POLYTECH LILLE

Rapport de projet :

Matrice de LED 3D

IMA4 2012-2013

Matthias De Bie - Pierre-Jean Petitprez

Table des matières

Merci à M. Boé et M. Vantroys, nos tuteurs de projet, pour l'aide qu'ils nous ont apportée.

Merci à M. Flamen pour sont aide inestimable et les nombreux conseils qu'il a pu nous offrir. Sans lui nous n'aurions ni matrice de LED ni carte électronique digne de ce nom.

Merci aux élèves IMA qui nous ont soutenus et qui ont approuvé notre projet.

Merci également aux enseignants Polytech'Lille qui pour certains ont manifesté de l'intérêt pour notre projet.

Dans le cadre du projet de fin de 4ème année nous avons décidé de réaliser une matrice de LED en 3 dimensions.

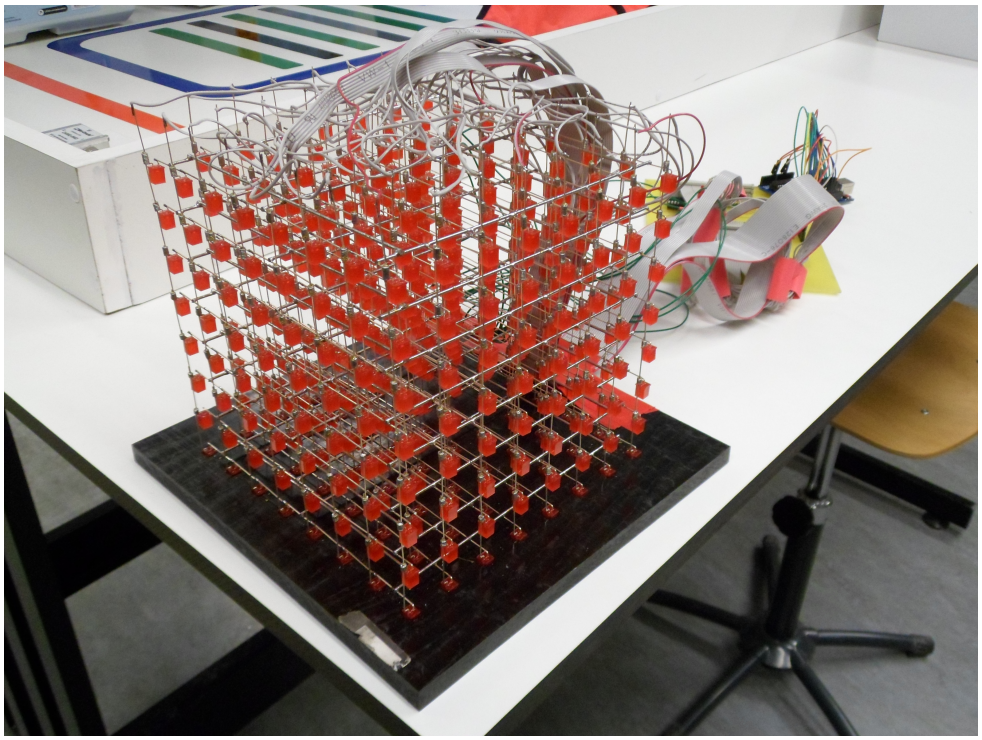

FIGURE 1 – Le cube de 512 LED

Ce projet permettra d'habiller les représentations officielles de Polytech'Lille et du département IMA, telles que la journée portes ouvertes. Le contexte technique est le suivant, nous disposions en début de projet d'un ensemble de LED rouges non montées, d'un Arduino (Uno ou Méga), du logiciel Altium Designer 2010 et de toutes les machines nécessaire à la fabrication de cartes électroniques. Sur une période de 15 semaines, soit environ 70 heures encadrées auxquelles il faut ajouter le temps passé en dehors des créneaux encadrés, nous avons donc proposé les solutions techniques, réalisé des prototypes, conçu les programmes et les cartes électroniques, schématisé l'implantation du cube de LED, assemblé l'ensemble des composants et réalisé des tests « fonctionnels ». La seule partie dont nous ne nous sommes pas occupés est le montage à proprement parler de la matrice de 512 LED.
Dans ce rapport, nous revenons sur l'élaboration de ce projet. Nous discuterons sur les solutions envisagées, tant matérielles que logicielles, puis nous présenterons celles que nous avons utilisées.

1 Solutions techniques

1.1 Partie matérielle

Le travail sur la partie matérielle consistait en la conception de différentes cartes. Voici une représentation simplifiée du montage complet prévu :

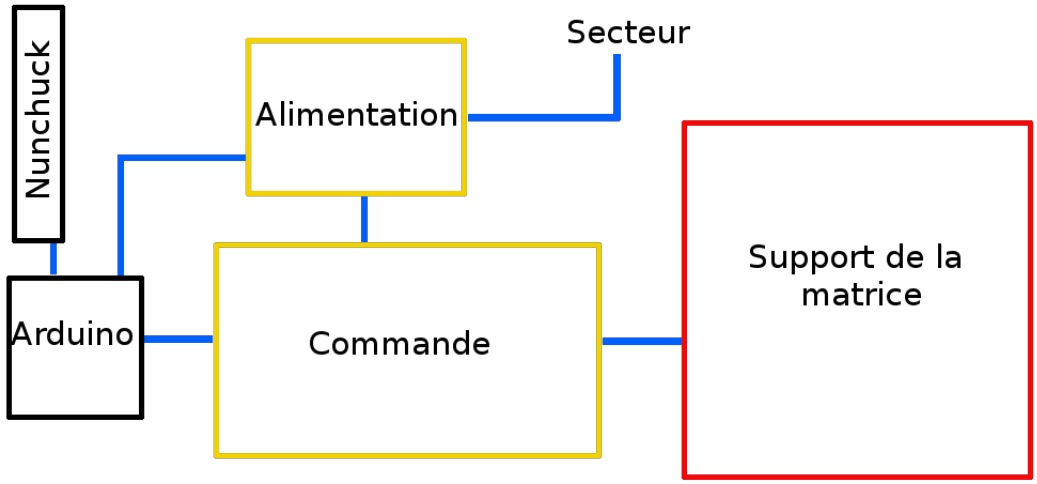

FIGURE 1.1.1 – Schéma simplifié du montage prévu

1.1.1 Carte de commande

La première carte est une carte capable de commander les 512 LED grâce à un Arduino, sachant qu'un Arduino Uno ne possède que 19 sorties dont 13 numériques. Plusieurs solutions nous ont semblé viables, voici lesquelles :

