MakeUrSport

Dossier de programmation

Suivi d'activités sportives sur Android

Introduction

Dans ce document, nous allons revenir sur le code de notre application. Nous allons ainsi montrer non plus, la philosophie et l'étude que nous avons réalisée de MakeUrSport, mais la façon dont nous l'avons implémenté. Ainsi, avant de nous plonger dans le code, nous allons tout d'abord présenter les concepts fondamentaux de la programmation Android.

Android est un système d'exploitation mobile, qui repose essentiellement sur une machine virtuelle Java. Pour pouvoir développer des applications pour Android, nous devons donc utiliser le langage de développement Java couplé au SDK (Software Development Kit) Android. Ce SDK nous offre des méthodes, et des classes à hériter pour interagir avec la plateforme Android.

Ainsi, la partie visible de l'application est composée d'Activity et de Fragment. Une Activity est la partie de l'application qui permet d'afficher des informations à l'écran et d'interagir avec l'utilisateur. Un fragment est un composant d'une Activity, qui gère son propre affichage, et possède son propre cycle de vie. Il permet de ne pas surcharger une Activity en affichant uniquement les informations utiles à l'écran.

Ensuite, il faut savoir que sous Android, tout le code que l'on place dans notre Activity et dans notre Fragment se déroule dans le processus simplifié principal (aussi appelé Thread UI). Il faut donc faire attention à ne pas surcharger le thread principal de l'application avec des actions lourdes. Pour cela nous avons choisi d'utiliser un outil très pratique, l'AsyncTask. Cette AsyncTask permet de créer un nouveau thread lors de son exécution, et d'effectuer certaines actions une fois que celle-ci se termine. Pour vérifier qu'il n'y a pas de surcharge sur le Thread UI, nous avons utilisé un outil appelé StrictMode sur un de nos téléphones de développement. Ce StrictMode permet d'empêcher tout traitement lourd dans le thread principal, et nous notifie s'il y a un problème.

De plus, il faut savoir que d'après notre cahier des charges, nous devions viser plus de 80% du marché des téléphones Android, et donc faire en sorte que notre application fonctionne sur des téléphones sous Android 2.2. Or, plusieurs fonctionnalités que nous souhaitions utiliser, c'est-à-dire les fragments et les différentes règles d'interface graphique proposées par Android, n'était pas disponible nativement sous ces versions. Nous avons donc dû utiliser la librairie de rétrocompatibilité d'Android, que nous avons couplé à la bibliothèque Action Bar Sherlock, qui nous a permis de respecter ces règles.

On trouve dans des projets Android, en plus des classes java, plusieurs fichiers XML. Des fichiers layout qui permettent de définir l'interface graphique. On trouve aussi un fichier strings.xml qui définit toutes les chaînes de caractère que nous affichons à l'utilisateur, ou encore des fichiers définissant les préférences, ou même certains styles appliqués à notre interface graphique. L'utilisation de ces divers fichiers XML permet de simplifier la gestion du code en Java et d'avoir un produit plus modulable.

Sommaire

Après la partie de programmation, nous avons obtenu une architecture composée de 23 classes réparties dans 4 paquetages.

Paquetage 1 : com.makeursport – 12 classes

Ce paquetage comporte les principales classes de notre application, c'est-à-dire les Activity, les Fragment, et les Dialog (boîtes de dialogues).

CourseEnCours : Le fragment qui permet d'afficher une course en cours (générée ou libre).

CourseFragment : Le fragment qui permet d'afficher une course depuis l'historique.

GenerationParcoursATask : AsyncTask qui permet la génération d'un parcours.

HistoriqueAdapter : Classe permettant de lier une Collection de Course à une vue (utilisée pour lister les courses dans l'historique).

HistoriqueFragment : Fragment permettant d'afficher la liste des courses (l'historique).

MainActivity : L'activité principale de notre application : elle permet de gérer les transitions entre les différents fragments.

MenuFragment : Fragment du menu qui s'affiche derrière notre fragment en cours.

MyMapFragment : Fragment contenant une carte GoogleMap, et qui s'utilise dans les classes CourseEnCours et CourseFragment.

MySSLSocketFactory : Classe permettant d'interagir avec l'API en ligne de Google de façon sécurisée, pour pouvoir générer un parcours.

ParcoursDialog : Boîte de dialogue permettant de proposer la génération d'un parcours à l'utilisateur.

SportDialog : Boîte de dialogue permettant de proposer à l'utilisateur de choisir son sport pratiqué.

SportifDialogActivity : Boîte de dialogue permettant de définir les principales caractéristiques du Sportif.

Paquetage 2 : com.makeursport.database – 2 classes

Ce paquetage contient les classes propres à notre base de données.

CourseDBHelper : Classe qui décrit la base de données, et nous permet d'interagir avec celle-ci.

CourseDS : Classe qui nous permet de faciliter les interactions avec notre base de données.

Paquetage 3 : com.makeursport.gestionCourse – 3 classes

Ce paquetage contient toutes les classes issues de la phase d'Analyse/Conception, il correspond à notre composant GestionCourse.

Course : Classe définissant une course, ses caractéristiques et ses méthodes.

EtatCourse : Enumération permettant de définir l'état d'une course (Course lancée, en pause ou arrêtée).

GestionnaireCourse : Classe permettant de gérer une course en la démarrant ou l'arrêtant.

GestionnaireHistorique : Classe permettant d'interagir avec l'historique et donc avec la base de données.

Sport : Enumération permettant de définir le sport pratiqué.

Sportif : Classe qui permet de définir l'utilisateur et ses caractéristiques.

Paquetage 4 : com.makeursport.preferences – 3 classes

Ce paquetage contient toutes les classes utiles pour la gestion des préférences.

FloatEditTextPreference : Composant d'une préférence utile pour utiliser des préférences sous forme de réel.

IntEditTextPreference : Composant d'une préférence utile pour utiliser des préférences sous forme d'entier.

Settings : Activity qui permet de définir et de modifier les préférences de notre application.

Extraits de code

Classe CourseEnCours

La classe CourseEnCours est un Fragment, qui représente la principale fonctionnalité de notre application : proposer le suivi d'activités sportives en temps réel.
Elle hérite de la classe SherlockFragment car c'est un fragment, et que l'on utilise la bibliothèque ActionBar Sherlock qui nous permet de suivre les recommandations de l'interface graphique sous Android.

Elle implémente LocationListener, qui nous permet d'être notifiés lorsque l'utilisateur change de position. En fait, nous instancions le LocationManager dans notre `onCreateView`, qui est la méthode qui se lance lors de l'affichage de notre Fragment, et nous demandons à recevoir des notifications quand l'utilisateur change de position. Ces notifications sont faites par le biais de la méthode onLocationChanged, qui est appelée dès que l'utilisateur bouge.

Elle implémente également GpsStatus.Listener car nous souhaitons aussi être tenu au courant de l'état du signal GPS sur le téléphone. Ainsi, en implémentant cette classe, nous sommes notifiés dès qu'il y a des nouveautés sur l'état du GPS, grâce à l'appel de la méthode onGpsStatusChanged.

