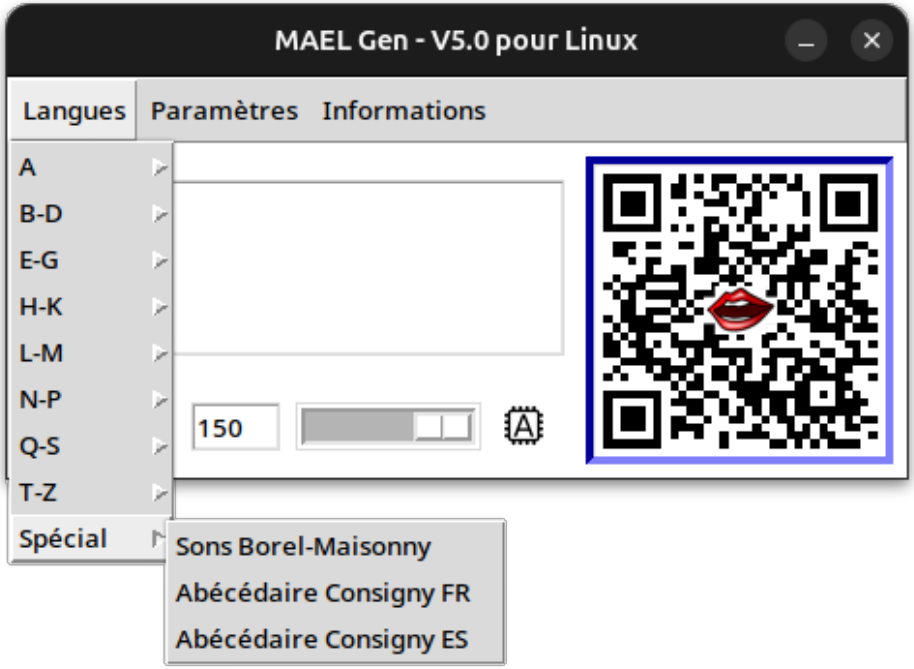
MAEL Gen / Scan

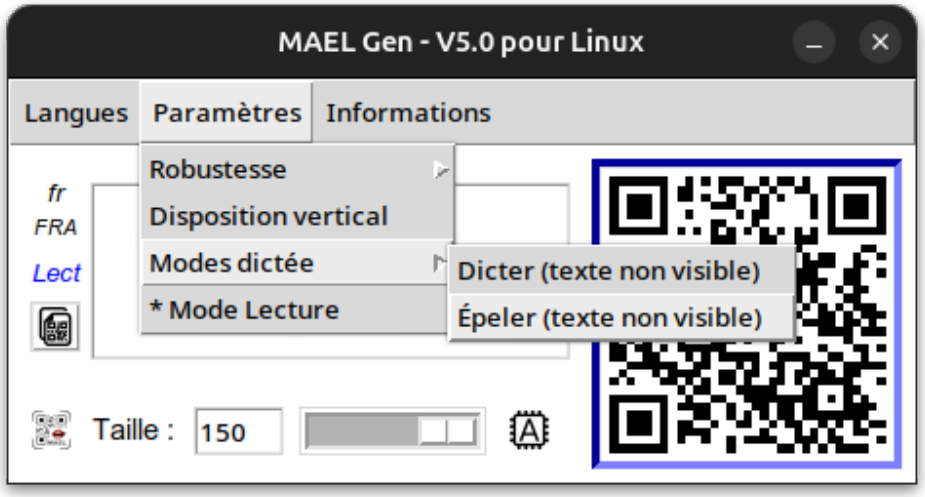
<u>Objectif</u> : Portage de MAEL Scan de « MIT App Inventor » vers « Kotlin MP » (Android et iOS)

1- Synthèse technique du fonctionnement de MAEL Gen

MAEL Gen permet au professeur de choisir la **<u>langue ou d'autres options</u>** :
- Écouter **les sons « Borel Maisonny »** avec une qualité mp3.
- Écouter les lettres et mots de l' « **Abécédaire Consigny » en français** avec une qualité mp3.
- Écouter les lettres et mots de l' « **Abécédaire Consigny » en espagnol** avec une qualité mp3.



MAEL Gen propose **<u>3 modes</u>** :
- Mode « **Lire** » → Mots prononcés et affichés.
- Mode « ~~**Dicter**~~ **Cacher** » → Mots prononcés mais non affichés.
- Mode « **Épeler** » → Caractères prononcés un par un, mais non affichés.
- Mode « **Dicter** » → Mots prononcés mais non affichés.
  → Ponctuation prononcée
  → Contrôle de la lecture affichée (Play / Pause / Stop)



<u>Robustesse</u> : plus elle est élevée, plus le code QR sera complexe (et devra être gros), plus il sera résistant à l'érosion.
<u>Disposition</u> : Verticale / Horizontale, pour une meilleure intégration dans l'espace de travail