- La première solution a été d'utiliser uniquement des décodeurs 3 vers 8 pour la sélection des étages et 4 vers 16 pour la sélection des LED. Malheureusement nous ne disposions plus assez de broches libres pour brancher un adaptateur Nunchuck sur Arduino Uno.
- La deuxième solution a été d'utiliser des circuits intégrés contenant 8 bascules D. toutes les entrées des bascules étaient reliées par un bus de données commun (8 bits de données) et les bascules mémorisaient un mot présent sur le bus lors d'un front d'horloge qui lui était propre. Cette solution faisait appel à 19 broches au moins et étant donné qu'il fallait brancher un adaptateur Nunchuck en plus sur les broches analogiques il nous manquait de l'espace.
- La dernière est une solution originale qui utilise 8 registres à décalages couplés à 8 bascules D de 8 bits. Les registres sont à chargement série et déchargement parallèle, le principe est donc de charger les 8 registres avec 8 bits de manière série pour ensuite les stocker et les afficher, pendant le temps d'affichage les autres séries de 8 bits sont injectées dans les registres. Cette solution utilise au maximum 13 broches de l'Arduino et au minimum 6 broches car les registres à décalages ont une sortie série qui permet de les assembler ensemble pour former un seul registre à décalage de 64 bits.

Pour ces trois solutions la sélection d'un étage de LED est toujours réalisée par un démultiplexeur 3 vers 8 qui permet de relier toutes les cathodes d'un étage à la masse via des transistors.

Nous nous somme donc basés sur le système qui utilise des registres à décalage car il permet d'utiliser la vitesse d'exécution de l'Arduino à défaut du nombre de sorties qui est vraiment limité. En tout il nous faut donc 13 sorties pour le contrôle de la matrice : 8 pour les entrées séries, 2 pour les signaux d'enable des registres et 3 pour le multiplexage des couches. L'alimentation des composants de commande (ceux qui n'alimentent pas directement les LED) est directement desservie par l'Arduino. Le typon de cette carte est disponible en annexe A.0.1.

Choix des composants :

- (x8) 74HCT595 – 8-bit Serial in/Parallel out Shift Register
- (x8) UDN2981 – 8-Channel Transistor Array Darlington Type Source
- (x1) 74HCT238 – 3-to-8 line Decoder
- (x1) ULN2803 - 8-Channel Transistor Array Darlington Type Sink

1.1.2 Carte de Nunchuck

FIGURE 1.1.2 – La carte Wiichuck telle que présentée chez Sparkfun

La deuxième carte est une simple carte d'adaptation du connecteur Nunchuck de manière à la brancher sur les pattes analogiques de l'Arduino comme un shield Arduino classique. Il se trouve que cette carte était déjà disponible dans les stocks de l'école car elle existe dans le commerce, vendue sous le nom de "Wiichuck", néanmoins nous avons tout de même créé un typon pour pouvoir tirer une autre carte de notre propre fabrication.

Cette carte est en réalité très simple, car elle ne fait que relier les pistes utilisées de la prise femelle du Nunchuck avec des broches permettant de se connecter sur un Arduino.

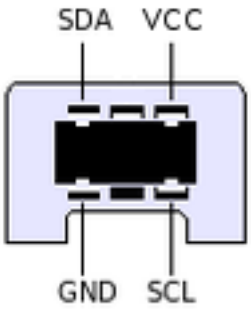

FIGURE 1.1.3 – Le connecteur femelle du Nunchuck

1.1.3 Carte d'alimentation

La 3ème carte est prévue pour alimenter l'ensemble du système à partir d'une seule prise électrique, de cette manière il est plus facile de déplacer l'ensemble (plutôt que de prévoir un bloc d'alimentation de test pour chaque branchement occasionnel). Ce bloc est assez simple et il permet de disposer de deux tensions, 12V et 5V respectivement pour l'Arduino et pour l'alimentation des LED. L'alimentation des LED doit pouvoir assurer un courant de 1A environ lorsque le maximum des LED est allumé. Le schéma de cette carte est basé sur un régulateur L7805CV qui permet de transformer une tension 12V en 5V en maintenant une tension stable et permettant un courant de 1,5A.

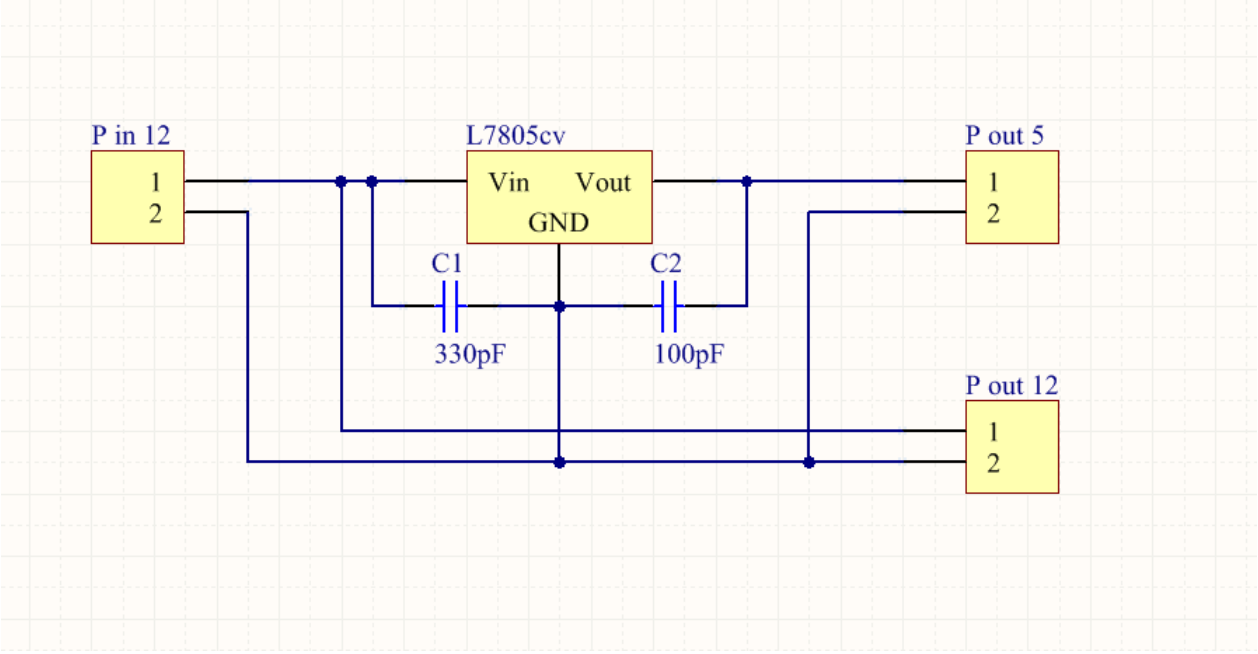

FIGURE 1.1.4 – Schéma de la carte d'alimentation

En raison de manque de temps et de contraintes techniques, cette carte n'a pu être réalisée. Pour les tests, nous avons utilisé une alimentation 5V séparée du bloc d'alimentation de l'Arduino.

