - c. Ensure that it fits the entire map into view (unless you want it to follow the player).
  - d. Ensure that this camera will render FogOfWar (this means you need it to point to post processing, or have the FogOfWarLegacy component on it).
  - e. Set the culling mask to render only what you want to see in the minimap.
2. Create a RenderTexture in your project. Set the resolution of this texture to whatever you feel necessary.
3. Make the minimap camera render to the RenderTexture you just created.
4. Create a RawImage component in a UI Canvas and set the texture to the RenderTexture.

This may not give you all of the visual you want, but you can utilize any normal unity behaviour to customise it to your needs. For example, you may want to have your units represented as icons or simplified shapes. If this is the case, you can put different objects as child objects to the units and have them rendered on a different layer mask that only the minimap can see.

## Contact

**Support Email:** [stu\\_pidd\\_cow@hotmail.com](mailto:stu_pidd_cow@hotmail.com)

Be sure to check out my other assets here: <https://assetstore.unity.com/publishers/9066>