Enfin, elle implémente TextToSpeech.OnInitListener de façon à pouvoir utiliser les fonctions de dictée vocale d'Android.

```
package com.makeursport;
/**
 * Activité de la CourseEnCours
 * Gère l'affichage des infos d'une course, géré par un @{link GestionnaireCourse}
 * Implémente {@link LocationListener} de manière à être informé lorsque
 * la position de l'utilisateur change, et {@link GpsStatus.Listener} pour être
 * informé de l'état du GPS en temps réel
 * @author L'équipe MakeUrSport
 *
 */
public class CourseEnCours extends SherlockFragment implements
LocationListener,Listener,TextToSpeech.OnInitListener {
	/**
	 * Tag utilisé pour le LOGCAT (affichage de message quand on debug)
	 */
	private final String LOGCAT_TAG=this.getClass().getCanonicalName();
	/**
	 * Le gestionnaire de la course en cours
	 */
	private GestionnaireCourse gestCourse;
	/**
	 * Le location manager gérant la localisation
	 */
	private LocationManager locationManager;
	/**
	 * Fragment contenant la carte Google Maps
	 */
	private MyMapFragment mapFragment;

	/**
	 * Valeur pour savoir si on reçoit bien les infos du GPS ou pas
	 */
	private boolean gpsActif=false;
	/**
	 * Contient le handler qui met à jour la vue toute les x ms
	 */
	private Handler updateHandler;
	/**
	 * Runnable exécutée toute les x ms par le Handler
	 */
```

```
	private final Runnable runnableMiseAJour = new Runnable() {
		public void run() {
			Log.d(LOGCAT_TAG+"_runnable", "Runnable.run()");
			int rappelerDans = 0;
			if(gpsActif) {
				mettreAJourView();
				rappelerDans=490;
			} else {
				Log.d(LOGCAT_TAG+"_runnable","GPS inactif, pas de mise a
jour");
				rappelerDans=1000;
			}
			updateHandler.postDelayed(this, rappelerDans);
		}
	};
	/**
	 * Le menu (dans l'actionbar) de notre activity
	 */
	private Menu menu;
	/**
	 * La vue de notre fragment. Permet d'interagir avec elle après le onCreateView
	 */
	private RelativeLayout vuePrincipale;

	/**
	 * On enregistre ici la dernière localisation connue.
	 */
	private Location dernierePosition = null;
	/**
	 * Texte à prononcer toutes les 5 mn pendant une course
	 */
	private TextToSpeech tts;

	/**
	 * String utilisée pour partager un parcours dans un Bundle
	 */
	public static final String PARCOURS = "com.makeursport.PARCOURS";
	/**
	 * String utilisée pour partager un Sport dans un intent
	 */
	public static final String SPORT_INTENT = "com.makeusrport.SPORTINTENT";
	/**
	 * Numéro de requête utilisé lors du lancement de {@link SportifDialogActivity}
	 */
	public static final int DIALOG_SPORTIF_REQUEST_CODE =7;
	/**
	 * Numéro de requête utilisé lors du lancement de {@link SportDialog}
	 */
	public static final int DIALOG_SPORT_REQUEST_CODE = 8;

	@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
		super.onCreateView(inflater, container, savedInstanceState);
		this.gestCourse=new GestionnaireCourse();

		this.tts = new TextToSpeech(this.getSherlockActivity(),this);
		SharedPreferences pref =
PreferenceManager.getDefaultSharedPreferences(this.getActivity());

	//On vérifie si le sportif existe, s'il n'existe pas, on ouvre un dialog pour lui
demander d'entrer les infos
	if(!pref.contains(this.getString(R.string.SP_annee_naissance))
			&& !pref.contains(this.getString(R.string.SP_poids))
			&& !pref.contains(this.getString(R.string.SP_taille))) {
		Intent demarrerDialogSportif = new
Intent(this.getActivity(),SportifDialogActivity.class);
		//Le startActivityForResult démarre le dialog mais se met en attente d'une
réponse
		//De façon à pouvoir traiter si l'utilisateur rentre les infos comme prévu
		this.startActivityForResult(demarrerDialogSportif,
DIALOG_SPORTIF_REQUEST_CODE);
```

```
        }

              //On met dans sa place réservée, le MapFragment
              FragmentManager fm = getChildFragmentManager();
              //Pour créer un nouveau MyMapFragment, on utilise newInstance
              //Plutôt qu'un constructeur. Conseillé par la doc
              this.mapFragment = MyMapFragment.newInstance(new
GoogleMapOptions().zoomControlsEnabled(false));
              fm.beginTransaction()
              .replace(R.id.mapfragment_location, mapFragment)
              .commit();

           if(this.getArguments() != null && this.getArguments().containsKey(PARCOURS)) {
                    ArrayList<LatLng> parcours =
this.getArguments().getParcelableArrayList(PARCOURS);
              this.mapFragment.setParcours(parcours);
           }

              //Initialisation du LocationManager
        this.locationManager = (LocationManager)this.getActivity()
                    .getSystemService(Context.LOCATION_SERVICE);
        //On lance le GPS Status Listener, pour qu'on nous dise quand le GPS marche
              this.locationManager.addGpsStatusListener(this);
              this.setHasOptionsMenu(true);//On signale que l'on veut recevoir les appels
concernant le menu de l'action bar
              this.vuePrincipale= (RelativeLayout)
inflater.inflate(R.layout.activity_course_en_cours, container, false);
         return this.vuePrincipale;
       }


    /**
     * Met à jour la vue, en affichant les infos de la course en cours.
     */
    private void mettreAJourView() {
       Log.d(LOGCAT_TAG, "Mise à jour de la vue");
       Course course = gestCourse.getCourse();
       TextView vitMoy_tv = (TextView)
vuePrincipale.findViewById(R.id.TV_vit_moyenne_valeur);
       vitMoy_tv.setText(course.getVitesseMoyenne() +
this.getString(R.string.unite_vitesse));
       TextView vitReel_tv = (TextView)
vuePrincipale.findViewById(R.id.TV_vit_reel_valeur);
       vitReel_tv.setText(course.getVitesseReelle() +
this.getString(R.string.unite_vitesse));
       TextView calories_tv = (TextView)
vuePrincipale.findViewById(R.id.TV_calories_valeur);
       calories_tv.setText(course.getCaloriesBrulees() + "");
       TextView distance_tv = (TextView)
vuePrincipale.findViewById(R.id.TV_distance_valeur);
       distance_tv.setText(course.getDistanceArrondi() +
this.getString(R.string.unite_distance));
       TextView duree_tv = (TextView) vuePrincipale.findViewById(R.id.TV_duree);
       long duree = course.getDuree();
       duree_tv.setText(String.format("%d:%02d:%02d", duree/(3600), (duree%3600)/(60),
(duree%(60))));
    }
    /**
     * Demande la mise à jour de la localisation.
     *
     * @see CourseEnCours#onLocationChanged()
     */
    private void demarrerLocationListener() {
              this.locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0,
0,this);//GPS, dès qu'on peut (toutes les 0s, 0m), on prévient this
              if(!this.gpsActif) {
                    this.signalerGPSInactif();
              }
    }
    /**
     * Appelé quand on sait que le signal GPS est inactif
     */
```

```
    private void signalerGPSInactif() {
       Log.d(LOGCAT_TAG,"En attente du signal GPS...");
       View voile = this.vuePrincipale.findViewById(R.id.voileInactif);
       AlphaAnimation animTransparence = new AlphaAnimation(0.0f,1.0f);
       animTransparence.setDuration(500);
       voile.setAnimation(animTransparence);
       animTransparence.startNow();
       voile.setVisibility(View.VISIBLE);
       this.gpsActif=false;


this.vuePrincipale.findViewById(R.id.TV_messageInactif).setVisibility(TextView.VISIBLE);
    }
    /**
     * Appelé quand le signal GPS devient actif
     */
    private void signalerGPSActif() {
       //Le GPS vient d'être déclaré actif, on démarre la course.
       if(this.gestCourse.getEtatCourse() == EtatCourse.CourseLancee) {//Si jamais on est
en CourseLancee et qu'on avait pas le GPS activé
              //Ca veut dire qu'on avait lancé la course et qu'on attendait l'activation
du signal GPS
              this.gestCourse.demarrerCourse(this.getSherlockActivity());
       }
       this.gpsActif=true;
       cacherVoile();
    }
    /**
     * Cache le "voile" du GPS inactif
     */
    private void cacherVoile() {
       View voile = this.vuePrincipale.findViewById(R.id.voileInactif);
       if(voile.getVisibility()==View.VISIBLE){
              voile.setVisibility(View.GONE);

       this.vuePrincipale.findViewById(R.id.TV_messageInactif).setVisibility(View.GONE);
       }
    }
    /**
     * Met fin à la mise à jour du location listener
     * (démarré à l'aide du {@link CourseEnCours#demarrerLocationListener})
     */
    private void stopperLocationListener() {
       locationManager.removeUpdates(this);
    }

    /**
     * Méthode appelée quand la position de l'utilisateur change
     * @param loc la position de l'utilisateur
     */
      public void onLocationChanged(Location loc) {
              if(this.gestCourse.getEtatCourse()== EtatCourse.CourseLancee) {
                     if(this.dernierePosition!=null) {
                            Log.d(LOGCAT_TAG + "_OnLocationChanged", "dernierePosition!
=null");
                            float vitesseReelle = loc.getSpeed();
                            this.gestCourse.mettreAJourCourse(vitesseReelle,
this.dernierePosition, loc);
                     }
                     this.dernierePosition=loc;


this.mapFragment.mettreAJourCarte(loc.getLatitude(),loc.getLongitude(),loc.getBearing(),
MyMapFragment.ZOOM_LEVEL);
                     if(!this.gpsActif) {
                            this.signalerGPSActif();
                     }
                     //On prononce les performances du sportif toutes les 5 mn
                     //On vérifie donc si la durée de la course en minutes est un multiple
de 5
                     boolean parlePrefs = PreferenceManager.getDefaultSharedPreferences(
```

```
		this.getSherlockActivity()).getBoolean(this.getString(R.string.SP_infos_audio),
false);
				if ((this.gestCourse.getCourse().getDuree()) % (60*5) == 0
						&& this.gestCourse.getCourse().getEtatCourse() ==
EtatCourse.CourseLancee
						&& parlePrefs
						&& this.gestCourse.getCourse().getDuree() != 0) {
					Log.d(LOGCAT_TAG, "parle !" + parlePrefs);
					speakTTS();
				} else {
					Log.d(LOGCAT_TAG, "pas de dictée vocale" + parlePrefs);
				}
			}
			else {
				Log.d(LOGCAT_TAG, "Course en pause... Les infos ne sont pas mise à
jour...");
			}
		}
		public void onProviderDisabled(String provider) {
			if(provider.equals(LocationManager.GPS_PROVIDER)) {