1.1.4 Carte de test

Pour pouvoir tester et surtout vérifier le bon fonctionnement de notre système nous avons eu besoin de créer une carte temporaire qui permet d'adapter un circuit intégré de type CMS (**Composant Monté en Surface**) en un circuit DIP (**Dual Inline Package** - composant traversant).

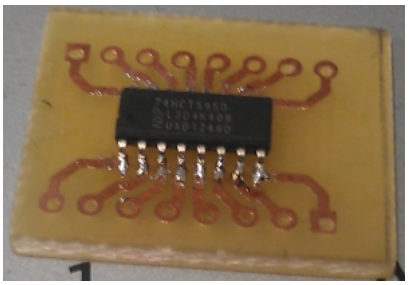

FIGURE 1.1.5 – Carte d'adaptation de CMS

1.2 Partie logicielle

Du côté logiciel, il nous faut programmer l'Arduino de sorte qu'il puisse commander la matrice, mais également gérer une partie de Snake. La majorité du code est disponible en annexe, et la totalité des programmes est également disponible sur la page wiki su projet (`https://projets-ima.polytech-lille.net:40079/mediawiki/index.php?title=Matrice_de_LED_3D`).

1.2.1 Description du programme

Le but final est de jouer à un jeu de type « Snake », mais à la différence des Snake habituels, celui-ci se joue en 3D. Pour ce faire, le programme Arduino se découpe en plusieurs fichiers (que l'on peut assimiler à des bibliothèques) afin de rendre le programme modulaire : un fichier principal contenant les fonctions indispensables pour l'Arduino que sont setup() exécutée une fois au démarrage et loop() qui tourne en boucle, un fichier pour les fonctions de gestion du Nunchuck, un fichier pour les fonctions de gestion de l'affichage et enfin un fichier pour les fonctions de l'algorithme du Snake. Ainsi il suffit de modifier un seul fichier pour adapter le programme à d'autres supports.

1.2.2 Choix de l'Arduino

Dès le début du projet, nous avons pu remarquer que le nombre de sorties et la RAM d'un Arduino Uno seraient les difficultés à contourner. Un moyen simple de les contourner aurait été d'utiliser un Arduino Mega, qui offre une cinquantaine de sorties numériques et une mémoire RAM disponible de 8 ko. Cependant, utiliser un Arduino Mega semblait disproportionné au vu du projet. C'est pourquoi nous nous sommes tournés vers un Arduino Uno, qui offre beaucoup moins de sorties et de mémoire, mais qui semblait plus approprié pour ce projet. De plus, optimiser le programme ainsi que les connexions matérielles dans le but d'utiliser la version Uno fut une très bonne expérience.

1.2.3 Choix des tableaux

L'utilisation de tableaux est indispensable car il faut un moyen de stocker les valeurs que vont prendre les LED de la matrice, en l'occurrence "0" ou "1" symbolisant l'état éteint ou allumé. Utiliser un seul tableau global nous a vite semblé impossible, car il fallait avoir la possibilité de réaliser des opérations sur le serpent. C'est pourquoi le serpent est représenté par un vecteur de 512 cases (la taille maximale qu'il peut atteindre correspond au nombre de LED disponibles). Dans un premier temps nous pensions stocker pour chaque case ses coordonnées sur la matrice (en mode [X][Y][Z], mais cela revenait à créer un tableau de 512 lignes ayant chacune 3 colonnes, soit un total de 1536 octets et cela rendait plus difficile l'algorithme. Au final nous avons opté pour un vecteur de 512 cases contenant chacune un nombre compris entre 0 et 511 qui correspond au numéro de la LED. Ceci rend très simple l'algorithme, car ainsi on ne manipule qu'une seule case.

Ensuite, notre premier algorithme utilisait 3 tableaux dont 2 uniquement pour le Snake, l'un pour l'état actuel du Snake et un deuxième qui reçoit l'état suivant, puis celui-ci était recopié dans le premier. Ceci n'était évidemment pas très optimisé, c'est pourquoi le deuxième tableau a très vite été supprimé, et les calculs pour définir l'état suivant se font directement sur le seul tableau restant, en commençant par la fin pour éviter les écrasements des valeurs. La suppression d'un tableau a simplement obligé à ajouter deux variables globales, une pour symboliser la tête du serpent et une autre contenant la taille du serpent.
Le deuxième tableau est utilisé pour stocker les états de chaque LED (allumée ou éteinte) pour ensuite envoyer ces données à la carte de commande. Celui-ci possède 64 octets, en effet une LED est codée sur un bit, et 64*8=512.

Voici une représentation schématique de la manière donc la matrice et le serpent sont gérés, pour l'exemple nous prenons uniquement la première couche de la matrice (figure 1.2.1).

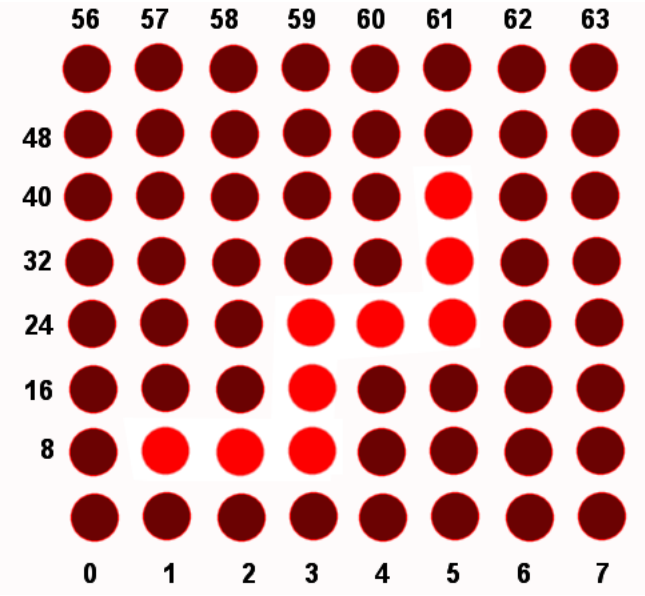

FIGURE 1.2.1 – Exemple avec la première couche de 64 LED

Les disques rouges représentent les LED allumées par le serpent, les disques rouge foncé représentent les LED éteintes.

