Fog of War

Table Of Contents

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 lightning fast to meet the needs of any RTS, MOBA or adventure game that needs a thick shroud of mist around it.

Features:

- Legacy, URP, URP 2D, and HDRP Support
- 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 hardware acceleration and 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

Requirements
- Unity 2020.1+

Limitations
- URP: **SSAO**, **Decals**, and other post processes that manipulate the depth buffer cause issues with FogOfWar. These effects will be automatically disabled. This will be fixed in a future release.
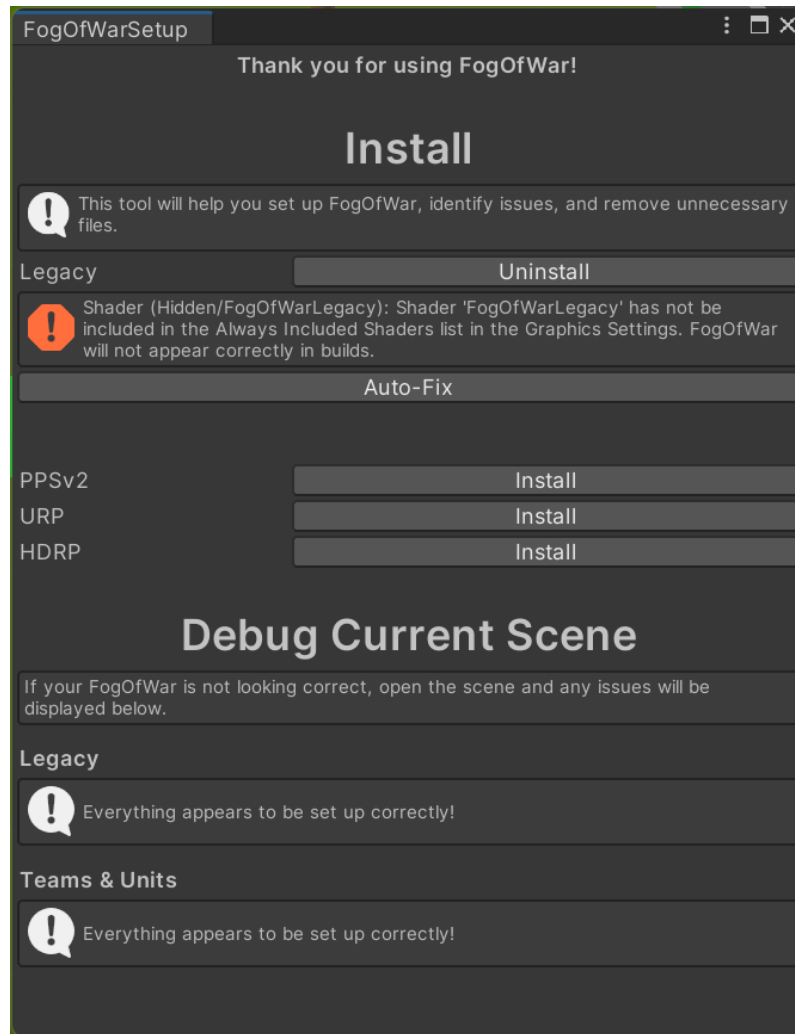
Installing

First, you must run the FogOfWar Setup. This can be found in the menu **Window -> FogOfWar Setup**.

Pick the render pipeline you want to install, and click the **Install** button.

If something went wrong, you may see an error such as this:

In this case, you can fix the error manually by following the instructions, or you can click on the **Auto-Fix** button.

Setting Up Your Scene

Place a FogOfWarTeam component anywhere in the scene. This will represent the fog that is visible for an individual team. You will need to configure your map parameters to suit your needs.

Next, add a FogOfWarUnit component to every unit. You will need to configure the shape parameters for each unit.

The final step is to link a camera to the fog. The steps required to do this depends on which render pipeline/post processing tech you are using.

- **Legacy**
    - Attach a **FogOfWarLegacy** component to the camera you want to use.
- **PPSv2**
    - Add a **FogOfWarPPSv2** effect to your **PostProcessProfile**. Make sure you have a **PostProcessLayer** in the scene.
    - The fog will be fully transparent by default, so tick **Fog Color** on the volume profile, and increase the alpha.

- **URP** & **HDRP**
  - On your **Volume** component, add the **FogOfWarURP** effect.
  - On your camera, ensure that **Post Processing** is ticked and that the **Volume Mask** is the same as the **Volume** you have placed in the same.
  - The fog will be fully transparent by default, so tick **Fog Color** on the volume profile, and increase the alpha.

Debugging Issues

The **FogOfWar Setup tool** allows you to debug issues with your render pipeline setup and scene configuration. You can use this to diagnose and repair issues.

Open the tool by going to **Window -> FogOfWar Setup**. If you have your scene open in the background (whether in edit mode or play mode), you will see the diagnosis results under **Debug Current Scene**, such as this:
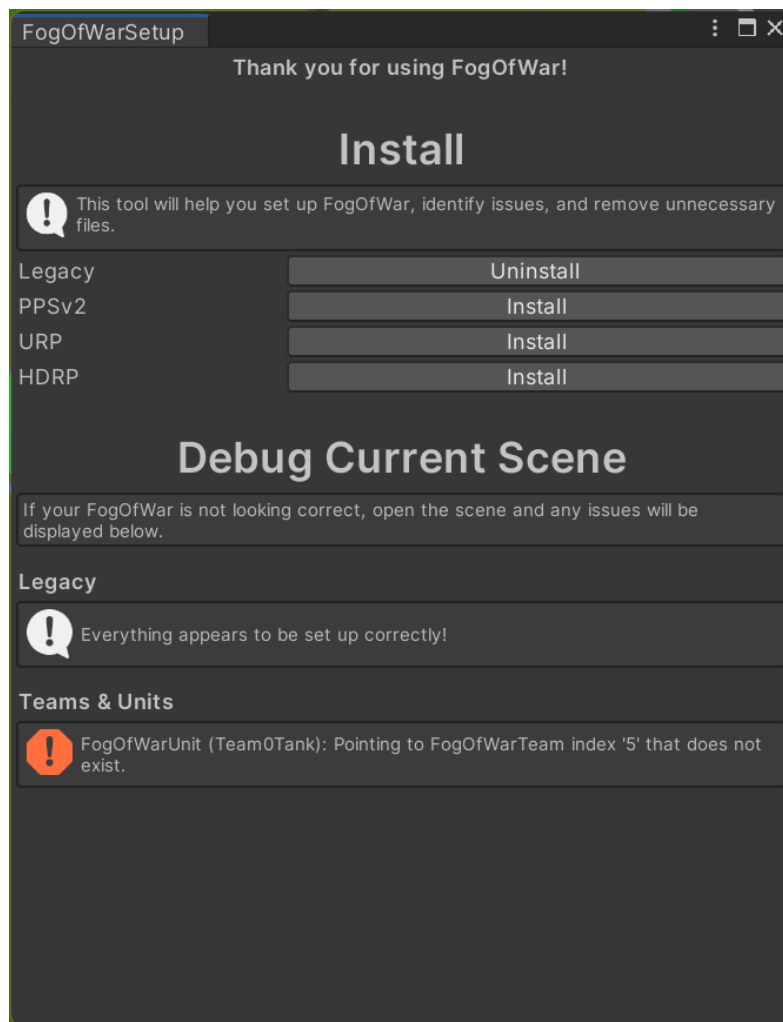
Follow the instructions to repair the issues until you see the '*Everything appears to be set up correctly!*' message.

