Dossier SAE4

Malo rupin

Zachary Khodja

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

Table des matières

Remerciements :

Un grand merci à Monsieur LUCIDARME pour nous avoir offert l'opportunité de travailler sur projet.

Merci à Abel COLET pour nous avoir prêté du matériel pour les soudures.

Introduction

Dans le cadre de notre projet de semestre 4, nous avons fait un test de faisabilité sur la création d'un échiquier électronique inspiré par le projet ChessUp de chess.com. En utilisant un magnétomètre pour la détection des pièces ainsi que des LEDs permettant un retour visuel à l'utilisateur.

Ainsi c'est dans l'objectif d'un jour concevoir un échiquier dans sa totalité que nous effectuons ce test de faisabilité, pour tester la faisabilité de détection de pièces grâce à un capteur à effet hall.

C'est donc sur ce test que porte ce dossier technique, nous vous détaillons les résultats en respectant ce plan :

D'abord par la présentation des objectifs et du cahier des charges, puis par l'exposition des différents choix technique que nous avons fait, suivis des différents livrables que nous avons rendus, et finalement nous ferons une phase de critique de nos résultats et conclurons par quelque informations importantes et idées pour le futur du projet.

1/ Cahier des charges & Objectifs

Notre objectif est donc de réaliser un test de faisabilité pour à terme créer un échiquier complet électronique.

Nous avons aussi pour objectif de facilité un maximum le travail des prochains groupes que ce soit avec des idées et schémas dans les documents, ou en écrivant le code du prototype en gardant en tête la finalité du projet.

Le cahier des charges de notre prototype consiste en :

- Avoir un retour visuel indiquant les coups possibles

- Obtenir une bonne détection des pièces (réussir à différencier facilement si une pièce est placée ou absente d'une case)

-Chainage des cartes électroniques

2/ Choix des composants et des technologies

2.1/ La carte Arduino Mega

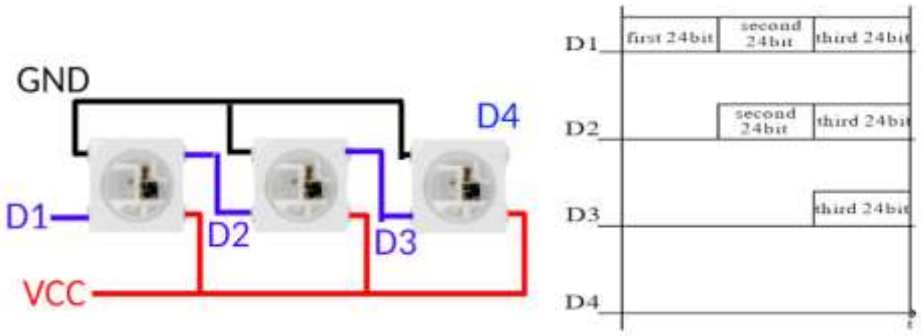
Pour le choix du microcontrôleur, nous avons choisi quelque chose de familier. En prenant une carte Arduino Mega, nous avons l'avantage des librairies pour contrôler les LEDs, pour communiquer en I^2C, et nous nous assurons d'avoir suffisamment de pin pour toutes les fonctionnalités que l'on pourrait imaginer. De plus cela nous permet de nous concentrer sur l'utilisation du capteur et des Leds et permettra de passer sur par exemple un ESP32 quand le projet sera plus avancé permettant ainsi de miniaturiser le tout.

2.2/ Les Leds

Au début de notre projet, les WS2812B nous ont été fortement conseillée. Nous avons naturellement fait le choix de les utiliser après nous être renseigné sur leur utilisation (lien vers la data-sheet : https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf).

Ce sont des LEDs RGB chainables, ce point est très important pour nôtre projet puisque nous avons besoin d'un minimum d'une LED par case soit 64 LEDs. Si chaque LEDs était contrôlée indépendamment, nous n'aurions pas pu toutes les connectées à l'ATMEMEGA.

Leur utilisation est plutôt simple, comme nous pouvons le voir sur le schéma si dessous, il suffit d'alimenter les LEDs sous 5 V, et de fournir un signal logique D1 depuis l'ATMEGA sur la première LED. Le autres LEDs sont chainées entre le Pin de sortie de la première vers le pin d'entré de la seconde. Chaque LED lit les 24 premiers bits de la trame qu'elle reçoit et transmet tout le reste aux suivantes.



Une libraire nommé FastLED permet de contrôler facilement les WS2812B (lien vers la librairie : http://fastled.io/docs/). Le code que nous avons dévloppé utlise principalement les outils de cette librairie.

2.3/ Le capteur

Pour ce qui est du choix du capteur, il y avait quelques contraintes à respecter :

-Communication en I^2C, pour faciliter son utilisation étant un protocole déjà vue en cours de plus il permet la communication avec plusieurs esclaves,
-8 adresses I^2C ou plus

C'est ce dernier point qui nous a posé le plus problème, en effet il nous fallait trouver un capteur avec au minimum 8 adresses différentes, pour nous permettre de constituer une ligne de l'échiquier.

Voici une liste des magnétomètres que nous avons étudié :

Magnétomètre	Compatible I^2C	Nombre d'adresses disponibles
HMC5883L	OUI	1
LIS2MDL	OUI	4
LIS3MDL	OUI	1
BMM150	OUI	2
A31301	OUI	16

Figure 1 : Tableau magnétomètres

C'est pour cela que nous avons décidé de prendre le capteur A31301, grâce a son grand nombre d'adresse il sera facile de faire un chainage et de créer une ligne entière.

