

WORLD POLITICAL MAP GLOBE EDITION

+ *Map Editor!*



Index

Introduction	3
QuickStart and Demo Scene	3
Support & Contact info	3
How to use the asset in your project	4
Custom Inspector Properties	4
Using the Bake Texture command	7
Using the Scenic styles	8
Mount Points	9
Reducing application build size	10
Integrating with Online Tile Map Systems	11
Virtual Reality (VR)	13
<i>Basic VR Usage</i>	<i>13</i>
<i>Custom Gazes & Pointers</i>	<i>13</i>
Programming Guide (API)	14
<i>Structure of the API</i>	<i>14</i>
<i>Public Properties</i>	<i>14</i>
Country object fields	14
Countries API	15
Province object fields	16
Provinces API	17
City object fields	18
Cities API	18
Mount Points related	19
Earth related	19
Navigation	20
User interaction	20
Labels related	21
Tile system related	22
<i>Other Methods</i>	<i>23</i>
<i>Events</i>	<i>23</i>
Additional Components	25
<i>World Map Calculator</i>	<i>25</i>
Using the distance calculator from code	27
<i>World Map Ticker Component</i>	<i>28</i>
<i>World Map Decorator</i>	<i>31</i>
<i>World Map Editor Component</i>	<i>33</i>
Main toolbar	34
Reshaping options	35
Create options	35
<i>Editing Tips</i>	<i>36</i>
<i>World Flags and Weather Symbols</i>	<i>37</i>

Introduction

Thank you for purchasing!

World Political Map – Globe Edition is a commercial asset for Unity 5.1.1 and above that allows to:

- ✓ Visualize the frontiers of 241 countries, +4000 provinces and states and the location of +7000 most important cities in the world without connecting to the Internet (integrated cartography).
- ✓ Ability to integrate with online map tile systems.
- ✓ Colorize, texture and also highlight the regions of countries and provinces/states as mouse hovers them.
- ✓ Automatically draw country labels.
- ✓ Add markers and line animations to the globe.
- ✓ Define custom mount points and customize its location and tags with the editor.
- ✓ Fly to a chosen country, city, province or location. It will make the globe rotate until the destination is reached.
- ✓ Imaginary lines: draw custom latitude, longitude and cursor lines.
- ✓ Ease choose between different catalogs included based on quality/size for frontiers and cities. Filter number of cities by population and/or size.
- ✓ Lots of customization options: colors, labels, frontiers, provinces, cities, Earth (7 styles including scenic with clouds, shadows, day/light and nice glow effects compatible with mobile and another even more advanced including physically based atmosphere scattering)...
- ✓ Works on Android and iOS (all styles except for the atmosphere scattering style).
- ✓ Comprehensive API and extra components: Calculator, Tickers, Decorator and even a Map Editor!
- ✓ Dedicated and responsive support forum.

You can use this asset to represent or allow the user choose a location in your game/application, in mission briefings, reports, statistical or educational software, etc.

QuickStart and Demo Scene

1. Import the asset into your project or create an empty project.
2. Open any of the demo scenes included in Demo folder.
3. Run and experiment with the demonstrations.

The Demo scenes contain a WorldMapGlobe instance (the prefab) and a Demo gameobject which has a Demo script attached which you can browse to understand how to use some of the properties of the asset from code (C#).

Support & Contact info

We hope you find the asset easy and fun to use.
Feel free to contact us for any enquiry.

Kronnect Games

Email: contact@kronnect.me

Kronnect Support Forum: <http://www.kronnect.me>

Unity Forum Thread: <http://forum.unity3d.com/threads/released-world-political-map-globe-edition.343245/>

How to use the asset in your project

Drag the prefab “WorldMapGlobe” from “Resources/Prefabs” folder to your scene.
Select the GameObject created to show custom properties:



Custom Inspector Properties

Show Earth: shows/hide the Earth. You can for example hide the Earth and show only frontiers giving a look of futuristic UI.

You may want to not hide the Earth, but instead use the CutOut style, which will hide the Earth, but will prevent the geographic elements and lines to be seen when they're on the back of the sphere.

Earth Style: changes current texture applied on the Sphere of the prefab.

Show Latitude/Longitude Lines: will activate/deactivate the layers of the grid. The stepping options allow you to specify the separation in degrees between lines (for longitude is the number of lines).

Grid Color: modifies the color of the material of the grid (latitude and longitude lines).

Grid Mode: overlay or masked, which will draw the grid only over oceans (note that masking the grid will incur in a performance hit – test it).

Show Cities: activate/deactivate the layer of cities.

Min Population and other city filters: allows you to filter the cities to be drawn by either minimum population (metropolitan population) or its class (can force to always show region or country capitals).

- **Min Population (K):** allows to filter cities from current catalog based on population (K = in thousands). When you move the slider to the right/left you will see the number of cities drawn below. Setting this to 0 (zero) will make all cities in the catalog visible.
- **Combine Meshes:** combines all cities meshes into a single mesh improving performance. Note that once combined the dynamic city scaler no longer work, the city icons will retain its current scale.
- **Frontiers Detail:** specify the frontiers data bank in use. Low detail is the default and it's suitable for most cases (it contains definitions for frontiers at 110.000.000:1 scale). If you want to allow zoom to

small regions, you may want to change to High setting (30:000:000:1 scale). Note that choosing high detail can impact performance on low-end devices.

- **Inland Frontiers:** show/hide continent borders. This option computes which frontiers segments are shared by two or more countries showing only unique segments (take this into account if you modify frontiers).
- **Show Countries:** show/hide all country frontiers. It applies to all countries, however you can colorize individual countries using the API. If you enable this option, make sure “Inland Frontiers” is disabled to avoid redrawing continent borders.
- **Frontiers Color:** will change the color of the material used for all frontiers.
- **Country Highlight Enabled:** when activated, the countries will be highlighted when mouse hovers them. Current active country can be determined using *countryHighlighted* property.
- **Country Highlight Color:** fill color for the highlighted country. Color of the country will revert back to the colorized color if used.
- **Draw Outline and Outline Color:** draws a colored border around the colorized or highlighted country.
- **Show Country Names:** when enabled, country labels will be drawn and blended with the Earth map. This feature uses RenderTexture and has the following options:
 - **Render Method:** you can choose either render the labels on a texture that wraps around the globe (Blended) or individual text mesh objects in world space (World Space).
 - **Texture Resolution:** controls the size of the RenderTexture used when Render Method is set to Blended, thus affecting to the resolution of the labels shown in the map. Low quality uses a texture of 2048x1024, Medium 4096x2048 and High 8192x4096.
 - **Relative Size:** controls the amount of “fitness” for the labels. A high value will make labels grow to fill the country area.
 - **Minimum Size:** specifies the minimum size for all labels. This value should be let low, so smaller areas with many countries don’t overlap.
 - **Labels and shadow color:** they affect the Font material color and alpha value used for both labels and shadows. If you need to change individual label, you can get a reference to the TextMesh component of each label with *Country.labelGameObject* field.
- **Show Provinces:** when enabled, individual provinces/states will be highlighted when mouse hovers them. Current active province can be determined using *provinceHighlighted* property.
- **Show Cursor:** will display a cross centered on mouse cursor. Current location of cursor can be obtained with *cursorLocation* property when *mouseOver* property is true.
- **Always Visible:** will not hide the cursor cross when pointer is outside the globe.
- **Respect Other UI:** will prevent any interaction with the globe if pointer is over other UI element.
- **Navigation Time:** time in seconds for the fly to commands. Set it to zero to instant movements.

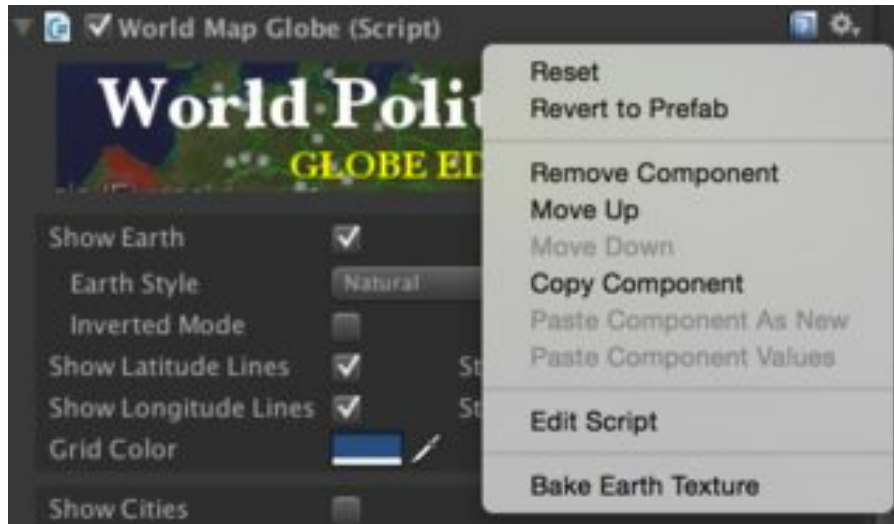
- **Autorotation Speed:** will make the Earth continuously rotate around its axis. Set it to zero to disable autorotation.
- **Allow User Rotation:** whether the user can rotate the Earth with the mouse. You can implement your own interactions setting this to false and modifying the rotation / position fields of the gameObject transform.
 - **Right Click Centers:** it will center the selection on the globe when pressing right mouse button.
 - **Constant Drag Speed:** prevents acceleration when dragging/rotating/zooming the globe.
 - **Keep Straight:** will ensure the globe is always vertically oriented at any moment.
- **Allow User Zoom:** whether the user can zoom in/out the Earth with the mouse wheel.
 - **Zoom Speed:** multiplying factor to the zoom in/out caused by the mouse Wheel (Allow User Zoom must be set to true for this setting to have any effect).
 - **Distance Min / Max:** minimum and maximum camera distance in World units.
 - **Zoom Tilt:** applies a skew to the view when approaching the globe.
- **Navigation Time:** default duration when navigating to a target country/province/city or location.
- **Navigation Mode:** this option is very important. You must decide if you want the Earth to be rotated when navigating to a target location or make the Camera rotate around the Earth instead. If you plan to use the asset as part as the UI of your application/game, then the default behaviour (Earth rotates) may suit better since it won't affect the main camera. Otherwise, choose "Camera rotates" which will make the camera fly around the Earth.
- **Follow Device GPS:** if set to true, the map will always be centered on the coordinates returned by the device GPS.
- **VR Enabled:** forces VR compatibility in normal mode (when inverted mode is enabled, VR compatibility is always on).

