TPs
Circuits Numériques et Microcontrôleurs

Polytech'Paris Sud PeiP2

Introduction

L'objectif de ces TPs est de découvrir un système embarqué et ses fonctionnalités. Nous développerons différentes applications embarquées qui utiliseront les entrées/sorties disponibles de la carte (voir § **Matériel**). Nous programmerons en C et nous utiliserons des bibliothèques et APIs (*Application Programmable Interface*) existantes codées en C++ (programmation objet, voir § **Logiciel**).

• Matériel ARM mbed

La carte NXP LPC1768 représentée sur la figure 1(a), basée sur le micro-contrôleur ARM Cortex M3 Core est utilisée pour le prototypage rapide d'applications embarquées. Elle dispose des éléments suivants : Interfaces USB, Ethernet, CAN, Mémoire Flash, 32Ko RAM, 512KB Flash, un Processeur 96 Mhz. La carte de développement [2] (voir figure 1(b)) dispose de différents capteurs/actionneurs (température, accéléromètre, joystick) et différents périphériques de sortie (afficheur LCD, speaker, leds).

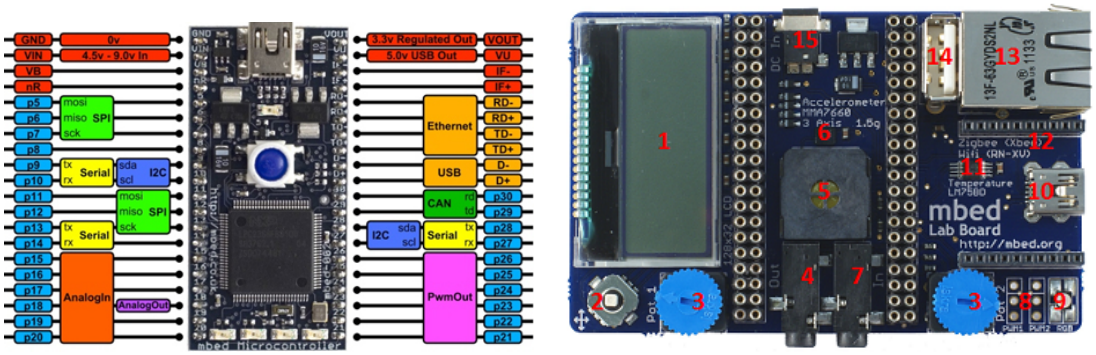
• Logiciel

◦ **Environnement de programmation.** La programmation du mbed se fait dans un environnement de développement en ligne [3] (*Online IDE* pour *Integrated Development Environment*) visible sur la figure 2.

On suivra les étapes suivantes :

— Créer un compte sur le site mbed [4].

1. `https://ecampus.paris-saclay.fr/`
2. Carte de développement : `http://developer.mbed.org/components/mbed-Application-Board/`
3. Online IDE :`https://studio.keil.arm.com/`
4. Site mbed : `https://os.mbed.com/`



(a) (b)

FIGURE 1 – (a) Carte mbed et ses broches d'E/S. (b) Carte d'application utilisée en TP. Nous utiliserons l'écran LCD (1), le joystick (2), les potentiomètres (3), l'accéléromètre (6), la led couleur (9), le thermomètre (11), la communication ZigBee (12).

— Dans le compilateur en ligne [5], créer un nouveau projet et compléter le main associé. On créera un projet par exercice. Il est possible d'importer des bibliothèques ou des programmes existants [6].
— Sélectionner la plateforme utilisée (*Target hardware*) : mbed LPC1768.
— Compiler votre programme.
— Connecter la carte mbed au port USB. La carte est alors détectée comme une simple clé USB sur laquelle on déposera le fichier .bin à exécuter.
— Appuyer sur le bouton Reset de la carte. Le code le plus récent est alors exécuté.

◦ **Le SDK (Software Development Kit).** Le kit de développement logiciel mbed (SDK) est une plate-forme codée en C/C++ par des dizaines de milliers de développeurs pour construire des projets rapides. Il fournit un ensemble de bibliothèques et d'APIs pour accéder aux différents périphériques. Les périphériques d'entrées/sorties sont définis comme des **objets**. Un objet s'apparente à une structure de données accessible par le biais de messages, appelés **méthodes**. Les données décrivent la structure interne de l'objet et sont appelées les **attributs**. Les attributs et les méthodes associés aux objets sont définis par des modules logiciels appelés **classes**. Un code peut déclarer plusieurs objets d'une même classe, chaque objet est alors une **instance** de la classe. La figure 3 montre un exemple dans le cas du mbed.

