Projet - Documentation – ULR - 2017

Développement sur smartphones

Géolocalisation et le framework MapKit

Introduction.

Lorsque l'on travaille sur la géolocalisation, le besoin de visualiser une carte vient naturellement. Nous allons voir dans ce document comment afficher une carte et permettre à l'utilisateur d'interagir avec elle; nous mettrons en œuvre le framework MapKit.

Partie 1. Framework MapKit

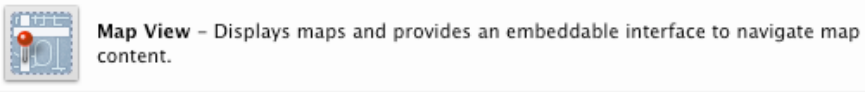
ETAPE 1 à réaliser

- Insérez dans la vue de l'application un objet de type MKMapView
- Importez le framework MapKit.framework
  - *Allez sur le nom du projet (dossier bleu au sommet de l'arborescence)*
  - *Cliquez sur le nom du projet sous Target*
  - *Choisissez l'onglet buildPhases, puis l'onglet Link BinaryWith Library, puis + et cherchez le MapKit.framework*
- Compilez et lancez le programme dans le simulateur
  Une carte du monde doit s'afficher avec laquelle vous pouvez interagir.
  Attention, vous devez être connecté en wifi et vous devez vous déconnecter du réseau filaire.

... plus de détails pour vous aider.

Le framework MapKit n'est pas intégré par défaut aux projets XCode. Pour ajouter un framework, il faut aller sur l'onglet Build Phases.

Ajoutons une vue Carte



Lançons l'application

La vue cartographique que nous venons d'insérer est de la classe MKMapView. Dans la suite, nous examinerons les caractéristiques les plus courantes de la classe MKMapView qui est au centre du framework MapKit.

Partie 2 . Connaitre la zone affichée

ETAPE 2 (à réaliser)

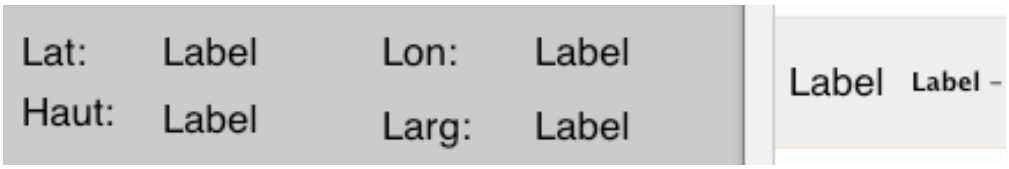
- De quoi sont composées les structures :
  MKCoordinateRegion, CLLocationCoordinate2D, MKCoordinateSpan
- Construisez une méthode affichant la longitude, la latitude, la largeur et la hauteur de la zone en cours
- Construisez une méthode mémorisant la zone en cours
- Construisez une interface permettant de gérer les fonctionnalités précédentes
- Utilisez la propriété *region* d'une instance de *MKMapView* et les propriétés *span* et *center* de *region* pour positionner la vue sur Paris.
- Utilisez la méthode MKCoordinateRegionMakeWithDistance pour faire un zoom sur une zone de taille 500 *500

... plus de détails pour vous aider.

La zone affichée sur la carte est accessible par la propriété *region* de la classe MKMapView. Nous allons en illustrer le fonctionnement en modifiant l'application Carte pour que l'utilisateur puisse en visualiser les caractéristiques.

Définir les Outlets

Sous Interface Builder, nous ajoutons quatre labels de texte sur la vue principale afin d'y afficher les coordonnées du centre de la classe (latitude et longitude) et la taille de la zone affichée (hauteur et largeur).



1. Sous Xcode, on ouvre le fichier TestMapViewController.h (ou SecondViewController pour une application à deux vues) pour y ajouter les outlets permettant d'accéder aux éléments de la vue principale : latitudeLabel, longitudeLabel, hauteurLabel, largeurLabel, carte.
2. Nous y définissons également le contrôleur comme répondant au protocole MKMapViewDelegate pour qu'il soit notifié des changements sur la carte.