Cleaning Up

You may want to remove all unnecessary files to keep your project clean for source control. These are all of the optional folders and files that can be removed if you don't want those features:

- *FogOfWar/Samples/*
- *FogOfWar/RenderPipelines/Legacy.unitypackage*
- *FogOfWar/RenderPipelines/PPSv2.unitypackage*
- *FogOfWar/RenderPipelines/URP.unitypackage*
- *FogOfWar/RenderPipelines/HDRP.unitypackage*
- *FogOfWar/RenderPipelines/Legacy/Samples/*
- *FogOfWar/RenderPipelines/PPSv2/Samples/*
- *FogOfWar/RenderPipelines/URP/Samples/*
- *FogOfWar/RenderPipelines/HDRP/Samples/*

Components

FogOfWarTeam
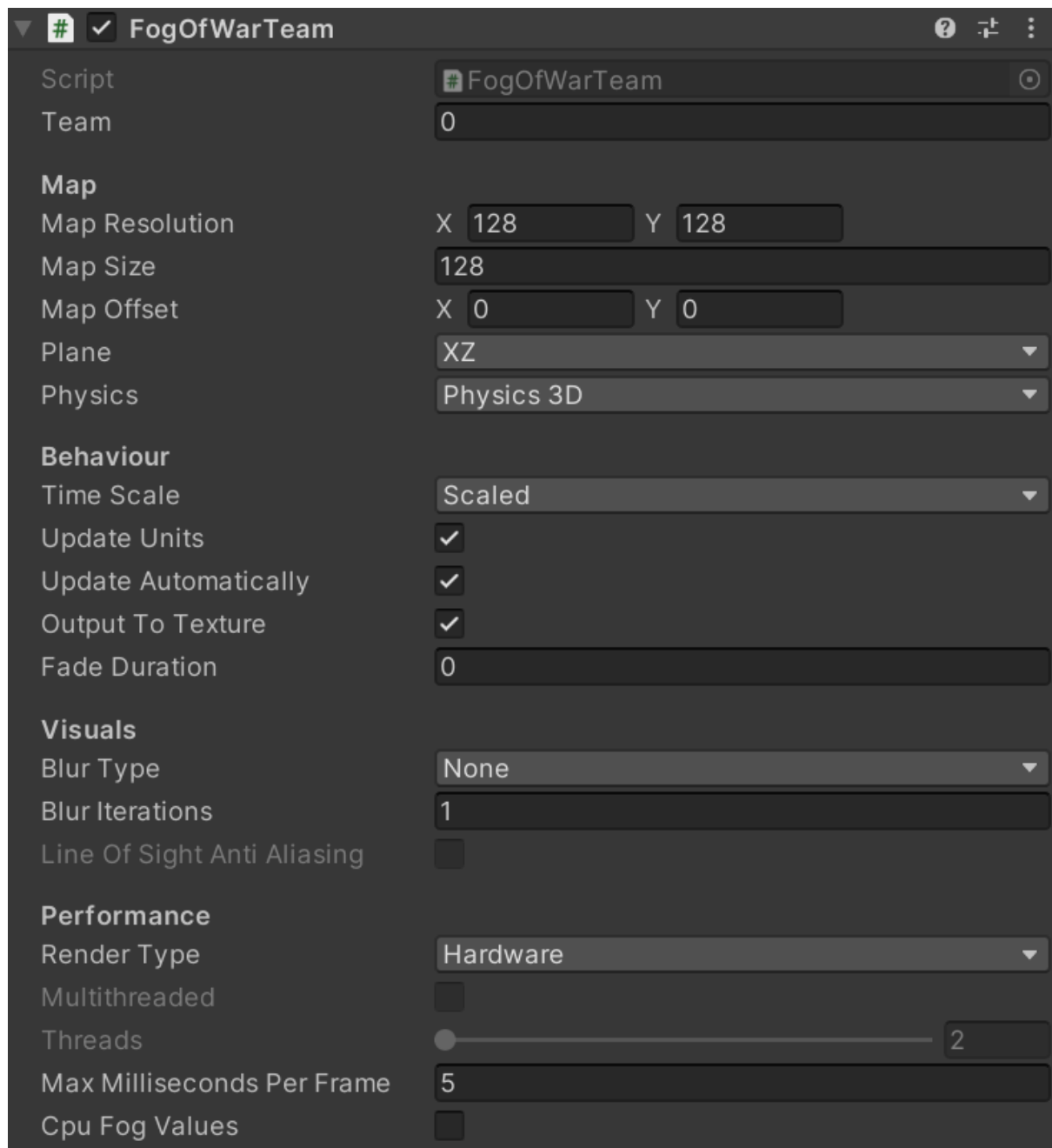
FogOfWarTeam represents everything that a single team sees. You can think of it as a fog texture made of pixels that represent how visible a point is on the map. There must be one of these for each team. At least one needs to be in the scene.

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

The fog contains a texture with 2 channels:

1. **Visible**: Represents the areas that are currently visible to a unit in this team. These areas are usually 100% cleared. This is stored in the *R* channel in the fog texture.
2. **Partially Visible**: Represents the areas that were, or currently are, visible to a unit in this team. These areas are usually partially cleared, but hide units from other teams. This is stored in the *G* channel in the fog texture.

