INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE ROUEN

EC Électronique pour l'ingénieur

---

# RAPPORT DE PROJET D'ÉLECTRONIQUE

---

***Titre du projet :***

Jeu de *casse-briques* avec joystick


***Auteurs :***

Gautier DARCHEN

Enora GICQUEL

Alexandre HUAT

Romain JUDIC


***ASI 3 — Groupe 1.1***

26 décembre 2015

# RÉSUMÉ

Durant notre premier semestre de notre formation dans le département ASI, en 3ème année donc, nous avons eu un projet d'électronique à réaliser. Nous avons alors choisi de réaliser un jeu, puisque c'est un moyen ludique de mêler informatique et électronique.

Nous avons choisi le jeu de casse-brique, qui est un cas d'école. Nous avons également choisi ce sujet car nous ne voulions pas forcément poursuivre un projet déjà entrepris par un groupe des années précédentes.

Ce projet nous a beaucoup apporté, tant sur le plan électronique qu'organisationnel. En effet, nous étions tous d'environs différents — Enora venant d'un IUT, Romain d'une Prépa MP, Alexandre & Gautier de STPI — et ce projet était une excellente opportunité d'apprendre des compétences de chacun.

---

# ABSTRACT

During the first semester of our studies in the ASI department, hence during the 3rd year, we had to carry out a project about electronics. Therefore, we chose to create a video game, while it's a funny way to combine computer science with electronics.

We chose the *Breakout* clone because this game is well-known. We also chose this subject because we didn't really want to continue a project that had already been undertaken by another group during the previous years.

This project brought many things to all of us, in the field of electronics and about the relationships between co-workers. Indeed we came from various trainings — Enora from an University Institutes of Technology, Romain from a "Prepa MP", Alexandre & Gautier from STPI — and this project was a great opportunity for us to learn from each other.

# Table des matières




Gautier DARCHEN
Enora GICQUEL
Alexandre HUAT
Romain JUDIC

# Introduction

Dans le cadre du cours d'électronique pour l'ingénieur, il nous a été proposé un projet sur quatre séances afin de concrétiser nos connaissances en fin de semestre. Ce projet est en effet un très bon moyen de mettre en pratique la théorie que nous avons appris en électronique, mais aussi en algorithmique, comme nous allons le voir.

Face à la consigne de réaliser un projet réalisable entièrement, donc assez court pour être réalisable et totalement fonctionnel, nous avons choisi de démarrer un nouveau projet au lieu d'en continuer un des années précédentes.

Il s'agit alors de mettre au point un jeu vidéo portable avec son boîtier et le jeu du casse-briques intégré. Le jeu est rendu possible grâce à l'utilisation d'un joystick. Ce projet impose donc un lien très étroit entre électronique et algorithmique.

Le travail a ainsi été divisé en trois parties : la conception électronique, celle du boîtier en impression 3D et enfin le développement logiciel du jeu.

# I — Présentation du projet

## I.1 — Une mini-console, jeu de casse-brique

La réalisation de notre console de jeu nécessitait :

- d'un boîtier qui servirait à protéger l'électronique du projet ;
- de matériel électronique :
    - d'un micro-contrôleur ;
    - d'un joystick ;
    - d'un écran ;
    - d'un PCB ;
    - d'autres composants à déterminer.
- d'un code à implanter sur le micro-contrôleur.

## I.2 — Comment jouer ?

Tout d'abord, le projet final est présenté comme un boîtier orange, sur lequel on peut voir un écran et un joystick. Pour allumer la mini-console de jeu, il suffit de brancher la pile grâce au trou réalisé sur le côté du boîtier.

Le jeu démarre ensuite sur un menu. Le joueur a alors la possibilité de choisir un niveau de difficulté entre 1 (facile) et 3 (difficile) par l'intermédiaire du joystick, ou d'accéder aux crédits. S'il choisit de jouer, l'utilisateur valide son choix — après avoir sélectionné le niveau — en se plaçant sur la rubrique "Jouer" et en appuyant sur le joystick.

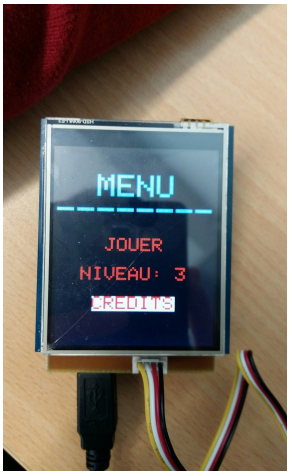



Figure I.2.1-Image du menu

La partie est alors lancée. Pour jouer, l'utilisateur peut faire naviguer sa barre de jeu (appelée *paddle* dans notre code) grâce au joystick. Il dispose de trois vies, une vie étant perdu dès lors que la balle tombe à côté de la barre de jeu. A chaque case touché,

le joueur se voit attribué un score ; ce score par brique cassée est plus élevé dans le niveau difficile que dans le niveau facile.


Figure I.2.2 - Affichage du score

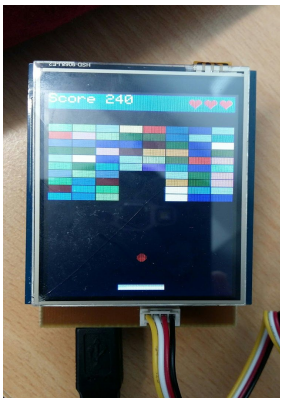



Figure I.2.3 - Partie en cours

Si toutes les briques sont détruites, la partie recommence avec la difficulté supérieure ce qui permet d'améliorer son score. La partie se termine quand le joueur ne dispose plus d'aucune vie. Lorsque la partie est finie, le score du joueur s'affiche, et ce dernier est renvoyé sur le menu.

# II — État de l'art

## II.1 — Étude de marché

Ci-dessous, quelques technologiques similaires à ce que nous voulons réaliser pour notre projet.

### II.1.1 — La Gamebuino

Il s'agit d'une console portable mise au point par Aurélien RODOT, ancien étudiant de l'ENISE de Saint-Etienne, en 2013.

La console a la dimension d'une carte bancaire. Elle est basée sur une carte Arduino. Elle contient quelques jeux rudimentaires par défaut et l'utilisateur a la possibilité d'en rajouter de nouveaux qu'il aura conçus ou téléchargés via Internet. Elle est modulable : on peut y ajouter un accéléromètre, un émetteur bluetooth et un module multijoueur Wi-Fi. Elle dispose d'un écran LCD de Nokia 5110 recyclé en noir et blanc. Son prix de vente est de 35 €.

Figure II.1.1.1 - Deux versions de la Gamebuino

### II.1.2 — La Game Boy Color (GBC)

Sortie en 1997, il s'agit de la première console portable de Nintendo disposant d'un écran couleur.

Ses caractéristiques :

- un écran LCD TFT de 2.6 pouces avec affichage en couleur ;
- un processeur Zilog Z80 Custom 8 bits ;
- 128 ko de RAM ;
- alimentée par 2 piles LR3, elle a une autonomie de batterie d'environ 13 h ;
- utilise des cartouches comme supports de jeux, elle supporte des car-

touches de 256 ko à 4 Mo ;

- elle propose plus de 450 jeux.


