# App Inventor 2を用いた語学学習アプリ作成の具体例

廣田, 大地

(Citation)
神戸大学国際コミュニケーションセンター論集,10:20-34

(Issue Date)
2013

(Resource Type)
departmental bulletin paper

(Version)
Version of Record

(JaLCDOI)
https://doi.org/10.24546/81005518

(URL)
https://hdl.handle.net/20.500.14094/81005518

# App Inventor 2 を用いた語学学習アプリ作成の具体例

廣田 大地 [1]

## 1. はじめに

近年スマートフォンやタブレット端末の普及により、教育現場でのアプリ活用の取り組みが活発に行われている。しかしながらその取り組みの内実はしばしば、使用法も定まらぬまま端末を大量購入するだけか、良くても既存のアプリをどのように授業の中に導入するかという点に終始しているように思われる。実際には今日、専門知識を持たなくてもアプリ開発が可能となっている中、教員が現場で培った知識がアプリ開発に十分生かされていないのは残念な限りである。そこで本稿においては、App Inventor というツールを用いて語学学習アプリを作成する方法を、筆者が専門とするフランス語教育を例にとって解説する。これにより、フランス語のみならず、様々な分野の教育関係者が自ら学習用アプリの開発に取り組むきっかけとなることを願う。

## 2. App Inventor について

App Inventor は、2010 年に Google 社が公開し、現在マサチューセッツ工科大学により提供されている Android アプリ作成ツールである。このツールでは、通常 Java で行うプログラミングの代わりに、様々なブロックをドラッグ・アンド・ドロップで組み合わせていくことで Android アプリを作成する。このツールを用いる利点の一つとして、導入のための準備がほとんど必要ないという点が挙げられる。Google アカウントを取得し、 http://ai2.appinventor.mit.edu にアクセスすれば、すぐに開発を始めることが出来る。また、使用するコンピュータと同一の無線 LAN ネットワークに接続した Android 端末があれば、容易に実機テストが行える。Eclipse 等一般的なアプリ開発環境を準備することと比較すると、圧倒的な利便性である [2]。

このツールのもう一つの優れた特徴として、初心者には敷居の高い難解なプログラミングの代わりに、レゴブロックを組み合わせるようにしてアプリ制作が行えるという点がある。それにより、視覚的・感覚的に各プログラムの役割を把握することが出来るため、プログラミングの経験の無い者が、アプリというシステムのおおまかな構造を把握するためにも有効である。あえて欠点を挙げるならば、初心者にも分かりやすいよう煩雑な設定が省略されてしまっているため本格的なプログラミングには向かないが、本稿で説明するような小規模のアプリであれば、十分なものを作り上げることが出来るだろう。

本稿では以下、まず初めに、フランス語入力を可能とするキーボードの機能を果たすアプリを作成する手

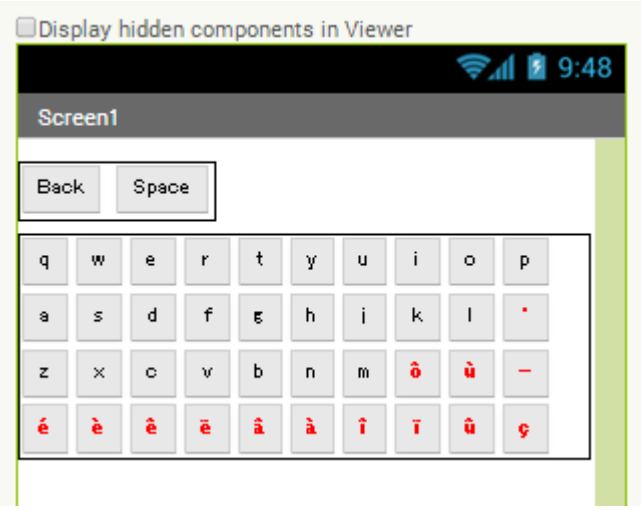
[1] 神戸大学国際コミュニケーションセンター　hirotadaichi@ruby.kobe-u.ac.jp

[2] そのような利便性は、2013 年末に行われた App Inventor2 へのバージョンアップによって達成されたのだが、それに伴い幾つかの設定変更が行われている。前バージョンに基づいた入門書であれば、日本語でも多数出版されているのだが、新バージョンに基づく入門書はまだ出版されていない。本稿の執筆はその点を補うという目的もある。

順を示す。スマートフォン等でもフランス語のアクセント記号の入力は可能だが、機種依存の機能であること、アプリの操作が一時中断されてしまうことを考慮すると、キーボードそのものをアプリに組み込んでしまうことが効果的である。続いて、そのアプリを基盤としたフランス語学習アプリの作成手順を示すこととする。

## 3. フランス語キーボードの作成

### 3. 1. 画面デザイン




新しくアプリ制作を始めるためには、まず好きな名前を付けて new Project を立ち上げる。画面右上に画面デザインを組み立てる「Designer」モードと、裏で動くプログラムを組み立てる「Blocks」モードとを切り替えるボタンがある。まずは「Designer」モードにおいて、右図のように各パーツを組み立てよう。パーツを組み立てる際には、画面左端の「Palette」の中にグループごとに配列された部品を「Viewer」の端末スクリーンの中にドラッグ・アンド・ドロップすれば良い。

それでは、入力した文字を表示するラベルのパーツを例に、実際の作業手順を解説しよう。まず「User Interface」グループの中から「Label」を探し、画面の中にドラッグしてみよう。すると自動的にパーツが配置される。配置されたパーツは画面やや右の「Components」リストにも表示されるので、「Label1」が選択されている状態で「Rename」をクリックし、「L_Answer」と変更する。多数のパーツを扱うため、名前は分かりやすく、かつ簡潔に付けることが望ましい。続いて、画面右端の「Properties」欄で各パーツの詳細設定を行う。「FontSize」を初期設定の 14 から 20 に変更し、「Text」にはパーツの初期名称が自動的に入っているため削除、「Width」を「Fill parent」に変更し親要素であるスクリーン自体と同じ横幅に広げる。以上でラベル「L_Answer」の設定は完了である。

