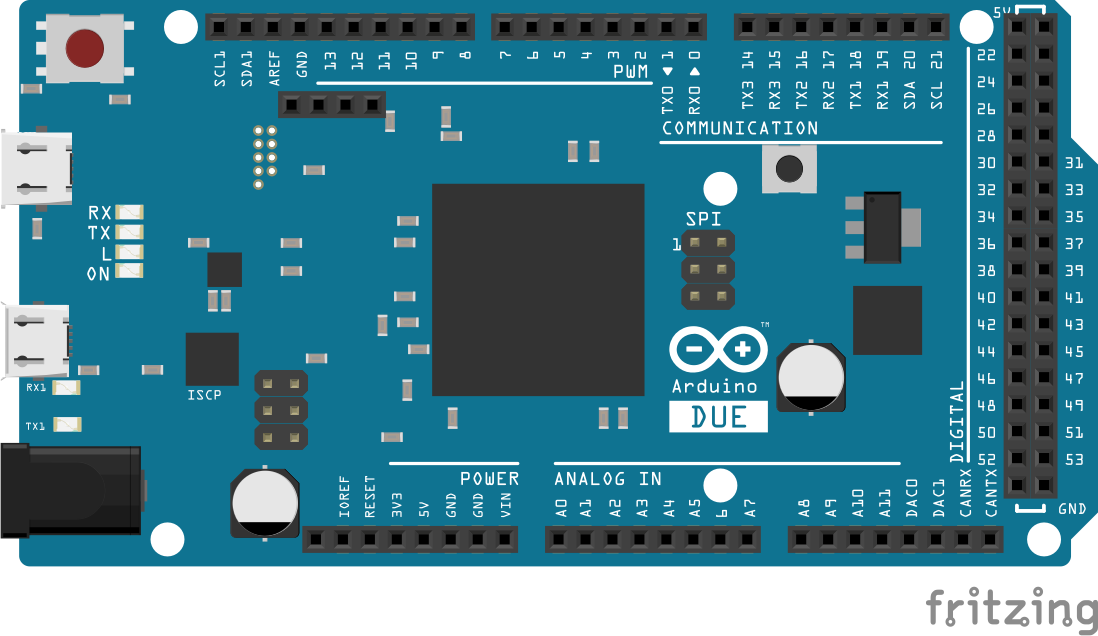


2017-2018

Rapport de TIPE

NAHMIAZ Thomas REBECQ Victor



Est-il possible de faire un jeu sous Arduino ?

PTSI 1

Table des matières

[Introduction : 2](#_Toc514431627)

[I. Mise en place : 2](#_Toc514431628)

[II. Conception : 5](#_Toc514431629)

[Création des modules : 5](#_Toc514431630)

[Gérer plusieurs modules en même temps : 10](#_Toc514431631)

[Outils informatiques utilisés : 11](#_Toc514431632)

[Faire un jeu réutilisable : 12](#_Toc514431633)

[Solutions : 12](#_Toc514431634)

[Mise en œuvre : 12](#_Toc514431635)

[III. Exploitation 12](#_Toc514431636)

Résumé :

Le projet proposé est un jeu physique sur Arduino, une bombe à désamorcer par équipe de deux joueurs. Il s’agit de deux énigmes à résoudre avant un temps imparti, un joueur se trouve face à la bombe, l’autre devant un manuel donnant la solution des énigmes. Les deux joueurs devront coopérer en se communiquant des informations afin de désamorcer la bombe.

Résultats ?

# Introduction :

L’objet de ce TIPE sera de réaliser un jeu physique sur arduino. Nous avons choisi le thème d’une bombe à désamorcer par équipe de deux joueurs.

Nous aborderons tout d’abord la mise en place du projet puis sa conception et enfin les résultats obtenus.

Plan : cahier des charges, planning … / exigences du cahier des charges / expériences etc.

# Mise en place :

1. Naissance du projet

Dès fin septembre, nous avions trouvé notre projet de TIPE : nous voulions tous les deux faire un jeu, jouable physiquement (avec des boutons, LEDs, ..).  
Après réflexion, et après avoir joué à la version ordinateur de [**Keep Talking and Nobody Explodes**](http://www.keeptalkinggame.com/), nous avons été convaincus que c’est ce jeu que nous voulions adapter physiquement.

Le principe du jeu est simple : 2 joueurs sont en équipe, et doivent désamorcer une bombe, composée d’énigmes. Un joueur voit la bombe, peut interagir avec elle mais ne connait pas la solution aux énigmes (appelés « Modules » dans la suite). L’autre joueur possède le manuel avec la procédure permettant de désamorcer chaque module, mais ne peut ni voir, ni toucher la bombe.  
C’est ce mélange de jeu coopératif, mêlant tensions et incompréhension, qui nous a montré le potentiel que pouvait avoir ce concept.

