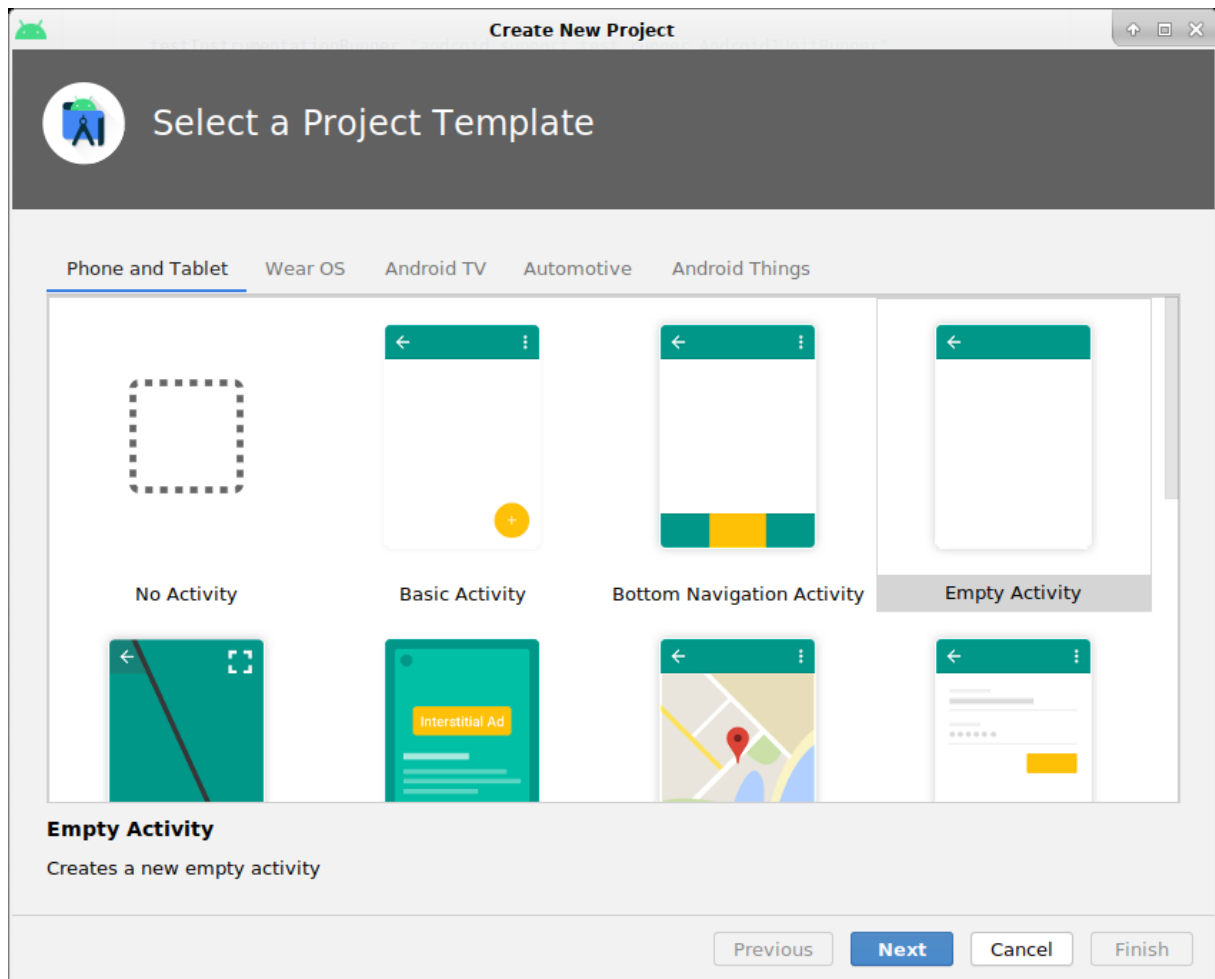


Découverte d'Android Studio

Création d'un premier projet Android

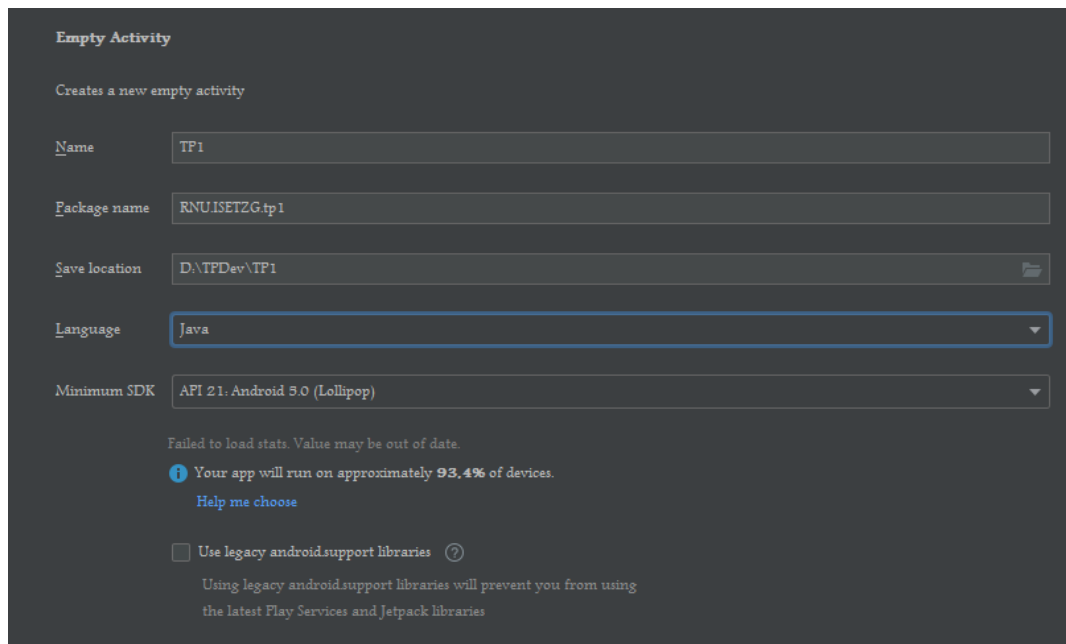
Au tout début, vous n'aurez aucun projet Android. Dans la fenêtre d'accueil, choisissez Start a new Android Studio project pour commencer à créer votre premier projet. Si internet n'est pas activé dans la salle, il faudra attendre un peu.