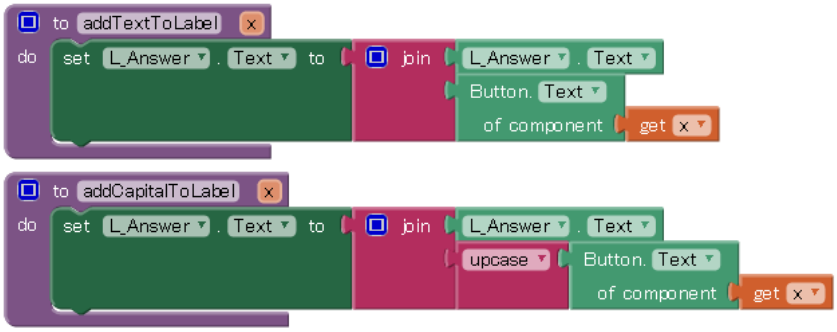
他のパーツに関しても、このような作業を下記の表に従って行うことで上掲キーボードの画面デザインを構築することが出来る。

| グループ | 部品の種類 | 名前 | プロパティ | プロパティ値 |
|---|---|---|---|---|
| User Interface | Label | L_Answer | FontSize | 20 |
| | | | Text | ※空にする |
| | | | Width | Fill Parent... |
| Layout | HorizontalArrangement | **HorizontalArrangement1** | | |
| Layout | TableArrangement | **TableArrangement1** | Columns | 10 |
| | | | Rows | 4 |
| | | | Width | Fill Parent... |
| **HorizontalArrangement1** の中 | | | | |
| User Interface | Button | Button__BackSpace | Text | Back |
| User Interface | Button | Button__Space | Text | Space |
| **TableArrangement1** の中 | | | | |

| | | | | |
|---|---|---|---|---|
| User Interface | Button | Button_q | Text | q |
| User Interface | Button | Button_w | Text | w |
| User Interface | Button | Button_e | Text | e |
| User Interface | Button | Button_r | Text | r |
| User Interface | Button | Button_t | Text | t |
| User Interface | Button | Button_m | Text | m |
| User Interface | Button | Button_e_aigu | Text | é |
| | | | TextColor | Red |
| User Interface | Button | Button_e_grave | Text | è |
| | | | TextColor | Red |
| User Interface | Button | Button__apostrophe | Text | ' |
| | | | TextColor | Red |

## 3. 2. ブロックの組み立て

画面設計が終わったら、続いて「Blocks」での作業に移ろう。キーボードアプリにおいて必要なのは、「もし○というキーが押されたら、画面に△と表示させる」といった内容の指示である。また、同じボタンが長押しされた際に、大文字を入力できるようにもしたい。しかしながら全てのボタンについてそのための指示を出すのは非効率なので、共通する部分は関数で共有し、「あるキーの名前が渡されたら、そのキーに表示されているテキストを画面に表示させる」という指示にする。大文字に関しても同様の関数を用意する。上図の 2 つのブロックがその関数になる。組み立て方は「Designer」モードと多少似ており、左端の「Blocks」リストから必要な部品をドラッグしていく。関数「addTextToLabel」のみ、以下に具体的な作業手順を記しておくので、それ以外のブロックについては、完成画像を参考に必要なブロックの場所を探しながら作業してみていただきたい。App Inventor は視覚的・感覚的に設計されているツールなので、しばらく作業をしていると自然と組み立て方が分かってくるはずである。




関数「addTextToLabel」の組み立て方

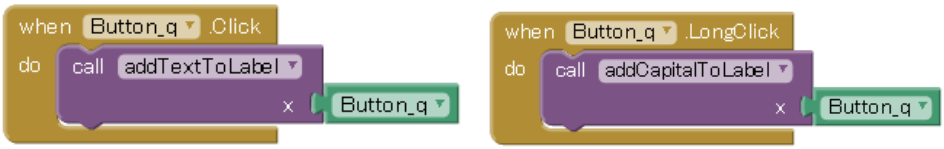
・「Built-in」＞「Procedures」をクリックして、開いたサブリストの中から一番上の「to procedure do」と書かれたブロックを、「Viewer」の好きな場所にドラッグする。続いて、ブロックの「procedure」と書かれた場所をクリックして、「addTextToLabel」と書き直す。

・「Screen1」＞「L_Answer」をクリックして、「set L_Answer.Text to」と書かれたブロックをドラッグして、図のように先ほど置いた紫色のブロックの中に結合させる。「カチッ」という音がすれば結合成功となる。

・「Built-in」＞「Text」をクリックして、開いたサブリストの中から「join」と書かれたブロックをドラックして、さきほどの緑色のブロックの右側に結合させる。

・再び「Screen1」>「L_Answer」をクリックして、今度は「L_Answer.Text」とのみ書かれたブロックをドラッグして、「join」ブロックの右上に結合させる。
・「Blocks」リスト最下部の「Any component」>「Any Button」をクリックして、「Button.Text of component」と書かれたブロックをドラッグして、「join」ブロックの右下に結合させる。
・紫色の「addTextToLabel」ブロック左上の歯車マークをクリックし、開いたサブウィンドウの中にある左側のパーツを右側のパーツに結合させる。するとのブロック右上にオレンジ色で「x」と書かれた部分が現れるので、そこにカーソルを合わせると、「get x」「set x to」という二つのブロックが浮かび上がる。「get x」の方をドラッグして、「Button.Text of component」ブロックの右に結合させる。

同様にして、関数「addCapitalToLabel」のブロックも組み立てるわけだが、この 2 つの関数ブロックを組み立てただけでは、キーボードは何の反応もしない。いずれかのキーボタンがタッチされたことを一つのイベントとして認識し、それを関数に結び付けるためのブロックが必要である。そこで、以下のようなイベント用ブロックを作成する。

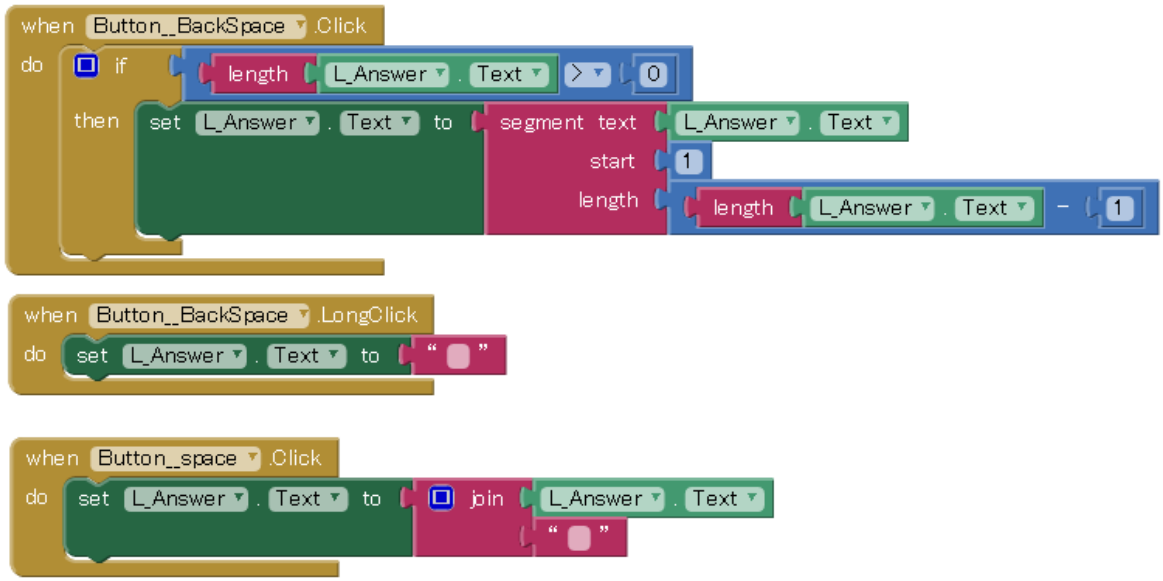



| | |
|---|---|
| イベント「Button_q.Click」 | Button_q が押された場合、部品「Button_q」を引数として、関数「addTextToLabel」を実行する。 |
| イベント「Button_q.LongClick」 | Button_q が長押しされた場合、部品「Button_q」を引数として、関数「addCapitalToLabel」を実行する。 |