1 Affichage d'un entier sur les leds

Nous allons utiliser les 4 leds de la carte mbed pour afficher en boucle les représentations binaires de 0 à 15.

1. Déclarer les 4 leds comme des objets `DigitalOut` puis créer un tableau associé aux 4 leds.
2. Créer une fonction de mise à 0 des 4 leds `void ResetLeds()`

5. Compilateur :`https://studio.keil.arm.com/`
6. Importation de code : voir `http://developer.mbed.org/handbook/Homepage`

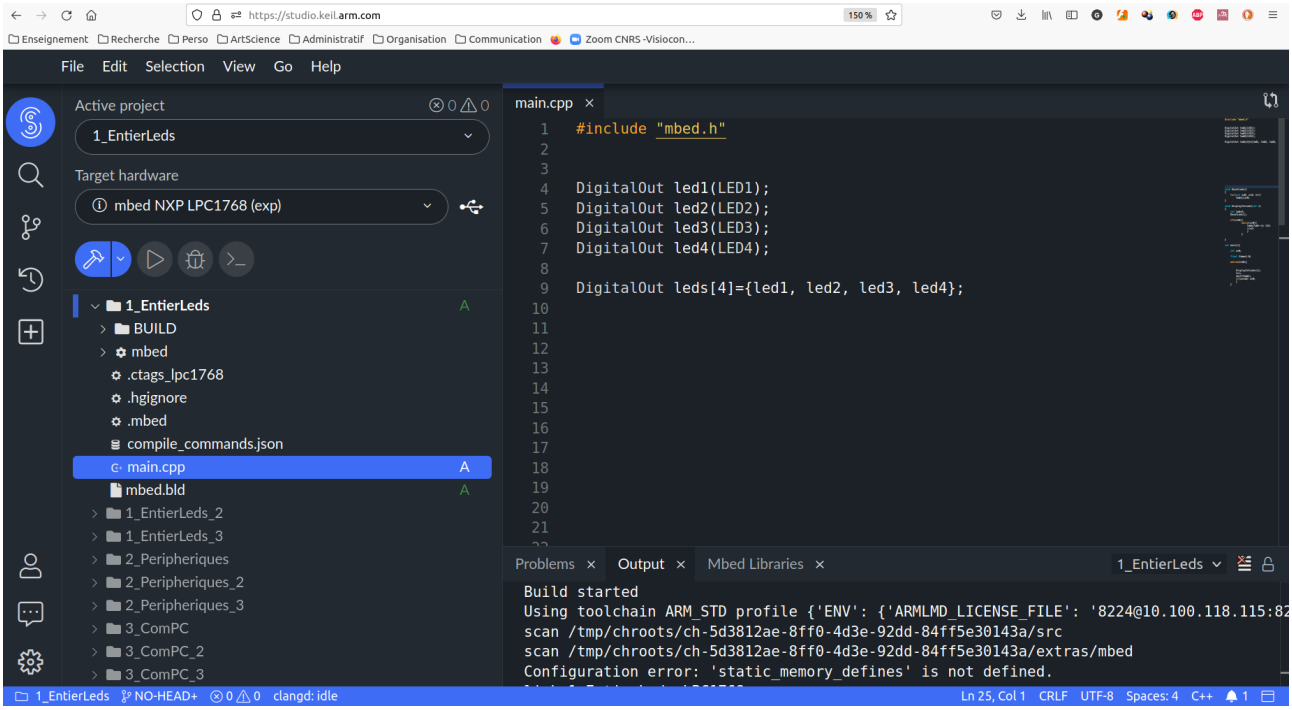


FIGURE 2 – Environnement de programmation Keil.

3. Créer une fonction d'allumage des leds `void DisplayIntLeds(int i)` qui prend en entrée un entier `i` (compris entre 0 et 15) et allume les leds de façon à afficher la représentation binaire de cet entier. Par exemple l'entier 0 n'allume aucune led, 15 allume les 4 leds.
4. Écrire le programme principal pour afficher les représentations binaires de 0 à 15 en boucle.
5. Compiler et tester le programme (voir dans § **Logiciel** comment charger le code sur la carte mbed).
6. Comparer le comportement du programme lorsque l'on utilise la fonction `wait( float t)` et lorsqu'on ne l'utilise pas.