				Toast.makeText(this.getSherlockActivity(),
this.getString(R.string.message_erreur_desactivation_GPS), Toast.LENGTH_LONG).show();
				this.gestCourse.stopperCourse();
			}
		}
		public void onProviderEnabled(String provider) {
			// Rien à faire ici
		}
		public void onStatusChanged(String provider, int status, Bundle extras) {
			Log.d(LOGCAT_TAG+"_onStatusChanged", "Provider:"+provider + " status:" +
status);
		}



		public void onGpsStatusChanged(int event) {
			String sEvent="";
			switch(event) {
			case GpsStatus.GPS_EVENT_FIRST_FIX :
				this.signalerGPSActif();
				sEvent="GPS_EVENT_FIRST_FIX";
				break;
			case GpsStatus.GPS_EVENT_SATELLITE_STATUS:
				sEvent="GPS_EVENT_SATELLITE_STATUS";
				Iterator<GpsSatellite> it =
this.locationManager.getGpsStatus(null).getSatellites().iterator();
				int i=0;
				while(it.hasNext()) {
					i++;it.next();
				}
				sEvent+=" (" + i + " satellites";
				break;
			case GpsStatus.GPS_EVENT_STARTED:
				sEvent="GPS_EVENT_STARTED";
				break;
			case GpsStatus.GPS_EVENT_STOPPED:
				sEvent="GPS_EVENT_STOPPED";
				break;
			default:
				sEvent="" + event;
			}
			Log.v(LOGCAT_TAG+"_onGPSStatusChanged", "Received : " + sEvent);
		}

		/**
		 * Lors d'un clic sur les boutons du menu
		 */
		@Override
		public boolean onOptionsItemSelected(MenuItem item) {
			switch(item.getItemId()) {
			case R.id.BT_playpause:
```

```
                        if(!
this.locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
                                Toast.makeText(this.getSherlockActivity(),
this.getString(R.string.GPS_desactive), Toast.LENGTH_LONG).show();
                        }

                        if(this.gestCourse.getEtatCourse()==EtatCourse.CourseArretee) {//Si
la course est arrétée : demarrage
                                this.gestCourse.demarrerCourse(this.getSherlockActivity());
                                this.mapFragment.initialiserCartePourNouvelleCourse();
                                this.mapFragment.mettreModeCourse();
                                this.updateHandler = new Handler();
                        this.updateHandler.post(runnableMiseAJour);
                        this.swapIcons(this.gestCourse.getEtatCourse());
                        this.demarrerLocationListener();
                        }
                        else if(this.gestCourse.getEtatCourse()==EtatCourse.CourseEnPause){//
Si la course est en pause : démarrage
                                this.gestCourse.reprendreCourse();
                                this.mapFragment.mettreModeCourse();
                        this.swapIcons(this.gestCourse.getEtatCourse());
                        this.updateHandler.post(runnableMiseAJour);
                        this.mapFragment.demanderNouveauPolyline();
                        }
                        else {//Sinon : la course est en cours, mettre en pause.
                                this.gestCourse.mettreEnPauseCourse();
                                this.mapFragment.arreterModeCourse();
                                this.swapIcons(this.gestCourse.getEtatCourse());
                        this.updateHandler.removeCallbacks(runnableMiseAJour);
                        }
                        break;
                case R.id.BT_stop:
                        if(this.gestCourse.getEtatCourse() != EtatCourse.CourseArretee) {
                                this.gestCourse.stopperCourse();
                        this.swapIcons(this.gestCourse.getEtatCourse());
                        if(dernierePosition != null) {
                                LatLng dernPos = new LatLng(dernierePosition.getLatitude(),
dernierePosition.getLongitude());

        this.gestCourse.getCourse().sauvegarderCourse(this.getSherlockActivity(),dernPos);
                        }
                        this.cacherVoile();
                                this.mapFragment.arreterModeCourse();
                        this.stopperLocationListener();
                        this.updateHandler.removeCallbacks(runnableMiseAJour);
                        }
                        break;
                case R.id.BT_sport:
                        Intent i = new Intent(this.getSherlockActivity(), SportDialog.class);
                        getActivity().startActivityForResult(i,DIALOG_SPORT_REQUEST_CODE);
                        break;
                //Si on appuie sur l'icone de l'appli
                case android.R.id.home:
                        ((SlidingFragmentActivity) getActivity()).toggle();
                        return true;
                default:
                        return super.onOptionsItemSelected(item);
                }
                return false;
        }

        /**
         * Echange l'icone play/pause en fonction de l'état de la course
         * @param etat l'état dans lequel vient tout juste de rentrer de la course
         */
        private void swapIcons(EtatCourse etat) {
                switch(etat) {
                case CourseArretee:
                case CourseEnPause:
                        menu.findItem(com.makeursport.R.id.BT_playpause)
                        .setIcon(this.getResources().getDrawable(R.drawable.ic_action_play));
                        break;
```

```
                case CourseLancee:
                        menu.findItem(com.makeursport.R.id.BT_playpause)

.setIcon(this.getResources().getDrawable(R.drawable.ic_action_pause));
                        break;

                }
        }
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
                inflater.inflate(R.menu.activity_course_en_cours, menu);
        this.menu=menu;
                changeSportIcone(gestCourse.getCourse().getSport());
    }

        /**
         * Est appelé dès que la dialog {@link SportifDialogActivity} ou {@link
SportDialog} est fermé.
         *
         * Plus précisement, onActivityResult est appelée dès qu'une activity démarré par
         * startActivityForResult (cf {@link CourseEnCours#onCreate()}) est fermée.
         * Elle sert à traiter le résultat retourné par cette activity.
         * Ici le clic sur le bouton cancel, ou le clicc sur le bouton confirm
         * @param requestCode le code requête mis en place lors du lancement de l'activity
         * @param resultCode le code retourné par l'activity
         * @param data les infos en plus retournées par l'activity
         */
        @Override
        public void onActivityResult(int requestCode, int resultCode, Intent data) {
                Log.v(this.LOGCAT_TAG, "Starting OnActivityResult from requestCode" +
requestCode);
                super.onActivityResult(requestCode, resultCode, data);
                if(requestCode==DIALOG_SPORTIF_REQUEST_CODE) {
                        switch(resultCode) {
                                case Activity.RESULT_CANCELED:
                                        Log.d(LOGCAT_TAG, "Le sportif ne veut pas rentrer
d'info : fermeture de l'appli");
                                        this.getActivity().finish();
                                        break;
                                case Activity.RESULT_OK://Si on a un retour ok


this.gestCourse.getCourse().setUser(Sportif.fromPrefs(this.getSherlockActivity()));
                                        break;
                        }
                } else if (requestCode == DIALOG_SPORT_REQUEST_CODE){
                        switch(resultCode) {
                        case Activity.RESULT_OK:
                                int sport =data.getExtras().getInt(SPORT_INTENT);
                                this.gestCourse.getCourse().setSport(Sport.getSport(sport));
                                this.changeSportIcone(Sport.getSport(sport));
                                break;
                        }
                }
        }

        /**
         * Change l'icone du sport
         * @param sport le sport à afficher
         */
        private void changeSportIcone(Sport sport) {
                int res=R.drawable.ic_action_course;
                switch(sport) {
                case COURSE:
                        break;
                case ROLLER:
                        res=R.drawable.ic_action_roller;
                        break;
                case VELO:
                        res=R.drawable.ic_action_velo;
                        break;
                }
```

```
                menu.findItem(com.makeursport.R.id.BT_sport)
                .setIcon(res);
        }
        @Override
        public void onDestroyView() {
                super.onDestroyView();
                Log.d(LOGCAT_TAG, "Suppression du GPSStatusListener");
                if(this.locationManager != null)
                        this.locationManager.removeGpsStatusListener(this);
                if(this.gestCourse != null && (this.gestCourse.getEtatCourse() ==
EtatCourse.CourseLancee || this.gestCourse.getEtatCourse()==EtatCourse.CourseEnPause)) {
                LatLng dernPos = new LatLng(dernierePosition.getLatitude(),
dernierePosition.getLongitude());

        this.gestCourse.getCourse().sauvegarderCourse(this.getSherlockActivity(),dernPos);
                }
                if(this.tts != null)
                        this.tts.stop();
        }

        public void onInit(int status) {
                if (status == TextToSpeech.SUCCESS) {
                        int result = this.tts.setLanguage(Locale.FRANCE);
                        if (result == TextToSpeech.LANG_MISSING_DATA || result ==
TextToSpeech.LANG_NOT_SUPPORTED) {
                                Log.e("TTS", "Langue non supportée");
                        }
                }
        }

        /**
         * Parle en TTS la phrase de message pour indiquer à l'utilisateur comment se
déroule sa course
         */
        private void speakTTS() {
                long d = this.gestCourse.getCourse().getDuree();
                String duree = String.format((Locale)null, "%d:%02d:%02d", d/(3600), (d
%3600)/(60), (d%(60)));
                String txt =
this.getString(R.string.message_tts,this.gestCourse.getCourse().getDistanceArrondi(),dure
e, this.gestCourse.getCourse().getVitesseMoyenne());
                tts.speak(txt, TextToSpeech.QUEUE_FLUSH, null);
        }
}
```

