Chapitre I

Objets Leaflet

I.1- Création de cartes avec leaflet

■ Historique

- Prédécesseurs « open source »: MapServer, GeoServer, et OpenLayers.

■ Etape 1 : Création de l'objet carte (map) centrée et avec niveau de zoom à 18 (maximum)

```
var map = L.map('maDiv', {
center: [48.858376, 2.294442],
zoom: 18
});
```

- Ou aussi (en plus court):

```
var map = L.map('maDiv').setView([48.858376, 2.294442],18);
```

- Conseil : Il est de bon ton de toujours centrer la carte avec les coordonnées [latitude, longitude] ainsi que de donner la valeur entière (de 1 à 18 en général) du niveau de zoom initial.
- A ce stade là, on dispose juste d'un objet map leaflet capable d'accueillir notre carte pour pouvoir l'afficher et la manipuler. A l'affichage, on a bien un cadre qui s'affiche mais pas de carte encore.

Création de cartes avec leaflet

■ Etape 2 : Ajout d'une carte que l'on appelle « tile layer » (couche de tuiles)

- Une couche de tuiles peut être considérée comme la carte de base sur laquelle on va travailler. Cela inclut l'imagerie qu'on peut rajouter (points, polygones, cercles, etc..). Une couche de tuiles est un service fourni par un serveur de tuiles. Celui-ci utilise en général un découpage en images de 256x256 pixels que l'on appelle des tuiles. Une carte est donc un ensemble de tuilles fournies par un serveur.
- On récupère donc ces images à partir du serveur en se basant sur la localisation et le zoom indiqués et en indiquant une URL du type /z/x/y.png (z pour zoom, x pour latitude, y pour longitude). Seules les tuiles nécessaires à couvrir la zone sont téléchargées. Lorsque l'on zoom ou que l'on se déplace, d'autres tuiles sont alors récupérées.
- Une couche de tuiles requiert au minimum l'URL vers le serveur de tuiles. Dans ce cours, nous utiliserons OpenStreetMap comme serveur de tuiles (notre serveur de cartes en quelque sorte).
- Code de l'URL pour le serveur de tuiles OpenStreetMap :

```
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);
/* s pour standard */
```

- Via OpenStreetMap, il existe d'autres types de cartes (plus spécifiques) que les cartes "Standard" comme "Cycle Map", "Transport Map", ou "MapQuest Open"

Création de cartes avec leaflet

■ Création d'une basemap (carte de base) simple

```
<html>
<head><title>Leaflet.js: les bases</title>
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.3/leaflet.css" />
</head>
<body>
<script src="http://cdn.leafletjs.com/leaflet-
0.7.3/leaflet.js"></script>
<div id="maDiv" style="width: 800px; height: 600px"></div>
<script>
var map = L.map('maDiv',
{
center: [48.858376, 2.294442],
zoom: 18
});
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);
</script>
</body>
</html>
```

- Rendu

I.2- Les fournisseurs (serveurs) de cartes

- Il en existe de nombreux. Certains exigent une inscription comme Google ou Bing.
- Nous présentons succinctement les 2 serveurs de tuiles Thunderforest et Stamen. Thunderforest fournit des tuiles d'extension à OpenStreetMap alors que Stamen fournit des tuiles disons plus artistiques.

Les fournisseurs (serveurs) de cartes

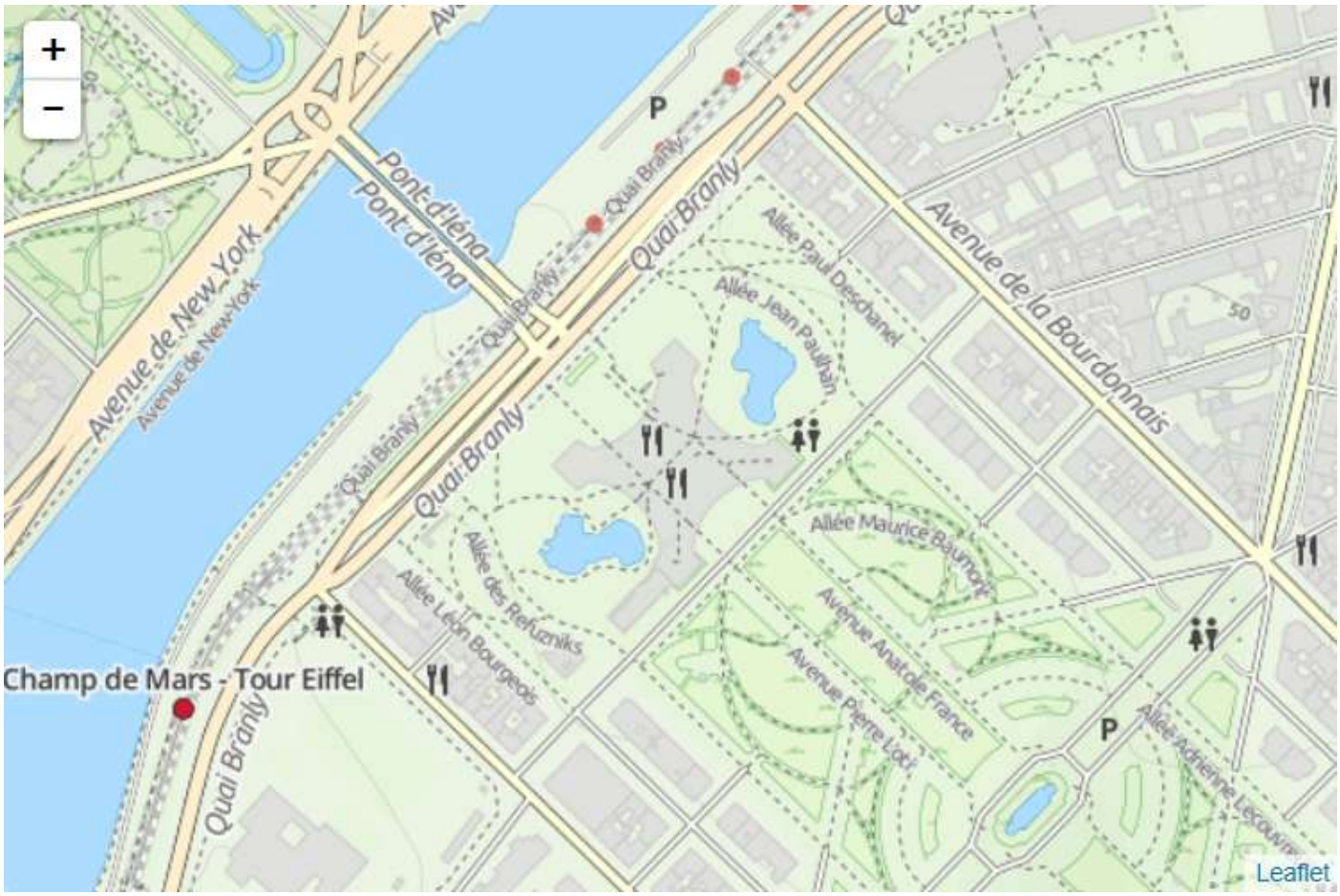
■ Thunderforest fournit 5 services de tuiles :

- OpenCycleMap
- Transport
- Landscape
- Outdoors
- Atlas (en cours de développement)

- Exemple avec le service de tuiles Outdoors

```
<body>
<div id="maDiv" style="width: 600px; height: 400px"></div>
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],16);
var couche = new
L.TileLayer('http://{s}.tile.thunderforest.com/outdoors/{z}/{x}/{y}.png');
map.addLayer(couche);
</script>
</body>
```

- A la place de landscape, on peut donc aussi utiliser cycle, transport, ou outdoors.



Les fournisseurs (serveurs) de cartes

■ Stamen fournit 6 couches de tuiles différentes dont :

- Terrain (disponible aux USA seulement)
- Watercolor
- Toner

- Remarque : Nécessité d'inclure une extension (pluggin)
- Exemple avec le service de tuiles Watercolor

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>Leaflet.js</title>
<meta charset="utf-8" />
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.3/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-0.7.3/leaflet.js"></script>
<script type="text/javascript"
src="http://maps.stamen.com/js/tile.stamen.js?v1.2.4"></script>
</head>
<body>
<div id="maDiv" style="width: 800px; height: 600px"></div>
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],15);
var couche = new L.StamenTileLayer("watercolor");
map.addLayer(couche);
</script>
</body>
</html>
```

Les fournisseurs (serveurs) de cartes

- Les couches Starmen ne sont pas censées être utilisées comme carte de base mais elles apportent un certain travail artistique dirons-nous.

I.3- Le service WMS Web Mapping Service

■ Ajout d'une couche de tuiles WMS Web Mapping Service

- WMS est un service permettant de transférer des images de cartes à travers le Web via le protocole HTTP. C'est une spécifications de l'Open Geospatial Consortium (OGC). Un exemple, parmi d'autres, de couche WMS à ajouter à une carte est l' Imagery Topo issue de United States Geological Survey (USGS).
- Le code suivant pemet d'ajouter une couche WMS à une carte leaflet :