1. Première maquette

Il fallait désormais prouver que ce concept était viable et réalisable.  
Après une première explication orale, nous nous sommes heurtés à un avis plutôt mitigé. En effet, le concept est peut-être facile à comprendre mais il était dur de voir si cette idée pouvait devenir un « jeu », et non rester une banale résolution d’énigme à 2.

Nous avons donc réalisé une maquette en carton de la bombe. Nous avons conçu 4 premiers modules (respectivement le module « Bouton », « Morse », « Accéléromètre » et « Films »). Les 4 premières ébauches de manuels sont en annexes.  
Un chemin précis (et unique, le jeu était difficilement rejouable) avait été déterminé. Pendant que l’un de nous deux faisait tourner les disques en carton attachés avec des attaches parisiennes, l’autre vérifiait que les modules étaient correctement réalisés, indiquait le temps restant, comptait les erreurs, ... C’est la première prise de conscience que le point important de ce sujet allait être de pouvoir rendre notre programme « multi-tâche ».

Avec cette maquette, nous avons effectué une dizaine d’expérimentations et tiré plusieurs enseignements :

* Premièrement, le jeu n’avait un intérêt que s’il y avait une certaine pression, ajoutée par la limite de temps. Sans elle, le jeu aurait été dénué d’intérêt.
* Ensuite, malgré les défauts apparents d’un tel système, ainsi que la grossièreté de la maquette, chaque groupe de personne testé (avec une limite de temps donc) eu un certain plaisir, une certaine envie de recommencer et de réussir. Cette maquette a d’ailleurs plutôt convaincu M.Delegue (qui faisait équipe avec M. Prachay).

Alors, confiants sur l’avenir du projet, nous nous sommes lancés dans la réalisation du programme.

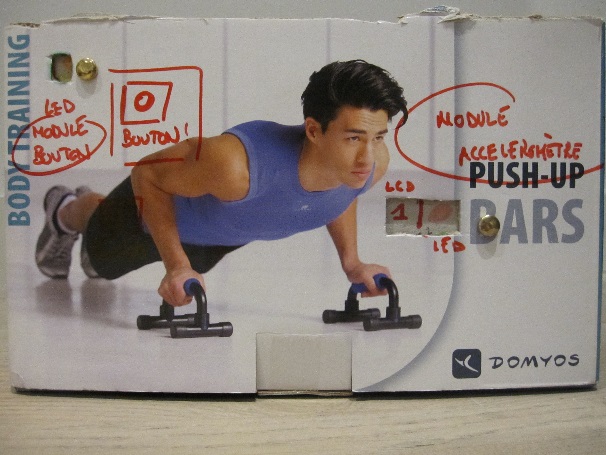
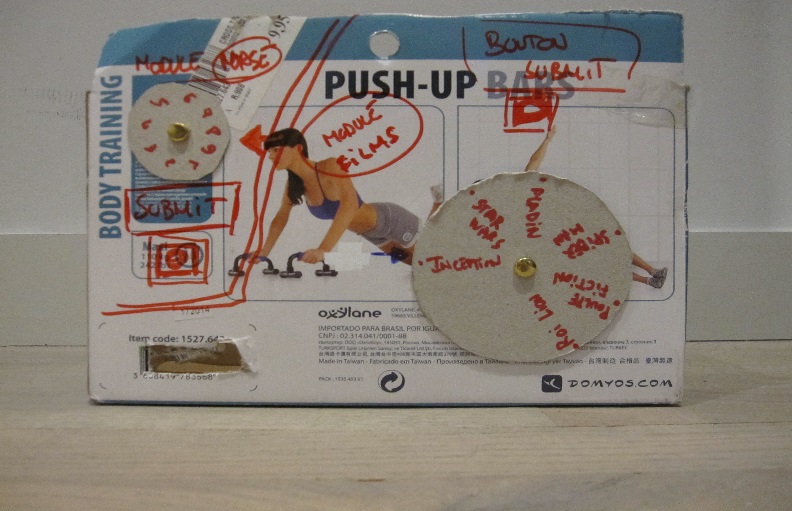


Figure photo de la maquette de la bombe

c)

Nous nous sommes basés sur le cahier des charges suivant pour réaliser ce projet :

|  |
| --- |
| Exigences : |
| Faire un jeu : Priorité aux interactions entre les joueurs |
| Utiliser le minimum d’entrées/sorties |
| Gérer plusieurs modules en même temps |
| Utiliser des sons |
| Avoir un programme propre |
| Pouvoir s’échanger des données pendant le projet |
| Avoir un manuel propre et intuitif |
| Faire un jeu réutilisable |
| Pouvoir ajouter d’autres modules facilement |