To save/load the fog values (such as to save to a file), you can do the following:

```
FogOfWarTeam team = FogOfWarTeam.GetTeam(0);
byte[] values = new byte[team.mapPixelCount];

// Write fog values to values array
team.GetFogValues(FogOfWarValueType.Partial, values);

// Replaces the current for with values array
team.SetFogValues(FogOfWarValueType.Partial, values);
```

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. This can be done via script by calling **Reinitalize()**.

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

Team	The index of which team this fog will clear for. All units that have the same team index will have the fog cleared.
Map	
Map Resolution	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.
Map Size	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.
Map Offset	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.
Plane	Which plane the fog will be rendered to. 3D is usually XZ and 2D is usually XY
Physics	What physics will be used for raycasting and collision detection. This is Unity's built-in 2D or 3D.
Behaviour	
Time Scale	Which delta time to use for animation.
Update Units	If turned off, units will not update. This saves a lot of performance if doing a turn-based game.
Update Automatically	When set to true, the fog will be updated automatically every frame. If false, you will need to call **FoW.FogOfWarTeam.GetTeam(0).ManualUpdate()** from code to update the fog. This is useful for turn-based games.
Output To Texture	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.

Fade Duration	How long it will take for fog to be revealed (in seconds). A value of 0 will reveal the fog instantly.
Visuals	
Blur Type	The type of blur to be applied to the fog texture after all of the units have been rendered. ● None: No blurring is applied. ● Gaussian3: 3x3 Gaussian Blur. ● Gaussian5: 5x5 Gaussian Blur. ● Antialias: 3x3 Gaussian Blur only on pixels that are on an edge.
Blur Iterations	How many times the blur will be applied. More iterations will result in a smoother appearance.
Line Of Sight Anti Aliasing	For software only. If enabled, can result in slightly smoother line of sight edges. This may have an impact on performance.
Performance	
Render Type	Which technique will be used to render the fog. ● **Hardware:** All fog calculations are done on the GPU. This is faster for large numbers of units, and/or large map resolutions. ● **Software**: All fog calculations are done on the CPU. This can be faster for very small numbers of units, and small map resolutions. It's recommended that you try **Hardware** first, as it will probably perform better than **Software** in most cases. Also, **Software** requires all **FogOfWarUnit**s that use textures, have *Read/Write* enabled in their import settings.
Multithreading	For software only. If enabled, the rendering will be done off of the main unity thread. This can result in better performance when there are large unit counts.
Threads	For multithreaded software only. This is the number of threads used to render **FogOfWarUnit**s (including the main thread).
Max Milliseconds Per Frame	If the fog spends longer than this number of milliseconds updating the fog texture, it will delay more rendering until the next frame. Set this to a high number to ensure all of the fog is rendered in a single frame.
CPU Fog Values	If true, **FogOfWarTeam.fogValues** will be retrieved. In software, this is free. In hardware, this involves pulling the texture data from the GPU, which is very slow. This can improve the reliability when calling **GetFogValues()**. It is strongly recommended to leave this off.

Properties

|	
int mapPixelCount	This property is how large an array must be when calling **GetFogValues()** and **SetFogValues()**. This will always be **mapResolution.x * mapResolution.y.**
Texture fogTexture	The final fog texture in *RGB24* format. The channels are: ● R: Visible ● G: Partially Visible ● B: Unused

Color32[] fogValues	The colors that can be found inside **fogTexture**. This will only be available in **Software** mode, or in **Hardware** when **cpuFogValues** is ticked. The channels are: ● R: Visible ● G: Partially Visible ● B: Unused ● A: Unused
UnityEvent onFogTextureChanged	Callback for whenever a new **fogTexture** has been created. This is usually when the map has been reinitialized.
UnityEvent onRenderFogTexture	Callback for whenever the **fogTexture** has been updated. This should be called almost every frame.

Functions

static FogOfWarTeam GetTeam(int teamID)	Returns the FogOfWarTeam for a particular team. If no team exists, null will be returned.
void Reinitialize()	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 at runtime by clicking the *Reinitalize* button on the FogOfWarTeam component inspector.
byte[] GetFogValues(FogOfWarValueType type)	Same as below, but will allocate the **byte[]** for you.
void GetFogValues(FogOfWarValueType type, byte[] values)	Copies the current fog values into the specified byte array. The values are the state of the fog as it was in the last rendered fog frame. Returned values are 0 for completely unfogged, and 255 for completely fogged. The size of the array should be **FogOfWarTeam.mapPixelCount**. It is recommended that you avoid calling this frequently when using **Hardware** mode with **cpuFogValues** disabled because it is bad for performance.
void SetFogValues(FogOfWarValueType type, byte[] values)	Copies the specified array into the current fog values. The values should be 0 for completely unfogged, and 255 for completely fogged. The size of the array should be **FogOfWarTeam.mapPixelCount**. It is recommended that you avoid calling this frequently when using **Hardware** mode because it is bad for performance.
Vector2i WorldPositionToFogPosition(Vector3 position)	Converts a world position to a fog pixel position. Values will be between 0 and mapResolution.

byte GetFogValue(FogOfWarValueType type, Vector3 position)	Returns the fog amount at a particular world position. 0 is fully unfogged and 255 if fully fogged.
void SetFog(FogOfWarValueType type, Bounds bounds, byte value)	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.
float ExploredArea(int skip = 1)	Returns how much of the map has been explored/unfogged, where 0 is 0% and 1 is 100%. Increase the skip value to improve performance but sacrifice accuracy.
void SetAll(FogOfWarValueType type, byte value = 255)	Sets the fog value for the entire map. Set to 0 for completely unfogged, to 255 for completely fogged.
float VisibilityOfArea(FogOfWarValueType type, Bounds worldbounds)	Checks the average visibility of an area. 0 is fully unfogged and 1 if fully fogged.
float ExploredArea(int skip = 1)	Returns how much of the map has been explored/unfogged, where 0 is 0% and 1 is 100%. Increase the skip value to improve performance but sacrifice accuracy.
void GetVisibleUnits(int teamIndex, byte maxGog, List<FogOfWarUnit> units)	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 255 is fully fogged. The list of units will not be cleared when called and is assumed to be empty.
void ManualUpdate(float timeSinceLastUpdate)	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.
void ApplyToMaterial(Material material, float outsidefogstrength = 1)	Applies all properties to the specified material that are required to detect fog for this team.