```
<script>
var map = L.map('map', {
  center: [48.858376, 2.294442],
  zoom: 6
});
var usgs =
L.tileLayer.wms("http://basemap.nationalmap.gov/arcgis/services/USGSTopo/MapSer
ver/WMSServer", {
  layers:'0',
  format: 'image/png',
  transparent: true,
  attribution: "USGS"
}).addTo(map);
</script>
```

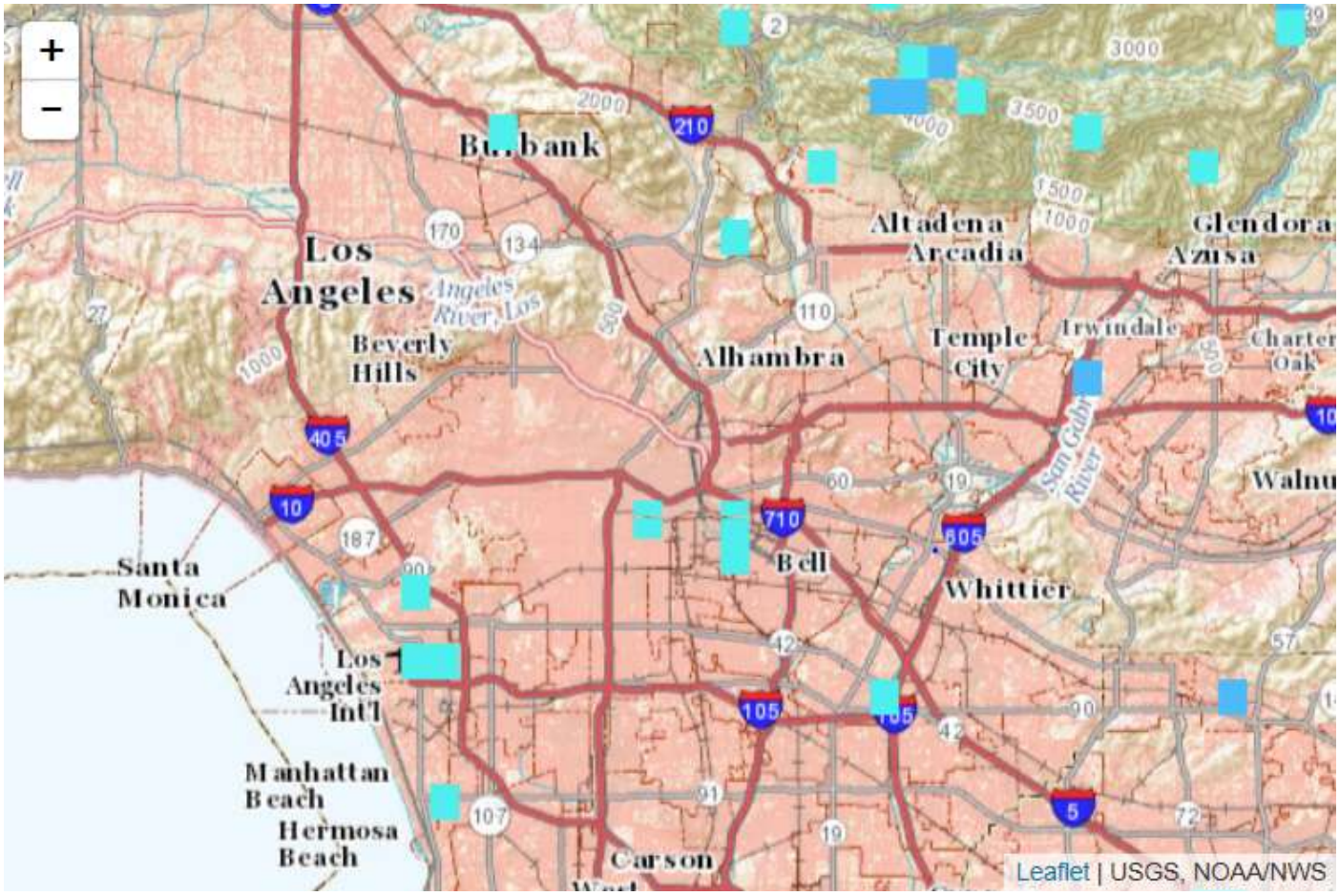
I.4- Multiples couches

- Par exemple, on va ajouter une couche WMS représentant la mosaïque (en direct live !) du radar météo de la National Weather Service (NWS) au dessus de la couche d'images satellite UGCS.

```
<script>
var map = L.map('map', {
  center: [34.052234, -118.243685], // Los Angeles
  zoom: 10
});
var usgs =
L.tileLayer.wms("http://basemap.nationalmap.gov/arcgis/services/USGSTopo/MapSer
ver/WMSServer", {
  layers:'0',
  format: 'image/png',
  transparent: true,
  attribution: "USGS"
}).addTo(map);
var nexrad =
L.tileLayer.wms("http://nowcoast.noaa.gov/wms/com.esri.wms.Esrimap/obs", {
  layers: 'RAS_RIDGE_NEXRAD',
  format: 'image/png',
  transparent: true,
  attribution: "NOAA/NWS"
}).addTo(map);
</script>
```

Multiples couches

- Remarque : Si on met la propriété (de la seconde couche, ici, la couche météo) transparent à false, les 2 cartes ne se superposent plus : seule la couche météo devient visible.



- Les couches WMS ne servent pas seulement de couche de base à une carte, elles peuvent être utilisées comme données additionnelles.

I.5- Multiples couches de base

- Exemple de code

```
<script src="http://cdn.leafletjs.com/leaflet-0.7.3/leaflet.js"></script>
</head>
<body>
<div id="map" style="width: 600px; height: 400px"></div>
<script>
var coucheRoute = new
L.TileLayer('http://otile{s}.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png', {
  maxZoom: 18,
  subdomains: ['1', '2', '3', '4'],
  attribution: 'Tiles Courtesy of <a href="http://www.mapquest.com/"
target="_blank">MapQuest</a>. Map data (c) <a
href="http://www.openstreetmap.org/" target="_blank">OpenStreetMap</a>
contributors, CC-BY-SA.'
});
var coucheSatellite = new
L.TileLayer('http://otile{s}.mqcdn.com/tiles/1.0.0/sat/{z}/{x}/{y}.png', {
  maxZoom: 18,
  subdomains: ['1', '2', '3', '4'],
  attribution: 'Tiles Courtesy of <a href="http://www.mapquest.com/"
target="_blank">MapQuest</a>.'
});
var coucheRelief = new
L.TileLayer('http://{s}.tile.thunderforest.com/landscape/{z}/{x}/{y}.png', {
  attribution: 'Tiles Courtesy of <a href="http://www.thunderforest.com/"
target="_blank">Thunderforest</a>.'
});
var coucheTransport = new
L.TileLayer('http://a.tile.thunderforest.com/transport/{z}/{x}/{y}.png', {
  attribution: 'Tiles Courtesy of <a href="http://www.thunderforest.com/"
target="_blank">Thunderforest</a>.'
});

var map = new L.Map('map', {
  center: new L.LatLng(48.858376, 2.294442),
  zoom: 4,
  layers: [
    coucheSatellite // Couche par défaut
  ]
});

L.control.layers({'Infrastructure Routière': coucheRoute, 'Vue Satellite':
coucheSatellite, 'Relief': coucheRelief, 'Transport':
coucheTransport},null,{collapsed:false}).addTo(map);
</script>
```

Multiples couches de base

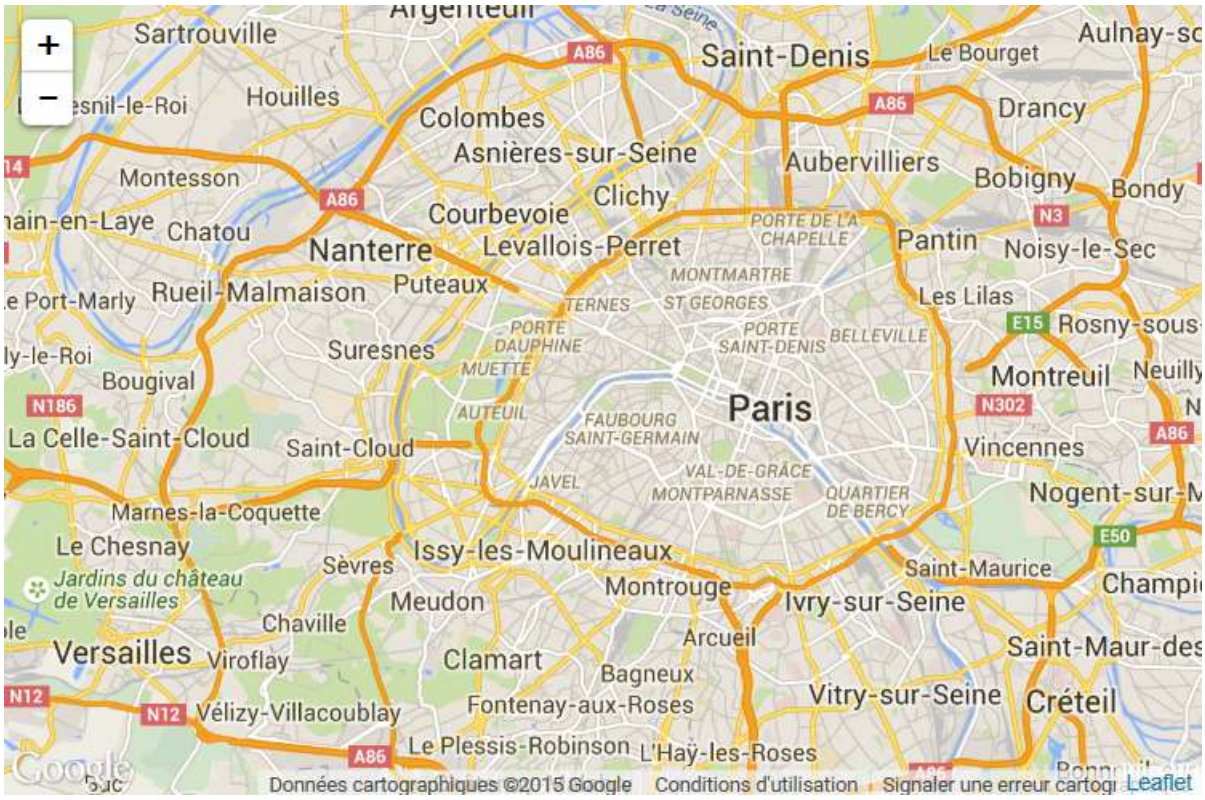
- Rendu Satellite (par défaut)

- Rendu Relief

I.6- Leaflet et cartes Google

- Intégrer une simple carte Google (nécessité d'un plugin et de l'API Google)
- Exemple 1: Carte de type ROADMAP

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>Leaflet.js</title>
<meta charset="utf-8" />
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.3/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-0.7.3/leaflet.js"></script>
<script src="http://maps.google.com/maps/api/js?v=3.2&sensor=false"></script>
<script src="http://matchingnotes.com/javascripts/leaflet-google.js"></script>
</head>
<body>
<div id="maDiv" style="width: 600px; height: 400px"></div>
<script>
var googleLayerROADMAP = new L.Google('ROADMAP');
var map = new L.Map('maDiv', {
      center: new L.LatLng(48.858376, 2.294442),
      zoom: 11,
      layers: [googleLayerROADMAP]
});
</script>
</body>
</html>
```



Leaflet et cartes Google

- Exemple 2: Plusieurs types de cartes Google (nécessité d'un plugin et de l'API Google)

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>Leaflet.js</title>
<meta charset="utf-8" />
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.3/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-0.7.3/leaflet.js"></script>
<script src="http://maps.google.com/maps/api/js?v=3.2&sensor=false"></script>
<script src="http://matchingnotes.com/javascripts/leaflet-google.js"></script>
</head>
<body>
<div id="maDiv" style="width: 600px; height: 400px"></div>
<script>
/* plugin permettant d'utiliser Google Maps comme serveur de tuiles pour
Leaflet */
var googleLayerROADMAP = new L.Google('ROADMAP');
var googleLayerSATELLITE = new L.Google('SATELLITE');
var googleLayerHYBRID = new L.Google("HYBRID");
var googleLayerTERRAIN = new L.Google('TERRAIN');

var baseMap = {
    "Route" : googleLayerROADMAP,
    "Satellite" : googleLayerSATELLITE,
    "Hybride" : googleLayerHYBRID,
    "Relief" : googleLayerTERRAIN
};

var map = new L.Map('maDiv', {
     center: new L.LatLng(48.858376, 2.294442),
     zoom: 11,
     layers: [googleLayerROADMAP]
});

var overlaysMaps = {
//Contrôles pour les éventuels layers contenant des données.
};

L.control.layers(baseMap, null, { collapsed: false }).addTo(map);
</script>
</body>
</html>
```

Leaflet et cartes Google

- Rendu Hybride

- Rendu Satellite

I.7- Ajout de données à une carte

■ Voici la liste des figures vectorielles de base (layer) avec lesquelles on peut enrichir une carte Leaflet. Elles sont abordées une à une dans le reste du document :

- Points et Marqueurs
- Marqueurs personnalisés
- Lignes et Poly lignes
- Polygones
- Rectangles et Cercles
- MultiPolylines et MultiPolygons

I.8- Points et Marqueurs

- On dessine souvent une carte pour mettre en avant un point donné de cette carte. Leaflet dispose pour cela de la classe Point. Cependant, on ne peut ajouter un simple point avec une icône. On est obligé en Leaflet d'utiliser la classe Marker. C'est avec un marqueur que l'on ajoute des points sur une carte Leaflet. Au minimum, la classe Marker requiert une latitude et une longitude :