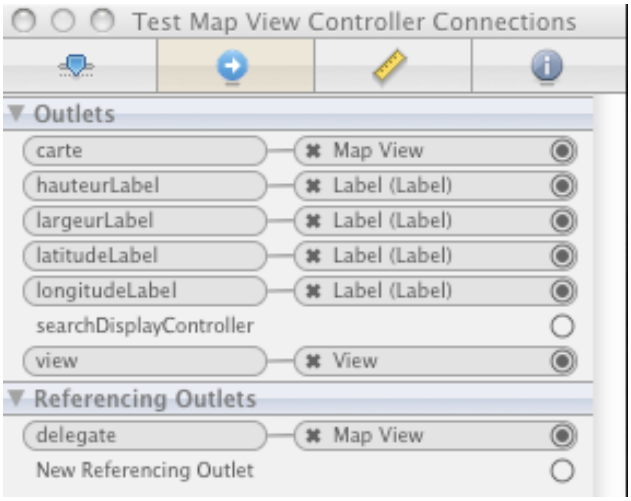
Interface de la classe CarteViewController.

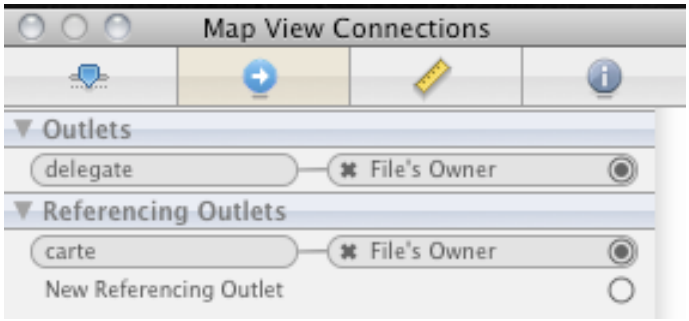
Remarque : le framework MapKit n'est pas inclus par défaut dans le projet.

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>

@interface TestMapViewController : UIViewController <MKMapViewDelegate> {

IBOutlet MKMapView * carte;
IBOutlet UILabel * latitudeLabel;
IBOutlet UILabel * longitudeLabel;
IBOutlet UILabel * hauteurLabel;
IBOutlet UILabel * largeurLabel;
}
@property(retain, nonatomic) MKMapView * carte;
@property(retain, nonatomic) UILabel * latitudeLabel;
@property(retain, nonatomic) UILabel * longitudeLabel;
@property(retain, nonatomic) UILabel * hauteurLabel;
@property(retain, nonatomic) UILabel * largeurLabel;

@end
```

Code source du contrôleur de vue

3. Sous Interface Builder, on attache les outlets du contrôleur aux éléments de la vue et on définit le délégué de la vue MKMapView comme étant le propriétaire du fichier.



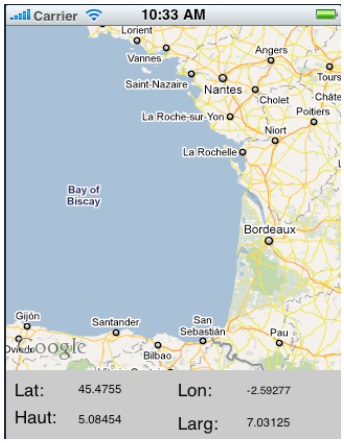


On ajoute les accesseurs des propriétés de la classe TestMapViewControlleur dans son fichier source. En nous appuyons sur la documentation ci-après, on ajoute la définition de la méthode -mapview : regionDidChangeAnimated : du protocole MKMapViewDelegate. Cette méthode est appelée chaque fois que la propriété region est modifiée. Le code est donc le suivant :

```
#import "TestMapViewController.h"

@implementation TestMapViewController
@synthesize carte, latitudeLabel, longitudeLabel, hauteurLabel, largeurLabel;