Dans la première boîte de dialogue, choisissez le type d'application souhaitée. Il faut déjà bien connaître celle que vous choisirez, car l'assistant va générer du code assez complexe. Pour le TP1, choisissez *Empty Activity*.



Type de projet

Dans la seconde boîte de dialogue, on vous demande le nom de l'application, mettez TP1, son paquetage Java : RNU.ISETZG.tp1, l'emplacement du dossier à créer pour le projet.



Empty Activity

Creates a new empty activity

Name: TP1

Package name: RNUISETZG.tp1

Save location: D:\TPDev\TP1

Language: Java

Minimum SDK: API 21: Android 5.0 (Lollipop)

Failed to load stats. Value may be out of date.

i Your app will run on approximately 93.4% of devices.

[Help me choose](#)

☐ Use legacy android.support libraries ?

Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

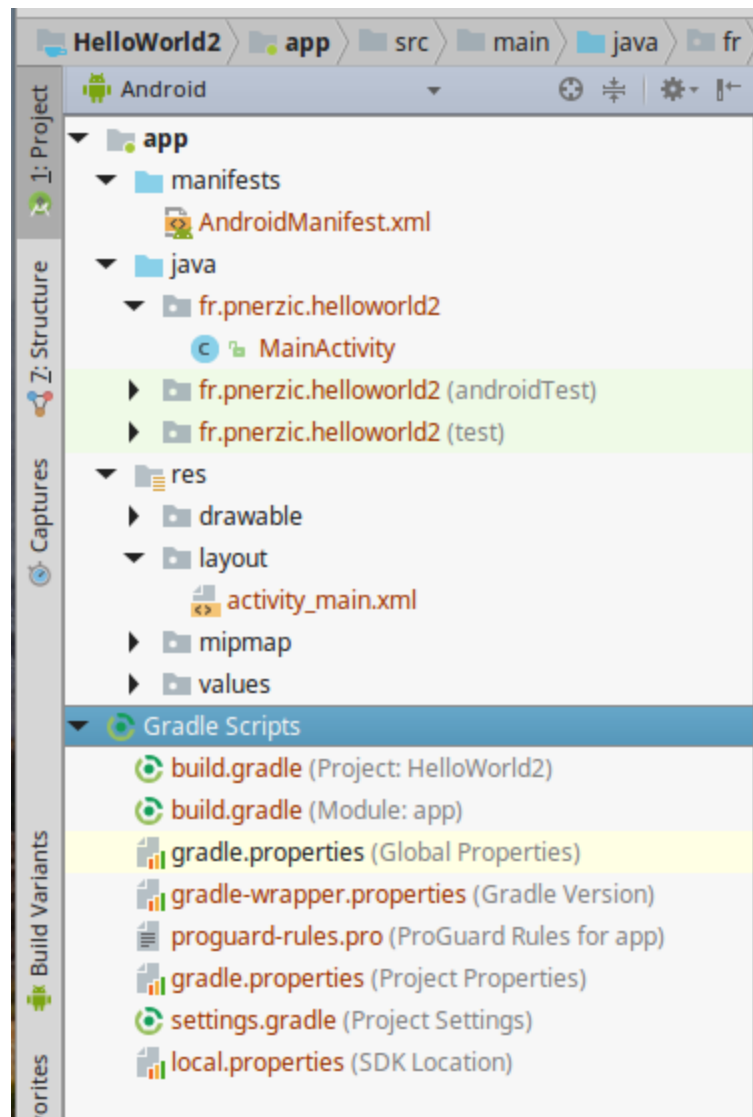
Détails du projet

Il faut indiquer le niveau d'API voulu, par exemple la 24. C'est la version minimale que devront avoir les smartphones pour installer votre application : ils devront être au moins aussi récents que cette API. Cela définit le niveau de fonctionnalités de votre application : s'il est trop élevé, peu de smartphones seront capables de faire tourner votre application ; trop bas, vous n'aurez pas beaucoup de fonctions agréables à votre disposition. Vous pouvez cliquer sur *Help me choose* pour avoir un graphique de la distribution des versions d'API dans le monde.