Classe GenerationParcoursATask

Cette classe est une AsyncTask qui nous permet de générer un parcours. Une AsyncTask permet d'effectuer les traitements longs en dehors du processus de l'interface graphique, de manière à ne pas bloquer celle-ci. Ici, le traitement long est la récupération du parcours (et donc la requête HTTP à Google Maps), et le déchiffrage de celle-ci. Une AsyncTask est toujours une classe paramétrée, qui prends trois types en paramètre : le type des données que doit manipuler l'AsyncTask (ici un LatLng), le type de données que doit renvoyer l'AsyncTask pendant qu'elle exécute son traitement (ici inutile donc Void), et enfin le type de données renvoyé par notre AsyncTask, et donc ici une ArrayList<LatLng> qui contient notre parcours constitué d'une liste de points de localisation (LatLng).

Notre AsyncTask est composée de 2 principales méthodes, dont une qui s'exécute arrière-plan et une sur le thread principal. La première est la méthode doInBackground. Cette méthode s'exécute donc en arrière-plan, et prend comme argument un LatLng représentant la position actuelle de l'utilisateur et permet d'effectuer la définition du point d'arrivée, la demande à l'API en ligne Google Maps via une requête HTTP de générer un parcours entre notre position actuelle et le point d'arrivée généré plus tôt. Enfin, ce parcours est décodé à l'aide de notre méthode decodeEncryptedGeopoints, pour retourner le parcours généré, puisque le résultat de la requête retourne des points de localisation cryptés.

La deuxième méthode importante d'une AsyncTask est l'onPostExecute. Cette méthode est appelée dès lors que le doInBackground est fini, et prend comme paramètre ce que retourne ce dernier, donc une liste de LatLng. Cette méthode s'exécute par contre dans le thread principal, et permet d'interagir avec notre application en elle-même. Ici, nous créons une nouvelle Intent (qui nous permet de passer des informations entre Fragment et Activity), dans laquelle nous mettons le parcours généré. Ensuite, nous demandons à changer de Fragment affiché pour une nouvelle CourseEnCours, qui prendra en compte le parcours généré.

```
package com.makeursport;
/**
 * ASyncTask qui permet de gérer la génération de parcours
 * en arrière-plan, puis affiche une CourseEnCours avec le parcours généré
 * @author L'équipe MakeUrSport
 */
public class GenerationParcoursATask extends
AsyncTask<LatLng,Void,ArrayList<LatLng>> {
	private final static String LOGCAT_TAG = "GenerationParcoursDialog";
	private MainActivity context;
	/**
	 * La distance du parcours à générer
	 */
	private float dist;
	/**
	 * Création de l'AsyncTask
	 * @param distance la distance que l'utilisateur souhaite parcourir
	 * @param context le contexte de l'activité en cours
	 */
	public GenerationParcoursATask(float distance, MainActivity context) {
		this.dist = (distance/2)*0.90f;
		this.context=context;
	}

	@Override
	protected ArrayList<LatLng> doInBackground(LatLng... ptDepart) {
		ArrayList<LatLng> geoPoints = null;
```

```
            //On génère un angle aléatoire dont la mesure est comprise entre 0
et 360°
            Random r = new Random();
            int angleAleatoire = r.nextInt(360);

            LatLng arrivee = pointDestination(ptDepart[0], this.dist,
angleAleatoire);
            Log.d(LOGCAT_TAG, "ptDpartLAT:" + ptDepart[0].latitude +
"ptArriveLAT" + arrivee.latitude);
            //On fabrique maintenant l'url pour envoyer une requête HTTP à
Google Maps
            String url = makeUrl(ptDepart[0], arrivee);

            //On exécute puis récupère le résultat de la requête HTTP
            DefaultHttpClient httpClient = new DefaultHttpClient();
            @SuppressWarnings("unused")
            HttpClient http = sslClient(httpClient);
            HttpGet requete = new HttpGet(url);
            HttpResponse res;
            try {
                  res = httpClient.execute(requete);
                  HttpEntity entity = res.getEntity();
                  String route = null; //résultat de la requête qui contiendra
les points géographiques cryptés
                  if (entity != null){
                        route = EntityUtils.toString(entity);
                  }
                  Log.d(LOGCAT_TAG,"Chemin reçu : " + route);
                  //On décode les points géographiques cryptés
                  geoPoints = decodeEncryptedGeopoints(route);

            } catch (ClientProtocolException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            return geoPoints;
      }

      @Override
      protected void onPostExecute(ArrayList<LatLng> result){
            Fragment f = new CourseEnCours();
            Bundle extras = new Bundle();
            extras.putParcelableArrayList(CourseEnCours.PARCOURS, result);
            f.setArguments(extras);
            context.switchContent(f);
      }



      /**
       * Génère un point de destination aléatoire
       * @param pointDepart le point de départ
       * @param dist la distance donnée
       * @param angleAleatoire angle aléatoire en DEGRES
       * @return le point de destination généré aléatoirement
       */
      private LatLng pointDestination(LatLng pointDepart, float dist, double
angleAleatoire){
            double lat, latV1, latV2, lon, lonV1, lonV2;
            LatLng pDest;

            dist = dist/6371; //On prend en compte le rayon de la Terre

            latV1 = Math.toRadians(pointDepart.latitude);
            lonV1 = Math.toRadians(pointDepart.longitude);
```

```
            latV2 = Math.sin(latV1)*Math.cos(dist) +
Math.cos(latV1)*Math.sin(dist)*Math.cos(angleAleatoire);
            lat = Math.asin(latV2);

            lonV2 = Math.atan2(Math.sin(angleAleatoire) * Math.sin(dist) *
Math.cos(latV1), Math.cos(dist) - Math.sin(latV1) *Math.sin(latV2));
            lon = lonV1 + lonV2;

            pDest = new LatLng(Math.toDegrees(lat),Math.toDegrees(lon));

            return pDest;
      }

      /**
       * Fabrique une url pour envoyer une requête HTTP à Google Maps qui
renverra les points géographiques
       * @param src le point source
       * @param dest le point de destination
       * @return l'url dont il faut se servir pour exécuter la requête
       */
      private String makeUrl(LatLng src, LatLng dest){

            StringBuffer url = new StringBuffer();

            url.append("http://maps.google.com/maps?f=d&hl=en");
            url.append("&saddr=");
            url.append(Double.toString(src.latitude));
            url.append(",");
            url.append(Double.toString(src.longitude));
            url.append("&daddr=");
            url.append(Double.toString(dest.latitude));
            url.append(",");
            url.append(Double.toString(dest.longitude));
            url.append("&ie=UTF8&0&om=0&output=dragdir&dirflg=w");

            return url.toString();
      }

      /**
       * Décode des points cryptés (algorithme de http://facstaff.unca.edu/
mcmcclur/googlemaps/encodepolyline/)
       * @param encoded la chaîne de caractère contenant les points à décoder
       * @return la liste des points décodés
       */
      private ArrayList<LatLng> decodeEncryptedGeopoints(String encoded){

      // Récupération des Latitudes et Longitudes encodés
      encoded = encoded.split("points:\"")[1].split("\",")[0];
      // Remplacement des \\ par des \ pour régler des problèmes
      encoded = encoded.replace("\\\\", "\\");

      //décodage
      ArrayList<LatLng> poly = new ArrayList<LatLng>();
    int index = 0, len = encoded.length();
    int lat = 0, lng = 0;

        while (index < len) {
            int b, shift = 0, result = 0;
            do {
                b = encoded.charAt(index++) - 63;
                result |= (b & 0x1f) << shift;
                shift += 5;
            } while (b >= 0x20);
            int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
            lat += dlat;
```

```
            shift = 0;
            result = 0;
            do {

                b = encoded.charAt(index++) - 63;
                result |= (b & 0x1f) << shift;
                shift += 5;
            } while (b >= 0x20);
            int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
            lng += dlng;

            LatLng p = new LatLng(lat/1E5,lng/1E5);
            poly.add(p);

        }
        return poly;
    }
    /**
     * Permet de décoder du ssl (https)
     * Source : http://stackoverflow.com/questions/7622004/android-making-
https-request/13485550#13485550 by rposky
     * @param client le HTTPClient à décoder
     * @return le nouveau httpClient
     */
    private HttpClient sslClient(HttpClient client) {
        try {
            X509TrustManager tm = new X509TrustManager() {

                    public void checkClientTrusted(
                                java.security.cert.X509Certificate[] chain,
                                String authType)
                                throws
java.security.cert.CertificateException {

                    }

                    public void checkServerTrusted(
                                java.security.cert.X509Certificate[] chain,
                                String authType)
                                throws
java.security.cert.CertificateException {

                    }

                    public java.security.cert.X509Certificate[]
getAcceptedIssuers() {
                            return null;
                    }
            };
            SSLContext ctx = SSLContext.getInstance("TLS");
            ctx.init(null, new TrustManager[]{tm}, null);
            SSLSocketFactory ssf = new MySSLSocketFactory(ctx);

ssf.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
            ClientConnectionManager ccm = client.getConnectionManager();
            SchemeRegistry sr = ccm.getSchemeRegistry();
            sr.register(new Scheme("https", ssf, 443));
            return new DefaultHttpClient(ccm, client.getParams());
        } catch (Exception ex) {
            return null;
        }
    }
}
```

Classe Course

La classe Course correspond à ce qui a été défini en analyse puis en conception. Elle contient une Course, avec toutes ses caractéristiques, et toutes les méthodes qui la concernent. Elle possède un ensemble de variables qui définissent la course, couplé avec des getters et des setters. Elle gère aussi le calcul des calories brûlées en fonction du sport pratiqué et de sa durée, de la vitesse et du poids de l'utilisateur. Ce calcul est d'abord effectué dans la méthode getMet qui détermine le MET (Metabolic Equivalent of Task) de l'activité physique pratiqué. Celle-ci nous renvoi donc un facteur qui nous aide à déterminer le nombre de calories brûlées par le sportif depuis le début de sa course.

