WORLD POLITICAL MAP
GLOBE EDITION

+ Map Editor!

RELEASE NOTES
V.5.3 / ABR.2016

Index

Introduction 3

QuickStart and Demo Scene 3

Support & Contact info 3

How to use the asset in your project 4

Custom Editor Properties 4

Using the Bake Texture command 7

Using the Scenic styles 8

Mount Points 9

Programming Guide (API) 10

Structure of the API *10*

Public Properties *10*

Country related 10

Province related 12

Cities related 13

Mount Points related 13

Earth related 13

User interaction 14

Labels related 14

Other Methods *15*

Events *15*

Additional Components 16

World Map Calculator *16*

Using the distance calculator from code 18

World Map Ticker Component *19*

World Map Decorator *22*

World Map Editor Component *24*

Main toolbar 25

Reshaping options 25

Create options 26

Editing Tips *27*

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.
- ✓ 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 beteween 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#).

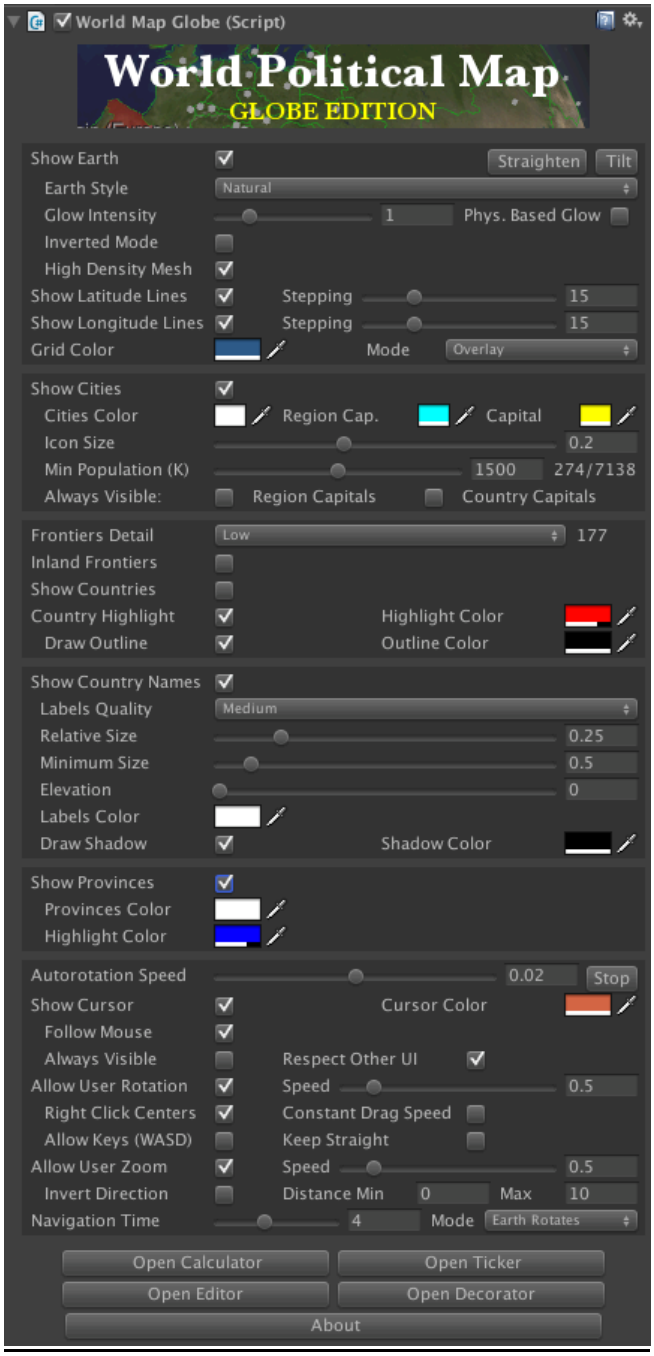
Support & Contact info

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 Editor 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.

- **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:

    - **Labels Quality**: controls the size of the RenderTexture, 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:** wether 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.

- **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.

