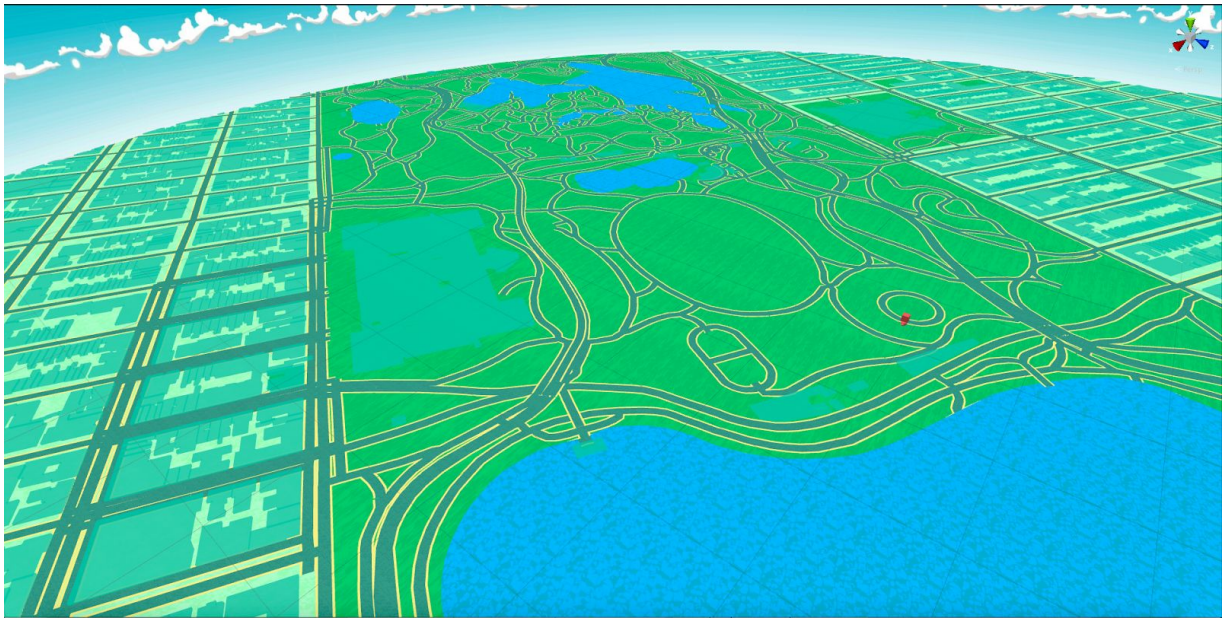
GO Map – 3D Map for AR Gaming

By Alan Grant

- *Map APIs and formats*
- *GoMap key concepts*
- *Documentation*
- *Avatar, Moving around the map*
- *First start guide*
- *POI - Demo Scene **(new!)***
- *Search Location Scene **(new!)***
- In-Editor map load
- *Combine and save meshes*
- *Environment, how to use map events (new!)*
- *FAQ*
- *When building to iOS*
- *Upgrading from previous versions*

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.



GoMap will save you a lot of time if you want to make any GPS/Map related application inside Unity 3D.
GoMap will render a full 3D map inside your scene **without the use of any raster map image**, everything you'll get is 100% generated by unity code.

Thing GoMap more of a 3D real world generator then a slippy map, even if they share most of the basic map functionalities.

Having a 3D world built in your scene allows you to develop map applications that go further the simple flat or 3D-style map visualization having, for instance, the capability to explore the world with a first person camera and add whatever game logic to it.

Map APIs and formats

GoMap uses vector map data to procedurally generate meshes in the scene. Vector map tiles contains all the information to render a map without the use of images. Vector tiles are way more efficient than raster (images) in terms of size, internet traffic, and resolution.

Starting with version 2.3 GoMap support protobuffer map files instead the simple json format. Protobuffer is a binary format that's significantly lighter and faster than json, and it's a standard among various map providers.
GoMap will decompress the downloaded data and parse it in order to build map features.

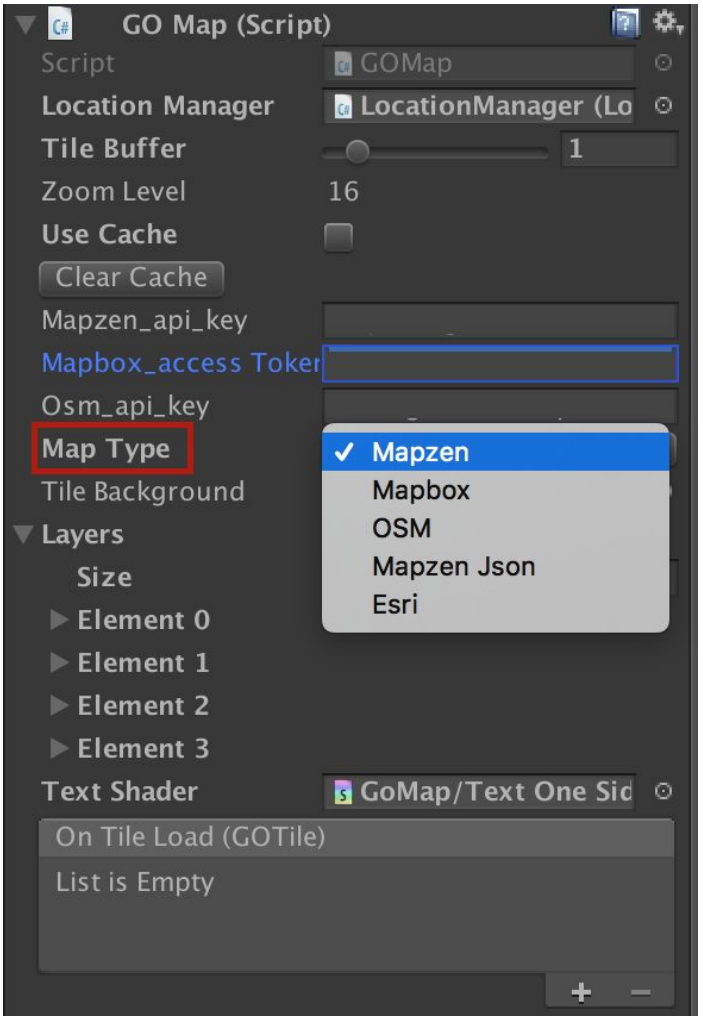
Since it's beginning GoMap as used the mapzen json data, and honestly I think it's still the best vector layer provider. From now on I suggest you to use mapzen data in the protobuffer format instead of json.