Choose Reset option from the gear icon to revert values to factory defaults.

Using the Bake Texture command

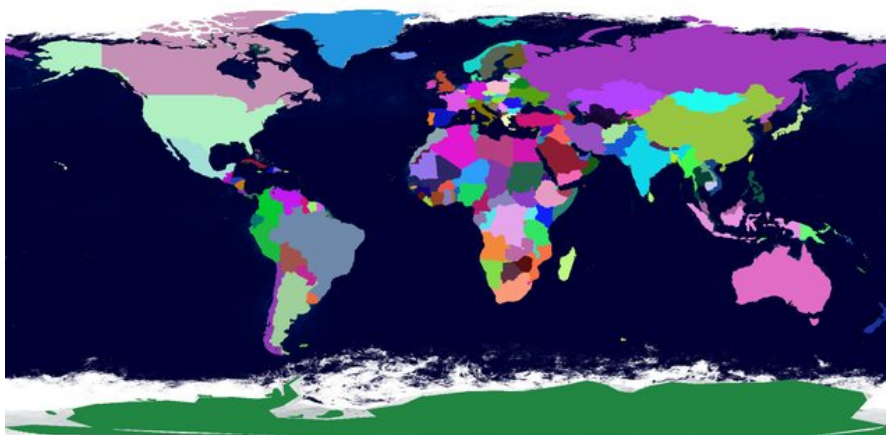
The Bake Texture command allows to visualize lots (or all) countries or provinces in colors without incurring in a performance hit (the colored areas are baked into the current texture so for those cases where you need to colorize lots of countries this will be the way to go).

This command is available from the gear in the inspector title bar:



The generated texture will be saved into Resources/Textures folder in the file EarthCustom. From this moment, you can select the new baked texture from the Style combo and choosing "Custom".

An example of a generated texture is shown below. To colorize all countries, instead of the Decorator component, a simple loop for all countries was used assigning a random color using the API `ToggleCountrySurface`.

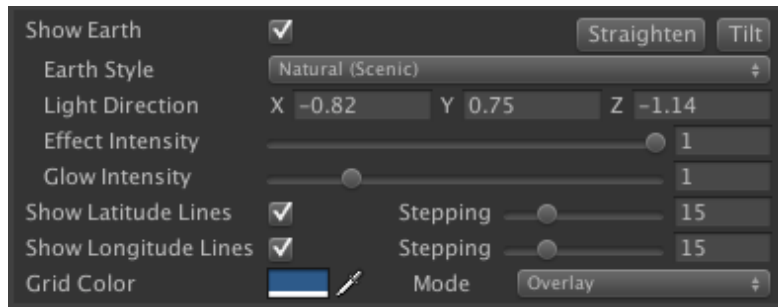


Using the Scenic styles

There're two "scenic" styles available from the Earth style combo (one which uses 2K textures and another high-res one which uses 8K textures).

This scenic style use custom planetary and glow shaders to provide an outstanding effect to the Earth map.

When you select a scenic style, some new global properties are shown in the inspector:

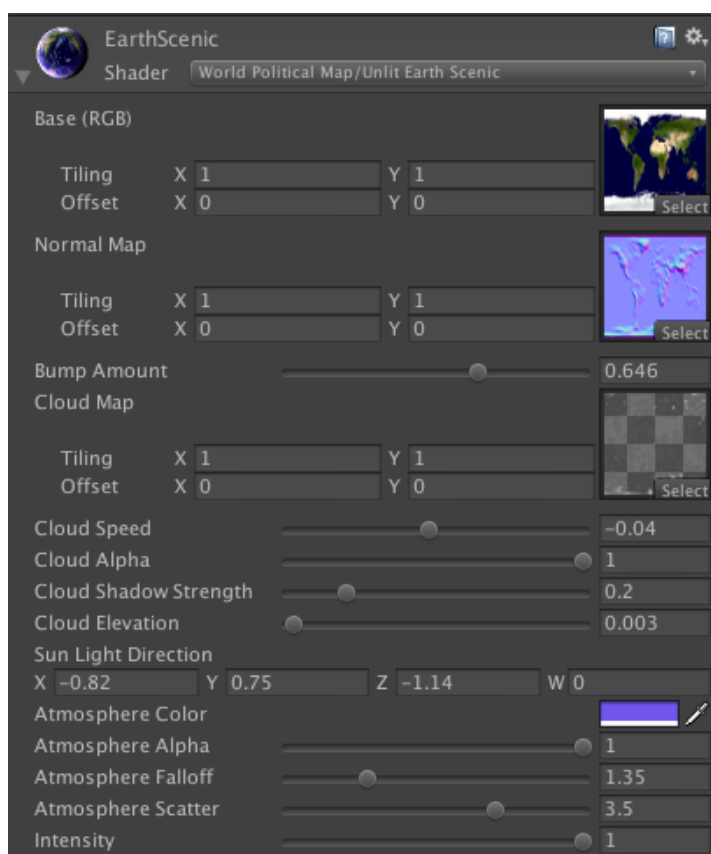


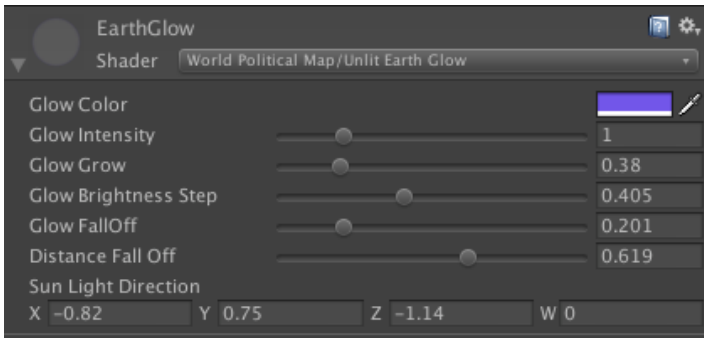
Light Direction: specify the Sun light direction. You can achieve day/light transitions changing this value.

Effect Intensity: controls how much of scenic effect is visible. This includes atmosphere, clouds and relief.

Glow Intensity: controls the brightness of the glow.

But there's more! If you want to fine control the look of the globe, you can go to the Scenic material and change the default values. To do so, scroll down the inspector and expand the Scenic material. You can do the same for the glow material which is attached to a game object called "WorldMapGlobeAtmosphere" which can be found under the WorldMapGlobe in the hierarchy:





Above image: properties of the Earth Glow shader shown in the inspector.