Figure II.1.2.1 – 3 GBC en différents coloris

### II.1.3 — La Microvision de MB

La Microvision, créée par Jay SMITH, est considérée comme la première console portable jamais commercialisée.

Elle propose :

- une douzaine de jeux dans la version américaine dont les classiques Pinball et Casse-Brique ;
- un écran LCD noir et blanc ;
- une alimentation assurée par 2 piles de 9 V pour une autonomie de 2-3 h seulement ;
- un potentiomètre pour se déplacer dans le jeu.


Figure II.1.3.1 - La Microvision de DB

## II.2 — La carte Arduino

Les cartes Arduino sont des circuits imprimés open-source basés sur le micro-contrôleur ATmega328 couramment utilisés pour des projets d'électronique.

Arduino est programmable via un langage proche du C qui lui est propre. C'est une carte portable qui s'adapte à tous les systèmes d'exploitation.

Sa taille, ses performances et son coût en font un matériel adéquat pour notre projet électronique. Il existe de nombreux modèles, l'Arduino Uno est couramment utilisée, simple, et semble suffire pour notre projet.

La carte Arduino permettra d'asservir l'écran LCD à partir des informations envoyées par le joystick. Elle contiendra donc les programmes du jeu et sera connectée au joystick et à l'écran.


Figure II.2.1 – Arduino Uno : vue de haut et de bas

Les caractéristiques de la carte sont les suivantes :

- Prix : 19.50 EUR sur lextronic
- Microcontrôleur : ATmega328
- Tension de fonctionnement : 5 V
- Tension d'alimentation (recommandée) : 7-12 V
- Tension d'alimentation (limites) : 6-20 V
- Nombre d'E/S numériques : 14 (dont 6 disposant d'une sortie PWM)
- Nombre d'entrées analogiques : 6 (utilisables en E/S numériques)
- Intensité max par E/S (5 V) : 40 mA
- Intensité max pour la sortie 3.3 V : 50 mA
- Intensité max disponible pour la sortie 5 V : Dépend de l'alimentation utilisé ; 500 mA max si le port USB est utilisé seul
- Mémoire Flash : 32 kb dont 0.5 kb utilisés par le bootloader
- SRAM : 2 kb (ATmega328)
- EEPROM : 1 KB (ATmega328)
- Vitesse d'horloge : 16 MHz

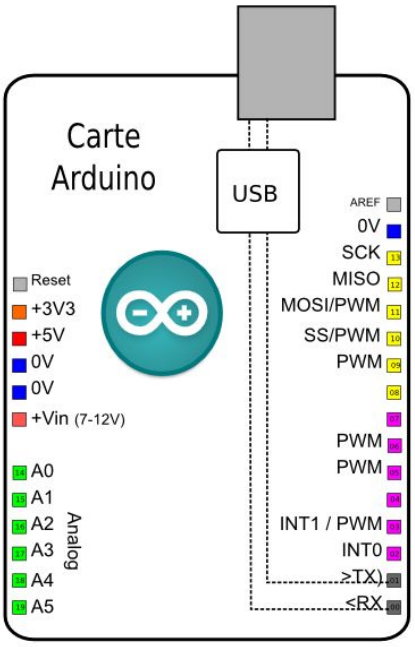



Figure II.2.2 – Schéma de brochage de l'Arduino Uno

La carte Arduino peut être alimentée via une connexion USB ou par une alimentation externe de 7-12 V (suffisamment de puissance et pas de surchauffe). L'alimentation externe peut passer par le jack de 2.1 mm de la carte. Des piles pourront servir d'alimentation.

Les broches d'E/S numériques fonctionnent en 5 V. Les broches PWM permettent de moduler un signal. Les broches analogiques quant à elle nous permettront de connecter un joystick à l'Arduino.

Des bibliothèques open-source sont disponibles pour programmer Arduino. Elles nous aideront à programmer l'affichage sur l'écran LCD. Ce dernier sera relié à la carte par les entrées/sorties numériques.

## II.3 — Les joysticks

Le joystick est un périphérique permettant à un utilisateur d'interagir avec une machine. Il est le plus souvent utilisé dans le cadre des jeux vidéo. Il existe de nombreux types de joysticks. Certains possèdent des boutons (pour multiplier les actions possibles par l'utilisateur), d'autres sont plus ergonomiques (la plupart du temps utilisés dans des simulateurs de vol)...

Dans notre cas, bien choisir ce périphérique est essentiel puisque cela déterminera la jouabilité des jeux qui seront développés. Dans le cadre de ce projet, nous allons développer des jeux assez simples, dans le sens il ne sera pas nécessaire de choisir un joystick très sophistiqué, et donc plus cher.

Nous allons donc ici voir quels sont les différents types de joysticks, de sorte à choisir celui qui sera le plus adapté à notre problème.

## II.3.1 — Le digital joystick

Ce type de joystick est a priori le plus utilisé pour les jeux sur PC. Ces joysticks sont composés d'un manche fixé sur un socle, sur lequel peuvent également se trouver un (ou plusieurs) bouton. Les joysticks de type digital sont relativement simplistes dans la mesure où ils permettent à l'utilisateur d'effectuer les actions : gauche, droite, haut. bas, et une action supplémentaire (un « big red button » qui sert le plus souvent de touche « feu »).

Figure II.3.1.1 – Un joystick digital : le Atari 2600 Joystick

## II.3.2 — Le paddle joystick

Le paddle joystick est un des types de joystick les plus anciens. Il consiste en un gros bouton que le joueur a la possibilité de tourner, pour par exemple contrôler une raquette dans l'ancêtre des jeux de tennis : le jeu de « Pong ». Ce bouton n'est autre qu'un simple potentiomètre. A côté de ce gros bouton peuvent également se trouver d'autres boutons poussoirs, offrant à l'utilisateur la possibilité d'effectuer d'autres actions.

Figure II.3.2.1 – Un paddle joystick : le Atari Paddle Joystick

## II.3.3 — Le joystick analogique

Les joysticks analogiques sont plus récents et sont ceux utilisés dans les manettes des consoles de jeux vidéo les plus récentes. Ils combinent les propriétés des joysticks numérique et paddle, présentés précédemment. En effet, comme sur les joysticks digital, ils permettent de diriger le jeu (haut, bas, gauche, droite). Par ailleurs, tout comme les joysticks paddle, ils utilisent un potentiomètre de sorte à mesurer l'intensité d'une commande. Ce joystick peut donc par exemple interpréter si le joueur souhaite orienter le jeu complètement vers la droite ou légèrement vers la droite selon s'il penche le joystick au maximum ou non. Enfin, la plupart du temps, les joysticks analogiques ont une dernière fonction : ils fonctionnent comme un bouton poussoir, ce qui permet une nouvelle fois d'offrir une fonctionnalité supplémentaire.

Figure II.3.3.1 – Un joystick analogique d'une manette de PS3

## II.3.4 — Les joysticks analogiques sophistiqués

Ils sont utilisés en majorité pour des jeux nécessitant de nombreuses interactions possible entre l'utilisateur et la machine. Par exemple, ce sont ces types de joysticks qui sont utilisés pour les simulateurs de vol. Ils comportent de nombreux boutons, sont très précis en terme d'intensité d'orientation (plus encore que le sont les analog joysticks classiques). Ceux-ci étant plus cher, il ne nous sera pas du tout nécessaire d'investir dans des joysticks aussi sophistiqués.