この 2 つのブロックは、キーボタン「q」のタッチを認識して関数に送るだけのものである。そのため、Click イベントについては 38 個のキーボタン全てに、LongClick イベントについては大文字の必要無い「'」「-」を除く 36 個のキーボタンに対して、それぞれ設定していく必要がある。縮小した画面で表示すると、下図のように膨大なブロックが必要となる [3]。

[3] App Inventor の最大の弱点がこの点であると言えるだろう。通常のプログラミング言語であれば、類似した命令文を繰り返し書くだけで良いが、それを視覚的に表現している App Inventor では、同じようなブロックを大量に作成する必要がある。さらには、内部に嵌め込んだブロックが異なるため、単なるコピー＆ペーストでも対応しきれない。そのため本稿では、後述のとおり、キーボードアプリの完成データをダウンロードできるアドレスを公開している。

以上のイベント認識のためのブロックに加えて、今回の語学学習アプリにおけるキーボードとしての最低限の機能を果たすためには、空白入力ボタンと削除ボタンが必要である。そこで、最後に次の 3 つのブロックを組み立てる。




| | |
|---|---|
| イベント「Button__BackSpace.Click」 | ボタン「Button__BackSpace」がクリックされたとき、もしもラベル「L_Answer」のテキストが 0 文字より長ければ、ラベル「L_Answer」のテキストの内、最後の 1 文字を切り取ったものを、ラベル「L_Answer」のテキストに入力する。 |
| イベント「Button__BackSpace.LongClick」 | ボタン「Button__BackSpace」が長押しされたとき、ラベル「L_Answer」のテキストを空にする。 |
| イベント「Button__Space.Click」 | ボタン「Button__Space」がクリックされたとき、ラベル「L_Answer」のテキストに「 」（半角スペース）を加える。 |

以上でキーボードアプリの組み立ては完成となる。組み立て完了後、事前に Android 端末でアプリ「MIT AI2 Companion」をインストールして立ち上げた上で、パソコン側画面上端の「Connect」>「AI Companion」から表示されるコードを入力すれば、組み立てたアプリの実機テストがすぐにできるので試してみよう。

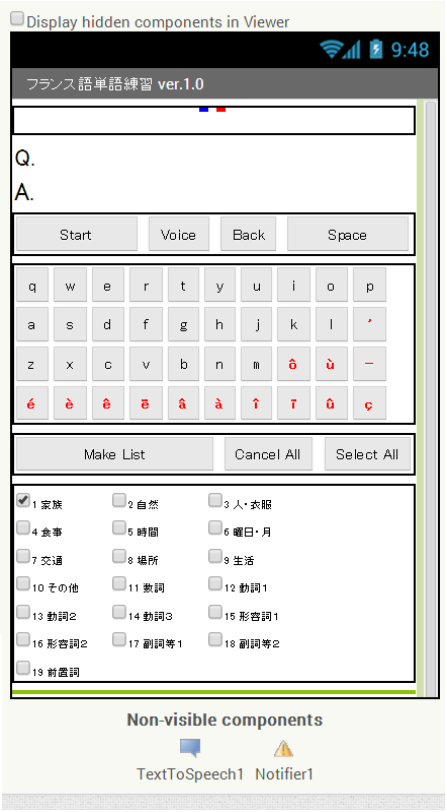
このキーボードアプリは、様々な語学学習アプリに応用することが可能ではあるが、同じようなブロックを繰り返し組み立てる必要があり、作成においては手間がかかる。そのため WEB 上にここまでの作業データを公開しておくので、必要があれば次のアドレスからダウンロードして使っていただきたい。

| |
|---|
| http://www.litterature.jp/numerique/French_Keyboard.aia |

ダウンロードしたファイルをデスクトップやマイドキュメントなどに保存し直した上で、App Inventor2 のウィンドウ上端にある「Project」>「Import project (.aia) from my computer」から、ダウンロードしたファイルを読み込むことが出来る。

## 4. フランス語学習アプリの作成

続いて本章では、前章において作成したフランス語キーボードアプリを基にして、フランス語学習アプリを作成する手順を示す。




### 4. 1. 画面デザイン

大まかな作業内容としては、作成済みの「入力内容表示部」「キーボード」に加えて、「スコア表示部」「問題表示部」「問題リスト作成ボタン」「問題リスト表示部」「作成された問題の日本語・フランス語表示部」を追加する。また、画面内には表示されないが、使用する機能として、自動音声機能「`TextToSpeech`」と通知ウィンドウ表示機能「`Notifier`」とを追加する。両機能も、パーツと同じく画面左端の「`Palette`」から探し、アプリ画面内にドラッグすれば自動的に組み込まれる。

完成レイアウトとしては右図のようになるが、そのために必要な部品とそのプロパティ設定について以下の表にまとめてある。背景色が薄青色になっている項目は、既に前章のキーボードアプリ作成時に設定済みであることを示している。

| グループ | 部品の種類 | 名前 | プロパティ | プロパティ値 |
|---|---|---|---|---|
| - | Screen | Screen1 | Title | フランス語単語練習 ver.1.0 |
| Layout | HorizontalArrangement | HorizontalArrangement2 | Width | Fill parent… |
| | | | AlignHorizontal | Center |
| User Interface | Label | L_Question | FontSize | 20 |
| | | | Width | Fill parent… |
| | | | Text | Q. |
| User Interface | Label | L_Answer | Text | A. |
| Layout | HorizontalArrangement | HorizontalArrangement1 | ※変更無し | |
| Layout | TableArrangement | TableArrangement1 | ※変更無し | |
| Layout | HorizontalArrangement | HorizontalArrangement3 | Width | Fill parent… |
| Layout | TableArrangement | TableArrangement2 | Columns | 3 |
| | | | Rows | 7 |
| | | | Width | Fill parent… |
| UserInterface | Label | L_WordList | Text | ※空にする |