```
var monMarqueur = L.marker([lat, long]).addTo(map);
```

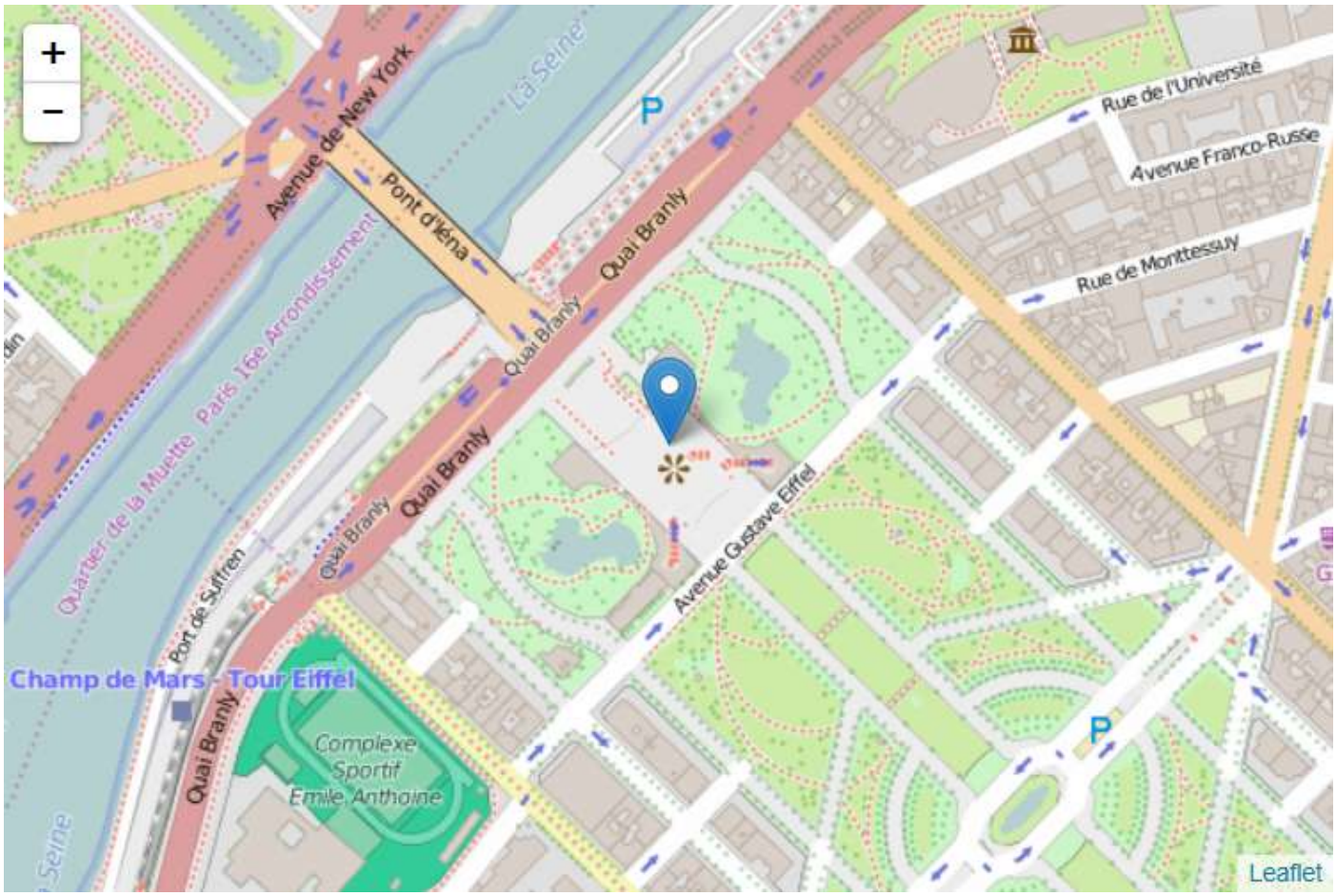
Ou

```
L.marker([lat,long]).addTo(map); // Ajoute le marqueur mais ne permet pas de manipuler le marqueur à postériori
```

- La classe Marker dispose d'options, d'événements et de méthodes. Il y a 10 options pouvant être spécifié à la création d'un marqueur:
  - icon
  - clickable
  - draggable
  - keyboard
  - title
  - alt : pour l'accessibilité
  - zIndexOffset
  - opacity
  - riseOnHover
  - riseOffset
- Les options clickable, draggable, keyboard, zIndexOffset, opacity, riseOnHover, et riseOffset ont toutes une valeur par défaut.

Points et Marqueurs

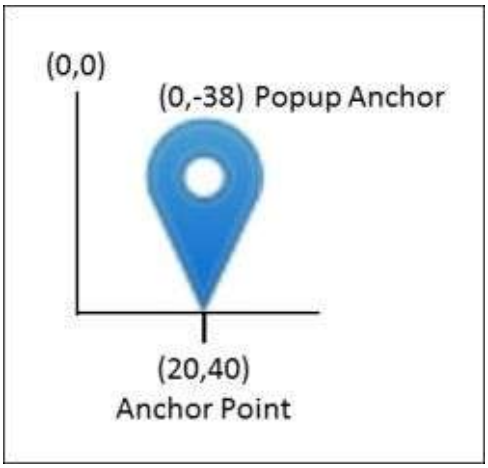
```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],16);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var monMarqueur = L.marker([48.858376, 2.294442],{title:"Tour Eiffel avec un
marqueur draggable",alt:"LA TOUR EIFFEL",draggable:true})
                .addTo(map);
</script>
```



I.9- Marqueurs personnalisés

En Leaflet, un marqueur est fait de 2 images: l'image du marqueur lui-même et celle de son ombre (facultative) pour créer de la profondeur. D'ailleurs, lorsque l'on télécharge Leaflet, on trouve dans un dossier images contenant les 2 représentations du marqueur par défaut: `marker-icon.png` et `marker-shadow.png`.

■ Pour créer l'icône d'un marqueur dans Leaflet, on doit créer une instance de la L.icon.



■ La classe L.icon prend en charge les options suivantes:

- iconUrl
- iconRetinaUrl
- iconSize
- iconAnchor
- shadowUrl
- shadowRetinaUrl
- shadowSize
- shadowAnchor
- popupAnchor
- className
- Remarque : la seule option obligatoire est iconUrl.

Marqueurs personnalisés

■ Les options : quelques explications

L'option `iconUrl` désigne l'URL de l'image de l'icône et `shadowUrl` désigne l'URL de l'image qui sert d'ombre.

Les options `iconSize` et `shadowSize` permettent de donner les dimensions de ces icônes : la largeur et la hauteur.

L'option `iconAnchor` indique les coordonnées pixels du point ou le marqueur et son icône touche la carte et où le pop-up touche l'icône.

L'option `popupAnchor` devrait être positionnée relativement à l'option `iconAnchor`.

```
/* Création de l'icône du marqueur */
var monIcone = L.icon({
  iconUrl: 'monImage.png',
  shadowUrl: 'sonOmbre.png',
  iconSize: [40, 60],
  shadowSize: [60, 40],
  iconAnchor: [20, 60],
  shadowAnchor: [20, 40],
  popupAnchor: [0, -53]
});
/* Création et utilisation du marqueur */
var monMarqueur = L.marker([48.858376, 2.294442],
{icon:monIcone}).addTo(map).bindPopup("Marqueur customisé !");
```

Marqueurs personnalisés

■ Définir des classes d'icones à l'aide de la classe L.Icon

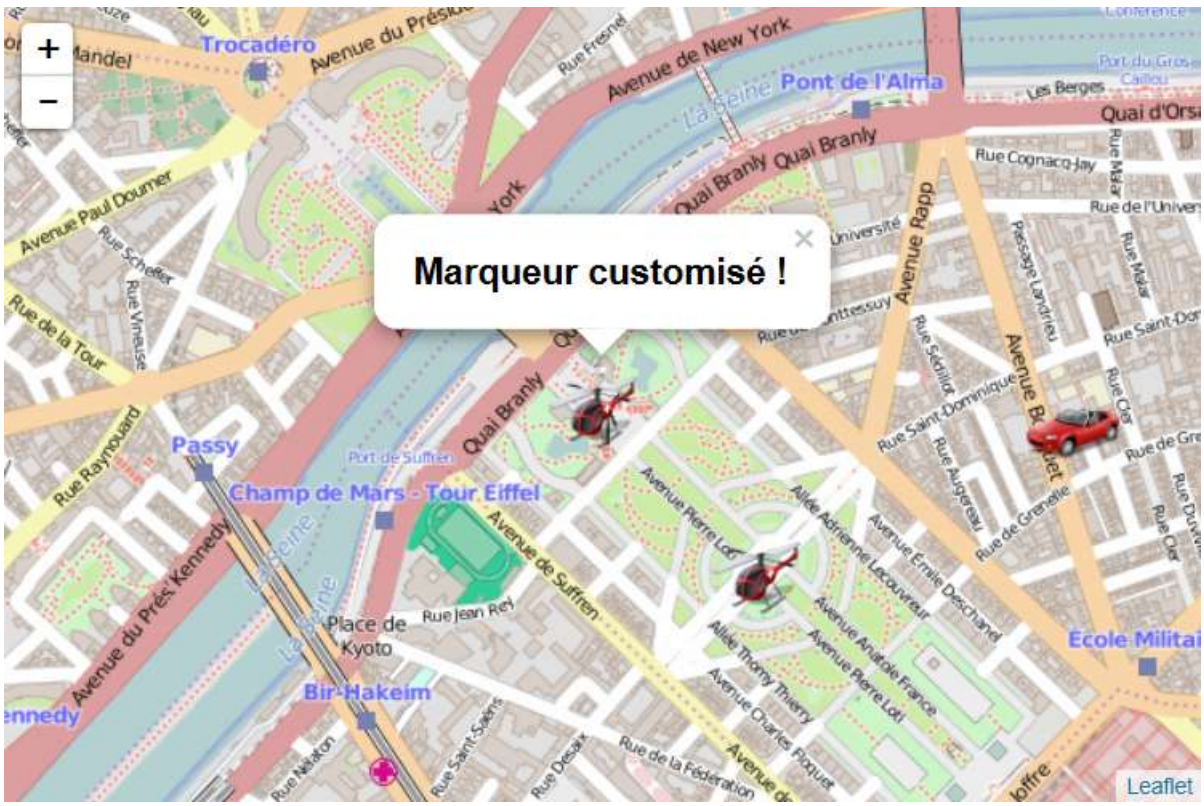
- On peut étendre la classe L.Icon pour créer se propre classe de marqueurs. Cela permet par exemple, de créer des marqueurs de couleurs variées en ne spécifiant la taille, les ancres et autres options qu'une seule fois.