Chaque LED possède un numéro, compris entre 0 et 511 pour la matrice 3D, de 0 à 63 pour la première couche. Si la LED 9 est la tête du serpent, alors le tableau représentant celui-ci contiendra ces valeurs (exprimées en décimal) :

[9][10][11][19][27][28][29][37][45]

Une fois la mise à jour effectuée, le tableau contenant les états des LED vaudra donc, exprimé en binaire, et en considérant que le bit de poids faible est à gauche :

[00000000][01110000][00010000][00011100][00000100][00000000][00000000]...

1.2.4 Gestion de la mémoire

La première version du code avait été développée sans prendre en compte la limitation en mémoire RAM de l'Arduino. En effet, en utilisant deux tableaux de 1024 octets chacun (le premier représentant le serpent à l'étape n et le second à l'étape n+1) et un troisième tableau de 512 octets pour la matrice (dans lequel un octet repésentait une LED), on dépassait très largement les 2 ko de mémoire disponible. C'est pourquoi le code a été repensé afin d'optimiser au mieux l'utilisation des tableaux et des variables. Le vecteur représentant le serpent doit pouvoir contenir des valeurs comprises entre 0 et 511, or un octet est trop petit pour coder les valeurs supérieures à 255. Chaque entier est donc codé sur deux octets, soit un total de 1024 octets pour ce tableau.

Sachant que la RAM de l'Arduino Uno est de 2 ko, et qu'il est préférable de ne pas dépasser environ 1,5 ko sous peine de voir apparaître des problèmes de gestion des variables et de pile, il ne reste au maximum que 512 octets disponibles pour stocker le tableau qui représente la matrice et également les nombreuses autres variables qui prennent une place non négligeable.

Etant donné la place perdue en utilisant des octets complets pour stocker l'état des LED, nous sommes partis dans une autre direction : associer un bit à une LED, ce qui ramène la taille du tableau associé à la matrice à 64 octets. Ceci a rendu le code légèrement plus complexe en utilisant des masques et autres opérations bit à bit au lieu d'une simple affectation par indice, mais cela a l'avantage d'être très compact et optimisé.

1.2.5 Gestion du Nunchuck

Ce qui rend ce Snake différent mis à part son affichage 3D, c'est aussi l'utilisation du Nunchuck de Nintendo pour le contrôler. Le joystick analogique permet de déplacer le serpent dans le plan horizontal, alors que l'accéléromètre permet de déplacer le serpent verticalement. Ce Nunchuck dialogue avec l'Arduino grâce au protocole I2C (utilisation de la bibliothèque Wire.h intégrée nativement dans l'Arduino IDE).

Dans un premier temps nous pensions également contrôler le Snake avec une Wiimote connectée

en Bluetooth. Cependant la connexion requise, Bluetooth HID, n'est pas supportée nativement par Arduino. La solution aurait été de prendre un shield Bluetooth externe mais cela compliquait le montage. Nous nous sommes donc limités à l'utilisation d'un Nunchuck connecté en filaire. L'utilisation du connecteur appelé « Wiichuck » (voir 1.1.2) dans le commerce permet de connecter facilement un Nunchuck sur un Arduino en reliant les 4 pistes utilisées sur le connecteur du Nunchuck sur les broches analogiques A0 à A3 de l'Arduino afin de satisfaire aux exigences du protocole I2C. Ceci oblige donc à configurer ces broches de manière à ce qu'elles alimentent le Nunchuck : il suffit de rediriger l'alimentation et la masse sur les broches A3 et A2, les deux autres sont utilisées pour les données et l'horloge.

1.2.6 Gestion de l'affichage

Afin de rendre l'affichage totalement indépendant du reste du programme, nous utilisons une interruption temporelle qui exécute la fonction de mise à jour de la matrice. Pour ce faire, nous utilisons la bibliothèque MsTimer2 (`http://playground.arduino.cc/Main/MsTimer2`) qui permet d'accéder facilement au timer nommé Timer2 et aux interruptions liées à ce timer. Ainsi nous pouvons fixer la période d'appel à cette fonction qui est la seule à modifier directement les sorties de l'Arduino, et donc la commande de la matrice.

Cette fonction gère les registres à décalage et la sélection des couches de la matrice.

1.2.7 Gestion du serpent

Au niveau de la gestion du serpent, le programme se découpe en plusieurs phases : d'abord on calcule la nouvelle position de la tête du serpent en fonction de la direction ordonnée par le Nunchuck, puis on vérifie si cette nouvelle position correspond à une pomme, dans ce cas elle est indiquée comme mangée et une nouvelle pomme est introduite. Ensuite on vérifie si la tête du serpent entre en collision avec la queue, ce qui mettra fin au programme et relance une nouvelle partie. Enfin, en cas de non-collision, le serpent est mis à jour dans sa globalité.

Les interruptions n'autorisent pas l'utilisation de fonctions de pause telles que "delay()". De plus, le timer utilisé pour l'affichage n'offre qu'une seule interruption. La fonction faisant avancer le serpent devant elle aussi être exécutée périodiquement (la période dépendant de la vitesse du serpent), nous avons choisi de tester à chaque tour si le nombre de millisecondes écoulées depuis la dernière exécution est supérieur à la période voulue. Ceci n'est pas très efficace algorithmiquement, il faudrait utiliser un second timer pour cette fonction. Cependant, il faut absolument que les valeurs provenant du Nunchuck soient complètes avant d'exécuter la fonction de déplacement du serpent. Utiliser une interruption pourrait alors poser problème, à moins d'implémenter des sécurités telles que des sémaphores, qui rendraient alors le code plus complexe.

2 Conception assistée par ordinateur

La conception est la partie inévitable et nécessaire qui prime sur toutes les autres puisque sans elle la création d'un circuit sur epoxy n'est pas possible. Le but de la conception est de produire un modèle imprimé de la carte, avec les pistes, les trous, les vias, les annotations ... Pour ce faire il est utile de créer un schématique qui introduit les différents composants et leurs routages sur un schéma, ce schéma n'est pas utilisable tel quel mais il permet par la suite d'avoir une aide visuelle pour placer les pistes, aussi appelée "chevelu".

Nous avons décidé d'utiliser le logiciel Altium pour réaliser notre carte car il s'agit d'une plate-forme efficace avec de nombreux outils adaptés à la CAO et nous avons plusieurs fois eu recours à ce dernier pendant nos cours à Polytech Lille. Dans notre projet le recours à ce type d'outils était indispensable car l'échelle des composants et des pistes était très petite, en effet nous utilisons des composants CMS.

