ArduinoMenu

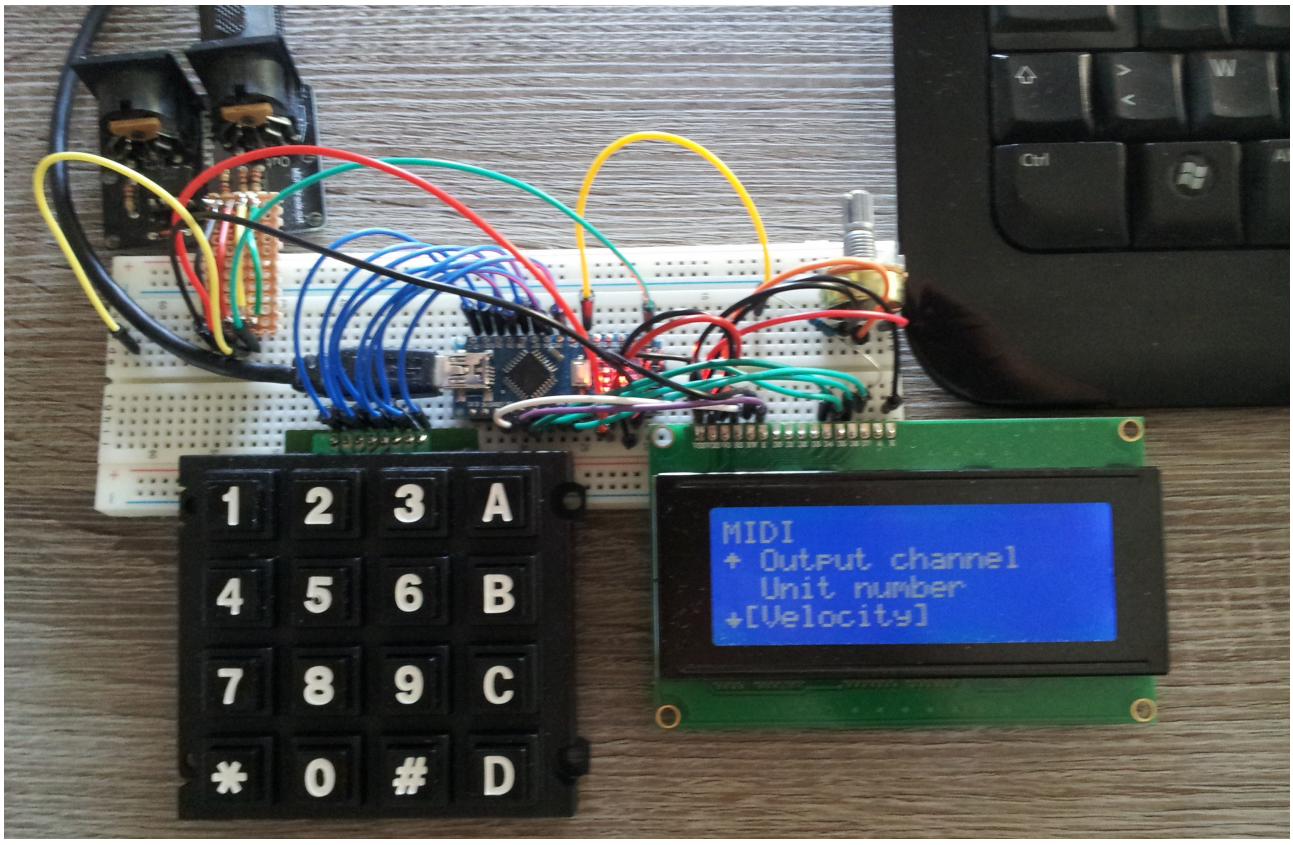


Table des matières

Partie menu

Définition de "Menu"

Un menu est une suite d'objets listés sur un écran. On peut s'y déplacer et modifier les objets. Voici une liste non exhaustive des objets que l'on peut rencontrer dans un menu.

- Sous-menu Il contient d'autres listes d'objects. On va également l'appeler *menu*.
- Variable On peut la voir et/ou la modifier.
- Fonction On peut l'exécuter.

Quelque soit son type, un objet a besoin d'un *label* pour être affichable dans le menu. Il faut faire la distinction entre la navigation et les objets. La navigation est le déplacement depuis un objet jusqu'à un autre. L'objet sert parfois à naviguer (cas du sous-menu) mais c'est une exception. Si on sélectionne un sous-menu, on ouvrira la liste de ses objects. Si on sélectionne une variable, c'est bien évidemment pour la lire ou la modifier. Si on sélectionne une fonction, c'est pour l'exécuter. Mais **tous les objets doivent être "navigable"**.

Représentation informatique d'un menu

Informatiquement parlant, un menu est un arbre dont chaque branche est un objet. La représentation la plus simple d'un arbre est le fichier XML.

Interpréter à la volée un fichier XML est une tâche lourde, c'est pourquoi elle ne sera pas faite dans le microcontrôleur mais en amont, au moment de la compilation, voire avant. Ecrivons donc d'abord sous forme XML le descriptif de ce que l'on voudrait pour notre menu.

- type menu, variable, fonction. Désormais, on mélange menus et sous-menus.
- cname Nom en language C de l'objet.
- label Texte qui représente l'objet deans le menu.

En XML, cette structure s'impose quasiment d'elle même. Faison simple, par exemple on ne va pas essayer d'écrire en différentes langues avec des caractères accentués... On se contentera de l'anglais qui a l'avantage de n'utiliser que des caractères ascii parfaitement définis.

Exemple de menu en XML

```
<?xml version="1.0" encoding="UTF-8"?>
<menuTree>
    <menu cname="globalEdit" label="GLOBAL  EDIT">
       <variable cname="channelIn" label="Input channel" />
       <variable cname="channelOut" label="Output channel" />
       <variable cname="programNumber" label="Program number" />
       <variable cname="arpeggiator" label="Arpeggiator" />
        <variable cname="clockIn" label="Midi clock in" />
        <variable cname="clockOut" label="Midi clock out" />
        <variable cname="keyClick" label="Keyboard echo" />
        <variable cname="audioBeat" label="Metronome" />
        <variable cname="sysEx" label="systemExclusive" />
        <variable cname="transposition" label="Transposition" />
        <function cname="dumpAll" label="Dump all" />
        <function cname="loadAll" label="Load all" />
        <function cname="saveAll" label="Save all" />
        <function cname="dumpGlobal" label="Dump global par." />
        <function cname="loadGlobal" label="Load global par." />
        <function cname="saveGlobal" label="Save global par." />
    </menu>
    <menu cname="sequenceEdit" label="SEQUENCE EDIT">
       <variable cname="groove" label="Groove" />
       <variable cname="gateMode" label="Gate duration" />
       <variable cname="lastStep" label="Last step" />
       <variable cname="ccNum" label="Ctrl chg number" />
        <function cname="initSeq" label="initialize seq." />
        <function cname="swapSeq" label="Swap sequences" />
        <function cname="copySeq" label="copy sequence" />
        <function cname="loadFactory" label="Load factory seq" />
        <function cname="dumpSeq" label="Dump sequence" />
        <function cname="loadSeq" label="Load sequence" />
        <function cname="saveSeq" label="Save sequence" />
    </menu>
    <menu cname="maintenance" label="MAINTENANCE">
        <function cname="factorySettings" label="Factory settings" />
        <function cname="testStepLeds" label="Test step leds" />
        <function cname="testIndLeds" label="Test ind. leds" />
        <function cname="testBeeper" label="Test Beeper" />
        <function cname="displayVersions" label="Versions" />
        <menu cname="submenu" label="TESTS">
            <variable cname="test1" label="test 1" />
            <variable cname="test2" label="test 2" />
            <variable cname="test3" label="test 3" />
            <variable cname="appVersion" label="Version number" />
        </menu>
    </menu>
</menuTree>
```

Autre représentation d'un menu

Une autre façon de représenter un menu est la liste indentée où chaque niveau de branche se représente par une tabulation supplémentaire.

