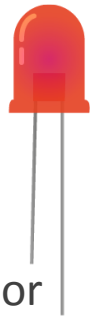
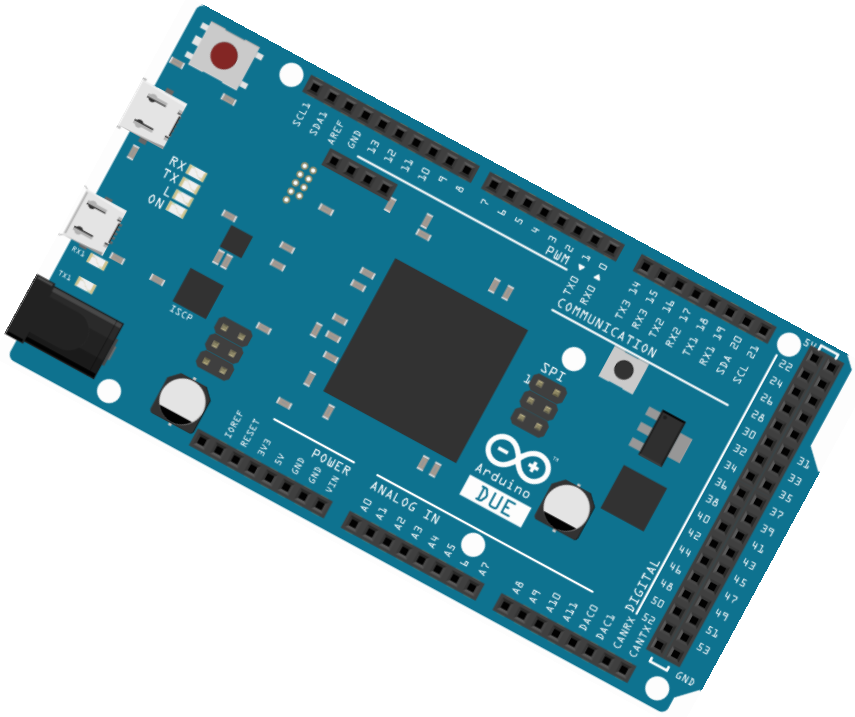
2017-2018

Rapport de TIPE

NAHMIAZ Thomas REBECQ Victor



Est-il possible de faire un jeu sous Arduino ?



PTSI 1

Table des matières

Résumé :

Le projet proposé est un jeu physique sur Arduino. Le principe est de désamorcer une bombe, en équipe de deux joueurs ou plus. Deux équipes coopéreront lors du jeu :

Le démineur : il peut voir la bombe, interagir avec elle mais ne connait pas le procédé pour désamorcer la bombe.

Les experts : ils possèdent un manuel qui permettra de résoudre les énigmes, mais ne peuvent pas interagir physiquement ou voir la bombe.

Ce jeu est donc purement coopératif, et la communication est primordiale. Les différents mini-jeux doivent être résolus dans un ordre quelconque mais dans un temps imparti de 5 minutes. Au-delà de ce temps ou si les joueurs font plus de 2 fautes, la bombe explosera.

Introduction :

Premièrement nous aborderons la mise en place des idées à propos du projet, ses premières maquettes physiques ainsi que les exigences du projet. Puis, nous montrerons comment nous avons procédé pour obtenir plusieurs mini-jeux, jouable en même temps sur Arduino. Enfin, nous réaliserons un ensemble de tests, et nous évaluerons comment le projet pourra désormais évoluer.

I. Mise en place :

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**, 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 incompréhensions, qui nous a montré le potentiel que pouvait avoir ce concept.

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 difficile 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 »). 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.

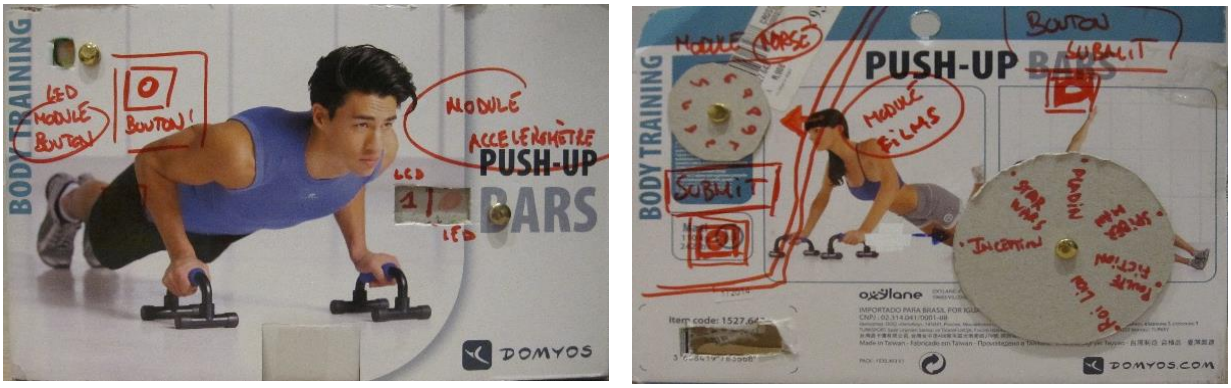


Figure 1 : Photo de la maquette de la bombe sur une boite en carton

Lancement du projet, mise en place des exigences

En se basant sur nos idées nous avons conçu le cahier des charges suivant que nous avons suivit tout au long de l'année :

Exigences :
Faire un jeu : Priorité aux interactions entre les joueurs.
Utiliser le minimum d'entrées/sorties.
Gérer plusieurs modules en même temps.
Pouvoir émuler des sons.
Avoir un programme propre.
Pouvoir s'échanger des données facilement pendant le projet (entre concepteurs du projet).
Avoir un manuel propre et intuitif.
Faire un jeu réutilisable, rejouable.
Pouvoir ajouter d'autres modules facilement.
Utiliser un accéléromètre pour donner une vraie dimension physique au jeu.