2 Lecture des capteurs/actionneurs et affichage

(On recopiera l'exo 1 pour réutiliser les fonctions). L'objectif de cet exercice est de lire les valeurs des potentiomètres et du thermomètre, de les afficher sur l'écran LCD et de représenter visuellement les valeurs des potentiomètres sur les leds (leds bleues et leds couleur).

1. Importer les bibliothèques associées au thermomètre [7] et à l'afficheur [8]. Ajouter dans le code les directives `#include "LM75B.h"` et `#include C12832.h`.

7. Thermomètre : `https://os.mbed.com/users/chris/code/LM75B/`
8. Afficheur : `https://os.mbed.com/users/chris/code/C12832/`

```c
#include "mbed.h"

AnalogIn pot(p19);          /* Potentiomètre défini comme un objet de classe AnalogIn  */
DigitalOut led(LED1);       /* LED : objet DigitalOut  */

int main()
{
    float ain;              /* Variable pour sauvegarde de la valeur de potentiomètre*/

    while(1) {
        ain = pot.read();   /* Lecture de la valeur du potentiomètre (entre 0 et 1).
                               read() est une méthode associée à la classe AnalogIn   */
        led = 1;            /* Allumage led        */
        wait(ain);          /* Attente en seconde (blocant) */
        led = 0;            /* Extinction led      */
        wait(ain);
    }
}
```

FIGURE 3 – Exemple de code. `p19` correspond au n° de la broche (`p` pour *pin* en anglais) associée au potentiomètre 1. Pour connaître les n° des broches, consulter le tableau fourni sous la carte de développement.

2. Identifier les entrées associées aux 2 potentiomètres et au thermomètre. Déclarer dans le programme un objet `AnalogIn` par potentiomètre et un objet LM75B pour le thermomètre.
3. Nous souhaitons afficher les 3 valeurs lues sur l'afficheur LCD.
   (a) Consulter le fichier `C12832.h` pour prendre connaissance des APIs disponibles et déclarer un objet C12832 pour le LCD.
   (b) Afficher sur 3 lignes différentes les 3 valeurs `po1`, `po2`, `t`.
4. Nous allons à présent visualiser la valeur `po1` à l'aide des leds : plus la valeur du potentiomètre sera élevée (entre 0 et 1) et plus il y aura de leds consécutives allumées (entre 0 et 4) comme le montre le tableau ci-dessous. Pour cela nous allons convertir une valeur flottante `v` comprise entre 0 et 1 vers une valeur entière qui pourra être affichée telle qu'elle par la fonction `DisplayIntLeds`.

Valeur flottante	Nombre de leds allumées
$0 \leq ... < 0.2$	0
$0.2 \leq ... < 0.4$	1
$0.4 \leq ... < 0.6$	2
$0.6 \leq ... < 0.8$	3
$0.8 \leq ... \leq 1$	4

   (a) Quelles sont les 5 valeurs entières qu'il faudra afficher pour allumer respectivement 0,1,...4 leds ? Les sauvegarder dans un tableau, par exemple `int patterns[5]`.
   (b) Dans une fonction `int AnalogIn2Index(float v, int n)`, mettre en place le passage de la valeur flottante $\in [0, 0...1, 0]$ vers un entier compris entre 0 et `n-1` $\in [0; 4]$ et retourner cet entier. Il servira d'indice `i` pour indexer le tableau `patterns` (il

suffira alors d'afficher sur les leds la valeur `patterns[i]`). C'est ce qu'on appelle une *Look Up Table* ou LUT). Il y a au moins deux solutions :
— une solution intuitive avec des `if` ;
— une solution plus optimale (au moins en termes de lignes de code) : $i = \lfloor v \times n \rfloor$ (valeur entière de $v \times n$).

(c) Écrire le programme principal et tester.

5. La valeur du potentiomètre 2 va être associée à la led couleur. Plus la valeur sera élevée (resp. faible) et plus la couleur sera chaude (resp. froide). La difficulté ici est de passer d'une valeur flottante à un triplet RGB. Une autre nouveauté est l'utilisation du PWM.