One style fits all
Just one click in the inspector and, using the same map style, you can switch map API.
Right now GoMap supports data from **Mapzen**, **Mapbox**, **Open Street Maps**.

Mapzen

Mapbox

Open Street maps



Each map APIs has a way to classify data in layers and kinds allowing you to make fine map customizations. GoMap features like styles for layers and kinds are available in all the 3 providers, with still a little more detail when using mapzen data.
Keep in mind that different map providers have different pricing plans for map requests and freemium tiers. Choose wisely =)

Here it is a basic feature compare among supported map APIs:

	Mapzen	Mapbox	OSM	Mapzen json	ESRI (beta!)
Basic map features	✔	✔	✔	✔	
Features Y sorting	✔			✔	
Street names	✔			✔	
Max zoom level	16	16	14	16	12? 14?
Uses protobuffer	✔	✔	✔		✔

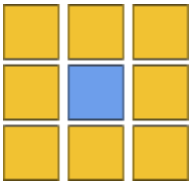
You can make your own API keys, and read the available documentation, for each service here: Mapzen, Mapbox, OSM, Esri

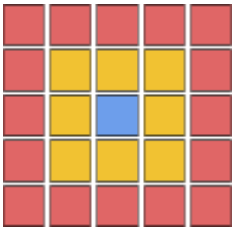
GoMap key concepts

Here are some key concepts I want to share before you keep reading this document; some basic GoMap core features that you have to keep in mind when developing your project using GoMap.

- ***Tile Buffer***
  This value is editable in the GoMap inspector and represent the number of tiles, in each direction, that are built around the user's location.
  A higher tile buffer means a wider map but more requests to do, exaggerate this value only if you are going to build the map inside your editor.


Tile Buffer 0 = 1 Tile


Tile Buffer 1= 9 Tiles


Tile Buffer 2= 25 Tiles

- ***Coordinates / units Ratio***
  GoMap uses a fixed scale for the map rendering. A real world meter always equals 1 unit in the scene. So if you measure a distance between coordinates in meters and a distance between their converted Vector3 the value would be the same.

The pro of using this scale is a better understanding of your game mechanics, allowing you to use a more comfortable scale. The con is that if you try to render very large map areas you're going to reach the max possible value that can be stored in a float.

- ***Zoom level***
  This value is editable in the Location Manager inspector, and it's the third value used to make requests to the map APIs: (X, Y, Z) -> (Tile.x, Tile.y, ZoomLevel).
  In classic map SDKs the zoomlevel value means how a tile is big in the map context, it adapts the map to the screen size on which it's displayed.
  GoMap has a different approach though, because of it's constant gps coordinates / units ratio. You can treat the zoomlevel like a parameter that means how a tile is wide, how much portion of map is included in the same tile. And of course how well defined are its features' geometries.
  A higher zoomlevel means bigger tiles, less requests given a desired map rect, and more time to process tile's data.

- ***World's origin***
  This could seem a tricky concept but I assure you that's not difficult to understand, and it will be transparent to your game development once you'll get it.
  Your scene origin point is a Vector3.zero, or (0,0,0).
  The world's origin is the GPS coordinate (lat,lng) of the center of the tile in which you open GoMap. (or the one you put in the location manager when you load inside the editor)
  That real world coordinate and your scene's origin are matched every time GoMap is loaded.

  How a map that has a different origin everytime it loads could be used?!
  That you are probably asking yourself right now don't you?

  More or less like any other map SDK you have already used. The center of the scene is not much relevant to you game mechanics. It's like the center of the map view in an iOS or Android application, you are not going to use it.

  You won't use directly Vector3 as objects transform.position but you would convert a GPS coordinate instead. In this way you'll never care about what the world's origin is.
  The conversion part is better explained in the chapter about Coordinates.cs, the script that you are going to use to convert GPS to Vector and vice versa.

Documentation

The Following documentation is based on the "New! 3D Demo" 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

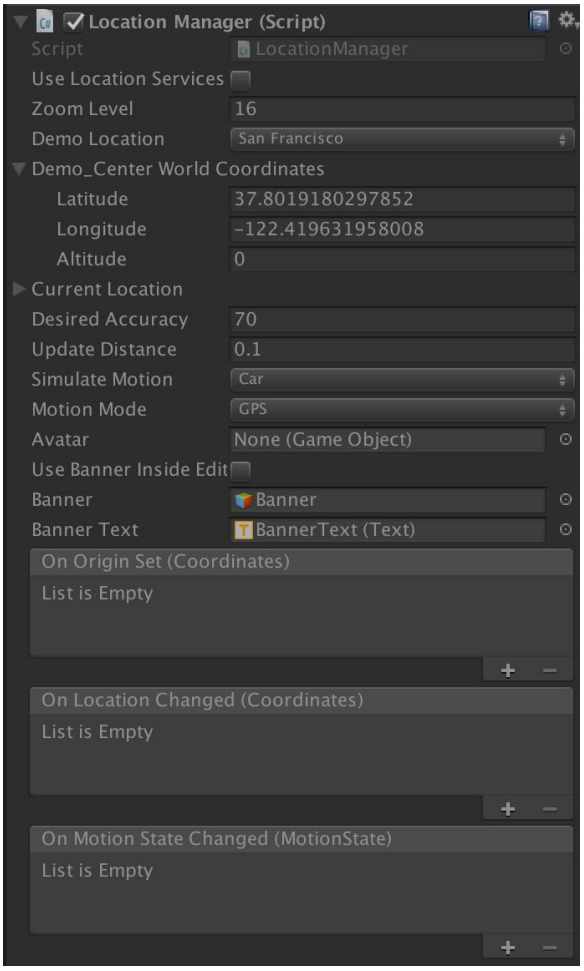
LocationManager.cs

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 motion 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: *described above
- (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.
- (enum) simulateMotion: the speed of the simulated motion when running inside editor.
- (enum)motionMode: switch between GPS motion and avatar based motion. (best described in the GPS/Avatar motion chapter)



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)

Events:
A series of events are available with locationManager if you want to use it inside your classes.

- **OnOriginSet**: Tells the exact moment when an origin is acquired. This means the first available gps location when running on a mobile phone and the set of a preset location when running in the editor. **This event is very important as it tells exactly when you can begin converting coordinates. Conversions made before this event return infinity value.**

- **OnLocationChange**: This is fired every time a new location is acquired.

- **OnMotionStateChanged**: You can use this event to trigger an animation on your character between the states of: Idle, Walk, Run. This event is fired every time the speed of your character changes significantly among the 3 states.