Mettre une exigence pour l’accéléromètre

Planning :

|  |  |
| --- | --- |
| Date | Objectif |
| Octobre | Mise en place concrète des différentes idées |
| Novembre | Réalisation du planning et du cahier des charges que l’on a adapté au fur et à mesure  Ebauche du manuel  Fabrication d’une maquette en carton |
| Décembre - Janvier | Réalisation des modules de manière séparée |
| Février - Mars | Mise en commun des modules |
| Avril - Mai | Réalisation du compte rendu  Organisation du programme final  Test du programme sur des gens  Faire le support du système |

# Conception :

## Création des modules :

Sans entrer dans une description fastidieuse du mini-jeu et/ou du programme, nous allons brièvement expliquer le principe de chaque module puis expliquer les principaux points.

**Module Keyboard**

(insérer une photo du schéma)

Principe : Le démineur doit jouer une séquence précise composée de 9 notes. Les 2 paramètres à prendre en compte sont :

* Le positionnement du « DO », qui peut changer de place parmi les 5 boutons. Cette position est déterminée grâce à un code.
* La séquence à jouer, qui varie selon les couleurs des LEDs allumées.

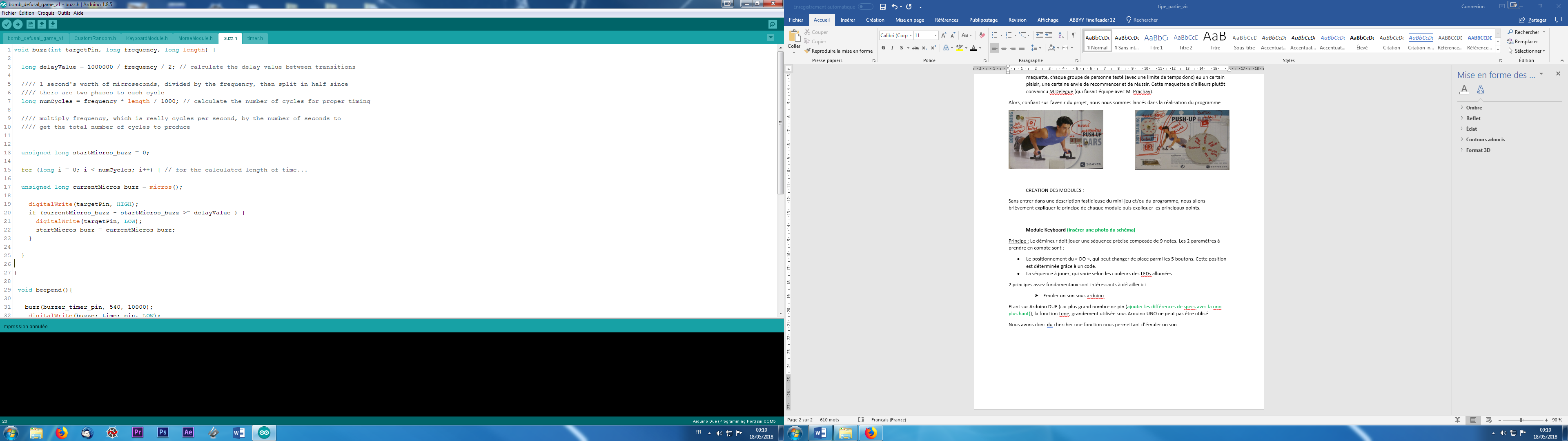
2 principes assez fondamentaux sont intéressants à détailler ici :

* Emuler un son sous arduino

Etant sur Arduino DUE (car plus grand nombre de pin (ajouter les différences de specs avec la uno plus haut)), la fonction *tone*, grandement utilisée sous Arduino UNO ne peut pas être utilisé.

Nous avons donc dû chercher une fonction nous permettant d’émuler un son.

(Programme trouvé ici https://www.faludi.com/2007/04/23/buzzer-arduino-example-code/)



Au-delà de trouver un programme qui fonctionne directement, il est intéressant de comprendre son fonctionnement.