IDX	LABEL	PARENT	CHILD	NEXT	PREVIOUS
0	GLOBAL EDIT	---	1	17	---
1	Input channel	0	---	2	---
2	Output channel	0	---	3	1
3	Program number	0	---	4	2
4	Arpeggiator	0	---	5	3
5	Midi clock in	0	---	6	4
6	Midi clock out	0	---	7	5
7	Keyboard echo	0	---	8	6
8	Metronome	0	---	9	7
9	systemExclusive	0	---	10	8
10	Transposition	0	---	11	9
11	Dump all	0	---	12	10
12	Load all	0	---	13	11
13	Save all	0	---	14	12
14	Dump global par.	0	---	15	13
15	Load global par.	0	---	16	14
16	Save global par.	0	---	---	15
17	SEQUENCE EDIT	---	18	29	0
18	Groove	17	---	19	---
19	Gate duration	17	---	20	18
20	Last step	17	---	21	19
21	Ctrl chg number	17	---	22	20
22	initialize seq.	17	---	23	21
23	Swap sequences	17	---	24	22
24	copy sequence	17	---	25	23
25	Load factory seq	17	---	26	24
26	Dump sequence	17	---	27	25
27	Load sequence	17	---	28	26
28	Save sequence	17	---	---	27
29	MAINTENANCE	---	30	---	17
30	Factory settings	29	---	31	---
31	Test step leds	29	---	32	30
32	Test ind. leds	29	---	33	31
33	Test Beeper	29	---	34	32
34	Versions	29	---	35	33
35	TESTS	29	36	---	34
36	test 1	35	---	37	---
37	test 2	35	---	38	36
38	test 3	35	---	39	37
39	Version number	35	---	---	38

Fils de

GLOBAL EDIT, *SEQUENCE EDIT* et *MAINTENANCE* sont frères. *Input channel* est enfant de *GLOBAL EDIT*, *MAINTENANCE* est parent de *Factory settings*. *Arpeggiator* est frère précédent de *Midi clock in* et est également frère suivant de *Program number*. *Input channel* n'a pas de frère précédent.

Vous avez bien sûr compris que l'on cherche une information dans une table grâce à l'index de l'objet, et que cet index a été généré par la position de l'objet dans le fichier XML.

Tables de navigation

Notre module *arduinoMenu* devra dont gérer les 4 listes PARENT, CHILD, NEXT (pour *prochain frère*) et PREVIOUS (pour *frère précédent*) afin de gérer la navigation dans le menu.

Il faut une valeur NO_ENTRY (représentée dans la liste précédente par `---`) qui signifie *il n'y a pas de parent*, *il n'y a pas d'enfant*, *il n'y a pas de frère suivant ou de frère précédent*. Cette valeur ne peut pas être zéro, car le zéro est l'index du premier objet d'un tableau. Si on fait tenir cet index sur un byte, il devrait être la valeur 0xff, c'est à dire la dernière valeur. Notre menu peut ainsi avoir 255 entrées (index de 0 à 254), puisque la 256ème (à l'index 255) est réservée pour la valeur NO_ENTRY.

Le travail ne serait pas fini sans générer quelques autres tables qui ne servent pas à la navigation proprement dite, mais qui sont néanmoins indispensables. Il s'agit des tables de labels, de fonctions, de fonctions d'édition et d'affichage de variables.

Table de types d'objets

Tant que l'on se contente de naviguer, pas de problème. Mais à partir du moment où on sélectionne un objet, il faut connaitre son type pour savoir quelle action engager.

Table de labels

Pour être affichable dans le menu, chaque objet doit avoir un label. Il doit donc y avoir également une table de labels dans le module *arduinoMenu*. Et pourquoi ne pas prendre le cname comme label? Parce que le cname a beaucoup de contraintes, en particulier la plus gênante : Il ne doit pas contenir d'espace.

S'il faut vraiment écrire des labels en plusieurs langues, la technique classique est de supprimer le label et d'utiliser le cname de l'objet comme une clef de traduction. Au moment d'afficher le texte, on cherche sa traduction dans la table de la langue voulue grâce à la clef. Cette solution est trop lourde pour notre microcontrôleur, on se contentera de l'ascii pur et dur de l'anglais.

Un moteur de menu doit non seulement gérer la navigation, mais aussi l'affichage et le rafraichissement de l'écran, l'affichage et la modification des variables ainsi que l'exécution des fonctions... Sinon, ça ne sert strictement à rien de les faire apparaitre dans le menu! Dans le cas de *arduinoMenu*, ce sera fait à base de *callbacks*. Le callback est le seul moyen informatique de séparer les modules ayant une tâche précise. On peut le faire sans callback, mais le prix à payer est que les modules seront imbriqués...On ne pourra pas utiliser le module de menu sans utiliser le module d'affichage ou de saisie! Du coup, ce petit moteur de menu perdrait toute sa portabilité.

Table de fonctions

Ces fonctions seront du type *void functionName()* qui est la signature la plus courante en C. Quand le menu détectera que l'utilisateur appelle cette fonction, il devra l'exécuter.

Table des fonctions d'affichage/édition des variables

Ces fonctions seront du type *void functionName(byte action)*. Quand le menu détectera que l'utilisateur appelle cette fonction, il devra l'exécuter. Le paramètre byte pourra prendre les valeurs suivantes sou forme de `#define` :

- `MENU_BROWSER_DATA_INCREASE` Incrémenter et afficher la variable.

- `MENU_BROWSER_DATA_DECREASE` Décrémenter et afficher la variable.
- `MENU_BROWSER_DATA_JUST_DISPLAY` Juste afficher la variable.

Quand le menu détectera que l'utilisateur veut voir ou éditer une variable, il devra l'afficher, scruter l'organe de saisie (clavier, boutons, encodeur incrémental etc...) de l'utilisateur pour modifier la variable et la réafficher si est est modifiée... Comme notre menu ne contient que le nom de la variable, il va falloir fabriquer le nom de la fonction d'affichage et de la fonction de modification à partir du nom de la variable. Par exemple le nom de variable *lastStep* va servir à fabriquer le nom de fonction *editLastStep*. C'est ainsi que le moteur de menu se contente d'être un moteur de menu et délègue les autres tâches aux parties du programme qui savent le faire.