Figure II.3.4.1 – Le joystick analogique sophistiqué Thrustmaster T-Flight Stick X

### II.3.5 — Notre choix de joystick

Pour notre projet, il serait confortable d'utiliser un joystick analogique. En effet, ce type de joystick est relativement précis (intensité d'orientation grâce aux potentiomètres) et n'est pas excessivement cher. Par ailleurs, nous aurons certainement besoin d'avoir des boutons sur notre manette de jeu, pour permettre à l'utilisateur d'effectuer d'autres actions qu'un simple déplacement.

Ainsi, au vue des différents critères que doit remplir le joystick que nous nécessitons, un joystick semble tout à fait correspondre à nos exigences : le « Nunchuck » de la Nintendo Wii. Il se trouve dans une gamme de prix raisonnable, possède un joystick analogique précis. Sur le « Nunchuck » se trouvent également deux boutons que nous pourrons utiliser lors du développement des jeux.

Enfin, nous savons que le « Nunchuck » dispose d'un accéléromètre intégré, que nous pourrons éventuellement utiliser par la suite si nécessaire et si les délais le permettent.

Figure II.3.5.1 – Le Nunchuck de la Nintendo Wii

## II.4 — L'écran LCD

L'écran LCD (Liquid Crystal Display traduit en français par « Écran à Cristaux Liquides ») est un périphérique utilisé dans quasiment tous les affichages électroniques. On le retrouve notamment dans des appareils électroniques tels que les montres, les tableaux de bord de voiture et les calculatrices. On peut également en trouver des formes plus complexes dans les écrans d'ordinateur et dans les écrans plats.

Les écrans LCD sont largement utilisés grâce à un coût de production assez faible, dépendant cependant de la taille et de la résolution de l'écran.

Dans le cadre de ce projet, nous développerons des jeux assez simples ne nécessitant qu'un petit écran et un nombre limité de pixels.

Nous allons donc voir les différents types d'afficheur disponible dans la gamme d'écran LCD afin de choisir le plus adapté à notre projet.

## II.4.1 — Afficheur alphanumérique

Les afficheurs alphanumériques sont les plus rencontrés dans les appareils électroniques. Ils permettent l'affichage de lettres, chiffres, ainsi que quelques caractères spéciaux. Les caractères sont prédéfinis et leur emplacement sélectionnable par un curseur. On n'a donc pas accès à chaque pixel mais à des sections définies par adresse.


Figure II.4.1.1 - Un afficheur alphanumérique : le module WH1602B1

## II.4.2 — Afficheur graphique monochrome

Les afficheurs graphiques permettent la production d'affichage beaucoup plus évolué puisqu'on a accès à chaque pixel. Ces écrans permettent donc une liberté totale quant à l'affichage des graphismes, mais demandent en contrepartie une mise en œuvre plus complexe.

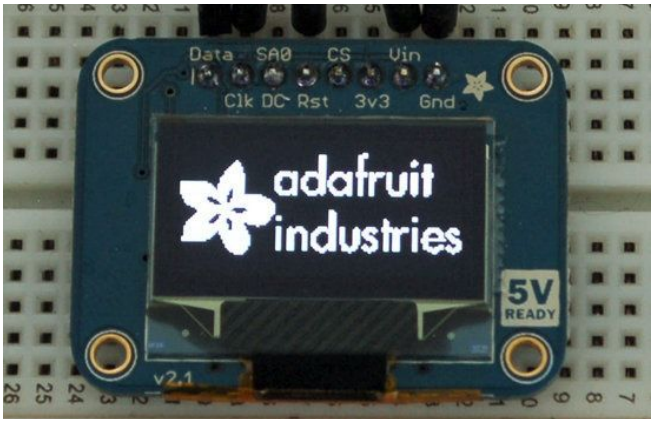



Figure II.4.2.1 - Un afficheur graphique monochromatique : le module ADA326

## II.4.3 — Afficheur graphique couleur

Les afficheurs graphiques couleur sont une évolution des afficheurs graphiques monochromes. Chaque pixel de couleur visible est en réalité composé de 3 sous-pixels : un rouge, un vert et un bleu. L'affichage est donc plus agréable mais

demande la gestion de 3 fois plus de pixels qu'un afficheur graphique monochrome de la même résolution.

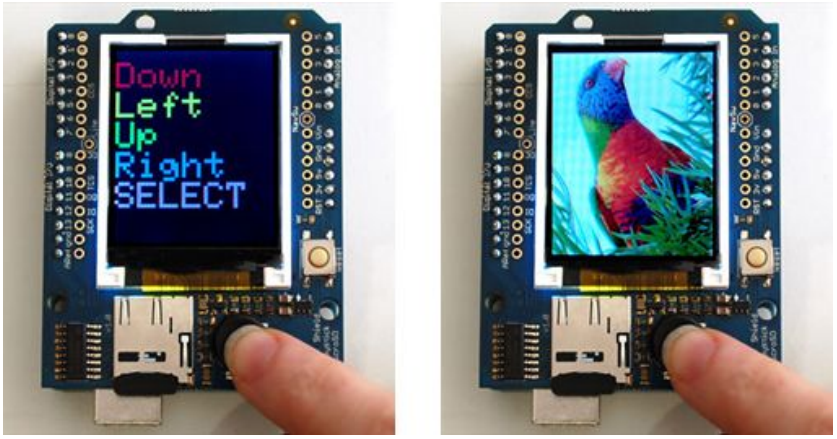



Figure II.4.3.1 - Un afficheur graphique monochrome : le module ADA802

## II.4.4 — Notre choix d'écran

Notre projet devant gérer l'affichage de jeux, l'utilisation d'un afficheur graphique est indispensable.

Cependant, un problème se pose avec l'exemple du casse brique qui demande, pour les briques, l'affichage d'éléments adjacents mais devant pourtant être distingués.

Deux solutions ont été envisagées :

- Un écran couleur permettant ainsi de différencier les éléments par des couleurs différentes
- Un écran monochrome ayant une résolution suffisante pour pouvoir se permettre de laisser un espace de pixels éteints entre chaque élément sans que cela nuise à la compréhension de l'affichage.

# III — Cahier des charges

## III.1 — Analyse fonctionnelle

- L'appareil s'allume en branchant le micro-contrôleur à son alimentation.
- Au démarrage, l'appareil affiche un menu de choix du jeu : “Casse-Brique” ou “Simon” avec la possibilité de choisir le niveau de jeu.
- L'utilisateur peut naviguer dans le menu à l'aide du joystick.
- Le joueur peut agir dans le jeu grâce au joystick.
- Le joueur peut mettre en pause ou quitter le jeu et revenir au menu à n'importe quel moment.
- Pour éteindre l'appareil, l'utilisateur doit débrancher le micro-contrôleur de l'alimentation.