Nous avons également mis en place un planning tout au long du projet afin de mieux nous organiser. Voir l'annexe 2 pour voir la représentation du planning sous forme de diagramme de GANTT.

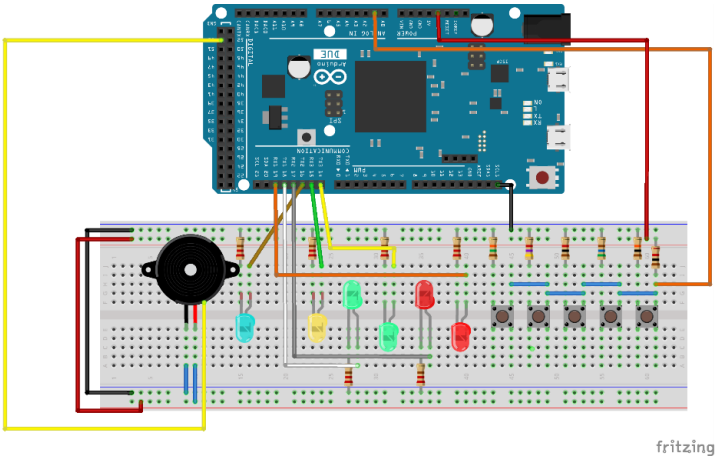
II. Conception :

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 de leur réalisation.

Module Keyboard

Figure 2 : Schéma du module Keyboard



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 d'entrées/sorties)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.

Name	Processor	Operating/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	UART
Uno	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/6	1	2	32	Regular	1
Due	ATSAM3X8E	3.3 V / 7-12 V	84 MHz	12/2	54/12	-	96	512	2 Micro	4

Figure 3 : Comparaison des caractéristiques entre Arduino UNO et DUE

```
void buzz(int targetPin, long frequency, long length) {

  long delayValue = 1000000 / frequency / 2; // calculate the delay value between transitions

  //// 1 second's worth of microseconds, divided by the frequency, then split in half since
  //// there are two phases to each cycle
  long numCycles = frequency * length / 1000; // calculate the number of cycles for proper timing

  //// multiply frequency, which is really cycles per second, by the number of seconds to
  //// get the total number of cycles to produce


  unsigned long startMicros_buzz = 0;

  for (long i = 0; i < numCycles; i++) { // for the calculated length of time...

  unsigned long currentMicros_buzz = micros();

    digitalWrite(targetPin, HIGH);
    if (currentMicros_buzz - startMicros_buzz >= delayValue ) {
      digitalWrite(targetPin, LOW);
      startMicros_buzz = currentMicros_buzz;
    }

  }
```

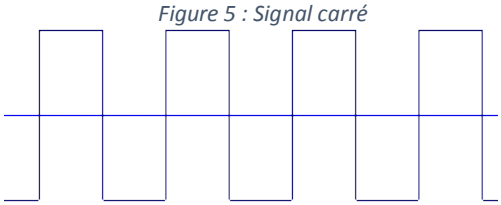
Figure 4: principe de la programmation du timer

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

3 attributs sont nécessaires pour cette fonction :

- La sortie du buzzer (buzzer actif, insérer une photo)
- La fréquence jouée (540 Hz pour un LA par exemple)
- La longueur de la note jouée

Figure 5 : Signal carré



- 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*.

➢ Boutons branchés en série

Une de nos contraintes était d'utiliser le moins d'entrées/sorties 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.

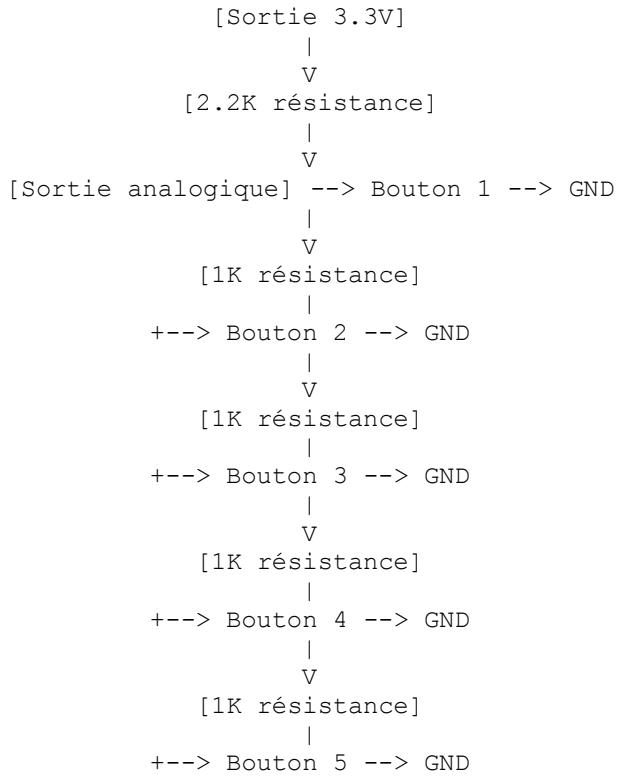


Figure 6 : Diagramme d'utilisation d'Analog Multi Button (source : gitHub librairie AnalogMultiButton)

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.

```
const int BUTTONS_PIN = A1;
const int BUTTONS_TOTAL = 5;
const int BUTTONS_VALUES[BUTTONS_TOTAL] = {196, 430, 573, 855, 962};

AnalogMultiButton button_keyboard(BUTTONS_PIN, BUTTONS_TOTAL, BUTTONS_VALUES)
```

Figure 7 : Déclaration des boutons du module « piano »

Module Timer :