Pour modifier l'adresse du capteur, cela peut ce faire via EEPROM mais nous avons décider de ne pas utiliser cette technologie car il y aurait un soucis lors du démarrage du système, sans adresses fixes comment communiquer à 8 ou plus appareils différents ? C'est ainsi que nous nous sommes tournés vers la fixation des adresses à l'aide de 2 ponts diviseurs de tension sur les entrées AD0 et AD1 (ou SA1/SA0) du capteur en s'aidant de ce tableau disponible dans la documentation du capteur.

Table 7: I²C Slave Address Decoding

Voltage on AD1, V_{A1} (× V_{CC_IO})	Voltage on AD0, V_{A0} (× V_{CC_IO})	4-Bit Code from ADR1 and ADR0 Voltages				Peripheral Address Bits							Slave Address
		E3	E2	E1	E0	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	1	1	0	0	0	0	0	96
	0.33	0	0	0	1	1	1	0	0	0	0	1	97
	0.67	0	0	1	0	1	1	0	0	0	1	0	98
	1	0	0	1	1	1	1	0	0	0	1	1	99
0.33	0	0	1	0	0	1	1	0	0	1	0	0	100
	0.33	0	1	0	1	1	1	0	0	1	0	1	101
	0.67	0	1	1	0	1	1	0	0	1	1	0	102
	1	0	1	1	1	1	1	0	0	1	1	1	103
0.67	0	1	0	0	0	1	1	0	1	0	0	0	104
	0.33	1	0	0	1	1	1	0	1	0	0	1	105
	0.67	1	0	1	0	1	1	0	1	0	1	0	106
	1	1	0	1	1	1	1	0	1	0	1	1	107
1	0	1	1	0	0	1	1	0	1	1	0	0	108
	0.33	1	1	0	1	1	1	0	1	1	0	1	109
	0.67	1	1	1	0	1	1	0	1	1	1	0	110
	1	1	1	1	1	I2C_SLV_ADDR							Programmed to 111 at the factory

Figure 2 : Tableau d'adresses I²C en décimale

3/ Livrables

3.1/ Matrices de LEDs

Nous avons créé plusieurs codes affins de tester la matrice de LED ci-dessous. Des exemples peuvent être trouvé depuis la librairie pour prendre la main.

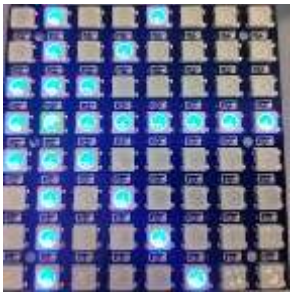


Figure 3 : Matrice de LED

Nous avons dû créer des fonctions plus spécifiques à partir de celles de la librairie FastLED. Ces nouvelles fonctions permettent de contrôler plus spécifiquement les LEDs. Le code de ces fonctions sont disponible en détail en annexe, voici ci-dessous la liste des fonctions les plus utiles :

Off_all_LED() permet d'éteindre toutes les LEDs

Case_exist(int casee) retourne 1 si la case est comprise dans le plateau d'échec, 0 sinon.

L'argument casee est une variable de type int ou la dizaine correspond aux lignes et l'unité aux colonnes.

Tower(int casee), qween(int casee), knight(int casee), bishop(int casee) et king(int casee) sont les fonctions permettant d'éclairer les cases sur laquelle une pièce peur aller en fonction de la case sur laquelle elle est grâce à l'argument casee.

Set_specific_LED(int case, long color) permet d'éclairer une case en choisissant sa couleur. Le code de sa couleur doit être écrit sous le format RGB avec 24 bits: 8 premiers pour la couleur bleu, les 8 suivants pour le vert et les 8 derniers pour le rouge (exemple FF0000 allume une LED en rouge).

Error_LED_IHM() est une fonction permettant de faire clignoté toutes les LEDs en rouge plusieurs fois consécutives. L'idée est de signalé une erreur à l'utilisateur. Elle pourrait être utilisé si on joueur se trompe de case par exemple.

2 mains présent en annexe permettent de tester ces fonctions et la matrice de LEDs, un premier permet de faire clignoter toutes les LEDs de la matrice.

La second main permet de testé les fonctions de déplacement des pièces, chaque fonction est testé sur chaque case consécutivement.

3.2/ Capteur

Pour la partie de la gestion du capteur, Mme G. DENECHEAU nous à conçu une carte test :