Coordinates.cs
Coordinates class 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 any 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);
```

And vice versa, this is how you get a GPS coordinates from a Vector3

```
Coordinates currentLocation = Coordinates.convertVectorToCoordinates
(avatar.transform.position);
```

Also from a Coordinate you can always get the center coordinate of the tile it's contained, given the current zoomlevel:

```
Coordinates tileCenter = someCoordinate.tileCenter(zoomLevel);
```

And the tile coordinates (column, row) as well:

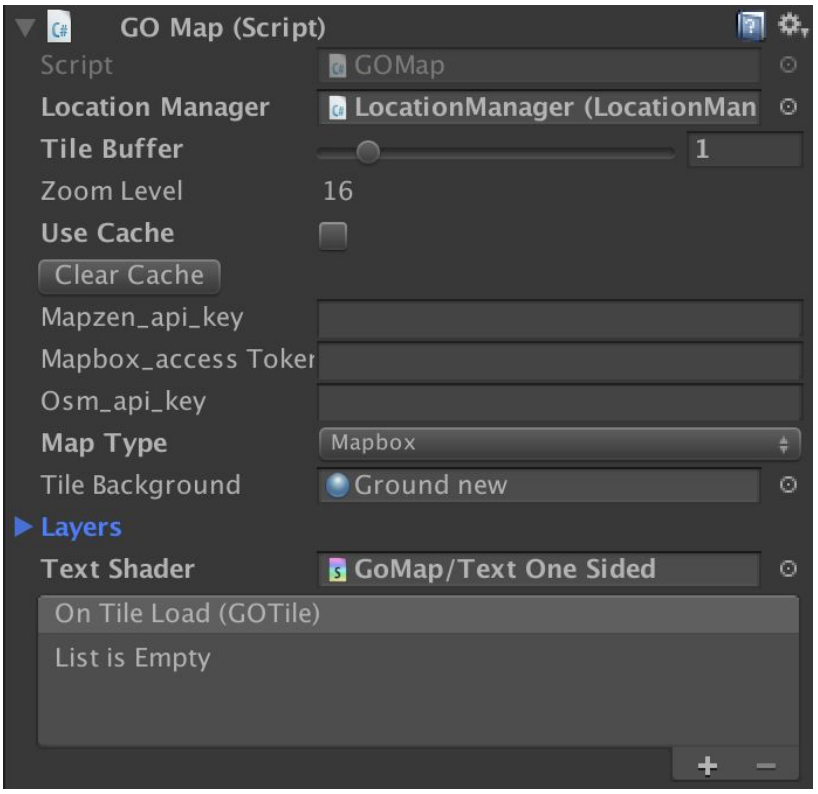
```
Vector2 tileCoords = someCoordinate.tileCoordinates(zoomLevel);
```

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.

GOMap.cs inspector is definitely the most important of the entire package, **it contains the basic map options** like which API to use, the tileBuffer value, and the style/materials for layers and

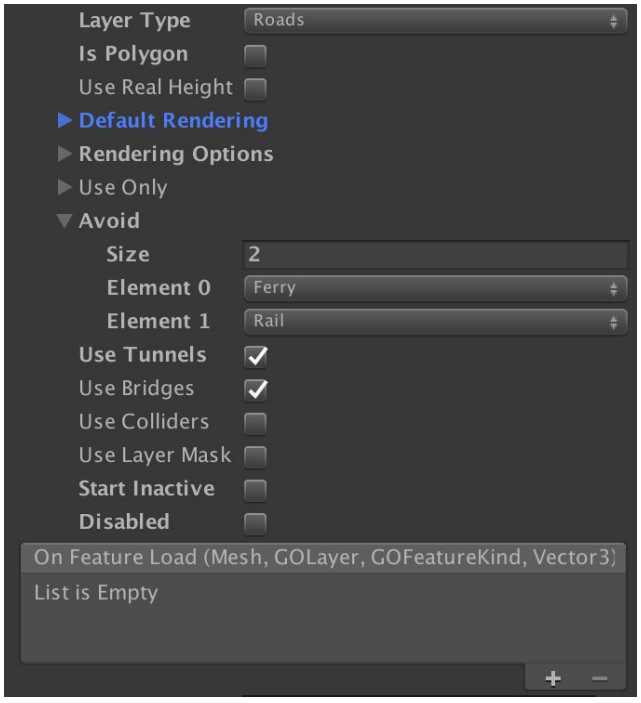


kinds.

Parameters:

- (LocationManager) locationManager: the instance of location manager inside the scene.
- (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).
- (int) zoomLevel: readonly
- (bool) useCache: when set to on GoMap will cache tiles locally, avoiding to make the same request twice.
- (button) clearCache: when pressed the whole cache folder is deleted. The default cache folder is the Application.persistentDataPath.
- (string) Mapzen,Mapbox,Osm api key: use this fields to input your own api keys.
- (enum) MapType: this selector allows you to switch among map services.
- (Material) tileBackground: if set the map will create a plane object (floor) with the specified material as a background for each tile.
- (List<GOLayer>) **Layers**: It is **the list of map layers used in the 3D rendering**.
- (bool) dynamicLoad: when set to off the map WILL NOT LOAD. Set to off only if you have already built the map with the in-editor loader.
- (Shader) textShader: The shader used to render street names.
- (UnityEvent<GOTile>) OnTileLoad: This event is fired whenever a tile is created, you can use it to add some logic after a tile creation. In the 3D demo scene it's used to target a method to spawn Ballons in the center of some tiles. (see GOEnvironment.cs)

GOLayer (Map.cs)
A layer, “roads” 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:

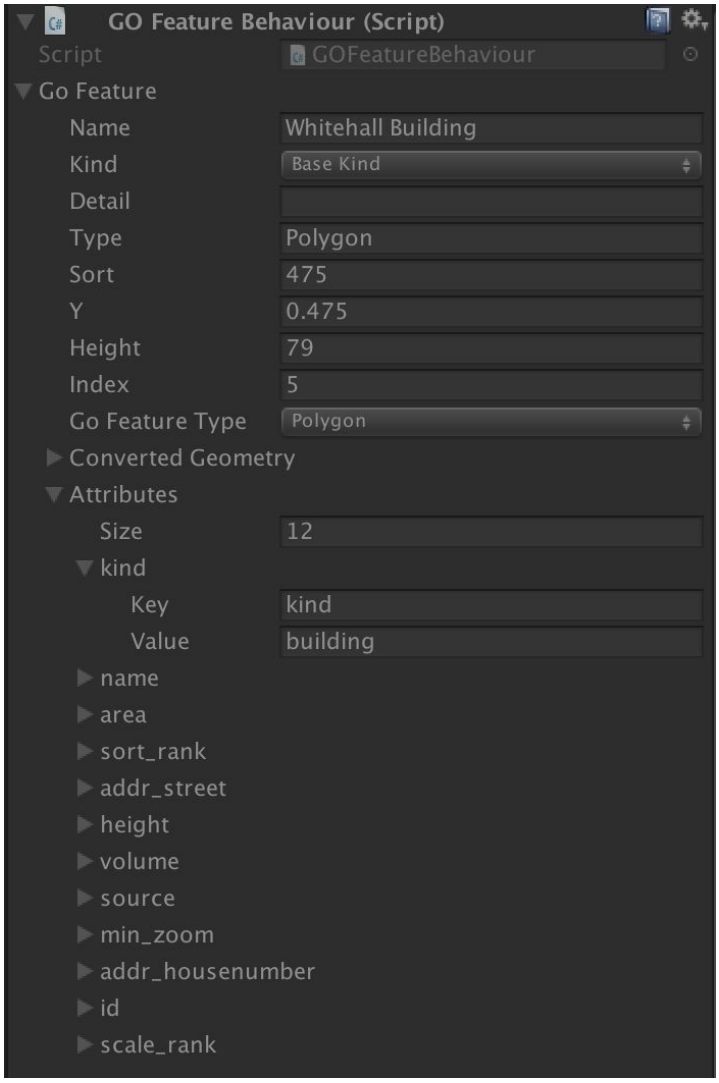
- (enum) LayerType: the enum value of the current layer, Buildings, Landuse, Water, Earth, Roads.
- (bool) isPolygon: defines if layer’s data is composed by map polygons or lines. (roads are defined as lines with a width).
- (bool) useRealHeight: enables the 3D building feature (buildings only).
- (RenderingOption) defaultRendering: contains all the default rendering options for the layer. Se the next paragraph for the “RenderingOption” class.
- (List<RenderingOption>) renderingOptions: overrides customization for special map elements.