3 attributs sont nécessaires pour cette fonction :

* Le pin du buzzer (buzzer actif, insérer une photo)
* La fréquence jouée (540 Hz pour un LA par exemple)
* La longueur de la note jouée
* Ce programme simule en réalité un signal carré (proche du signal sinusoidal désiré).
* Dans un premier temps, le programme calcule la durée d’une période du signal carré (*delayValue*).
* Puis est calculé le nombre de cycles (*numCycles*) qui vont être nécessaire pour jouer le signal carré.
* Le « temps courant », juste avant de rentrer dans la boucle qui va émuler le signal carré, est noté *start\_Micros\_buzz*.
* Enfin, on entre dans une boucle qui passe dans l’état « Allumé » dès l’entrée dans cette boucle. Le temps courant est à chaque fois actualisé dans la variable *currentMicros\_buzz*.
* Quand le buzzer aura été allumé plus de temps qu’une période (*currentMicros\_buzz - start\_Micros\_buzz >= delayValue*), il ne jouera plus de son. A ce moment, la variable *start\_Micros\_buzz* est réinitialisé.
* Cette boucle tournera autant de fois que nécessaire, c’est-à-dire autant de fois que *numCycles*.

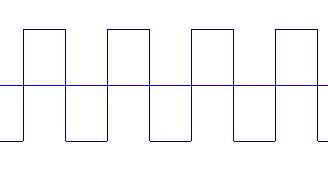


Figure : signal carré

* Boutons branchés en série

Une de nos contraintes étaient d’utiliser le moins de pin possibles. Ayant beaucoup de composants à brancher, nous ne pouvions pas utiliser 5 sorties uniquement pour les boutons. Nous avons donc utilisé une des sorties analogiques de l’arduino, puis branché en série les 5 boutons.

<https://github.com/dxinteractive/AnalogMultiButton>

[Sortie 3.3V]

|

V

[2.2K résistance]

|

V

[Sortie analogique] --> Bouton 1 --> GND

|

V

[1K résistance]

|

+--> Bouton 2 --> GND

|

V

[1K résistance]

|

+--> Bouton 3 --> GND

|

V

[1K résistance]

|

+--> Bouton 4 --> GND

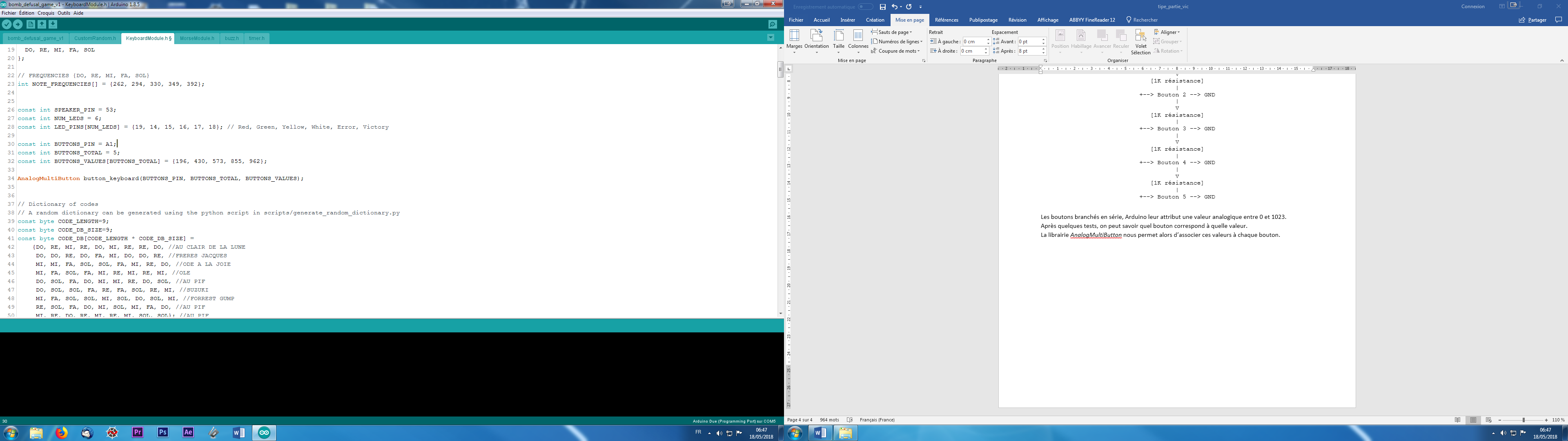
|

V

[1K résistance]