## III.2 — Contraintes

- Le budget est de 100 €.
- Le produit final ne doit pas dépasser 18×8×8 $cm^3$.

## III.3 — Composants électroniques

- Micro-contrôleur Arduino
- Joystick à 4 directions avec un bouton poussoir
- Écran LCD compatible avec Arduino

# IV — Programmation

La programmation sur le microcontrôleur a été une part importante de ce projet. D'une part pour faire le lien entre les différents composants électroniques mais également pour l'implémentation en elle-même du jeu.

La programmation se faisant sur une carte Arduino, le code a été réalisé dans un langage qui lui est propre : le langage Arduino, basé sur le C++. Le programme devait donc permettre : l'implémentation du jeu de casse-brique, l'interaction du joueur *via* le joystick et la visualisation sur l'écran de l'état du jeu.

## IV.1 — Le casse-brique

Le développement du casse-brique se base essentiellement sur le déplacement de la balle. Le jeu dépend entièrement de ses déplacements et de ses collisions.

La position de la balle est donc définie par 2 variables correspondant à ses coordonnées x et y, et son déplacement par deux vecteurs qui, additionnés, correspondent à un vecteur vitesse. Chaque déplacement de la balle (séparée par quelques centièmes de seconde) correspond donc à l'addition des vecteurs aux coordonnées de la balle.

Les déplacements de la balle sont altérés lors de collisions avec les bords de l'aire de jeu, une brique ou la barre de jeu. Il faut donc vérifier à chaque mouvement de la balle si sa position correspond également à celle d'un de ces obstacles. Pour les bords de l'aire de jeu, la vérification est simple car le test se fait sur les mêmes valeurs tout au long de la partie (ici les dimensions de l'écran).

Le test pour la barre de jeu se fait plus dynamiquement étant donné que sa position change constamment au cours de la partie. Le test est plus compliqué pour les briques car il faut à la fois calculer si la balle se trouve à la position d'une brique mais aussi vérifier que cette brique n'a pas déjà été détruite. Pour représenter l'état des briques, nous avons choisi l'utilisation d'un tableau où chaque case identifie une brique, si elle est intacte la valeur de la case sera 1 ou 0 si elle a été brisée.

Une fois la collision détectée, il suffit de modifier les vecteurs correspondant au mouvement de la balle. Une simple inversion du vecteur *x* pour une collision sur un obstacle à la verticale suffit ou sur le vecteur y pour l'horizontale. Pour une brique, il faut également changer son état dans la case correspondante du tableau. Par contre, le changement de trajectoire est différent pour la barre de jeu. En effet suivant l'endroit de la barre que la balle touche, un angle est calculé et est ajouté à celui formé entre la trajectoire de la balle et la barre. De plus, à chaque collision la balle accélère légèrement grâce à une augmentation de la

valeur d'un des vecteurs ce qui permet d'augmenter la difficulté du jeu au fur et à mesure de la partie.

D'autres fonctionnalités ont également été ajoutées. Des vies qui, au nombre de 3 en début de partie, diminuent lorsque la balle atteint le bord inférieur de l'aire de jeu et met fin à la partie lorsque la valeur atteint 0. On évalue également un score correspondant au nombre de briques brisées et à la difficulté de la partie en cours.

## IV.2 — L'utilisation du joystick

L'interaction de l'utilisateur sur le jeu se fait par l'intermédiaire du joystick qui contrôle la barre de jeu. Sa position est gérée, comme la balle, par des coordonnées *x* et *y*. Le *y* est défini en début de partie et seul le *x* est influencé par le joystick.

Le joystick étant connecté aux pins de l'Arduino, une fonction est incluse pour lire la valeur envoyée. La nouvelle position de la barre est calculée en fonction de la direction et de l'amplitude du mouvement effectué par le joystick.

## IV.3 — L'affichage sur l'écran

La visualisation de l'état du jeu est primordiale pour l'utilisateur. L'écran utilisé pour ce projet permettait l'utilisation de librairies spécifiques pour la carte Arduino. Ainsi, il était possible d'utiliser des fonctions prédéfinies afin d'écrire du texte ou de dessiner des formes géométriques.

Nous n'avions donc qu'à fournir les coordonnées de chaque élément à dessiner aux fonctions correspondantes et nous avons pu voir apparaître un disque pour la balle et des rectangles pour la barre de jeu et les briques.

Nous avons tout de même rencontré quelques difficultés par rapport au temps de dessin de l'écran. En effet, étant donné que l'état du jeu change constamment, il nous semblait logique d'effacer l'écran et de tout redessiner entre chaque changement. Mais la fonction de remplissage de l'écran prenait beaucoup trop de temps pour avoir un affichage fluide. Il a donc fallu trouver une astuce pour pallier à ce problème et la solution a été de redessiner à chaque changement d'état la balle et la barre à la même position que précédemment mais de la même couleur que le fond et ensuite de les dessiner de la bonne couleur à leur nouvelle position. Les briques, elles, sont dessinées en début de partie et lorsque l'une d'elles est détruite, un rectangle de la même couleur que le fond est dessiné pour la recouvrir.

# V — Étude technique

## V.1 — Étude du boîtier

Pour la réalisation de notre projet, nous avions besoin d'un support matériel. En effet, il était indispensable d'utiliser un boîtier permettant à la fois de protéger toute l'électronique de l'usure et de l'extérieur, mais aussi dans un soucis d'esthétique et aussi de faciliter l'utilisation de la petite " console de jeu de casse-brique ".

Pour ce faire, nous avions plusieurs possibilités. Il nous était en effet possible :

- d'utiliser un boîtier dont le modèle 3D avait déjà été conçu pour les projets des années précédentes. Ce choix, bien que solution évidente de facilité, nous aurait contraint à utiliser un boîtier dont les dimensions et les finitions ne correspondaient pas aux besoins que requérait notre projet.

- de modéliser nous-mêmes le un boîtier selon un modèle 3D à confectionner à l'aide d'un logiciel de *CAO*[1]. Ce choix nous a paru de prime abord le plus judicieux puisqu'il nous laissait *a priori* libres de créer un boîtier correspondant au maximum à nos attentes.

  Nous avons donc décidé de créer notre propre modèle 3D.

### V.1.1 — La modélisation du boîtier

#### V.1.1.1 — Le boîtier

Tout d'abord, nous avons dû nous atteler à la modélisation du projet très tôt, avant même d'avoir fini de choisir les composants matériels comme le *joystick*, par exemple. De plus, aucun d'entre nous n'avait jamais manipulé de logiciel de *CAO*, ce qui nous a certainement fait perdre un certain temps.

Dans nos premières esquisses, nous avions déjà la forme du boîtier. Nous savions en effet que la boîte serait nécessairement creuse, avec un emplacement pour l'écran, et un trou circulaire pour le *joystick*. Cependant, étant donné le fait que notre bon de commande n'était au tout début pas terminé, nous n'avions pas les dimensions exactes qu'il nous fallait pour le boîtier. Nous avons alors réalisé le boîtier sous SolidWorks®, à des dimensions que nous savions arbitraires.