Découverte de la fenêtre de travail

L'interface d'Android Studio est un peu déroutante au début. Vous verrez rapidement qu'on ne se sert que d'une toute petite partie, vite apprise. Il y a des boutons tout autour de la fenêtre : Project, Structure, Captures, Favorites, Build Variants, Terminal, Android Monitor, Messages, TODO, etc. Ce sont des boutons à bascule, exclusifs entre eux sur un même côté, qui affichent des onglets ; celui qui est actif est grisé. Par exemple dans l'image ci-dessous, c'est l'onglet Project qui est ouvert.

La structure du projet est visible à gauche, comme dans Eclipse. Les fichiers sont arrangés de manière pratique : manifeste, sources java et ressources, grâce au choix déroulant Android juste au-dessus.

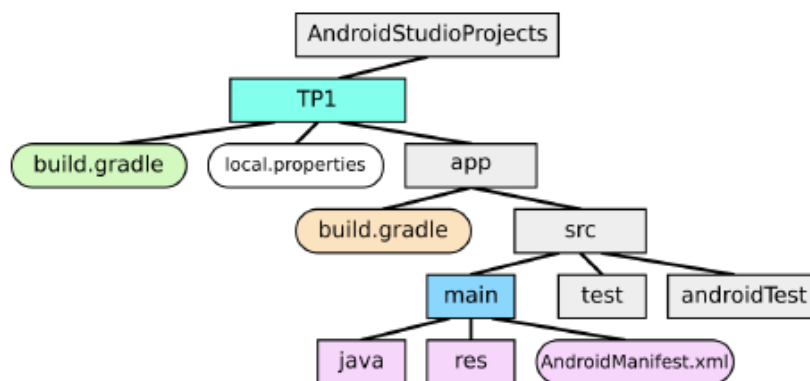


Inventaire du projet

Prenez quelques instants pour déplier les dossiers manifests, java et res et regarder ce qu'il y a dedans (ça a été décrit en cours).

Structure du projet

Allez voir le dossier correspondant au projet, dans le dossier AndroidStudioProjects\TP1 de vos documents, le nom complet est dans le titre de la fenêtre Android Studio :



Structure des fichiers d'un projet Android

- Le dossier du projet contient le premier build.gradle, celui en vert qui indique les dépôts à utiliser pour la compilation. Le fichier local.properties contient le chemin du SDK.
- Le dossier app contient le second build.gradle, celui en orange qui décrit les caractéristiques du projet : API et dépendances.
- Le dossier app/src/main en bleu contient le cœur du projet : manifeste, sources et ressources.
- app/src/main/java contient les dossiers correspondant au package que vous avez déclaré, avec les sources Java tout au bout du package.

- app/src/main/res/drawable contiendra des images vectorielles utilisées dans votre projet.
- app/src/main/res/layout contient les dispositions d'écran des activités de votre projet (voir le TP2).
- app/src/main/res/menu contient les menus contextuels et d'application du projet
- app/src/main/res/mipmap contient les icônes bitmap dans diverses résolutions.
- app/src/main/res/values contient les constantes du projet, dont les chaînes de caractères avec toutes leurs traductions.

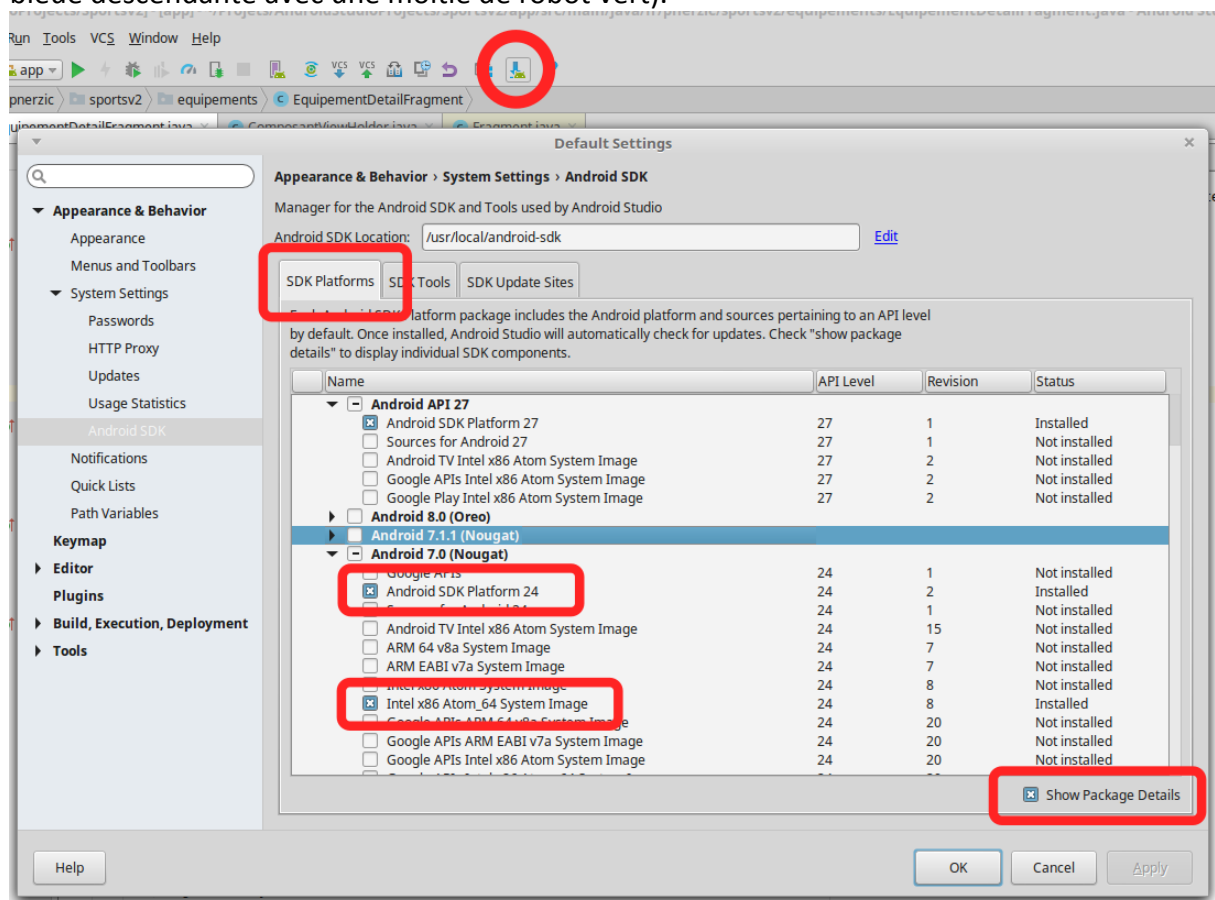
Ces fichiers n'apparaissent pas ainsi dans l'interface, ils sont regroupés autrement.