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

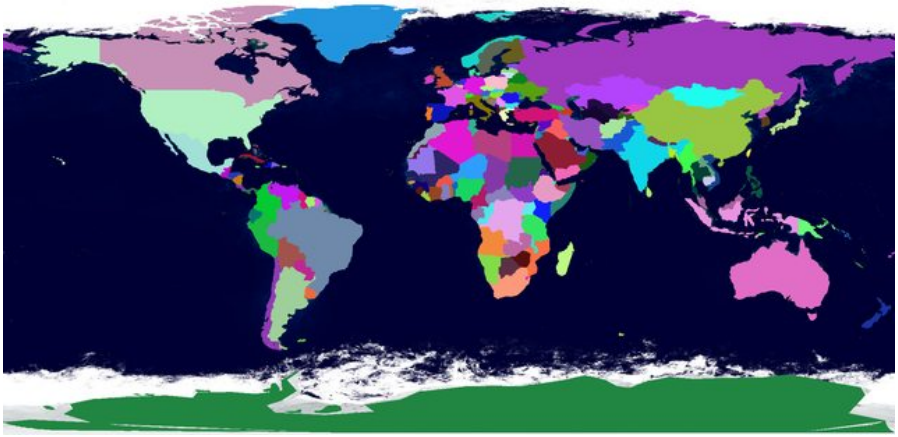
Using the Bake Texture command

The Bake Texture command allows to visualize lots (or all) countries or provinces in colors without incurring in a performance hit (the colorized 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.

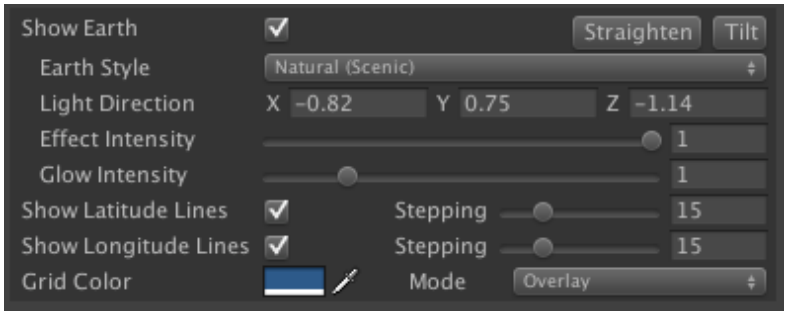


Using the Scenic styles

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 oustanding effect to the Earth map.