(a) La figure 4 montre les fonctions permettant de passer de `po2` à R, G et B. Donner les équations de chacune des courbes.

(b) Déclarer les valeurs RGB comme 3 sorties PWM et regarder quelles sont les APIs associées [9].

(c) Écrire les fonctions `void R(float v)`, `void G(float v)` et `void B(float v)` permettant de passer d'une valeur flottante (dans notre cas la valeur du potentiomètre) aux valeurs à affecter aux sorties RGB. **Attention : avant d'affecter le résultat à la broche, il faudra inverser le résultat car la led fonctionne en logique inversée.**

(d) Écrire le programme principal et tester. Pour vérifier le code, nous conseillons de tester individuellement `void R(float v)`, `void G(float v)` puis `void B(float v)` en mettant les 2 autres sorties à 1, avant de tester les 3 ensemble.

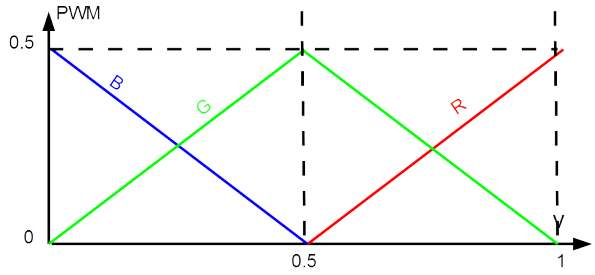


FIGURE 4 – Passage d'une valeur flottante aux valeurs RGB.

3 Le piano. Communication carte ↔ PC

Nous allons utiliser le *Speaker* de la carte pour jouer de la musique au clavier. Le volume sera contrôlé par le potentiomètre 1, et les notes seront associées à différentes touches du clavier du PC, selon 2 modes de fonctionnement :
— Touches de 1 jusqu'à + : 12 notes. L'octave (voir la figure 5) sera donnée par le potentiomètre 2. Un ensemble de 12 fréquences à l'octave 3 est donné dans `exo3.txt`.

9. `PwmOut` : `https://os.mbed.com/handbook/PwmOut`

— Touches associées aux 26 lettres de l'alphabet (dans l'ordre azerty...) : 12 notes dans 2 gammes différentes + 3 notes dans une troisième gamme.

Les caractères seront transmis via la liaison USB à l'aide d'un terminal [10]. Si vous utilisez `Terminal.exe`, il faudra sélectionner (en haut à gauche) le port COM associé au mbed puis s'y connecter. Ensuite, les caractères seront rentrés dans la fenêtre située tout en bas.

Ensuite, lorsque l'on appuiera sur "entrée" (CR ou *carriage return*) : le mbed enverra la valeur des potentiomètres vers le PC. Cette valeur sera alors visible sur le terminal du PC.

NOTES	octave	-1	0	1	2	3	4	5	6	7	8	9
Do	C	16.3 Hz	32.7 Hz	65 Hz	131 Hz	262 Hz	523 Hz	1 046.5 Hz	2 093 Hz	4 186 Hz	8 372 Hz	16 744 Hz
Do diese ou Re bemol	C# / Db	17.3 Hz	34.6 Hz	69 Hz	139 Hz	277 Hz	554 Hz	1 109 Hz	2 217 Hz	4 435 Hz	8 870 Hz	17 740 Hz
Re	D	18.3 Hz	36.7 Hz	74 Hz	147 Hz	294 Hz	587 Hz	1 175 Hz	2 349 Hz	4 698 Hz	9 396 Hz	18 792 Hz
Re diese ou Mi bemol	D# / Eb	19.4 Hz	38.9 Hz	78 Hz	156 Hz	311 Hz	622 Hz	1 244.5 Hz	2 489 Hz	4 978 Hz	9 956 Hz	19 912 Hz
Mi	E	20.5 Hz	41.2 Hz	83 Hz	165 Hz	330 Hz	659 Hz	1 318.5 Hz	2 637 Hz	5 274 Hz	10 548 Hz	21 098 Hz
Fa	F	21.8 Hz	43.6 Hz	87 Hz	175 Hz	349 Hz	698.5 Hz	1 397 Hz	2 794 Hz	5 588 Hz	11 176 Hz	
Fa diese ou Sol bemol	F# / Gb	23.1 Hz	46.2 Hz	92.5 Hz	185 Hz	370 Hz	740 Hz	1 480 Hz	2 960 Hz	5 920 Hz	11 840 Hz	
Sol	G	24.5 Hz	49.0 Hz	98 Hz	196 Hz	392 Hz	784 Hz	1 568 Hz	3 136 Hz	6 272 Hz	12 544 Hz	
Sol diese ou La bemol	G# /Bb	26.0 Hz	51.9 Hz	104 Hz	208 Hz	415 Hz	831 Hz	1 661 Hz	3 322 Hz	6 645 Hz	13 290 Hz	
La	A	27.5 Hz	55.0 Hz	110 Hz	220 Hz	440 Hz	880 Hz	1 760 Hz	3 520 Hz	7 040 Hz	14 080 Hz	
La diese ou Si bemol	A# / Bb	29.1 Hz	58.0 Hz	117 Hz	233 Hz	466 Hz	932 Hz	1 865 Hz	3 729 Hz	7 458 Hz	14 918 Hz	
Si	B	30.8 Hz	62.0 Hz	123 Hz	247 Hz	494 Hz	988 Hz	1 975 Hz	3 951 Hz	7 902 Hz	15 804 Hz	