Pour ce module, nous avons utilisé un afficheur 7 segments

```
time_elapsed_since_launched_sec = (millis() - time_timer_launched_) / 1000; //en s

if(time_elapsed_since_launched_sec != last_time_elapsed_) //Beep - Beep every sec
{
  //Serial.println("A beep is playing");
  state_ = TIMER_SOUND;
  return;
}

if(time_elapsed_since_launched_sec >= time_max_bomb)
{
  //Serial.println("Bomb exploded (in timer module)");
  time_out = true;
  state_ = PAUSE;
  return;
}

sevseg.setNumber(resting_time(time_elapsed_since_launched_sec), 2);
sevseg.refreshDisplay();
```

```
int resting_time(int currentTime) {
  int resting_time_in_min_and_sec=0;
  int resting_min = (time_max_bomb - currentTime) / 60;
  int resting_sec = (time_max_bomb - currentTime) % 60;
  return resting_time_in_min_and_sec = resting_min * 100 + resting_sec;
}
```

Figure 8 : Principes de la programmation du timer

Dès que la bombe est lancée, le temps courant est sauvegardé dans la variable *time_timer_launched*.
Ainsi est calculé 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 :

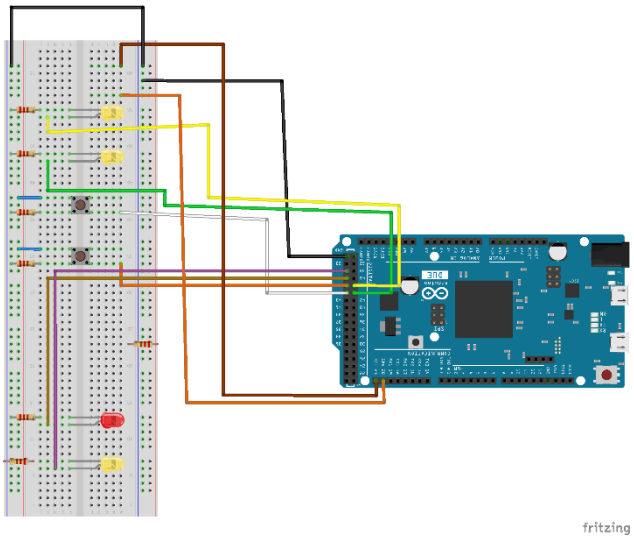


Figure 9 : Schéma du module Morse

Principe: Le joueur devant la bombe doit communiquer des signaux lumineux qu'il voit en morse afin que l'autre joueur déchiffre le bon mot parmi les 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 la commande *switch case* pour choisir le mot en fonction de la valeur apportée par la commande Randomseed : l'avantage de cette commande par rapport à un *if .. else* est de raccourcir et clarifier nettement la syntaxe.

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.

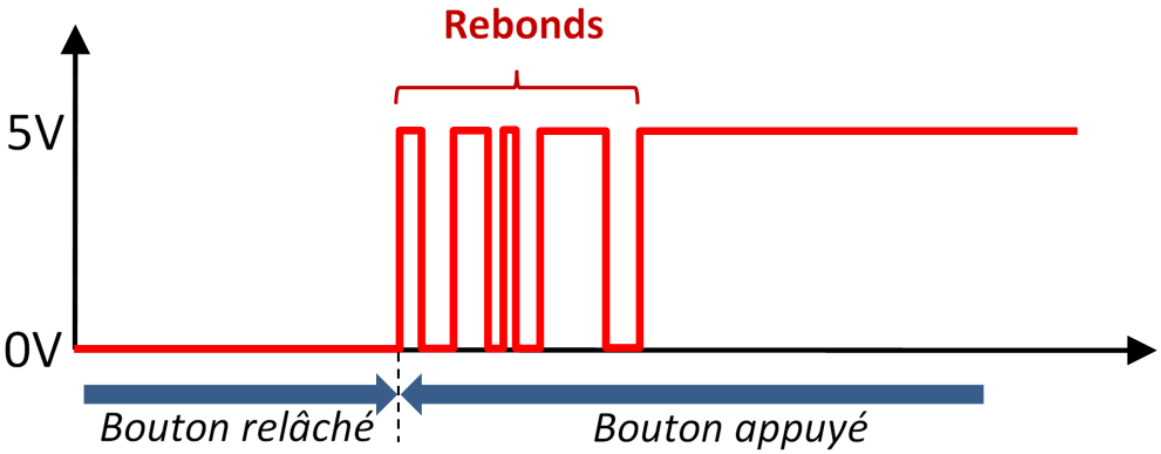


Figure 10: Schéma du signal émis par le rebond d'un bouton

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 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 :

```
void loop() {

  fct_blinkled1();
  fct_blinkled2();

}


void fct_blinkled1() {
  currentMillis = millis();
  if (etatled1 == LOW)
  {
    if (currentMillis - start_led1 >= blinkled1)
    {
      etatled1 = HIGH;
      start_led1 = currentMillis;
      digitalWrite(led1, etatled1);
    }
  }

  else
    {
      if (currentMillis - start_led1 >= blinkled1)
      {
        etatled1 = LOW;
        start_led1 = currentMillis;
        digitalWrite(led1, etatled1);
      }
    }
```

```
void fct_blinkled2() {
  currentMillis = millis();
  if (etatled2 == LOW)
  {
    if (currentMillis - start_led2 >= blinkled2)
    {
      etatled2 = HIGH;
      start_led2 = currentMillis;
      digitalWrite(led2, etatled2);
    }
  }

  else
    {
      if (currentMillis - start_led2 >= blinkled2)
      {
        etatled2 = LOW;
        start_led2 = currentMillis;
        digitalWrite(led2, etatled2);
      }
    }

}
```

Figure 11 : Programme de deux Leds clignotant de manière désynchronisée

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).