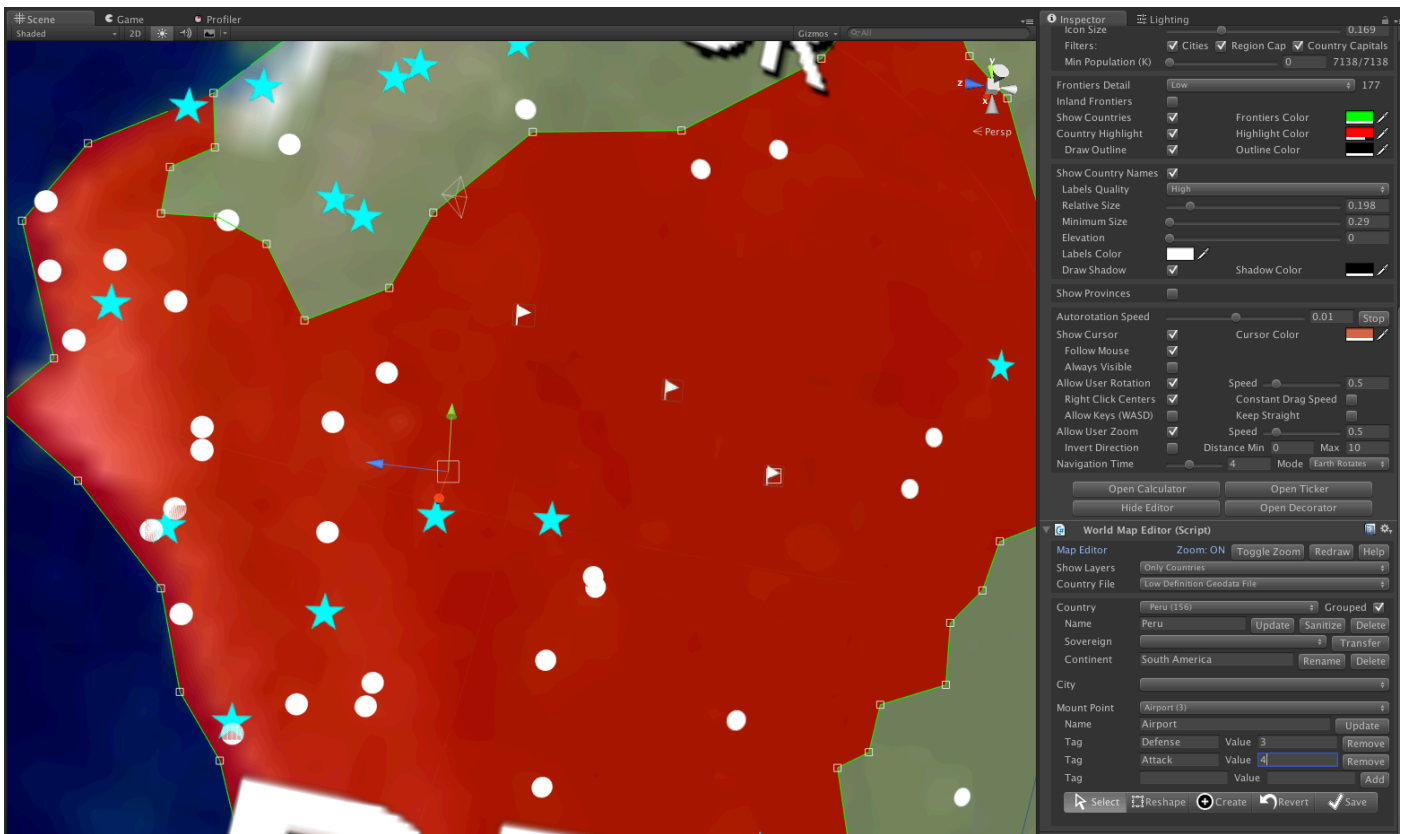
Mount Points

Mount Points are user-defined markers on the map created in the Map Editor. Basically a mount point is a special location that includes a name, a class identifier (an user-defined number) and a collection of tags.

Mount Points are useful to add user-defined strategic locations, like airports, military units, resources and other landmarks useful for your application or game. To better describe your mount points, WPM allows you to define any number of tags (or attributes) per mount point. The list of tags is implemented as a dictionary of strings pairs, so you can assign each mount point information like ("Defense", "3") and ("Attack", "2"), or ("Capacity", "10"), ("Mineral", "Uranium") and so on.

Note that Mount Points are invisible during play mode since they are only placeholder for your game objects. The list of mount points is accessible through the mountPoints property of the map API.

Mount Points appear during design time (not in playmode) as a flag:



Reducing application build size

You can greatly reduce the size of your build removing unused textures and geodata files:

- 17 Earth styles are included with associated textures that ranges from 2K to 8K image files. **Determine which styles you don't use and remove the textures you don't use** from Resources/Textures folder. You could also remove custom shaders that you don't use located in Resources/Shaders folders.

Earth style	Textures used (*.PNG files located in Resources/Textures folder)
Natural (2K, Unlit)	Earth
Natural (2K, Standard Shader)	Earth
Natural (2K, Scenic)	Earth, EarthElevationMap, EarthClouds
Natural (2K, Scenic + City Lights)	Earth, EarthElevationMap, EarthClouds, EarthCityLights
Alternate Style 1 (2K)	Earth2
Alternate Style 2 (2K)	Earth4
Alternate Style 3 (2K)	Earth5
Natural (8K, Unlit)	EarthHighRes8k
Natural (8K, Standard Shader)	EarthHighRes8k
Natural (8K, Scenic)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k
Natural (8K, Scenic + City Lights)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Natural (8K, Scenic Scatter)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k
Natural (8K, Scenic Scatter + City Lights)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Natural (8K)	EarthHighRes8k
Natural (16K)	Earth16K_BL, Earth16K_BR, Earth16K_TL, Earth16K_TR
Natural (16K, Scenic)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k
Natural (16K, Scenic + City Lights)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Natural (16K, Scenic Scatter)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k
Natural (16K, Scenic Scatter+ City Lights)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Custom	EarthCustom

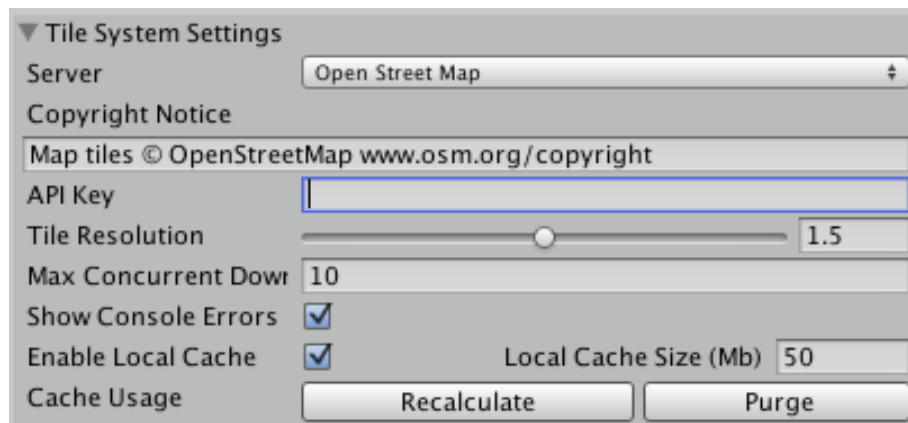
- By default, all textures are provided with TrueColor settings in Import Settings (uncompressed). Consider enabling the compression for the textures (ie. use Automatic Compression setting). **Compressing the textures will greatly reduce the build size of your application but can also reduce the image quality a bit (mostly unnoticeable).**
- You may also remove some geodata files located in Resources/Geodata folder. For example, if you don't use provinces, you can remove the provinces10 file. You can also remove the frontiers geodata file corresponding to a resolution you don't use (countries110 is low definition vs countries10 which is high definition frontiers).

Please contact us for any question you may have.

Integrating with Online Tile Map Systems

Since V9, World Political Map Globe Edition can integrate with different tile servers to enable new Earth styles that provides progressive zoom detail. To enable this feature just select “Tiled” as the Earth style in the inspector.

Once you select the “Tiled” Earth style, a new section in the inspector will be enabled:



Server: choose one of the provided tile servers. You can extend the list by editing the file WorldMapGlobeTileServers.cs inside the Scripts folder.

API Key: each server provides their tiles subject to terms of use. Most servers allows you to use them for free as long as you put their copyright notices visible in your application UI. Please refer to each server documentation online as their terms of use can change. Some servers may require you to sign up and obtain an API key or even purchase a special license if you exceed usage limitation for free tiers.

Currently the tile servers included are:

Tile Services	Terms of use / Copyright page
OpenStreetMap	http://www.osm.org/copyright
Stamen	http://maps.stamen.com
Carto	https://carto.com/location-data-services/basemaps/
Wikimedia Atlas	https://wikimediafoundation.org/wiki/Maps_Terms_of_Use
Thunderforest	http://thunderforest.com/terms/
OpenTopoMap	https://opentopomap.org/credits
MapBox	https://www.mapbox.com/pricing/
Sputnik	http://corp.sputnik.ru/maps

Tile Resolution: determines the maximum scaling when zooming in. A higher value will provide the most sharp resolution but also will lead to more tile downloads.

Max Concurrent Downloads: determines the maximum number of tile downloads at any given time. This value can be increased depending of the quality and bandwidth of your Internet connection.

Show Console Errors: if enabled, any tile request or download error will be printed out to the console if running inside Unity Editor or to the player.log file if running in a build).

Enable Local Cache: stores downloaded tiles locally into your device. The local cache size defaults to 50 Mb. The cache will automatically remove older tiles. You may increase this value to allow offline tile browsing.

Cache Usage: displays current usage of the local cache space. Allows you to remove those files.

Note that you have also access to a tile API set of functions and properties which you can use to control and customize the tile system feature with scripting.

Virtual Reality (VR)

Basic VR Usage

World Political Map Globe Edition works great in VR. When using it with a VR headset, two different modes can be used:

- “Inverted Mode” which sits you at the center of the globe and it rotates around you. This option is located under the Earth style in the inspector and automatically enables VR compatibility (you don’t need to configure anything else). Note that this option is not available for all Earth styles.
- “Normal Mode”. This is the mode where you see the entire Earth in front of you. In this case you need to verify that the “VR Enabled” parameter is checked (located at bottom of inspector).

To enable VR raycasting support, check the VR Enable toggle under Devices section of WPM Globe inspector.

This is enough to support basic virtual reality platforms such as Google VR compatible devices, Gear VR, and others.

Custom Gazes & Pointers

Unity’s VR Samples Gaze & VREyeRayCast

WPM Globe Edition also supports gaze selections through VREyeRayCast, included in Unity VR Samples. With VREyeRayCast, WPM Globe will properly hide current selection if the gaze is over an interactable element (like a menú). To support VREyeRayCast, you need to import these VR scripts from Unity and also edit WPMInternal.cs and uncomment the line that reads:

```
// #define VR_EYE_RAY_CAST_SUPPORT
```

Google VR Pointer & Controller Touch

To enable country selection and globe interaction using controller touch edit WPMInternal.cs and uncomment the line that reads:

```
// #define VR_GOOGLE
```

Programming Guide (API)

You can instantiate the prefab “WorldMapGlobe” or add it to the scene from the Editor. Once the prefab is in the scene, you can access the functionality from code through the static instance property:

```
WorldMapGlobe map;  
  
void Start () {  
    map = WorldMapGlobe.instance;  
    ...  
}
```

(Note that you can have more WorldMapGlobe instances in the same scene. In this case, the instance property will return the same object. To use the API on a specific instance, you can get the WorldMapGlobe component of the GameObject).