C'est l'une des phases qui a demandé le plus de notre temps car une conception de circuit aussi complet ne se fait pas sans quelques précautions. Parmi les choses à vérifier :

- La taille des pistes doit être au moins égale à 27mil
- La taille des vias et des trous en extérieur doit être de 80mil et intérieur à 16mil minimum Ces deux règles sont nécessaires à la fois pour éviter une erreur de précision lors de l'impression mais aussi pour assurer que les pistes puissent supporter le courant.
- Les pistes ne doivent pas être à angle droit car les contraintes mécaniques pourraient les détacher
- le placement et le routage des composants doit être simple et fonctionnel, surtout lorsqu'il y a des composants montés en surface et traversants
- Il est préférable de diviser l'alimentation s'il existe une partie commande et une partie puissance
- L'espace qu'occupe la carte doit être réduit surtout s'il s'agit d'un projet embarqué
- Éviter au maximum les vias sur les cartes double face

Pour suivre toutes les règles ci dessus nous avons dû remanier le projet à plusieurs reprises voire même reprendre les routages depuis le début.

Lors de la conception d'une telle carte, un logiciel comme Altium a toujours besoin d'une librairie dans laquelle il pourra trouver les schémas, le brochage et les "footprints" d'un composant. Malheureusement pour nous aucune librairie connue du logiciel ne contenait ce dont nous avions besoin. Il a donc était nécessaire de recréer une librairie personnalisée pour répondre à nos besoins.

La création d'une librairie s'adresse habituellement à des professionnels et comprend toutes les informations possibles sur un composant depuis son nom jusqu'à une représentation 3D de l'espace physique utilisé dans un boîtier. Il était évident que nous n'avions pas besoin de toutes les informations, nous avons donc seulement représenté le schéma et le footprint comprenant la position des pattes à imprimer sur le circuit. Notre librairie personnalisée s'est donc vue dotée de deux composants (le registre à décalage 74HCT595 et le démultiplexeur 74HCT238).

3 Conception du cube de LED

La partie maîtresse de notre projet, le cube de LED, a été parmi les plus longues à réaliser car il s'agit d'un travail très minutieux et qui requiert une dextérité et une expérience de la soudure que nous n'avons pas. C'est pour cela que la création de la matrice a été confiée à M. Flamen. Néanmoins nous avons entièrement décrit le schéma de principe et le design global du cube. À l'intention de M. Flamen nous avons créé un guide au format PDF pour expliquer les caractéristiques de notre projet et la manière dont cette matrice devait être construite.

Comme expliqué précédemment, notre matrice est composée de 512 LED soit 8x8x8 LED. Cette configuration serait très difficilement réalisable s'il fallait relier chacune des LED, pour remédier à cette difficulté nous avons fait appel aux propriétés électroniques des LED en combinaison avec la logique du cube. Nous avons décidé de définir quelle LED s'allume en fonction de si elle est connectée à une source de tension d'un côté et à une masse de l'autre. Ainsi toutes les LED sont reliées par leur anode par groupe de 8 verticalement ce qui nous donne 64 groupes, et toutes les LED sont reliées par leur cathode par groupe de 64 horizontalement soit 8 groupes qui forment 8 étages de LED empilés verticalement.

Voici un schéma pour illustrer le principe de sélection d'une LED :

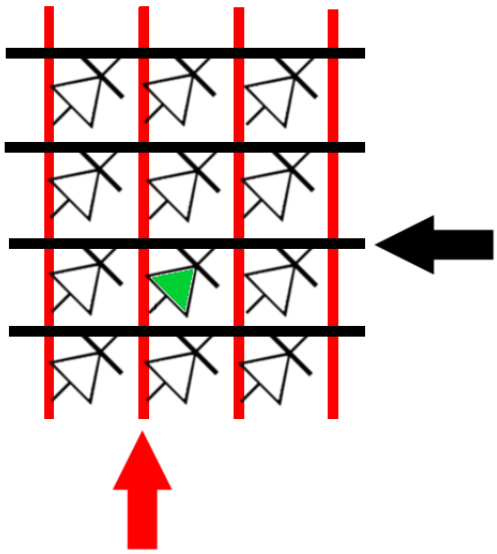

FIGURE 3.0.2 – Exemple de sélection d'une LED

Les traits rouges sont les anodes, les traits noirs sont les cathodes. En sélectionnant une anode commune à 8 LED et une cathode commune à 64 LED, on peut allumer la LED dont l'anode et la cathode correspondent à celles sélectionnées.

Théoriquement on peut donc allumer 64 LED au maximum en une seule fois, par exemple les 64 LED du premier étage. Cette limite ne peut pas être contournée sans utiliser plus de fils, or comment résoudre ce problème à partir de là ?

La réponse est simple et repose sur la faculté que l'homme a de ne pas pouvoir voir plus de 42 images par seconde environ (cette valeur peut changer en fonction de l'individu mais dans l'ensemble cette valeur suffit). En effet si l'homme ne peut voir plus de 42 images par secondes, alors il devient facile de le tromper avec une illusion. Si une LED est allumée seulement 1 fois

sur 8 à une fréquence supérieure à 42 Hz, alors l'œil humain ne perçoit pas la différence entre le moment où elle est allumée et le moment où elle est éteinte, il ne perçoit qu'une baisse de luminosité globale. Cette caractéristique est due à la persistance rétinienne et c'est ce principe que nous utilisons dans notre système. L'affichage est décomposé en 8 étapes qui se répètent indéfiniment, étage après étage les LED sont reliées à la masse par leur cathode et il suffit de les relier à la source par leur anode pour allumer celles qui doivent l'être.

Ainsi en synchronisant grâce à un micro-contrôleur on peut déterminer quelles LED sont allumées à n'importe quel endroit de la matrice, et ce sans que lœil humain ne se rende compte de l'alternance effectuée.

4 Pour aller plus loin

A la fin de ces quelques semaines, nous n'avons pas pu mener à terme tout ce qui était prévu, notamment en raison d'une arrivée très tardive de composants commandés. Afin d'être complètement opérationnel, ce projet nécessiterait quelques heures de travail supplémentaires.

4.1 Matériel

Au niveau matériel, il manque quelques cartes. En effet, il nous restait peu de temps pour la réaliser, et il valait mieux passer ce temps à finaliser proprement la carte de commande et sa connexion à la matrice grâce à des nappes de type Parallel ATA ou IDE telles qu'on en trouve dans les ordinateurs. Il a fallu sertir ces nappes sur les connecteurs et surtout y souder les résistances sans lesquelles le courant serait trop important et pourrait endommager les LED.