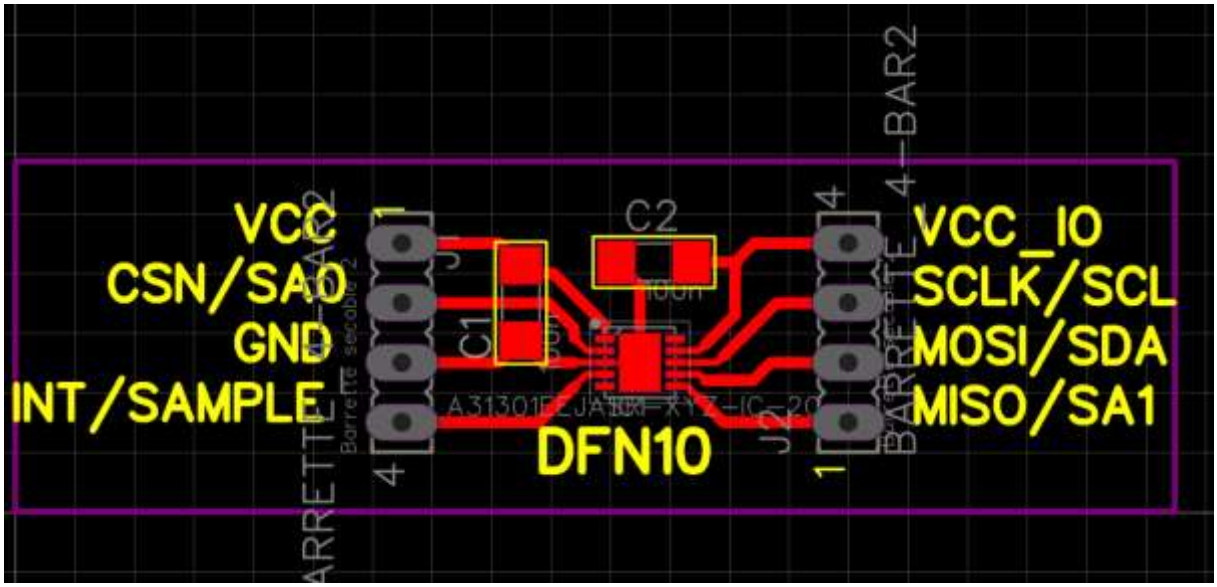


Figure 4 : Carte test S/o Denecheau

Ainsi à l'aide de ce capteur et de ce câblage sur une breadboard:

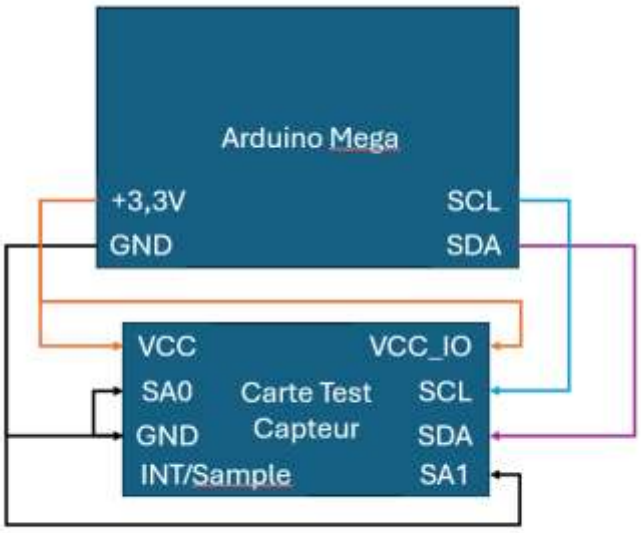


Figure 5 : Schéma câblage carte test

Pour communiquer avec le capteur il est nécessaire de connaitre son adresse, c'est pour cela que pour faciliter le test les entrées SA1/SA0 (=AD1/AD0)

Table 15: Output Registers

MSByte Address	LSByte Address	Register Name	MSByte								LSByte							
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	0x1D	TEMPERATURE	0	0	0	0	TEMPERATURE_12B											
0x1E	0x1F	X_CHANNEL	0	X_CHANNEL_15B														
0x20	0x21	Y_CHANNEL	0	Y_CHANNEL_15B														
0x22	0x23	Z_CHANNEL	0	Z_CHANNEL_15B														
0x24	0x25	ANGLE	0	ANGLE_15B														
0x26	0x27	RADIUS	0	RADIUS_15B														
0x28	0x29	SINE	0	SINE_15B														
0x2A	0x2B	COSINE	0	COSINE_15B														
0x2C	0x2D	X_RAW_FILT_OUT	0	X_RAW_FILT_OUT_15B														
0x2E	0x2F	Y_RAW_FILT_OUT	0	Y_RAW_FILT_OUT_15B														
0x30	0x31	Z_RAW_FILT_OUT	0	Z_RAW_FILT_OUT_15B														

Figure 6 : Tableau de registre

Nous nous concentrons sur l'axe Z étant l'axe qui sera le plus impacté par la présence ou non d'un aimant au-dessus du capteur.

Comme on peut le voir sur le tableau ci-dessus, les données sont sur 15 bit avec le 15e bit indiquant le signe +/- ainsi pour récupérer la valeur il faut réassembler le tout :

```
float moy_capt_Z(){
  double total=0;
  int N = 30;
  for (int i=0;i<N;i++){
    uint8_t msb=Request_info(ADDRESS_SLAVE,REGISTER_MSB_Z);// On récupére la valeur du MSB
    uint8_t lsb=Request_info(ADDRESS_SLAVE,REGISTER_LSB_Z);// On récupére la valeur du LSB

   // On assemble les 2 valeurs pour avoir une valeur sur 15bit
    int16_t combined = ((msb & 0x7F) << 8) | lsb;

    // On vérifie le bit de signe
    if (combined & 0x4000)
    {
        combined |= 0x8000; // On met le 16e bit a 1 pour correspondre
    }

    total+=(float)combined;
  }
  return total/(float)N;
}
```

Figure 7 : Code_Axe_Z

Avec :
ADDRESS_SLAVE= 0x60 (96 en décimale)
REGISTER_MSB_Z = 0x22
REGISTER_LSB_Z = 0x23

On en fait une moyenne pour lisser les valeurs obtenues réduisants les bruits et les perturbations externes.