Structure of the API

To make it easier to find properties and methods in the source code, it has been subdivided in several files with same prefix WorldMapGlobe*:

WorldMapGlobe.cs: contains basic functionality to access the API, like WorldMapGlobe.instance above.

WorldMapGlobeCities: all related to cities. Similar for Countries, Provinces, Mount Points, Earth And Continents, as well as Markers and Lines.

Finally, all properties and methods related with user interaction is available in WorldMapGlobeInteraction.

Public Properties

Country object fields

country.name: name of the country.

country.hidden: makes the country invisible on the map.

country.continent: name of the continent to which the country belongs to.

country.regions: list of physical regions. Each region contains a list of frontier points.

country.mainRegionIndex: the index of the biggest region in the regions list.

country.center: center of the country in sphere coordinates. Use this for flying to the country.

country.customLabel: overrides the country name with a custom string (just for the label).

country.labelColorOverride: set to true to override the country color just for the label.

country.labelColor: custom color of the country label (needs labelColorOverride = true).

country.labelFontOverride: overrides the default font setting for this country.

country.labelVisible: if country name must be drawn.

country.labelRotation and labelOffset: custom rotation and displacement for this country label.

Countries API

map.countries: return a List<Country> of all countries records.

map.countryHighlighted: returns the Country object for the country under the mouse cursor (or null).

map.countryHighlightedIndex: returns the index of country under the mouse cursor (or null if none).

map.countryRegionHighlightedIndex: returns the index of the region of currently highlighted country.

map.countryLastClicked: returns the index of last clicked country.

map.countryRegionLastClicked: returns the index of last clicked country region.

map.enableCountryHighlight: set it to true to allow countries to be highlighted when mouse pass over them.

map.fillColor: color for the highlight of countries.

map.showCountryNames: enables/disables country labeling on the map.

map.showOutline: draws a border around countries highlightes or colored.

map.outlineColor: color of the outline.

map.showFrontiers: show/hide country frontiers. Same than inspector property.

map.frontiersDetail: detail level for frontiers. Specify the frontiers catalog to be used.

map.frontiersColor: color for all frontiers.

map.GetCountry(name): given a country name returns the Country object.

map.GetCountry(index): given a country index returns the Country object. Same than map.countries[index].

map.GetCountryIndex(name): returns the index of the country in the collection.

map.GetCountryIndex(localPosition): returns the country that contains a map coordinate.

map.GetCountryNearToPoint(localPosition): returns the country that contains a map coordinate or the country whose center is nearest to the map coordinate.

map.GetCountryUnderSpherePosition(spherePoint, out countryIndex, out countryRegionIndex): returns the index of the country and region located in sphere point provided.

map.ToggleCountrySurface(name, visible, color): colorize one country with color provided or hide its surface (if visible = false).

map.ToggleCountrySurface(index, visible, color): same but passing the index of the country instead of the name.

map.ToggleCountryMainRegionSurface(index, visible, color, Texture2D texture): colorize and apply an optional texture to the main region of a country.

map.ToggleCountryRegionSurface(countryIndex, regionIndex, visible, color): same but only affects one single region of the country (not province/state but geographic region).

map.HideCountrySurface(countryIndex): un-colorize / hide specified country.

map.HideCountryRegionSurface(countryIndex): un-colorize / hide specified region of a country (not province/state but geographic region).

map.HideCountrySurfaces: un-colorize / hide all colored countries (cancels ToggleCountrySurface).

map.CountryNeighbours (int countryIndex): returns a List of Countries which are neighbours of specified country index. Note that a Country can have one or more land regions (separated) which can have different neighbours each. This method returns all neighbours for all land regions of the country.

map.CountryNeighboursOfMainRegion (int countryIndex): returns a list of neighbours of the specified country having into account only the main land region of the country.

map.CountryNeighboursOfCurrentRegion (): returns a list of countries neighbours of currently selected country's region.

map.GetCountryUnderSpherePosition(spherePoint, out countryIndex, out countryRegionIndex): returns true/false and the index of the country/region of a country under specified sphere position (you can use the calculator component to convert lat/lon to spherePoint).

map.GetCountryRegionSurfaceGameObject(countryIndex, regionIndex): returns the game object corresponding to the colored surface of a given country and region, or null if that country has not been colored yet.

map.GetVisibleCountries: returns a list of visible countries on the screen.

Province object fields

province.name: name of the province.

province.countryIndex: index of the country in the countries list.

province.regions: list of physical regions. Each region contains a list of frontier points.

province.mainRegionIndex: the index of the biggest region in the regions list.

province.center: center of the country in sphere coordinates. Use this for flying to the province.

province.mainRegionArea: approximate surface area (width x height) of the main region.

Provinces API

map.provinces: return a List<Province> of all provinces/states records.

map.provinceHighlighted: returns the province/state object in the provinces list under the mouse cursor (or null if none).

map.provinceHighlightedIndex: returns the province/state index in the provinces list under the mouse cursor (or null if none).

map.GetProvinceIndex(countryIndex, provinceName): returns the province index matching the given name and belonging to specified country.

map.GetProvinceIndex(localPosition): returns the province that contains a map coordinate.

map.GetProvinceNearToPoint(localPosition): returns the province that contains a map coordinate or the province whose center is nearest to the map coordinate.

map.GetProvinceUnderSpherePosition(countryIndex, spherePoint, out provinceIndex, out provinceRegionIndex): returns the index of the province and region located in sphere point provided. Must provide the country index to which the province belongs to.

map.provinceLastClicked: returns the index of last clicked province.

map.provinceRegionLastClicked: returns the index of last province region clicked.

map.showProvinces: show/hide provinces when mouse enters a country. Same than inspector property.

map.provincesFillColor: color for the highlight of provinces.

map.provincesColor: color for provinces/states color.

map.ToggleProvinceSurface(name, visible, color): colorize one province with color provided or hide its surface (if visible = false).

map.ToggleProvinceSurface(index, visible, color): same but passing the index of the country instead of the name.

map.HideProvinceSurface(countryIndex): un-colorize / hide specified province.

map.HideProvinceSurfaces: un-colorize / hide all colorized provinces (cancels ToggleProvinceSurface).

map.ProvinceNeighbours (int provinceIndex): returns a List of Provinces which are neighbours of specified province index. Note that a province can have one or more land regions (separated) which can have different neighbours each. This method returns all neighbours for all land regions of the province.

map.ProvinceNeighboursOfMainRegion (int provinceIndex): returns a list of neighbours of the specified province having into account only the main land region of the province.

map.ProvinceNeighboursOfCurrentRegion (): returns a list of province neighbours of currently selected province's region.

map.GetProvinceRegionSurfaceGameObject(countryIndex, regionIndex): returns the game object corresponding to the colored surface of a given province and region, or null if that province has not been colored yet.

map.GetVisibleProvinces: returns a list of visible provinces on the screen.

City object fields

city.name: name of the city.

city.fullName: returns name of city + province + country names.

city.countryIndex: index of the country in the countries list.

city.province: name of the province in the provinces list.

city.unifySphereLocation: position of the city in sphere coordinates.

city.population: approximate metropolitan population.

city.cityClass: type of city (country capital, region capital, normal city).

city.latitude / longitude: latitude and longitude of the city.

Cities API

map.cities: return a List<City> of all cities records.

map.cityHighlighted: returns the city under the mouse cursor (or null if none).

map.cityLastClicked: returns the index of last clicked city.

map.showCities: show/hide all cities. Same than inspector property.

map.minPopulation: the minimum population amount for a city to appear on the map (in thousands). Set to zero to show all cities in the current catalog. Range: 0 .. 17000.

map.cityClassFilter: bitwise filter which specifies the class of cities to be drawn (Normal/Region Capitals/Country Capitals). See CITY_CLASS_FILTER enum for bit flags.

map.numCitiesDrawn: number of cities actually drawn.

map.citiesColor: color for the normal cities.

map.citiesRegionCapitalColor: color for the region capital cities.

map.citiesCountryCapitalColor: color for the country capital cities.

map.cityIconSize: custom scaling for city icons.

map.cityClassAlwaysShow: bitwise mask for the type of cities to be drawn (2 = region capitals, 4 = country capitals).

map.combineCityMeshes: whether the cities meshes should be combined into one, improving performance when lot of cities are drawn. If set to true, automatic city scaling based on distance to camera will be disabled.

map.GetVisibleCities: returns a list of visible cities on the screen.

map.GetCityNames: returns a list of city names.

map.GetCityIndex: returns the index of a given City based on different criteria or a random city.

Mount Points related

map.mountPoints: return a List<MountPoint> of all mount points records.

map.GetMountPointNearPoint: returns the nearest mount point to a location on the sphere.

map.GetMountPoints: returns a list of mount points, optionally filtered by country and province.

map.GetVisibleMountPoints: returns a list of mount points that are “visible” on the screen.

Earth related

map.showEarth: show/hide the planet Earth. Same than inspector property.

map.earthStyle: the currently texture used in the Earth.

map.autoRotationSpeed: the speed of the automatic/continuous rotation of the Earth.

map.showLatitudeLines: draw latitude lines.

map.latitudeStepping: separation in degrees between each latitude line.

map.showLongitudeLines: draw longitude lines.

map.longitudeStepping: number of longitude lines.

map.gridColor: color of latitude and longitude lines.

map.earthScenicLightDirection: Sun light direction for the scenic styles.

map.earthScenicAtmosphereIntensity: intensity of the scenic effect (0-1).

map.earthScenicGlowIntensity: brightness of the glow (0-5).

map.earthTexture: returns an instantiated texture of the Earth. Useful for texture drawing. See demo scene 7.