- (GOFeatureKind[]) 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.
- (GOFeatureKind[]) avoid: use this list to select features kind not to render inside the map.
- (bool)useTunnels: flag on to show underground roads. (roads layer only).
- (bool)useBridges: flag on to show bridges. (roads layer only)
- (bool)useColliders: flag on to add colliders on objects.
- (bool)useLayerMask: flag on to add objects to unity layermasks, named after the GoMap layers. Note that you’ll have to add layer masks manually to your project in order to make this feature work.
- (bool) Start inactive: the layer is downloaded and crated but set inactive (not visible).
- (bool) Disabled: the layer is not downloaded at all, so it is not created.

GORenderingOption (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".

The rendering option used for "defaultRendering" has to always have "BaseKind" as the GOFeatureKind parameter.

Parameters:

- (GOFeatureKind) Kind: the kind to override.
- (Material) material: the material used to render objects of that kind.
- (Material) outlineMaterial: material used for the roads outline.
- (Material) roofMaterial: the material used to render rooftops.
- (List<Material>)materials: A list of materials to be used for this kind of object. One of them is chosen randomly. The random has a seed based on gps coordinates so it's always the same among the various builds/users of GoMap.
- (int) lineWidth: width of the lines (lines only).
- (int) outlineWidth: the witdh of the road outline.
- (bool) useStreetNames: when possible tells GoMap to render stretnames for this kind of features.
- (int) polygonHeight: height of polygons (polygons only).
- (int) tag: the unity tag used for this kind of objects. Note that you'll have to add tags manually to your project in order to make this feature work.



GOFeatureBehaviour.cs

Every map object that's loaded by GoMap has this component attached.
This component contains all the information retrieved from the API used to download the data and some information added by GoMap.

The Attributes section is the exact "properties" data that comes from the server. You'll notice that some APIs have a more complete set of properties per features and others have fewer.

In a decreasing order the more complete properties list are contained in data coming from:

- Mapzen
- Mapbox
- OSM
- ESRI

Avatar, moving around the map:

MoveAvatar.cs, GOOrbit

You'll find the Avatar group in every GoMap demo scene and it contains the character, the main camera and the basic motion components.
There are two ways to move your character in the GoMap world, one it's with GPS and the other is using some custom keyboard controls and an avatar prefab. Depending on which of the above you chose you have to configure GoMap accordingly to make it build the world at runtime.

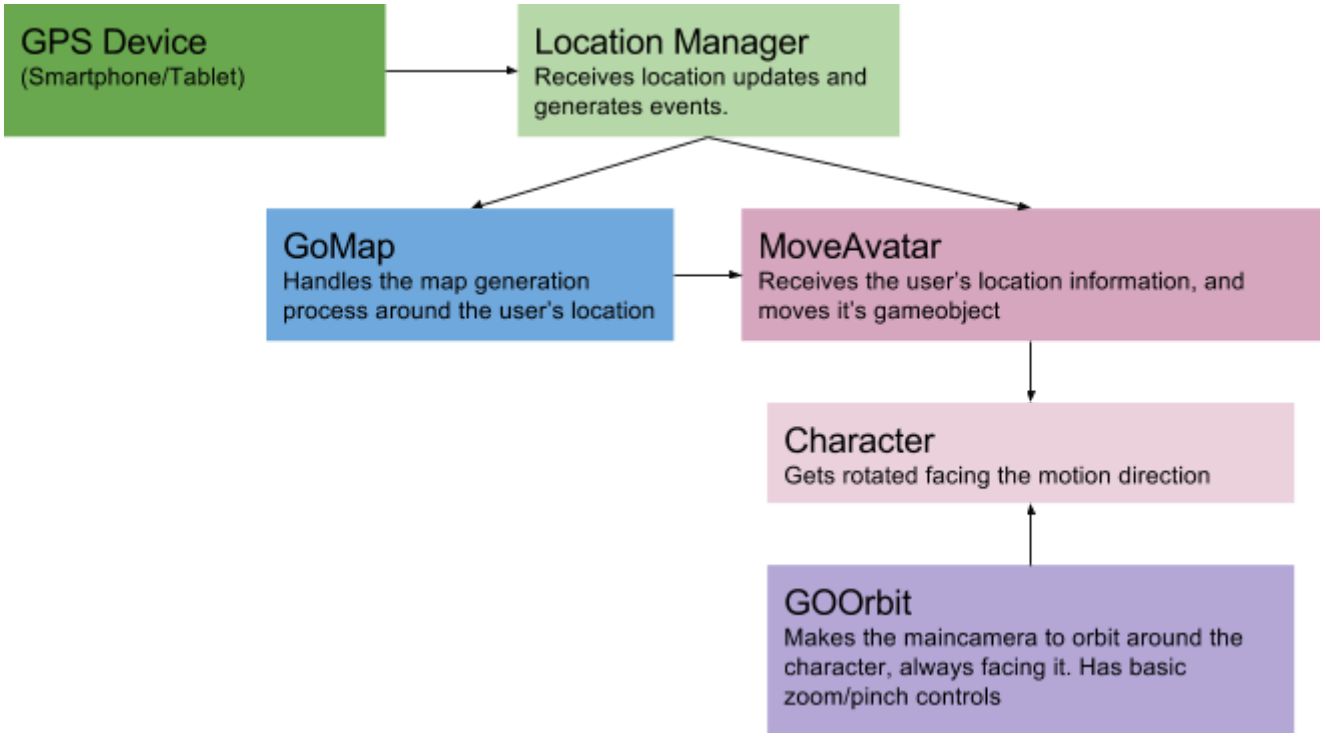