En plaçcant un aimant au dessus du capteur on obtient des valeurs cohérentes, le capteur fonctionne donc bien.

La communication avec le capteur et la gestion des leds étant réussie, nous pouvons passé à l'étape suivant, le chainage de 2 de nos cartes.

3.3/ Case & chainage

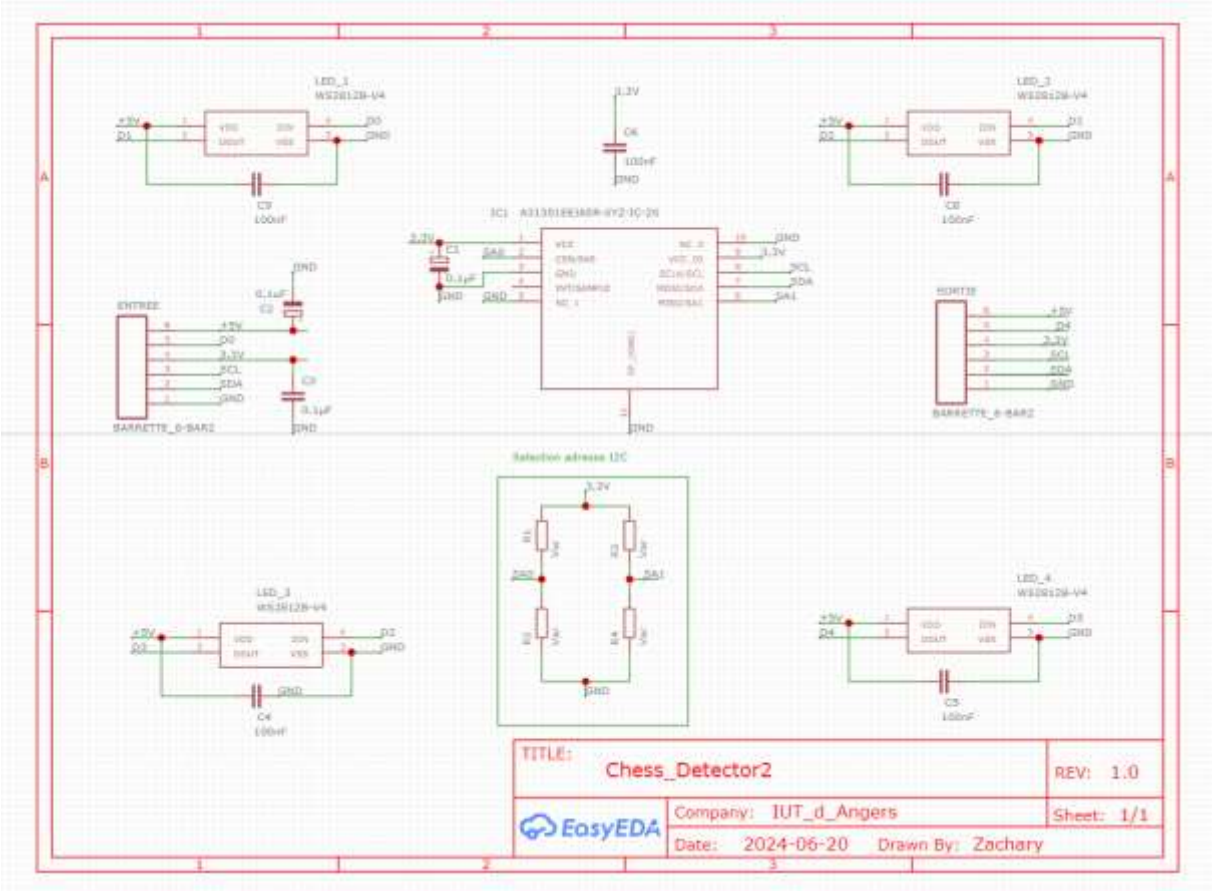


Figure 8: Schematic

Pour ce dernier livrable, nous avons placé 2 de nos cartes à suivre grâce aux borniers placé sur les extrémités des cartes partageant ainsi la même alimentation (+5V pour les LEDS et +3,3V pour le capteur), masse, SDA/SCL et DIN pour le contrôle des leds. Représenté par ce schéma :