```
var monModeleIcone = L.Icon.extend({
  options:{
    shadowUrl: 'monOmbre.png',
    iconSize: [40, 60],
    shadowSize: [60, 40],
    iconAnchor: [20, 60],
    shadowAnchor: [20, 40],
    popupAnchor: [0, -53
  }
});
/* Ensuite, il suffit de donner des images de chaque couleur pour disposer
de plusieurs icones avec les mêmes caractéristiques communes */
var iconeRouge = new monModeleIcone ({iconUrl: 'monMarqueurRouge.png'});

var iconeVerte = new monModeleIcone ({iconUrl: 'monMarqueurVert.png'});

/* Il ne reste plus qu'à cérer et utiliser nos 2 marqueurs */
var monMarqueurRouge = L.marker([48.858376, 2.294442],
{icon:iconeRouge}).addTo(map).bindPopup("Mon marqueur Rouge");

var monMarqueurVert = L.marker([48.86, 2.30],
{icon:iconeVerte}).addTo(map).bindPopup("Mon marqueur Vert.");
```

Marqueurs personnalisés

■ Exemple complet : Personnaliser l'icône d'un marqueur

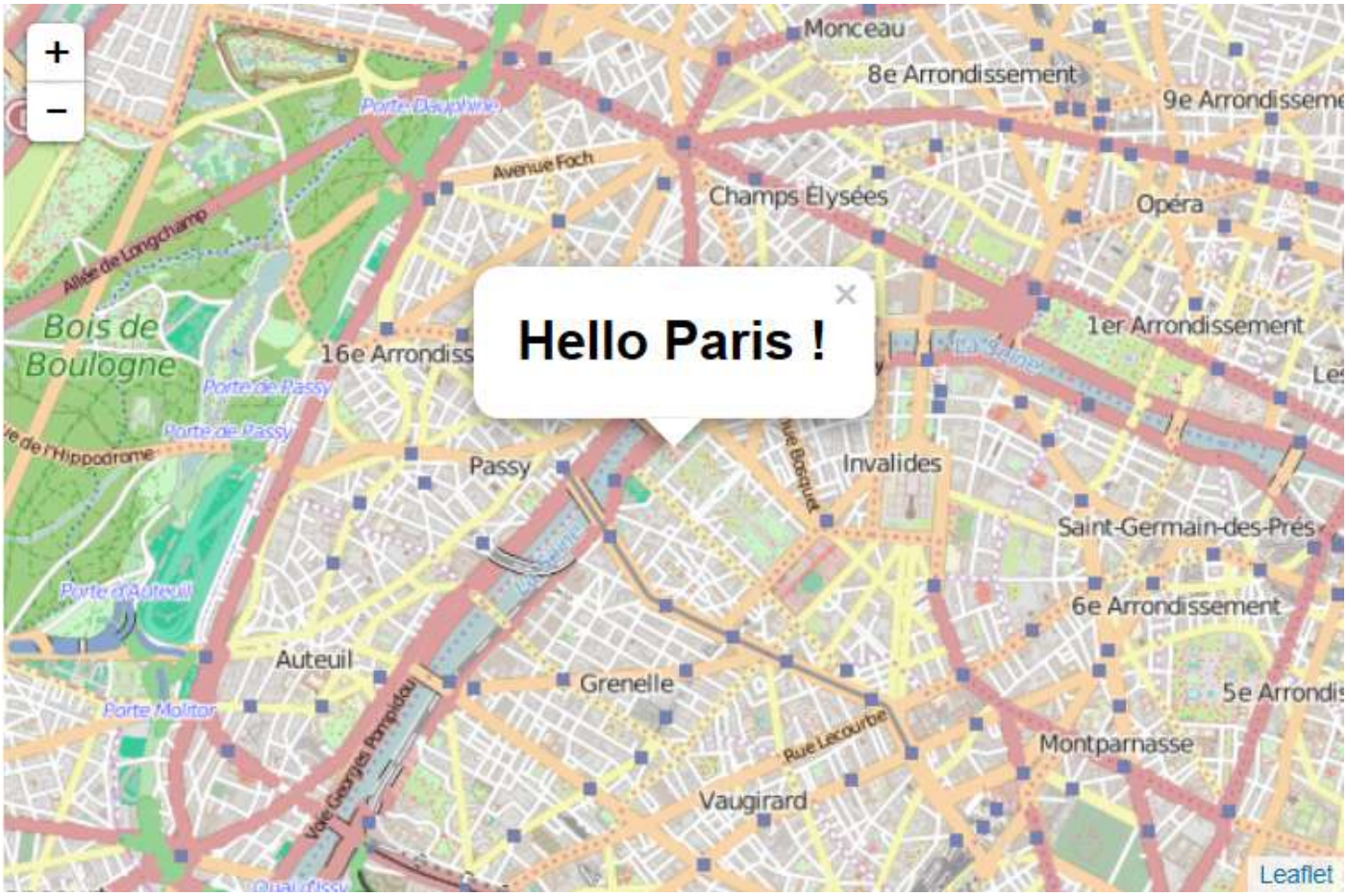
```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],15);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

/* Création d'icônes */
var monIcone1 = L.icon({
  iconUrl: 'helico48x48.png', /* Image à FOND TRANSPARENT !! */
  shadowUrl: '', /* Pas obligatoire, mais avec une ombre, ca fait plus pro */
  iconSize: [48, 48],
  //shadowSize: [50, 20],
  iconAnchor: [30, 20],
  //shadowAnchor: [10, 20],
  popupAnchor: [0, -24]
});
var monIcone2 = L.icon({
  iconUrl: 'cabriolet48x48.png' /* Seule option exigée */
});
/* Création et utilisation d'une icône pour un marqueur */
var monMarqueur=L.marker([48.858376, 2.294442], {icon:
monIcone1}).addTo(map).bindPopup("<h2>Marqueur customisé !</h2>");
var monMarqueur=L.marker([48.85606, 2.29794], {icon:
monIcone1}).addTo(map).bindPopup("<h2>Marqueur customisé !</h2>");
var monMarqueur=L.marker([48.8588, 2.30348], {icon:
monIcone2}).addTo(map).bindPopup("<h2>Mon autre marqueur customisé !</h2>");
</script>
```



I.10-Pop-up

■ Il est possible de créer un pop-up sans l'associer absolument à un marqueur et donc de l'afficher seul :

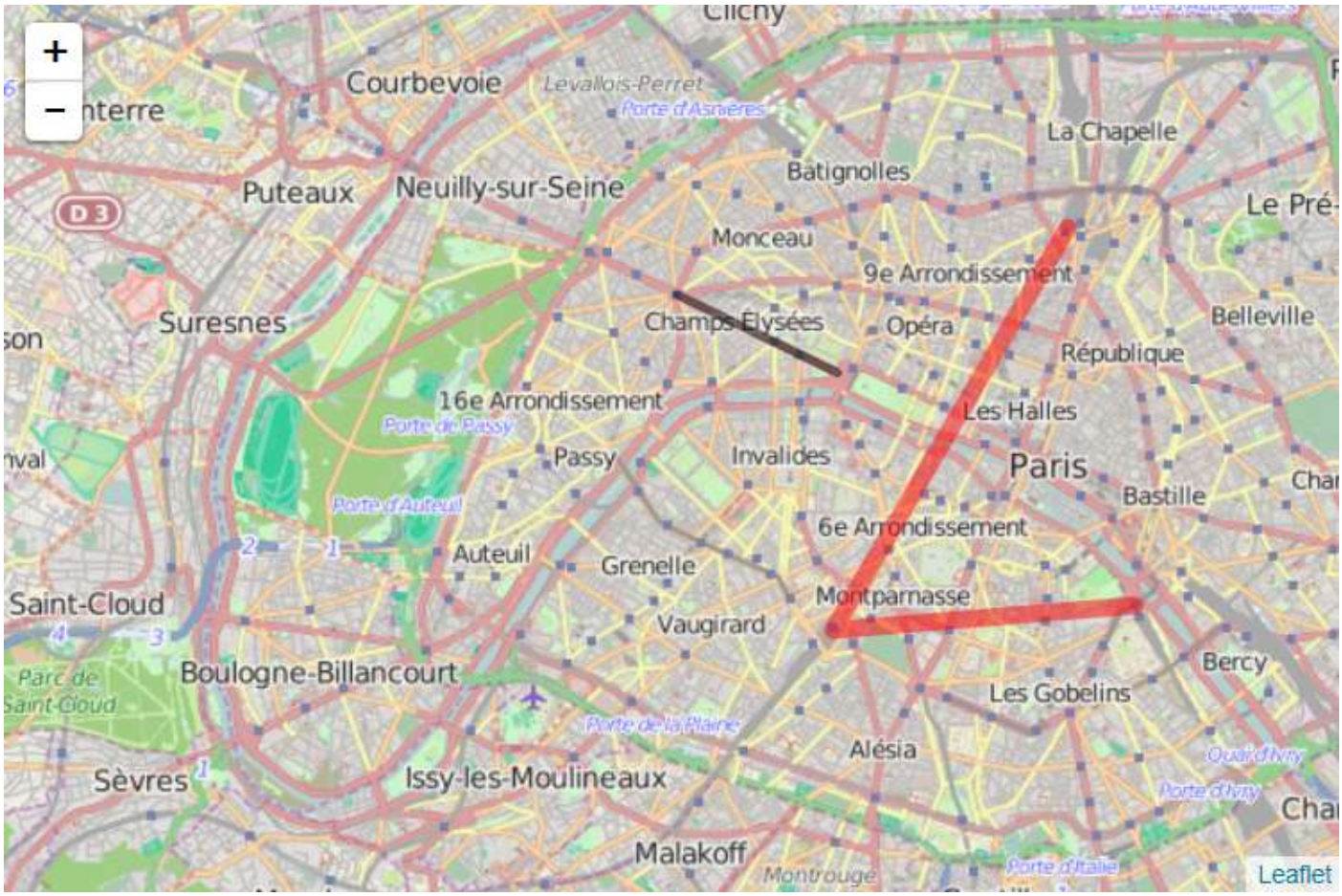
```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],13);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var monPopup = L.popup();
monPopup.setLatLng([48.858376, 2.294442])
        .setContent("<h1>Hello Paris !</h1>")
        .openOn(map);
</script>
```



I.11-Lignes et Poly lignes

- La premier objet vectoriel dont on dispose est la ligne et plus généralement la poly-ligne (ensemble de segments de droite, au moins un). En Leaflet, on dispose de la classe Polyline pour dessiner une simple ligne à l'aide d'un segment ou une poly-ligne avec plusieurs segments. Poly lignes et polygones étendent la classe Path de Leaflet. On ne peut faire appel directement à la classe Path mais on dispose bien entendu de toutes ses méthodes, ses évènements et ses attributs.

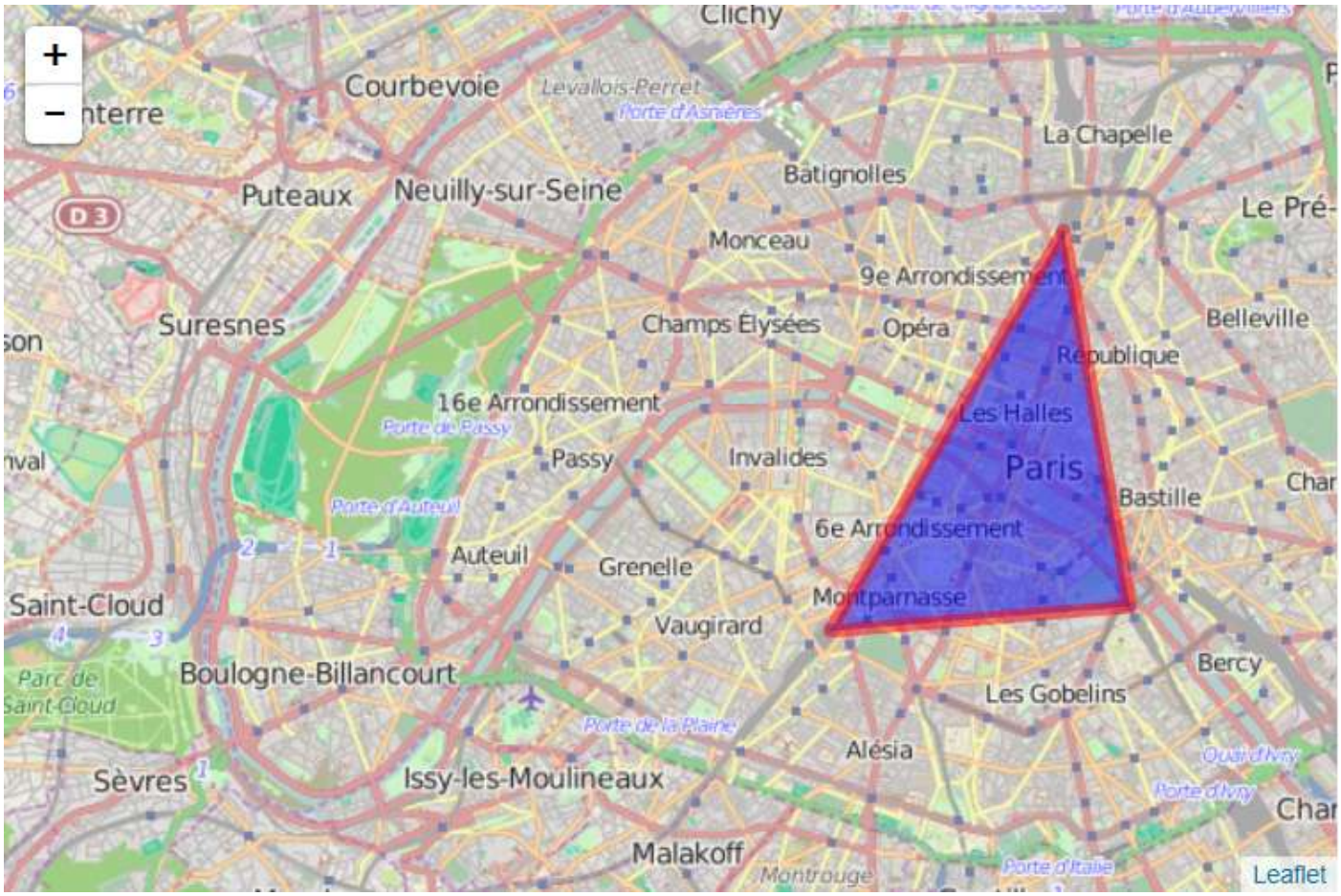
```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var ligne = L.polyline([[48.8738, 2.29504],[48.8659, 2.31983]], {color:
'black',weight:4})
            .addTo(map);
var polyligne = L.polyline([[48.88044, 2.35533],[48.83997, 2.31921],[48.84232,
2.36563]], {color: 'red',weight:8})
            .addTo(map);
</script>
```



I.12-Polygones

- Un polygone est juste une poly-ligne fermée. Leaflet dispose de classes différentes pour dessiner deux polygones « particuliers » : le cercle et le rectangle (voir plus loin). Pour un polygone, Leaflet clôt automatiquement la poly-ligne (pas besoin de re-spécifier le point de départ en fin de listes de points).

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var polygone = L.polygon([[48.88044, 2.35533],[48.83997, 2.31921],[48.84232,
2.36563]], {color: 'red',weight:6,fillColor:'blue',fillOpacity:0.5})
                .addTo(map);
</script>
```