|

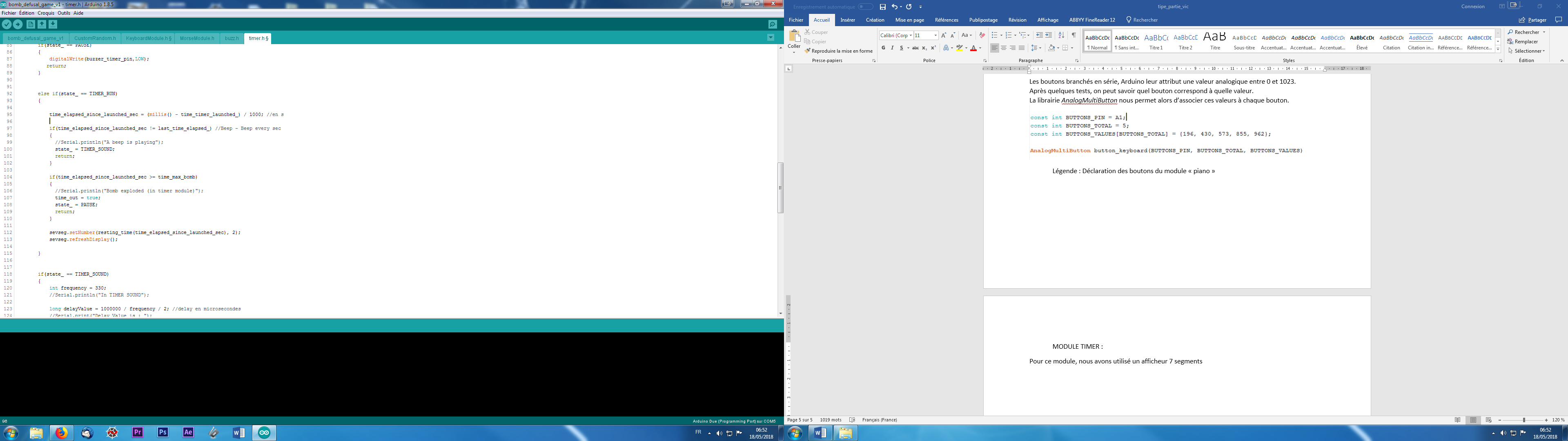
+--> Bouton 5 --> GND

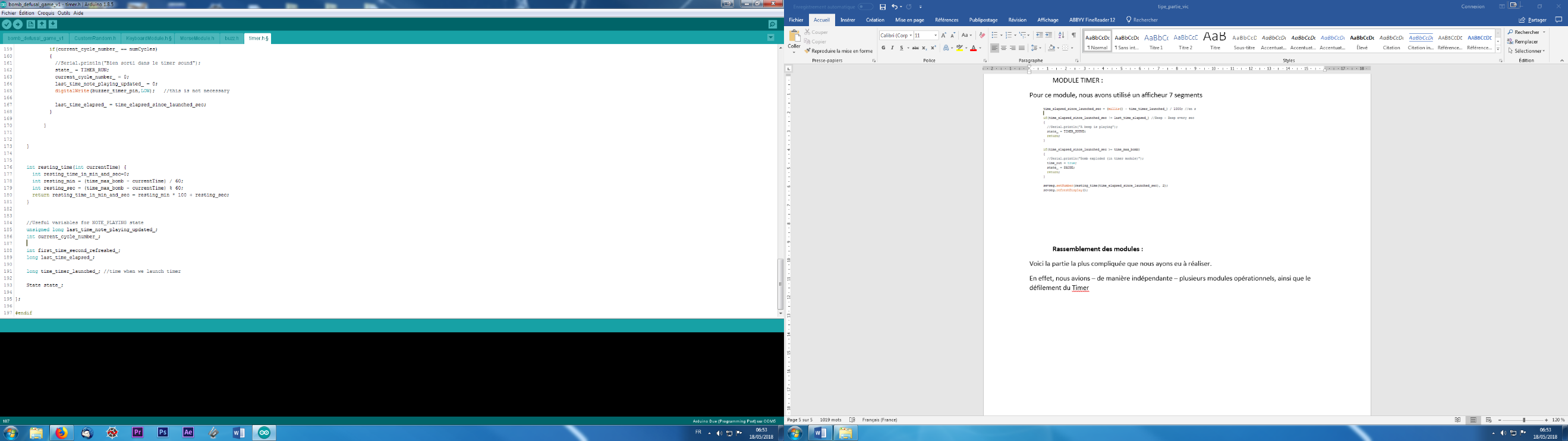
Arduino attribue une valeur analogique aux boutons branchés en série entre 0 et 1023.  
Après quelques tests, on peut savoir quel bouton correspond à quelle valeur.  
La librairie *AnalogMultiButton* nous permet alors d’associer ces valeurs à chaque bouton. 

Déclaration des boutons du module « piano »

Module Timer :

Pour ce module, nous avons utilisé un afficheur 7 segments





Dès que la bombe est lancée, le temps courant est sauvegardé dans la variable *time\_timer\_launched*.  
Ainsi est calculée le temps depuis lequel la bombe a été lancée (*time\_elapsed\_since\_launched\_sec*).  
Avec une boucle Tant Que, on limite le temps d’écoulement du timer avec la variable préalablement définie *time\_max\_bomb*.