Par la suite, une fois le bon de commande achevé, nous avions une idée plus précise sur ce qu'il nous faudrait mettre exactement dans le boîtier, et donc un aperçu plus détaillé des dimensions à prévoir. Cependant, nous n'étions pas

[1] CAO : Conception Assistée par Ordinateur.

certains d'obtenir tout le matériel commandé pour la réalisation de ce projet. En effet, pour le *joystick* par exemple, il existe un certain nombre de composants plus ou moins similaires. Pour ce qui est du diamètre du *joystick*, les différents constructeurs ont choisi des tailles semblables, environ 33 mm, mais pour la hauteur, cela était légèrement plus compliqué (nous ne savions pas exactement quelle datasheet consulter…).

Peu de temps avant la fin, suite à plusieurs difficultés que nous détaillerons par la suite, nous sommes parvenus à trouver les dimensions exactes de notre boîtier :

| **Longueur** | **Largeur** | **Profondeur** |
|---|---|---|
| 150 mm | 70 mm | 50 mm |

Voici le boîtier dans sa version finale :

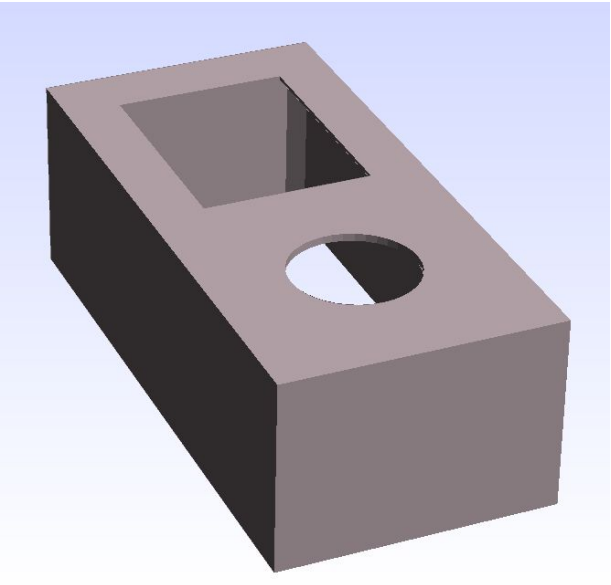


Figure V.1.1.1 — Version finale du boîtier

### V.1.1.2 — Le couvercle

Pour fermer l'ensemble, il nous fallait évidemment un couvercle. Ce dernier est relativement simple, il s'agit d'une plaque de $150 \times 70 = 10500$ mm² = 105 cm². Il est épais de 2.5 mm. En voici une illustration :

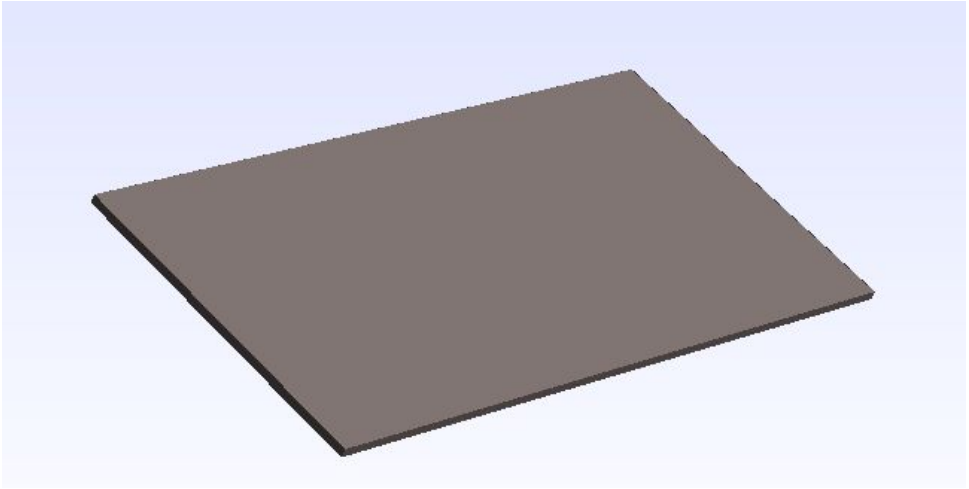


Figure V.1.1.2.1 — Socle permettant de fermer le boîtier

Sur le boîtier, nous avons ajouté à l'intérieur aux quatre angles des surfaces pleines triangulaires de très faible surface, comme on peut le voir sur la figure suivante. Ces surfaces nous ont par la suite permis visser le couvercle sur le boîtier pour maintenir l'ensemble clos.

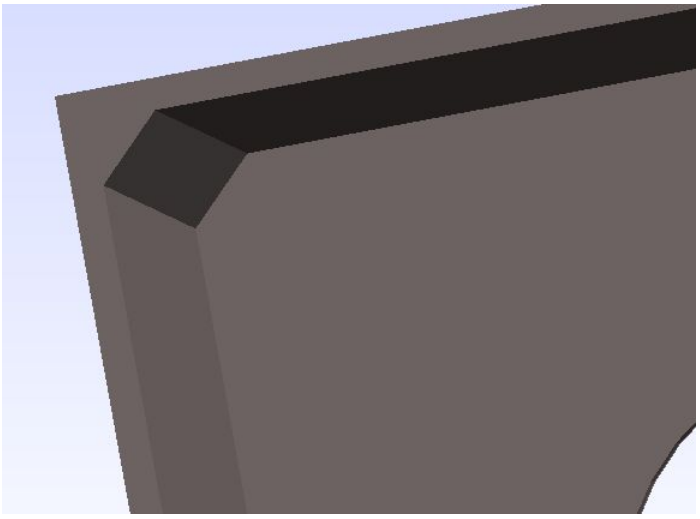


Figure V.1.1.2.2 — Surface triangulaire permettant de fixer le couvercle sur le boîtier

### V.1.1.3 — Le support du *joystick*

Le *joystick* nécessite d'affleurer à la surface du boîtier pour permettre au joueur de le manipuler aisément. Ainsi, il nous fallait concevoir un support pour tenir ce *joystick*. Ce support — que nous avons finalement décidé de créer en tant que pièce à part — consiste simplement en un " U " dont les dimensions sont :

| Longueur | Largeur | Profondeur |
|---|---|---|
| 48 mm | 25 mm | 22 mm |

Nous pouvons voir ce socle sur la figure suivante. Nous avons choisi de coller ce support directement sur le boîtier.