FogOfWar Renderer
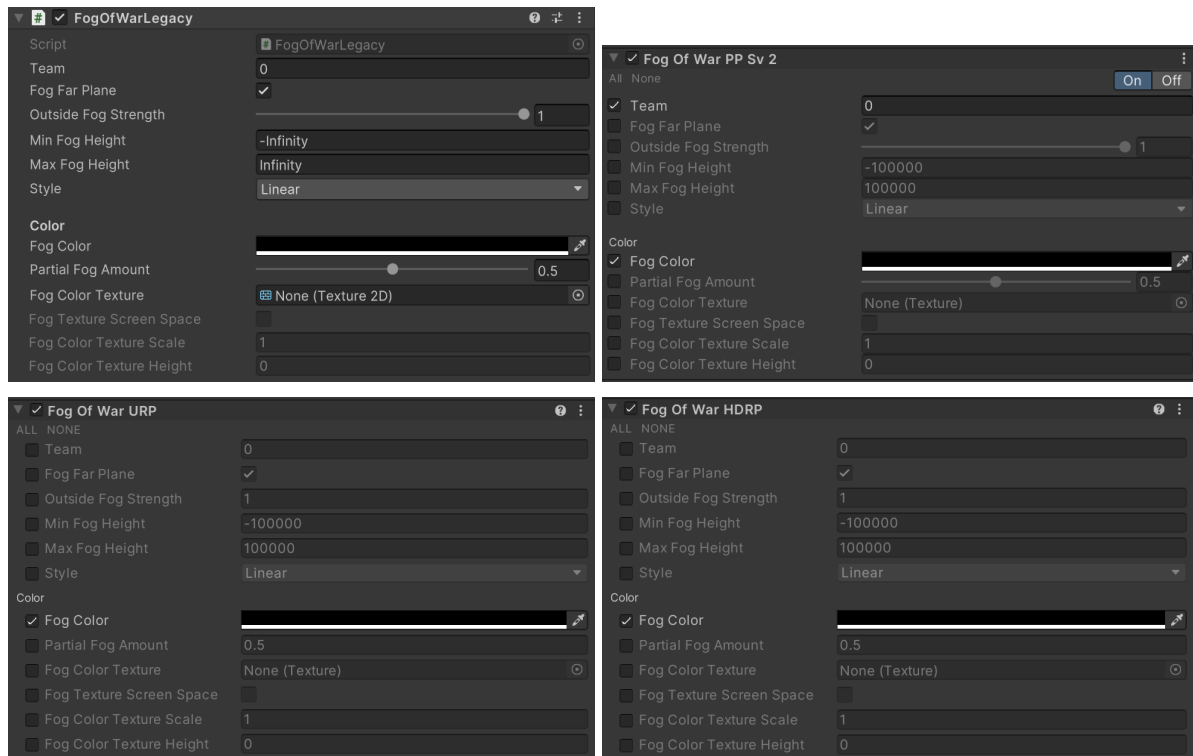
Depending on which render pipeline and post processing technology you are using, you will need to set this up differently.

	FogOfWarLegacy	**PPSv2**	**Volume System**
Legacy	✓	✓	✗
URP	✗	✓	✓
HDRP	✗	✗	✓

See the *Setup* section above for how to set this up properly for your use case.

Map	
Team	The index of which team will be displayed. This will be the same index as is set on the *FogOfWarTeam* component.
Fog Far Plane	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.
Outside Fog Strength	Specifies how strong the fog will be outside of the fog map area. 1 = fully fogged, 0 = no fog.
Min Fog Height	The minimum height that fog can appear.
Max Fog Height	The maximum height that fog can appear.
Style	The visual style of the fog. The styles are: ● Point: A point filter is applied to the texture. This will give it a pixelated appearance.

	- Linear: A bilinear filter is applied to the texture. This will give it smooth transitions. - SDF: An SDF filter will be applied, with a cutoff at 0.5. This will have hard edges, but with smooth curves.
Color	
Fog Color	The color of the fog. When using the *FogOfWarTextureSource* component, the alpha value will determine how transparent the fogged area will be (you usually want the alpha to be zero).
Partial Fog Amount	How visible the partial fog areas should be.
Fog Color Texture	A texture that can be applied over the top of the fog.
Fog Texture Screen Space	If true, the displayed texture will be in screen space. *fogColorTextureScale* and *fogColorTextureHeight* will not be used.
Fog Color Texture Scale	The size that the fog color texture appears to be in world space.
Fog Color Texture Height	The illusion of height that the fog color texture is at.

FogOfWarTextureSource

| ▼ # ✓ Fog Of War Texture Source (Script)		❷ ⊋ ⋮
Script	# FogOfWarTextureSource	⊙
Target Camera	◼◖ CloudCamera (Camera)	⊙
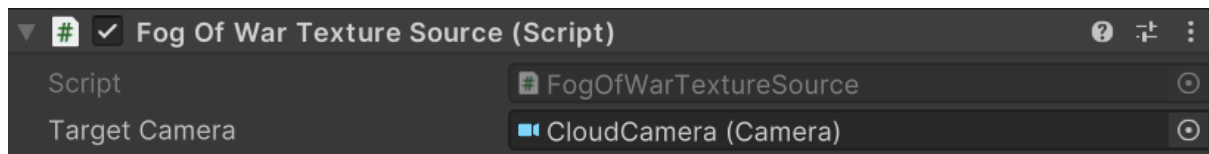
A fog texture source enables you to replace the fog with the rendering of another camera. This can be used to create weird effects where the fog won't be just a single color (such as a starry-space backdrop or animated clouds).

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 **FogOfWarTextureSource** component to the same *GameObject* as your **Camera** component.
3. Set the target camera on the **FogOfWarTextureSource** component to be the camera that you made earlier.

Target Camera	The camera that will be used to render over the fog.

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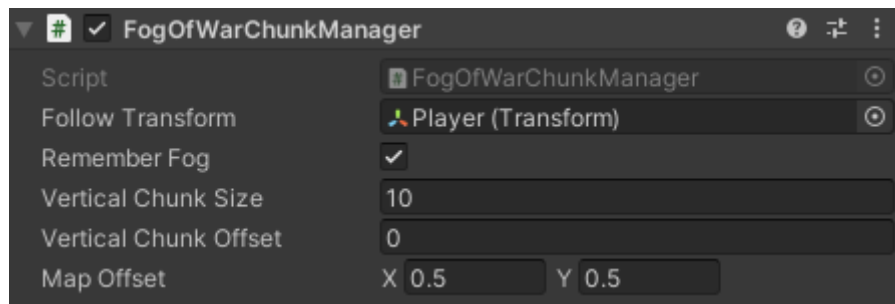
FogOfWarChunkManager will automatically move the map to center on the player. This allows for an infinite map that can run infinitely in all directions without a huge performance impact. The chunk manager also allows for vertical chunks. This is useful for games that cover multiple floors.