Pour les tests, nous avons donc alimenté les composants grâce à un générateur 5V. Néanmoins, la carte telle que prévue initialement serait très simple à réaliser, car nous avons tout le matériel nécessaire : le régulateur L78005CV, les condensateurs de découplage, les résistances...

Ensuite, la plaque sur laquelle la matrice doit être posée doit aussi être réalisée, il s'agit d'une carte reliant la nappe d'alimentation aux anodes et cathodes des LED. Pour le moment cette connexion se fait par fils volants.

Enfin, dans un souci de protection et d'esthétique, il est prévu que la matrice soit enfermée dans un coffrage de Plexiglas. Ce coffrage n'a pas pu être fabriqué lors de ce projet.

4.2 Logiciel

Au niveau logiciel, nous avons pensé à de nombreuses fonctionnalités supplémentaires, mais nous nous sommes concentrés sur le principal.

Premièrement, il faut revoir la fonction permettant d'envoyer les données à la matrice, car nous n'avons pas eu le temps de la programmer de façon optimale et l'affichage ne répond pas toujours correctement. Une fois l'affichage totalement fonctionnel, on peut imaginer de nombreuses applications possibles.

Ensuite, on peut s'intéresser à la façon d'améliorer le programme principal, c'est-à-dire le jeu Snake. Une fonctionnalité initialement prévue était une sélection de difficulté, qui changeait la vitesse de déplacement du serpent. Ceci est en fait simple à réaliser, il suffit de créer une fonction "menu" au démarrage qui permet de sélectionner la difficulté, par exemple en tournant le nunchuck dans un sens ou dans un autre.

On peut également penser à un moyen de sauvegarder les scores. Ceci est possible en utilisant des variables sauvegardées dans la mémoire flash de l'Arduino, qui n'est pas volatile.

Lors de ce projet, nous avons dû mener du début à la fin le développement de cette matrice 3D ainsi que des cartes et des programmes permettant de la commander. La conception et la réalisation de toutes les parties de ce projet nous ont mené à utiliser nos connaissances en électronique et en informatique, et nous ont permis de développer de nouvelles connaissances, notamment sur la manière de concevoir un circuit imprimé.

Même si le projet n'a pu être complètement terminé faute de temps, nous avons conçu une base suffisante pour permettre à d'autres étudiants ou professeurs de reprendre le projet et d'y ajouter les fonctionnalités manquantes, dans l'optique d'une présentation lors d'évènements à l'école comme les journées Portes Ouvertes.

Annexe A : Carte électronique

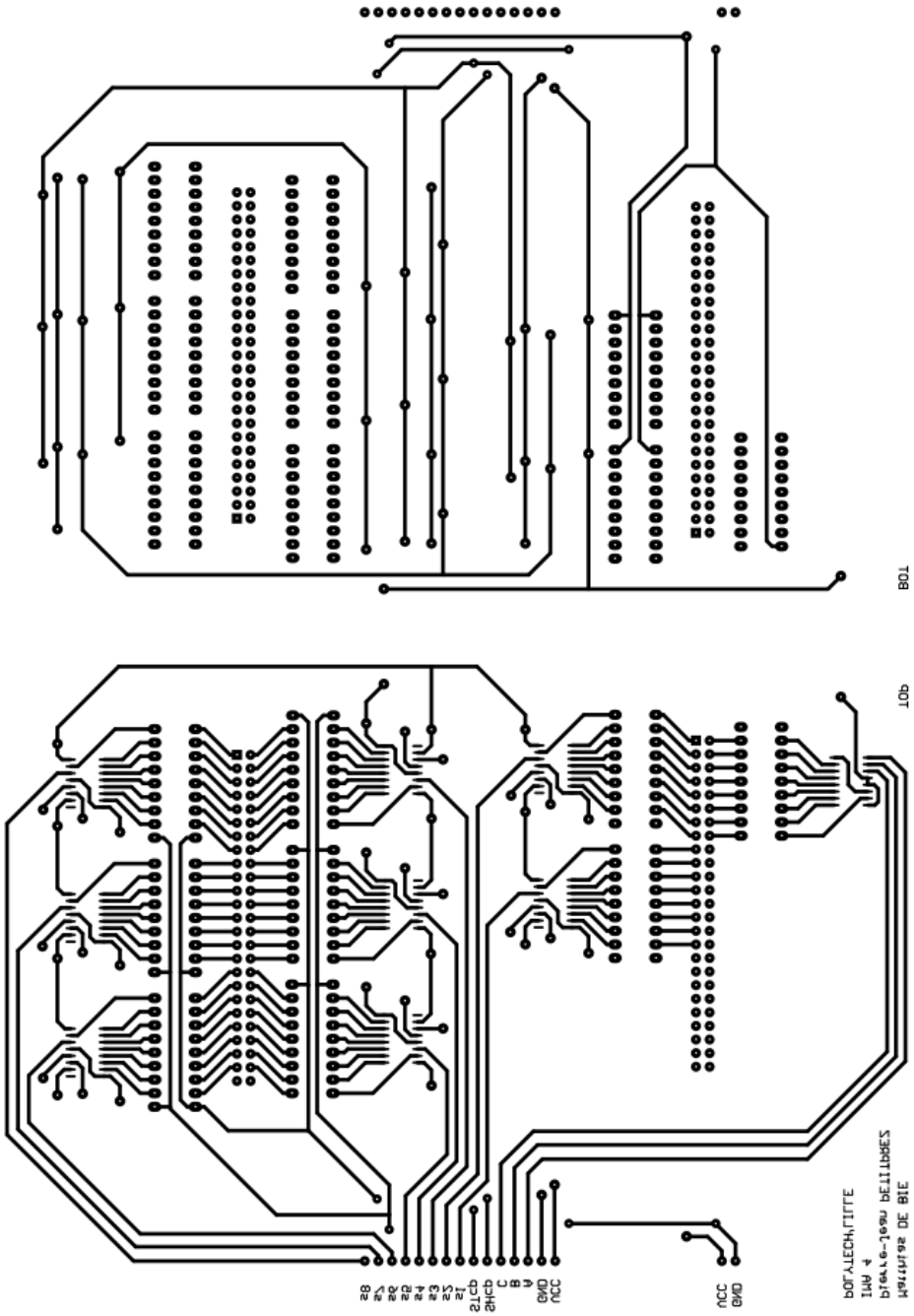

FIGURE A.0.1 – Carte de commande. Faces supérieure et inférieure

Annexe B : Programme Arduino

Ce programme se décompose en plusieurs fichiers, ceci afin de maximiser la modularité. Les fichiers d'entête ("headers") sont volontairement absents car inutiles ici.