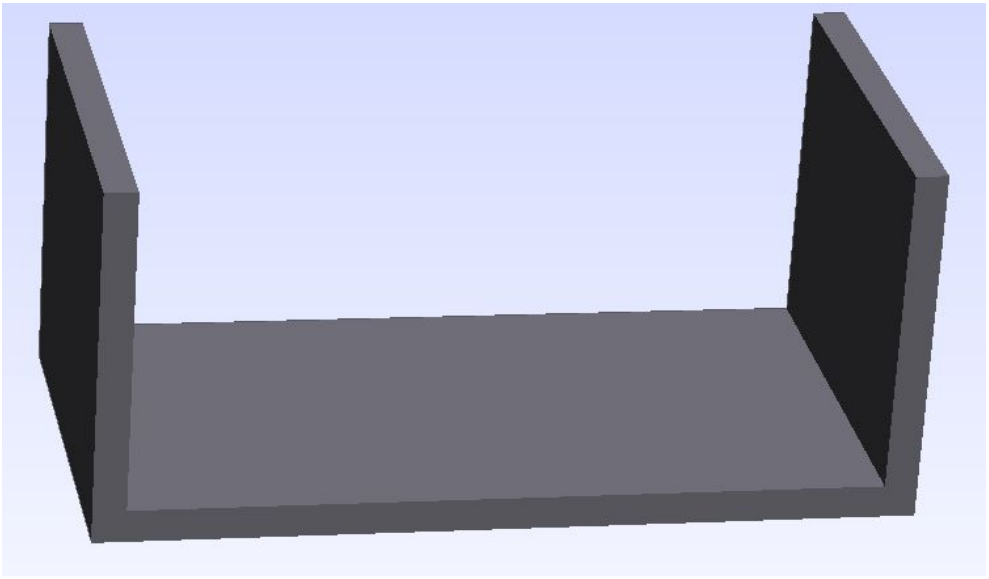


Figure V.1.1.3 — Support en forme de “U” permettant de maintenir le *joystick*

## V.1.2 — Les difficultés rencontrées lors de la modélisation

Lors de la conception du modèle 3D de notre boîtier, nous avons été confrontés à plusieurs difficultés. Certains de ces problèmes concernaient la conception en elle-même, et d'autres concernaient la phase d'impression 3D.

**Les problèmes liés à la conception**

Étant donné le fait qu'aucun d'entre nous ne savait manipuler un outil de CAO tel que SolidWorks, nous avons eu quelques soucis en terme d'efficacité sur la conception du boîtier. C'est par des recherches et l'aide de personnes confirmées en ces pratiques que nous sommes parvenus à trouver des solutions.

Par ailleurs, un autre problème était simplement l'accès à des ordinateurs équipés de SolidWorks pour travailler sur la modélisation du boîtier. En effet seules des salles appartenant aux départements Méca, EP ou STPI sont équipées de ce logiciel. Il nous fallait donc attendre qu'aucun groupe de ces divers départements n'ait cours pour que nous puissions travailler sur la modélisation.

Enfin, à quelques jours du rendu final des fichiers de modélisation pour l'impression 3D, nous avons été confronté à un autre problème : les licences SolidWorks des étudiants ASI n'étaient plus valables.

**Les difficultés liées à l'impression 3D**

Tout d'abord, nous n'avions aucunement conscience des aspects à respecter lors de la modélisation 3D pour que l'impression 3D des pièces soit possible. Nous avons alors cherché en premier lieu à créer des pièces qui soient esthétiques. C'est ici que nous avons rencontré notre premier problème. En effet, comme on peut voir sur la figure suivante, nous avions par exemple réalisé un congé arrondi autour du trou réservé au *joystick* pour rendre la pièce plus belle. Par la suite, nous avons compris qu'il était très compliqué d'imprimer une pièce de la sorte puisque, la boîte étant creuse, l'imprimante 3D l'imprimerait dans le sens opposé (ouverture du socle vers le haut) et le relief du congé serait impossible à gérer.

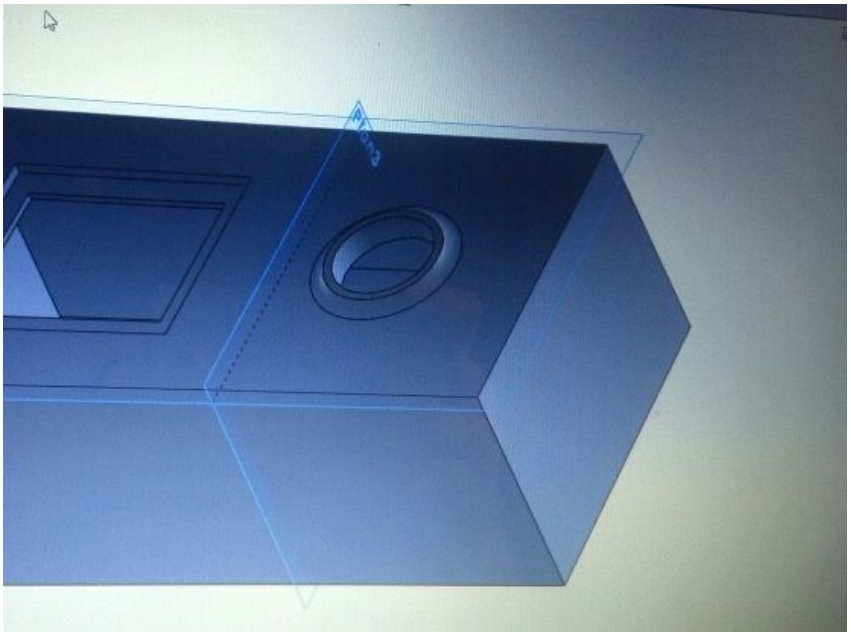
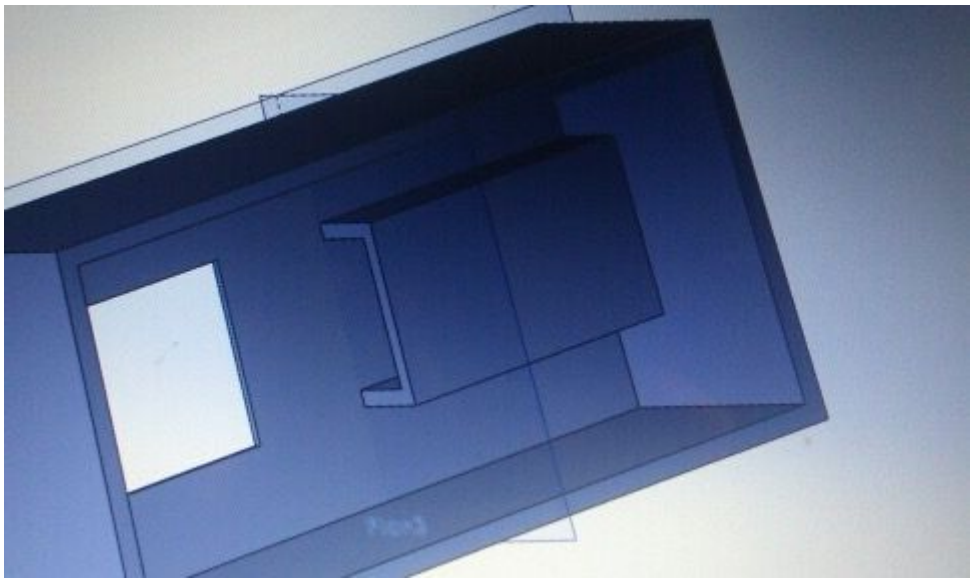


Figure V.1.2.1.1 — Un des problèmes rencontrés : le congé autour de l'emplacement du *joystick*

Ensuite, nous avions pensé créer le support en "U" pour le *joystick* directement dans le boîtier, comme on peut le voir sur la figure suivante, pour avoir une seule pièce à imprimer (et non pas plusieurs à imprimer séparément et à assembler ensuite).