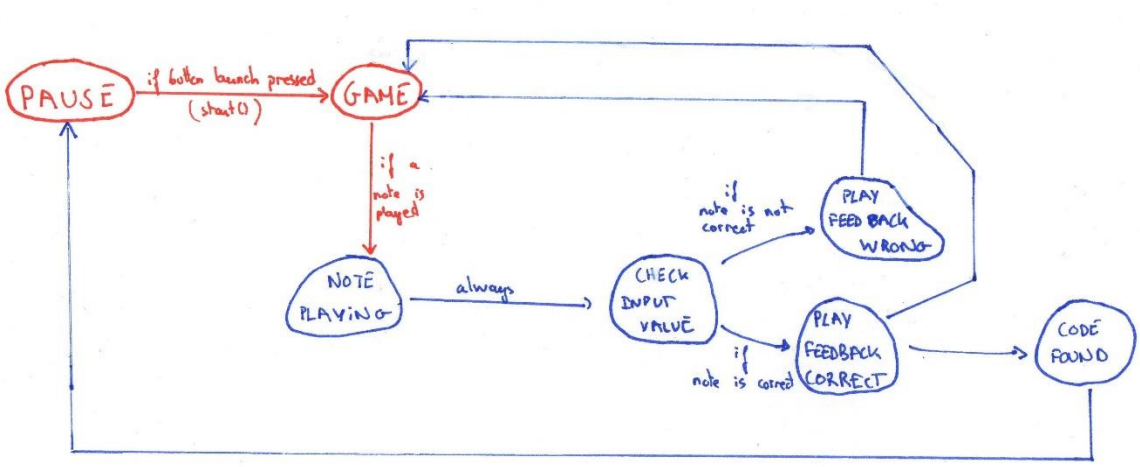


Figure 12 : Diagramme d'états du module Keyboard

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 ou « Timer ». (Voir annexe 2).

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 ne 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 sortie 0 qui n'est branchée à rien. Cette fonction permet de générer une valeur aléatoire différente à chaque mise sous tension de la carte.

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. Notre projet sur Github :

https://github.com/trickart73/tipe_ktane/tree/master/bomb_defusal_game_v1

Le logiciel **Fritzing** a été utilisé 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

III. Exploitation

Nous avons réalisé plus d'une vingtaine de tests avec des binômes différents. Il a fallu au début trouver le juste milieu au niveau de la difficulté et corriger les bogues rencontrés.

Joueurs		Temps (en min)			Remarques	
		Temps total	Temps Morse	Temps Piano	Rmq (en cours de jeu)	Rmq (après jeu)
Papa	Maman	6 (explosée)	6	NON TROUVE	Difficulté à lire le morse	Manuel non clair
		XXX			Bug : Bouton Keyboard s'active tout seul	
		6 (explosée)	3	NON TROUVE	Morse trouvé facilement	
		6 (explosée)	NON TROUVE	NON TROUVE	Morse trop dur	
		6 (explosée)	NON TROUVE	NON TROUVE	Morse trop dur	
		6 (explosée)	NON TROUVE	3		
		XXX			Confusion entre "1" et "I"	
		Trouvée : 6' restantes	4	2	Déchiffrage led trop facile	
Thibault	Maman					
		6 (explosée)	5	NON TROUVE	Pas le temps pour le piano	Morse : mieux
		Trouvée : 2'30 restantes	1'30	2'	Beaucoup trop facile le morse pour Thibault	

Figure 13 : Compte rendu des premières parties jouées

Nous voyons bien qu'au fur et à mesure des parties, les joueurs progressent.

Toutefois, les joueurs ayant testé le jeu ont été satisfaits et ont ressenti la frustration de la défaite des premières parties, leur donnant envie de rejouer.

Il est aussi important de noter que depuis ces 10 premières tentatives – où de nombreuses modifications ont été apportées au projet – aucun bogue (venant du programme ou de la conception des manuels) n'a été constaté. En effet, sur les 20 tentatives suivantes, le seul problème que nous ayons eu est un débranchement de fils (dû aux grands nombres de connexions).

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

Modifications	
Modifications manuels	Modifications programme
Changements : rôle + clair des deux joueurs / morse tourne	Augmentation espace entre les lettres
	Résolution bug bouton keyboard
	Diminution espace entre les lettres
	Augmentation espace entre les lettres
Rajout sens bouton keyboard	
	"I" devenu "L"
Rajout schémas module	Rajout du diagramme (2nde partie module keyboard)
	Rajout : led erreur / validation du morse / compte morse /
	Diminution espace entre les lettres

Figure 14 : Modifications apportées suites aux tests

Conclusion :

Le cahier des charges du projet a en grande partie été rempli. Le jeu est désormais opérationnel.

Les cartes Arduino regorgent de possibilités et sont donc un bon support pour ce projet. En plus d'avoir une grande communauté, les accessoires électroniques sont peu chers et permettent de réaliser des maquettes facilement. Les cartes Arduino ont une grande capacité de calcul et de stockage étant donné que le code ne prend que très peu d'espace. Le programme que nous avons conçu n'occupe que 7% de la mémoire de la carte et permet donc un large développement du projet ultérieurement.

Pour finir, que peux devenir le projet ? Aujourd'hui, nous avons un jeu fonctionnel, comprenant 2 modules. Le jeu est grandement réutilisable avant de devenir facile. De plus, nous avons réalisé l'ensemble des modules selon la même architecture, en utilisant des classes. Sans rentrer dans les détails de conception, cela nous a permis d'obtenir un fichier « main » court et propre, idéal de tout programme. Les autres modules sont réalisés dans des fichiers à part (les « headers », avec pour extension « .h »).

