

GO Map – 3D Map for AR Gaming

By Alan Grant

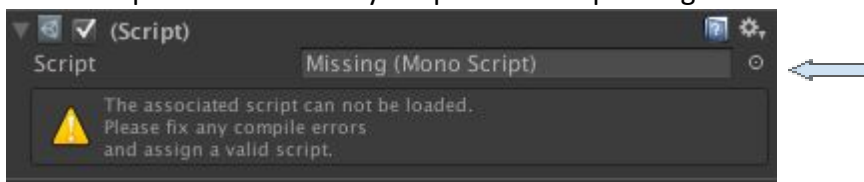
- [1.5 -> 1.6 Migration guide](#)
- [Documentation](#)
- [First start guide](#)
- [POI - Demo Scene](#)
- [FAQ](#)

1.5 -> 1.6 Migration guide

If you are downloading Go Map for the first time you can skip this section.

In the 1.6 version namespace GoMap is added and some classes were refactored. This may cause the package update to give some errors. Nothing to worry about! Do the following steps to quickly fix this issue.

1. **BACKUP YOUR CODE:** Please do a full backup of your working code. Just to be certain.
2. Manually delete Map.cs and SmartTile.cs as they are not used anymore. Unity should remove them for you but it might happens that it doesn't. Don't worry and remove them yourself. This scripts are replaced from **GOMap.cs** and **GOTile.cs**.
3. If you had a Map.cs attached to a gameobject in a scene and now the script is "missing", just replace it with GOMap.cs script, do it from the round button inside the inspector next to the script name. None of your previous map configuration will be lost.



4. If you had one of these classes modified it's a bit more difficult. I think the best is to merge what you did inside the new GOMap.cs and GOTile.cs classes. Note that these classes may change again (not the name) in the next releases.
5. Play inside the editor, it should work fine now.
6. If you have a FileHanlder.cs error is probably because your unity is set to compile for WebPlayer. See the FAQ section.
7. If you are still having troubles don't hesitate to contact me at alangrant.unity@gmail.com! And sorry for the inconvenience, but the class renaming had to be done.

Documentation

GO Map is a real-time 3D map renderer for AR Gaming purposes.

It's really easy to use and customize, just build it on your device and go outside to try!

Or use it inside editor, it works great too =)

Every element of the map is customizable inside the editor without touching a single line of code.

The Following documentation is based on the "Demo scene" scene. After reading this document you can safely delete anything inside the "demo" folder and make your own map.

The scene contains 3 game objects:

- Location Manager
- Map
- Avatar

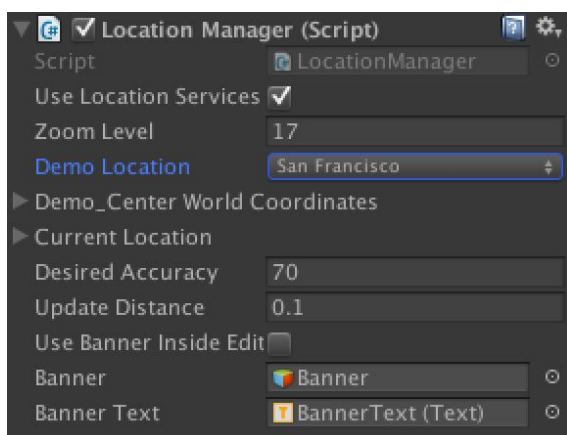
Location manager handles the GPS positioning (live or demo) and Map is the 3D map factory. Avatar is a game-like setup (character, clouds, gestures, land, etc) and it's completely (and recommended) editable / replaceable.

Location Manager

LocationManager.cs

Using the unity Input.Location class handles the gps positioning of the user's device, converting the GPS coordinates in unity world coordinates (2D x,z). It uses events and delegation to notify the map that the users has moved to refresh the map tiles.

You can simulate GPS movement while playing inside the editor using W,A,S,D keys.