Notre classe Course est aussi capable de gérer l'état d'une course. En effet, non seulement elle arrive à prendre en compte son état, et propose un modificateur publique (setEtatCourse) pour cet état, mais en plus elle gère toute seule la mise en pause d'une course.

```
package com.makeursport.gestionCourse;

/**
 * Course avec ses différentes caractéristiques
 *
 * @author l'équipe MakeUrSport
 */
public class Course {
	private static final String LOGCAT_TAG = "Course";
	/**
	 * Le sport pratiqué pour cette course
	 */
	private Sport sport;
	/**
	 * Le début de la course en ms
	 */
	private long debutCourse;
	/**
	 * La durée de la course (mise à jour que lorsqu'une course est finie) en
s
	 */
	private long duree;
	/**
	 * Le temps de mise en pause pendant une course en ms
	 */
	private long tempsPause;
	/**
	 * Le timestamp au moment où on a mis la course en pause (ms)
	 */
	private long debutPause;
	/**
	 * La distance parcourue lors de cette course, en KM
	 */
	private double distance;
	/**
	 * La vitesse réelle de l'utilisateur en KM/H
	 */
	private float vitesseReelle;
	/**
	 * L'état de la course en ce moment
	 */
	private EtatCourse etatCourse;
	/**
	 * La date de la course
	 */
	private Date date;
	/**
```

```
	 * Le sportif pratiquant le sport
	 */
	private Sportif user;
	/**
	 * L'identifiant de la course
	 */
	private int id;
	/**
	 * Dernière localisation de l'utilisateur, renseigné uniquement
	 * si la course est sélectionnée depuis l'historique.
	 */
	private LatLng dernierePos;

	/**
	 * Création d'une course vide
	 */
	public Course() {
		this.setDistance(0);
		this.setVitesseReelle(0);
		this.etatCourse=EtatCourse.CourseArretee;
		this.setDate(new Date());
		this.setTempsPause(0);
		this.setSport(Sport.COURSE);
		this.tempsPause=0;
	}
	/**
	 * Création d'une course à partir de toutes ses informations
	 * @param id l'id de la course
	 * @param date la date de début de la course
	 * @param distance la distance parcourue lors de la course
	 * @param duree la durée de la course
	 * @param sport le sport pratiqué
	 */
	public Course(int id,long date,double distance ,long duree, Sport sport){
		Log.d(LOGCAT_TAG,"Création d'une course : (" + date + ") de " +
distance + "km de " + duree + "s");
		this.setDate(new Date(date));
		this.setDistance(distance);
		this.setSport(sport);
		this.setDuree(duree);
		this.etatCourse=EtatCourse.CourseArretee;
		this.id=id;
	}
	/**
	 * Met la durée de la course à jour
	 * @param duree la durée de la course, en millisecondes
	 */
	private void setDuree(long duree) {
		this.duree = duree;
	}
	/**
	 * Récupération de l'id de la course
	 * @return l'id de la course
	 */
	 public int getId() {
		 return this.id;
	 }

	/**
	 * @return la durée de la course en secondes
	 */
	public long getDuree() {
		if(this.etatCourse==EtatCourse.CourseArretee) {
			return this.duree;
		}
```

```
            return (new Date().getTime() - this.getTempsPause() -
this.debutCourse)/1000;
      }


      /**
       * Met à jour le début de la course
       * @param debutCourse le moment de début de la course
       */
      public void setDebutCourse(long debutCourse) {
            this.debutCourse = debutCourse;
      }


      /**
       * Récupérer la durée de temps en pause pendant cette course
       * @return la durée pendant laquelle l'utilisateur a été en pause (dans
tempsPause)
       */
      public long getTempsPause() {
            return this.tempsPause;
      }

      /**
       * Modifie la durée de temps en pause de cette course
       * @param tempsPause le temps en pause
       */
      public void setTempsPause(long tempsPause) {
            this.tempsPause=tempsPause;
      }

      /**
       * rajoute une durée au temps de pause
       * @param tempsPause rajoute tempsPause au temps que l'on a passé en pause
       */
      public void addTempsPause(long tempsPause) {
            this.setTempsPause(this.getTempsPause()+tempsPause);
      }
      /**
       * Calcule la vitesse moyenne de l'utilisateur pour cette course
       * @return vitesse moyenne de la course en km/h
       */
public double getVitesseMoyenne () {
      Log.d(LOGCAT_TAG,"Vitesse Moy : " + (double) this.getDistance() + " / " +
(double) this.getDuree());
      double vitKms = this.getDistance() / this.getDuree();
      if(Double.isNaN(vitKms)) {
            vitKms=0;
      }
      return (double) (Math.floor(vitKms*3600*100)/100);
      }


      /**
       * Retourne la distance parcourue de la course
       * @return distance de la course en km
       */
      private double getDistance() {
            return this.distance;
      }
      /**
       * Retourne la distance arrondie parcourue de la course
       * @return la distance de la course en km, arrondie au centième
       */
      public float getDistanceArrondi() {
```

```
            Log.w(LOGCAT_TAG, this.getDistance() + " : " +
(Math.floor(this.getDistance()*100)/100));
            return (float) (Math.floor(this.getDistance()*100)/100);
      }


      /**
       * Met à jour la distance de la course
       * @param distance en km
       */
      public void setDistance(double distance) {
            this.distance = distance;
      }
      /**
       * Rajoute une distance à la course
       * @param distance en km
       */
      public void addDistance(double distance) {
            this.distance+=distance;
      }

      /**
       * Retourne la vitesse réelle de la course
       * @return Vitesse réelle en ce moment en km/h
       */
      public float getVitesseReelle() {
            return (float) Math.floor((this.vitesseReelle*100)/100);
      }


      /**
       * Met à jour la vitesse réelle de la course
       * @param vitesseReelle en km/h
       */
      public void setVitesseReelle(float vitesseReelle) {
            this.vitesseReelle = vitesseReelle;
      }


      /**
       * récupère les calories brûlées de la course
       * @return les calories brûlées
       */
      public float getCaloriesBrulees() {
            if(this.getUser() == null) {
                  Log.e(LOGCAT_TAG,"getUser() == null");
            } else if(this.getSport()==null) {
                  Log.e(LOGCAT_TAG,"sport==null");
            }
            float duree = ((float)this.getDuree()) /60.0F;
            double caloriesBrulees =
this.calculCaloriesBrulees(this.getUser().getPoids(),duree,this.getMet(this.getS
port(), this.getVitesseMoyenne()) );

            return (float) (Math.floor(caloriesBrulees*100)/100);
      }
      /**
       * Sauvegarde cette course dans l'historique
       * @param context Le contexte de l'application
       */
      public void sauvegarderCourse(Context context, LatLng dernierePosition) {
            Log.d(LOGCAT_TAG, "Sauvegarde de la course...");
            if(this.getDuree() > 1) {
                  GestionnaireHistorique gest = new
GestionnaireHistorique(context);
                  gest.enregistrerCourse(this, dernierePosition);
```

```
                  Toast.makeText(context,
context.getText(R.string.course_sauvegardee), Toast.LENGTH_LONG).show();
            } else {
                  Log.w(LOGCAT_TAG, "Ou pas : this.getDuree()=" +
this.getDuree());
            }
      }

      /**
       * Récupère l'état de la course
       * @return l'état de la course
       */
      public EtatCourse getEtatCourse() {
            return this.etatCourse;
      }


      /**
       * Met à jour l'état de la course,  en prenant en compte les temps de
pause
       * @param etatCourse le nouvel état de la course
       */
      public void setEtatCourse(EtatCourse etatCourse) {
            if(this.etatCourse==EtatCourse.CourseEnPause &&
etatCourse==EtatCourse.CourseLancee) {
                  this.addTempsPause(new Date().getTime() - this.debutPause);
            }
            else if(this.etatCourse==EtatCourse.CourseLancee &&
etatCourse==EtatCourse.CourseEnPause) {
                  this.debutPause = new Date().getTime();
            } else if(this.etatCourse == EtatCourse.CourseArretee && etatCourse
== EtatCourse.CourseLancee) {
                  this.setDebutCourse(new Date().getTime());
            } else if(etatCourse == EtatCourse.CourseArretee) {
                  Log.d(LOGCAT_TAG + "_arretCourse", "On remplace this.duree par
la duree("+this.getDuree());
                  this.duree = this.getDuree();
            }
            this.etatCourse = etatCourse;
      }


      /**
       * Récupère le sportif
       * @return le sportif
       */
      public Sportif getUser() {
            return this.user;
      }


      /**
       * Met à jour le sportif de la course
       * @param user le sportif
       */
      public void setUser(Sportif user) {
            Log.d(LOGCAT_TAG, "mise à jour de l'user");
            this.user = user;
      }



      /**
       * Retourne la date de début de la course
       * @return the date
       */
```

```
	public Date getDate() {
		return date;
	}

	/**
	 * Change la date de début de la course
	 * @param date la date à stocker
	 */
	public void setDate(Date date) {
		this.date = date;
	}

	/**
	 * Récupère le sport de la course
	 * @return le sport
	 */
	public Sport getSport() {
		return sport;
	}

	/**
	 * Met à jour le sport de la course
	 * @param sport
	 */
	public void setSport(Sport sport) {
		this.sport = sport;
	}

	/**
	 * Met à jour la dernière position de l'utilisateur
	 */
	public void setDernierePos(LatLng pos) {
		this.dernierePos=pos;
	}
	/**
	 * Retourne la dernière position de l'utilisateur
	 */
	public LatLng getDernierPos() {
		return this.dernierePos;
	}

	/**
	 * Calcule le nombre de calories brulées
	 * @param poids le poids du sportif en kg
	 * @param duree la durée de la course en minutes
	 * @param met le MET du sport,
	 * @return le nombre de calories brûlées par le sportif
	 */
	private double calculCaloriesBrulees(float poids, double duree, float met)
{
		double nbCaloriesBrulees = 0;
		nbCaloriesBrulees = duree*(met*3.5*poids)/200;
		return nbCaloriesBrulees;
	}

	/**
	 * Calcule le MET d'un sport à une vitesse particulière
	 * @param sport le sport pratiqué
	 * @param vitesse l'allure du sportif (en m/s)
	 * @return le MET du sport
	 */
		private float getMet(Sport sport, double vitesse){
		float met = 1.0f;
		if(vitesse<0.5) {
			met = 0.0f;
		}
```

```
            else if(sport==Sport.COURSE && vitesse > 1.5) {
                  if(vitesse<4){
                        met = 2.3F;
                  }
                  else if(vitesse<=4.8){
                        met = 3.3F;
                  }
                  else if(vitesse<=5.5){
                        met = 3.6F;
                  }
                  else if(vitesse<=10){
                        met = 7.0F;
                  } else {
                        met = 7.5F;
                  }
            }
            else if(sport==Sport.VELO && vitesse > 1.5){
                  if(vitesse<16){
                        met = 4.0F;
                  }
                  else{
                        met = 5.5F;
                  }
            }
            else if(sport==Sport.ROLLER && vitesse > 1.5) {
                        if(vitesse<10){
                              met = 6;
                        }
                        else {
                              met = 7;
                        }
            }

            return met;
      }
}
```