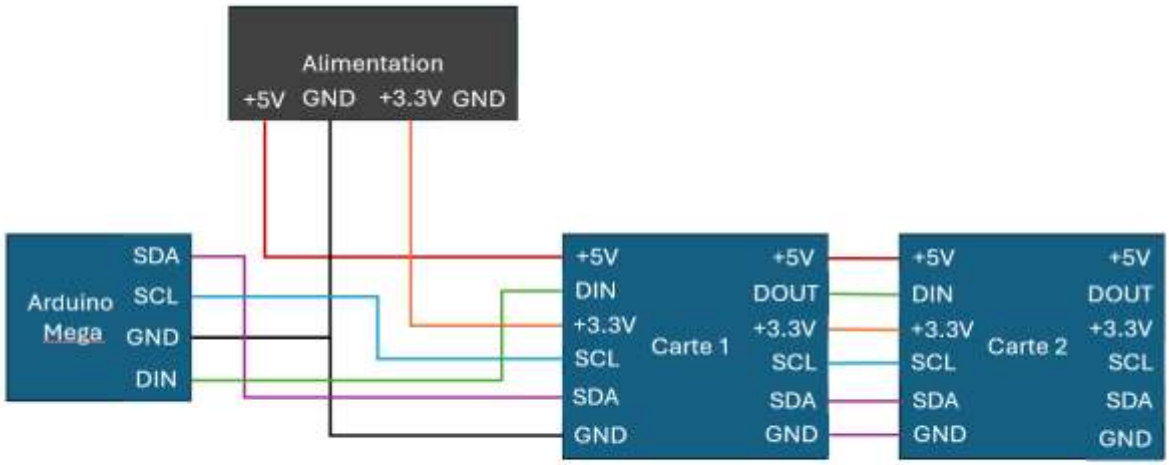
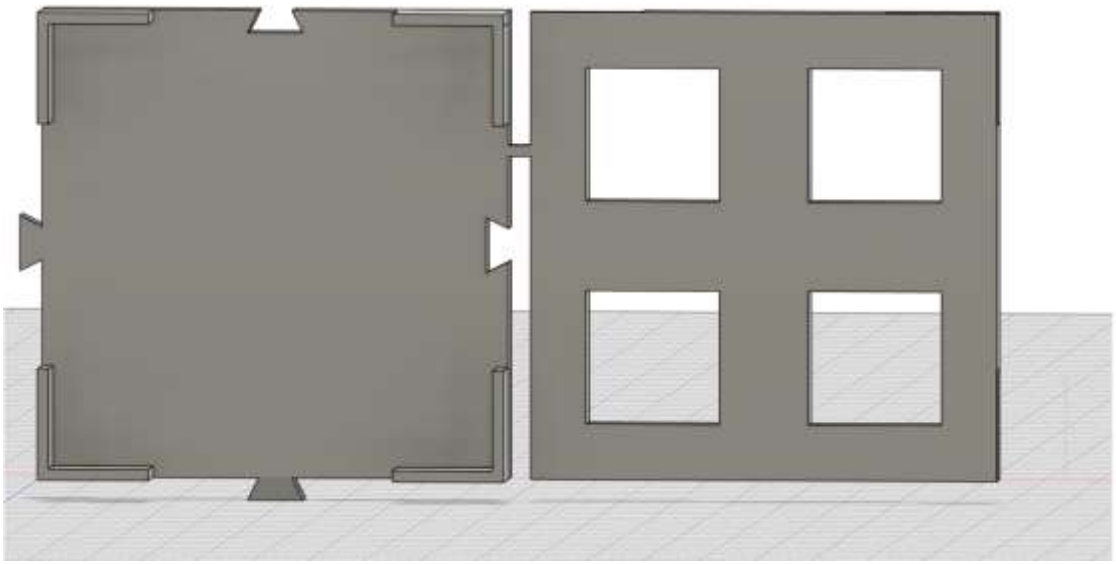


Figure 9 : Schéma de 2 cartes

En déplaçant un aimant au-dessus des cases on arrive à récupérer les valeurs du capteur de la carte 1 et de la carte 2 de façons distinctes, de plus il n'y a presque pas de bruit produit par l'aimant, si bien dimensionné, lorsqu'il est placé au-dessus de la carte. Pour ce qui est des leds cela fonctionne aussi très bien, le contrôle se fait facilement aussi.

De plus, pour aller plus loin dans l'idée du chainage, nous avons décidé de dimensionner une case d'échiquier dans lesquels nous pouvons placer une carte et mettre un toit emboîtable, ces cases peuvent s'emboiter entres-elles comme un puzzle permettant la création d'un échiquier complet.



4/ Critique

D’après notre prototype le projet est faisable, on a un bon contrôle des LEDS, la communication avec les différents capteurs se fait et il n’y a pas trop de pollution entre les capteurs.

Cependant, la soudure du capteur était un vrai souci, la taille du capteur ridiculement petit rends la tâche très compliquée et pour cette seule raison il faudrait peut-être réfléchir à utiliser un autre capteur, hormis ce fait le capteur est parfait.

De plus, les cartes électroniques peuvent encore être améliorée, par exemple en réfléchissant à un moyen de les chainer sans câbles.

Pour ce qui est des impression 3D, il faudra avoir une réflexion sur la propagation de la lumière provenant des LEDs afin d’avoir une bonne diffusion. Il sera aussi nécessaire d’avoir des cases de couleurs noirs et de couleur blanches.

Nous n’avons pas pris le temps de créer une librairie. Nous avons rapidement essayé mais avons rencontré des difficultés à cause des variables globales que nous utilisons depuis les fonctions. Il serait plus propre d’en créer une pour les LEDs et une pour la communication I^2C.

5/ Futur du projet

Pour la suite de ce projet nous avions réfléchis à plusieurs points,

On pourrait contrôler les capteurs par lots de 8 et donc par ligne de 8 cases via un multiplexeur de channels I^2C comme le PCA9848PWJ. De cette façon, nous pourrions utiliser 8 adresses différentes I^2C spécifiées en HARD.