On se rend compte que fabriquer cet ensemble de tables en écrivant du code C est titanesque, sans parler du risque d'erreurs de saisie ni de déboguage. C'est pourquoi le script python ***menuGenerator.py*** a été créé. Il fait très exactement la création des tables citées plus haut en analysant le fichier XML décrivant le menu. Il évite ainsi un travail vraiment long et fastidieux, sans parler des risques d'errreurs humaines.

Bouts de code généré

A partir de l'exemple de fichier XML plus haut, voici les bouts de code généré par le script ***menuGenerator.py*** mis bout à bout. Ils peuvent être des tableaux C mais aussi de simples déclarations qui se suivent une à une. Les voici dans leur intégralité, quasiment sans commentaires, mais si vous avez suivi jusqu'ici, ça ne devrait par être un problème pour vous. Un header a été inséré, et si on lui rajoute la somme des bouts de code généré, on obtient le fichier *menuData.h* que voici dans son intégralité.

```
#ifndef menuData_h
#define menuData_h

/*
 * file : menuData.h
 * Copyright (c) Pfeuh <ze.pfeuh@gmail.com>
 * All rights reserved.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

/*******************************************/
/*                  WARNING!               */
/* This file is generated by an application, */
/*   if you edit it, it will be overwritten  */
/*         by the next generation!           */
/*******************************************/

#include <Arduino.h>
```

```
#define MENU_GENERATOR_PY_VERSION "1.00"

#define MENU_BROWSER_NB_ENTRIES 40
#define MENU_BROWSER_NB_VARIABLES 18
#define MENU_BROWSER_NB_FUNCTIONS 18

// Table of parent              00040
// Table of child               00040
// Table of next                00040
// Table of previous            00040
// Table of labels              00517
// Table of pointers to labels  00080
// Table of function callbacks  00080
// Table of edit callbacks      00080
// Table of types               00040
// ---------------------------------
// TOTAL                        00957

// functions suppposed ready to execute
extern void dumpAll();
extern void loadAll();
extern void saveAll();
extern void dumpGlobal();
extern void loadGlobal();
extern void saveGlobal();
extern void initSeq();
extern void swapSeq();
extern void copySeq();
extern void loadFactory();
extern void dumpSeq();
extern void loadSeq();
extern void saveSeq();
extern void factorySettings();
extern void testStepLeds();
extern void testIndLeds();
extern void testBeeper();
extern void displayVersions();

// edit & display functions of variables suppposed ready to execute
extern void editChannelIn(bool direction);
extern void editChannelOut(bool direction);
extern void editProgramNumber(bool direction);
extern void editArpeggiator(bool direction);
extern void editClockIn(bool direction);
extern void editClockOut(bool direction);
extern void editKeyClick(bool direction);
extern void editAudioBeat(bool direction);
extern void editSysEx(bool direction);
extern void editTransposition(bool direction);
extern void editGroove(bool direction);
extern void editGateMode(bool direction);
extern void editLastStep(bool direction);
extern void editCcNum(bool direction);
extern void editTest1(bool direction);
extern void editTest2(bool direction);
extern void editTest3(bool direction);
extern void editAppVersion(bool direction);

const byte parentTable[MENU_BROWSER_NB_ENTRIES] PROGMEM =
{
```

```
    /* 000 */ MENU_BROWSER_NO_ENTRY,
    /* 001 */ 0,
    /* 002 */ 0,
    /* 003 */ 0,
    /* 004 */ 0,
    /* 005 */ 0,
    /* 006 */ 0,
    /* 007 */ 0,
    /* 008 */ 0,
    /* 009 */ 0,
    /* 010 */ 0,
    /* 011 */ 0,
    /* 012 */ 0,
    /* 013 */ 0,
    /* 014 */ 0,
    /* 015 */ 0,
    /* 016 */ 0,
    /* 017 */ MENU_BROWSER_NO_ENTRY,
    /* 018 */ 17,
    /* 019 */ 17,
    /* 020 */ 17,
    /* 021 */ 17,
    /* 022 */ 17,
    /* 023 */ 17,
    /* 024 */ 17,
    /* 025 */ 17,
    /* 026 */ 17,
    /* 027 */ 17,
    /* 028 */ 17,
    /* 029 */ MENU_BROWSER_NO_ENTRY,
    /* 030 */ 29,
    /* 031 */ 29,
    /* 032 */ 29,
    /* 033 */ 29,
    /* 034 */ 29,
    /* 035 */ 29,
    /* 036 */ 35,
    /* 037 */ 35,
    /* 038 */ 35,
    /* 039 */ 35,
};

const byte childTable[MENU_BROWSER_NB_ENTRIES] PROGMEM =
{
    /* 000 */ 1,
    /* 001 */ MENU_BROWSER_NO_ENTRY,
    /* 002 */ MENU_BROWSER_NO_ENTRY,
    /* 003 */ MENU_BROWSER_NO_ENTRY,
    /* 004 */ MENU_BROWSER_NO_ENTRY,
    /* 005 */ MENU_BROWSER_NO_ENTRY,
    /* 006 */ MENU_BROWSER_NO_ENTRY,
    /* 007 */ MENU_BROWSER_NO_ENTRY,
    /* 008 */ MENU_BROWSER_NO_ENTRY,
    /* 009 */ MENU_BROWSER_NO_ENTRY,
    /* 010 */ MENU_BROWSER_NO_ENTRY,
    /* 011 */ MENU_BROWSER_NO_ENTRY,
    /* 012 */ MENU_BROWSER_NO_ENTRY,
    /* 013 */ MENU_BROWSER_NO_ENTRY,
    /* 014 */ MENU_BROWSER_NO_ENTRY,
    /* 015 */ MENU_BROWSER_NO_ENTRY,
    /* 016 */ MENU_BROWSER_NO_ENTRY,
```

```
    /* 017 */ 18,
    /* 018 */ MENU_BROWSER_NO_ENTRY,
    /* 019 */ MENU_BROWSER_NO_ENTRY,
    /* 020 */ MENU_BROWSER_NO_ENTRY,
    /* 021 */ MENU_BROWSER_NO_ENTRY,
    /* 022 */ MENU_BROWSER_NO_ENTRY,
    /* 023 */ MENU_BROWSER_NO_ENTRY,
    /* 024 */ MENU_BROWSER_NO_ENTRY,
    /* 025 */ MENU_BROWSER_NO_ENTRY,
    /* 026 */ MENU_BROWSER_NO_ENTRY,
    /* 027 */ MENU_BROWSER_NO_ENTRY,
    /* 028 */ MENU_BROWSER_NO_ENTRY,
    /* 029 */ 30,
    /* 030 */ MENU_BROWSER_NO_ENTRY,
    /* 031 */ MENU_BROWSER_NO_ENTRY,
    /* 032 */ MENU_BROWSER_NO_ENTRY,
    /* 033 */ MENU_BROWSER_NO_ENTRY,
    /* 034 */ MENU_BROWSER_NO_ENTRY,
    /* 035 */ 36,
    /* 036 */ MENU_BROWSER_NO_ENTRY,
    /* 037 */ MENU_BROWSER_NO_ENTRY,
    /* 038 */ MENU_BROWSER_NO_ENTRY,
    /* 039 */ MENU_BROWSER_NO_ENTRY,
};

const byte nextTable[MENU_BROWSER_NB_ENTRIES] PROGMEM =
{
    /* 000 */ 17,
    /* 001 */ 2,
    /* 002 */ 3,
    /* 003 */ 4,
    /* 004 */ 5,
    /* 005 */ 6,
    /* 006 */ 7,
    /* 007 */ 8,
    /* 008 */ 9,
    /* 009 */ 10,
    /* 010 */ 11,
    /* 011 */ 12,
    /* 012 */ 13,
    /* 013 */ 14,
    /* 014 */ 15,
    /* 015 */ 16,
    /* 016 */ MENU_BROWSER_NO_ENTRY,
    /* 017 */ 29,
    /* 018 */ 19,
    /* 019 */ 20,
    /* 020 */ 21,
    /* 021 */ 22,
    /* 022 */ 23,
    /* 023 */ 24,
    /* 024 */ 25,
    /* 025 */ 26,
    /* 026 */ 27,
    /* 027 */ 28,
    /* 028 */ MENU_BROWSER_NO_ENTRY,
    /* 029 */ MENU_BROWSER_NO_ENTRY,
    /* 030 */ 31,
    /* 031 */ 32,
    /* 032 */ 33,
    /* 033 */ 34,
```

```
   /* 034 */ 35,
   /* 035 */ MENU_BROWSER_NO_ENTRY,
   /* 036 */ 37,
   /* 037 */ 38,
   /* 038 */ 39,
   /* 039 */ MENU_BROWSER_NO_ENTRY,
};

const byte previousTable[MENU_BROWSER_NB_ENTRIES] PROGMEM =
{
   /* 000 */ MENU_BROWSER_NO_ENTRY,
   /* 001 */ MENU_BROWSER_NO_ENTRY,
   /* 002 */ 1,
   /* 003 */ 2,
   /* 004 */ 3,
   /* 005 */ 4,
   /* 006 */ 5,
   /* 007 */ 6,
   /* 008 */ 7,
   /* 009 */ 8,
   /* 010 */ 9,
   /* 011 */ 10,
   /* 012 */ 11,
   /* 013 */ 12,
   /* 014 */ 13,
   /* 015 */ 14,