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 on the **FogOfWarTeam** component 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 on the **FogOfWarTeam** component has no effect when the chunk manager is enabled

Follow Transform	The game object that the fog should follow. This is usually the player.
Remember Fog	If false, the chunk manager will completely discard all fog information once a chunk is left.
Vertical Chunk Size	The size of a floor before the next vertical chunk is loaded. This should be the distance from one floor to another.
Vertical Chunk Offset	The offset from zero where the first vertical chunk is.
Map Offset	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

byte GetFogValue(FogOfWarValueType type, Vector3 pos)	Same as FogOfWarTeam.GetFogValue(), but can pull value from unloaded chunk data.
void Clear()	Removes all cached chunk data.
byte[] Save()	Saves all chunk data into a single byte[] that can be loaded at a later time.

| bool Load(byte[] data) | Load previously saved byte[] of the chunk data. |

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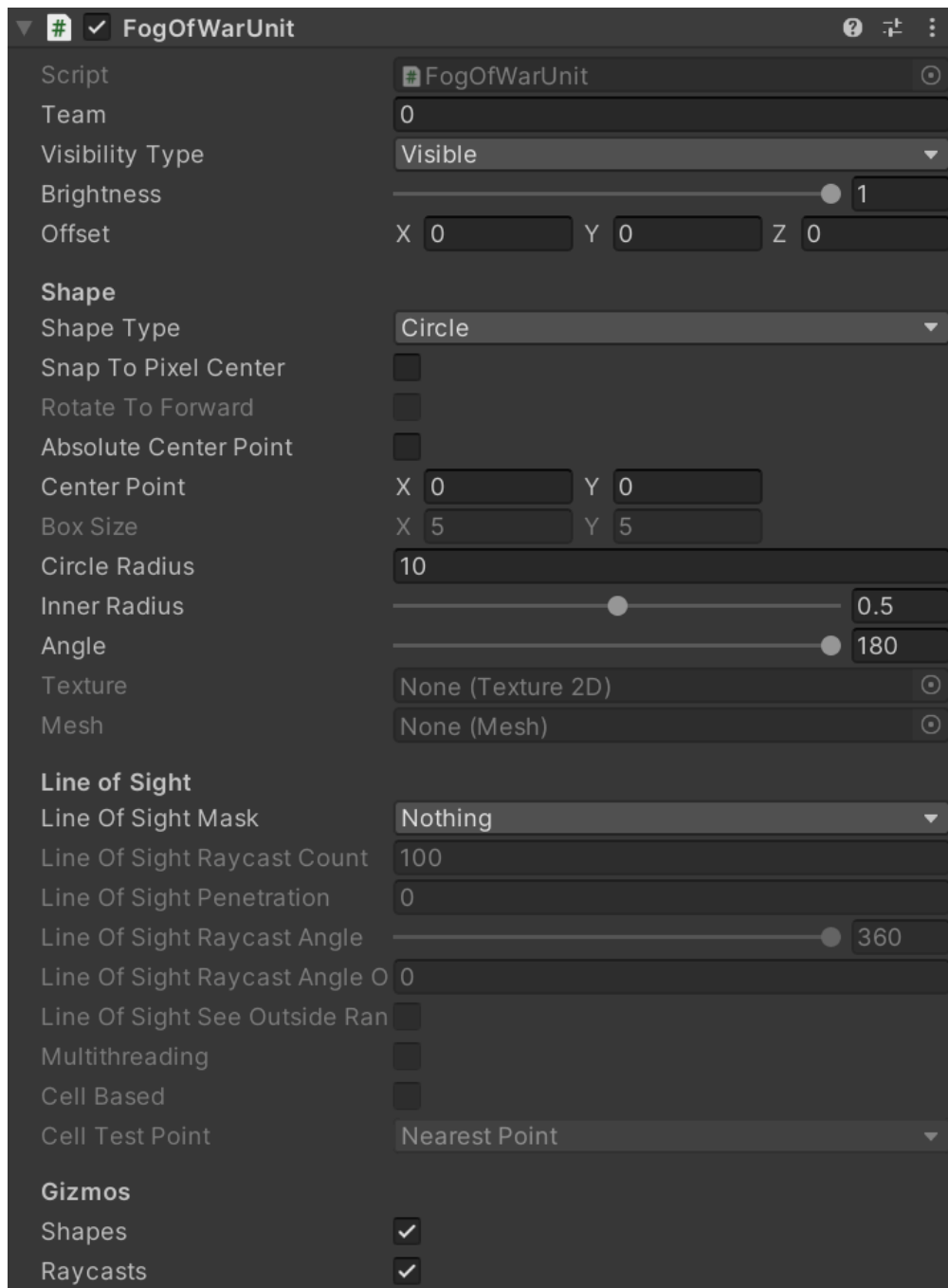
FogOfWarUnit should be placed on every **GameObject** that will reveal areas in the fog.

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.

Team	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.
Visibility Type	The type of visibility for this unit. *Visible* will permanently clear the fog so the unit is visible. *Partial* will clear the fog but not make the unit visible. *VisibleOnly* will clear the fog so the unit is visible, but only temporarily.
Brightness	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.
Offset	The offset of the unit's center point in world coordinates. The center point is used to cast rays and move the shape around.
Shape	
Shape Type	Can be Circle, Box or Mesh.
Snap To Pixel Center	If true, the unit's position will be snapped to the center of the fog pixel.
Rotate To Forward	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.
Absolute Center Point	The center point of the unit's shape.
Box Size	The size of the box. Only applicable for Box shapes.
Circle Radius	How far the unit can see. This is used to define the circle shape.
Inner Radius	How smooth the edge of the circle is. Only applicable to Circle shapes.
Angle	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.
Texture	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!
Mesh	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.
Line of Sight	
Line Of Sight Mask	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.
Line Of Sight Raycast Count	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.
Line Of Sight Penetration	How far through objects that the unit's vision will penetrate through.
Line Of Sight Raycast Angle	The maximum angle that raycasts are used.
Line Of Sight Raycast Angle Offset	The offset of the raycast angle start point. Use this to rotate the raycast angles.