Après avoir soumis cette pièce pour l'impression 3D, il nous a été dit que celle-ci n'était pas convenable puisque l'impression aurait duré plus de 20 heures. Nous avons compris par la suite qu'une fois de plus, ce problème était lié aux contraintes imposées par l'imprimante 3D : ne pas créer une pièce vide, car l'imprimante chercherait à remplir ce vide par de la matière plastique, ce qui est en

très grande partie responsable de la durée de l'impression. C'est pour cette raison que



nous avons opté pour la décision d'imprimer le support en “U” à part.

Figure V.1.1.2.1 — Un des problèmes rencontrés : le support en “U” initialement dans le boîtier

Après avoir corrigé ces quelques erreurs, nous avons à nouveau soumis nos pièces pour l'impression. Une nouvelle fois, l'impression n'était pas réalisable, car le logiciel d'impression indiquait que cette dernière aurait duré environ 10 heures.

Nous avons alors réduit l'épaisseur du boîtier de 3 mm à 2 mm. Nous avons aussi réduit l'ensemble des dimensions du boîtier pour limiter au maximum la taille des pièces à imprimer.

L'impression était convenable et c'est ainsi que nous avons obtenu notre boîtier final.

# V.2 — Conception du PCB

## V.2.1 — Le schéma

La mise en oeuvre électronique consistait à assembler l'Arduino, un joystick, un écran LCD et une alimentation pour l'Arduino UNO. Par défaut, nous disposions d'une Arduino UNO et d'un écran Adafruit 2.8" TFT LCD. Puis nous avons choisi :

- une pile de 9 V pour l'alimentation, l'Arduino fonctionnant sur du 7-12 V ;
- un joystick seeed Grove pour sa taille et la facilité à le connecter à l'Arduino, étant déjà monté sur un mini PCB.

Pour relier la pile à l'Arduino nous avons opté pour un cordon à fiche de 2.1 mm (aucune connexion au PCB). Le joystick sera connecté indirectement par un câble Grove à 4 broches au PCB. L'écran se fixe quant à lui directement sur l'Arduino.

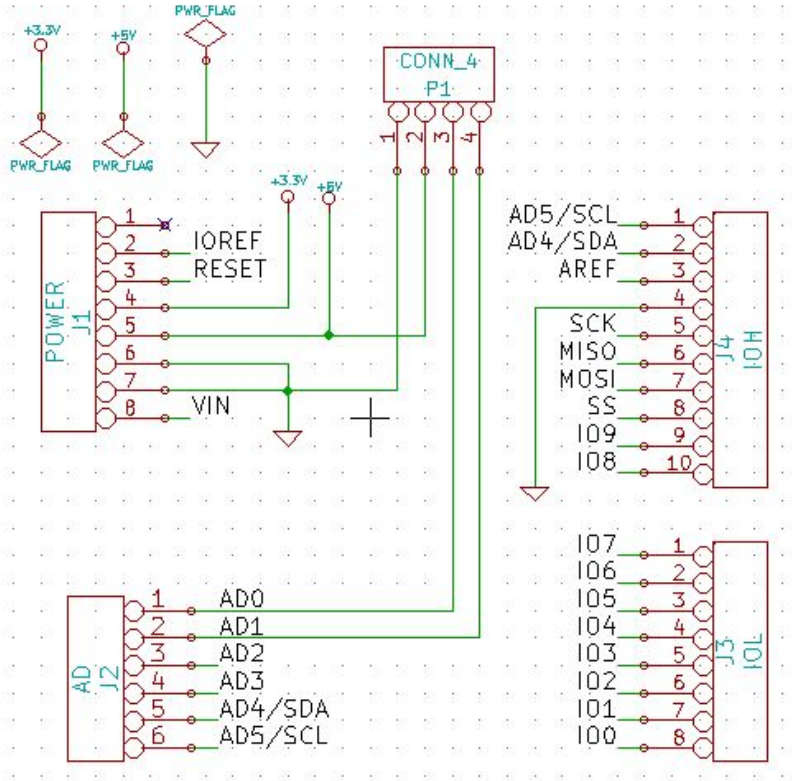



Figure V.2.1.1 - Schéma du PCB sous KiCad

Pour réaliser le PCB sous KiCad nous avons téléchargé une bibliothèque pour Arduino UNO R3[2]. Comme nous le voyons ci-dessus, le CONN_4 est le connecteur à 4 broches du joystick. Ce dernier s'alimente en 5 V sur l'Arduino et ses pins X et Y sont reliés aux pins A0 et A1 de l'Arduino. Ce branchement est possible car selon sa

[2] http://www.idreammicro.com/post/Shield-Arduino-R3

datasheet, l'écran n'utilise comme pins analogiques que A5 et A6. Tous les autres pins du schéma sont ceux de l'Arduino.

À partir de ce schéma, nous sommes arrivés à l'empreinte de PCB suivante :

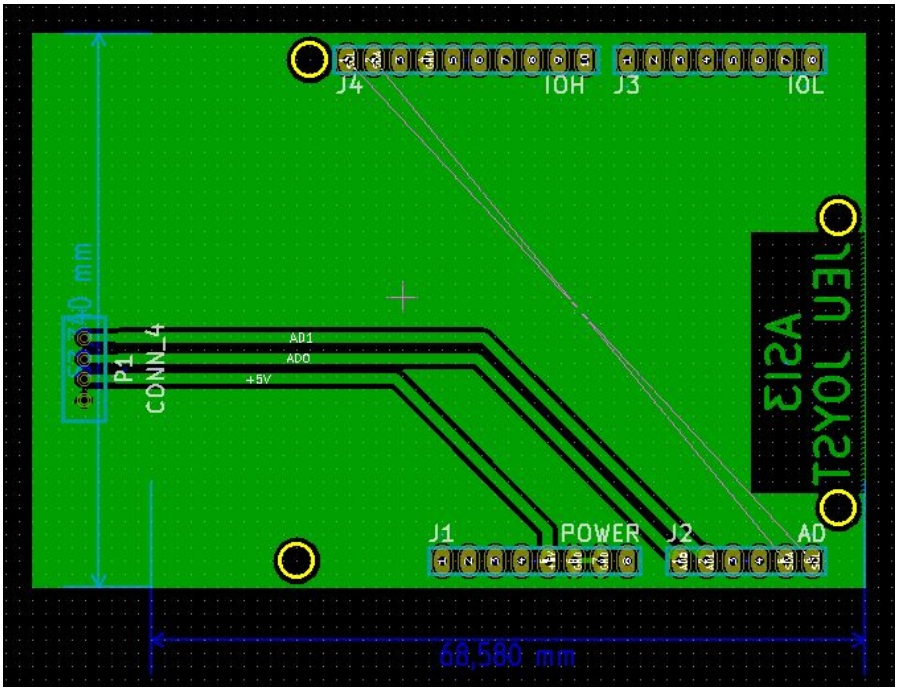



Figure V.2.1.2 - Empreinte du PCB sous KiCad

Les pins sont correctement reliés comme sur le schéma. Tout en vert nous avons un plan de masse. Sur la droite nous avons inscrit du texte qui indique le côté piste lorsqu'il est lisible. Un petit travail supplémentaire de cette étape fut de concevoir une empreinte pour le connecteur Grove du joystick car celui-ci avait un pas de 2 mm au lieu de 2.54 mm.