Navigation

map.navigationTime: time in seconds to fly to the destination (see FlyTo methods).

map.followDeviceGPS: if set to true, the map will be centered on the latitude and longitude coordinates returned by the device GPS.

map.FlyToCountry(name): start navigation at *navigationTime* speed to specified country. The list of country names can be obtained through the *countries* property.

map.FlyToCountry(index): same but specifying the country index in the *countries* list.

map.FlyToProvince(name): start navigation at *navigationTime* speed to specified province. The list of provinces names can be obtained through the *provinces* property.

map.FlyToProvince (index): same but specifying the province index in the *provinces* list.

map.FlyToCity(name): start navigation at *navigationTime* speed to specified city. The list of city names can be obtained through the *cities* property.

map.FlyToCity(index): same but specifying the city index in the *cities* list.

map.FlyToLocation (x, y, z): same but specifying the location in local Unity spherical coordinates.

map.FlyToLocation (latitude, longitude, duration): same but using the given latitude and longitude coordinates.

map.GetCurrentMapLocation: returns the sphere coordinates of the center of the visible map from camera. Similar to *cursorLocation* but restricted to the center of the globe. Note that *cursorLocation* requires the pointer to hover the globe while *GetCurrentMapLocation* will always return the center of the map from the camera perspective.

User interaction

map.mouselsOver: returns true if mouse has entered the Earth's sphere collider.

map.allowUserRotation/map.allowUserZoom: enables/disables user interaction with the map.

map.mouseWheelSensibility: multiplying factor for the zoom in/out functionality.

map.invertZoomDirection: switch direction of zoom when using the mouse wheel.

map.showCursor: enables the cursor over the map.

map.cursorFollowMouse: makes the cursor follow the map.

map.cursorLocation: current location of cursor in local coordinates (by default the sphere is size (1,1,1) so x/y/z can be in (-0.5,0.5) interval. Can be set and the cursor will move to that coordinate.

map.constraintPosition, map.constraintAngle, map.constraingPositionEnabled: restricts user rotation so a specified center (contrainPosition) is always less than constraintAngle degrees from the center of the screen.

map.tilt: skew factor (0..1) when zooming in.

map.centerOnRightClick: enables auto-centering on country/province under mouse.

map.rightClickRotates: enables globe rotation when holding right mouse button.

map.rightClickRotatingClockwise: changes direction of rotation of globe with holding right mouse button.

[Labels related](#)

map.countryLabelsSize: this is the relative size for labels. Controls how much the label can grow to fit the country area.

map.countryLabelsAbsoluteMinimumSize: minimum absolute size for all labels.

map.labelsQuality: specify the quality of the label rendering (Low, Medium, High).

map.showLabelsShadow: toggles label shadowing on/off.

map.countryLabelsColor: color for the country labels. Supports alpha.

map.countryLabelsShadowColor: color for the shadow of country labels. Also supports alpha.

map.labelsFaceToCamera: if set to true, labels will rotate automatically to ensure they can be easily read.

map.countryLabelsEnableAutomaticFade: if set to true, labels will fade in/out depending on screen size.

map.countryLabelsAutoFadeMaxHeight: max height of a label relative to screen height (0..1).

map.countryLabelsAutoFadeMaxHeightFallOff: gradient for the fade out of label.

map.countryLabelsAutoFadeMinHeight: max height of a label relative to screen height (0..1).

map.countryLabelsAutoFadeMinHeightFallOff: gradient for the fade in of label.

Tile system related

map.tileServer: the tile server to use in current session. Can be changed at any time and tiles will be refreshed automatically.

map.tileServerAPIKey: the string to be appended to any tile server URL request. Usually has the form of "apikey=xxxxx" (without quotes).

map.tileServerCopyrightNotice: the required copyright to show in your application according to the tile server terms of use.

map.tileMaxConcurrentDownloads: maximum number of concurrent tile downloads. Defaults to 10.

map.tileEnableLocalCache: enables local caching of tiles. Defaults to true.

map.tileMaxLocalCacheSize: size of the local cache in Mb. Defaults to 50.

map.tileQueueLength: current length of the download queue. 0 means all pending tiles have been downloaded. Some downloads may be cancelled if their tiles are no longer visible.

map.tileConcurrentDownloads: current number of concurrent downloads.

map.tileCurrentZoomLevel: current zoom level according to tile system standards (in WPM from 5 to 19).

map.tileWebDownloads: total number of tiles downloads from the web.

map.tileWebDownloadsTotalSize: total size in bytes of tiles downloads from the web.

map.tileCacheLoads: total number of tiles loaded from the local cache.

map.tileCacheLoadsTotalSize: total size in bytes of tiles loaded from the local cache.

map.tileCurrentCacheUsage: total size in bytes of tiles stored in the local cache.

map.tileLastError: last error occurred when downloading tiles (if any). Null is no error so far.

map.tileLastErrorDate: date of the last error when downloading tiles (if any).

map.PurgeTileCache(): removes any downloaded tile in the local cache.

Other Methods

map.ToggleContinentSurface(name, visible, color): colorize all countries belonging to specified continent with color provided or hide its surface (if visible = false).

map.HideContinentSurface(name): un-colorize / hide specified continent.

map.Show / map.Hide: faster alternative to toggling gameobject for showing/hiding globe.

Events

In addition to above methods you can listen to the following events (check out the Demo.cs script for sample code):

OnLeftClick(Vector3 spherePosition);

Occurs when user clicks on the globe with left button.

OnRightClick(Vector3 spherePosition);

Occurs when user clicks on the globe with right button.

OnDrag(Vector3 spherePosition);

Occurs when user moves the mouse over the map while holding left button.

OnCityEnter(int cityIndex);

Occurs when cursor hits a city.

OnCityExit(int cityIndex);

Occurs when cursor leaves a city.

OnCityClick(int cityIndex);

Occurs when user clicks over a city.

OnCountryBeforeEnter(int countryIndex, int regionIndex, ref bool ignoreCountry);

Occurs when cursor is about to enter a country. You can set ignoreCountry to boolean to ignore this country from highlighting.

OnCountryEnter(int countryIndex, int regionIndex);

Occurs when cursor is about to enter a country.

OnCountryExit(int countryIndex, int regionIndex);

Occurs when cursor leaves a country.

OnCountryClick(int countryIndex, int regionIndex);

Occurs when user clicks over a country.

OnProvinceBeforeEnter(int provinceIndex, int regionIndex, ref bool ignoreProvince);

Occurs when cursor is about to enter a province. You can set ignoreProvince to true so this province won't be highlighted.

OnProvinceEnter(int provinceIndex, int regionIndex);

Occurs when cursor enters a province.

OnProvinceExit(int provinceIndex, int regionIndex);

Occurs when cursor leaves a province.

OnProvinceClick(int countryIndex, int regionIndex);

Occurs when user clicks over a province.

Additional Components

World Map Calculator

This component is useful to:

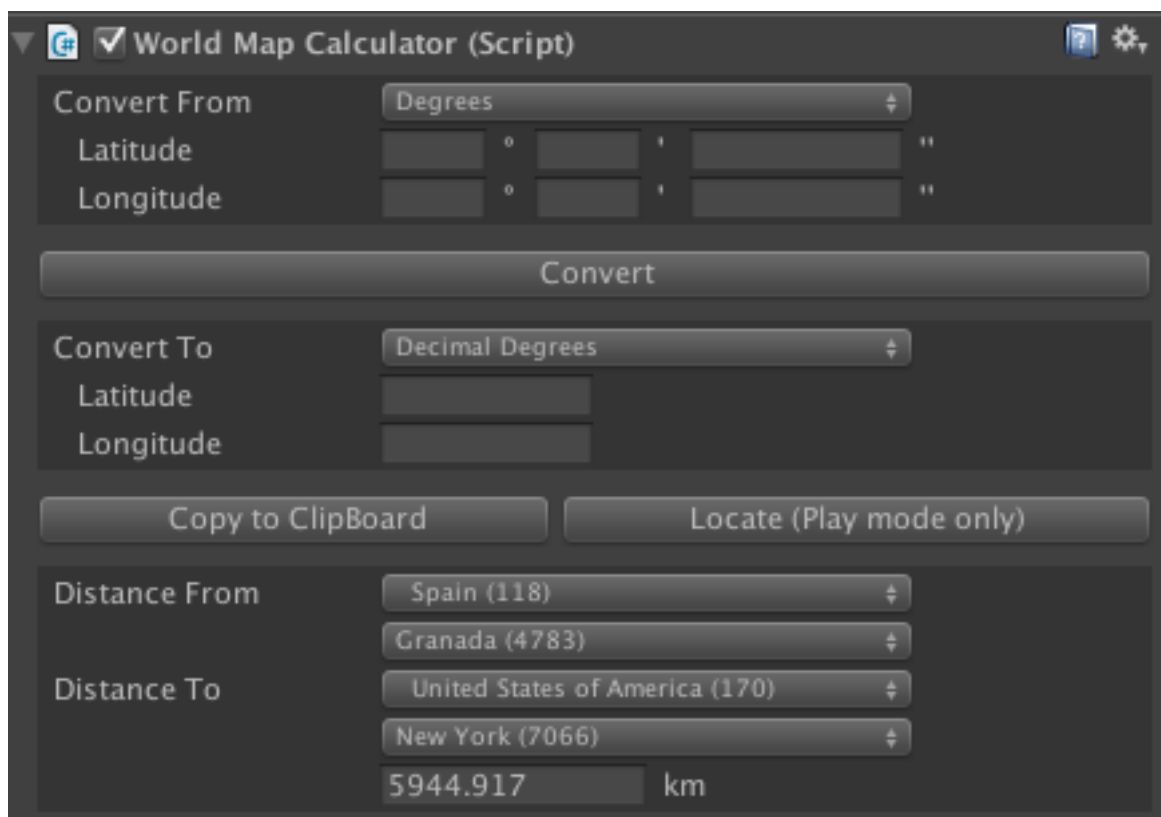
1. Convert units from one coordinate system to another (for instance from plane coordinates to degrees and viceversa).
2. Calculate the distance between cities.