Découverte du SDK Android

On va regarder les outils de développement : le *Software Development Toolkit* d'Android.

Gestionnaire de SDK

Ouvrez le gestionnaire de SDK en cliquant sur le bouton entouré en rouge (c'est une flèche bleue descendante avec une moitié de robot vert).



SDK Manager

Cochez Show Package Details pour voir tous les éléments installables : différentes machines virtuelles, avec ou sans Google Maps, etc.

L'onglet SDK Tools montre les outils installés, par exemple Android Emulator, Android SDK Tools. Ne vous inquiétez pas de ne rien y comprendre au début.

Notez l'emplacement du SDK pour l'exercice suivant, *Android SDK Location* : D:\android\sdk. Fermez la fenêtre avec le bouton Cancel.

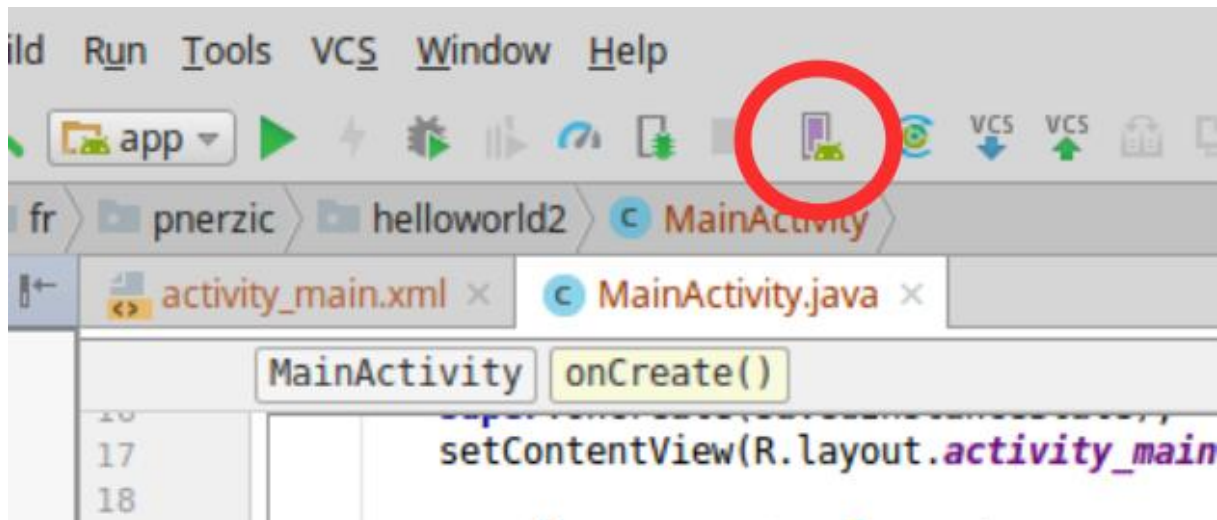
Contenu du SDK

Avec le navigateur de fichiers du système, ouvrez le dossier du SDK et regardez ce qu'il y a dedans :