Parameters:

- (bool) useLocationServices: toggles between using GPS positioning or fixed location. It is always false when playing inside editor.
- (int) zoomLevel: the default map tiles zoomLevel. Greater is the zoomLevel (0-22) better is the polygons and lines detail.
- (enum) demoLocation: a list of Location presets to use when playing inside editor or without locationServices. If it's set to "custom" then the demo_Center World Coordinates is used (lat, lon, alt).
- (int) desiredAccuracy: the value in meters to accept the location input data.
- (float) updateDistance: the value in meters to consider location changes.

Banner:

The Location Manager object has a canvas in its hierarchy that is just an animated top bar showing the location status.

- (bool) useBannerInsideEditor: animates the canvas while playing inside the editor.
- (Panel) banner: the banner object. (disables animation if set to null)
- (Text) Banner text: the text object. (disables animation if set to null)

Coordinates.cs

As said before you don't have to edit any code to use or customize this map, but in case you would you should put attention to this particular class.

Coordinates.cs is widely used inside the package to wrap GPS coordinates and converting them to and from Vector3 coordinates.

It offers a series of useful methods to work with geographic data, use it if you want to add some Location logic to your app.

This is how you convert a lat lon to vector3 and place objects on map!

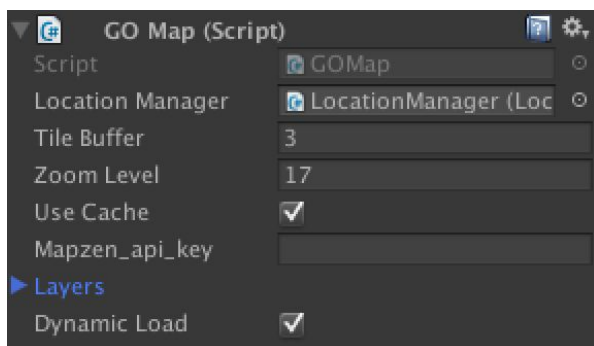
```
Coordinates coordinates = new Coordinates (lat, lng,0);  
gameObject.transform.localPosition = coordinates.convertCoordinateToVector(0);
```

Map

GOMap.cs

The Map script holds map settings and customization, it is responsible for the creation of map tiles and their destruction while moving in the real world.

All map data is taken from the MapZen vector maps API, once downloaded is cached on the device and used to create the map geometries.

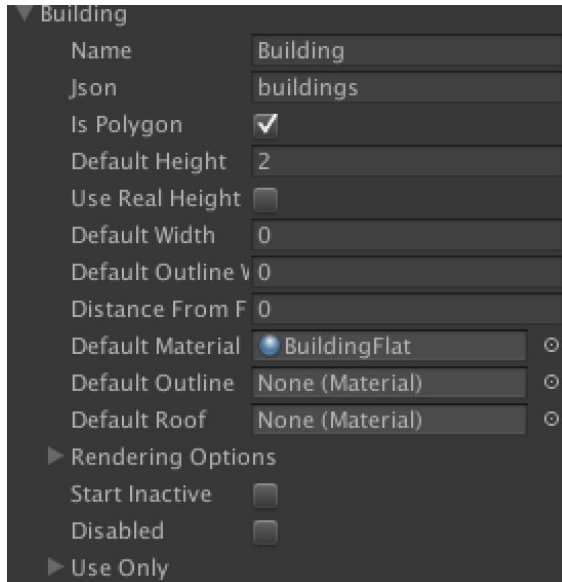


Parameters:

- (LocationManager) locationManager: the instance of location manager inside the scene.
- (int) zoomLevel: the default map tiles zoomLevel. Greater is the zoomLevel (0-22) better is the polygons and lines detail. If set to zero the LocationManager zoomLevel value is used.
- (int) TileBuffer: how many tiles the map will buffer around the user' location, this value has to be changed accordingly to the camera depth you want to use and to the zoomLevel value. A tile buffer value of 3 means that map will download 3 tiles in each direction from the current location (49 tiles).
- (bool) delayedFeatureLoad: toggles between creating map geometries all at once or delayed in time.
- (List<Layer>) Layers: It is **the list of map layers used in the 3D rendering**. The demo scene uses most of the important map features present in the MapZen api, for the complete list please refer at their documentation: <https://mapzen.com/documentation/vector-tiles/layers/>

Layer (Map.cs)

A layer, “buildings” for example, is a group of objects representing a specific kind of map geometry. The layer class offers a wide range of graphic customization features along with downloading and filtering options.