| | | | | |
|---|---|---|---|---|
| | | | Width | Fill parent... |
| Media | TextToSpeech | TextToSpeech1 | Country | FRA |
| | | | Language | fra |
| UserInterface | Notifier | Notifier1 | NotifierLength | Short |
| HorizontalArrangement2 の中 | | | | |
| User Interface | Label | L_Cleared | FontSize | 10 |
| | | | Text | ※空にする |
| | | | Width | 25 pixels |
| User Interface | Label | L_Blue | BackgroundColor | Blue |
| | | | Text | ※空にする |
| User Interface | Label | L_Red | BackgroundColor | Red |
| | | | Text | ※空にする |
| User Interface | Label | L_Rest | FontSize | 10 |
| | | | Text | ※空にする |
| | | | Width | 25 pixels |
| HorizontalArrangement1 の中 | | | | |
| User Interface | Button | Button__Start | Text | Start |
| User Interface | Button | Button_Voice | Text | Voice |
| HorizontalArrangement3 の中 | | | | |
| User Interface | Button | B_Make_List | Text | Make List |
| User Interface | Button | B_Cancel_All | Text | Cancel All |
| User Interface | Button | B_Select_All | Text | Select All |
| TableArrangement2 の中 | | | | |
| User Interface | CheckBox | CheckBox1 | Checked | TRUE |
| | | | Text | 1 家族 |
| User Interface | CheckBox | CheckBox2 | Text | 2 自然 |
| User Interface | CheckBox | CheckBox3 | Text | 3 人・衣服 |
| User Interface | CheckBox | CheckBox4 | Text | 4 食事 |
| User Interface | CheckBox | CheckBox5 | Text | 5 時間 |
| User Interface | CheckBox | CheckBox6 | Text | 6 曜日・月 |
| User Interface | CheckBox | CheckBox7 | Text | 7 交通 |
| User Interface | CheckBox | CheckBox8 | Text | 8 場所 |
| User Interface | CheckBox | CheckBox9 | Text | 9 生活 |
| User Interface | CheckBox | CheckBox10 | Text | 10 その他 |
| User Interface | CheckBox | CheckBox11 | Text | 11 数詞 |

| | | | | |
|---|---|---|---|---|
| User Interface | CheckBox | CheckBox12 | Text | 12 動詞１ |
| User Interface | CheckBox | CheckBox13 | Text | 13 動詞２ |
| User Interface | CheckBox | CheckBox14 | Text | 14 動詞３ |
| User Interface | CheckBox | CheckBox15 | Text | 15 形容詞１ |
| User Interface | CheckBox | CheckBox16 | Text | 16 形容詞２ |
| User Interface | CheckBox | CheckBox17 | Text | 17 副詞等１ |
| User Interface | CheckBox | CheckBox18 | Text | 18 副詞等２ |
| User Interface | CheckBox | CheckBox19 | Text | 19 前置詞 |

ブロックの名称や設定などは多少違いがあってもアプリとしては機能するが、唯一、19 個の CheckBox の Text を「1 家族」のように、必ず「半角数字＋半角スペース＋項目名」とするようにしてほしい。後述の関数「makeWordList」の中で「split at spaces」というブロックを用い、この CheckBox の Text を半角スペースで切り分けた 1 つ目の要素（＝1～19 の半角数字）を項目番号として使用するためである。

## 4. 2. ブロックの組み立て

画面デザインが終われば、先ほどのキーボードアプリ作成時と同じように、「Blocks」での作業に移ろう。今回は新たに「変数」を用いてアプリを組み立てていく。先ほどと同様、一例のみブロックの作業手順の詳細を次に記す。