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

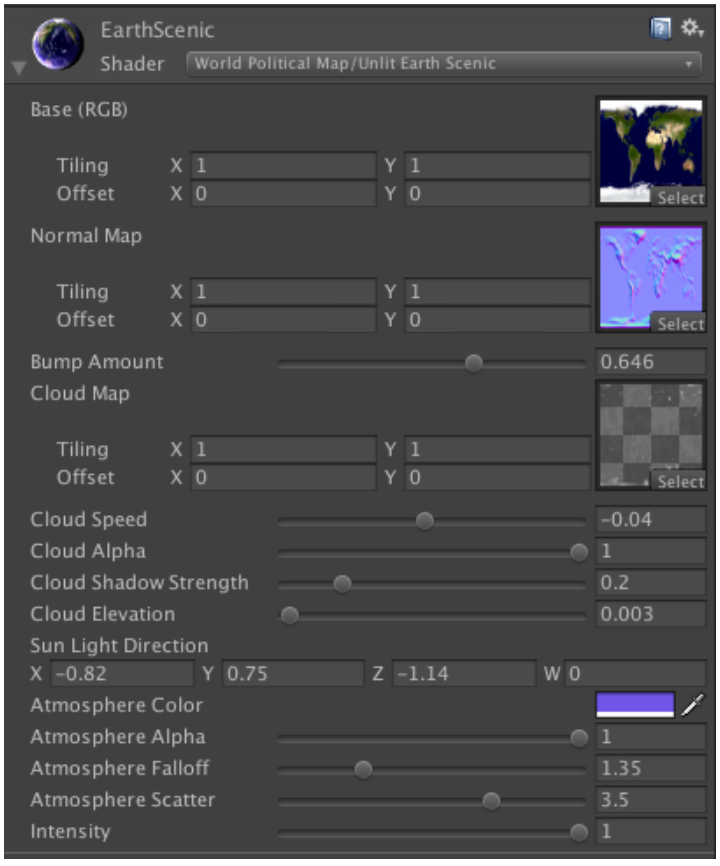


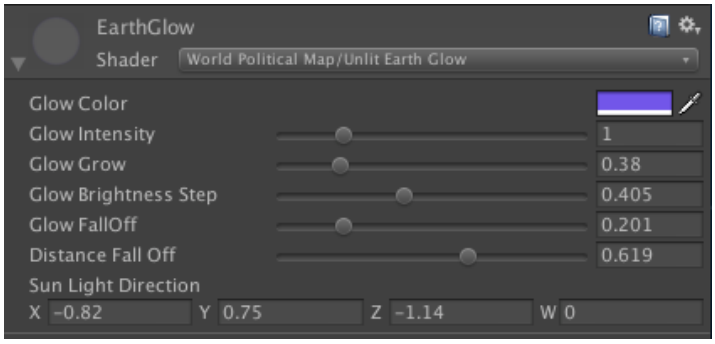
Light Direction: specify the Sun light direction. You can achieve day/light transitions changing this value.

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 showin in the inspector.