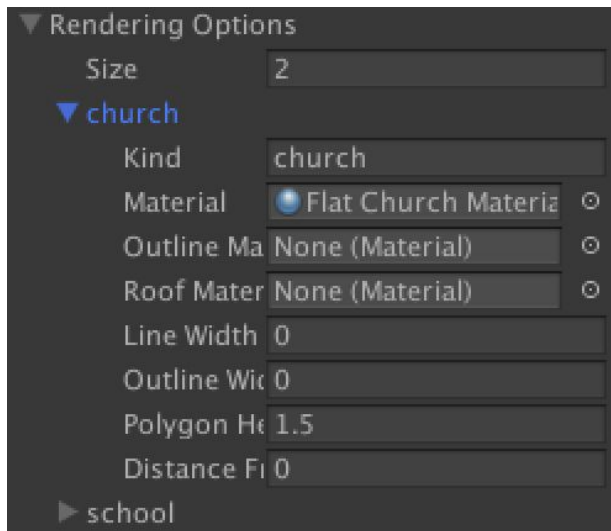


Parameters:

- (string) Name: the name showing inside the editor for the layer.
- (string) Json: the name used by the MapZen API.
- (bool) isPolygon: defines if layer's data is composed by map polygons or lines. (roads are defined as lines with a width).
- (int) defaultHeight: height of polygons (polygons only).
- (bool) useRealHeight: enables the 3D building feature (polygons only).
- (int) defaultWidth: width of the lines (lines only).
- (int) distanceFromFloor: the default y value of layer's objects.
- (Material) defaultMaterial: the material used to render layer's objects.
- (Material) defaultOutline: the material used to render the lines outline (lines only).
- (Material) defaultRoof: the material used to render building roofs (polygons only).
- (bool) Start inactive: the layer is downloaded and created but set inactive (not visible).
- (bool) Disabled: the layer is not downloaded at all, so it is not created.
- (RenderingOptions) renderingOptions: overrides customization for special map elements.
- (string[]) useOnly: use this list to select to render only specified kinds of objects. A list with a size of 0 means that every kind is rendered.

RenderingOption (Map.cs)

Layers contains object of various kind that you may want to render differently from the default style. If a rendering option is added the object will use the values specified in it instead of defaults. In the demo scene churches and schools have a different material and every road kind has a different width. While doing this kind of customization is strongly recommended to look at: <https://mapzen.com/documentation/vector-tiles/layers/> for a full list of map “kinds”.



Parameters:

- (string) Kind: the kind used by the MapZen API.
- (Material) material: the material used to render objects of that kind.
- (int) polygonHeight: height of polygons (polygons only).
- (int) lineWidth: width of the lines (lines only).
- (int) distanceFromFloor: the default y value.

Avatar

MoveAvatar.cs, TwoXOrbitGesture.cs

As said before the Avatar group is just for demo purposes.

The game object hierarchy contains:

- Land: a plane that's used as main floor for the map.
- Clouds: a fake skybox (sphere) rotating around its center.
- Camera and light: the scene's main camera with a directional light attached.
- Robot: The 3D model used as the moving character.

The two only scripts you would reuse are MoveAvatar.cs (on the avatar gameobject) and the TwoXOrbitGesture.cs (on the main camera).

Move Avatar:

It moves the entire avatar group around the map (floor, clouds, character, camera). It really needs only a location manager instance to work, the (gameobject)AvatarFigure is just to orient the character facing the movement direction.

TwoXOrbitGesture:

This script, attached to the main camera, provides orbit and zoom camera controls; both working with mouse and touch.

Orbit: moves the camera around the character at a certain distance always facing it. Activate this gesture moving the finger (or mouse) around the character.

Zoom: changes the distance between the camera and the character, as the max distance value is reached the angle between the camera and the character changes accordingly. Activate this gesture with pinch (two fingers) or scroll wheel.