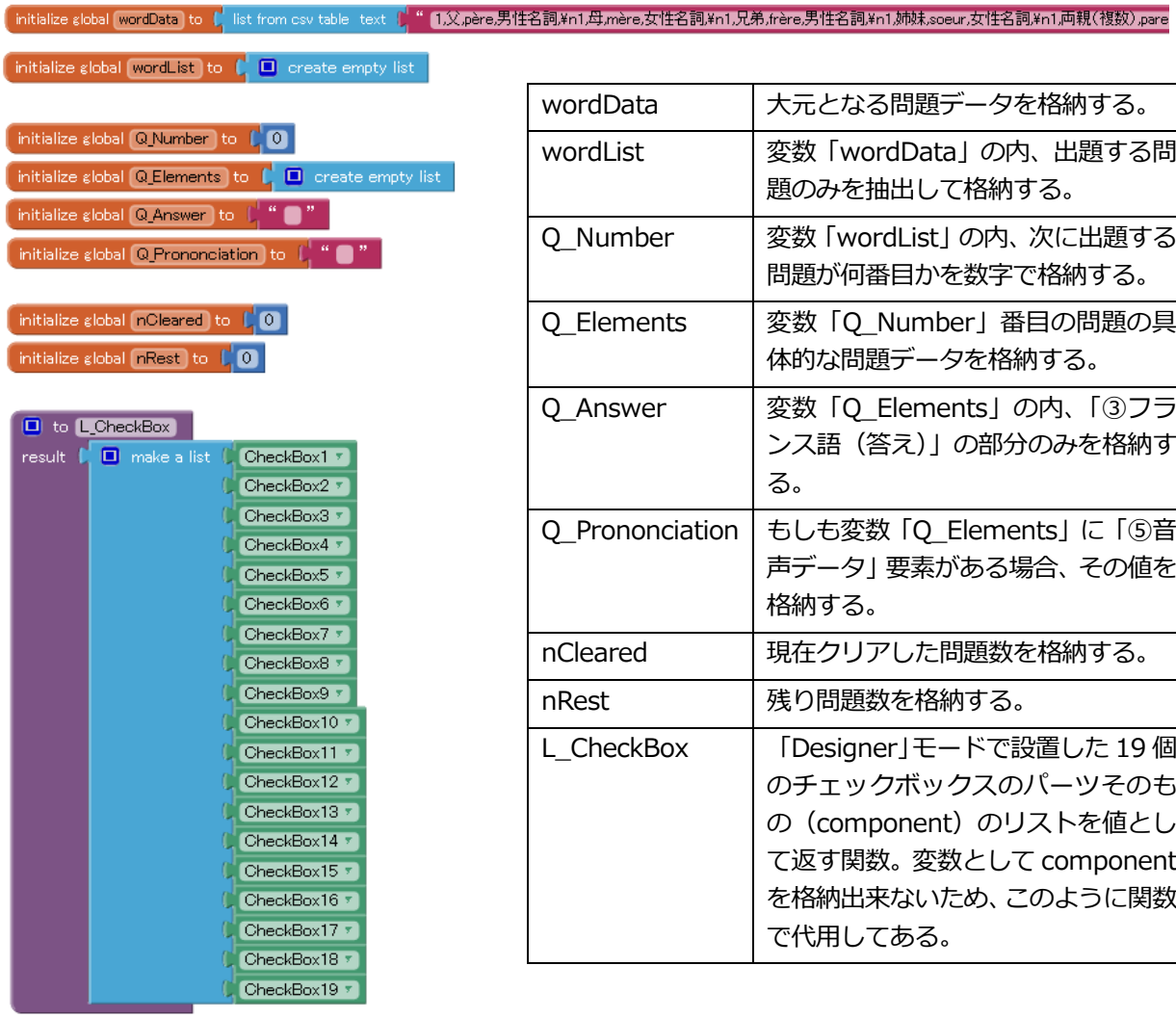
変数「wordData」の組み立て方

・「Built-in」＞「Variables」をクリックして、開いたサブリストの中から一番上の「initialize global name to」と書かれたブロックを、「Viewer」の好きな場所にドラッグする。そのブロックの「name」と書かれた場所をクリックして、「wordData」と書き直す。続いて、「Built-in」＞「Lists」をクリックして、開いたサブリストの中から「list from csv table text」と書かれたブロックをドラックして「wordData」の右にはめ込む。「Built-in」＞「Text」をクリックして、開いたサブリストの中から一番上の「"　"」と書かれたブロックを先ほどのパーツの右にはめ込む。最後に、「Text」ブロックの空白部分に単語リストのデータを入力する。

単語リストのデータ形式は、①項目番号、②日本語（問題）、③フランス語（答え）、④品詞、⑤音声データとなっている。通常「⑤音声データ」は空欄だが、自動音声が正しく作成されない綴りの場合、正しい発音に近いものを出力するための綴りを特別に入力している。例えば、第一番目の単語データである「1,父,père,男性名詞,」は、「1」が CheckBox1 の Text に入力した「1 家族」に対応する①項目番号であり、「父」が「②日本語（問題）」、「père」がユーザに入力を求める「③フランス語（答え）」、「男性名詞」は問題出題時に補足情報として提示される「④品詞」となっている。「③フランス語（答え）」である「père」という綴りは、Android の自動音声作成機能で正しく音声が作成されるため、「⑤音声データ」の入る箇所には区切り記号の「,」を入れるのみでデータは無い。ちなみにそれぞれの単語データは改行記号である「¥n」によって区切られている。データ量が多いので、言語コード UTF-8 のテキスト形式で保存したデータを次のアドレスのページからダウンロード、もしくは表示された画面から直接コピーして使っていただきたい。

```
http://www.litterature.jp/numerique/French_Quiz_wordData.txt
```

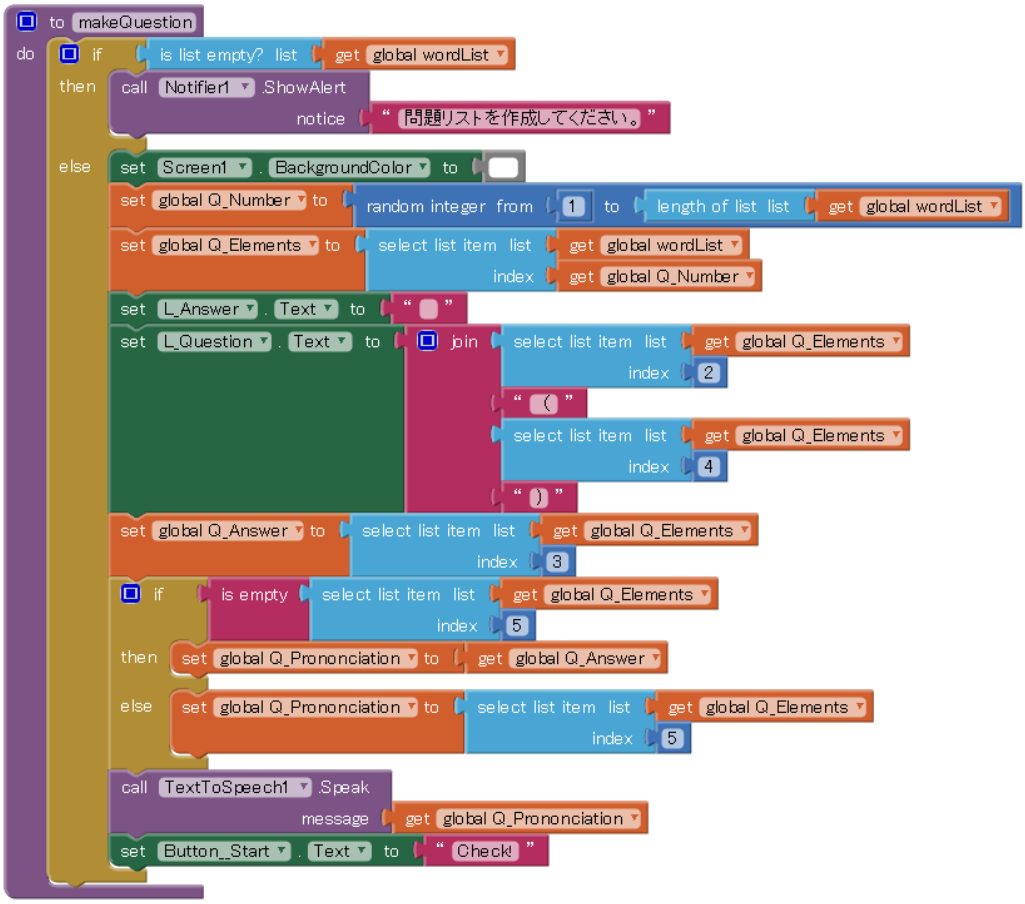
他にも幾つかの変数を作成するので、ブロックの完成図とともに、名称と役割の一覧を以下に記す。




| | |
|---|---|
| wordData | 大元となる問題データを格納する。 |
| wordList | 変数「wordData」の内、出題する問題のみを抽出して格納する。 |
| Q_Number | 変数「wordList」の内、次に出題する問題が何番目かを数字で格納する。 |
| Q_Elements | 変数「Q_Number」番目の問題の具体的な問題データを格納する。 |
| Q_Answer | 変数「Q_Elements」の内、「③フランス語（答え）」の部分のみを格納する。 |
| Q_Prononciation | もしも変数「Q_Elements」に「⑤音声データ」要素がある場合、その値を格納する。 |
| nCleared | 現在クリアした問題数を格納する。 |
| nRest | 残り問題数を格納する。 |
| L_CheckBox | 「Designer」モードで設置した 19 個のチェックボックスのパーツそのもの（component）のリストを値として返す関数。変数として component を格納出来ないため、このように関数で代用してある。 |