You may also use this component to capture the current cursor coordinates and convert them to other coordinate system.

You may enable this component in two ways:

- From the Editor, clicking on the “Open Calculator” button at the bottom of the World Map Globe inspector.
- From code, using any of its API through **map.calc** accessor. The first time you use its API, it will automatically add the component to the globe gameObject.

On the Inspector you will see the following custom editor:



You may access the conversion API of this componet from code through **map.calc** property. The conversión task involves 3 steps:

1. Specify the source unit (eg. "**map.calc.fromUnit = UNIT_TYPE.DecimalDegrees**").
2. Assign the source parameters (eg. "**map.calc.fromLatDec = -15.281**")
3. Call **map.calc.Convert()** method.

4. Obtain the results from the fields **map.calc.to*** (eg. "**map.calc.toLatDegrees**", "**map.calc.toLatMinutes**", ...).

Note that the conversion will provide results for decimal degrees, degrees and spherical coordinates. You don't have to specify the destination unit (that's only for the inspector window, in the API the conversion is done for the 3 types).

To convert from Decimal Degrees to any other unit you use:

```
map.calc.fromUnit = UNIT_TYPE.DecimalDegrees
map.calc.fromLatDec = <decimal degree for latitude>
map.calc.fromLonDec = <decimal degree for longitude>
map.calc.Convert()
```

To convert from Degrees, you do:

```
map.calc.fromUnit = UNIT_TYPE.Degrees
map.calc.fromLatDegrees = <degree for latitude>
map.calc.fromLatMinutes = <minutes for latitude>
map.calc.fromLatSeconds = <seconds for latitude>
map.calc.fromLonDegrees = <degree for longitude>
map.calc.fromLonMinutes = <minutes for longitude >
map.calc.fromLonSeconds = <seconds for longitude >
map.calc.Convert()
```

And finally to convert from X, Y, Z (normalized) you use:

```
map.calc.fromUnit = UNIT_TYPE.SphereCoordinates
map.calc.fromX = <X position in local sphere coordinates >
map.calc.fromY = <Y position in local sphere coordinates >
map.calc.fromZ = <Z position in local sphere coordinates>
map.calc.Convert()
```

The results will be stored in (you pick what you need):

```
map.calc.toLatDec = <decimal degree for latitude>
map.calc.toLonDec = <decimal degree for longitude>
map.calc.toLatDegrees = <degree for latitude>
map.calc.toLatMinutes = <minutes for latitude>
map.calc.toLatSeconds = <seconds for latitude>
map.calc.toLonDegrees = <degree for longitude>
map.calc.toLonMinutes = <minutes for longitude >
map.calc.toLonSeconds = <seconds for longitude >
map.calc.toX = <X position in local sphere coordinates >
map.calc.toY = <Y position in local sphere coordinates >
map.calc.toZ = <Z position in local sphere coordinates>
```

You may also use the property **map.calc.captureCursor = true**, and that will continuously convert the current coordinates of the cursor (mouse) until it's set to false or you right-click the game window.

Using the distance calculator from code

The component includes the following two APIs to calculate the distances in meters between two coordinates (latitude/longitude) or two cities of the current selected catalogue.

map.calc.Distance(float latDec1, float lonDec1, float latDec2, float lonDec2)

map.calc.Distance(City city1, City city2)

map.calc.Distance(Vector3 spherePosition1, Vector3 spherePosition2)

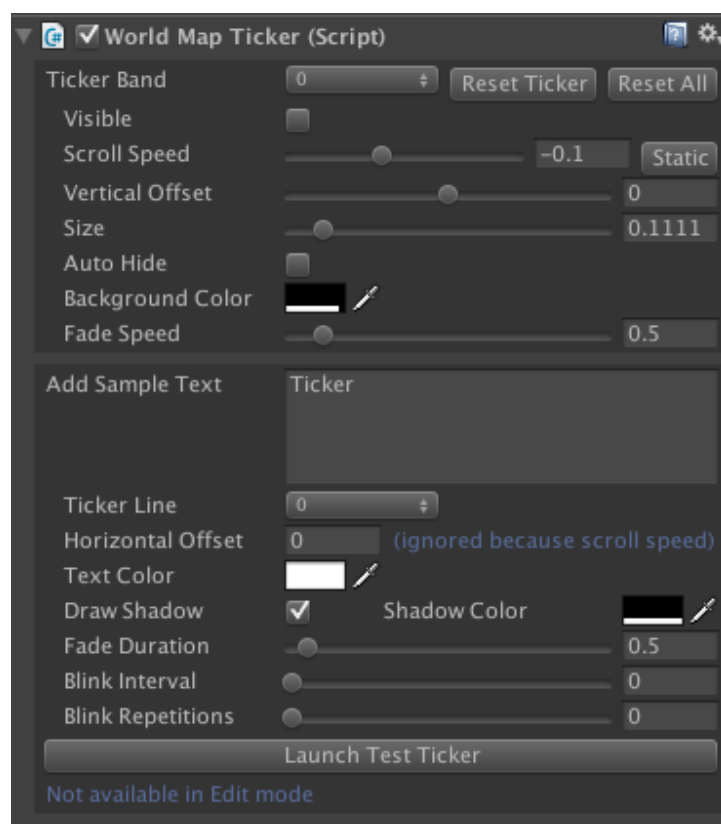
World Map Ticker Component

Use this component to show impact banners over the map. You can show different banners, each one with different look and effects. Also you can add any number of texts to any banner, and they will simply queue (if scrolling is enabled).

Similarly to the World Map Calculator component, you may enable this component in two ways:

- From the Editor, clicking on the “Open Ticker” button at the bottom of the World Map Globe inspector.
- From code, using any of its API through **map.ticker** accessor. The first time you use its API, it will automatically add the component to the globe gameObject.

On the Inspector you will see the following custom editor:



The top half of the inspector corresponds to the Ticker Bands configurator. You may customize the look&feel of the 9 available ticker bands (this number could be incremented if needed though). Notes:

- Ticker bands are where the ticker texts (second half of the inspector) scrolls or appears.
- A ticker band can be of two types: scrollable or static. You make a ticker band static setting its scroll speed to zero.
- Auto-hide will make the ticker band invisible when there're no remaining texts on the band.
- The fade speed controls how quickly should the band appear/disappear. Set it to zero to disable the fade effect.

It's important to note that everything you change on the inspector can be done using the API (more below).

In the second half of the inspector you can configure and create a sample ticker text. Notes:

3. Horizontal offset allows you to control the horizontal position of the text (0 equals to zero longitude, being the range -0.5 to 0.5).
4. Setting fade duration to zero will disable fading effect.
5. Setting blink interval to zero will disable blinking and setting repetitions to zero will make the text blink forever.

The API can be accessed through `map.ticker` property and exposes the following methods/fields:

`map.ticker.NUM_TICKERS`: number of available bands (slots).

`map.ticker.tickerBands`: array with the ticker bands objects. Modifying any of its properties has effect immediately.

`map.ticker.GetTickerTextCount()`: returns the number of ticker texts currently on the scene. When a ticker text scrolls outside the ticker band it's removed so it helps to determine if the ticker bands are empty.

`map.ticker.GetTickerTextCount(tickerBandIndex)`: same but for one specific ticker band.

`map.ticker.GetTickerBandsActiveCount()`: returns the number of active (visible) ticker bands.

`map.ticker.ResetTickerBands()`: will reset all ticker bands to their default values and removes any ticker text they contain.

`map.ticker.ResetTickerBand(tickerBandIndex)`: same but for an specific ticker band.

`map.ticker.AddTickerText(tickerText object)`: adds one ticker text object to a ticker band. The ticker text object contains all the necessary information.