FIGURE 5 – Fréquences des notes pour différentes octaves.

3.1 Speaker

La carte dispose d'un speaker sur la broche 26. Dans un premier temps, nous allons le tester.

1. Récupérer le code fourni dans `exo3.txt`.
2. Déclarer le speaker comme une sortie contrôlée par PWM.
3. Écrire une fonction `void PlayNote(float frequency, float volume)` qui modifie la période (1/fréquence) et le volume du speaker.
4. Compléter le programme principal pour faire un premier test de cette fonction, par exemple jouer la note Do avec un volume 0.9.

3.2 Potentiomètres

Intéressons-nous aux fonctions associées aux potentiomètres.

1. Compléter le programme principal pour que le volume de la note jouée soit maintenant fourni par le potentiomètre 1. Tester.

10. Terminals : `https://developer.mbed.org/handbook/Terminals`

2. Reprendre la fonction `int AnalogIn2Index(float v, int n)` de l'exo 2 pour passer de la valeur fournie par le potentiomètre 2 à un indice allant de `0` à `n-1` qui permettra de choisir une gamme parmi les `n`.
3. Afficher les valeurs de volume, octave et note sur le LCD. Pour la note, il sera plus simple d'afficher simplement un chiffre de 1à 7 pour indiquer une note entre DO et SI.

3.3 Lecture des caractères envoyés par le PC

Le mbed va lire les caractères envoyés par le PC.

1. Consulter l'exemple de communication série [11] pour mettre en place la communication.
2. Détecter chacun des 7 chiffres et jouer la note correspondante.
3. Détecter chacune des lettres et jouer la note correspondante.

3.4 Énvoi vers le PC

1. Modifier le programme pour que, lorsque le caractère "entrée" est détecté, les valeurs `volume`, `octave` et `notes` soient envoyées vers le PC et lisibles sur le terminal.
2. Tester l'ensemble de l'application.

4 Chenillard contrôlé par joystick

Nous allons mettre en place un chenillard sur les 4 leds bleues. Le joystick permettra de contrôler ce chenillard, comme indiqué dans le tableau suivant. Le tableau vous donne également les numéros de broches.

Joystick	`DigitalIn`	Fonction
$\rightarrow$	`R(p16)`	Décalage à droite
$\leftarrow$	`L(p13)`	Décalage à gauche
$\uparrow$	`U(p15)`	Allumage d'une led à la fin du chenillard
$\downarrow$	`D(p12)`	Extinction de la dernière led du chenillard
.	`X(p14)`	Pause du chenillard puis extinction des 4 leds lorsque l'appui dure plus de 3 s

La figure 6 illustre le fonctionnement attendu, dans le cas d'un décalage à gauche.

4.1 Utilisation du joystick

1. Recopier le squelette de programme fourni dans `exo5.txt`. Les 5 entrées `DigitalIn` associées aux 5 signaux du joysticks sont définies.