Pour ce qui est de la propagation de la lumière voici quelques mesures que nous avons pu faire :

Proportion de Leds allumées	100%	50%	33%
Distance de la plaque (depuis le haut du pcb)	12mm	16/17 mm	20mm

Avant de refaire un nouveau routage, il serait intéressant de comparer la propagation de la lumière en fonction du nombre de LED, leur écartement, et leur luminosité. Il faudrait aussi vérifier la limite du nombre de LED chainable.

Une étude sur la consommation de courant d'une case serait aussi importante à réalisé afin de parfaire la prochaine case.

6/ Annexe

Loop pour tester la matrice de LEDs (penser à mettre `NUM_LEDS à 64`):

```
for (int i=0; i<NUM_LEDS;i++){
    leds[i] = red;
```

```
  }
  FastLED.show(); delay(100);

  for (int i=0; i<NUM_LEDS;i++){
    leds[i] = blue;
  }
  FastLED.show();
  delay(100);
  for (int i=0; i<NUM_LEDS;i++){
    leds[i] = green;
  }
  FastLED.show();
  delay(100);
```

Loop pour tester les déplacements des pièces(penser à mettre `NUM_LEDS à 64`) :

```
int choice=0;
  for (int m=0; m<5;m++){
    for (int l=1;l<9;l++){
     for (int c=1;c<9;c++){
       off_all_LED();
       if(m==0) tower(l*10+c);
       if (m==1) knight(l*10+c);
       if(m==2) qween(l*10+c);
       if(m==3) bishop(l*10+c);
       if(m==4) king(l*10+c);
        FastLED.show();
        delay(100);
      }
    }
  }
```

Loop pour tester la communication avec le capteur :

```
  get_all_sensor();
  Serial.print("X:");
  Serial.print(capt[0][0]/5.);
  Serial.print("\tY:");
  Serial.print(capt[0][1]/5.);
  Serial.print("\tZ:");
  Serial.print(capt[0][2]/5.);
  Serial.print("\tAngle:");
  Serial.println(angle(ADDRESS_CAPT1));
  delay(1000);
```

Loop pour tester les cases (penser à mettre NUM_LEDS à 8):

```
 get_all_sensor();
 off_all_LED();
  if(capt[0][2]>20){
    nRGB(4,255,capt[0][2]-20,capt[0][2]*1.2);
    nRGB(5,255,capt[0][2]-20,capt[0][2]*1.2);
    nRGB(6,255,capt[0][2]-20,capt[0][2]*1.2);
    nRGB(7,255,capt[0][2]-20,capt[0][2]*1.2);

  }
   if (capt[1][2]>20){
    nRGB(0,50,capt[1][2]-5,capt[1][2]*1.2);
    nRGB(1,50,capt[1][2]-250,capt[1][2]*1.2);
    nRGB(2,50,capt[1][2]-5,capt[1][2]*1.2);
    nRGB(3,50,capt[1][2]-250,capt[1][2]*1.2);

  }
  FastLED.show();
  delay(100);
```