Enfin, l’afficheur 7 segment est rafraichi et affiche continuellement le temps restant. La fonction *resting\_time* permet de séparer les minutes et les secondes, pour afficher ces 2 éléments distinctement sur l’afficheur.

Module Morse :

Principe: Le joueur devant la bombe doit communiquer des signaux qu’il voit en morse afin que l’autre joueur déchiffre le bon mot parmi neufs possibles.

Les mots sont directement convertis dans le programme afin de pouvoir changer facilement de mot sans erreurs. La fonction *ToUpperCase* transcrit les mots en majuscules pour avoir moins de lettres à coder, ensuite la fonction *replace* change toutes les lettres en suites de « . » et « - » selon le langage morse français. Il ne reste plus qu’à traduire les mots en signaux lumineux grâce à, entre autres, la fonction *millis* détaillée plus loin.

Nous avons utilisé les commande switch case pour choisir le mot en fonction de la valeur apportée par la commande Randomseed : plus simple que d’utiliser if else (partie simplification du code)

Dans le message traduit en morse il y’a quatre caractères : « . » , « - », « ; », « | » qui correspondent respectivement au court, au long, à l’espacement entre deux lettres et à l’espacement entre chaque mot. Dans ce projet, un seul mot est traduit, « | » correspond donc à l’espacement entre chaque mot mais également à l’espacement entre chaque cycle puisque que la LED traduit le mot en boucle.

Rebond du bouton :

L’appui sur un bouton poussoir peut ne pas transmettre qu’un seul contact mais plusieurs, dépendant de la qualité du bouton.

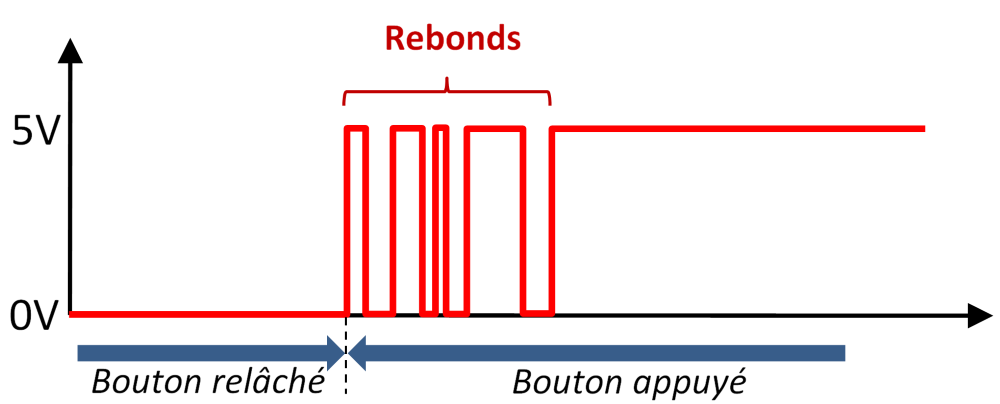


schéma du signal émis par le rebond d'un bouton

Il y’a deux façons de résoudre ce problème :

Utiliser un condensateur en parallèle pour adoucir le passage transitoire ou attendre logiciellement un court instant (250 millisecondes) avant le décompte d’un autre appui sur le bouton pour être sûr de ne pas prendre en compte les rebonds. C’est la deuxième solution qui a été choisie.

Module accéléromètre

Il était prévu mais n’a pas été intégré dans le projet faute de temps. Le joueur devait bouger un cube en fonction des indications de l’autre joueur. Ce module aurait rajouté une véritable dimension physique au jeu par rapport à un jeu virtuel. L’intention initiale était de faire des translations du cube dans l’espace. Or l’accéléromètre ne relève que l’accélération et l’orientation dans l’espace. Il n’était pas possible de relever avec précision les mouvements effectués. Nous avions donc adapté le manuel du démineur afin que le joueur doive effectuer des rotations du cube ce que l’accéléromètre permettait de faire avec plus de précision.