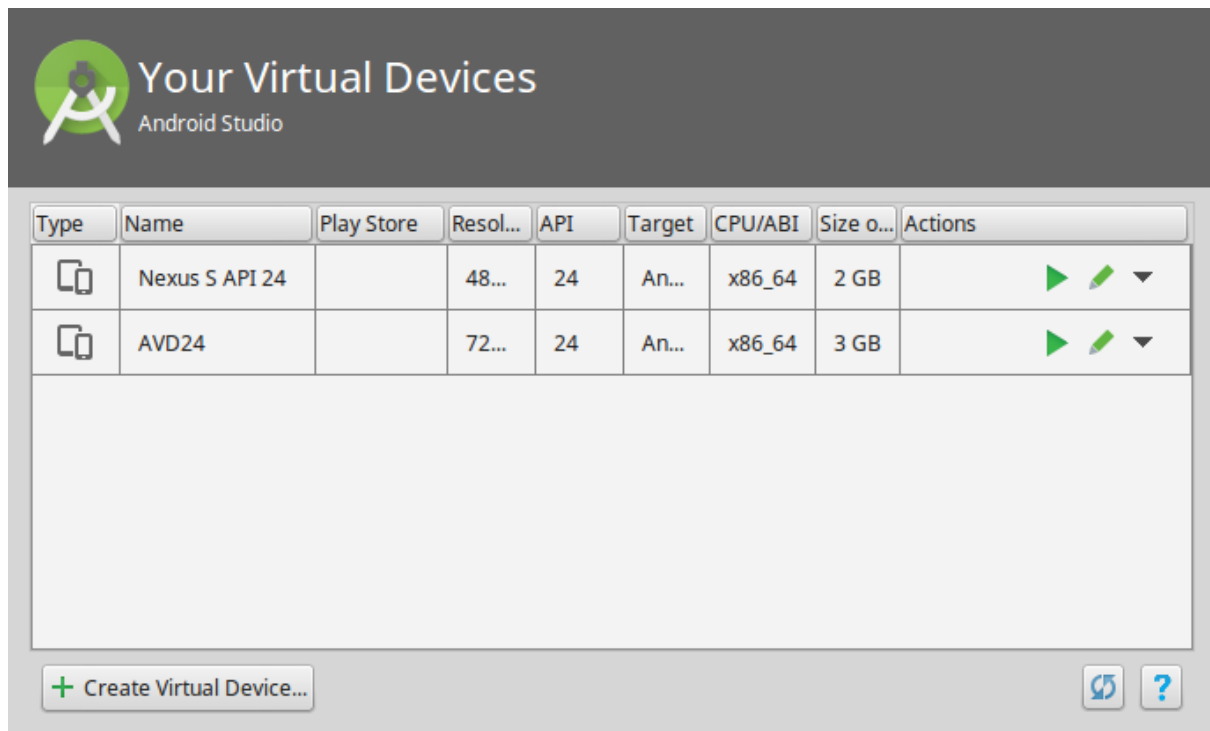
- tools qui contient les outils de gestion du SDK et des AVD (emulator.exe),
 - platform-tools qui contient quelques programmes utiles comme adb.exe (adb sous Linux)
- Ces programmes sont des commandes qui sont normalement disponibles dans une fenêtre appelée *Terminal* en bas d'Android Studio. En principe, vous devriez pouvoir taper adb dans ce shell et ça vous affiche l'aide de cette commande. On va s'en servir un peu plus bas.
- system-images qui contient les images de la mémoire de tablettes virtuelles. Juste pour comprendre pourquoi il n'y a que ça, regardez la taille des fichiers .img. Ce sont des images de disques virtuels à lancer avec qemu.

Android Virtual Device : AVD

On revient encore dans Android Studio, pour lancer le gestionnaire d'AVD.



Bouton du gestionnaire d'AVD



Gestionnaire d'AVD

Normalement, vous devez déjà trouver un AVD existant. Vous pourrez l'utiliser pour les TP, même s'il a un écran un peu trop grand pour nos besoins.

Création d'un AVD si nécessaire

Commencez à créer un smartphone virtuel, avec le bouton + Create Virtual Device. Ensuite, vous avez beaucoup de choix possibles. Le principe est de ne pas créer un smartphone virtuel trop riche. Si le smartphone virtuel est trop puissant, trop gros (RAM et taille d'écran), il va occuper énormément de place en mémoire et sur le disque. Vous pourriez vous retrouver à court.

Un autre point important est le CPU du smartphone virtuel. Les vrais smartphones fonctionnent en général avec un processeur ARM. Ça désigne un modèle de programmation (registres et instructions) spécifiques qui doit être simulé sur votre ordinateur. Le mieux est de choisir un AVD avec un processeur de type amd64 qui ne devra pas être simulé, mais simplement lié, comme avec Docker ou VirtualBox.

Une solution prudente consiste donc à choisir l'AVD le moins rutilant. Par exemple, choisissez le type Phone, modèle 5.1 480x800 en *mdpi* ; puis dans l'écran suivant, changez d'onglet pour x86 images et choisissez une des API installées, la 27 Android 8.1 par exemple, puis dans le dernier écran, activez Show Advanced Settings et changez la taille de la RAM : au lieu de 343 Mo, mettez 512 Mo, et décochez Enable Device Frame. Validez la création de ce smartphone virtuel.

La création d'un AVD prend du temps. Il faut générer l'image de la mémoire virtuelle, entre 1 et 4 Go. Une fois que c'est fini, l'AVD apparaît dans la liste et vous pouvez le lancer en cliquant sur le triangle vert à droite. Notez qu'elle met également beaucoup de temps à devenir disponible.

Pour ne pas perdre de temps plus tard, vous devrez faire attention à ne pas la fermer.