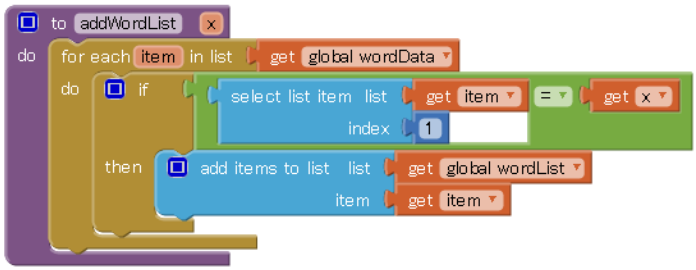
変数ブロックには「`global`」という表記があるのでその意味を解説しておこう。変数にはプログラムのどこででも呼び出せる「グローバル変数」と、その変数を定めた領域内でしか呼び出せない「ローカル変数」とがあり、「`initialize`」によって一度定義された上記の変数はグローバル変数として、プログラムのどこででも用いることが出来る。反対に、関数の中で引数をもとにしたローカル変数「`x`」と、配列変数をもとにしたループ構文で使われるローカル変数「`item`」は、それ以外の場所で再利用することは出来ない。

変数ブロックの組み立てが終わったら、関数ブロックとイベントブロックとを組み立てていこう。以下のページに組み立て完成図と各ブロックの機能とを記すので、同じように組み立てていただきたい。なお、既に作成したブロックを参照することがあるので、以下に提示されている順番で組み立てていくことを勧める。



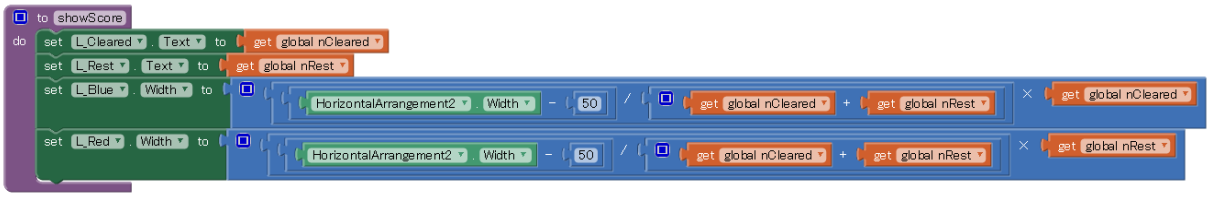

関数「makeQuestion」

| | |
|---|---|
| ・もし、配列変数「wordList」が空ならば、 | |
| | 「問題リストを作成してください」というテキストで注意アラートを表示させる。 |
| 空でなければ、 | |
| | ・画面全体の背景色に「白」を設定する。<br>・変数「Q_Number」に、1～配列変数「wordList」のリスト数の間で作ったランダム数を入力する。<br>・変数「Q_Elements」に、配列変数「wordList」の変数「Q_Number」番目の変数を入力する。<br>・ラベル「L_ANswer」のテキストを空にする。<br>・ラベル「L_Question」のテキストに、配列変数「Q_Elements」の２番目の要素、「(」、配列変数「Q_Elements」の４番目の要素、「)」の４つを足したものを入力する。<br>・変数「Q_Answer」に配列変数「Q_Elements」の３番目の要素を入力する。 |
| | ・もし、配列変数「Q_Elements」の５番目の要素が空であれば、 |
| | | ・変数「Q_Prononciation」に、変数「Q_Answer」の値を入力する。 |
| | 空でなければ、 |
| | | ・配列変数「Q_Elements」の５番目の要素を変数「Q_Prononciation」に入力する。 |
| | ・変数「Q_Prononciation」の値をもとに、フランス語の自動音声を出力する。<br>・ボタン「Button__Start」のテキストを「Check!」にする。 |



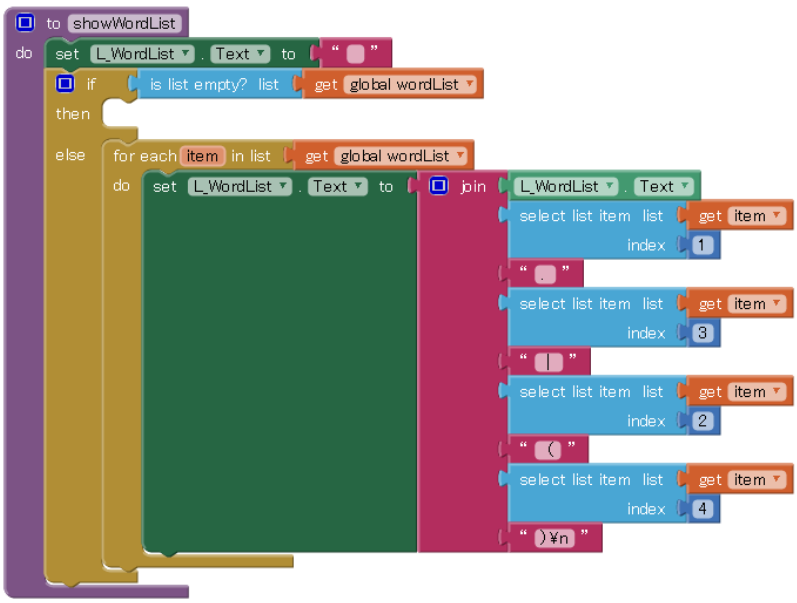

関数「addWordList」

| | | |
|---|---|---|
| ・配列変数「wordData」の中身を「item」として、その数だけ以下の作業を繰り返す。 | | |
| | ・もしも、「item」の１番目の項目が、この関数の引数「x」と等しければ、 | |
| | | 配列変数「wordList」に「item」の中身を加える。 |