Parameters:

- (gameobject) target: the orbit target.
- (float) distance: starting distance value.
- (float) orbitSpeed: multiplier for orbit gesture.
- (float) pinchSpeed: multiplier for pinch gesture.
- (float) YMin, YMax, DistanceMin, DistanceMax: tweak these values as you prefer to adjust the distance and y ranges.

First start guide

This part is going to explain how to create a (simple) map from scratch, though it's recommended to start by making a copy of the demo scene and experimenting with it.

1. Create a new scene.
2. From the top menu GameObject select 3D Object/ GO Map, this will add a location manager and a map to your scene.
3. Play the scene, you will see buildings and roads of the central park area (New York).
4. Start customizing the map object inside the inspector as you prefer, add materials, edit sizes, add new layers, etc.

POI - Demo Scene

(GOPlaces.cs, GO4Square.cs)

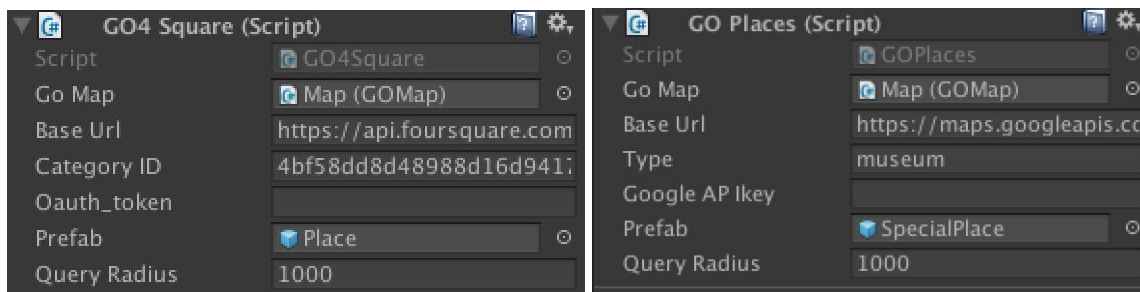
The POI Demo Scene is an example showing a map with some points of interest added. The scene uses Google Places and Foursquare APIs so in order to correctly fetch the POI data you will have to insert a Google Developer API key and/or a Foursquare oauth token.

This scene is intended for demo purposes only, the code in GOPlaces.cs and GO4Square.cs classes is short and well commented so use it to have an idea of how to fetch data and put it onto the map.

If you want to use these classes in your final product please read well the terms and condition of the Google Place and Foursquare services.

<https://developers.google.com/places/web-service/intro>

<https://developer.foursquare.com/docs/venues/search>



Parameters:

- (GOMap) : the map used in your scene.
- (string) baseUrl: url of the API.
- (string) categoryId: string identifier of the POI category in Foursquare.
(https://api.foursquare.com/v2/venues/categories&oauth_token=)
- (string) Type: string identifier of the POI category in Google Places.
- (string) oauth_token: the token you get after a login in Foursquare.
- (string) googleAPIKey: the developer key you get after the Google Developer registration.
- (GameObject) prefab: The prefab you want to use for representing the map data.
- (queryRadius): The radius of the query, where the center is your current location. When the user approaches to the previous query margin the data is loaded again.

FAQ

- **There's an error on FileHandler.cs, can't compile:** Unity is set to build on WebGL which is not supported. Build for any supported platform (iOS, Android, Standalone) and it will fix it.
- **How can I add a GameObject at some GPS position on the map?:** It's very easy, read the POI section here in the documentation and try the "POI - Demo Scene" to have an example or just use the "dropPin(double lat, double lng, GameObject go)" method in GOMap class.

The basic GPS-Vector3 conversion is just this.

```
Coordinates coordinates = new Coordinates (lat, lng, 0);
```

```
gameObject.transform.localPosition = coordinates.convertCoordinateToVector(0);
```

- **The map seems to have stopped loading data, why?:** It's probably because you have cached some tiles with a different layer content than the one you're using right now. Try disabling the cache (useCache bool on GOMap.cs) and run again, then put the cache back