La totalité du code (Loop vide, à remplacer par le code ci-dessus) :

```
#include <FastLED.h>
#include <Wire.h>
#define NUM_LEDS 8 //number of LED that are used
#define red 0xFF0000
#define green 0x00FF00
#define blue 0x0000FF
#define blank 0x000000

////////////////////////////////////variable used with the I²C
byte ADDRESS_SLAVE = 0x60 ;
byte ADDRESS_CAPT1 = 0x60 ;
byte ADDRESS_CAPT2 = 0x62 ;
byte REGISTER_MSB = 0x22;
byte REGISTER_LSB = 0x23;
byte REGISTER_MSB_X = 0x1E;
byte REGISTER_LSB_X = 0x1F;
byte REGISTER_MSB_Y = 0x20;
byte REGISTER_LSB_Y = 0x21;
byte REGISTER_MSB_Z = 0x22;
byte REGISTER_LSB_Z = 0x23;
byte REGISTER_MSB_ANGLE = 0x24;
byte REGISTER_LSB_ANGLE = 0x25;
```

```
byte READ_LENGTH = 8;
int MSB, LSB, Z;
int capt[64][3];
int capt_address[64]={0x60,0x62};
long int C,A,B;

//////////////////////////////////variable used with LEDs
int knight_calculus[8]={19,21,8,12,-19,-21,-8,-12};
int king_calculus[8]={10,11,1,9,-9,-1,-10,-11};
int
game[64]={"btower","bknight","bbishop","bking","bqween","bbishop2","bknight2",
"btower2"
              "bpawn1","bpawn2","bpawn3","bpawn4","bpawn5","bpawn6","bpawn7","
bpawn8"
              "none","none","none","none","none","none","none","none"
              "none","none","none","none","none","none","none","none"
              "none","none","none","none","none","none","none","none"
              "none","none","none","none","none","none","none","none""
              "wpawn1","wpawn2","wpawn3","wpawn4","wpawn5","wpawn6","wpawn7","
wpawn8"
              "wtower","wknight","wbishop","wking","wqween","wbishop2","wknigh
t2","wtower2"};//stock the actual position of all pieces

int turn="w";//to know who is playing
int wpawn[8]={1,1,1,1,1,1,1,1};//to know if whit pawn can moove 2 square
int bpawn[8]={1,1,1,1,1,1,1,1};//to know if black pawn can moove 2 square
CRGB leds[NUM_LEDS];



void setup() {
  FastLED.addLeds<NEOPIXEL, 7>(leds, NUM_LEDS);
  FastLED.setBrightness(5 );//change ;luminosity of the LED (5 is usefull for
testing, 255 is max value )
  Wire.begin();
 Wire.setClock(9600); // set I2C 'full-speed'
 Serial.begin(9600);
  }
void loop() {
  ////////////////////////////////////////////insert loop
here//////////////////////////////////////////
}

int transforme(int n){
  //Do:translate column and ligne to the refering LED
  //int n: la dizaine représente la ligne, l'unité la colone
  //return: the number of the LED

  n-=10;
```

```
  n-=2*(n/10);
  n-=1;
  return n;

}

void set_specific_LED(int n, long color){
  //Do:power a specific LED withe the choosed color
  //int n: la dizaine représente la ligne, l'unité la colone (23 représente
ligne 2 colobe 3)
  //int color:color of the LED set with 24 bits, package of 8 are used to code
1 color. The order is Red Green Blue.
  //return: nothing


  //hard math happening here
  n=transforme(n);

 //set the specific LED
  leds[n] = color;

}



void off_all_LED(void){
  //Do:unpower all LEDs
  //return: nothing
  for (int i=0; i<NUM_LEDS;i++){
    leds[i] = blank;
  }

}
int case_exist(int casee){
  //Do:usefull to know if the case is in the rage of a chess game (8 ligne and
8 column)
  //int casee: la dizaine représente la ligne, l'unité la colone (23
représente ligne 2 colobe 3)
  //return: return 1 if case exist, return 0 if she doesn't
  if(((casee%10)>0) && ((casee%10)<9) && ((casee/10)<9) && ((casee/10)>0))
return 1;
  else return 0;
}

void tower(int casee){
  //Do:show everywhere the tower can go
  //int casee: la dizaine représente la ligne, l'unité la colone (23
représente ligne 2 colobe 3)
  //return: return 1 if case exist, return 0 if she doesn't
```

```
  if (case_exist(casee)) set_specific_LED(casee,green);//show green where the
piece is
  int colone=1;//use to find the nexte case
  int ligne =10;
  while (case_exist(casee+colone)){//check if there is case next to the piece,
if there is, show blue, and look at for the next one
    if (case_exist(casee+colone)) set_specific_LED(casee+colone,blue);
    colone+=1;
  }
  colone=1;
while (case_exist(casee-colone)){
    if (case_exist(casee-colone)) set_specific_LED(casee-colone,blue);
    colone+=1;
  }
while (case_exist(casee+ligne)){
    if (case_exist(casee+ligne)) set_specific_LED(casee+ligne,blue);
    ligne+=10;
  }
  ligne=10;
while (case_exist(casee-ligne)){
    if (case_exist(casee-ligne)) set_specific_LED(casee-ligne,blue);
    ligne+=10;
  }


}

void bishop(int casee){
  //Do:show everywhere the bishop can go
  //int casee: la dizaine représente la ligne, l'unité la colone (23
représente ligne 2 colobe 3)
  //return: return 1 if case exist, return 0 if she doesn't
  if (case_exist(casee)) set_specific_LED(casee,green);//show green where the
piece is
  int checked_casee=casee+11;
  while (case_exist(checked_casee)){
    set_specific_LED(checked_casee,blue);
    checked_casee+=11;
  }
   checked_casee=casee-11;
  while (case_exist(checked_casee)){
    set_specific_LED(checked_casee,blue);
    checked_casee-=11;
  }
   checked_casee=casee+9;
  while (case_exist(checked_casee)){
    set_specific_LED(checked_casee,blue);
    checked_casee+=9;
```

```
  }
   checked_casee=casee-9;
  while (case_exist(checked_casee)){
    set_specific_LED(checked_casee,blue);
    checked_casee-=9;
  }

}

void qween (int casee){
  //Do:show everywhere the qween can go
  //int casee: la dizaine représente la ligne, l'unité la colone (23
représente ligne 2 colobe 3)
  //return: return 1 if case exist, return 0 if she doesn't
  bishop(casee);
  tower(casee);
}

void knight(int casee){
  //Do:show everywhere the knight can go
  //int casee: la dizaine représente la ligne, l'unité la colone (23
représente ligne 2 colobe 3)