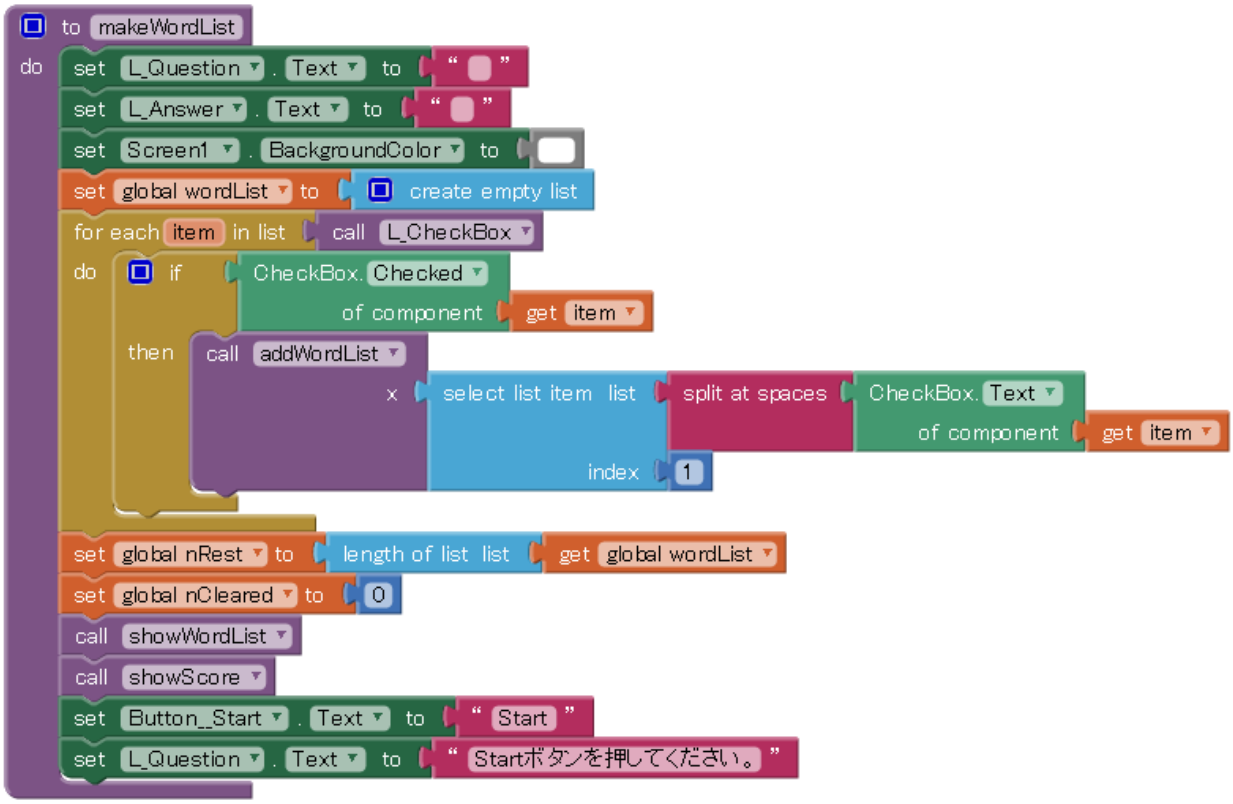

関数「showScore」

・ラベル「L_Cleared」のテキストに、変数「nCleared」の値を入力する。
・ラベル「L_Rest」のテキストに、変数「nRest」の値を入力する。
・ラベル「L_Blue」の幅を、(レイアウト「HorizontalArrangement2」の幅-50)/(変数「nCleared」の値+変数「nRest」の値)* 変数「nCleared」の値に変更する。
・ラベル「L_Red」の幅を、(レイアウト「HorizontalArrangement2」の幅-50)/(変数「nCleared」の値+変数「nRest」の値)* 変数「nRest」の値に変更する。



関数「showWordList」

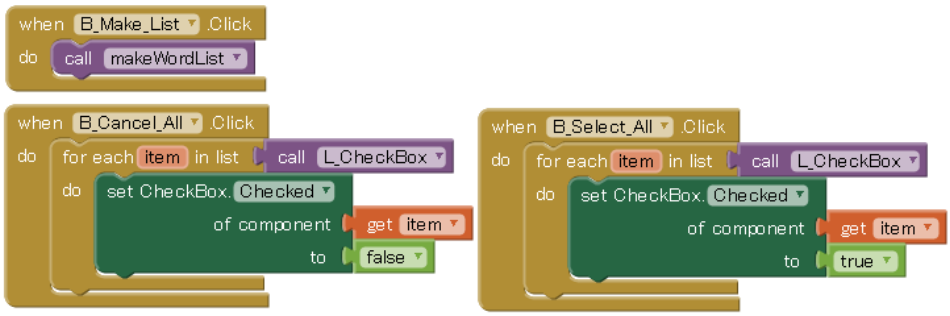
| | | |
|---|---|---|
| ラベル「L_WordList」のテキストを空にする。 | | |
| もしも、配列変数「wordList」が空ならば、 | | |
| | （※何もしない） | |
| 空でないならば、 | | |
| | 配列変数「wordList」の各要素を「item」として、その数だけ以下の作業を繰り返す。 | |
| | | ラベル「L_WordList」のテキストに、「item」の１番目、"."、「item」の３番目、"|"、「item」の２番目、"（"、「item」の４番目、"）"+「改行」を組み合わせたものを加える。 |