- (void) mapView: (MKMapView *) mapView
regionDidChangeAnimated:(BOOL) animated {
    MKCoordinateRegion region =carte.region;
    latitudeLabel.text=[NSString stringWithFormat:@"%g",region.center.latitude);
    longitudeLabel.text=[NSString stringWithFormat:@"%g",region.center.longitude);
    hauteurLabel.text=[NSString stringWithFormat:@"%g",region.center.latitudeDelta);
    largeurLabel.text=[NSString stringWithFormat:@"%g",region.center.longitudeDelta);
    }
```

Des erreurs se sont glissés dans le code. Pourrez-vous les corriger !!!
Ne pas oubliez de faire un carte.delegate=self dans la méthode -(void) viewDidLoad.

Le résultat de l'exécution de l'application est représenté sur la figure ci-dessous.



Propriété Région

La propriété region de la classe MKMapView est une structure de type MKCoordinateRegion composée de :

- center qui est une structure de type CLLocationCoordinate2D elle-même composée de :
  - latitude de type CLLocationDegrees ;
  - longitude de type CLLocationDegrees.
- span qui est une structure de type MKCoordinateSpan composée de :
  - latitudeDelta de type CLLlocationDegrees ;
  - longitudeDelta de type CLLocationDegrees.

Ainsi une région sur la carte est définie par son centre (latitude et longitude) et par sa taille, elle-même exprimée en écarts de latitude et de longitude. Un écart de latitude de un degré représente une hauteur de 111km. Un écart de longitude de un degré représente une largeur qui dépend de la latitude : 111km à l'équateur et 0 aux pôles.

Contrôler la zone affichée

Maintenant que nous savons extraire la zone affichée dans une instance de la classe MKMapView, nous allons compléter notre application Carte afin qu'elle nous permette de mémoriser une région pour y revenir plus tard.

Définition de l'interface

Nous ajoutons deux boutons sur l'interface utilisateur : "Definir Zone" et "Retrouvez Zone". Nous ajoutons ensuite une action pour chacun de ces boutons dans l'interface du contrôleur de vue de l'application, respectivement defineZone et retreiveZone. Il reste à lier les boutons et les actions sous Interface Builder.

-(IBAction) defineZone ;
-(IBAction) retrieveZone ;

Toujours dans l'interface du contrôleur de vue, on déclare une nouvelle propriété zone de type MKCoordinateRegion. Cette propriété n'étant pas une référence, elle doit être déclarée avec la clause assign au lieu de retain.
MKCoordinateRegion zone :

@property(assign, nonatomic) MKCoordinateRegion zone ;

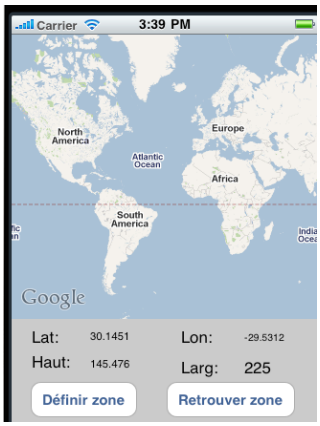
Code du contrôleur

Dans le fichier TestMapViewController.m, on synthétise les accesseurs de la nouvelle propriété zone, puis on définit les méthodes pour les actions.

Remarque : nous utilisons ici la méthode –setRegion :animated : de la classe MKMapView pour définir la région à visualiser sur la carte.

```
-(IBAction) defineZone{
        self.zone = carte.region;
}

-(IBAction) retrieveZone{
        [carte setRegion:self.zone animated:YES];
}
```

Résultat :



Partie 3. Appréhender les différents types de vue

ETAPE 3 (à réaliser)

- Rajoutez une instance de UISegmentedControl
- Dans « Jump to Definition » (CRTL + Clic) recherchez la propriété qui donne l'index du segment(ed control) sélectionné
- Ecrivez une méthode qui permet de récupérer et d'afficher dans la fenêtre de sortie l'index sélectionné (vous devez *caster* le sender pour qu'il puisse utiliser la propriété du segment(ed control))
- Vérifiez que votre programme fonctionne
- Allez à la définition de MKMapView. Quelles sont les différentes valeurs que peuvent prendre la propriété mapType ?
- Construisez une méthode qui permet de changer le style de map

... plus de détails pour vous aider.

La classe MKMapView dispose des propriétés suivantes pour modifier son comportement :

- mapType de type MKMapType ;
- scrollEnabled de type Booléen ;
- zoomEnabled de type Booléen.

scrollEnabled et zoomEnabled permettent d'autoriser respectivement le déplacement de la carte et le zoom par l'utilisateur. Ces propriétés ont la valeur YES par défaut.

MKMapTypeStandard, pour visualiser le plan ;
MKMapTypeSatellite, pour visualiser la vue satellite ;
MKMapTypeHybrid, pour visualiser la vue satellite augmentée d'information

On modifie l'application Carte pour que l'utilisateur puisse choisir entre les différents types de visualisation.

- *(IBAction) changeType : (id) sender ;* ➔*(.h)*