11. Classe `Serial` : `https://os.mbed.com/handbook/Serial`

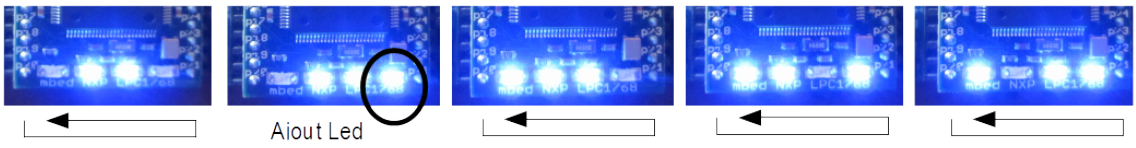


FIGURE 6 – Exemple de décalage à gauche.

2. Pour tester le joystick, on affichera sur le LCD le nom des signaux activés sur le joystick (`D`, `U`, `L`, `R`, `X`).
3. Dans la suite de l'exo, il faudra constamment mémoriser le sens courant de défilement du chenillard, qui sera forcément droite ou gauche, et ce même si l'on appuie sur `U`, `D` ou `X`. Modifier le code pour afficher à tout instant le sens courant du chenillard, et non plus seulement le signal activé.

4.2 Décalages

(Il n'est pas obligatoire de suivre les étapes ci-dessous si vous avez une autre solution)

La fonction `void DisplayIntLeds(int val)` écrite dans l'exercice 1 affiche sur les leds la représentation binaire de la variable `val` ($\in [0...15]$) déclarée en variable globale. Le défilement du chenillard va consister à modifier la valeur `val` soit pour faire une rotation des 4 bits (à gauche ou à droite) soit pour rajouter des bits à 1 ou à 0.

1. Au début du programme principal, on mettra `val` à 1 et on appellera `void DisplayIntLeds(int val)` pour initialiser le chenillard avec 1 led allumée.
2. Écrire une procédure `void ShiftRight()` qui décale d'un bit vers le LSB (*Less Signifant Bit* situé à droite) la représentation binaire de `val`. Lorsque le LSB disparaît à droite, il doit réapparaître à gauche sur le MSB (*Most Signifant Bit*).
3. Écrire `void LedOnRight()` qui modifie `val` pour allumer une led à la fin de chenillard dans le cas d'un décalage à droite. Associer le décalage au signal `U` du joystick. Tester.
4. Écrire `void LedOffRight()` qui modifie `val` pour éteindre la dernière led du chenillard dans le cas d'un décalage à droite. Associer le décalage au signal `D` du joystick. Tester.

À présent, vous avec un chenillard qui fonctionne, mais uniquement dans un sens. Nous allons rajouter une commande permettant de choisir le sens de défilement.

1. Modifier le programme afin que :
   — lorsque l'on appuie sur le `L` du joystick, le chenillard se mette dans le mode « décalage à gauche »
   — lorsque l'on appuie sur le `R` du joystick, il se mette en mode « décalage à droite » .
2. De la même façon que pour le décalage à droite, écrire `void ShiftLeft()`, `void LedOnLeft()` et `void LedOffLeft()`, modifier le programme principal et tester.
3. Modifier le programme principal pour que la valeur `val` soit figée tant que l'on appuie sur `X`. On créera une fonction `void PauseStop()`. Celle-ci va être complétée dans la suite de l'exercice.

4.3 Mesure de temps

Pour l'instant, le chenillard est figé lorsque l'on appuie sur `X`. Nous allons maintenant compléter la fonction `void PauseStop()` pour que les leds s'éteignent lorsque l'appui dépasse 2s d'appui. Nous avons eu l'occasion d'utiliser le timer par l'API `wait(float t)` qui effectue une attente bloquante pendant un temps t déterminé. Nous allons à présent l'utiliser pour mesurer une durée.

1. Prendre connaissance des APIs disponibles dans la classe `Timer` sur `https://os.mbed.com/handbook/Timer`.
2. Modifier le programme principal pour :
   — mesurer le temps d'appui sur `X` ;
   — figer le chenillard tant que le temps <3s ;
   — éteindre les leds lorsque le temps >3s.