2- Interprétation du contenu des codes QR par MAEL Scan

Avancement du développement : 🔴 🟡 🟢

Préfixe		Rôle en mode **Lecture**	Suffixe		
			#c (mode **Cacher**)	**#e** (mode **Épeler**)	**#d** (mode **Dictée**)
Langue (Synthèse vocale)	∅	Mots prononcés et affichés. Si préfixe absent : doit être interprété comme <frFRA> par défaut. 🟢	Mots prononcés en <frFRA> mais non affichés. 🟢	Caractères prononcés en <frFRA> mais non affichés. 🟢 → Nécessité d'un **dictionnaire de correspondances** chr / « à prononcer »	Mots prononcés en <frFRA> mais non affichés. → ponct. oralisée → Affichage lecteur 🔴
	<lgPAY> **proposés dans MAEL Phrase**	🟢 <esESP> (par défault) / <esUSA>	Idem en <xxXX> 🟢	Idem en <xxXX> 🟡	Idem en <xxXX> 🔴
		🟢 <enGBR> (par défault) / <enUSA> / ...	🟢	🟡	🔴
		🟢 <ptBRA> (par défault) / <ptPRT>	🟢	🔴	🔴
		🟢 <zhCHN>	🟢	Kanji prononcés un par un ? 🔴	🔴
		🟡 50 autres langages potentiellement déployables	🟡	🔴	🔴
Spécial (audios mp3)	***lg***	🟢 Aller chercher dans l'application elle-même les lettres et les mots de la « Frise Consigny » au format **mp3**. Disponibles : ***fr*** et ***es***	/	/	/
	-bm-	🟢 Aller chercher dans l'application elle-même les sons de la méthode « Borel Maisonny » au format **mp3**. Disponible en fr : **-bm-**	/	/	/
	/gd/	🟡 Utiliser l'ID Google Drive contenu dans le code QR pour aller chercher le fichier mp3 correspondant sur le Drive et le lire. → besoin d'internet ! (Pb de nb d'accès / min)	/	/	/
	/ml/	🔴 Utiliser l'ID Cloud MAEL contenu dans le code QR pour aller chercher le fichier mp3 correspondant sur le Cloud MAEL et le lire. → besoin d'internet !	/	/	/

PSEUDO-CODE KOTLIN

3- Utilisation des ‘class’

En Kotlin, le plus simple serait de créer une « class » (Impossible avec MIT App Inventor) depuis le contenu décodé du code QR :

```
public class qr_content {

   val lang: String = "fr"                          // from prefix
   val country: String = "FRA"                      // from prefix
   val special: String = ""                         // from prefix
   val message: String = "Bienvenue sur MAEL"       // from payload
   val mode: String = "lire"                        // from sufix

}
```

→ Utiliser les ‘**regex**’ pour extraire **lang** et **country** de <xx**XXX**> ! Et pour **#x** !

→ Pour que MAEL Gen puisse économiser les 3 lettres du pays, créer un dictionnaire des pays par défaut dans MAEL Scan et ne stipuler le pays que lorsqu’il ne s’agit pas du pays par défaut (donc seulement pour **fr**, **en**, **es** et **pt**) ⇒ Il faudra modifier MAEL Gen !!!

Utiliser **when** pour analyser ‘special’ et ‘mode’ et ‘lang’ quand ‘country’ n’est pas spécifié. (> pas en 4e position...)

```
when (qr_1.mode) {

        NULL -> mode("lire")
        "#d" -> mode("dicter")
        "#e" -> mode("epeler")

        else -> { println("Error of mode") }
}
```

```
when (qr_1.special) {

   NULL -> mode()  // Gérer les modes
   "-bm-" -> borel_m()
   "*fr*" -> abc_cons("fr")
   "*es*" -> abc_cons("es")
   "/gd/" -> google_drive()
   "/ml/" -> mael_drive()

   else -> { print("Error of special") }
}
```

Exemples d’instanciations :

```
val qr_1 = qr_content("es", "USA", "", "Hello", "dicter")
```

```
val qr_2 = qr_content("", "", "*fr*", "AVION", "")
```

```
val qr_3 = qr_content("", "", "-bm-", "oin", "")
```

4- Utilisation de l’objet créé depuis le code QR et la ‘class’

Si :

```
qr_content.special = NULL
```

Utiliser :

```
qr_content.lang
```

`qr_content.country`

Pour configurer la synthèse vocale…

`qr_content.mode`

pour gérer la sortie

5- Fonctions nécessaires

- Une fonction de **décryptage** (UTF-8) → déjà écrite 🟢
- Une fonction pour **instancier l'objet** correspondant au contenu du code QR (Avec des **regex**)
- Un fonction pour gérer les **modes**
- Une fonction pour gérer la sortie de la **synthèse vocale** en fonction de l'objet et du mode en cours
- Une fonction pour gérer la lecture audio locale des **spéciaux *xx* et -xx-** (Se mettre en mode « lire » en mode spécial)
- Un fonction pour gérer la lecture audio en streaming des **spéciaux /xx/** (Se mettre en mode « lire » en mode spécial)