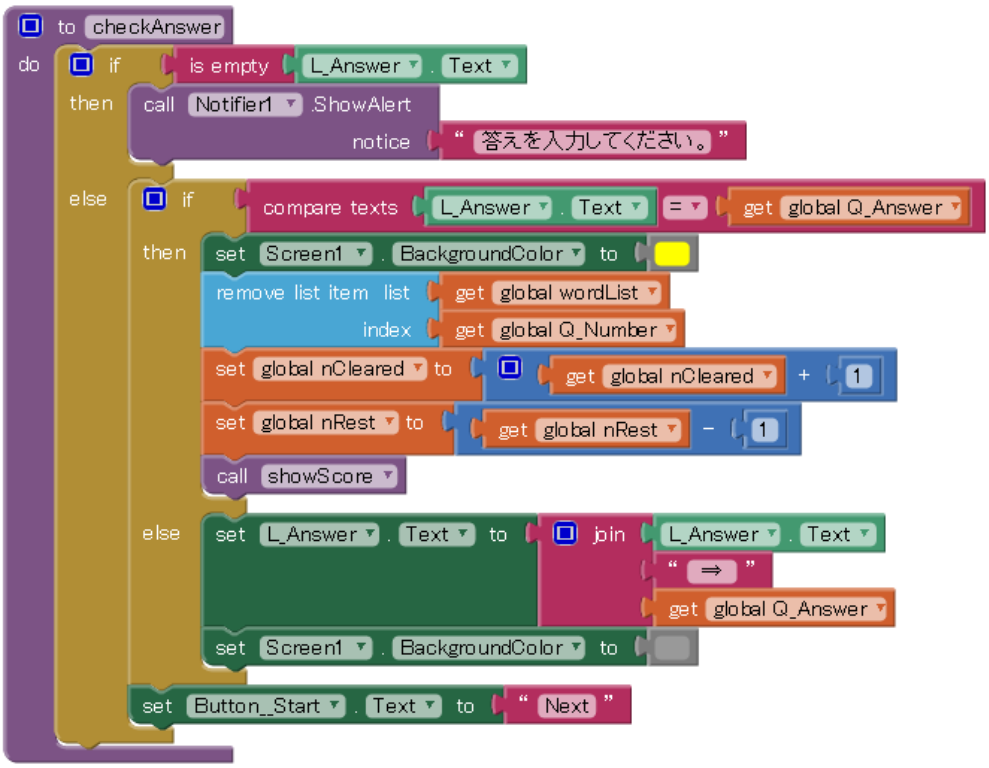

関数「makeWordList」

| | | |
|---|---|---|
| ・ラベル「L_Question」のテキストを空にする。<br>・ラベル「L_Answer」のテキストを空にする。<br>・画面全体の背景色を「白」にする。<br>・配列変数「wordList」を空にする。<br>・関数「L_CheckBox」の中身を「item」として、その数だけ以下の作業を繰り返す。 | | |
| | ・もし、チェックボックス「item」がチェックされているならば、 | |
| | | ・チェックボックス「item」のテキストを「半角スペース」で分割した一番初めの要素を引数として、関数「addWordList」を実行する。 |
| ・変数「nRest」に配列変数「wordList」の個数を入力する。<br>・変数「nCleared」に「0」を入力する。 | | |

| |
|---|
| ・関数「showWordList」を実行する。<br>・関数「showScore」を実行する。<br>・ボタン「Button__Start」のテキストを「Start」に変更する。<br>・ラベル「L_Question」のテキストを空にする。 |

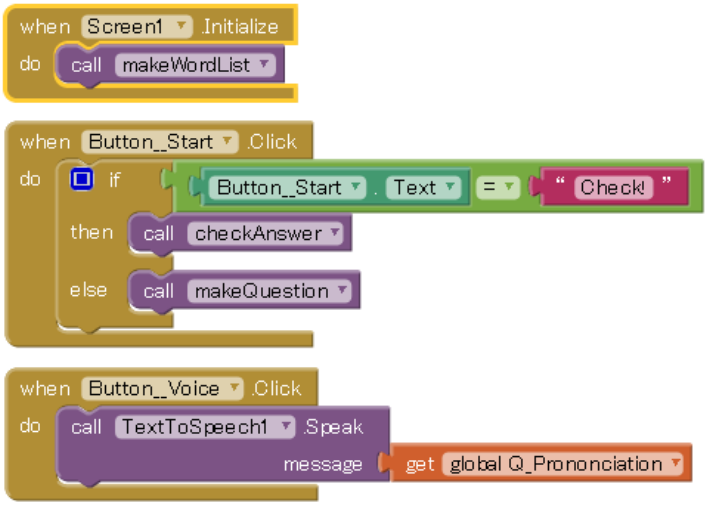



| | |
|---|---|
| イベント「B_Make_List.Click」 | ボタン「B_Make_List」が押された場合、関数「makeWordList」を実行する。 |
| イベント「B_Cancel_All.Click」 | ボタン「B_ Cancel_All」が押された場合、関数「L_CheckBox」の中身を「item」として、その数だけ、「item」のチェックをオフにするという作業を繰り返す。 |
| イベント「B_Select_All.Click」 | ボタン「B_Select_All」が押された場合、関数「L_CheckBox」の中身を「item」として、その数だけ、「item」のチェックをオンにするという作業を繰り返す。 |



関数「checkAnswer」

| |
|---|
| ・もし、ラベル「L_Answer」のテキストが空ならば、 |
| ・「答えを入力してください。」というテキストで注意アラートを表示させる。 |
| 空でなければ、 |
| ・もし、ラベル「L_Answer」のテキストが、変数「Q_Answer」と等しければ、 |
| ・画面全体の背景色を「黄色」に設定する。<br>・配列変数「wordList」から、変数「Q_Number」番目の要素を削除する。<br>・変数「nCleared」に 1 を加える。<br>・変数「nRest」から 1 を引く。<br>・関数「showScore」を実行する。 |
| 空でなければ、 |
| ・配列変数「Q_Elements」の 5 番目の要素を変数「Q_Prononciation」に入力する。 |
| ・変数「Q_Prononciation」の値をもとに、フランス語の自動音声を出力する。<br>・ボタン「Button__Start」のテキストを「Check!」にする。 |




| | |
|---|---|
| イベント「Screen1.Initialize」 | アプリ画面の起動時に、関数「makeWordList」を実行する。 |
| イベント「Button__Start.Click」 | ボタン「Button__Start」が押された時、ボタン「Button__Start」のテキストが「Check!」と等しければ関数「CheckAnswer」、等しくなければ関数「makeQuestion」を実行する。 |
| イベント「Button__Voice.Click」 | ボタン「Button__Voice」が押された時、変数「Q_Prononciation」の値を用いて自動音声の読み上げ機能を実行する。 |

全てのブロックの組み立てが終わったら、キーボードアプリの時と同じように、Android 端末での実機テストを試してみよう。また、App Inventor ウィンドウの上部「Build」>「APP (Save .apk to my computer)」から、Android アプリファイルとしてデータを保存すれば、メールの添付ファイルなどで他の Android 端末にアプリを送りインストールすることも可能である。

## 5. まとめ

以上、フランス語学習アプリを例として、App Inventor によるアプリ制作方法を解説してきた。それぞれのブロックの役割などは特に説明していないが、とりあえず完成図を見ながら各ブロックを探し出して、同じものを組み立て、全て組み立て終わったら、実機上で実際に動くことを確認していただきたい。そうすれば、Android アプリ制作というものがそれほど難しいものではないことが実感できることだろう。

もう少しアプリ制作に取り組んでみたいという方には、完成したフランス語学習アプリを基に、様々な変更を加えていくことで自分独自のアプリを作り上げていくことをお勧めする。とりわけフランス語以外の言語教育に携わっている方には、フランス語から他言語への問題変更に取り組んでみていただきたい。その際には、変数「wordData」だけでなく、「Designer」モードの「TextToSpeech」の言語設定も変更する必要があるだろう。また、1～19 までに分類した問題リストの項目番号を改める場合には、「Designer」モードの「CheckBox」だけでなく、「Blocks」の関数「L_CheckBox」との対応も確認しておく必要がある。

本稿においては、最低限の解説しか行うことが出来なかったが、作業を進める中で様々な疑問が出てくることだろう。その際には、Google などの検索エンジンで他の App Inventor ユーザ等により書かれた解説を検索しながら解決して行っていただきたい。日本語での解説はいまだ不十分ではあるが、英語で書かれた解説はネット上に豊富にある。また、Youtube には動画でのブロック組み立て解説なども挙げられているので、より全般的な解説を見たいときには役立つだろう。

百聞は一見に如かず。少しでも興味を抱いた方は次のアドレスからさっそく App Inventor を始めてみてはいかがだろうか。

| http://ai2.appinventor.mit.edu |
|---|