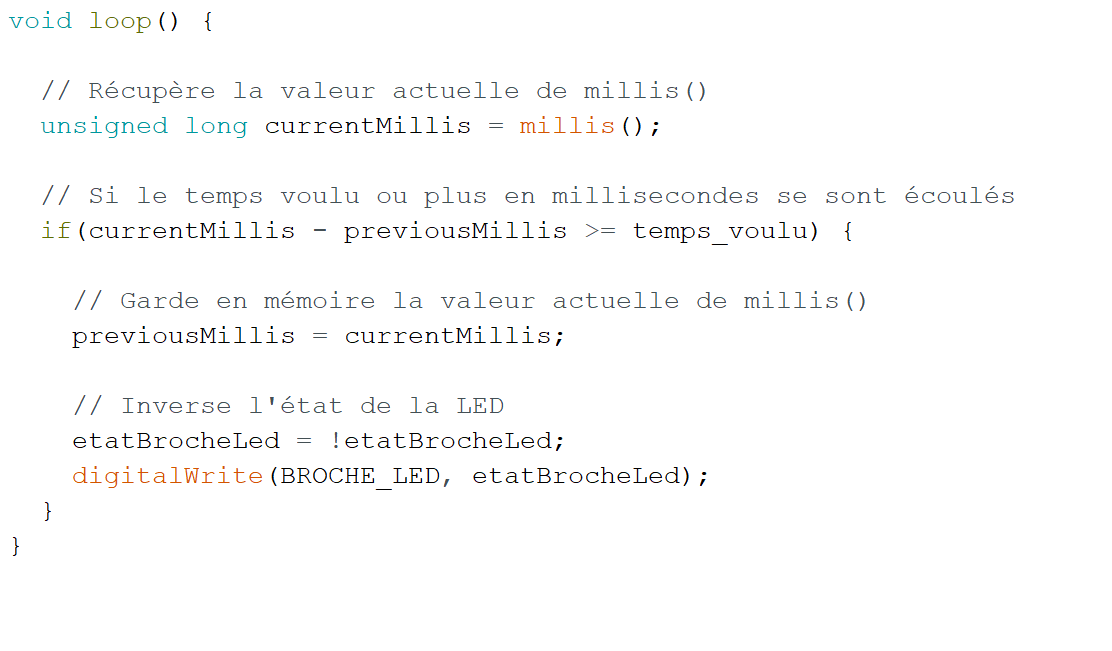
Une bibliothèque Arduino est dédiée à l’accéléromètre. Après avoir inclus la bibliothèque et initialisé l’accéléromètre il suffit de relever les valeurs relevées par celui-ci. Le joueur aurait dû mettre le cube dans la position indiquée par le manuel et appuyer sur un bouton pour relever la valeur des positions selon les axes X, Y et Z.

## Gérer plusieurs modules en même temps :

Voici la partie la plus compliquée que nous ayons eu à réaliser.

La fonction *delay* est une fonction bloquante qui ne permet pas d’effectuer plusieurs tâches en parallèle. Ainsi, à chaque émulation de son (qui utilise la fonction *delayMicros*), le programme se gelait entièrement (l’afficheur 7 segments était principalement impacté). Nous avons donc opté pour l’utilisation de la fonction *millis* qui gère le temps sans être une fonction bloquante selon le modèle :

Pour expliquer ce concept, voici un programme non bloquant :



Ici, les 2 LEDs clignotent de manière complètement indépendante. Avec la fonction *delay*, il serait impossible de réaliser un tel programme.

Pour réaliser cela, nous avons contacté **Henri Rebecq**, frère de Victor, ingénieur diplômé de l’école Paris Telecom et travaillant actuellement sur sa thèse à l’ETH de Zurich. Nous n’avions aucune idée de comment réaliser cet assemblage de tous les modules, autant logiquement que techniquement (le langage arduino étant proche du C, langage que nous ne connaissons pas bien).

Pour utiliser cette fonction *millis*, le programme a donc besoin de savoir – à chaque instant – dans quel état il se situe.

Pour cela, Henri nous a conseillé de réaliser une machine à état. Nous allons ici voir un exemple sur le module « Piano », sachant que les autres modules ont été réalisé d’une manière similaire (par souci de cohérence du programme).

INSERER LE DIAGRAMME D’ETAT DU KEYBOARD MODULE

Selon le diagramme ci-dessus, nous avions donc défini plusieurs états :

* PAUSE : état initial du module. Lorsque la fonction *reset* est appelé (par l’appui d’un bouton par exemple), l’état passe à GAME.
* GAME : état principal, où l’appui sur un bouton est détecté. Plusieurs sous-états se suivent :
  + NOTE\_PLAYING : suit directement l’état GAME ; permet de jouer directement la note correspondant au bouton venant d’être appuyé
  + CHECK\_INPUT\_VALUE : traite l’entrée du bouton (entrée correcte ou fausse).
  + PLAY\_FEEDBACK\_CORRECT : allume une LED verte.
  + PLAY\_FEEDBACK\_WRONG : allume une LED rouge et recommence le code à zéro.
  + CODE\_FOUND : Le jeu revient à son état initial (PAUSE)

Le même principe a été appliqué pour le module « Morse » ainsi que pour le « timer ».