Classe GestionnaireHistorique

La classe GestionnaireHistorique sert à interagir avec notre base de données. Elle regroupe quatre principales méthodes, et a pour classes privée différentes AsyncTask qui nous aident à ne pas lancer les traitements avec la base de données dans le thread de l'interface graphique. De plus, nous avons à chaque fois besoin du contexte du fragment en cours pour pouvoir mettre à jour l'interface graphique. C'est pour cela que chacune des AsyncTask interagit avec un contexte différent en fonction du Fragment dans lequel elles sont lancées.

Les 4 principales méthodes sont :

- enregistrerCourse qui permet d'enregistrer une course donnée en paramètre dans la base de données
- selectToutesLesCourses qui permet de sélectionner toutes les courses de la base de données
- selectionnerCourse qui permet de sélectionner une course dont l'identifiant est passé en paramètre depuis la base de données
- et supprimerCourse qui nous permet de supprimer une course dont l'identifiant est passé en paramètre

```
package com.makeursport.gestionCourse;

/**
 * La classe de Gestionnaire Historique qui permet d'interagir avec notre base
de données.
 * C'est grâce à elle que des classes comme {@link CourseFragment} ou {@link
HistoriqueFragment}
 * ont accès à cette base de données
 * @author L'équipe MakeUrSport
 *
 */
public class GestionnaireHistorique {

	public static final String LOGCAT_TAG = "GestionnaireHistorique";
	/**
	 * La base de données avec laquelle on veut interagir
	 */
	private CourseDS bdd;
	/**
	 * L'{@link HistoriqueFragment} depuis lequel notre GestionnaireHistorique
est lancé
	 */
	private HistoriqueFragment hFragment;

	/**
	 * Le {@link CourseFragment} depuis lequel notre GestionnaireCourse est
lancé
	 */
	private CourseFragment courseFragment;
	/**
	 * Constructeur par défaut.
	 * @param context
	 */
	public GestionnaireHistorique(Context context) {
		this.bdd = new CourseDS(context);
	}
	/**
	 * Constructeur lorsqu'il est appelé par un HistoriqueFragment
	 * @param f le fragment en question
	 */
```

```
	public GestionnaireHistorique(HistoriqueFragment f) {
		this(f.getSherlockActivity());
		this.hFragment=f;
	}
	/**
	 * Constructeur lorsqu'il est appelé par un CourseFragment
	 * @param f le fragment en question
	 */
	public GestionnaireHistorique(CourseFragment f) {
		this(f.getSherlockActivity());
		this.courseFragment=f;
	}

	/**
	 * Enregistre une course dans la base de données, en arrière-plan
	 * @param course la base de données à exécuter
	 */
	public void enregistrerCourse(Course course, LatLng dernierePos) {
		InsertCourseATask asynctask = new InsertCourseATask(dernierePos);
		asynctask.execute(course);
	}

	/**
	 * Récupère toutes les courses de la base de données et met à jour la vue
	 * <strong>Attention</strong>, doit être lancé depuis un
HistoriqueFragment
	 */
	public void selectToutesLesCourses(){
		SelectToutesCoursesATask asynctask = new SelectToutesCoursesATask();
		asynctask.execute();
	}

	/**
	 * Selectionne une course et mets à jour la vue
	 * <strong>Attention</strong>, doit être lancé depuis un CourseFragment
	 * @param id l'identifiant de la course à récupérer
	 */
	public void selectionnerCourse(int id){
		SelectCourseATask asynctask= new SelectCourseATask();
		asynctask.execute(id);
	}

	/**
	 * Supprime une course de la base de données, et change le fragment en
cours.
	 * <strong>Attention</strong> cette méthode doit être appelée uniquement
depuis un CourseFragment
	 * @param idCourse l'identifiant de la course à supprimer
	 */
	public void supprimerCourse(Integer idCourse){
		DeleteCourseATask asynctask = new DeleteCourseATask();
		asynctask.execute(idCourse);
	}

	 /**
	  * AsyncTask sélectionnant toutes les courses de la base de données, et
mettant à jour
	  * le HistoriqueFragment.
	  */
	private class SelectToutesCoursesATask extends AsyncTask<Void,
Void,ArrayList<Course>>
	{
		@Override
		protected ArrayList<Course> doInBackground(Void... args) {
			bdd.open();
```

```
            ArrayList<Course> courses = bdd.selectListesCourses();
            bdd.close();
            return courses;
      }

      @Override
      protected void onPostExecute(ArrayList<Course> result) {
            if(result!=null) {
            hFragment.modifierAdapter(result);
          Log.d(LOGCAT_TAG,"Selection de toutes les Courses effectué");
            }
      }
 }
/**
 * ASyncTask permettant d'insérer une course dans la base de données.
 */
private class InsertCourseATask extends AsyncTask<Course,Void ,Void>
{
  LatLng pos;
 public InsertCourseATask(LatLng pos) {
      this.pos=pos;
 }
 @Override
 protected Void doInBackground(Course... course) {
      bdd.open();
      bdd.insertInfoCourse(course[0],pos);
      bdd.close();
      return null;
 }

}
/**
 * ASyncTask permettant de supprimer une course de la base de données, et
 * qui échange le fragment en cours
 */
private class DeleteCourseATask extends AsyncTask<Integer, Void,Void>
{
 @Override
 protected Void doInBackground(Integer... course) {
      bdd.open();
      bdd.deleteCourse(course[0]);
      bdd.close();
            return null;


 }
 @Override
 protected void onPostExecute(Void arg) {
      if (courseFragment.getSherlockActivity() instanceof MainActivity) {
            courseFragment.retourHistoriqueFragment();
      }
 }
}
/**
 * AsyncTask chargé de récupérer une course depuis la base de données
 * et qui modifie la courseFragment
 */
private class SelectCourseATask extends AsyncTask<Integer, Void,Course>
{
 @Override
 protected Course doInBackground(Integer... args) {
      bdd.open();
      Course maCourse = bdd.selectCourse(args[0]);
      bdd.close();
      return maCourse;
 }
```

```
        @Override
        protected void onPostExecute(Course result) {
              if(result==null){
                    Log.d(LOGCAT_TAG, "PROBLEME selection d'une course");
              }
              else{

        result.setUser(Sportif.fromPrefs(courseFragment.getSherlockActivity()));
                    courseFragment.modifView(result);
              }
        }
      }
}
```

Gestion de la base de données

Afin de mieux comprendre comment se passent les interactions avec une base de données sous Android, nous allons prendre un exemple depuis GestionnaireHistorique, puis nous allons remonter dans les classes et les méthodes utilisées pour mieux comprendre comment cela affecte la base de données.