In the MoveAvatar script (attached to avatar) you'll find a very useful Event and property to **control your character animation state** accordingly to the speed of motion. And this states consider the smooth lerp motion.
Available states are:

- Idle
- Walk
- Run

GPS motion mode

To enable this mode you have to select "GPS" under "motionMode" in the LocationManager inspector. In this way the location manager gets the location coordinates from the GPS device of your smartphone, converts them into unity vectors, and tells GoMap to build the terrain tiles.
Your avatar group has to have the "MoveAvatar.cs" component attached to it and linked with LocationManager and your character prefab.
Basically when the app is running it works like this:

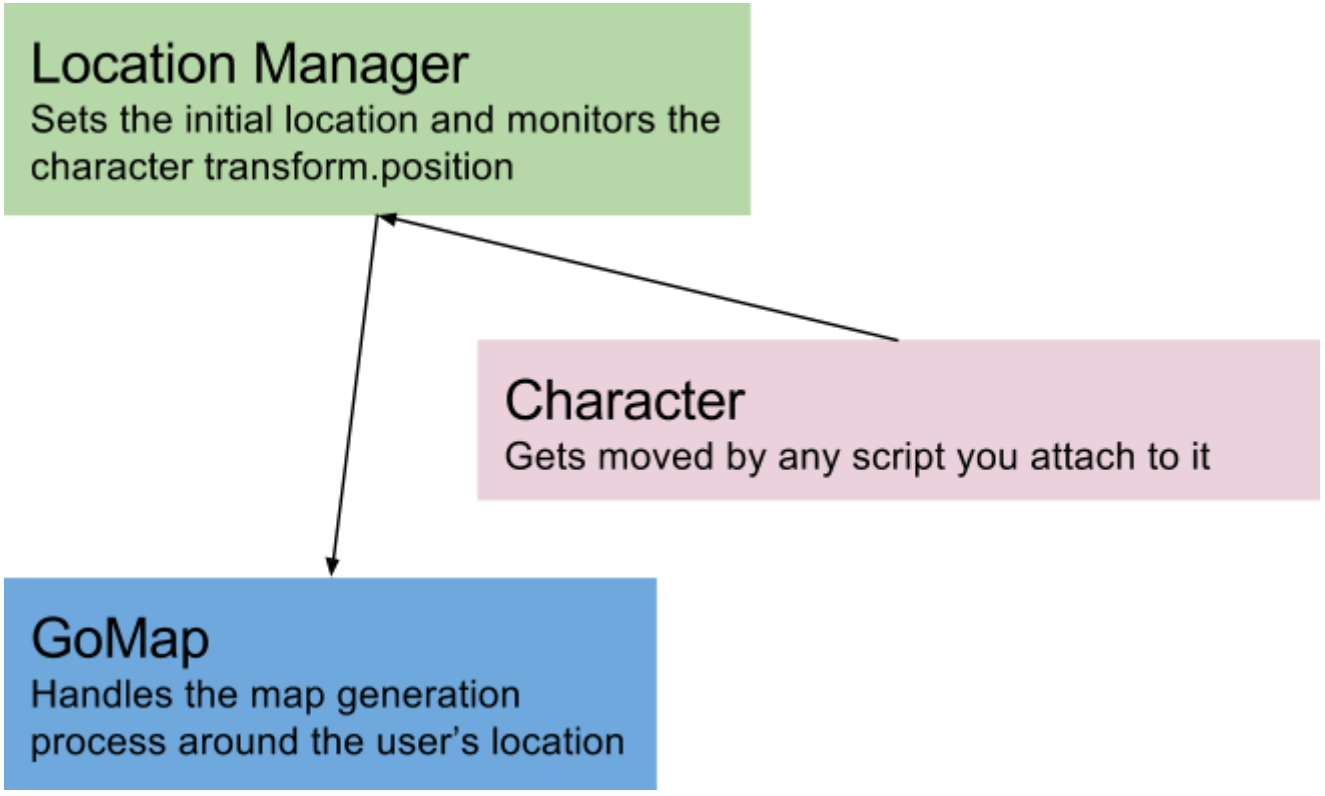
1. Location manager receive the location updates from unity GPS apis and sends events to the registered classes: GOMap and MoveAvatar.
2. GOMap builds the nearby map.
3. MoveAvatar receives the new user's currentLocation and converts it to Vector3.
4. The linked character is rotated accordingly to the motion direction.
5. The main camera keeps orbiting the character on the xz plane.



Avatar motion mode
To enable this mode you have to select “AVATAR” under “motionMode” in the LocationManager inspector and link your avatar gameobject to it. In this way the location manager will constantly monitor the avatar’s transform.position, converting it to gps coordinates system. The converted gps location it’s used to generate the currentLocation events and build the terrain tiles.
Your avatar group has to have some script to move around the world, independently from gps updates.
Note that you have to select an initial location in the LocationManager from the list or input some coordinates and select “custom”.

Basically when the app is running it works like this:

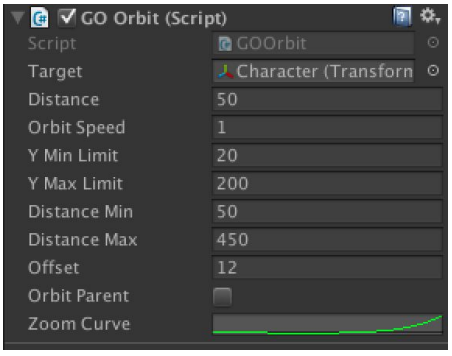
1. Location manager sets the world origin as the chosen demo location then starts to monitor your avatar transform.position.
2. GOMap builds the nearby map.
3. Move your avatar with the controls you want, and the location manager will be aware of that, loading a new map section when your character is about to reach the edge of the map.



GOOrbit:
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.
- zoomCurve: This is only on the GOOrbit script, and it's made to design your own zoom camera movement.



First start guide

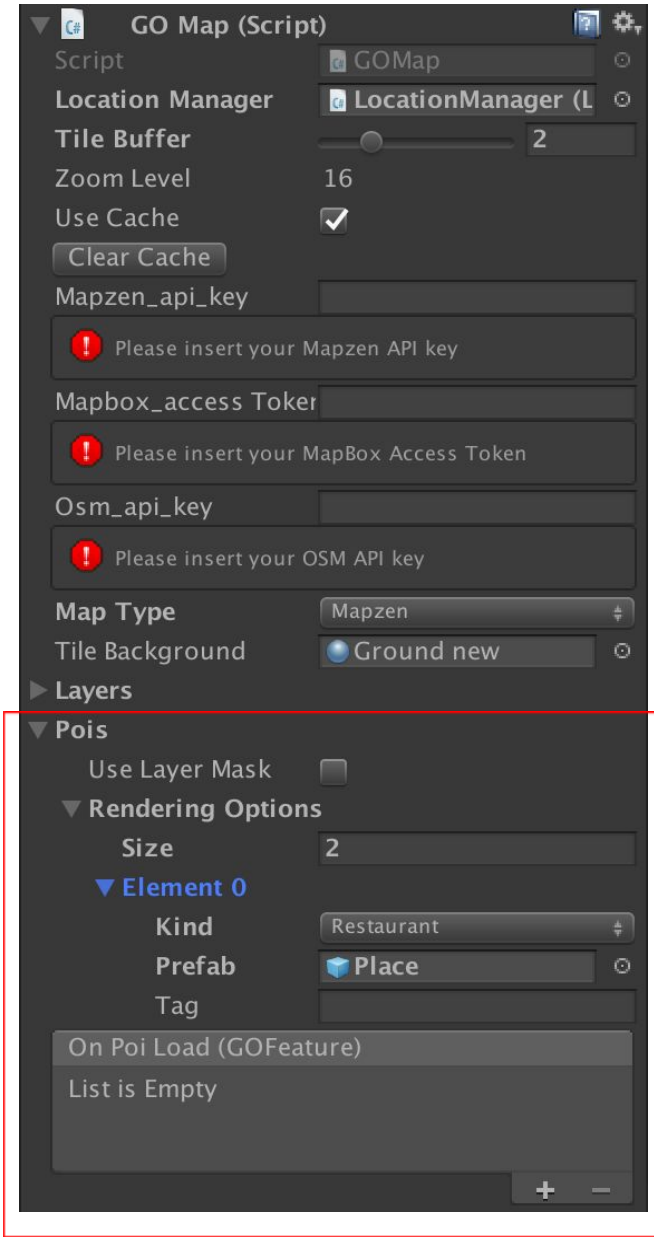
This part is going to explain how to create a (simple) map from scratch, tough it's recommended to start by **making a copy of the demo scene** and experimenting with it.

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

GoMap now features “native” POIs (Points of interest) from Open Street Maps using the Mapzen, Mapbox or OSM apis.