  //return: return 1 if case exist, return 0 if she doesn't
  if (case_exist(casee)) set_specific_LED(casee,green);//show green where the
piece is
  for (int i=0; i<8;i++){//check every possible case which a knight can go
    if(case_exist(casee+knight_calculus[i]))
set_specific_LED(casee+knight_calculus[i],blue);//show blue where the knight
can go
  }
}

void king (int casee){
  //Do:show everywhere the king can go
  //int casee: la dizaine représente la ligne, l'unité la colone (23
représente ligne 2 colobe 3)
  //return: return 1 if case exist, return 0 if she doesn't
  if (case_exist(casee)) set_specific_LED(casee,green);//show green where the
piece is
  for (int i=0; i<8;i++){//check every possible case which a knight can go
    if(case_exist(casee+king_calculus[i]))
set_specific_LED(casee+king_calculus[i],blue);//show blue where the king can
go
  }


}
void error_LED_IHM(void){
  //Do:a bit of IHM if somone does somthing dumb
```

```
  //return: return 1 if case exist, return 0 if she doesn't


  for(int k=0;k<5;k++){//every LED will blink red k times
    for (int i=0; i<NUM_LEDS;i++){ //apply red everywhere
      leds[i] = red;
    }

    FastLED.show();//show red

    delay(100);//wait a bit

     for (int j=0; j<NUM_LEDS;j++){ //apply blank everywhere
      leds[j] = 0x000000;
     }
    FastLED.show();//show litteraly nothing
    delay(100);//wait a bit
  }



}



int Request_info (int Slave,int registre){
   //Do:request something to a slave with I²C
   //int Slave: adrresse of the slave
   //int registre: register of the information that we want to get
   //return:the information that we want

   int data;
   int octet=1;
   Wire.beginTransmission(Slave);
   Wire.write(registre);  // set register for read
   Wire.endTransmission(false); // false to not release the line

   Wire.requestFrom(Slave,octet); // request bytes from register XY
   byte buff[octet];
   Wire.readBytes(buff, octet);


  return buff[0];
   }

float moy_capt_Z(byte ADDRESS){
   //Do:request the Z value of the magnetic field to a specific slave with I²C
```

```
   //int ADDRESS: adresse of the slave
   //return:the Z value of the magnetic field
  double total=0;
  int N = 30;
  for (int i=0;i<N;i++){
    uint8_t msb=Request_info(ADDRESS,REGISTER_MSB_Z);
    uint8_t lsb=Request_info(ADDRESS,REGISTER_LSB_Z);

   // Combine the two bytes into a 15-bit value
    int16_t combined = ((msb & 0x7F) << 8) | lsb;

    // Check if the sign bit (the 15th bit) is set
    if (combined & 0x4000) {
        // If the sign bit is set, extend the sign to the 16th bit
        combined |= 0x8000; // Set the 16th bit for correct 16-bit signed
representation
    }

    total+=(float)combined;
  }
  return total/(float)N;
}

float moy_capt_X(byte ADDRESS){
    //Do:request the X value of the magnetic field to a specific slave with
I²C
   //int ADDRESS: adresse of the slave
   //return:the X value of the magnetic field
  double total=0;
  int N = 30;
  for (int i=0;i<N;i++){
    uint8_t msb=Request_info(ADDRESS,REGISTER_MSB_X);
    uint8_t lsb=Request_info(ADDRESS,REGISTER_LSB_X);

   // Combine the two bytes into a 15-bit value
    int16_t combined = ((msb & 0x7F) << 8) | lsb;

    // Check if the sign bit (the 15th bit) is set
    if (combined & 0x4000) {
        // If the sign bit is set, extend the sign to the 16th bit
        combined |= 0x8000; // Set the 16th bit for correct 16-bit signed
representation
    }

    total+=(float)combined;
  }
  return total/(float)N;
}
```

```
float moy_capt_Y(byte ADDRESS){
    //Do:request the Y value of the magnetic field to a specific slave with
I²C
   //int ADDRESS: adrresse of the slave
   //return:the Y value of the magnetic field
  double total=0;
  int N = 30;
  for (int i=0;i<N;i++){
    uint8_t msb=Request_info(ADDRESS,REGISTER_MSB_Y);
    uint8_t lsb=Request_info(ADDRESS,REGISTER_LSB_Y);

   // Combine the two bytes into a 15-bit value
    int16_t combined = ((msb & 0x7F) << 8) | lsb;

    // Check if the sign bit (the 15th bit) is set
    if (combined & 0x4000) {
        // If the sign bit is set, extend the sign to the 16th bit
        combined |= 0x8000; // Set the 16th bit for correct 16-bit signed
representation
    }

    total+=(float)combined;
  }
  return total/(float)N;
}



float angle(byte ADDRESS){
    //Do:request the angle of the magnetic field to a specific slave with I²C
   //int ADDRESS: adrresse of the slave
   //return:the angle of the magnetic field
  double total=0;
  int N = 30;
  for (int i=0;i<N;i++){
    uint8_t msb=Request_info(ADDRESS,REGISTER_MSB_ANGLE);
    uint8_t lsb=Request_info(ADDRESS,REGISTER_LSB_ANGLE);

    // Combine the two bytes into a 15-bit value
    uint16_t combined = ((msb) << 8) | lsb;

    total+=(float)combined;
  }
  return  0.0109863*total/(float)N;
}

void nRGB(int n,long int R ,long int G, long int B){
    //Do:set color of a specific LED, value above 255 are set to 255, value
bellow 0 are set to 0
```

```
   //int n: value of the LED
   //int R,G,B: value of the color of R G and B
   //return:nothing
  if (R<0) R=0;
  if (R>255) R= 255;
   if (G<0) G=0;
  if (G>255) G= 255;
   if (B<0) B=0;
  if (B>255) B= 255;
  Serial.print("R:" );
  Serial.print(R );
  Serial.print("G:" );
  Serial.print(G );
  Serial.print("B:" );
  Serial.println(B );

  leds[n].setRGB(R,G,B);

}

void get_all_sensor(){
    //Do:request the Z, X and Y value of the magnetic field to all slave with
I²C, save them in capt
   //return:nothing
  for(int i=0;i<2;i++){
      capt[i][0]=moy_capt_X(capt_address[i]);
      capt[i][1]=moy_capt_Y(capt_address[i]);
      capt[i][2]=moy_capt_Z(capt_address[i]);
  }
}
0
```