I.13-Rectangles et Cercles

■ Cercles et rectangles sont considérés comme des polygones mais disposent de classes dédiées dans Leaflet.

■ Rectangle

- Pour créer un rectangle, il faut une instance de la classe L.rectangle() avec une paire de latitude et longitude pour le coin supérieur gauche et le coin en bas à droite. La classe étend L.polygon(), on a donc accès aux mêmes options, méthodes, et évènements:

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var monRectangle = L.rectangle([[48.87782, 2.26085],[48.84408, 2.3587]],
{color: "red", weight: 8,fillColor:"blue"})
                    .addTo(map);
</script>
```

Rectangles et Cercles

■ Cercle

- Pour créer un cercle, on a besoin d'une instance de la classe L.circle() avec le point du centre et un rayon (en mètres) comme paramètres. Comme la classe Rectangle, la classe Cercle étend la classe Path et hérite donc des mêmes caractéristiques.

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var monCercle = L.circle([48.85307, 2.34991], 1500.0,{color: "red", weight:
8,fillColor:"blue"})
                .addTo(map);
</script>
```

- Ici on a spécifié le centre avec un rayon de 1500 mètres.

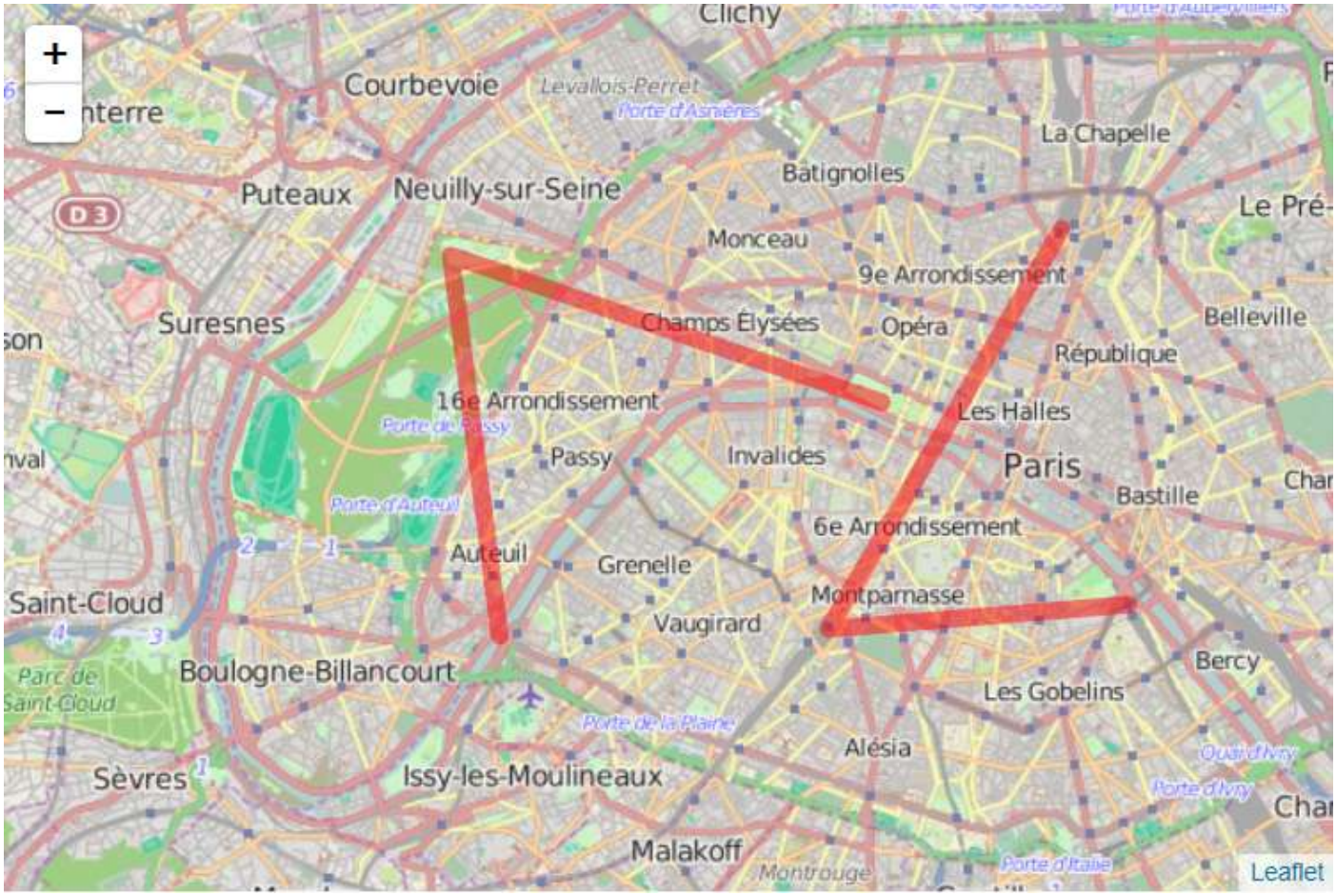
I.14-MultiPolylines et MultiPolygons

- Pour décrire les MultiPolylignes et MultiPolygones, on utilise les crochets [et] pour chacun d'entre eux. En plus d'utiliser les crochets pour chaque latitude-longitude. Gare aux erreurs d'inattention.
- Les MultiPolygones et MultiPolylignes partagent les même pop-up.

MultiPolylines

- La création d'une MultiPolyligne ressemble à celle d'une poly ligne, sauf que l'on passe de multiples longitudes et latitudes (un ensemble par poly ligne).