Si jamais le lancement échoue, c'est qu'il y a eu un problème d'application des stratégies Windows lors du démarrage de la machine. Deux variables, %HOMEDRIVE% et %HOMEDIR% n'ont pas de valeur.

Il faut redémarrer l'ordinateur et tout relancer

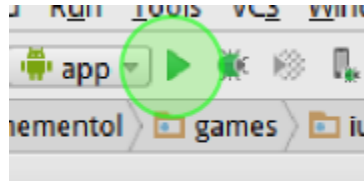
Android Debug Bridge : ADB

adb devices Affiche la liste des smartphones connectés : les réels par USB et les virtuels actifs.

Programmation

Exécution

Le lancement de l'application est simple. Il suffit de cliquer sur le bouton triangulaire vert dans la barre d'outils (celui qui est entouré en vert ci-dessous).



Bouton de lancement

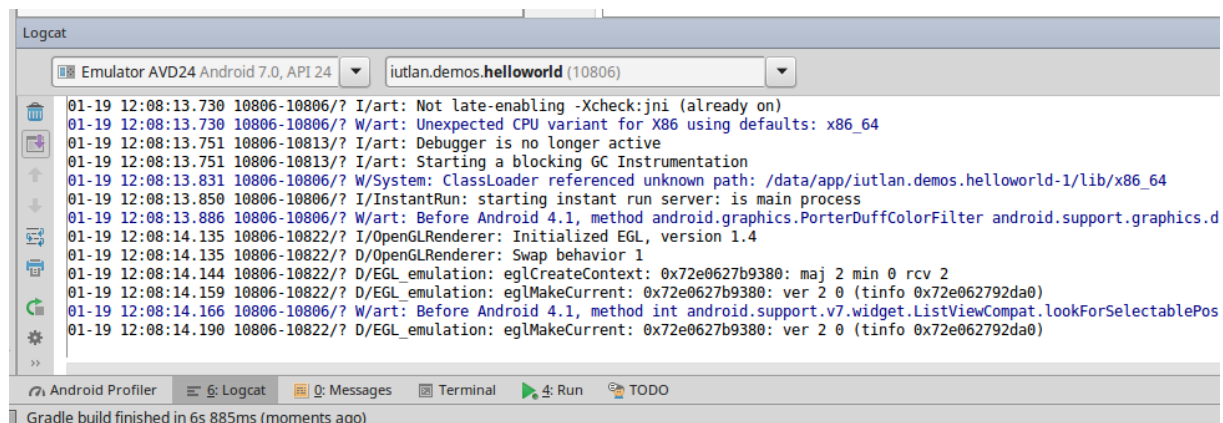
Ensuite, s'il n'y a aucune tablette connectée ni AVD en cours, Studio vous demande quel AVD il faut lancer. Il en faut un qui soit au moins du niveau d'API de votre application. Cochez Use same device for future launches.

Affichage d'un message

Ouvrez la source MainActivity.java et trouvez la méthode onCreate. Cela doit ressembler à la source suivante. Complétez-le avec les instructions qui permettent de faire afficher un message dans la fenêtre LogCat.

```
import android.util.Log;
...
public class MainActivity extends Activity {
    public static final String TAG = "hello";
    @Override
    protected void onCreate(...) {
        super.onCreate(...);
        setContentView(R.layout.activity_main);
        Log.i(TAG, "Salut !");
    }
}
```

Ouvrir la fenêtre Android Monitor, onglet LogCat. NB: l'interface est légèrement différente dans les nouvelles versions de Studio.



LogCat

Cette fenêtre permet de voir les messages émis par la tablette. Le bouton en forme de poubelle vide ces messages. Vous pouvez changer le niveau d'affichage : verbose, debug, info. . . ou créer

un filtre basé sur le TAG pour ne voir que les messages utiles.

Lancez l'exécution du projet. Vous devriez voir une ligne similaire à celle-ci

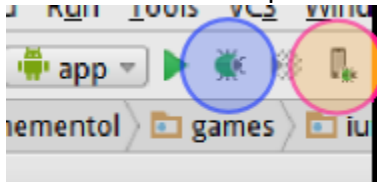
```
01-27 09:58:38.310 992-992/fr.iutlan.tp1 I/hello: Salut !
```

Mise au point (Debug)

Nous allons expérimenter quelques aspects du débogueur intégré dans Studio. On va être un peu limités par la petitesse du programme. On va seulement pouvoir découvrir quelques techniques très utiles : points d'arrêt et exécution pas à pas.

. Lancement en mode debug

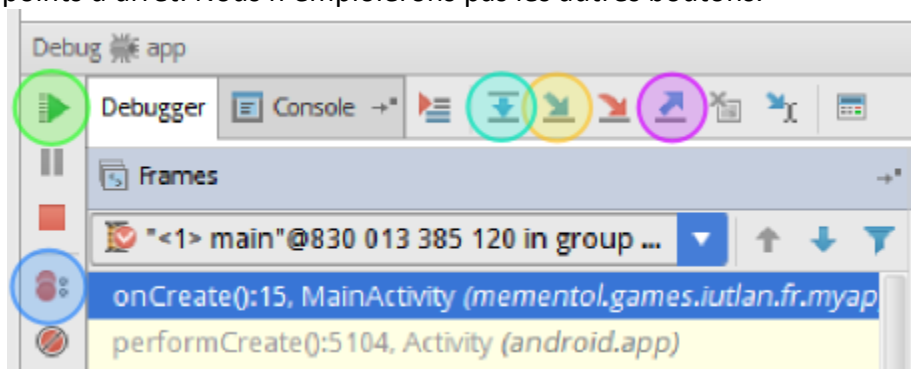
Normalement, il suffit de cliquer sur le bouton bleu pour lancer la mise au point.