   /* 016 */ 15,
   /* 017 */ 0,
   /* 018 */ MENU_BROWSER_NO_ENTRY,
   /* 019 */ 18,
   /* 020 */ 19,
   /* 021 */ 20,
   /* 022 */ 21,
   /* 023 */ 22,
   /* 024 */ 23,
   /* 025 */ 24,
   /* 026 */ 25,
   /* 027 */ 26,
   /* 028 */ 27,
   /* 029 */ 17,
   /* 030 */ MENU_BROWSER_NO_ENTRY,
   /* 031 */ 30,
   /* 032 */ 31,
   /* 033 */ 32,
   /* 034 */ 33,
   /* 035 */ 34,
   /* 036 */ MENU_BROWSER_NO_ENTRY,
   /* 037 */ 36,
   /* 038 */ 37,
   /* 039 */ 38,
};

/*   0 */ const char labelGlobalEdit[] PROGMEM = "GLOBAL  EDIT";
/*   1 */ const char labelChannelIn[] PROGMEM = "Input channel";
/*   2 */ const char labelChannelOut[] PROGMEM = "Output channel";
/*   3 */ const char labelProgramNumber[] PROGMEM = "Program number";
/*   4 */ const char labelArpeggiator[] PROGMEM = "Arpeggiator";
/*   5 */ const char labelClockIn[] PROGMEM = "Midi clock in";
/*   6 */ const char labelClockOut[] PROGMEM = "Midi clock out";
/*   7 */ const char labelKeyClick[] PROGMEM = "Keyboard echo";
/*   8 */ const char labelAudioBeat[] PROGMEM = "Metronome";
```

```
/*   9 */ const char labelSysEx[] PROGMEM = "systemExclusive";
/*  10 */ const char labelTransposition[] PROGMEM = "Transposition";
/*  11 */ const char labelDumpAll[] PROGMEM = "Dump all";
/*  12 */ const char labelLoadAll[] PROGMEM = "Load all";
/*  13 */ const char labelSaveAll[] PROGMEM = "Save all";
/*  14 */ const char labelDumpGlobal[] PROGMEM = "Dump global par.";
/*  15 */ const char labelLoadGlobal[] PROGMEM = "Load global par.";
/*  16 */ const char labelSaveGlobal[] PROGMEM = "Save global par.";
/*  17 */ const char labelSequenceEdit[] PROGMEM = "SEQUENCE EDIT";
/*  18 */ const char labelGroove[] PROGMEM = "Groove";
/*  19 */ const char labelGateMode[] PROGMEM = "Gate duration";
/*  20 */ const char labelLastStep[] PROGMEM = "Last step";
/*  21 */ const char labelCcNum[] PROGMEM = "Ctrl chg number";
/*  22 */ const char labelInitSeq[] PROGMEM = "initialize seq.";
/*  23 */ const char labelSwapSeq[] PROGMEM = "Swap sequences";
/*  24 */ const char labelCopySeq[] PROGMEM = "copy sequence";
/*  25 */ const char labelLoadFactory[] PROGMEM = "Load factory seq";
/*  26 */ const char labelDumpSeq[] PROGMEM = "Dump sequence";
/*  27 */ const char labelLoadSeq[] PROGMEM = "Load sequence";
/*  28 */ const char labelSaveSeq[] PROGMEM = "Save sequence";
/*  29 */ const char labelMaintenance[] PROGMEM = "MAINTENANCE";
/*  30 */ const char labelFactorySettings[] PROGMEM = "Factory settings";
/*  31 */ const char labelTestStepLeds[] PROGMEM = "Test step leds";
/*  32 */ const char labelTestIndLeds[] PROGMEM = "Test ind. leds";
/*  33 */ const char labelTestBeeper[] PROGMEM = "Test Beeper";
/*  34 */ const char labelDisplayVersions[] PROGMEM = "Versions";
/*  35 */ const char labelSubmenu[] PROGMEM = "TESTS";
/*  36 */ const char labelTest1[] PROGMEM = "test 1";
/*  37 */ const char labelTest2[] PROGMEM = "test 2";
/*  38 */ const char labelTest3[] PROGMEM = "test 3";
/*  39 */ const char labelAppVersion[] PROGMEM = "Version number";

const char *const labelsTable[MENU_BROWSER_NB_ENTRIES] PROGMEM =
{
    /*   0 */ labelGlobalEdit,
    /*   1 */ labelChannelIn,
    /*   2 */ labelChannelOut,
    /*   3 */ labelProgramNumber,
    /*   4 */ labelArpeggiator,
    /*   5 */ labelClockIn,
    /*   6 */ labelClockOut,
    /*   7 */ labelKeyClick,
    /*   8 */ labelAudioBeat,
    /*   9 */ labelSysEx,
    /*  10 */ labelTransposition,
    /*  11 */ labelDumpAll,
    /*  12 */ labelLoadAll,
    /*  13 */ labelSaveAll,
    /*  14 */ labelDumpGlobal,
    /*  15 */ labelLoadGlobal,
    /*  16 */ labelSaveGlobal,
    /*  17 */ labelSequenceEdit,
    /*  18 */ labelGroove,
    /*  19 */ labelGateMode,
    /*  20 */ labelLastStep,
    /*  21 */ labelCcNum,
    /*  22 */ labelInitSeq,
    /*  23 */ labelSwapSeq,
    /*  24 */ labelCopySeq,
    /*  25 */ labelLoadFactory,
    /*  26 */ labelDumpSeq,
```

```
    /*  27 */ labelLoadSeq,
    /*  28 */ labelSaveSeq,
    /*  29 */ labelMaintenance,
    /*  30 */ labelFactorySettings,
    /*  31 */ labelTestStepLeds,
    /*  32 */ labelTestIndLeds,
    /*  33 */ labelTestBeeper,
    /*  34 */ labelDisplayVersions,
    /*  35 */ labelSubmenu,
    /*  36 */ labelTest1,
    /*  37 */ labelTest2,
    /*  38 */ labelTest3,
    /*  39 */ labelAppVersion,
};

const PROGMEM MENU_BROWSER_FUNCTION_PTR execFunctionsTable[18] =
{
    /*  11   0 */ &dumpAll,
    /*  12   1 */ &loadAll,
    /*  13   2 */ &saveAll,
    /*  14   3 */ &dumpGlobal,
    /*  15   4 */ &loadGlobal,
    /*  16   5 */ &saveGlobal,
    /*  22   6 */ &initSeq,
    /*  23   7 */ &swapSeq,
    /*  24   8 */ &copySeq,
    /*  25   9 */ &loadFactory,
    /*  26  10 */ &dumpSeq,
    /*  27  11 */ &loadSeq,
    /*  28  12 */ &saveSeq,
    /*  30  13 */ &factorySettings,
    /*  31  14 */ &testStepLeds,
    /*  32  15 */ &testIndLeds,
    /*  33  16 */ &testBeeper,
    /*  34  17 */ &displayVersions,
};

const PROGMEM MENU_BROWSER_EDIT_PTR editFunctionsTable[18] =
{
    /*   1   0 */ &editChannelIn,
    /*   2   1 */ &editChannelOut,
    /*   3   2 */ &editProgramNumber,
    /*   4   3 */ &editArpeggiator,
    /*   5   4 */ &editClockIn,
    /*   6   5 */ &editClockOut,
    /*   7   6 */ &editKeyClick,
    /*   8   7 */ &editAudioBeat,
    /*   9   8 */ &editSysEx,
    /*  10   9 */ &editTransposition,
    /*  18  10 */ &editGroove,
    /*  19  11 */ &editGateMode,
    /*  20  12 */ &editLastStep,
    /*  21  13 */ &editCcNum,
    /*  36  14 */ &editTest1,
    /*  37  15 */ &editTest2,
    /*  38  16 */ &editTest3,
    /*  39  17 */ &editAppVersion,
};

const enum menuOptionType itemTypeTable[40] PROGMEM =
{
```

```
    /* 000 */ menuTypeMenu,
    /* 001 */ menuTypeVariable,
    /* 002 */ menuTypeVariable,
    /* 003 */ menuTypeVariable,
    /* 004 */ menuTypeVariable,
    /* 005 */ menuTypeVariable,
    /* 006 */ menuTypeVariable,
    /* 007 */ menuTypeVariable,
    /* 008 */ menuTypeVariable,
    /* 009 */ menuTypeVariable,
    /* 010 */ menuTypeVariable,
    /* 011 */ menuTypeFunction,
    /* 012 */ menuTypeFunction,
    /* 013 */ menuTypeFunction,
    /* 014 */ menuTypeFunction,
    /* 015 */ menuTypeFunction,
    /* 016 */ menuTypeFunction,
    /* 017 */ menuTypeMenu,
    /* 018 */ menuTypeVariable,
    /* 019 */ menuTypeVariable,
    /* 020 */ menuTypeVariable,
    /* 021 */ menuTypeVariable,
    /* 022 */ menuTypeFunction,
    /* 023 */ menuTypeFunction,
    /* 024 */ menuTypeFunction,
    /* 025 */ menuTypeFunction,
    /* 026 */ menuTypeFunction,
    /* 027 */ menuTypeFunction,
    /* 028 */ menuTypeFunction,
    /* 029 */ menuTypeMenu,
    /* 030 */ menuTypeFunction,
    /* 031 */ menuTypeFunction,
    /* 032 */ menuTypeFunction,
    /* 033 */ menuTypeFunction,
    /* 034 */ menuTypeFunction,
    /* 035 */ menuTypeMenu,
    /* 036 */ menuTypeVariable,
    /* 037 */ menuTypeVariable,
    /* 038 */ menuTypeVariable,
    /* 039 */ menuTypeVariable,
};

#endif
```