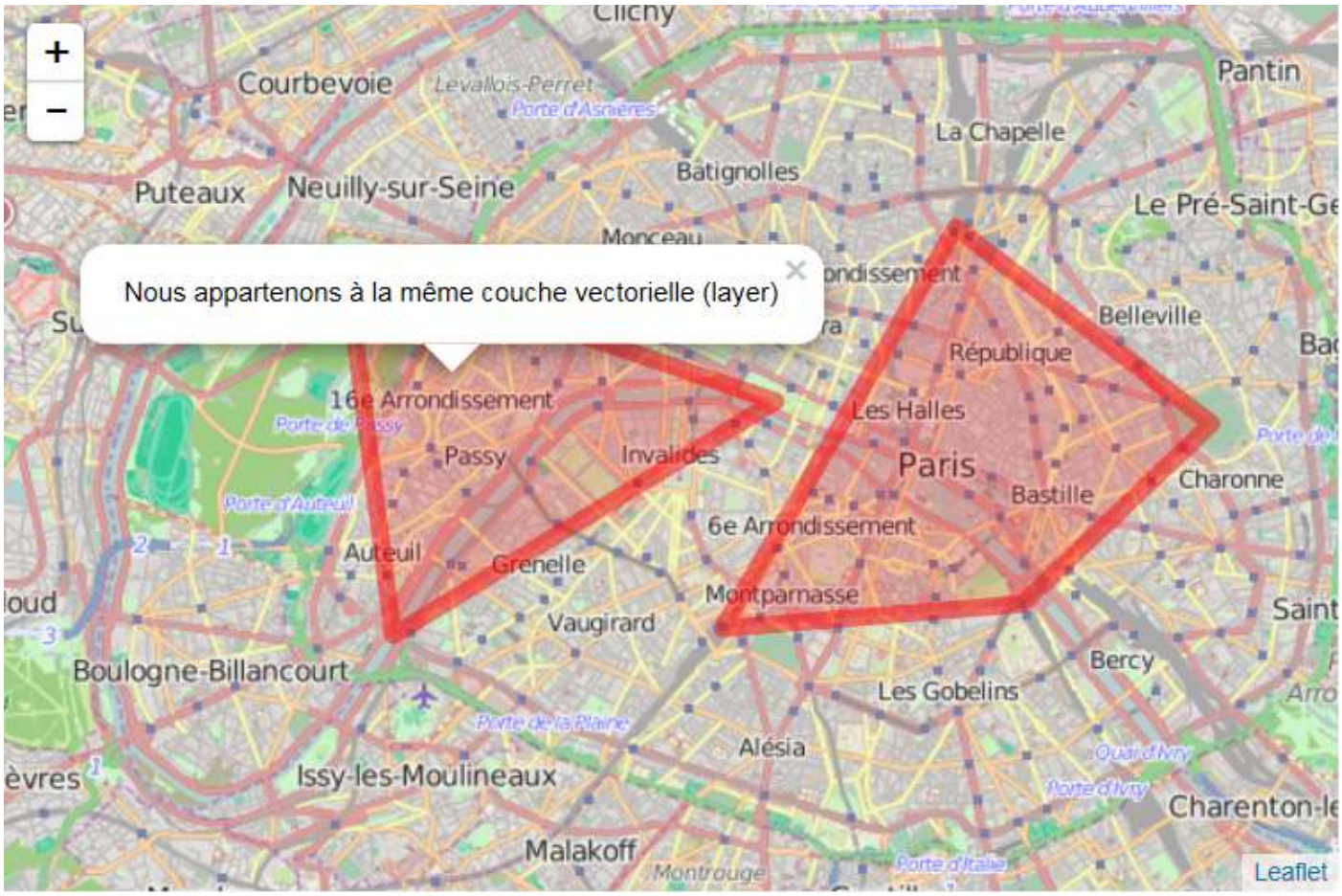
```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);
var multipolyline = L.multiPolyline([[[48.83929, 2.26833],[48.87782,
2.26085],[48.86301, 2.32732]],[[48.88044, 2.35533],[48.83997,
2.31921],[48.84232, 2.36563]]],{color: 'red',weight:8})
                    .addTo(map);
</script>
```



MultiPolygons

- La création d'un MultiPolygone est aussi similaire à la création d'une MultiPolyligne. Rappel : Leaflet ferme automatiquement chaque polygone.

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var multipolygon = L.multiPolygon([[[48.83929, 2.26833],[48.87782,
2.26085],[48.86301, 2.32732]],[[48.88044, 2.35533],[48.83997,
2.31921],[48.84232, 2.36563],[48.86075, 2.39454]]],{color: 'red',weight:8})
            .addTo(map)
            .bindPopup("Nous appartenons à la même couche vectorielle (layer)");
</script>
```



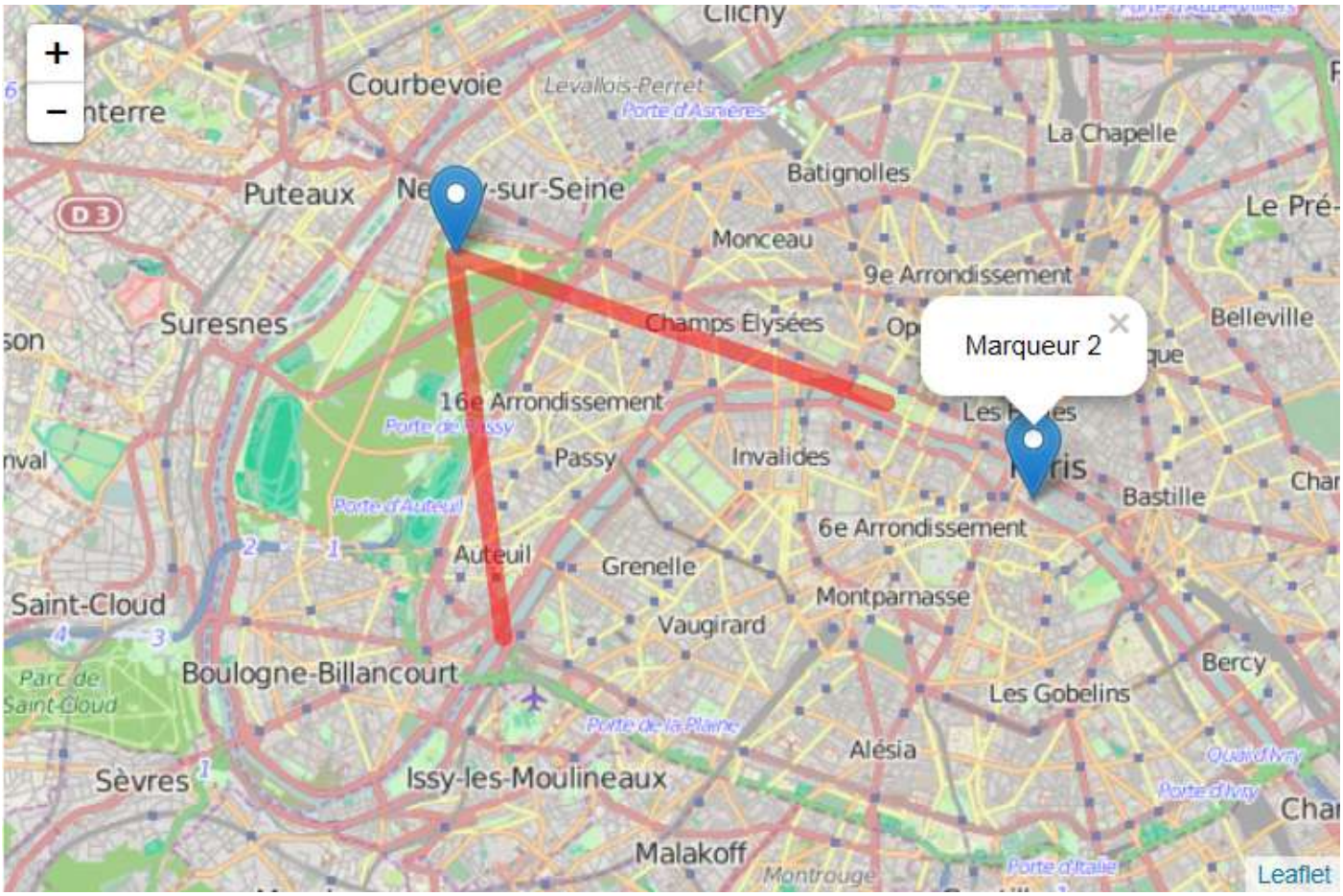
I.15-Regroupements de figures vectorielles

- Les MultiPolylignes et les MultiPolygones permettent de combiner vde multiple poly lignes et polygones. Ceci est utile lorsque l'on désire créer un groupe de figures vectorielles de même type.
- Mais lorsque l'on désire mélanger des figures vectorielles de types différents (un marqueur et un cercle par exemple), on doit alors recourir aux « layer group » ou aux « feature group » de Leaflet.
- Les 2 compositions sont similaires mais différent quelque peu sur la prise en compte de la gestion des évènements et sur l'association de pop-up.

I.16-Layer Group

- Un layer group permet d'ajouter de multiples figures vectorielles (layers) de différent type à une carte, pour les gérer ensuite comme une seule et même figure (layer).

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var monMarqueur1 = L.marker([48.87782, 2.26085])
                    .bindPopup("Marqueur 1");
var maPolyline = L.polyline([[48.83929, 2.26833],[48.87782, 2.26085],[48.86301,
2.32732]], {color: 'red',weight:8})
                    .bindPopup("Polyline");
var monMarqueur2 = L.marker([48.85307, 2.34977])
                    .bindPopup("Marqueur 2");
var monLayerGroup = L.layerGroup([monMarqueur1, maPolyline])
                    .addTo(map);
monLayerGroup.addLayer(monMarqueur2);
</script>
```



Layer Group

- Comme auparavant, on doit utiliser la méthode addTo() après avoir créé le layer group et les figures le composant :

```
var monLayerGroup=L.layerGroup([marqueur, polyline]).addTo(map);
```

- Pour éviter de transmettre les figures en paramètre de la création, on peut aussi utiliser la méthode addLayer() pour ajouter une gigure (layer) à un layer group:

```
monLayerGroup.addLayer(marqueur2);
```

- La méthode removeLayer() sert à retirer une figure du groupe :

```
MonLayerGroup.removeLayer(marqueur);
```

- Dès que l'on supprime une figure d'un groupe, elle n'est plus affichée car on a fait appel à la méthode addTo() sur le groupe et non individuellement, sur la figure elle-même. A ce moment là, si ca nous chante, on peut afficher la figure individuellement en l'ajoutant de nouveau à la carte mais à titre individuel et donc indépendamment du groupe auquel elle appartenait :