```
- (IBAction) changeType:(id)sender{                    ➔(.m)
        int val = [(UISegmentedControl *)sender selectedSegmentIndex];
        switch (val) {
                case 0:
                        carte.mapType=MKMapTypeStandard;
                        break;
```

```
        case 1:
            carte.mapType=MKMapTypeSatellite;
            break;
        case 2:
            carte.mapType=MKMapTypeHybrid;
            break;
    }}
```

Annoter la carte

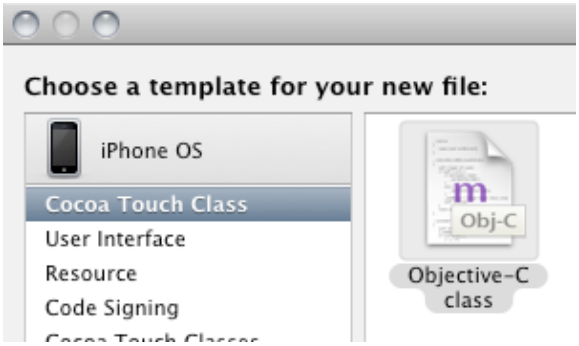
Nous allons terminer ce parcours du framework MapKit par la mise en œuvre du protocole MKAnnotation et de la méthode –addAnnotation: qui permettent d'ajouter sur la carte des marqueurs en forme d'épingle à tête.

Création d'une annotation

Il n'y a pas de classe spécifique pour contenir une annotation. N'importe quel objet fait l'affaire pourvu qu'il respecte le protocole MKAnnotation qui définit trois propriétés :

- coordinate, propriété obligatoire de ype CLLocationCoordinate2D, pour définir l'emplacement du marqueur sur la carte ;
- title et subTitle, propriétés optionnelles de type NSString * ; es chaînes de caractères sont affichées lorsque l'utilisateur touche le marqueur.

Sous Xcode, on crée une nouvelle classe Annotation dérivant de NSObject. Nous complétons ensuite l'interface de la classe pour y déclarer qu'elle adopte le protocole MKAnnotation et ses propriétés.




```
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>

@interface Annotation : NSObject
```

```
<MKAnnotation> {
CLLocationCoordinate2D coordinate;
NSString * title;
NSString * subTitle;
}
@property(assign,nonatomic) CLLocationCoordinate2D coordinate;
@property(retain,nonatomic) NSString * title;
@property(retain,nonatomic) NSString * subTitle;

@end
```

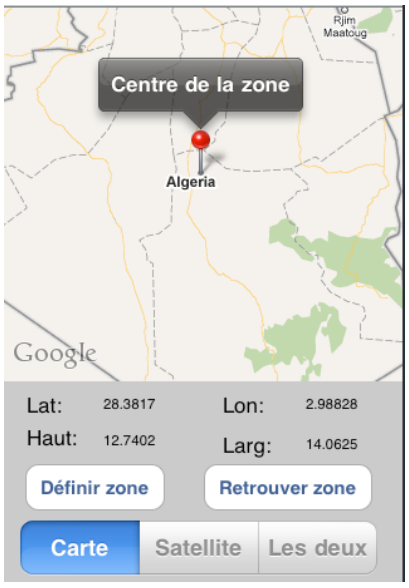
Il nous faut ensuite compléter la définition de la classe en définissant les accesseurs pour les propriétés dans le fichier Annotation.m. Cette classe est un réceptacle de données, elle ne contient pas d'autres méthodes.

```
#import "Annotation.h"
@implementation Annotation
@synthesize coordinate, title, subTitle;
@end
```

Afficher un marqueur

Sous Xcode, nous ouvrons le fichier TestMapViewController.m pour y déclarer la classe Annotation. Il faut modifier la méthode –defineZone pour ajouter un marqueur au centre de la classe.