Tout d'abord, imaginons qu'une de nos classes appelle supprimerCourse sur la course d'id 10. Il faut alors faire une instanciation de DeleteCourseATask puis l'exécuter, avec l'id de la course à supprimer.

L'AsyncTask DeleteCourseATask utilise alors notre variable bdd (de type CourseDS), pour ouvrir la base de données, appeler la méthode deleteCourse et refermer la base de données.

La méthode open() de CourseDS nous permet d'ouvrir une base de données à l'aide de notre classe CourseDBHelper, et la méthode close() ferme cette base de données.

Voyons donc maintenant plus en détail la méthode deleteCourse utilisée ci-dessus.

Code de la méthode deleteCourse de la classe CourseDS

```
	/**
	 * Suppression d'une course de la base de données
	 * @param id l'identifiant de la course
	 * @return true si pas de problèmes, false sinon
	 */
	public boolean deleteCourse(int id) {
		Log.d(LOGCAT_TAG,"Deleting course with id=" +id);
		return database.delete(CourseDBHelper.TABLE_COURSE,
CourseDBHelper.COLUMN_COURSE_ID + "=" + id, null) >0;
	}
```

Nous voyons donc que notre méthode deleteCourse utilise une variable database (qui est de type SQLiteDatabase, et utilise la méthode delete, qui prend comme paramètres le nom de la table, et les différents paramètres (notamment la condition where) qui nous permettent de sélectionner la ligne à supprimer. On remarque aussi que les noms de tables et de colonnes ne sont pas écrits en clair, mais utilise des champs de la classe CourseDBHelper qui décrit la base de données pour simplifier les interactions avec celle-ci.

Nous avons ainsi notre classe GestionnaireHistorique avec laquelle nous interagissons directement, et qui aide à ne pas exécuter les requêtes sur le thread principal de l'application. Associé à notre GestionnaireHistorique, une classe CourseDS contient les principales requêtes que nous souhaitons utiliser. De plus, cette classe interagit avec un CourseDBHelper qui facilite grandement l'interaction avec la base de données, en la décrivant de façon à pouvoir récupérer facilement les noms de tables ou de colonnes, mais aussi nous permettre d'ouvrir, de fermer et de mettre à jour la base de données.

Layout d'une course en cours

En programmation Android, tout ce qui concerne l'interface graphique est décrite dans des fichiers XML (layout) qui sont associés à une Activity. Ces fichiers layout permettent donc de décrire cette interface graphique avec laquelle on pourra interagir depuis l'Activity associé. Un layout est séparé en diverses Vues (View) qui correspondent en fait aux différents outils que l'on peut utiliser pour afficher une information à l'écran. On retrouve dans les View les plus connues les RelativeLayout et les LinearLayout qui permettent de regrouper plusieurs View. L'un les positionne de façon relative (en plaçant une vue à côté, ou au-dessus d'une autre), tandis que l'autre les positionne l'un à la suite des autres, de façon horizontale ou verticale. Enfin, pour afficher des informations textuelles à l'écran, nous utilisons des TextView. Cependant, le cas de la carte est un cas particulier. En effet, une carte n'est pas une View toute simple, c'est en fait un fragment. C'est pour cela que nous utilisons un FrameLayout qui nous permet en fait de réserver la place pour le fragment dans l'interface, pour ensuite le remplacer depuis le code Java.

Le layout fourni ci-dessous est celui qui est couplé avec l'Activity CourseEnCours. Celui-ci sert donc à afficher en temps réel les informations à propos de la course. Comme sur nos maquettes données dans le dossier d'Analyse/Conception, nous avons décidé de réserver une grande partie de l'interface graphique pour la carte de façon à ce que l'utilisateur puisse se repérer. Ensuite, nous avons séparé le reste de la partie visible de l'écran en 4 pour afficher les informations à propos de la course : le nombre de calories brûlées, la vitesse réelle et la vitesse moyenne, et la distance parcourue. De plus, nous avons choisi d'afficher la durée sur la carte, de façon à gagner de la place, et à ne pas gêner de trop l'utilisateur.

On retrouve également en bas de page les références à un voile. Ce voile s'active lorsque le GPS est inactif, et est désactivé par défaut au lancement de l'application.

Nous avons donc dans le layout ci-dessous :

Un RelativeLayout qui englobe tout notre layout. Un LinearLayout qui contient toutes les informations de notre course. Ce LinearLayout qui contient nos informations est quant à lui divisé en 3 : premièrement il contient un RelativeLayout qui englobe la carte et la durée. Ensuite, un autre LinearLayout qui correspond à la première ligne des informations de la course en cours. Enfin, la dernière ligne des informations de la course, contenue dans un autre LinearLayout.

En direct enfant de notre RelativeLayout principal, on retrouve aussi le voile inactif et le TextView avec le message qui l'accompagne.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/RL_couse_en_cours_main_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

      <LinearLayout
          android:id="@+id/activity_course_en_cours_layout"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:orientation="vertical">

          <RelativeLayout
            android:id="@+id/RL_mapetduree"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:layout_weight="4">
```

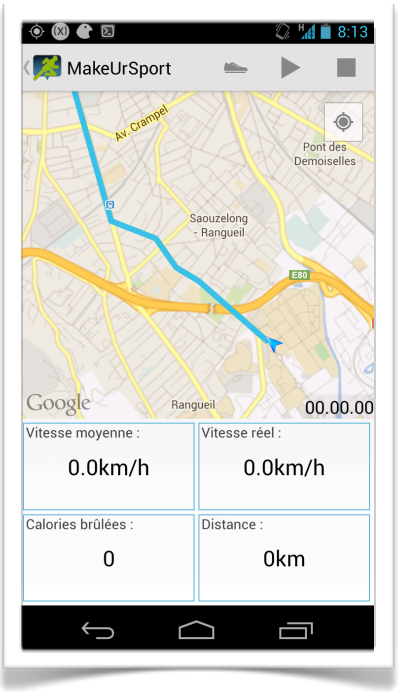
```
  <FrameLayout
        android:id="@+id/mapfragment_location"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

    <TextView
        android:id="@+id/TV_duree"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        style="@style/dureeStyle"
        android:text="@string/duree_par_defaut" />

</RelativeLayout>

  <LinearLayout android:id="@+id/ligne_1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        style="@style/Tile.Line1"
        >
              <RelativeLayout android:id="@+id/RL_vit_moy"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    style="@style/Tile.Tile.Left"
                    >

                    <TextView
                        android:id="@+id/TV_vit_moyenne_titre"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:text="@string/vitmoy_titre"
                        style="@style/Tile.Title">
                    </TextView>

                    <TextView
                        android:id="@+id/TV_vit_moyenne_valeur"
                        style="@style/Tile.Value"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_alignParentBottom="true"
                        android:layout_alignParentTop="true"
                        android:gravity="center"
                        android:text="@string/vitmoy_par_defaut" >
                    </TextView>

              </RelativeLayout>

              <RelativeLayout android:id="@+id/RL_vit_reel"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    style="@style/Tile.Tile.Right"
                    >

                    <TextView
                        android:id="@+id/TV_vit_reel_titre"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:text="@string/vitreel_titre"
                        style="@style/Tile.Title">
                    </TextView>

                    <TextView
                        android:id="@+id/TV_vit_reel_valeur"
                        style="@style/Tile.Value"
```

```
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_alignParentBottom="true"
                        android:layout_alignParentTop="true"
                        android:gravity="center"
                        android:text="@string/vitreel_par_defaut" >

                    </TextView>

            </RelativeLayout>

</LinearLayout>

<LinearLayout android:id="@+id/ligne_2"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      style="@style/Tile.Line2">

      <RelativeLayout android:id="@+id/RL_calories"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            style="@style/Tile.Tile.Left"
            >

            <TextView
                android:id="@+id/TV_calories_titre"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/calories_titre"
                style="@style/Tile.Title">
            </TextView>

            <TextView
                android:id="@+id/TV_calories_valeur"
                style="@style/Tile.Value"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_alignParentBottom="true"
                android:layout_alignParentTop="true"
                android:gravity="center"
                android:text="@string/calories_par_defaut" >
            </TextView>

      </RelativeLayout>

      <RelativeLayout android:id="@+id/RL_distance"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            style="@style/Tile.Tile.Right"
            >

            <TextView
                android:id="@+id/TV_distance_titre"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/dist_titre"
                style="@style/Tile.Title">
            </TextView>