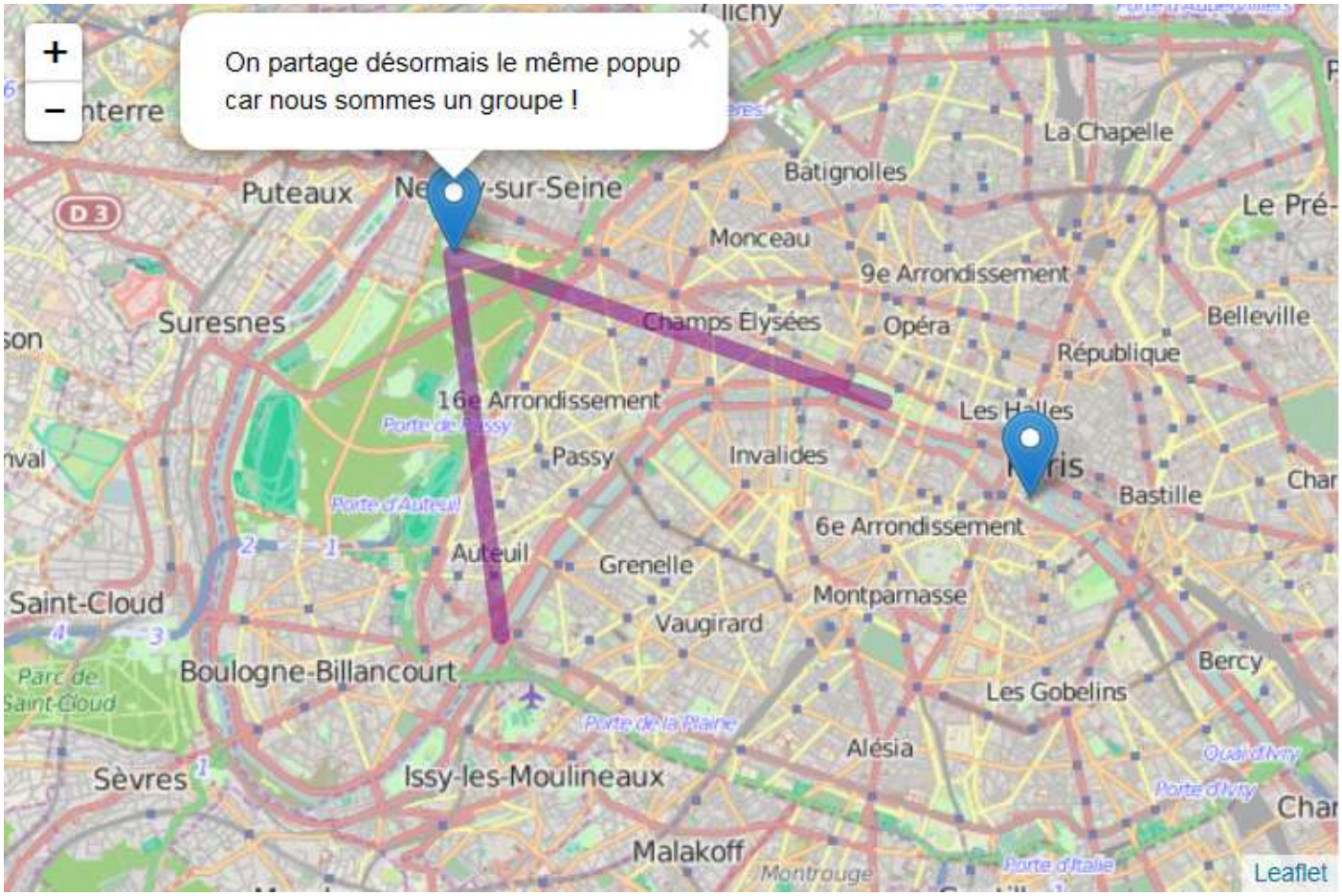
```
marqueur.addTo(map);
```

- Important : Dans le cas des group layers, toutes les options de style et pop-ups doivent être assignés à la figure (layer) dès leur création AVANT de les mettre dans le groupe.
- On ne peut pas assigner un style ou un des pop-ups à un layer group dans sa globalité. Ce sont les feature groups qui le permettent.

I.17-Feature Group

- Un feature group est similaire à un layer group, sauf qu'il l'étend en traitant les évènements liés à la souris et en fournissant également la méthode bindPopup(). Le constructeur pour un feature group est le même que celui d'un layer group.

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var monMarqueur1 = L.marker([48.87782, 2.26085])
                            .bindPopup("Marqueur 1");
var maPolyline = L.polyline([[48.83929, 2.26833],[48.87782, 2.26085],[48.86301,
2.32732]], {color: 'red',weight:8})
                            .bindPopup("Polyline");
var monMarqueur2 = L.marker([48.85307, 2.34977]).bindPopup("Marqueur 2");
var monFeatureGroup = L.featureGroup([monMarqueur1, monMarqueur2, maPolyline])
                            .addTo(map)
                            .setStyle({color:'purple',opacity:0.5})
                            .bindPopup("On partage désormais le même
popup<br/>car nous sommes un groupe !");
</script>
```



Feature Group

■ Remarque

- La méthode setStyle() s'applique aux figures d'un feature group qui elles-mêmes disposent d'une méthode setStyle(). Une poly ligne étend la class Path et donc dispose de la méthode setStyle(). Les marqueurs n'ont pas de méthode setStyle(). Ils gardent leur style inchangé si on les retire d'un feature group contrairement à une poly ligne par exemple qui se verrait appliquer sa propre méthode de style setStyle().

I.18-Popups

- A pop up est destiné à afficher une information à l'utilisateur. La manière la plus simple d'ajouter un pop-up à un marqueur, une poly ligne, un polygone c'est de faire appel à la méthode bindPopup(). La méthode bindPopup() permet d'insérer un contenu HTML au pop-up.
- On peut aussi créer une instance de la classe popup et lui assigner de multiples objets.

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],12);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);
var monMarqueur = L.marker([48.858376, 2.294442]).addTo(map);
var html ="<h3>Mon marqueur avec popup</h3><p>La Tour
Eiffel</p><ul><li>Construction de 1887 à 1889</li><li>Hauteur avec antenne
324m</li><li>Nombre d'ascenseurs 6</li></ul><center><img src='Tour
Eiffel.png'></center>";
var monPopup = L.popup({keepInView:true,closeButton:false}).setContent(html);

monMarqueur.bindPopup(monPopup);
// OU directement
// monMarqueur.bindPopup(html);
</script>
```

■ Exemple avec généralisation via des fonctions

```
function creerPopup(x){
  return L.popup({keepInView:true,closeButton:false}).setContent(x);
}

var m1 = L.marker([48.858376, 2.294442])
.addTo(map).bindPopup(creerPopup("Texte1"));

var m2 = L.marker([48.75, 2.25])
.addTo(map).bindPopup(creerPopup("Texte2"));
```

I.19-Mapping pour les mobiles

- Un des avantages du « mapping » en JavaScript est que les mobiles peuvent exécuter le même code via un simple navigateur standard sans addon quel qu'il soit.
- Bien entendu Leaflet tournent sur des mobiles mobile comme iPhone, iPad, ou encore Android. N'importe quel page web avec Leaflet tourne sans modifications sur un mobile. Néanmoins, on peut souhaiter adapter son application à l'usage des mobiles (style, zoom à 2 doigts, etc..).
- Par exemple la classe L.map() dispose d'une méthode locate(), qui permet d'utiliser l'API de géo localisation du W3C. Cette API permet d'obtenir la localisation d'un utilisateur suivant son adresse IP, les informations de son réseau wifi ou encore à partir des données GPS de son mobile.
- Mais avant toute chose, il faut veiller à adapté le style d'affichage à un écran de smartphone.
- Exemple de style adapté aux mobiles

```
<style>
body {
padding: 0;
margin: 0;
}
html, body, #map {
height: 100%;
}
</style>
```

- Balise meta pour les mobiles

```
<meta name="viewport" content="width=device-width, initial-
scale=1.0, maximum-scale=1.0, user-scalable=no">
```

- Si la cible d'affichage sont les mobiles, on ajoute (dans l'entête de la page web) la balise précédente qui modifie le « viewport » sous lequel le site va être affiché. On établit le « viewport » à la largeur du mobile afin d'obtenir un ratio d'affichage de1:1. Enfin, on interdit de pouvoir redimensionner la page web ce qui n'affecte pas la fonctionnalité de zoom d'une carte.

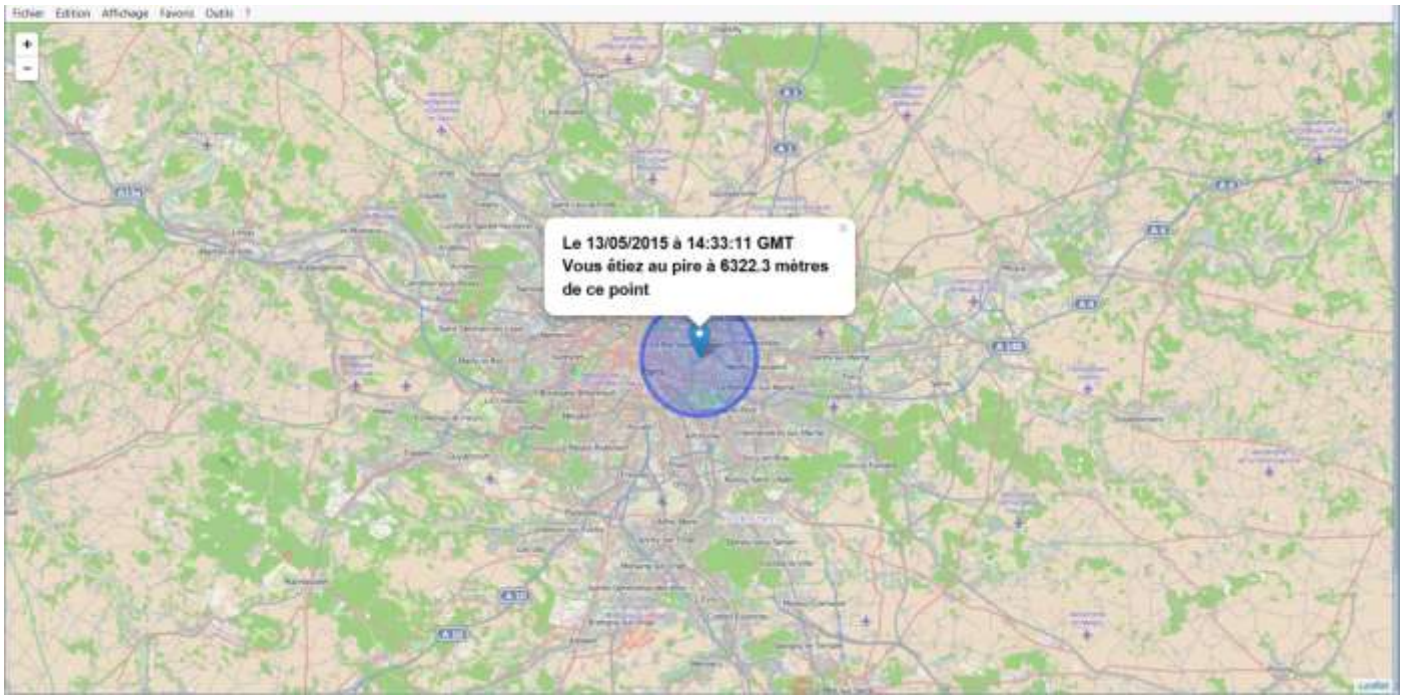
Mapping pour les mobiles

- Exemple complet

```
<!DOCTYPE html>
<html lang="fr">
<head>
<title>Leaflet.js</title>
<meta charset="utf-8" />
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-
0.7.3/leaflet.css" />
<script src="http://cdn.leafletjs.com/leaflet-0.7.3/leaflet.js"></script>
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-
scale=1.0, user-scalable=no" />
<style>
body {
  padding: 0;
  margin: 0;
}
html, body, #maGrandeDiv {
  height: 100%;
}
</style>
</head>
<body>
<div id="maGrandeDiv"></div>
<script>
var map = L.map('maGrandeDiv');
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

function onLocationFound(e) {
  var radius = e.accuracy / 10;
  var d = new Date();
  L.marker(e.latlng).addTo(map)
      .bindPopup("<h3>Le "+d.toLocaleDateString()+ " à
"+d.toLocaleTimeString()+" GMT<br/>Vous êtiez au pire à " + radius + " mètres
de ce point</h3>")
     .openPopup();
  L.circle(e.latlng, radius).addTo(map);
}

function onLocationError(e) {
  alert("Impossible de vous localiser ! Ou bien vous ne voulez pas ?");
}

map.on('locationerror', onLocationError);
map.on('locationfound', onLocationFound);
map.locate({setView: true, maxZoom:14});
</script>
</body>
</html>
```

Mapping pour les mobiles

- Rendu



I.20-Les évènements

- Leaflet gère un grand nombre d'évènements pour chacune de ses classes de base. Généralement un évènement (qui survient) sur un objet permet d'exécuter une fonction pour réagir à l'évènement. Par exemple, quand l'utilisateur clique sur la carte, cela génère l'évènement `'click'` sur l'objet carte.

```
map.on('click', function(e) {
    alert(e.latlng);
});
```

- Leaflet utilise la notion de « event listeners » et la programmation évènementielle. On peut donc ajouter un listener (une fonction d'écoute) puis ensuite le supprimer :

```
function onClick(e) { ... }

map.on('click', onClick); // Positionner un handler
map.off('click', onClick); // Déprogrammer un handler
```

- Objet Event
  - L'objet Event est l'objet reçu en argument de la fonction listener lorsqu'un évènement est déclenché. Il contient des informations souvent utiles à propos de l'évènement :

```
map.on('click', function(e) {
  alert(e.latlng);
  // e is an event object (MouseEvent in this case)
});
```

  - Voici le contenu de l'objet de base event. Tous les autres objets Event (chacun associé à une classe gérée par Leaflet) contient également ces propriétés :

propriété	type	description
type	String	Le type d'évènement (par exemple 'click').
target	Object	L'objet ayant déclenché l'évènement (la source).