```
#import "TestMapViewController.h"
#import "Annotation.h"


-(IBAction) defineZone{
self.zone = carte.region;
Annotation * annotation = [[Annotation alloc] init];
annotation.coordinate = self.zone.center;
annotation.title = @"Centre de la zone";
[carte addAnnotation:annotation];
[annotation release];
}
```



Résultat. Si nous touchons le marqueur défini en même temps que la zone, le texte s'affiche :

Partie 4. Finaliser l'application

ETAPE 4 (à réaliser)

- Réalisez l'application finale illustrée ci-dessous (application iOS de type *tabbed Application*).

... plus de détails pour vous aider.

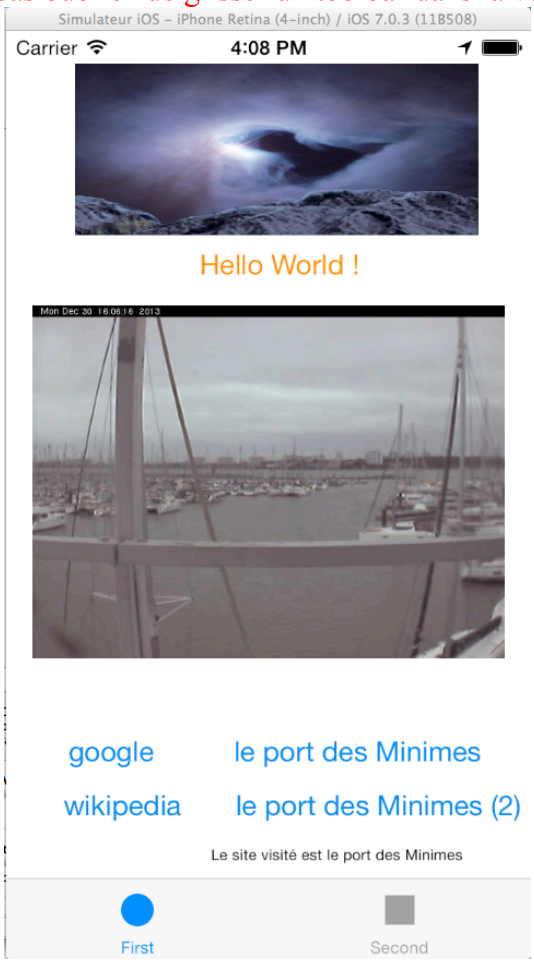
A vous de voir où il faut mettre ces différents lignes de code !! Elles permettent de créer des boutons, à partir du code, dans la toolbar que vous aurez positionnée auparavant dans le fichier nib de la vue. positionParis, positionLaRochelle sont des méthodes de type IBAction à écrire. showData, également de type IBAction restera une méthode vide pour l'instant.

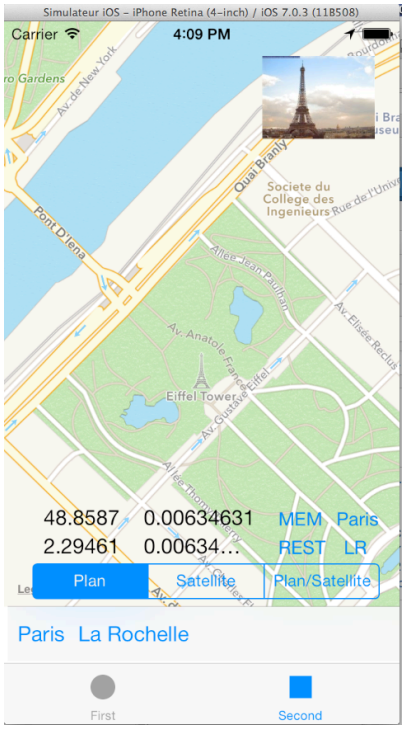
```
self.carte.showsUserLocation=YES;
self.carte.delegate=self;

UIBarButtonItem *boutonTour=[[UIBarButtonItem alloc] initWithTitle:@"Paris"
style:UIBarButtonItemStyleBordered target:self action:@selector(positionParis:)];

UIBarButtonItem *boutonStyle=[[UIBarButtonItem alloc] initWithTitle:@"Style"
style:UIBarButtonItemStyleBordered target:self action:@selector(positionLaRochelle:)];

UIBarButtonItem *boutonZoom=[[UIBarButtonItem alloc] initWithTitle:@"Zoom"
style:UIBarButtonItemStyleBordered target:self action:@selector(showData:)];

NSArray *listeDesBoutons=[[NSArray alloc] initWithObjects:boutonTour,boutonStyle, boutonZoom, nil];

self.toolbar.items=listeDesBoutons;
```

déclaration :

```
@property(nonatomic,retain) IBOutletUIToolbar *toolbar;
```

Ne pas oubliez de glisser un toolbar dans la vue et de déclarer le getter et le setter.





Partie 5. Une classe swift dans un projet objective-C

Création d'un classe swift AnnotationSwift (*indication : créer un nouveau fichier de type swift*)
Ele doit vérifier le protocole MKAnnotation. Valider la création du fichier *TestMap-Bridging-Header.h au moment de la création du fichier swift*