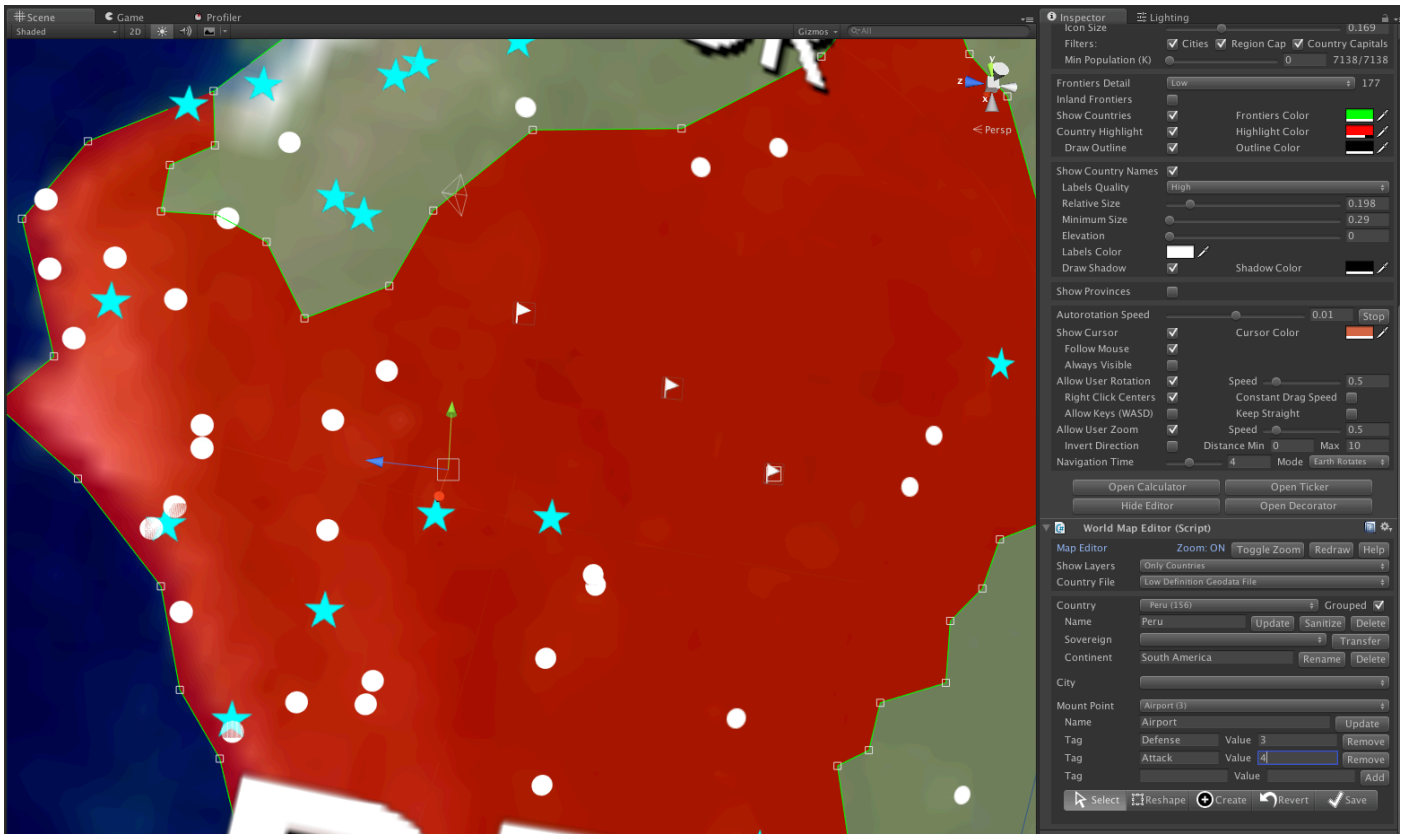
Mount Points

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 accesible through the mountPoints property of the map API.

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



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 returns 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 related

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

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

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

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

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 colorized countries (cancels ToggleCountrySurface).

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.

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

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

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).

Province related

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.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.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.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).

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.

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

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

Cities related

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 mínimum 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.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.

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.

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).

User interaction

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

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

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 (contraintPosition) is always less than constraintAngle degrees from the center of the screen.

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.

Other Methods

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

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

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

Events

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

delegate void **OnCityEnter**(int cityIndex);
delegate void **OnCityExit**(int cityIndex);
delegate void **OnCityClick**(int cityIndex);
delegate void **OnCountryEnter**(int countryIndex, int regionIndex);
delegate void **OnCountryExit**(int countryIndex, int regionIndex);
delegate void **OnCountryClick**(int countryIndex, int regionIndex);
delegate void **OnProvinceEnter**(int provinceIndex, int regionIndex);
delegate void **OnProvinceExit**(int provinceIndex, int regionIndex);
delegate void **OnProvinceClick**(int countryIndex, int regionIndex);

Additional Components

World Map Calculator

This component is useful to:

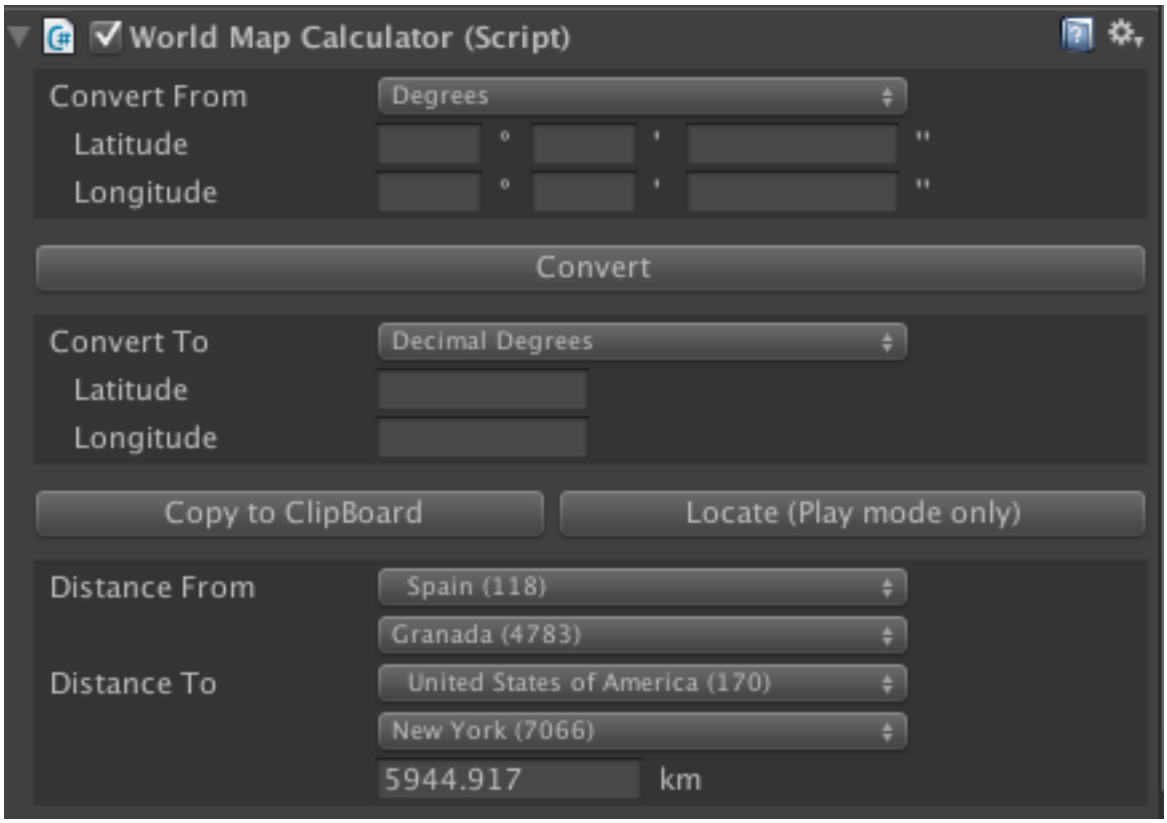
1. Convert units from one coordinate system to another (for instance from plane coordinates to degrees and viceversa).
2. Calculate the distance between cities.