Line Of Sight See Outside Range	If true, anything beyond the range specified in LineOfSightRaycastAngle will be visible.
Multithreading	Enables multithreaded raycasting using the job system. This option is only available in Unity 2018.1 and higher.
Cell Based	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 collisions properly. You would generally want to leave **Line Of Sight Penetration** to zero if this is enabled.
Cell Test Point	"The point in a fog cell that will be raycasted to. This will only be used if cellBased is set to true.
Gizmos	
Shapes	If enabled, the Scene View will show red gizmos for the shape of the unit.
Raycasts	If enabled, the Scene View will show red gizmos for what raycasts will be made to determine line of sight visibility.

Useful Functions

byte GetFogValue(Vector3 position)	Returns the fog value as visible to only this unit. 0 is fully unfogged and 255 if fully fogged.

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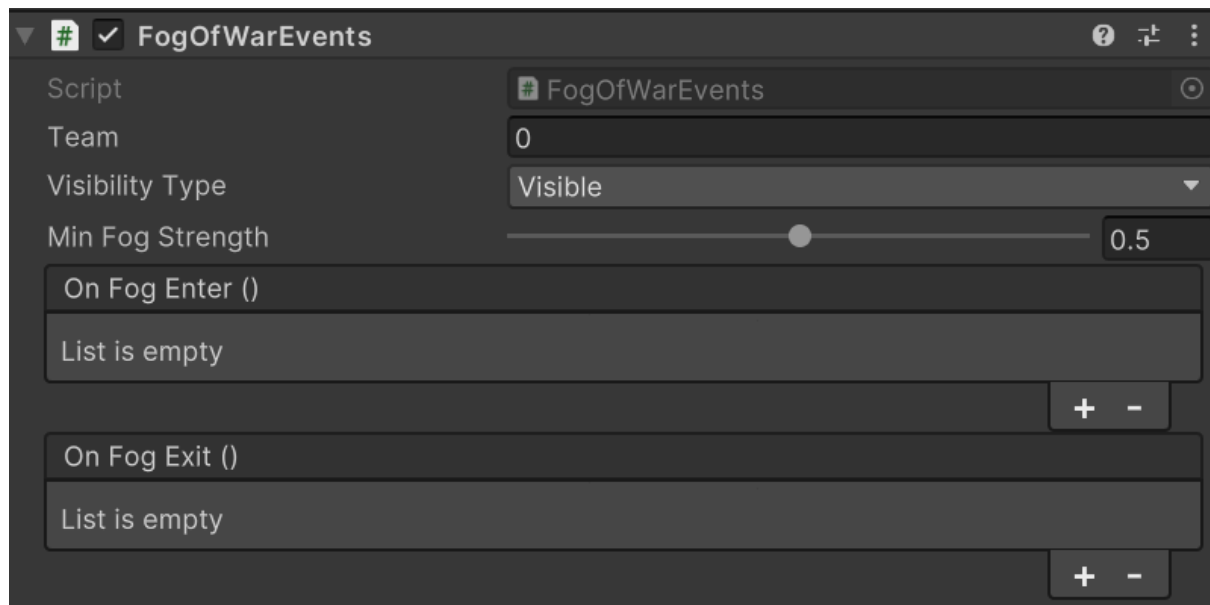
FogOfWarEvents allows you to set events of what should happen when this **GameObject** enters or exits the fog. The position is based off the **GameObject**'s **Transform**.

Team	Which FogOfWar team will be tracked.
Visibility Type	The channel of the fog texture on the **FogOfWarTeam** it read.
Min Fog Strength	The minimum fog strength until the OnFogEnter is registered.
On Fog Enter	An event that is called when the object enters fog that is a greater strength than MinFogStrength.
On FogExit	An event that is called when the object exits fog that is a greater strength than MinFogStrength.

FogOfWarHideInFog
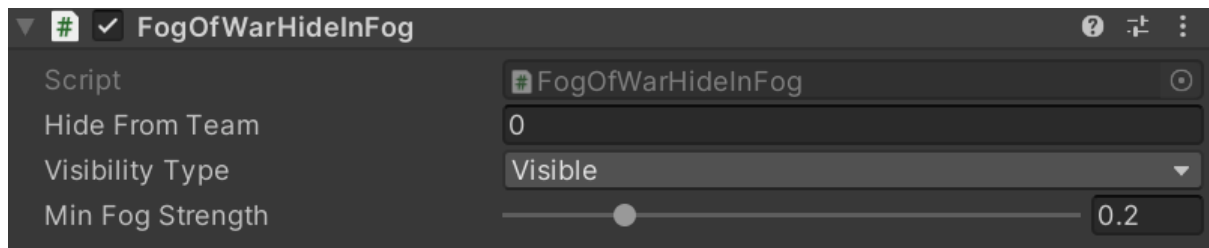
HideInFog will hide the object when it enters fog. The object will be made invisible by enabling and disabling the Renderer, UI Canvas or UI Graphic attached to this GameObject. You will need one of these components for each team that you want to hide from.

Hide From Team	Which FogOfWarTeam will be tracked.
Visibility Type	The channel of the fog texture on the **FogOfWarTeam** it read. You will usually want this set to Visible.
Min Fog Strength	The minimum fog strength until the object is made invisible.