Nous pouvons donc rajouter facilement des modules, à l'unique condition qu'ils ne soient pas bloquants !
Dans la conception en elle-même, Arduino reste un prototypage, avec des plaques d'essais. Nous pourrions facilement envisager de remplacer chaque module par un circuit imprimé, qui peut facilement se rajouter au jeu. Nous pourrions aussi rajouter une boîte plus adaptée, ou encore une batterie externe, ou un système de difficulté (Facile ou Difficile), car nous avons remarqué durant les phases tests que la difficulté n'était pas ressentie de la même manière pour un étudiant en sciences qu'un étudiant en lettres (des calculs parfois conséquents peuvent être effectués).

Enfin, si nous décidons d'améliorer ce TIPE par la suite, nous pourrions aussi le porter sur Arduino UNO (avec l'aide d'un **74HC595**, permettant de démultiplier le nombre de sorties/entrées sur l'Arduino). Techniquement parlant, l'Arduino Uno serait largement capable de recevoir et d'exécuter un tel programme.
L'ultime étape serait d'utiliser un microcontrôleur (chipset) : l'Arduino ne deviendrait alors utile que pour téléverser le programme (interface chipset / ordinateur).

Remerciements

Nous remercions bien évidemment nos 2 professeurs, M. **Lanoix** et M. **Delegue**, qui nous ont guidé tout du long de ce projet. Un grand merci à **Henri Rebecq**, sans qui nous n'aurions pu réaliser ce projet tant son aide a été précieuse. Les connaissances qu'il nous as transmises - tant la logique et le raisonnement de construction d'un programme complexe, que des outils techniques comme les classes – nous seront certainement utile pour le reste de nos années de programmeur. Un merci aux parents (qui ont toujours été les premiers testeurs de versions pas toujours très abouties) et à tous ceux qui ont pu tester le jeu, leurs retours sont indispensables à l'amélioration du jeu.

Bibliographie :

- CFaury, « Le bouton poussoir », http://arduino.blaisepascal.fr/index.php/2017/11/30/le-bouton-poussoir/, 02/18

- WMLogistic, « Code Morse avec une Led », https://forum.arduino.cc/index.php?topic=492308.0, 12/17

- TamiaLab, « Faire plusieurs choses à la fois avec une carte Arduino », https://www.carnetdumaker.net/articles/faire-plusieurs-choses-la-fois-avec-une-carte-arduino/, 03/18

- David LeGall, « Un minuteur à base d'Arduino », http://wikifab.org/wiki/Timer_:_Un_minuteur_%C3%A0_base_d%27Arduino, 11/17

- Coding Badly, « The reliable but not very sexy way to seed random », http://forum.arduino.cc/index.php/topic,66206.0.html, 01/18

- Rob Faludi, « Buzzer Arduino Example Code », https://www.faludi.com/2007/04/23/buzzer-arduino-example-code/, 12/17

- Dxinteractive, « AnalogMultiButton », https://github.com/dxinteractive/AnalogMultiButton, 01/18