The `demo.cs` script used in the Demo scene contains the following code showing how to use the API:

```
// Sample code to show how tickers work
void TickerSample() {
    map.ticker.ResetTickerBands();

    // Configure 1st ticker band: a red band in the northern hemisphere
    TickerBand tickerBand = map.ticker.tickerBands[0];
    tickerBand.verticalOffset = 0.2f;
    tickerBand.backgroundColor = new Color(1,0,0,0.9f);
    tickerBand.scrollSpeed = 0;    // static band
    tickerBand.visible = true;
    tickerBand.autoHide = true;

    // Prepare a static, blinking, text for the red band
    TickerText tickerText = new TickerText(0, "WARNING!!");
    tickerText.textColor = Color.yellow;
    tickerText.blinkInterval = 0.2f;
    tickerText.horizontalOffset = 0.1f;
    tickerText.duration = 10.0f;
```



```
// Draw it!
map.ticker.AddTickerText(tickerText);

// Configure second ticker band (below the red band)
tickerBand = map.ticker.tickerBands[1];
tickerBand.verticalOffset = 0.1f;
tickerBand.verticalSize = 0.05f;
tickerBand.backgroundColor = new Color(0,0,1,0.9f);
tickerBand.visible = true;
tickerBand.autoHide = true;

// Prepare a ticker text
tickerText = new TickerText(1, "INCOMING MISSLE!!");
tickerText.textColor = Color.white;

// Draw it!
map.ticker.AddTickerText(tickerText);
}
```

World Map Decorator

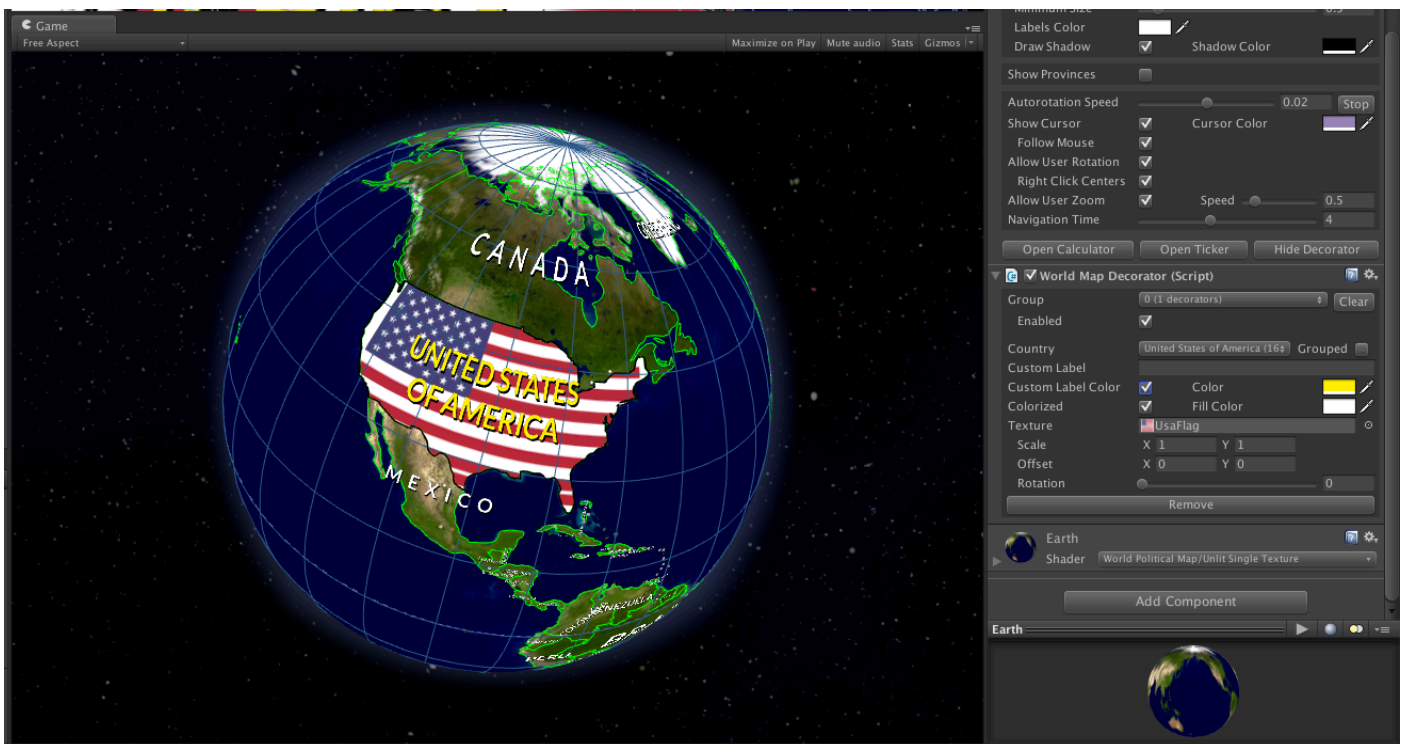
This component is used to decorate parts of the map. Current decorator version supports:

- ✓ Customizing the label of a country
- ✓ Colorize a country with a custom color
- ✓ Assign a texture to a country, with scale, offset and rotation options.

You may use this component in two ways:

- From the Editor, clicking on the “Open Decorator” button at the bottom of the World Map Globe inspector.
- From code, using any of its API through **map.decorator** accessor. The first time you use its API, it will automatically add the component to the globe gameObject.

In the Editor, this component has this interface (on the right):



Important!

Decorators also expose an API but it's better to use the public existing API at runtime to colorize/texturize countries and provinces (see `ToggleCountrySurface` and similar functions). Use decorators preferably at design time to customize the look of the map.

The API of this component has several methods but the most important are:

map.decorator.SetCountryDecorator(int groupIndex, string countryName, CountryDecorator decorator)

This will assign a decorator to specified country. Decorators are objects that contains customization options and belong to one of the existing groups. This way you can enable/disable a group and all decorators of that group will be enabled/disabled at once (for instance, you may group several countries in the same group).

map.decorator.RemoveCountryDecorator(int groupIndex, string countryName)

This method will remove a decorator from the group and its effects will be removed.

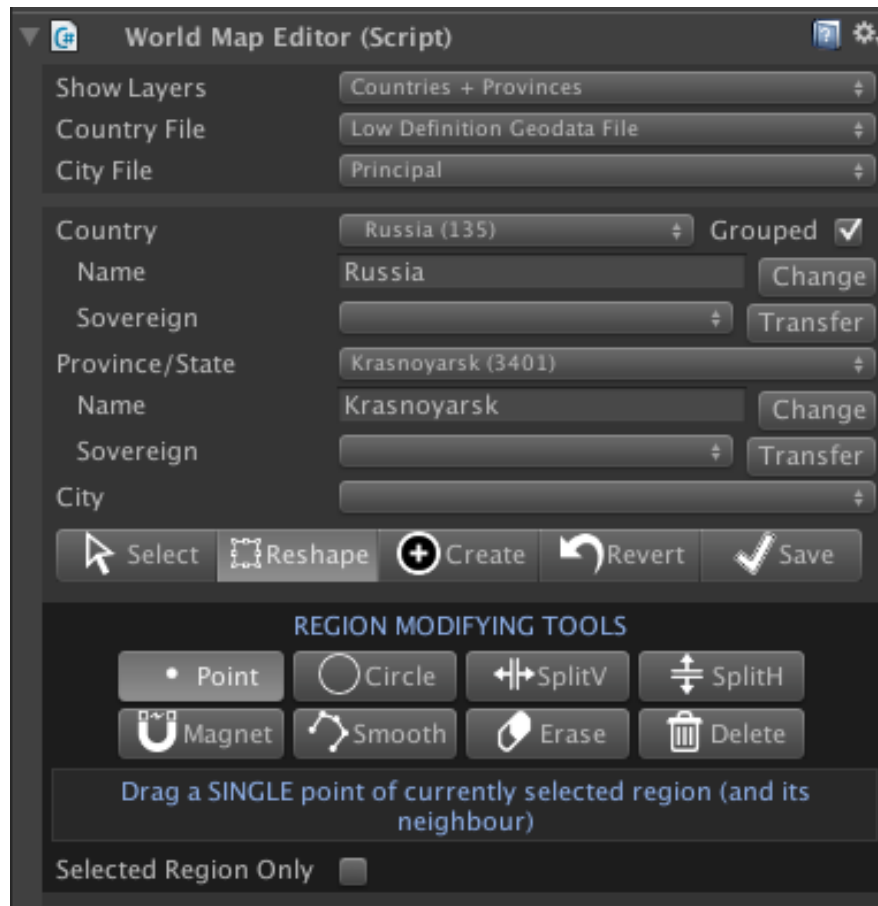
A decorator object has the following fields:

- **countryName**: the name of the country to be decorated. It will be assigned automatically when using SetCountryDecorator method.
- **customLabel**: leave it as "" to preserve current country label.
- **isColorized**: if the country is colorized.
- **fillColor**: the colorizing color.
- **labelOverridesColor**: if the color of the label is specified.
- **labelColor**: the color of the label.
- **labelVisible**: sets the label visible or hidden.
- **labelOffset**: specifies a manual offset for the label with respect to the country center. A default value of (0,0) will make the label to automatically shift if needed.
- **labelRotation**: manual rotation for the label in degrees (0-359). If set to zero, the label can be automatically rotated by the system.
- **texture**: the texture to assign to the country.
- **textureScale**, **textureOffset** and **textureRotation** allows to tweak how the texture is mapped to the surface.

World Map Editor Component

Use this component to modify the provided maps interactively from Unity Editor (it doesn't work in play mode). To open the Map Editor, click on the "Open Editor" button at the bottom of the World Map inspector.