Bouton de mise au point

Mais s'il y a une erreur de connexion (message Error running app: Unable to open debugger port dans la barre d'état tout en bas), il faut d'abord « attacher » Studio à l'AVD. Il faut cliquer sur le bouton Attach debugger to Android process entouré en fuschia ci-dessous. Vous serez peut-être obligé(e) de relancer l'AVD.

Les boutons utiles pour mettre le programme au point. Le bouton entouré en vert permet de faire repartir le programme jusqu'au prochain point d'arrêt. Le bouton cyan permet d'exécuter la prochaine ligne sans rentrer dans les méthodes imbriquées s'il y en a. Le bouton orange entre dans les appels imbriqués de méthodes. Le bouton violet exécute à pleine vitesse jusqu'à la sortie de la fonction puis s'arrête. Le bouton entouré en bleu foncé affiche les points d'arrêt. Nous n'emploierons pas les autres boutons.



Bouton d'exécution pas à pas

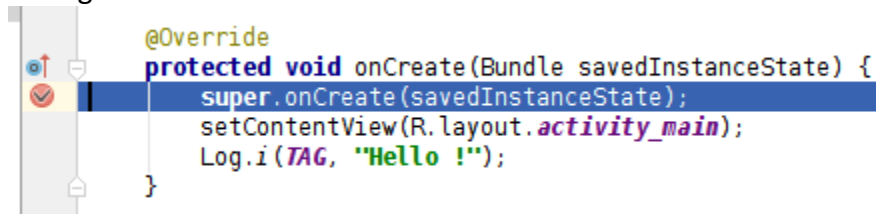
La mise au point repose sur ces deux techniques :

- Lancer en mode normal et attendre que ça se plante. L'exécution s'arrête alors exactement sur l'instruction qui est mauvaise, par exemple celle qui déclenche une NullPointerException.

- Mettre un *point d'arrêt* avant l'endroit qu'on pense mauvais puis exécuter ligne par ligne pour voir ce qu'il y a dans les variables et guetter l'erreur.

Placement d'un point d'arrêt

Cliquez dans la marge gauche au début de la méthode onCreate(). Ça place un gros point rouge dans la marge.



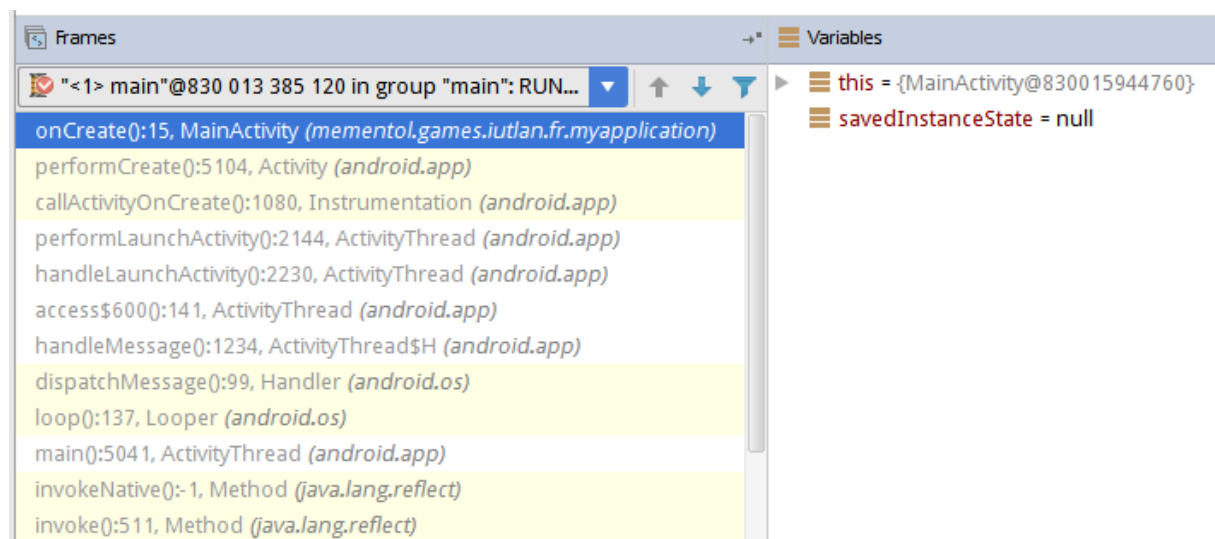
Point d'arrêt dans le source

La fenêtre Breakpoints, qu'on obtient en cliquant sur le bouton entouré en bleu foncé dans la figure précédente, liste tous les points d'arrêts que vous avez placés. N'en mettez pas trop dans un programme, vous aurez du mal à suivre tout ce qui peut se passer. Mettez vos méthodes au point les unes après les autres, pas tout en même temps.