Conclusion

Ca y est, on a le code des données de notre menu. On ne sait pas encore quoi en faire, mais on l'a. On se doute bien qu'il va falloir l'insérer quelque part dans les fichiers .c, .cpp ou .h de la future bibliothèque, mais sans plus. Le plus simple est de créer un fichier *menuData.h* qui sera inclus **uniquement** par *menu.cpp* (hé oui, le module *menu* sera un module .cpp).

Il y a dans le générateur une petite partie de code inutilisée jusqu'à présent, c'est la création des squelettes des fonctions. Car ce que le générateur ne peut pas vérifier, c'est l'existence de ces fonctions dans le reste du code. Pour chaque fonction non trouvée, il y a bien sûr une erreur de compilation. Si on appele cette petite partie, on peut démarrer une application à partir de son menu, ceci pour se concentrer évidemment sur le menu, finies les erreurs de compilation, car même si elles sont vides, les fonctions existent. Pour appeler cette petite partie, il suffit de rajouter -c (pour *createEmptyFunction*) à la ligne de commande qui lance ***menuGenerator.py***. Travailler sur une

partie d'une application alors que les autres parties de l'application n'existent pas encore est un souhait très courant chez les développeurs.

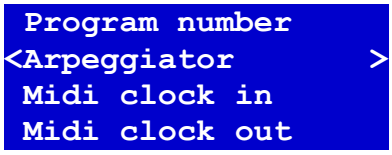
Partie affichage

Cette partie est très spécifique au côté embarqué de la famille Arduino du fait des périphériques d'affichage utilisés, ce sont souvent des afficheurs 7 segments, des lcd en mode texte, rie, a voir donc avec des moniteurs VGA, ni des écrans tactiles etc...