On the Inspector you will see the following custom editor:



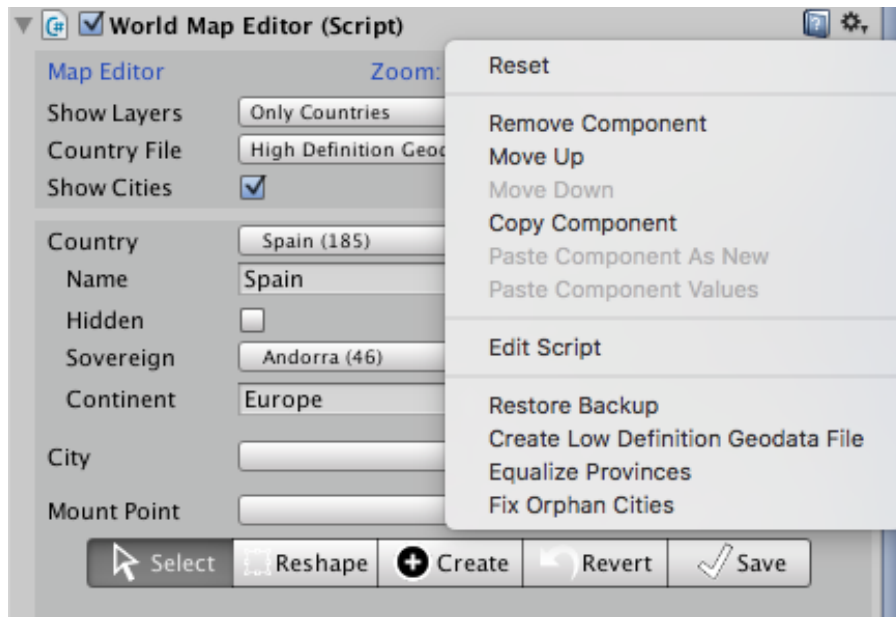
Description:

- **Show Layers:** choose whether to visualize countries or countries + provinces. high layer to modify.
- **Country File:** choose which file to edit:
 - Low-definition geodata file (110m:1 scale)
 - High-definition geodata file (30m:1 scale)
- **Country:** the currently selected country. You can change its name or "sell" it to another country clicking on transfer.
- **Province/State:** the currently selected province/state if provinces are visible (see Show Layers above). As with countries, you can change the province's name or even transfer it to another country.
- **City:** the currently selected city.

Main toolbar

- **Select:** allows you to select any country, province or city in the Scene view. Just click over the map!
- **Reshape:** once you have either a country, province or city selected, you can apply modifications. These modifications are located under the Reshape mode (see below).
- **Create:** enable the creation of cities, provinces or countries.
- **Revert:** will discard changes and reload data from current files (in Resources/Geodata folder).
- **Save:** will save changes to files in Resources/Geodata folder.

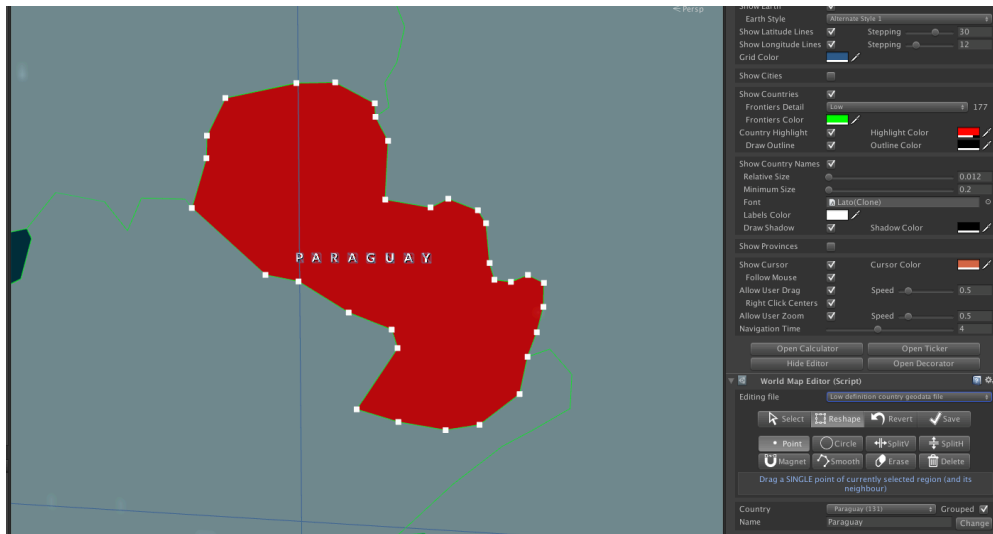
If you click the gear icon on the inspector title bar, you will see 2 additional options:



- **Restore Backup:** the first time you save changes to disk, a backup of the original geodata files will be performed. The backed up files are located in Backup folder inside the main asset folder. You may manually replace the contents of the Resources/Geodata folder by the Backup contents manually as well. This option do that for you.
- **Create Low Definition Geodata File:** this option is only available when the high-definition geodata file is active. It will automatically create a simplistic and reduced version (in terms of points) and replace the low-definition geodata file. This is useful only if you use the high-definition geodata file. If you only use the low-definition geodata file, then you may just change this map alone.
- **Equalize Provinces:** this option allows you to merge provinces in each country so the resulting number is in the given range. Useful for generating simplified provinces map.
- **Fix Orphan Cities:** this option will search any city without province or country assigned and fix it. It will assign the country that surrounds the city (or the nearest one). It will also take the province surrounding the city and assign to it. Once you execute this option, remember to save!

Reshaping options

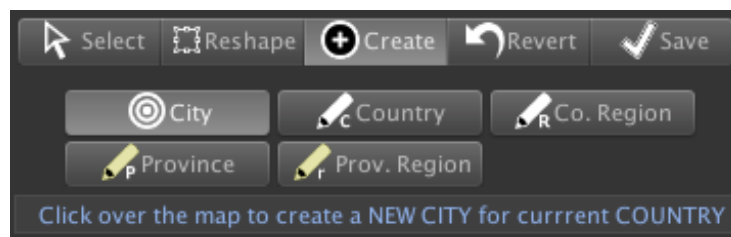
When you select a country, the Reshape main option will show the following tools:



- **Point tool:** will allow you to move one point at a time. Note that the corresponding point of the neighbour will also be moved along (if it exists). This way the frontier between two regions is easily modified in one step, avoiding gaps between adjacent regions.
- **Circle tool:** similar to the point tool but affects all points inside a circle. The width of the circle can be changed in the inspector. Note that points nearer to the center of the circle will move faster than the farther ones unless you check the “Constant Move” option below.
- **SplitV:** will split vertically the country or region. The splitted region will form a new country/province with name “New X” (X = original country name)
- **SplitH:** same but horizontally.
- **Magnet:** this useful works by clicking repeatedly over a group of points that belong to different regions. It will try to make them to join fixing the frontier. Note that there must be a sufficient number of free points so they can be fused. You can toggle on the option “Agressive Mode” which will move all points in the circle to the nearest points of other region and also will remove duplicates.
- **Smooth:** will create new points around the border of the selected region.
- **Erase:** will remove the points inside the selection circle.
- **Delete:** will delete selected region or if there're no more regions in the current country or province, this will remove the entity completely (it disappear from the country /province array).

Create options

In “Create mode” you can add new cities, provinces or countries to the map:



Note that a country is comprised of one or more regions. Many countries have only one region, but those with islands or colonies have more than one. So you can add new regions to the selected country or create a new country. When you create a new country, the editor automatically creates the first / main region.

Also note that the main region of a country is the biggest one in terms of euclidean area. Provinces have also regions, and can have more than one.

Editing Tips

This section contains useful tips for a correct usage of the Map Editor. Please read carefully before contacting us with any issue.

1. Before start making changes, determine if you need the high-definition frontiers file or not. Current release of the asset is quite optimized so unless you have specific low-end platform requirements we'd recommend you to use the high definition mode.
Of course if you don't need such detail in your project, then you can just work with the low-definition file. Note that the high-def and low-def files are different. That means that changes to one file will not affect the other. This may duplicate your job, so it's important to decide if you want to modify both maps or only the low-def map.
2. We strongly recommend using the editor with a globe scale of at least 1000. You may switch between normal scale (1,1,1) and "editing scale" (1000,1000,1000) pressing the "Toggle Zoom" button in the editor inspector.
3. When you decide to modify the high-definition file, you will want to use the command "Create Low Definition Geodata File" from the gear command, and review the low-def map afterwards.
4. If you make any mistake using the Point/Circle tool, you can Undo (Control/Command + Z or Undo command from the Edit menu). Please note that undo is not supported for complex operations, like creating a new country, deleting or transferring regions. So save often!
5. You can use the Revert button and this will reload the geodata files from disk (changes in memory will be lost) – so if you saved before performing a complex operation and it went "bad" you must click "Revert" (which in fact acts as an Undo except for you have to save first!).
6. If you modified the geodata files in Resources/Geodata and want to recover original files, you can use the Restore Backup command from the gear icon, or manually replace the Resources/Geodata files with those in the Backup folder. As a last resort you may replace current files with the originals in the asset .unitypackage.
7. Starting V4, there're two new buttons in the Editor inspector:
 - Redraw: this will delete any children from the globe and redraws every object/layer. This is a reset button for the scene but won't discard any change to the frontiers.
 - Sanitize: this option can be useful if for any reason the frontiers of a country goes wrong. This option will check for self-crossing segments in the polygon and will remove them. It can't fix every problem, but most of the time you don't see a country correctly filled in color is due to a self-crossing polygon.
8. Remember to visit us at kronnect.com for new updates and additional questions/support. Thanks.

World Flags and Weather Symbols

This package is available as a separate purchase and includes +270 vector and raster images of country flags and weather symbols:

For more information, please consult the Asset Store page:

<https://www.assetstore.unity3d.com/#!/content/69010>

Once imported into the project, the names of the flag texture files equal to the country names used in our map assets so you can add flag icons or texture countries with their flag with minimal effort.

The code to texture the country surface with its flag would be:

```
// Get reference to the API
WorldMapGlobe map = WorldMapGlobe.instance;

// Choose a country
string countryName = "China";

// Load texture for the country
Texture2D flagTexture = Resources.Load<Texture2D> ("Flags/png/" + countryName);

// Apply texture to the country main region (ignore islands)
int countryIndex = map.GetCountryIndex(countryName);
map.ToggleCountryMainRegionSurface(countryIndex, true, flagTexture);
```