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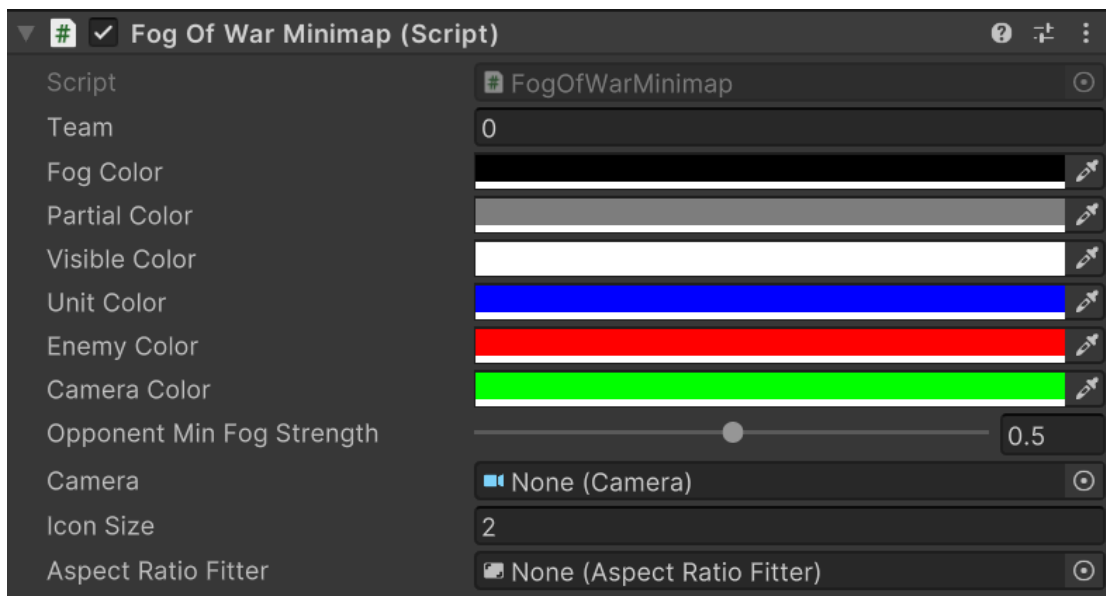
WARNING: It is not recommended to use this due to performance issues and may be removed in future releases. See the FAQ for a better solution.

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.

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

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.onFogTextureChanged** is called.

Migrating

From 1.26 to 1.27+

Make sure to delete your old FogOfWar folder before installing *1.25*.

Some changes that may affect you if migrating from a previous version:

- Rename *team* in **FogOfWarHideInFog** to *hideFromTeam*.
- Rename **FogOfWarClearFog** to **FogOfWarTextureSource**.

From 1.24 to 1.25+

Make sure to delete your old FogOfWar folder before installing *1.25*.

Lots of changes were made in this release. Some changes that may affect you if migrating from a previous version:

- Installation of the render pipeline/post processing tech will need to be done through the **Window -> FogOfWar Setup** menu item.
- It is recommended to use *Hardware* mode on the **FogOfWarTeam** component. *Hardware* mode now has feature parity with *Software*, and is generally significantly better with performance.
- The **HideInFog** component has been renamed to **FogOfWarHideInFog**.
- Renamed **FogOfWarUnit**'s offset to *centerPoint*, and *absoluteOffset* to *absoluteCenterPoint*
- Renamed **FogOfWarTeam**'s *filterMode* to *lineOfSightAntiAliasing*
- Replaced rendering component's *pointFiltering* with a **FogOfWarStyle** enum
- **FogOfWarTeam** has replaced various fog value properties and functions to have separate values for visible and partial fog textures.
- **FogOfWarTeam.finalTexture** has been removed as it is up to the renderer to decide how they are merged.
- Removed **FogOfWarTeam.blurAmount** because it is now being automatically calculated.
- **FogOfWarTeam**'s *fogInDuration* and *fogOutDuration* have been combined into *fogDuration*.
- **FogOfWarEvents** and **FogOfWarHideInFog** now have a *visibilityType* enum that may affect previous values.
- **FogOfWarMinimap** has more values for setting colors.

From 1.16 to 1.17+

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 big 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.

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 **FogOfWarTextureSource** 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:

- Use *Hardware* mode on the **FogOfWarTeam** component.
- Turn off *CPU Fog Values* on the **FogOfWarTeam** component. This will prevent you from calling the following methods in *Hardware* mode:
    - **FogOfWarTeam.ExploredArea()**
    - **FogOfWarTeam.VisibilityOfArea()**
    - **FogOfWarTeam.GetFogValue()** – when retrieving *FogOfWarValueType.Visible*
    - **FogOfWarTeam.GetFogValues()**
    - **FogOfWarTeam.VisibilityOfArea()**
    - **FogOfWarTeam.VisibilityOfArea()**

- When in *Hardware* mode, avoid called to the following methods, as they will result in expensive GPU calls:
  - **FogOfWarTeam.GetFogValues()**
  - **FogOfWarTeam.SetFogValues()**
- If you have a team that is not being displayed, disable the **FogOfWarTeam** component, or untick *Update Units* and *Output To Texture* of the **FogOfWarTeam** component.
- Reduce the *Map Resolution* on the **FogOfWarTeam** component.
- Reduce the number of **FogOfWarUnit**s in the scene.
- Reduce each **FogOfWarUnit**'s vision radius.
- Disable *Fade Duration* on the **FogOfWarTeam** component.
- If using *Software* mode on the **FogOfWarTeam** component, try enabling *multithreading* and increase the number of threads.
- Reduce or disable blur passes on the **FogOfWarTeam** component.
- 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.
  - Reduce the number of rays or cells.

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 for a team, save it to a byte[], and load it back in, by doing the following:

```
FogOfWarTeam team = FogOfWarTeam.GetTeam(0);
byte[] values = new byte[team.mapPixelCount];
```

```
// Write fog values to values array
team.GetFogValues(FogOfWarValueType.Partial, values);

// Replaces the current for with values array
team.SetFogValues(FogOfWarValueType.Partial, values);
```

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. You can check for these issues by running the setup tool found in the menu **Window -> FogOfWar Setup**.

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 scene.
   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 visuals you want, but you can utilize any normal unity behaviour to customize 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.

Release Notes

v1.28

- Fix for blur shader not outputting correct fog values
- Fixed SDF using inverse of partial fog
- A few documentation improvements

v1.27

- Increased Unity version requirement to 2020.1+
- Unity 6 compatibility
- Fixed hardware mode not applying when calling FogOfWarTeam.SetFogValues()
- Fixed flicker on chunk manager chunk change in hardware mode
- Made Circle FogOfWarUnit shape auto-disable rotateToForward
- Fixed some issues with CustomFogShader.shader
- Optimized FogOfWarUnit.GetVisibility() and FogOfWarTeam.GetFogValue()
- Renamed FogOfWarHideInFog.team to hideFromTeam
- Added timeScale to FogOfWarTeam
- Fixed FogOfWarChunkManager.rememberFog unnecessarily reading from GPU when set to false
- Fixed FogOfWarChunkManager.mapOffset not being taken into account for some calculations
- Made FogOfWarTeam.SetFog() round outward (meaning you can now specify a size of zero)
- Fixed various issues with the sample scenes
- Improved calculations when converting between fog and world space (this was most notable when switching between chunks at low resolutions)
- Added FogOfWarUnit.visibilityType to allow rendering to only partial, or visible without writing to partial
- Optimized FogOfWarTeam.VisibilityOfArea()
- Added GPU texture reading capabilities for GetFogValue(), VisibilityOfArea() and ExploredArea() methods in FogOfWarTeam
- Made partialFogAmount and outsideFogStrength have sliders on FogOfWarURP and FogOfWarHDRP
- Slight optimization when compiling textures in FogOfWarTeam
- Renamed FogOfWarClearFog to FogOfWarTextureSource
- Fixed issue in Software mode rendering Cell Based where pixel at local coordinate (0, 0) would be incorrect
- Added compatibility with URP render graph
- Fixed "'R8_SRGB' is not supported" warning in HDRP hardware mode