Annexe B.1 : Fichier principal (Snake.ino)

```
#include <Wire.h>
#include <MsTimer2.h>
#include "Nunchuck.h"
#include "Affichage.h"
#include "Snake.h"
#include "Cube.h"
#include <Arduino.h>

//Declarations variables pour le nunchuck
const byte DEADZONEX = 30;
const byte DEADZONEY = 20;
const byte DEADZONEZ = 10;
const byte DEADZONEJOYX = 30;
const byte DEADZONEJOYY = 30;
byte cpt,choix=4;
byte data_nunchuck[6];
int accelX,accelY,accelZ,accelXCentre,accelYCentre,accelZCentre;
int joyX,joyY,joyXCentre,joyYCentre;
byte boutonZ,boutonC;

//Declarations variables pour l'affichage
const byte NB_X_MAX = 8;
const byte NB_Y_MAX = 8;
const byte NB_Z_MAX = 8;
const int NB_LED = 512;
const byte NB_OCTETS = NB_LED/8;
byte matrix[NB_OCTETS];
int inc = 0;


//Declarations variables pour le snake
const byte numX = 8;
const byte numY = 8;
const byte numZ = 8;
byte direc; //direction du snake
int snake[NB_LED];
byte startpoint = 10;
int snakehead = 0;
int snakesize = 1;
int eat = 0;
int apple = 0;
byte collision = 0;
int SnakeSpeed = 400;
```

```
unsigned long lastMillis = 0;


/*********************************/

void setup() {
  setup_cube(); // si on utilise le Cube 3D
  // setup_affichage() //  si on utilise la matrice 2D de Sparkfun
  nunchuck_init();
  MsTimer2::set(20, interrup_aff_cube);
  //periode de 20 ms, appel de la fonction commandant le cube
  MsTimer2::start();
  SnakeNewGame();
}

void loop() {
  nunchuck_read_data();
  nunchuck_reinit_Z();
  SnakeNewDirectionJoystick();
  Snake();
  delay(80);
}
```

Annexe B.2 : Gestion du Nunchuck : Nunchuck.cpp

```
#include "Nunchuck.h"
#include "Arduino.h"
#include <Wire.h>

void nunchuck_setpowerpins(){
  // Uses port C (analog in) pins as power & ground for Nunchuck
  //From Wiichuck demo, Tod E. Kurt, 2008
#define pwrpin PORTC3
#define gndpin PORTC2
    DDRC |= _BV(pwrpin) | _BV(gndpin);
    PORTC &=~ _BV(gndpin);
    PORTC |=  _BV(pwrpin);
    delay(100);  // wait for things to stabilize
}

void nunchuck_init(){
 nunchuck_setpowerpins();
 Wire.begin();
 /* Initialisation du nunchuck */
 Wire.beginTransmission(NUNCHUCK_ADDRESS);
 Wire.write(0xF0);
 Wire.write(0x55);
 Wire.endTransmission();

 Wire.beginTransmission(NUNCHUCK_ADDRESS);
 Wire.write(0xFB);
 Wire.write(0x00);
 Wire.endTransmission();

 nunchuck_calibrate();
}

void nunchuck_read_data(){
    // on demande 6 octets au nunchuck
   Wire.requestFrom(NUNCHUCK_ADDRESS, 6);

   cpt = 0;
    // tant qu'il y a des donnees
   while(Wire.available()){
      // on recupere les donnees
     data_nunchuck[cpt] = Wire.read();
     cpt++;
   }

    // on reinitialise le nunchuck pour la prochaine demande
   Wire.beginTransmission(NUNCHUCK_ADDRESS);
   Wire.write(0x00);
   Wire.endTransmission();
```

```
    if(cpt >= 5){
      // on extrait les donnees
      joyX = data_nunchuck[0] - joyXCentre;
      joyY = data_nunchuck[1] - joyYCentre;
      accelX = ((data_nunchuck[2] << 2) + ((data_nunchuck[5] >> 2) & 0x03))
                                  - accelXCentre; //MSB sur data[2] + LSB sur data[5]
      accelY = ((data_nunchuck[3] << 2) + ((data_nunchuck[5] >> 4) & 0x03))
                                  - accelYCentre;
      accelZ = ((data_nunchuck[4] << 2) + ((data_nunchuck[5] >> 6) & 0x03))
                                  - accelZCentre;
      boutonZ = data_nunchuck[5] & 1; //boutons Z et C : 0 si appuye, 1 si relache
      boutonC = (data_nunchuck[5] >> 1) & 1;
    }

}

void nunchuck_calibrate(){
  accelXCentre = 0;
  accelYCentre = 0;
  accelZCentre = 0;
  joyXCentre = 0;
  joyYCentre = 0;
  nunchuck_read_data(); //le premier tour est toujours faux
  delay(100);
  nunchuck_read_data();
  accelXCentre = accelX;
  accelYCentre = accelY;
  accelZCentre = accelZ;
  joyXCentre = joyX;
  joyYCentre = joyY;
}



void nunchuck_reinit_Z(){
  if(boutonZ==0) nunchuck_calibrate();
}
```

Annexe B.3 : Gestion de l'envoi à la matrice : Cube.cpp

```
#include "Cube.h"

void setup_cube(){
  pinMode(SHCP, OUTPUT);
  pinMode(STCP, OUTPUT);
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(S4, OUTPUT);
  pinMode(S5, OUTPUT);
  pinMode(S6, OUTPUT);
  pinMode(S7, OUTPUT);
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
}

void interrup_aff_cube(){
  short j,k;

    // Boucle des lignes (8 lignes, 64 LEDs)
    for(j=0;j<8;j++){
      // Boucle des LEDs (8 LEDs)
      // utilisation du tableau matrix
      digitalWrite(S0,matrix[(inc*8)+(j)]);
      digitalWrite(S1,(matrix[(inc*8)+(j)] >> 1) & 1);
      digitalWrite(S2,(matrix[(inc*8)+(j)] >> 2) & 1);
      digitalWrite(S3,(matrix[(inc*8)+(j)] >> 3) & 1);
      digitalWrite(S4,(matrix[(inc*8)+(j)] >> 4) & 1);
      digitalWrite(S5,(matrix[(inc*8)+(j)] >> 5) & 1);
      digitalWrite(S6,(matrix[(inc*8)+(j)] >> 6) & 1);
      digitalWrite(S7,(matrix[(inc*8)+(j)] >> 7) & 1);

      // Front montant de l'horloge des registres a decalage
      digitalWrite(SHCP,HIGH);
      digitalWrite(SHCP,LOW);
    }

    // Demultiplexeur
    if(inc!=0){
      digitalWrite(A,(inc==1 | inc==3 | inc==5 | inc==7) ? HIGH : LOW);
      digitalWrite(B,(inc==2 | inc==3 | inc==6 | inc==7) ? HIGH : LOW);
      digitalWrite(C,(inc==4 | inc==5 | inc==6 | inc==7) ? HIGH : LOW);
    }
    else{
      digitalWrite(A,LOW);
      digitalWrite(B,LOW);
      digitalWrite(C,LOW);
```

```
    }

    // Front montant du signal de chargement des bascules D
    digitalWrite(STCP,HIGH);
    digitalWrite(STCP,LOW);
    inc++;
    if(inc>7)inc=0;
}
```