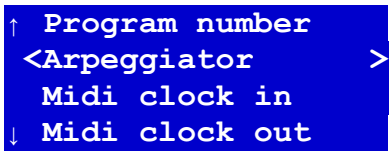
Le module *menuDisplay* utilise un lcd 4 lignes de 20 caractères. Niveau hardware, la communication se fait en I2C sur un mode 8 bits. Si vous voulez autre chose... Il vous faudra écrire votre module en vous inspirant de celui là.

Comment afficher un menu?

La façon la plus simple d'afficher un menu est une liste. On entoure l'objet qui a le focus par une paire de balises ou de crochets. Comme par exemple ici :



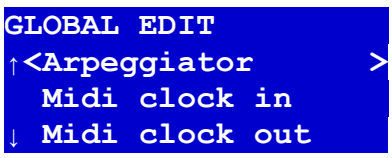
On comprend facilement que l'objet qui a le focus est *Arpeggiator*. Par contre, on ne voit pas qu"il y a d'autres objets devant *Program number* et derrière *Midi clock out*. C'est pourquoi on va redéfinir 2 caractères pour représenter une flèche vers le haut et une flèche vers le bas. Utiliser quelques caractères redéfinis fait d'ailleurs partie des fonctionalités de base des afficheurs lcd de type texte.



On constate que le texte le plus large est celui de l'objet Arpeggiator, à cause de la balise fermante. Si on se donne la peine de compter, on arrive à 20 caractères, ce qui est pile la largeur de notre lcd. On a du coup la première constante de notre module d'affichage, c'est 20 – largeur de < - largeur de > - largeur de ↑ = 17, ce qui va être la taille maximum des labels que l'on peut afficher. D'où cette ligne quelque part dans un fichier .h:

```
#define LABEL_MAX_SIZE 17
```

Il ya encore une autre chose que l'on ne voit pas, c'est la position de notre liste dans l'arborescence. C'est pourquoi on va consacrer la première ligne de l'afficheur au chemin, par exemple comme ceci :



On voit qu'à partir du 2ème niveau d'arborescence, on va avoir un problème de place sur notre première ligne, car 2 x 17 = 34, ce qui est déjà beaucoup plus que ce qui est physiquement faisable. Dans certains menus évolués il y a un bouton "*Were am I?*" qui permet d'afficher momentanément le chemin complet. On va plutôt écrire un algorithme qui prend le début et la fin du chemin en mettant 3 points au millieu.

Force est de constater que l'affichage des menus en majuscules et des variables/functions en normal donne une vision de l'arborescence plus précise, on sait intuitivement qu'il y a d'autres menus derrière les majuscules et une variable/fonction sinon. On peut partir de cette constation pour écrire une première règle de conception des menus, mais ce n'est absolument pas une obligation.

On arrive tout doucement au bout de la gestion de la partie navigation dans les menus, il reste la partie affichage/édition des variables et exécution des fonctions.Comment afficher un menu?

Partie affichage/édition de variable

Je rappelle que cette partie est assurée par une partie externe à *arduinoMenu*. Il y a justement une table générée qui permet l'accès aux fonctions d'affichage édition.

Mais rien n'empêche de lui fournir un écran préformaté, une sorte de template, de sorte qu'elle l'utilise quand elle doit afficher quelque chose. La fonction d'affichage de la variable n'a plus quà poser la valeur de la variable au bon endroit sur l'écran. Cette façon de procéder permet bien sûr d'avoir un moteur de menu beaucoup plus petit. Voici une idée d'écran que l'on pourrait fournir, la dernière ligne servant à afficher la valeur de la variable.

```
GLOBAL EDIT
<Transposition>
Edit value:
-12
```

Partie affichage de fonction

Comment, il faut afficher quelque chose quand on exécute une fonction? Hé oui, si on exécutait la fonction sans aucun retour visuel, on aurait toujours un doute : La fonction a t-elle été exécutée ou pas? Donc proposition d'écran (optionnel) pour avant l'exécution:

```
MAINTENANCE/TESTS
<test 1>
Hit button to exec.:
```

Et après exécution :

```
MAINTENANCE/TESTS
<test 1>

Execution done!
```

Récapitulatif

Notre partie affichage sera donc le module *menuDisplay* capable d'effectuer les tâches

suivantes, invisibles par l'utilisateur :

- refreshBrowserScreen Rafraichissement de l'écran de navigation.
- showEditVariableScreen Afficher l'écran d'édition de variable.
- showVariable Afficher la valeur de la variable au bon endroit de l'écran.
- showPrefunctionScreen Afficher l'écran de prélancement de fonction.
- showPostfunctionScreen Afficher l'écran de postlancement de fonction.

Et bien sûr quelques fonctions pour l'utilisateur qui aura un mode où le menu sera en sommeil et où il pourra profiter de l'écran et du périquérique de saisie.

- clearScreen Efface l'écran.
- setBlink Fait clignoter le curseur. Ou pas.
- gotoXY Positionne le curseur.
- userPrint L'utilisateur affiche ce qu'il veut où il veut.

On peut dire que n'importe quel périphérique de sortie capable d'afficher à un endroit précis d'un écran pourra servir à afficher un menu, il suffit d'implémenter les fonctions listées plus haut.

Si on ne se sert du menu que pour configurer, (très rarement, donc) on peut envigager un terminal PC connecté en USB au lieu d'un ecran LCD... Mais il faut un terminal "amélioré", celui de l'environnement de développement Arduino ne peut ni repositionner le curseur ni effacer l'écran.

Partie commandes

Par commande, on entend :

- Se déplacer dans le menu.
- Sélectionner une variable.
- Editer une variable.
- Sélectionner une fonction.
- Exécuter une fonction ou annuler.

Notre partie commandes sera donc le module *menuinput* capable d'effectuer ces tâches. Dans le cas de la famille Arduino, le moyen le plus courant est un joystick tout ou rien. Les 4 directions suffisent, il n'y a même pas besoin de boutons supplémentaires!

- haut, bas se déplacer, incrémenter/décrémenter une variable.
- droite entrer dans un sous menu, valider une fonction.
- gauche sortir d'un sous-menu, annuler une fonction.

Un autre périphérique courant est l'encodeur incrémental. Il est composé d'un axe rotatif et d'un click.

- Tourner à droite/gauche se déplacer, incrémenter/décrémenter une variable.
- click court (normal) entrer dans un sous menu, valider une fonction.
- click long sortir d'un sous-menu, annuler une fonction.

Il faut néanmoins être absolument sûr de son encodeur, car s'il a des rebonds aléatoires, son

utilisation devient vite un cauchemar, en particulier au niveau des click longs et courts.