```
import Foundation
import MapKit
@objc class AnnotationSwift: NSObject, MKAnnotation {
    let title: String?
    let locationName: String
    let discipline: String
    let coordinate: CLLocationCoordinate2D

    init(title: String, locationName: String, discipline: String, coordinate: CLLocationCoordinate2D) {
        self.title = title
        self.locationName = locationName
        self.discipline = discipline
        self.coordinate = coordinate

        super.init()
    }

    var subtitle: String? {
        return locationName
    }
}
```

Vous pouvez ensuite utiliser cette classe à l'intérieur de votre programme de la manière suivante :
(N'oubliez pas d'effectuer l'import dans la classe objective-c (.m): `#import "TestMap-swift.h")`

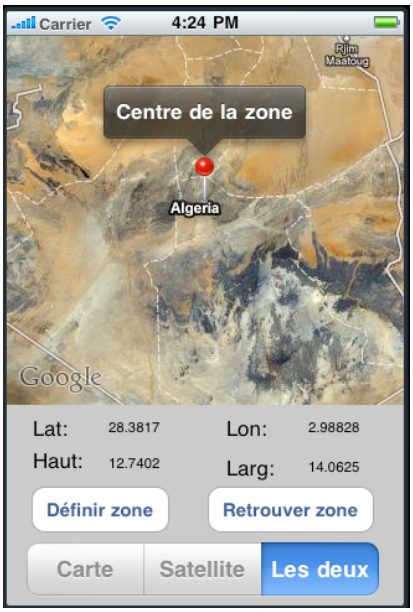
```
AnnotationSwift *annotationSwift=[AnnotationSwift alloc];
annotationSwift=[annotationSwift initWithTitle:@"Port des minimes" locationName: @"Capitainerie" discipline:
@"Plaisance" coordinate:region.center];

[self.myMapView addAnnotation:annotationSwift];
```

SYNTHESE

Nous connaissons maintenant les différentes technologies mises en œuvre pour la géolocalisation de l'appareil, et nous savons mettre en œuvre le framework Core Location (documentation précédente: le gestionnaire de géolocalisation, instance de la classe CLLocationManager, et son protocole de délégué CLLocationManagerDelegate). Nous avons en effet détaillé les classes décrivant la position de l'appareil :

- CLLocation pour la position sur le globe terrestre ;
- CLHeading pour l'inclinaison de l'appareil par rapport au Nord, géographique ou magnétique.

Enfin, dans ce document, nous avons appris, à utiliser le framework MapKit, la vue MKMapView et son délégué MKMapViewDelegate pour :



- visualiser et manipuler une carte ;
- connaitre la zone géographique visualisée ;
- modifier le mode de visualisation de la carte ;
- insérer des annotations sur la carte.

Annexe – Protocole MKMapViewDelegate

```
Tasks
Responding to Map Position Changes
              -mapView:regionWillChangeAnimated:
              -mapView:regionDidChangeAnimated:
Loading the Map Data
              -mapViewWillStartLoadingMap:
              -mapViewDidFinishLoadingMap:
              -mapViewDidFailLoadingMap:withError:
Tracking the User Location
              -mapViewWillStartLocatingUser:
              -mapViewDidStopLocatingUser:
              -mapView:didUpdateUserLocation:
              -mapView:didFailToLocateUserWithError:
Managing Annotation Views
              - mapView:viewForAnnotation:
              - mapView:didAddAnnotationViews:
              - mapView:annotationView:calloutAccessoryControlTapped:
Dragging an Annotation View
              - mapView:annotationView:didChangeDragState:fromOldState:
Selecting Annotation Views
              - mapView:didSelectAnnotationView:
              - mapView:didDeselectAnnotationView:
Managing Overlay Views
              - mapView:viewForOverlay:
              - mapView:didAddOverlayViews:


mapView:regionDidChangeAnimated:
Tells the delegate that the region displayed by the map view just changed.
- (void)mapView:(MKMapView *)mapView regionDidChangeAnimated:(BOOL)animated
Parameters
mapView
```

The map view whose visible region changed.
animated
If YES, the change to the new region was animated.
Discussion
This method is called whenever the currently displayed map region changes. During scrolling, this method may be called many times to report updates to the map position. Therefore, your implementation of this method should be as lightweight as possible to avoid affecting scrolling performance.

Tasks
Position Attributes
- coordinateÊÊrequired property
- setCoordinate:
Title Attributes
- title
- subtitle

coordinate
The center point (specified as a map coordinate) of the annotation. (required) (read-only)

@property (nonatomic, readonly) CLLocationCoordinate2D coordinate

subtitle
Returns the string containing the annotationÕs subtitle.
- (NSString *)subtitle
Return Value
The subtitle string.

itle
Returns the string containing the annotationÕs title.
- (NSString *)title
Return Value : The title string.

Code pour la gestion des lieux de proximité inscrits dans le fichier json.

```
- (NSDictionary *) markersFromJSON:(NSData *) googleJson
    {

        NSError *jsonError=nil;
        NSDictionary *result =[NSJSONSerialization
                               JSONObjectWithData:googleJson
                               options:0L
                               error:&jsonError];

        if(result == nil || jsonError != nil) {
            return nil;
        }
        return result;
    }

- (IBAction) showData:(id)sender {
    Annotation *annotation;
    MKCoordinateRegion region;

    NSString *path=[[NSBundle mainBundle] pathForResource:@"googlemarkers"
ofType:@"json"];

    NSData *googleData=[NSData dataWithContentsOfURL:[NSURL fileURLWithPath:path]];

    NSDictionary *googleMarkers=[self markersFromJSON:googleData];
    NSArray *markerList=[googleMarkers objectForKey:@"markers"];
    for (NSDictionary *cMarker in markerList)
    {
        NSLog(@"Marqueur:%@(%.8f; %.8f)",
              [cMarker objectForKey:@"html"],
              [[cMarker objectForKey:@"lat"] doubleValue],
              [[cMarker objectForKey:@"lng"] doubleValue]);
```

```
    }

    NSLog(@"latitude=%f",myMapView.region.center.latitude);
    NSLog(@"longitude=%f",myMapView.region.center.longitude);

    markerList=[self filterListByDistance:markerList
centerOnLat:myMapView.region.center.latitude andLong:myMapView.region.center.longitude
maxDistance:10000.0];

    region=MKCoordinateRegionMakeWithDistance(region.center,10000.0,10000.0);

    for (NSDictionary *cMarker in markerList)
    {
        annotation=[[Annotation alloc] init];
        region.center.latitude= [[cMarker objectForKey:@"lat"] doubleValue];
        region.center.longitude=[[cMarker objectForKey:@"lng"] doubleValue];
        annotation.coordinate=region.center;
        annotation.title=[cMarker objectForKey:@"html"];
        [self.myMapView addAnnotation:annotation];
        [self.myMapView setRegion:region animated:YES];
    }
}


- (NSArray *) filterListByDistance:(NSArray *) markers centerOnLat:(double) lat
andLong:(double) lng maxDistance:(double) cutoff {

    NSMutableArray *result=[NSMutableArray array];

    for (NSDictionary *cMarker in markers)
    {
        double latitude=[[cMarker objectForKey:@"lat"] doubleValue];
        double longitude=[[cMarker objectForKey:@"long"] doubleValue];

        if ([self pythagorianDistanceFromLat:latitude flng:longitude
                                        slat:lat  slng:lng] < cutoff)
        {
            NSLog(@"%f",[self pythagorianDistanceFromLat:latitude flng:longitude slat:lat
slng:lng]);
            [result addObject:cMarker];
        }
    }

    return result;
}


/**
 * pythagorianDistanceFromLat:flng:slat:slng:
 */

- (double) pythagorianDistanceFromLat:(double)firstlat flng:(double)firstlng
                           slat:(double)seclat slng:(double)seclng
{


    CLLocation *location1=[[CLLocation alloc] initWithLatitude:firstlat
longitude:firstlng];
    CLLocation *location2=[[CLLocation alloc] initWithLatitude:seclat
longitude:firstlng];
    double distance=[location1 distanceFromLocation:location2];

    /*
    sqrt(
         pow( abs((seclat * 3600) - (firstlat * 3600)), 2 )
         + pow( abs((seclng * 3600) - (firstlng * 3600)), 2 )
         );
    return (distanceSecondes * 31); */
    NSLog(@"distance=%f",distance);
    return (distance);
}
```