Annexes

Annexe 1

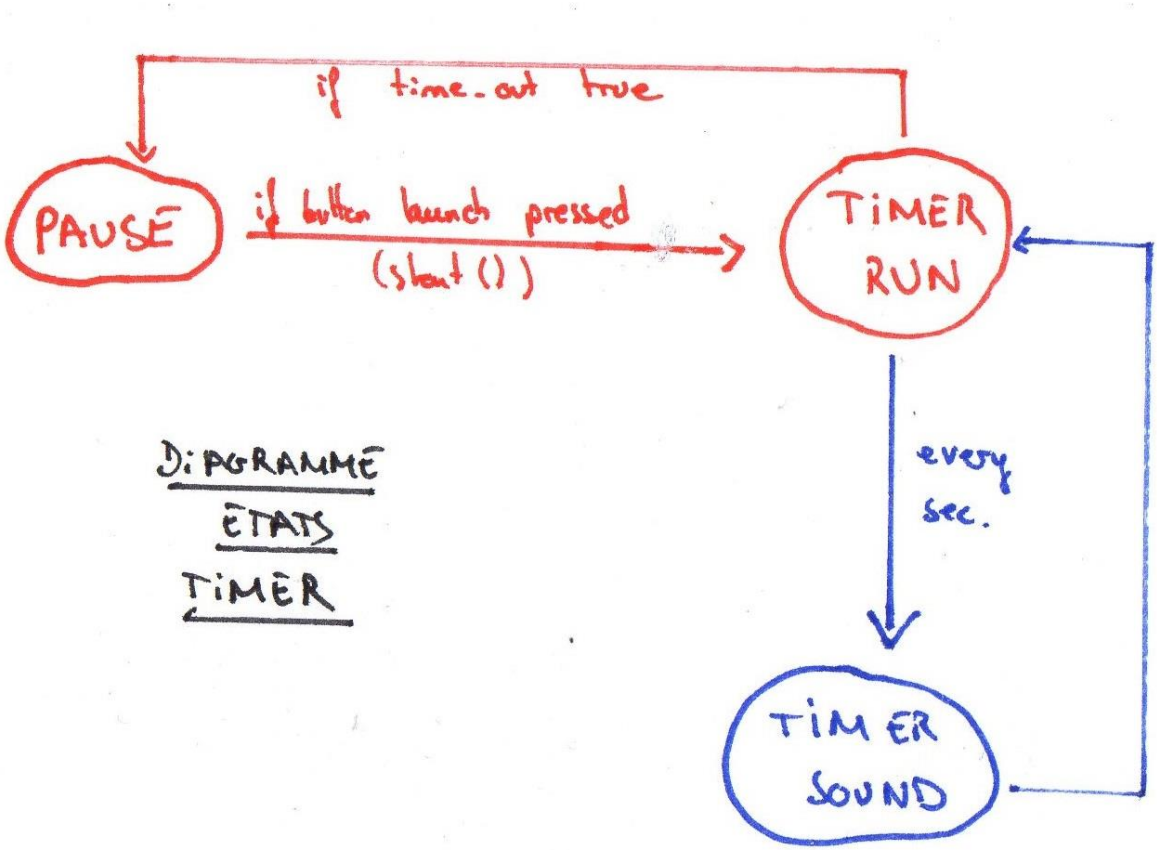
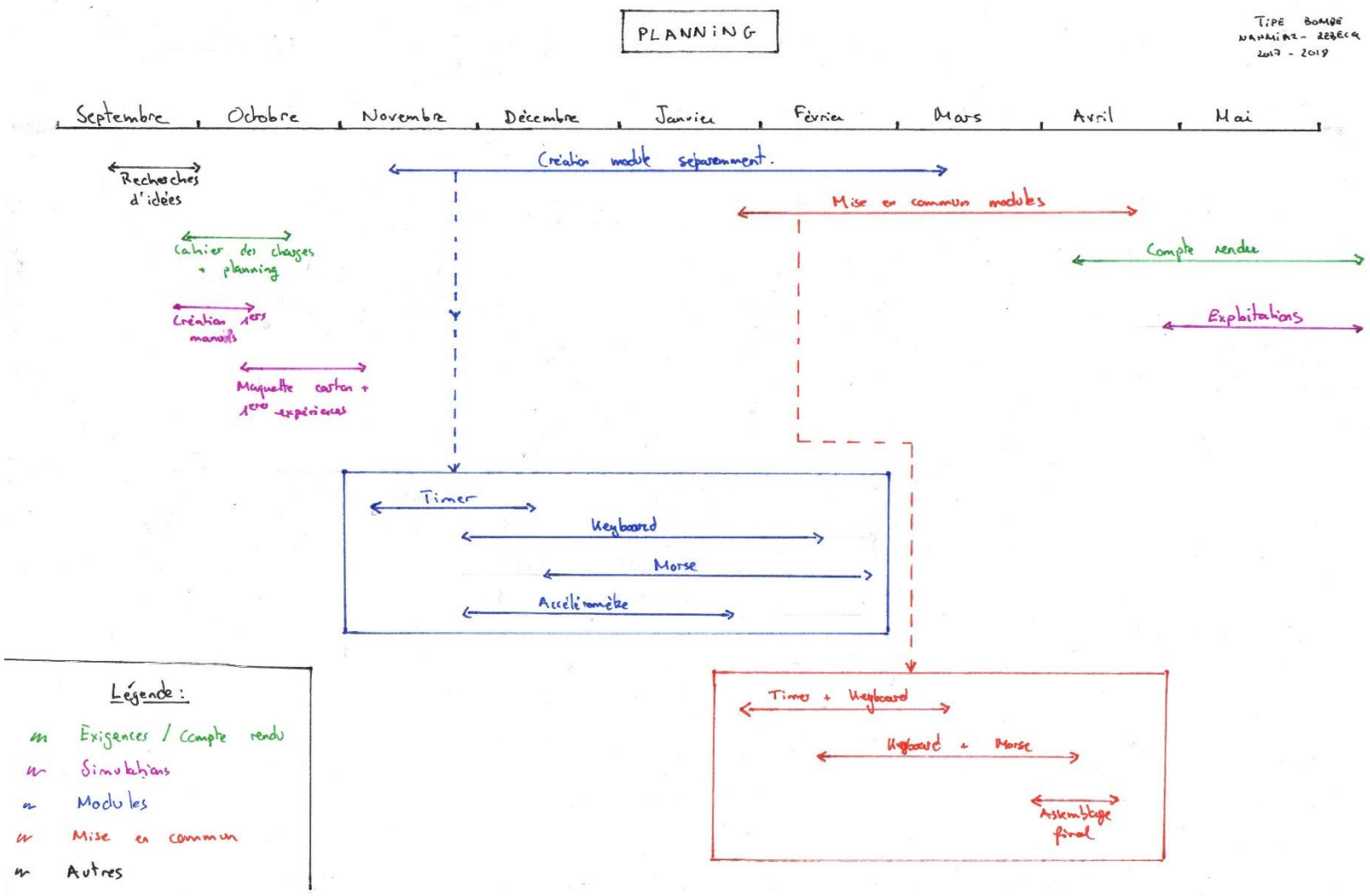


Figure 15 : Diagramme d'états Timer

Annexe 2 :

Figure 16: Planning (GANTT)



Annexe 3 :

HALL OF FAME

A l'heure actuelle, un seul duo à réussi à désamorcer la bombe en 5 minutes dans les conditions suivantes :

- Pas de triche
- Aucune aide des créateurs du jeu

Clémence Potin & Antoine Vermorel

(temps restant : 52 secondes)

Annexe 4 :