5 Le jeu de billes avec accéléromètre

L'objectif de ce est de réaliser un jeu de bille contrôlé par accéléromètre. Via le LCD, on simulera une bille qui roule sur un plateau fermé, les bords du plateau étant les limites de l'écran LCD. L'accéléromètre permettra de modifier l'orientation du plateau et ainsi faire rouler la bille plus ou moins vite en fonction de l'angle d'inclinaison. L'objectif sera d'amener la bille à une cible, dont la position changera à chaque nouvelle partie, de manière aléatoire. À chaque fois que la cible est atteinte, on marque des points et la cible se déplace au *niveau* suivant. Chaque partie est chronométrée. Plus on avance dans le jeu, plus le niveau augmente c'est-à-dire qu'il faut être de plus en plus rapide.

Un accéléromètre est un capteur de mesure de l'accélération, qui peut être due à la gravité ('g' accélération statique 9.8 m/s^2) ou au mouvement. En mesurant l'accélération statique, on peut en déduire les angles et la direction de l'inclinaison de la carte. Le capteur utilisé ici est un Freescale MMA7660FC à 3 axes (voir bibliothèques : `https://os.mbed.com/components/MMA7660/`. On peut lire 3 valeurs flottantes XYZ qui correspondent aux inclinaisons suivant les 3 axes, comme le montre la figure 7.

5.1 Tests

Cette partie visent à déterminer et vérifier les valeurs de X,Y,Z fournies par l'accéléromètre.

1 Écrire une fonction `ReadXYZ()` qui lit les 3 valeurs d'inclinaison fournies par l'accéléromètre.

2 Écrire une fonction `DisplayXYZ()` qui affiche sur l'écran LCD les 3 inclinaisons avec une précision de 2 chiffres après la virgule.

3 Tester le programme de lecture et d'affichage des inclinaisons. Vous pourrez constater que le capteur est très sensible et que les valeurs qu'il fournit sont instables.

4 Compléter `ReadXYZ()` pour introduire un lissage des valeurs. Pour cela, les nouvelles valeurs X,Y,Z lissées seront calculées comme la moyenne de `N` valeurs brutes successives du capteur. Cela a pour conséquence de ralentir la fréquence de mise à jour des valeurs

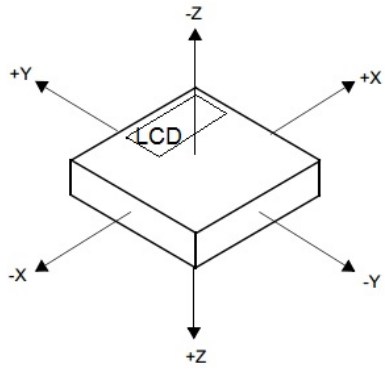


FIGURE 7 – Les 3 axes de l'accéléromètre.

XYZ, mais le processeur est suffisamment rapide pour que ça n'ait pas de conséquence sur notre application. Dans un premier temps, on prendra `N`=100.

5 Écrire une fonction `ReadN()` qui va lire la valeur fournie par le potentiomètre 1 pour modifier le paramètre `N` entre 0 et 500.

5.2 Déplacement de la bille

Déplacer la bille en fonction des valeurs de l'accéléromètre. Celle-ci peut se déplacer dans toutes les directions. La vitesse de la bille dépend de l'amplitude de l'inclinaison de la carte. La bille peut rebondir sur n'importe quel bord et avec n'importe quel angle. Vous êtes libre de la stratégie utilisée.

5.3 Jeu

1. Ajouter une cible circulaire de taille un peu plus grande que la bille.
2. Mettre en place le marquage des points. La cible est atteinte lorsque le centre de la bille est suffisamment proche du centre de la cible.
3. Mettre en place une temporisation. Si la cible n'est pas atteinte dans le temps imparti (1 min par exemple), on affichera *Game over*. Si la cible est atteinte, le temps de jeu alloué sera légèrement réduit au niveau suivant (de 10s par exemple).
4. Mettre en place le comptage des points.
5. Modifier la position de la cible aléatoirement à chaque nouveau jeu (niveau).

6 Pour ceux qui ont terminé

Vous avez le choix entre trois projets de différentes difficultés : le snake ou le Tétris. S'il vous reste 1 séance ou moins, optez pour le jeu de billes. S'il vous reste 2 séances, vous pouvez choisir le Snake ou le Tétris. Pour ceux qui avancent très rapidement, vous pouvez faire plusieurs projets.