Les évènements

■ Les méthodes de gestion des évènements permettant de modifier/surveiller l'état d'une carte Leaflet :

Method	Returns	Description
addEventListener(<String> *type*, <Function> *fn*, <Object> *context?*)	this	Adds a listener function (fn) to a particular event type of the object. You can optionally specify the context of the listener (object the this keyword will point to). You can also pass several space-separated types (e.g. 'click dblclick').
addOneTimeEventListener(<String> *type*, <Function> *fn*, <Object> *context?*)	this	The same as above except the listener will only get fired once and then removed.
addEventListener(<Object> *eventMap*, <Object> *context?*)	this	Adds a set of type/listener pairs, e.g. {click: onClick, mousemove: onMouseMove}
removeEventListener(<String> *type*, <Function> *fn?*, <Object> *context?*)	this	Removes a previously added listener function. If no function is specified, it will remove all the listeners of that particular event from the object. Note that if you passed a custom context to addEventListener, you must pass the same context to removeEventListener in order to remove the listener.
removeEventListener(<Object> *eventMap*, <Object> *context?*)	this	Removes a set of type/listener pairs.
removeEventListener()	this	Removes all listeners. An alias to clearAllEventListeners when you use it without arguments.
hasEventListeners(<String> *type*)	Boolean	Returns true if a particular event type has some listeners attached to it.
fireEvent(<String> *type*, <Object> *data?*)	this	Fires an event of the specified type. You can optionally provide

		an data object — the first argument of the listener function will contain its properties.
clearAllEventListeners()	this	Removes all listeners to all events on the object.
on(…)	this	Alias to addEventListener.
once(…)	this	Alias to addOneTimeEventListener.
off(…)	this	Alias to removeEventListener.
fire(…)	this	Alias to fireEvent.

Les évènements

■ Exemple 1

- A chaque fois que l'utilisateur clique n'importe où sur la carte, on affiche un message à l'aide de la méthode alert().

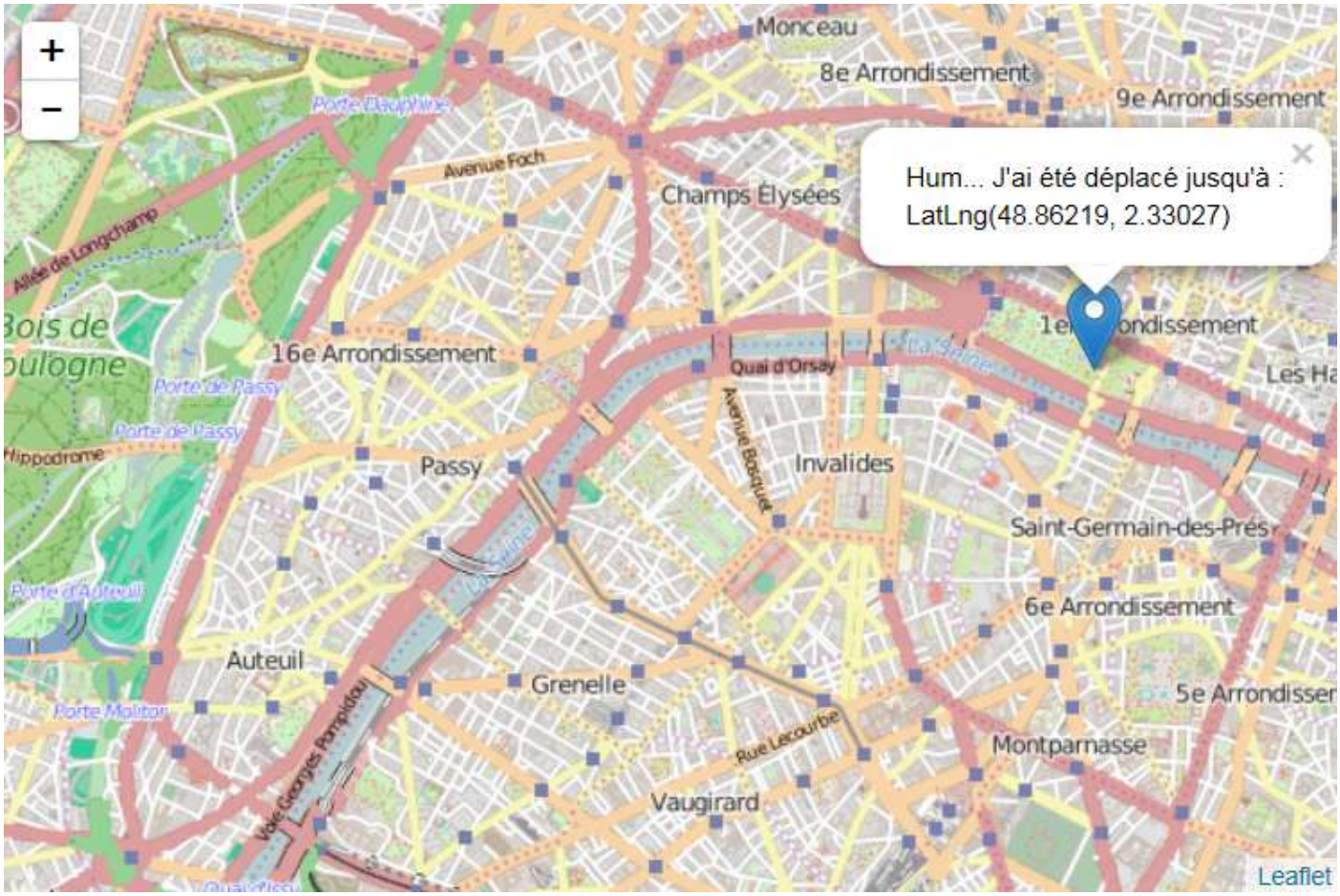
```
map.on('click',function(e){
  var coord = e.latlng.toString().split(',');
  var lat = coord[0].split('(');
  var long = coord[1].split(')');
  alert("Vous avez cliqué sur la carte à la LATITUDE: "+ lat[1]+"
et à la LONGITUDE:"+long[0]) ;
});
L.marker(e.latlng).addTo(map);
```

Les évènements

■ Exemple 2

- On peut créer un marqueur avec la propriété draggable à true. Les marqueurs gèrent 3 évènements en relation avec le « drag »: dragstart, drag, et dragend.

```
<script>
var map = L.map('maDiv').setView([48.858376, 2.294442],13);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);

var monMarqueur = L.marker([48.858376, 2.294442],{title:"Tour Eiffel",alt:"TOUR
EIFFEL",draggable:true}).addTo(map);

function ouSuisJe(){
  monMarqueur.bindPopup("Hum... J'ai été déplacé jusqu'à
:<br/>"+String(monMarqueur.getLatLng())).openPopup();
}

monMarqueur.on('dragend',ouSuisJe);
</script>
```



I.21-Ma première carte « riche »

- Code

```
<script>
var coucheSatellite = new
L.TileLayer('http://otile{s}.mqcdn.com/tiles/1.0.0/sat/{z}/{x}/{y}.png', {
  subdomains: ['1', '2', '3', '4']
});
var coucheRelief = new
L.TileLayer('http://{s}.tile.thunderforest.com/landscape/{z}/{x}/{y}.png');

var map = new L.Map('maDiv', {
   center: new L.LatLng(45.31198, 2.5664),
   zoom: 4,
   layers: [ coucheSatellite ]
});
map.attributionControl.setPrefix(''); // On ne veut plus afficher 'Leaflet'

var IconeFeuille = L.Icon.extend({
  options: {
     shadowUrl: 'leaf-shadow.png',
     iconSize:     [38, 95],
     shadowSize:   [50, 64],
     iconAnchor:   [22, 94],
     shadowAnchor: [4, 62],
     popupAnchor:  [-3, -76]
    }
});
iconeOrange = new IconeFeuille({iconUrl: 'leaf-orange.png'});

var france = L.marker([46.76997, 2.26758]).bindPopup("<img src='drapeaux/120px-
Drapeau_de_la_France_svg.png'/>").openPopup();
var italie = L.marker([40.99317, 15.31055]).bindPopup("<img
src='drapeaux/120px-Drapeau_de_lItalie_svg.png'/>").openPopup();
var espagne = L.marker([38.67179, -3.63867]).bindPopup("<img
src='drapeaux/120px-Drapeau_de_lEspagne_svg.png'/>").openPopup();
var algerie = L.marker([32.47826, 2.14893]).bindPopup("<img
src='drapeaux/algerie.png'/>").openPopup();
var pays = L.layerGroup([france, italie, espagne, algerie]);

var paris = L.marker([48.8705, 2.31592], {icon: iconeOrange}).bindPopup("<h2>
PARIS </h2>").openPopup();
var rome = L.marker([41.89246, 12.50024], {icon: iconeOrange}).bindPopup("<h2>
ROME </h2>").openPopup();
var madrid = L.marker([40.45029, -3.68262], {icon:
iconeOrange}).bindPopup("<h2> MADRID </h2>").openPopup();
var alger = L.marker([36.70894, 3.0542], {icon: iconeOrange}).bindPopup("<h2>
ALGER </h2>").openPopup();
var capitales = L.layerGroup([paris, rome, madrid, alger]);
```

Ma première carte « riche »

- Code (suite et fin)

```
var baseMaps = {
  'Satellite': coucheSatellite,
  'Relief': coucheRelief
};
var overlayMaps = {
  'Drapeaux': pays,
  'Capitales': capitales
};
L.control.layers(baseMaps, overlayMaps, {collapsed:false}).addTo(map);
</script>
```

- Rendu 1 : Affichage initial

Ma première carte « riche »

- Rendu 2 : Les marqueurs de capitales

- Rendu 3 : Les popups de capitales

Ma première carte « riche »

- Rendu 4 : Les marqueurs de drapeaux

- Rendu 5 : Les popups de drapeaux

Ma première carte « riche »

- Rendu 6 : La totale