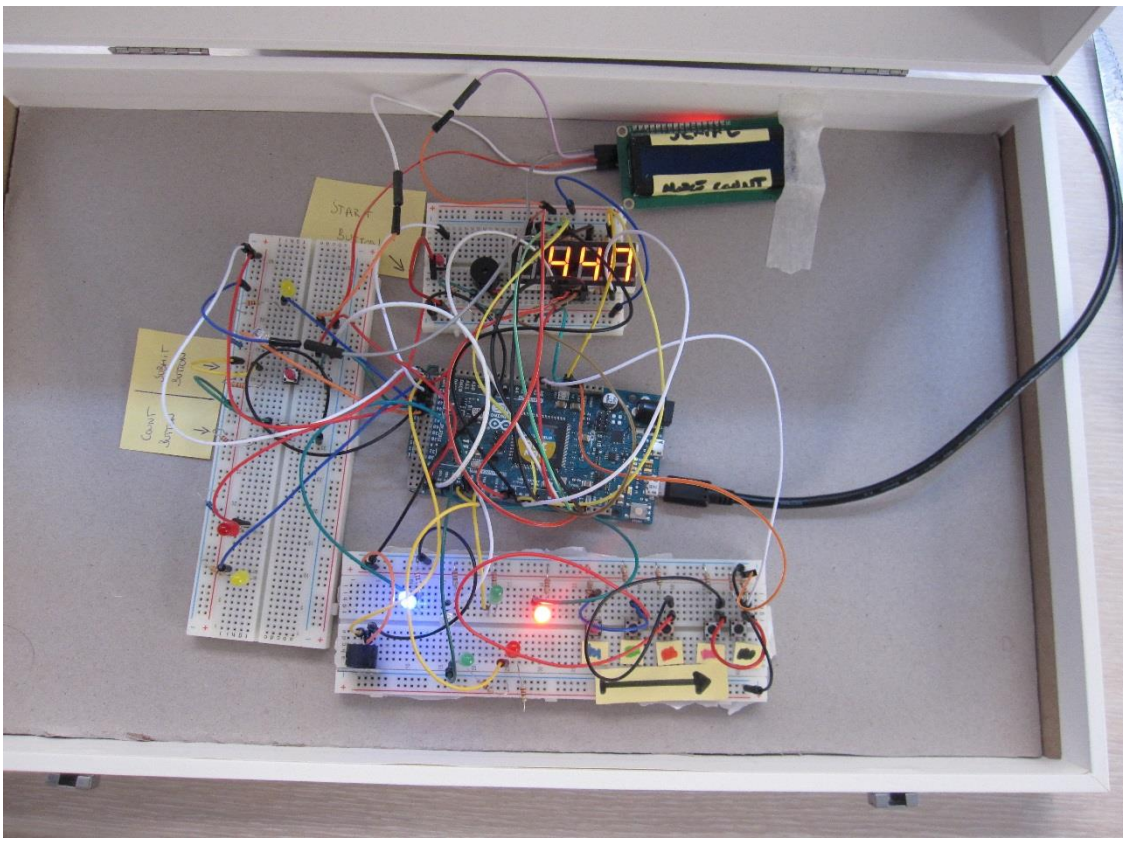


Figure 17 : Photo jeu dans son ensemble

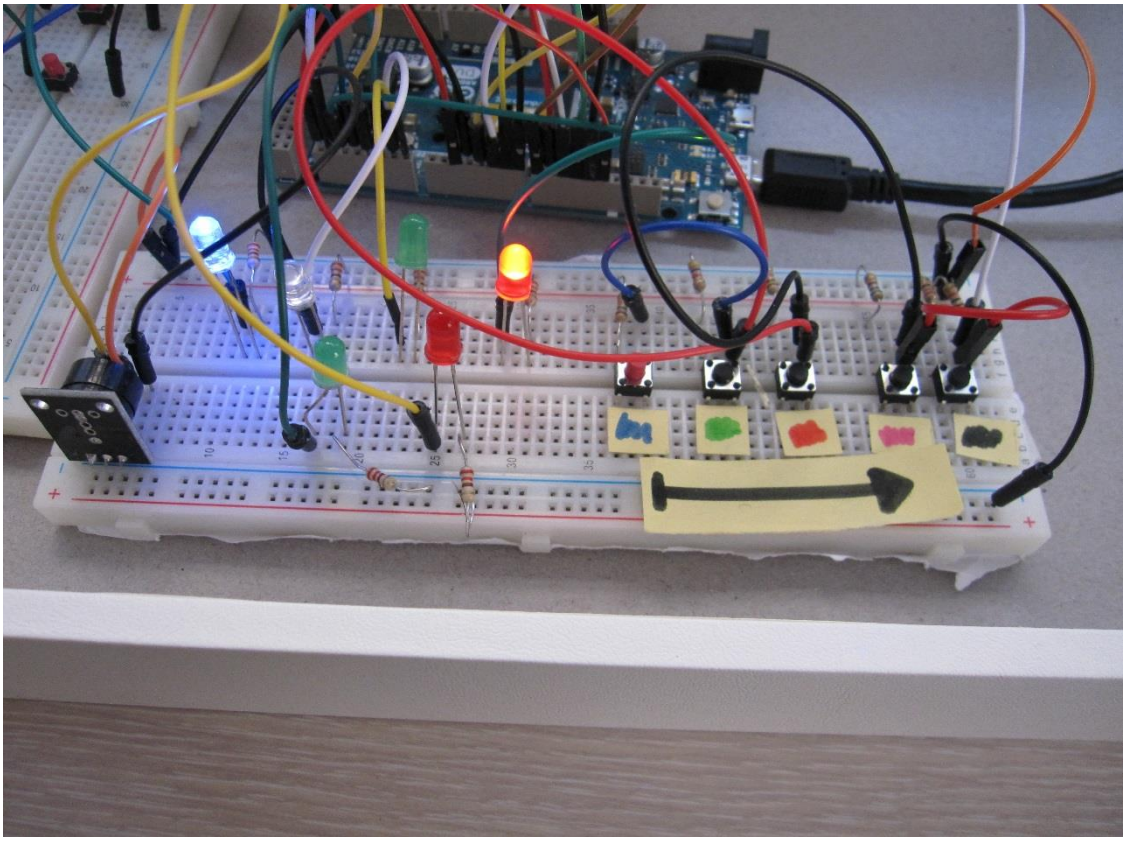


Figure 18 : Photo Keyboard Module