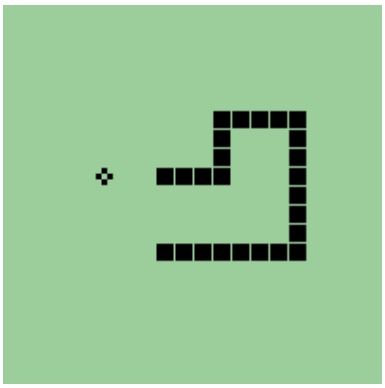


FIGURE 8 – Le fameux jeu snake, popularisé par Nokia sur téléphone portable.

6.1 Snake

À l'aide du joystick, le jouer contrôlera les déplacements d'un serpent (affiché sur le LCD). Le serpent se déplace en continu et devra manger des pastilles pour grandir. Le joueur ne peut indiquer que la direction à suivre (en haut, en bas, à gauche, à droite) afin d'éviter que la tête du serpent ne touche son propre corps, auquel cas il risque de mourir. Un exemple est donné sur la figure 8. Plusieurs variantes du jeu sont possibles :

— la vitesse du serpent peut augmenter au fil d'une partie, ou selon différents niveaux de difficulté.
— des obstacles peuvent être ajoutés

6.2 Tétris

L'objectif est de réaliser un jeu Tetris sur carte Application Board en utilisant les entrées sorties numériques et analogiques, les interruptions, les timers et l'affichage LCD.

6.2.1 Commande du jeu et organisation générale

Le jeu se commandera en utilisant le joystick gauche/droite pour déplacer l'objet, haut pour le faire pivoter l'objet, et le potentiomètre pour accélérer ou ralentir de jeu. Le jeu Tétris consiste à essayer d'assembler des objets constitués de 4 cases et descendant dans un écran pour former des lignes entières. Une ligne entière disparaît de l'écran.

6.2.2 Animation

L'écran LCD a une résolution de 32×128 pixels. Nous garderons 12 pixels en haut et 20 pixels en bas pour des utilisations ultérieures. L'écran de jeu sera donc de 32×96 pixels, chaque case sera formée de 4×4 pixels, le jeu comportera 8×24 cases. La partie en bas de l'écran servira à afficher le score.

1. Avec la méthode `fillrect` de la bibliothèque `C12832`, dessinez un objet barre (4×1 cases) descendant le long de l'écran et s'arrêtant une fois le bout du tableau atteint (pixel 96).

Pour optimiser l'affichage, vous utiliserez `fillrect` pour effacer l'ancienne barre et fillrect pour dessiner la nouvelle barre.

2. La barre descendra toutes les secondes (wait) dans un premier temps. Dans un second temps, la valeur d'attente entre chaque descente sera affectée d'un coefficient donné par la valeur d'un potentiomètre (`pot2`).
3. Par interruption, l'appui sur les touches gauche et droite du joystick permettra de déplacer la barre à l'écran (vous testerez que la barre ne sort pas de l'écran quand elle bouge).
4. L'appui sur la touche haut permettra de tourner l'objet. Il faudra, pour tourner la barre, que l'espace soit disponible pour effectuer l'opération (si, quand elle tourne, la barre sort de l'écran, il faudra soit refuser le mouvement soit déplacer la barre).
5. Une fois que la barre arrive en bas de l'écran, la barre est placée dans le tableau et une nouvelle barre apparaît en haut de l'écran. Le tableau est affiché à l'écran à ce moment.
6. Ajoutez un test, à chaque fois que la barre descend, qui vérifie que la future place est libre. Si elle n'est pas disponible, la barre est placée dans le tableau d'obstacles et ce tableau est à nouveau affiché.
7. Réalisez un test, lorsque la barre est copiée dans le tableau, testant à quelle hauteur dans l'écran se trouvait la barre. Si cette hauteur est nulle ou faible, le jeu se termine et affiche "perdu".
8. Effectuez un test qui, lorsque la barre touche un obstacle, teste si cela forme une ligne entière. Si c'est le cas, cette ligne est éliminée (toutes les lignes au-dessus sont décalées vers le bas).
9. Affichez le score en bas de l'écran. Ce score sera calculé en fonction de nombre de lignes éliminées et de la vitesse du jeu
10. Mettez en œuvre un timer qui, toutes les X secondes, diminuera le coefficient d'attente de descente de barre (initialement fixé à 1s).