Examen de la pile d'appels

Lorsque le programme s'arrête, soit sur une erreur non prévue, soit sur un point d'arrêt, Studio affiche la pile d'exécution (imbrication des appels de méthodes) dans la fenêtre Debug.

Le mot Frames désigne les appels imbriqués. Ils contiennent le nom de la méthode appelée, ses paramètres et ses variables locales.



Pile d'appels

Une grande partie des appels ne viennent pas de votre application mais du système Android. Un autre problème est qu'une application Android implique plusieurs *threads*. Parfois, il faut arriver à comprendre quel thread plante.

Exécution pas à pas

On va maintenant tenter de réparer un programme qui ne fonctionne pas. Remplacez le source de MainActivity par tout ceci :

```

package RNU.ISETZG.tp1;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import java.util.Arrays;
public class MainActivity extends AppCompatActivity
{
    private TextView textView;
    private final int[] tableau = new int[]{1, 3, 6, 8, 9};
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = findViewById(R.id.textview);
        display(tableau);
    }
    private void display(int[] tab)
    {
        textView.setText(Arrays.toString(tab));
    }
    /** appelée quand on clique sur le bouton */
    public void onClick(View view)
    {
        permutation(tableau);
        display(tableau);
    }
    /**
     * fait une permutation circulaire croissante des valeurs :
     * [a, b, c, d] => [d, a, b, c]
     * @param tab tableau à permuter
     */
    private void permutation(int[] tab)
    {
        int dernier = tab[tab.length-1];
        for (int i=0; i<tab.length; i++) {
            recopierSurSuivant(tab,i);
        }
        tab[0] = dernier;
    }
    /**
     * place la valeur de la case pos du tableau tab dans la case pos+1
     * @param tab tableau à permuter
     * @param pos indice de l'élément à copier sur le suivant
     */
    private void recopierSurSuivant(int[] tab, int pos)
    {
        int courant = tab[pos];
        tab[pos + 1] = courant;
    }
}

```

```
}  
}
```

Et remplacez tout le fichier res/layout/main_activity.xml par ceci.

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
    tools:context=".MainActivity"  
    android:orientation="vertical">  
    <TextView  
        android:id="@+id/textview"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:gravity="center"/>  
    <Button  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:text="Permutation !"  
        android:onClick="onButtonClick"/>  
</LinearLayout>
```

Régalez LogCat pour n'afficher que des messages du niveau warn ou error.

Au lancement, cette application plante quand on clique sur le bouton. Cherchez l'exception dans le LogCat. Il y a un lien à cliquer pour arriver sur la ligne coupable. Corrigez cette erreur : mettez `length-1` au lieu de `length`.

S'il y a une autre exception, faites ce qu'il faut pour la corriger.

Ensuite, on voit que le programme ne fonctionne pas bien. Il ne plante plus mais ne semble pas faire ce qui est attendu : permuter les valeurs affichées. Faites comme si vous ne compreniez pas son fonctionnement. On va regarder vivre le programme en traçant son exécution.

Mettez un point d'arrêt sur la première instruction de la méthode `permutation` (ou sur la méthode elle-même), puis lancez en mode mise au point, et enfin cliquez sur le bouton de l'application pour déclencher le point d'arrêt.

D'abord, on va regarder le contenu de la variable `tableau`. Comme c'est elle qu'on va surveiller, mais qu'elle est passée en paramètre sous un nom différent dans plusieurs méthodes, on va ajouter une surveillance, un *watch* dessus. Cliquez sur la variable `tableau` dans le source et avec un clic droit, sélectionnez `Add to Watches....` Ça va la rajouter dans la liste des variables toujours affichées à droite. Dépliez-la (il faudra la déplier à chaque appel d'une nouvelle méthode). Comme c'est un petit tableau, ses valeurs sont toutes affichées. Déjà, on constate que l'ordre de ces valeurs est le même que sur l'écran de l'AVD, donc le bug ne vient probablement pas de l'affichage.

Puis essayez les modes suivants, en surveillant les valeurs de `tableau` et de l'indice `i` de la boucle :

- Instruction par instruction avec la touche F7 (step into) : elle rentre dans les appels aux méthodes. Suivez les modifications des variables dans la fenêtre Variables. Voyez que la variable dernier a pris la valeur de la dernière case du tableau. Ensuite, dans la boucle, vous voyez que l'exécution entre dans la méthode recopierSurSuivant et l'exécute également pas à pas.

Quand vous avez compris l'intérêt, vous pouvez couper court avec MAJ+F8 (step out) qui exécute le reste d'une méthode à pleine vitesse jusqu'à sa sortie.

- Ligne par ligne : appuyer sur F8 (step over) pour exécuter la prochaine ligne. S'il y a un appel à une méthode, il est fait à pleine vitesse. On se sert de ce mode quand on sait que les sous-méthodes sont bonnes.

Avez-vous pu comprendre ce programme ? Pourquoi fait-il du mauvais travail ? Sur Android aussi, on peut faire des erreurs d'algorithmique grossières. Corrigez le programme pour qu'il fonctionne correctement.

Supprimez l'exécution en cours à l'aide du bouton Stop app (carré rouge dans la barre d'outils verticale à gauche).