Annexe B.4 : Gestion du tableau symbolisant la matrice : Affichage.cpp

```
#include "Affichage.h"
#include "Snake.h"


// Les 3 fonctions ci-dessous s'utilisent si on veut utiliser le programme
// sur une matrice 2D (8x8). Ne pas oublier de mettre NB_LED a 64 dans ce cas.

/*
char spi_transfer(volatile char data){
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait the end of the transmission
  {
  };
}

void setup_affichage(){
  byte clr;
  pinMode(DATAOUT,OUTPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(CHIPSELECT,OUTPUT);
  digitalWrite(CHIPSELECT,HIGH); //disable device
  SPCR = B01010001;               //SPI Registers
  SPSR = SPSR & B11111110;        //make sure the speed is 125KHz
  clr=SPSR;
  clr=SPDR;
  delay(10);
}


void affichage_envoi_byte(){
  //delay(100);
  digitalWrite(CHIPSELECT,LOW); // enable the ChipSelect on the backpack
  delayMicroseconds(100);
  for(int i=0; i<NB_OCTETS; i++){
      for(int j=0; j<8; j++){
          spi_transfer((matrix[i] >> j) & 1);
      }
  }
  digitalWrite(CHIPSELECT,HIGH); // disable the ChipSelect on the backpack
  //delayMicroseconds(500);
}

*/

void clearMatrix_byte() {
    // remise a 0 de la matrice
  for (int i=0; i<NB_OCTETS; i++){
    matrix[i] = 0;
  }
```

```
}


int puissance(int a, int b){
// La fonction puissance de math.h n'est pas fiable (2^7 = 127 ??)
  int c = 1;
  for(int i=0;i<b;i++) c *= a;
  return c;
}


void updateMatrix_byte(){ //Stocke dans le tableau matrix les leds
 // allumees par le serpent et la pomme
// Dans matrix un bit = une led
 clearMatrix_byte();
 int i = 0;
 while(i<snakesize && i<NB_LED){
   matrix[snake[i]/8] |= puissance(2,(snake[i] % 8));
   i++;
 }
matrix[apple/8] |= puissance(2,(apple%8));

}
```

Annexe B.5 : Gestion du serpent : Snake.cpp

```
#include "Arduino.h"
#include "Snake.h"
#include "Nunchuck.h"
#include "Affichage.h"


/*** snake new direction ***/
void SnakeNewDirectionJoystick(){
  //Directions
  //0 = haut
  //1 = droite
  //2 = bas
  //3 = gauche
  // 4 et 5 pour la 3e direction
  if(joyX>DEADZONEJOYX) direc = 3;
  else if(joyX<-DEADZONEJOYX) direc = 1;
  else if(joyY>DEADZONEJOYY) direc = 0;
  else if(joyY<-DEADZONEJOYY) direc = 2;
  else if(axeY<-DEADZONEY) direc = 4;
  else if(axeY>DEADZONEY) direc = 5;
}


/*** snake new game ***/
void SnakeNewGame(){
  clearMatrix_byte();
  snakesize = 1; // taille du serpent : 1 LED
  snakehead = startpoint;
  snake[0] = snakehead;
  SnakeNewApple();
  direc = 1; // choix arbitraire d'une direction pour commencer
  collision = 0;
  updateMatrix_byte();
}

/*** generate apple ***/
void SnakeNewApple(){
  apple = random(NB_LED);
  for (int i = 0; i < snakesize; i++){ //pour eviter que la nouvelle pomme
  //n'apparaisse sous le snake
    if (snake[i] == apple) {
      SnakeNewApple();
    }
  }
}

/*** snake check apple ***/

void SnakeCheckApple(){
```

```
  if (snakehead == apple){
     eat = 1; // flag indiquant que la pomme est mangee
     if(snakesize<NB_LED) snakesize++; // on agrandit le snake
     SnakeNewApple();
  }
}


/*** snake check collision ***/
void SnakeCheckCollision(){
  for (int i = 1; i < snakesize; i++) {
    if (snake[i] == snakehead) {
      collision = 1;
    }
  }
}

/*** snake calculate new head position ***/
void SnakeNewHead(){
  if(direc==2){ //down
    if ((snakehead + numX) > ((numY * numX - 1)))
         snakehead = snakehead % numX;
    else snakehead += numY;
  }
  else if(direc==1){ //right
    if ((snakehead - (snakehead % numX)) == (((snakehead+1) - (snakehead+1) % numX)))
         snakehead += 1;
    else snakehead = snakehead - (numX - 1);
  }
  else if(direc==0){ //up
    if ((snakehead - numX) < 0) snakehead += (numY - 1) * numX;
    else snakehead -= numX;
  }
  else if(direc==3){ //left
    if (snakehead == 0) // probleme avec 0 -> on ne peut pas aller a la case -1
        // -> il faut aller a la case 7
      snakehead = numX-1;
    else if ((snakehead - (snakehead % numX)) == (((snakehead-1) - (snakehead-1) % numX)))
        snakehead -= 1;
    else snakehead += numX - 1;
 }
 else if(direc == 4){ //3e direction vers le haut : changement d'etage
   snakehead += numX * numY;
   if(snakehead > NB_LED) snakehead -= NB_LED;
 }
 else if(direc == 5){
   snakehead -= numX * numY;
   if(snakehead < 0) snakehead += NB_LED;
 }
}
```

```
/*** snake move ***/
void Snake(){
  if (millis()-lastMillis >= SnakeSpeed){
    SnakeNewHead();
    SnakeCheckApple();
    SnakeCheckCollision();

    if(collision == 0){ //si pas de collision
        for(int i=snakesize-1; i>0; i--) snake[i] = snake[i-1];
        snake[0] = snakehead;

        if (eat == 1) eat = 0; //remise a 0 du flag indiquant si la pomme est mangee
        updateMatrix_byte();
    }

    else {
      endGame(); //fin du jeu si collision
    }
    lastMillis = millis();
  }
}


/*** End game ***/
void endGame(){
  delay(1000);
  SnakeNewGame();
}
```