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)

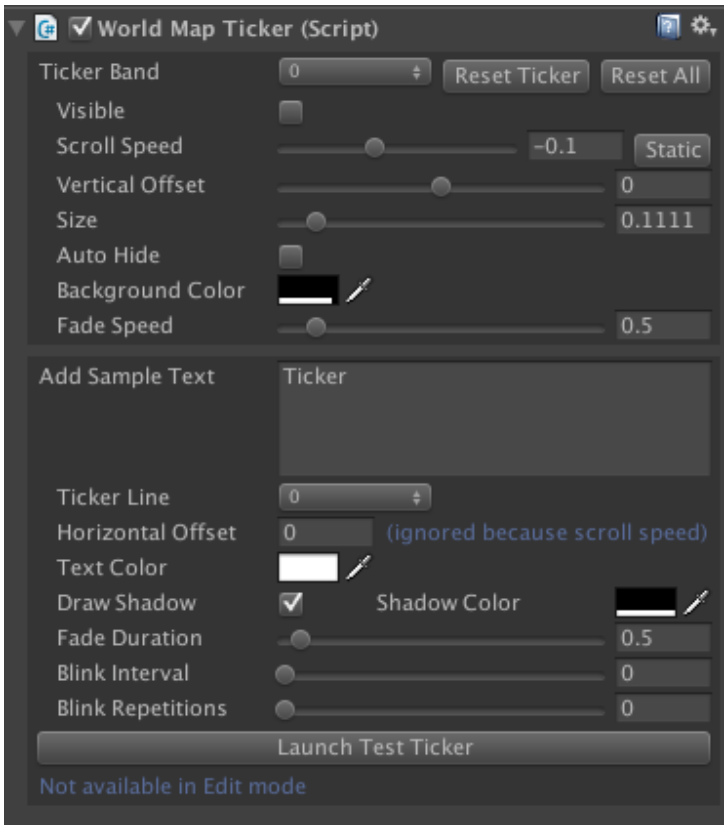
World Map Ticker Component

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 neccesary 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):

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:

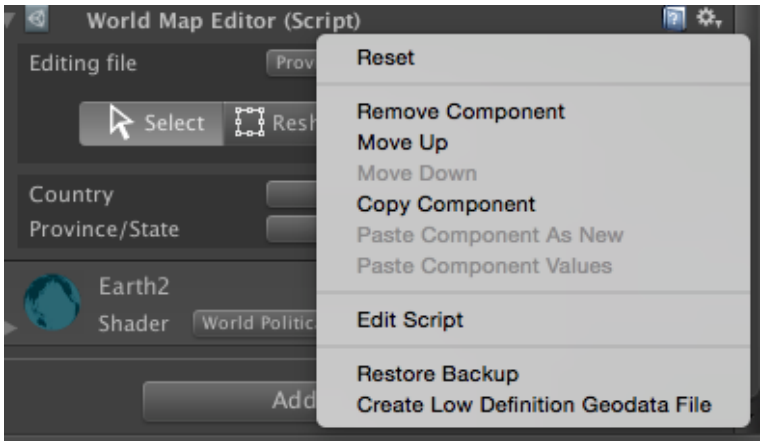
Description:

6. **Show Layers**: choose whether to visualize countries or countries + provinces.hich layer to modify.
7. **Country File**: choose which file to edit:
    - Low-definition geodata file (110m:1 scale)
    - High-definition geodata file (30m:1 scale)
8. **Country:** the currently selected country. You can change its name or "sell" it to another country clicking on transfer.
9. **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 ore ven transfer it ot another country.
10. **City:** the currently selected city.