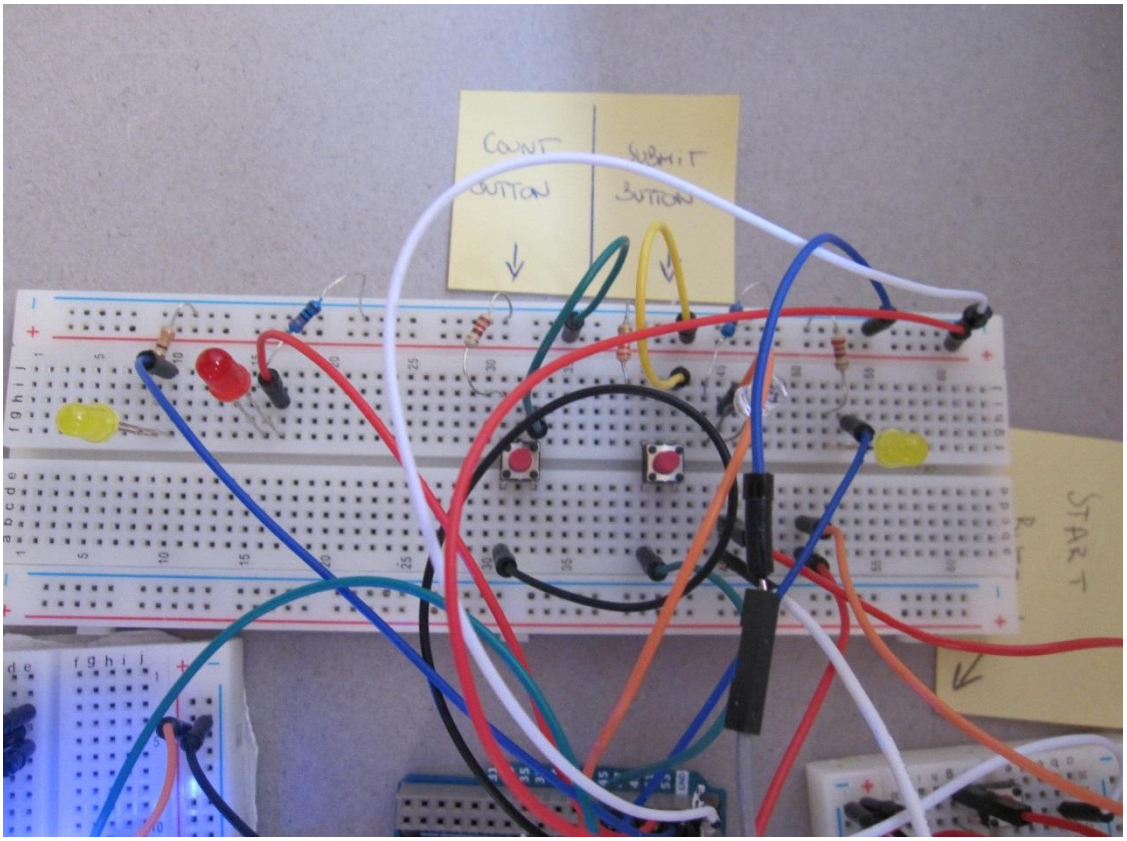


Figure 19 : Photo Morse Module

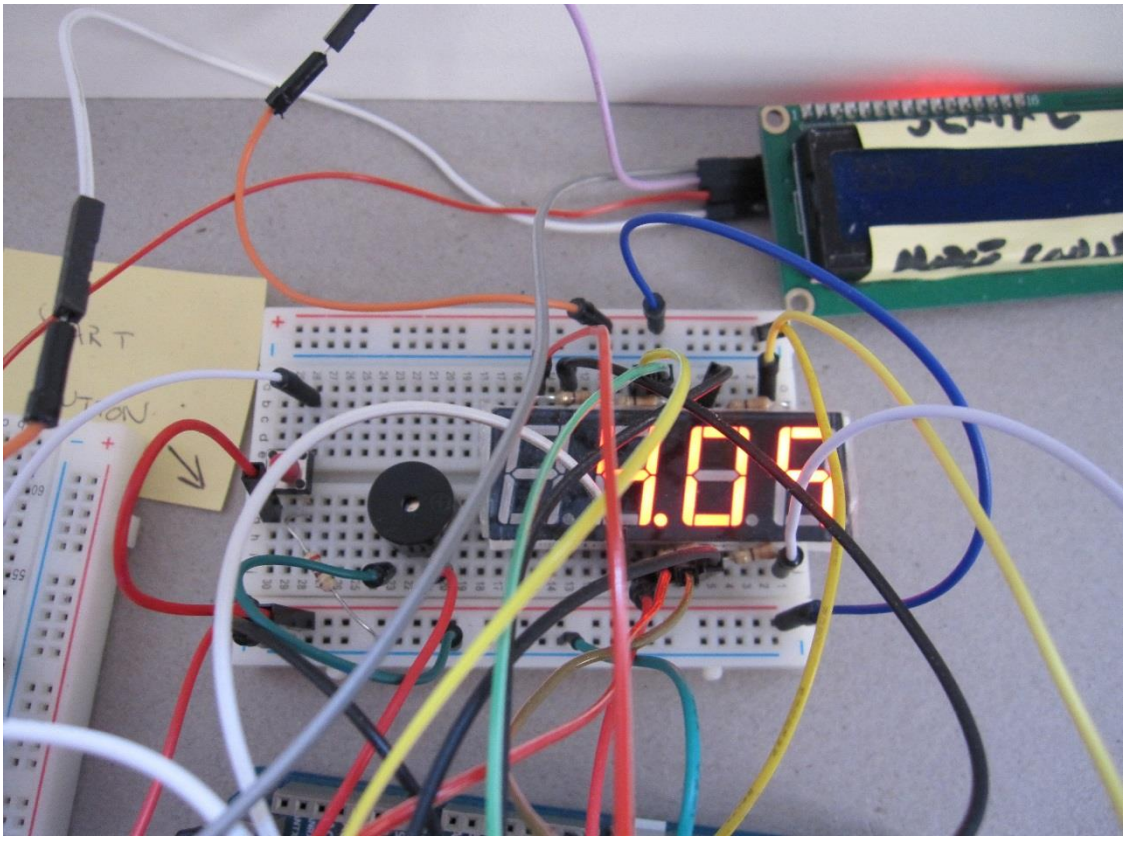


Figure 20 : Photo Timer