            <TextView
                android:id="@+id/TV_distance_valeur"
                style="@style/Tile.Value"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_alignParentBottom="true"
```

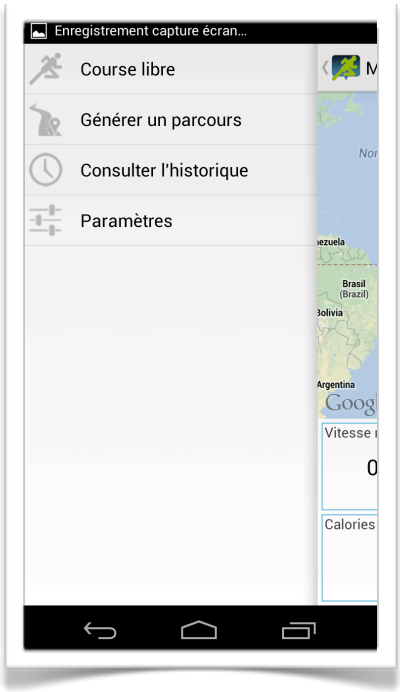
```
                        android:layout_alignParentTop="true"
                        android:gravity="center"
                        android:text="@string/dist_par_defaut" >

                    </TextView>

                </RelativeLayout>

            </LinearLayout>

        </LinearLayout>

        <View
            android:id="@+id/voileInactif"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:background="@color/voile_inactif_color"
            android:visibility="gone"/>

        <TextView
            android:id="@+id/TV_messageInactif"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentLeft="true"
            android:text="@string/message_inactif"
            android:visibility="gone"
            />

</RelativeLayout>
```

Captures d'écran

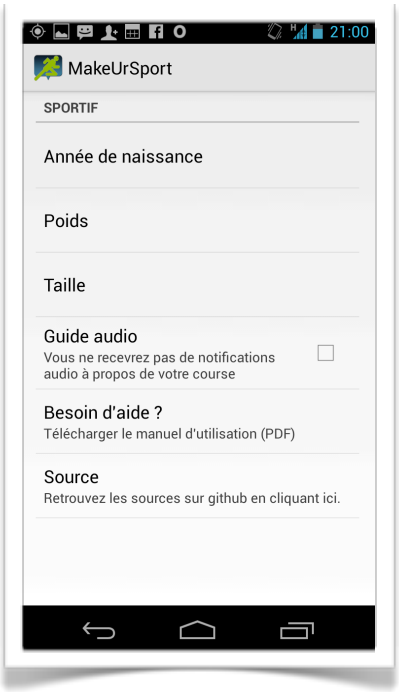
Suite à la programmation, nous avons enfin réussi à obtenir un résultat concret. Nous avons décidé de présenter ici ces résultats au travers de captures d'écran du rendu final de notre application.



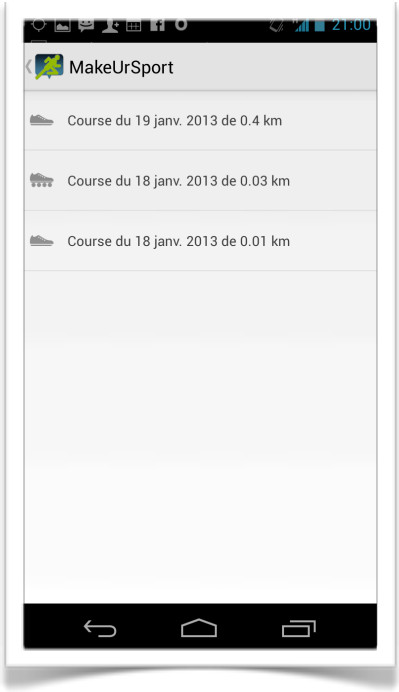
Capture 1 : Suivi d'une course (avec ici un parcours généré).



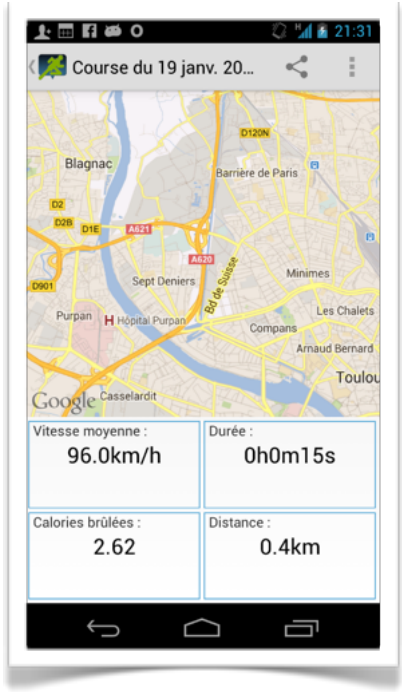
Capture 2 : Menu de notre application.



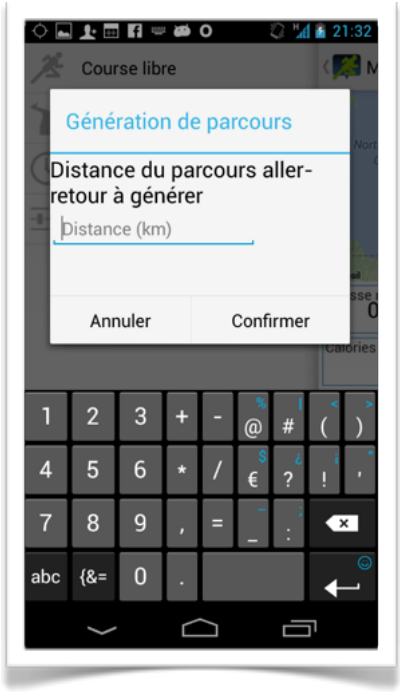
Capture 3 : Préférences.



Capture 4 : Visualisation de l'historique.



Capture 5 : Visualisation d'une course depuis l'historique.



Capture 6 : Boîte de dialogue proposant la génération d'un parcours.

Conclusion

Nous avons mis à disposition dans ce dossier de programmation la plupart des algorithmes, classes, et autres informations intéressantes et nouvelles que nous avons étudiés pendant la phase de programmation. Nous n'avons pas mis tout le code source de l'application dans ce document, nous avons sélectionné les extraits qui nous semblaient les plus pertinents. L'ensemble de ce dossier présente ce que nous avons dû apprendre pour réaliser notre projet, c'est-à-dire tout ce qui était spécifique au développement Android. Au cours des différents points abordés, nous avons pu voir les éléments les plus techniques de notre application tels que les AsyncTask, les fragments, ou encore la gestion de la base de données. Ainsi, grâce aux différents concepts que nous avons appris, nous pouvons dire que nous sommes maintenant capable de développer une application Android dans son intégralité, même s'il existe encore beaucoup de concepts propres à Android que nous n'avons pas abordé ici.

Glossaire

ActionBar : Barre au-dessus de toutes les applications Android permettant d'effectuer certaines actions (démarrer une course...). Il faut utiliser cette barre si on souhaite respecter les règles de design Android.

ActionBarSherlock : Bibliothèque permettant l'utilisation de l'ActionBar dans des versions plus anciennes d'Android.

Activity : Partie visible de l'application, qui permet l'affichage de plusieurs éléments à l'écran et qui permet l'interaction entre l'utilisateur et l'interface graphique.

API : Aplication Programming. C'est une interface fournie par un programme informatique. Elle permet l'interaction des programmes les uns avec les autres, de manière analogue à une interface homme-machine.

AsyncTask : C'est un outil permettant d'exécuter des insctructions en dehors du Thread principal.

Dialog : Boîte de dialogue permettant d'interagir avec l'utilisateur.

Eclipse : IDE (Integrated Development Environment) recommandé par Google pour développer des applications Android.

Fragment : Morceau d'Activity possédant son propre cycle de vie et qui permet d'éviter la surcharge d'une Activity.

HTTP : Hyper Text Transfert Protocol. C'est un protocole de communication client-serveur.

IDE : Environnement de Développement Intégré. C'est un programme regroupant un ensemble d'outils pour le développement logiciel.

ImageView : Vue caractérisée par une image.

Layout : Fichier XML décrivant l'interface graphique.

MET : Metabolic Equivalent of Tasks. Indice qui varie en fonction du poids de l'utilisateur et qui est nécessaire au calcul des calories brûlées.

SDK : System Development Kit. C'est un ensemble d'outils qui permet aux développeurs de créer des applications d'un type défini, ici Android.

StrictMode : Outil disponible depuis les denrières versions d'Android pour vérifier qu'on n'exécute pas de code lourd dans le Thread principal.

TextView : Vue caractérisé par du texte.

Thread : Processus représentant l'exécution d'un ensemble d'instructions du langage machine d'un processeur, il peut être exécuté en parallèle d'autres threads.

Thread UI (thread principal) : C'est le Thread principal d'une application. Il faut faire attention à ne pas le surcharger, car c'est lui qui gère l'affichage. Si jamais on effectue de longs traitement dedans, cela bloquera l'utilisateur ce qui peut entrainer une fermeture soudaine de l'application.

TTS (Text To Speach) : Procédé qui permet de faire prononcer du texte à notre smartphone, il permet par exemple de donner des informations à un coureur via ses écouteurs lorsqu'il effectue une activité sportive.

View : Elément graphique qui compose une Activity.

XML : eXtensible Markup Language. C'est un langage de programmation de balisage, qui permet de faciliter les échanges et les transferts d'informations.