Main toolbar

11. **Select:** allows you to select any country, province or city in the Scene view. Just click over the map!
12. **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).
13. **Create:** enable the creation of cities, provinces or countries.
14. **Revert**: will discard changes and reload data from current files (in Resources/Geodata folder).
15. **Save**: will save changes to files in Resources/Geodata foler.

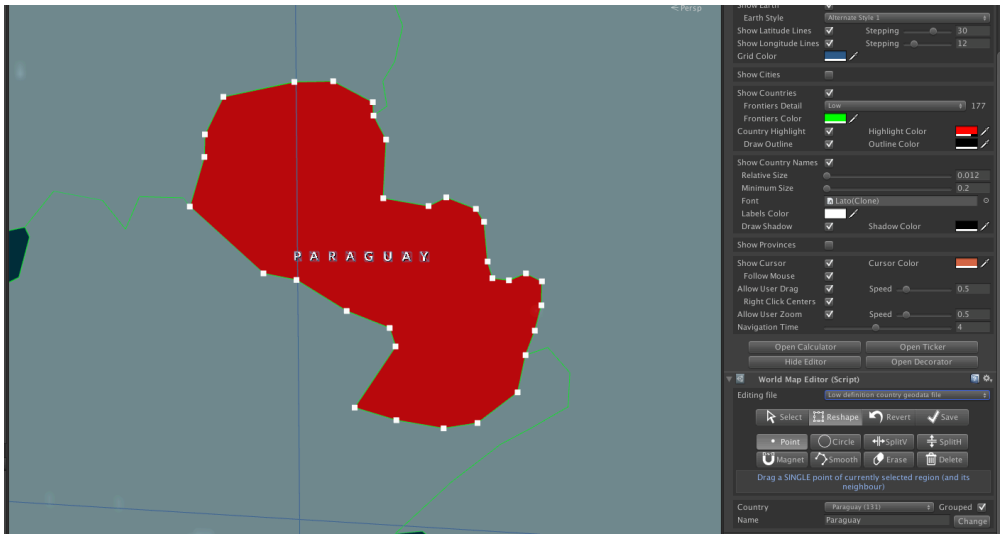
If you click the gear icon on the inspector title bar, you will see 2 additional options:



16. **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.
17. **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.

Reshaping options

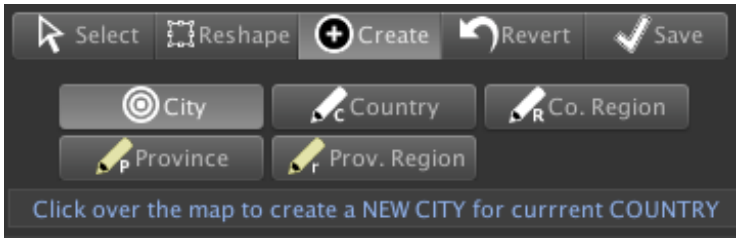
When you select a country, the Reshape main option will show the following tools:



18. **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 exits). This way the frontier between two regions is easily modified in one step, avoiding gaps between adjacent regions.
19. **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.
20. **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)
21. **SplitH**: same but horizontally.
22. **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.
23. **Smooth**: will create new points around the border of the selected region.
24. **Erase**: will remove the points inside the selection circle.
25. **Delete**: will delete selected region or if there're no more regiosn 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 comprissed 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 transfering regions. So save often!

5. You can use the Revert button and this will reload the geodata files from disk (changes in memory wiil 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 Resourcess/Geodata files with those in the Backup folder. As a last resort you may replace currnet 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 quesitons/support. Thanks.