(VOIR LE DIAGRAMME D’ETAT DU TIMER A RAJOUTER).

La fonction *attachInterrupt* permet d’arriver au même résultat. A chaque appui sur un bouton celle-ci effectue une tâche en priorité sur les autres. Cette fonction n’est cependant pas durable sur des programmes longs et ne peut pas être utilisée un grand nombre de fois en tant qu’une seule fonction ne peut être appelée dans le même temps.

Henri nous a aussi conseillé d’améliorer la lisibilité de notre programme. Pour cela, nous avons mis nos variables dans la même casse : en anglais, sans majuscule, avec un underscore. Nous avons aussi réalisé nos modules dans la même idée (machines à états appelant les mêmes états principaux : PAUSE et GAME).

## Faire un jeu réutilisable :

### Solutions :

Nous avons décidé d’utiliser des valeurs aléatoires et de programmer plusieurs chemins possibles en fonctions de ces valeurs pour que le jeu présente différents cas et qu’ainsi les parties de soient pas répétitives. Ces différents chemins se feront directement à l’intérieur du module comme pour le morse ou au travers du numéro de série. Les chemins changent à chaque redémarrage de la carte pour proposer des parties différentes.

### Mise en œuvre :

Nous avons utilisé la fonction random(min, max) pour générer des variables aléatoires. Il est à noter que la valeur minimale est incluse et que la valeur maximale est exclue, c’est pourquoi dans le module morse pour neuf mots nous avons pris la valeur alea = random(1,10).

La fonction random ne génère qu’une valeur aléatoire par téléversement, et non pas à chaque redémarrage de la carte. Nous avons donc rajouté la fonction randomSeed(analogRead(0)) qui lit une grandeur physique sur la pin 0 qui n’est branchée à rien. Cette fonction permet de générer une valeur aléatoire différente à chaque mise sous tension de la carte en plus d’obtenir des valeurs plus aléatoires.

## Outils informatiques utilisés :

Nous avons utilisé le site d’échange de données GitHub afin de réunir les programmes et autres fichiers dans un dossier en ligne. Cela permet le partage des programmes et d’autres fichiers afin de les ouvrir sur n’importe quel ordinateur ayant les droits d’accès au projet.

Le logiciel Fritzing pour réaliser les schémas du projet, ce logiciel permet également de simuler un projet et de créer les plans en circuit imprimé. Un schéma du circuit est essentiel pour pouvoir communiquer aux autres comment comprendre et réaliser un certain programme.

Nous avons aussi utilisé l’hébergeur de vidéos YouTube pour partager les vidéos de test.

Enfin nous avons évidemment utilisé Arduino pour la programmation du projet

# Exploitation

Nous avons réalisé plusieurs test avec des binômes de personnes différente. Il a fallu au début trouver le juste milieu au niveau de la difficulté et corriger les bugs rencontrés.

Les premiers essais sont difficiles mais les joueurs apprennent vite et s’améliorent au fil des parties.

Nous avons filmé le premier essai de Mr. Lanoix devant la bombe (disponible ici : <https://www.youtube.com/watch?v=MbPvuHy70Hc&feature=youtu.be> )

# Conclusion :

Le cahier des charges du projet a en grande partie été rempli. Le jeu est opérationnel et ne présente pas de bugs.

Les cartes Arduino regorgent de possibilités et sont donc un bon support pour ce projet le programme n’occupe que 7% de la mémoire de la carte Arduino Due, d’autant plus que la communauté y est grande et que le projet est continuel.

SIMULATIONS (EXCEL)

CONTINUITE DU PROJET (CLASSE / RAJOUTE MODULE / CIRCUIT IMPRIME)

# Bibliographie :

* <http://arduino.blaisepascal.fr/index.php/2017/11/30/le-bouton-poussoir/> schéma du rebond du bouton et explications
* <https://forum.arduino.cc/index.php?topic=492308.0> programme pour coder un mot en morse
* <https://www.carnetdumaker.net/articles/faire-plusieurs-choses-la-fois-avec-une-carte-arduino/> multitâche sur Arduino
* <http://www.locoduino.org/spip.php?article131> tableau ?
* <http://wikifab.org/wiki/Timer_:_Un_minuteur_%C3%A0_base_d%27Arduino> Timer
* <http://forum.arduino.cc/index.php/topic,66206.0.html> <https://github.com/sebnil/DueFlashStorage/blob/master/examples/DueFlashStorageExample/DueFlashStorageExample.ino> custom random