Conclusion

Le module *arduinoMenu* est constitué des 4 modules suivants:

- Module menuBrowser Navigation dans les menus.
  - `menuBrowser.cpp`
  - `menuBrowser.h`
  - `menuData.h`
- Module menuDisplay Affichage des différents écrans.
  - `MenuDisplay.hpp`
  - `MenuDisplay.h`
- Module menuInput Saisie des données utilisateur.
  - `menuInput.cpp`
  - `menuInput.h`
- Module arduinoMenu Fait le lien entre l'application et les 3 autres modules.
  - `arduinoMenu.hpp`
  - `arduinoMenu.h`

Ces 4 modules sont embarqués dans la bibliothèque *arduinoMenu*. Il y a aussi le module python ***menuGenerator.py*** qui doit être exécuté en amont de la compilation du micro afin de ne pas saisir à la main toutes les données dont le moteur de menu a besoin.

Chaque module est totalement indépendant grâce au système de *callbacks*, le développeur peut personnaliser ce qu'il veut de façon assez simple. Dans l'état, cette bibliothèque est founie pour:

- Afficheur LCD 20 x 4 text en I2C
- Joystick tout ou rien 4 directions.

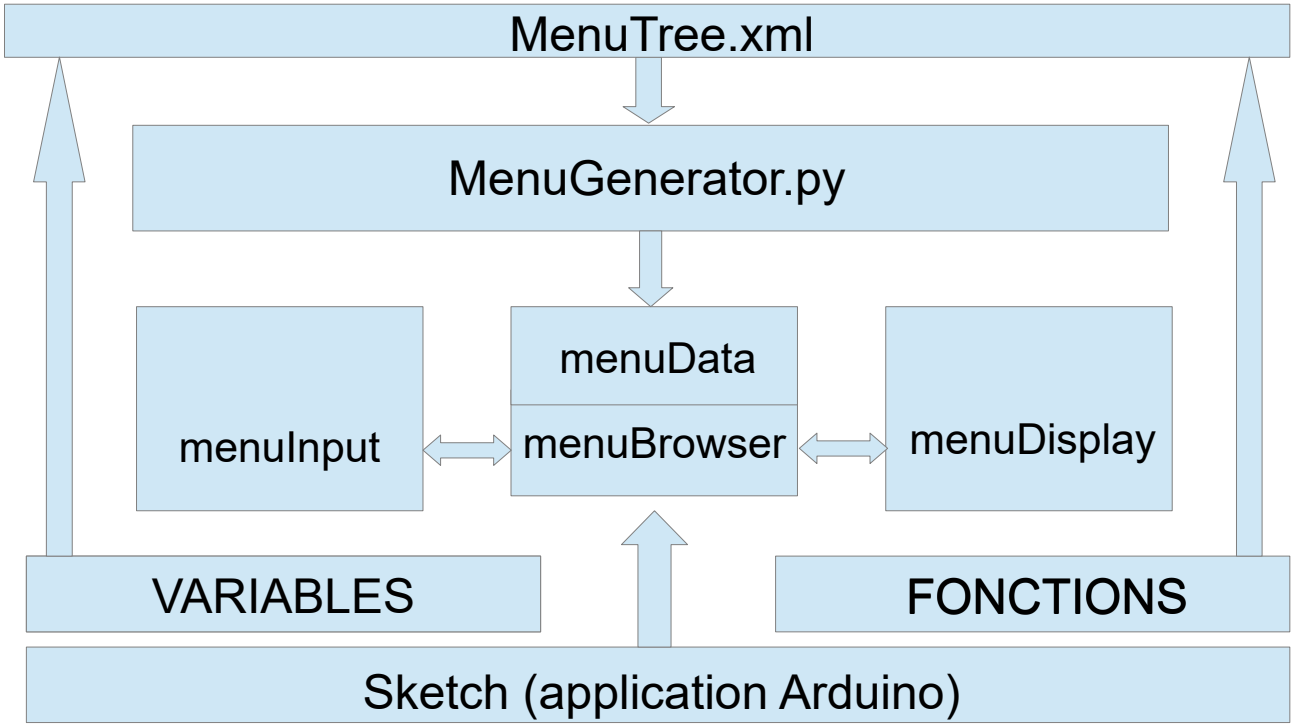
Il y a dans le répertoire *examples* un exemple d'utilisation avec le terminal PC en USB, ça peut servir de base pour se construire un moteur de menu personnalisé. Avantage non négligeable, il fonctionne sans aucun hardware additionnel, ce qui est le rêve pour un développeur logiciel.

Ecriture d'autres périphériques d'entrée et de sortie

Cela ne devrait poser aucun problème. Le principe est de garder le fichier .h de la classe concernée (*menuInput* ou *menuDisplay*) - **ce qui veut dire conserver la signature des méthodes publiques** - et de modifier le code pour s'adapter au spécificités des périphériques.

Organigramme

Un dessin valant mieux qu'un long discours, voici l'organigramme de l'interaction entre l'application (appelée sketch chez Arduino), *arduinoMenu* et *menuGenerator.py*. Le fichier sharedFunctions.h n'est pas représenté car il est anecdotique, c'est une aide au développement mais dans l'absolu l'utilisateur devrait le faire tout seul. D'ailleurs il devrait éditer toutes ces fonctions autogénérées, ne serait ce que pour implémenter un min-max.



Configuration de menuGenerator.py

Il y a un fichier pour ça, il s'appelle *menuGenerator.cfg* et est autodocumenté. Il y a également deux templates servant à construire les fonctions et l'édition/affichage de variables. On peut soit les utiliser soit en écrire d'autres.

Concernant l'écriture des modules sous forme de classes

Je ne suis pas un spécialiste du cpp, ayant surtout écrit du code c pour de petits micros embarqués. J'ai été confronté à un problème innatendu, c'est l'impossibilité de faire un callback sur une méthode de classe.

En effet, si on essaie de faire passer une méthode de classe en callback, on obtient l'erreur suivante:

```
browser->setRefreshBrowserScreen(display->refreshBrowserScreen);
```

```
arduinoMenu.cpp:49:69: error: invalid use of non-static member function
```

Je me suis tiré d'affaire en partant du principe qu'un menu est toujours un singleton dans un petit système embarqué. Pour chaque méthode de classe susceptible d'être appellée en callback, j'ai

créé un pointeur et une méthode statiques (c'est à dire ne faisant pas partie de la classe).

Déclaration dans le fichier .h de la classe display :

```
extern MENU_DISPLAY* MENU_DISPLAY_SINGLETON;
extern void MENU_DISPLAY_refreshBrowserScreen();
```

Dans le fichier cpp de la classe display :

```
MENU_DISPLAY* MENU_DISPLAY_SINGLETON;

void MENU_DISPLAY_refreshBrowserScreen()
{
    MENU_DISPLAY_SINGLETON->refreshBrowserScreen();
}

MENU_DISPLAY::MENU_DISPLAY()
{
    MENU_DISPLAY_SINGLETON = this;
}
```

Il y a sûrement des solutions plus dans l'esprit cpp, mais celle là fonctionne parfaitement. La méthode de classe est encapsulée dans une méthode statique, et du coup c'est elle qui est appellée quand il y a besoin d'un callback :

```
browser->setRefreshBrowserScreen(MENU_DISPLAY_refreshBrowserScreen);
```

Périphériques d'entrée

Le port série (over USB)

Ce port a eu un réel intérêt pendant le développement du projet. Il en a toujours un si le menu ne sert qu'en période de maintenance... Pas besoin d'implémenter un clavier et un écran si ça ne doit servir qu'occasionellement.