You’ll notice a new section in the GOMap.cs component about POIs in which you can add a prefab for a POI kind (for example restaurants) and it will be drop automatically on the map wherever a Restaurant is present in the downloaded map data.
This is a huge improvement for your location based games!

It’s never been so easy to add various kind of prefabs to the map and without any additional request.



The demo scene “POI - Demo.unity” is preset with 2 different prefabs for the Restaurants and Cafes of different color.
You’ll notice that if you walk near them they’ll start to animate.

If you want to add POIs of other kinds to your map just increase the **Size** of the **Rendering Options** list and select a different **Kind** in the new object.

For each POI kind it’s possible to select a **prefab** that will copied and dropped on the map, a **Tag** (Unity Tag) and some Listener for the **OnPoiLoad** event.

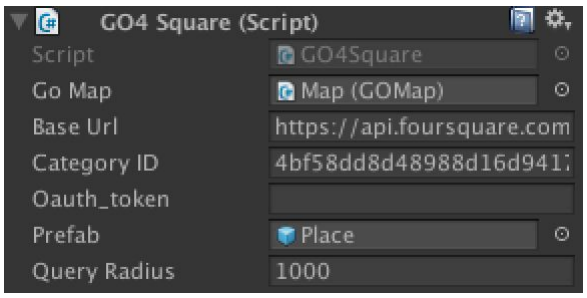
Start Inactive and Disabled are meant for the whole POI Layer as the other classic GoMap Layers.

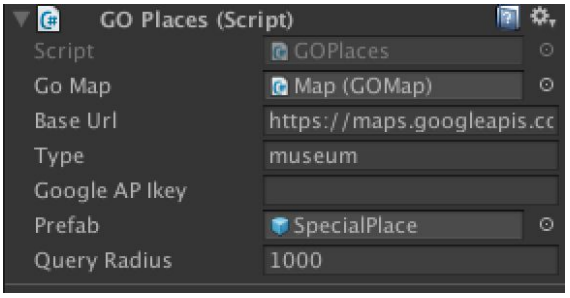
You can still find the GOPlaces.cs and GO4Square.cs components and prefabs in the GoMap bundle but not in this demo scene anymore.

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 other kinds of 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.

In order to correctly fetch the POI data using Google Places or Foursquare you will have to insert a Google Developer API key and/or a Foursquare oauth token.

https://developers.google.com/places/web-service/intro
https://developer.foursquare.com/docs/venues/search

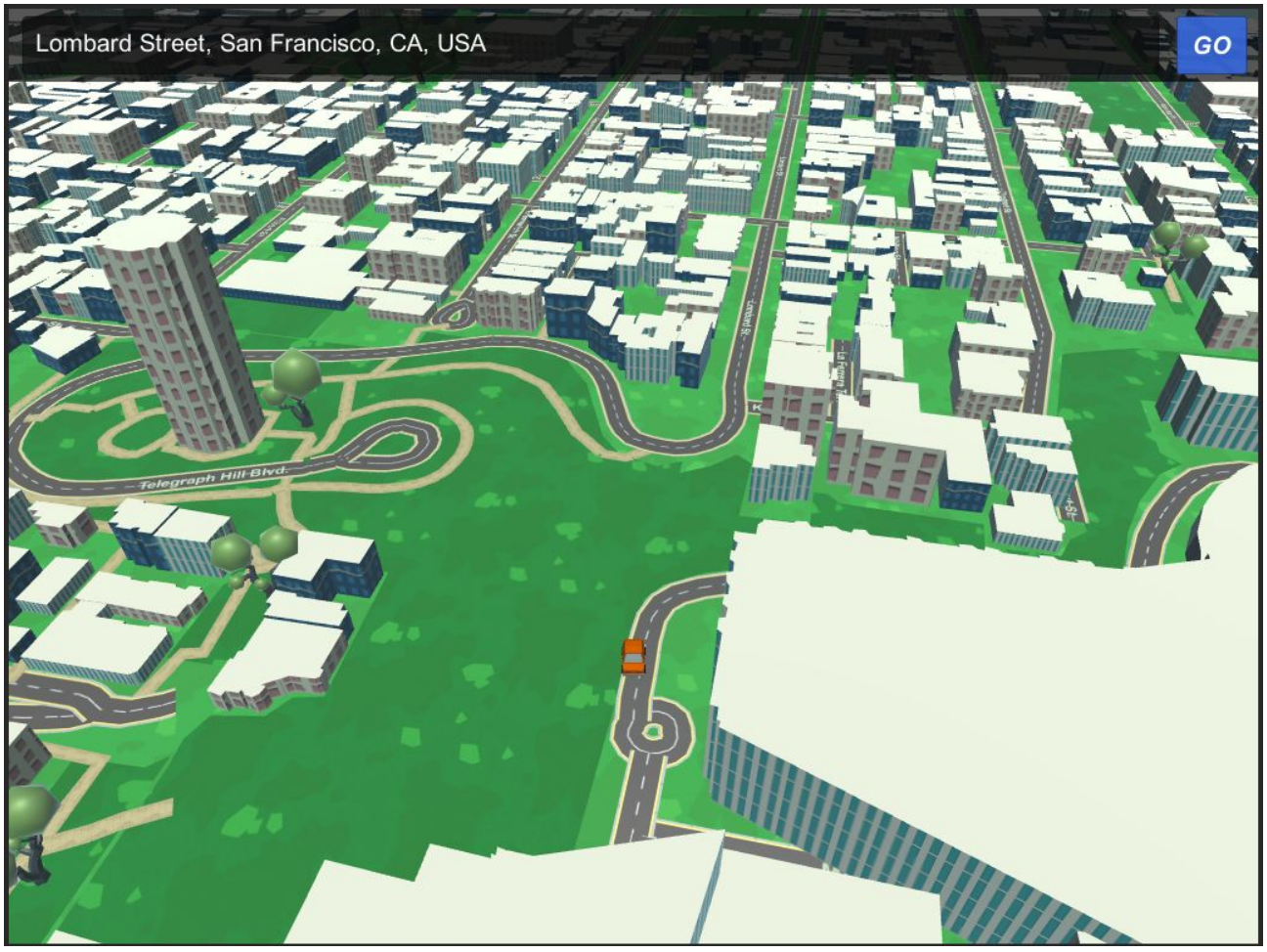




Parameters:

- (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.

Search Location - Demo Scene
(GOToLocationDemo.cs, GOCarAvatar.cs)



With this scene I wanted to show another possible use of GOMap that's not GPS based.
This scene shows a car Avatar that you can move across the map using only a tap (click) gesture and a Search Bar in which you can search for places from all the world or input some coordinates in the lat,lng format.

Of course you'll notice the new city theme too =)

But let's see what's under the hood:

The Search Bar is a Canvas in which some UI objects are placed together to make an adapting input field with a "dropdown" menu. I suggest you to take a look to the Canvas hierarchy to better understand how it's (few) components work.
The real job is done inside the **GOToLocationDemo**.cs component that uses the Mapzen search API to reverse geocode addresses into GPS coordinates. It works like any other map search box, for example if you input "Telegraph Hill" and press GO (or enter key) the API will answer with just an address for "Telegraph Hill, San Francisco, CA, USA" and it's relative GPS position.
The component will manage to create a new tab under the search bar that will teleport to that location when clicked.

The job of teleporting you to another point of the world and rebuild the map is done with this simple command:

locationManager.SetLocation (location.coordinates);

Where location.coordinates is a classic GOMap style Coordinates.cs object. You can use this line of code whenever you want to change map location by teleporting.

An important thing to say it's about how the **LocationManager** is set up. The "MotionMode" is set to Avatar and the "Avatar" field is linked with the GOCar Prefab.
This is a classic setup to use the position of a prefab wandering on the map instead of GPS to build the map around your character.
Basically in this way the Locationmanager will monitor the position of your character (car in this case) and accordingly destroy/build portion of map.

GOCarAvatar.cs is a (very) simple gesture to move a car on the map by your finger. It works fine with the already known GOOrbit.cs allowing you to "steer".
Feel free to replace it with a real car prefab that uses colliders or whatever you think it's best for your app/demo =)

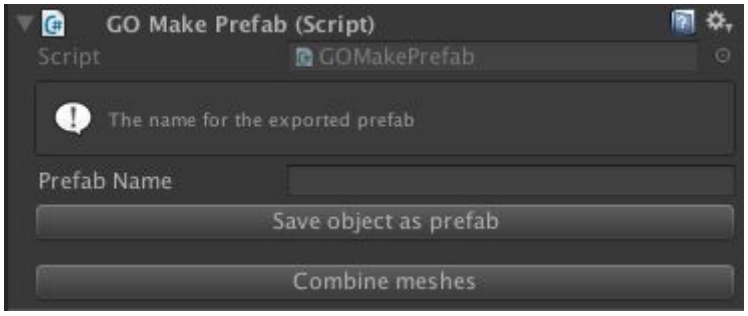
In-Editor map load
(GOMapEditor.cs)

With the version 2.0 you can preload the map inside your scene and not use it live with gps. This feature allows you to create awesome real world cities to set your game in.
To load a map in your scene just add this component to an already attached GOMap script and press the “Load Map in Editor” button. To destroy it and create a new one press the Destroy button.
Once the map is loaded you can edit it, save it (as a scene), and place your game in it.

Tip: After you have loaded the map remember to remove the GOMap script or **unflag the “dynamicLoad”** to prevent it reloading the map.

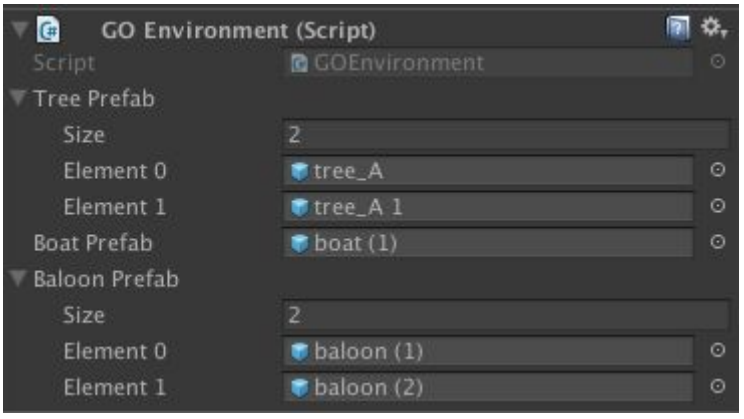
Combine and save meshes
(GOMakePrefab.cs)

With the version 2.0 you can combine and save meshes (like buildings) as prefabs. This could be very useful if you want to save some real building and use it in other contexts.
If you want to save a mesh that is already loaded attach the GOMakePrefab script to its GameObject **give it a name** and press the save button.
If you want to save a group of meshes (like a complex building) attach the GOMakePrefab script to a container GameObject insert a name and press save.
A new directory is automatically created when saving the first mesh at:
GO Map - 3D Map For AR Gaming/Exported

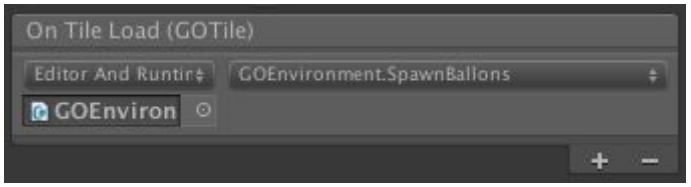


Environment, how to use map events
(GOLayer , GOMap.cs , GOEnvironment.cs)

GOMap generates two kinds of events that you can use to add some other logic to the map creation. The GOEnvironment script is just a way to use them, use it as inspiration for build other stuff or do your custom game logic.



OnTileLoad
This event is fired every time a tile is created. A GOTile parameter is passed in input so you can know everything about the area that is being created.
In the GOEnvironment example this event is used to spawn a Balloon in some of the tile centers at a random height.