v1.26

- Added FogOfWarUnit.GetFogValue() method for querying visibility for an individual unit
- Fixed Fade Duration not working in Hardware mode
- Fixed TargetParameterCountException being thrown in 2019.4
- Fixed a few compile errors in 2019.4
- Removed flashing when switching chunk with the chunk manager

- Added movement in 2D sample scene when using touch controls
- Fixed null reference when calling FogOfWarTeam.GetFogValues()
- Fixed FogOfWarHardwareVisibleToPartial.shader not being included in Always Included Shader list

v1.25

- Fixed android URP displaying upside down
- Fixed sensitivity of demo scenes on touch devices
- Fixed FogOfWarURP shader not working with older hardware
- Fixed OverflowException when setting FogOfWarUnit.lineOfSightRaycastCount <= 0
- Added HideInFog script to 3D sample opponent units
- Fixed misleading documentation for FogOfWarTeam.ManualUpdate()
- Fixed demo scene now loading legacy scenes
- Fixed FogOfWarTeam.GetVisibleUnits() iterating through the wrong unit list
- Fixed compile errors for unity versions earlier than Unity 2018
- Fixed warning from Unity 2021.2
- Renamed FogOfWarUnit.lineOfSightRaycastOffset to lineOfSightRaycastAngleOffset
- Simplified demo sample scenes so they work with all render pipelines
- Put URP and HDRP into unitypackages to avoid compile errors after installing
- Fixed FogOfWarEvents OnFogEnter and OnFogExit doing the opposite
- Added height range for fog on all axes
- Fixed HDR not applying when FogOfWar is enabled
- Fixed major thread leak when using multithreading
- Automatically set UniversalRendererData.intermediateTextureMode to Always for Unity 2021.2
- Fixed URP cameraColorTarget error in Unity 2020.2
- Removed depth buffer requirement for hardware mode
- Removed SV_VertexID shader dependency
- Added FogOfWarChunkManager.GetFogValue()
- Fixed some incorrect chunk position calculations in FogOfWarChunkManager
- Fixed stack overflow with FogOfWarTeam.GetCurrentFogValues()
- Fixed RaycastCommand warning in 2022.2
- Put FogOfWarTeamEditor in FoW namespace
- Added gizmo toggles for FogOfWarUnit and made them work in edit mode
- Renamed FogOfWarUnit's offset to centerPoint, and absoluteOffset to absoluteCenterPoint
- Renamed FogOfWarTeam's filterMode to lineOfSightAntiAliasing
- Replaced rendering component's pointFiltering with a FogOfWarStyle enum
- Renamed FogOfWarTeam Multithreading header to Performance
- FogOfWarTeam now separates visible and partial fog textures (final texture has been removed as it is up to the renderer to decide how they are merged)
- Removed FogOfWarTeam.blurAmount because it is now being automatically calculated
- Replaced old fog fading animation with a new performant system (combines FogOfWarTeam's fogInDuration and fogOutDuration into fogDuration)
- FogOfWarUnit's shape texture respects the texture filtering
- Added visibilityType to FogOfWarEvents
- Added visibilityType to HideInFog

- Added separate colors for fog, partial and visible on FogOfWarMinimap
- Merged FogOfWarTeam's Get*FogValues()/Set*FogValues()/Get*FogValue() into GetFogValues()/SetFogValues()/GetFogValue() which lets you specify the source
- Hardware mode now has feature parity with Software mode, and is no longer considered experimental
- Fixed some inconsistencies between FogOfWarUnit's absoluteCenterPoint and rotateToForward
- Fixed some inconsistencies between FogOfWarUnit's boxSize and circleRadius when dealing with mesh shapes
- Added property drawers to enable/disable relevant fields
- Added support for URP 2D renderer
- Added warning message when using unsupported FogOfWarUnit.texture with multithreading
- Added vertex color support to FogOfWarUnit mesh shapes in hardware mode only
- Added warning and auto fix when SSAO is enabled with URP
- Added FogOfWarSetup window to automate installation and debugging issues in the scene
- Big improvements to general error checking
- Renamed HideInFog component to FogOfWarHideInFog
- Greatly reduced GPU -> CPU readback, and added FogOfWarTeam.cpuFogValues
- Added FogOfWarUnit.cellTestPoint to specify how cell-based cell visibility is tested
- Made FogOfWarUnit cell-based line of sight support physics 3D and non-XY planes
- Added output texture display in FogOfWarTeam inspector at runtime

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 anti-flickering 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

- Initial release with basic demo

Tell Me About Your Game!

As an asset developer, I am always curious as to where FogOfWar has ended up. If you have a game that has been released, or is close to release, feel free to send me an email or post on the FogOfWar forum thread.

I would like to put together a montage video of the many games that use FogOfWar, and can help get the name of your game out there!

Refund Policy

Refunds will only be granted if all of the following conditions are met:

- Unity Terms of Service and EULA requirements are met
- The customer has put in reasonable effort to rectify the issue
- If the reason is due to a bug:
    - The bug is reproducible by the asset developer
    - The asset developer cannot repair the bug in under 7 days

Refunds will be processed through Unity's customer support and will not be paid directly by the asset developer.

When you want to process a refund, follow this link:
https://support.unity.com/hc/en-us/articles/205060439-How-do-I-request-a-refund-on-an-asset

Contact

Support Email: stu_pidd_cow@hotmail.com

When sending an email, be sure to supply the following:

- FogOfWar package version
- Unity version
- Render pipeline and version

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

I also have some cool stuff here: https://www.brendankeesing.com/

If you have found FogOfWar useful, please leave a review on the asset store, as it helps get the word out!