Le clavier matriciel

Seules quatres touches sont utilisées, c'est assez redondant avec le joystick. Si vous avez un clavier a au moins 4 colonnes ou 4 lignes, un seul gpio supplémentaire vous permettra de rajouter les 4 flèches directionelles... Il faudra bien sûr revoir les connexions.

L'encodeur incrémental

C'est un moyen simple et élégant de naviguer dans un menu. Probablement un des plus utilisés en embarqué.

Le joystick

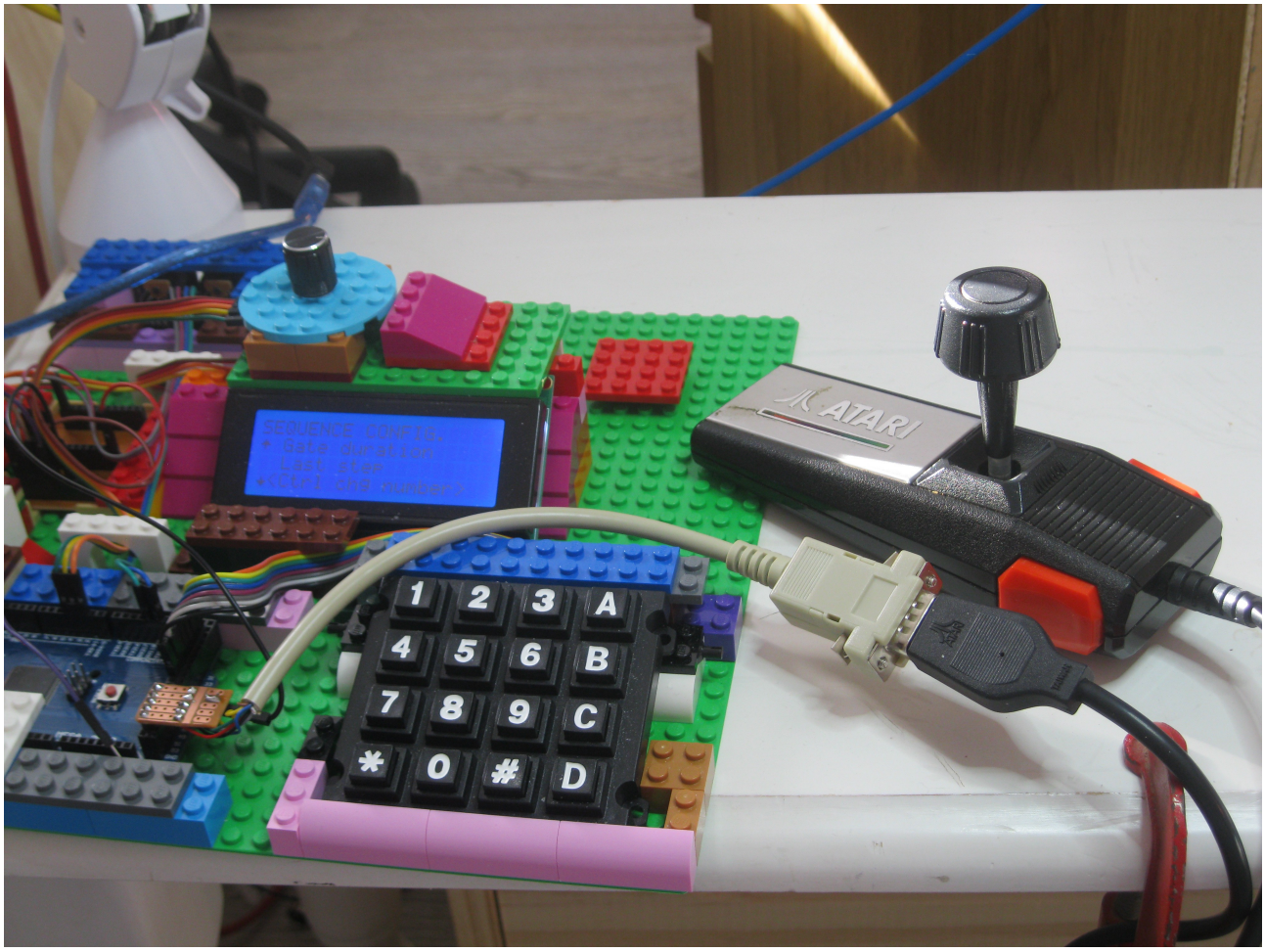
Avant d'aller plus loin, précisons qu'un joystick esr remplaçable par 4 boutons simples (un de plus si on veut un *bouton fire*) , car il a un côté ludique indéniable.

Les joysticks Atari des années 70 se connectaient sur un port SUBD-9, le connecteur mâle sur l'ordinateur et la femelle sur le câble du joystick.

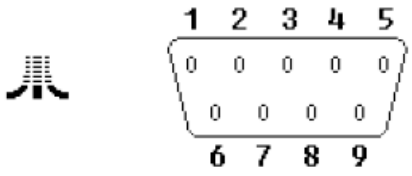
L'Atari 800XL en possédait 2. L'Atari 800 (la génération précédente) en possédait 4. Le brochage est assez simple, on part du principe que les entrées sont en l'air et que la sélection d'une direction ou l'appui sur le bouton FIRE court-circuite la ligne avec le 0 volt. Ne pas oublier de configurer les gpio de l'Arduino en INPUT_PULLUP pour garantir le résultat sur toute la gamme de joysticks existants actuellement. Pour la même raison, ignorez complètement les pins non concernées, ce port servait aussi à connecter une tablette tactile, des raquettes, un track ball... Il faut savoir qu'il y a également un 5 volts sur ce port. J'ai cannibalisé une rallonge SUBD-9 achetée à vil prix chez Pearl pour ajouter la connectique Arduino sur la moitié du câble avec le connecteur mâle. Voici sa référence:

https://www.pearl.be/article/PE807/rallonge-serie-sub-d-9-1-80m

Et voici ce joystick d'anthologie connecté à un Arduino pour naviguer dans le menu .



On aperçoit également la carte Arduino Mega, un codeur incrémental, un LCD 4x20, un clavier matriciel 4x4 et un encodeur incrémental.



1. Joystick Up
2. Joystick Down
3. Joystick Left
4. Joystick Right
5. B input Paddle/Touch Tablet
6. Input Trigger
7. +5V
8. Gnd
9. A input Paddle/Touch Tablet

http://www.mixinc.net/atari/pinouts/joystick.htm

1.	rouge	Up
2.	noir	Down
3.	bleu	Left
4.	gris	Right
5.	marron	-----
6.	vert	Fire
7.	blanc	-----
8.	jaune	GND
9.	vert	-----

Les couleurs sont celles de la rallonge SUBD-9 du câble Pearl.

Le connecteur est celui de l'ordinateur (pas celui du joystick) vu de l'extérieur.