OnFeatureLoad
This event is available for every layer separately and it is fired every time a new map feature is created. A Feature is a map geometry with a mesh representing it.
The parameter passed are:

- The mesh created
- the GOLayer object that fired the event
- the "kind" of the feature (see this for list of object kinds)
- the center of the mesh

In the GOEnvironment example this event is used to add trees in every park or garden and to place boats in some of the water areas.

FAQ

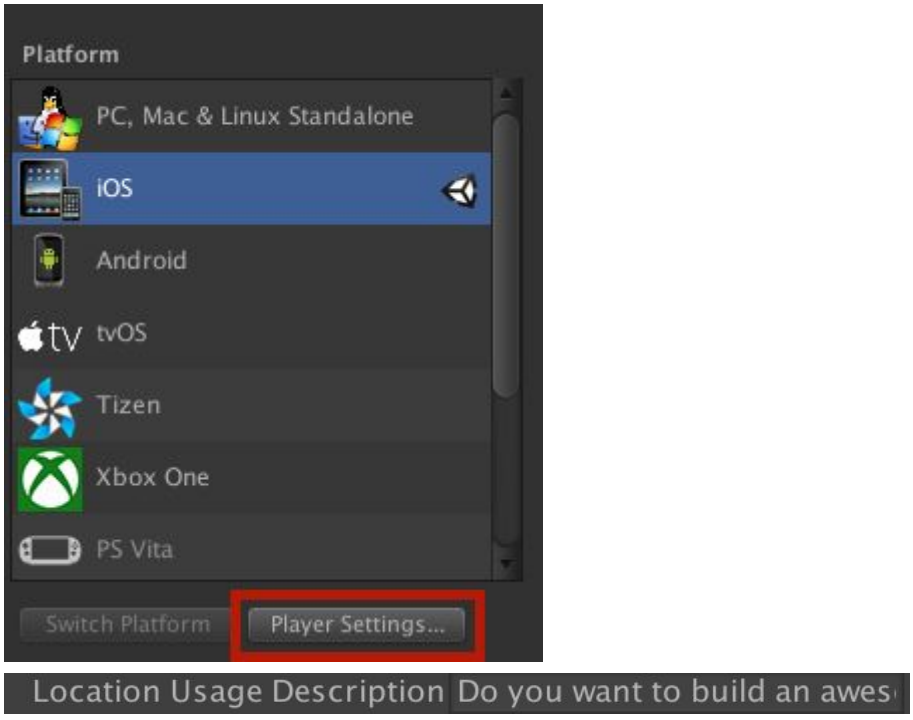
- **There's an error on FileHandler.cs, can't compile**: Unity is set to build on WebPlayer which is not supported. Build for any supported platform (iOS, Android, Standalone) and it will fix it.
- **How can I add a GameObject at some GPS position on the map?**: It's very easy, read the POI section here 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

When building to iOS
When building to any iOS device remember to add the localization string message that will be copied inside the info.plist file of your app and prompted to users while asked for localization usage permission.
If you don't add this message the app won't ask your users for the localization usage and the map won't load.

You can find it under the iOS Player Settings, at the field "Location Usage Description".



Upgrading from previous versions

Unity code updater sometimes messes things up, so I'd like to give you some guidelines that makes your update procedure safer and faster.

First of all something about GoMap code: I don't discourage to edit some GoMap classes to best fit your project or to add new features, but you have to understand that this changes will be overwritten by any new version of GoMap. If you have changed one of the GoMap classes you and need to update to a new GoMap version you have to merge your changes manually to the new files.
More than editing the GoMap files you maybe should override functions and make extensions, try to change the GoMap code the less possible if you are going to need updates.

The same is true in the case of demo scenes; you shouldn't edit them in the same GoMap file but make a copy and edit it.

If you have built your game **around GoMap**, without editing the core classes, you can **sleep safe**. The GPS/Unity conversion system won't change so all your game logic will still work.

This is the smooth upgrading procedure:

1. Backup your project, you should do this independently by the updates of GoMap.
2. Erase the GoMap - 3D Map for AR gaming folder from the unity project manager.
3. Do the unity update of GoMap from the built in asset store window.
4. Open your custom scenes and see if the GoMap objects are broken or settings are somehow different and edit them.
5. If something goes wrong, you have your backup and you can still restore it.
6. For any issue don’t hesitate to contact me at alangrant.unity@gmail.com