Ci-dessous, le résultat final après impression et soudure. Pour relier le *shield* de l'écran à l'Arduino nous avons utilisé des connecteurs femelles 6, 8 et 10 broches.

Le perçage des trous fut rapide mais la difficulté était de bien aligner chacun des pins. La soudure fut plus difficile car nos interpistes sont très petits. Cela causa des problèmes une fois tous les composants soudés car il y avait de nombreux faux contacts (des pins reliés à la masse), le cuivre n'ayant pas été correctement retiré de la plaque.

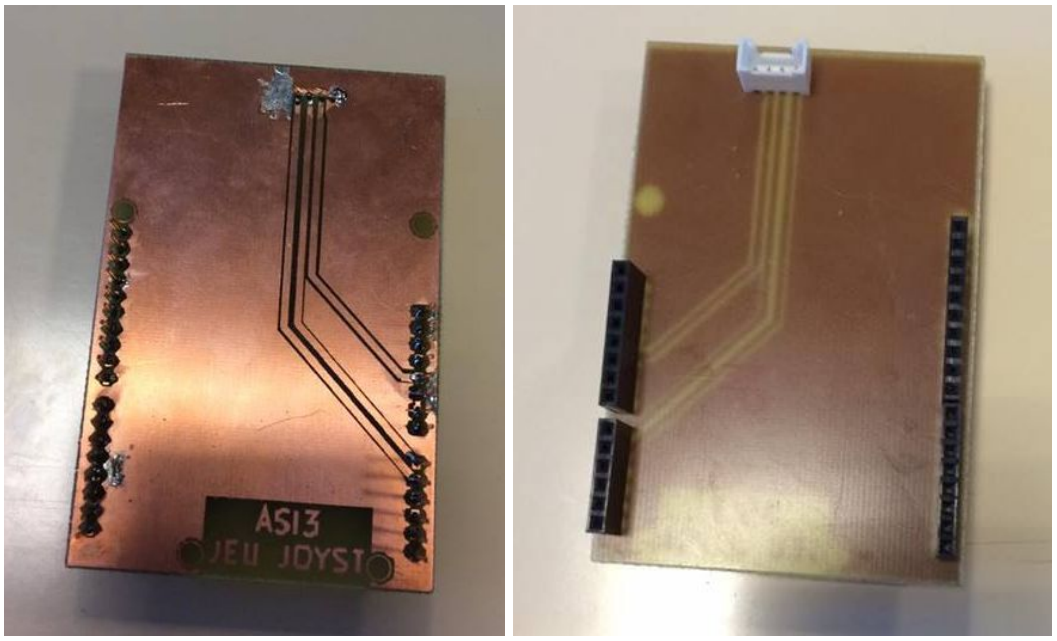



Figure V.2.1.3 - PCB côté cuivre et côté composant

Comme on le voit à gauche, les petits inter-pistes ont rendu la soudure assez difficile, de l'étain a débordé à certains endroits.

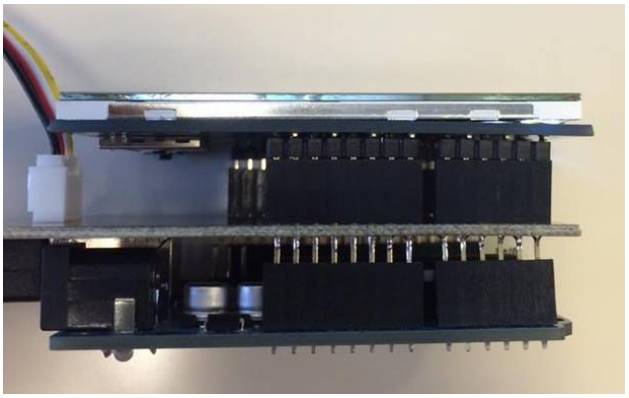


Figure V.2.1.4 - Le shield monté sur le PCB monté sur l'Arduino

Le montage des différents étages du projet était difficile car les pins n'étaient pas bien alignés ni vraiment équidistants.

# VI — Évolutions envisagées

Ce projet mêlant électronique et programmation, il y a automatiquement diverses améliorations et évolutions envisageables. Arrivés à la fin du temps imparti, voici ce que nous pouvons proposer à ce sujet :

- A propos du code, une idée évidente serait d'ajouter un(des) nouveau(x) jeu(x) à celui déjà présent. Il serait aussi possible d'ajouter des niveaux de difficulté plus élaborés comme par exemple des briques "spéciales" avec des actions associées, des balles multiples ou encore une configuration initiale des briques différente.

- Pour ce qui est de l'électronique, il est possible d'ajouter une gestion de l'alimentation via un interrupteur ainsi que d'ajouter des boutons de commande pour les nouveaux jeux et pour de nouvelles fonctionnalités.

- Concernant le boîtier, il est possible d'optimiser l'espace occupé en mettant au point une modèle 3D plus proche des composants électroniques.

- Lors de la rédaction du cahier des charges, nous avions prévu de développer une option permettant au joueur de mettre le jeu en pause et de revenir au menu. Mais cela impliquait un développement assez fastidieux pour une option qui n'était pas essentielle. Dans le temps qui nous était imparti, nous avions pour but d'aller à l'essentiel et de rendre un jeu fonctionnel.

Le principe simple de ce projet permet en effet d'imaginer bien d'autres changements. Pourquoi d'ailleurs ne pas détourner l'objet de son but premier pour s'en resservir comme d'une manette de commande grâce au joystick ? Libre choix ensuite aux potentiels successeurs des promotions suivantes.

# VII — Bon de commande

Bon de commande

**Commande de composants électroniques**
**L. HENRIET Pôle EEAS**

**COMMANDE GOTRONIC**

| Prévision Matériel | Référence | Prix HT | Prix TTC | Quantité | Sous Total | Lien |
|---|---|---|---|---|---|---|
| Carte Arduino UNO | 25950 | 16,25 € | 19,50 € | 1 | 19,50 € | http://www.gotronic.fr/art-carte-arduino-uno-12420.htm |
| Pile Alcaline 9V 6LR61 | 9435 | 2,17 € | 2,60 € | 1 | 2,60 € | http://www.gotronic.fr/art-pile-alcaline-9v-6lr61-5573.htm |
| Module Joystick Grove COM90133P | 31297 | 5,42 € | 6,50 € | 1 | 6,50 € | http://www.gotronic.fr/art-module-joystick-grove-com90133p-19059.htm |
| Shield écran TFT tactile ADA1651 | 32494 | 32,67 € | 39,20 € | 1 | 39,20 € | http://www.gotronic.fr/art-shield-ecran-tft-tactile-ada1651-21306.htm |
| Cordon alimentation 9V + connecteur 5,5 × 2,1 mm | 48165 | 1,63 € | 1,95 € | 1 | 1,95 € | http://www.gotronic.fr/art-cordon-alimentation-pile-9v-19414.htm |
| Connecteur ARD010 femelle longues broches | 48004 | 0,42 € | 0,50 € | 1 | 0,50 € | http://www.gotronic.fr/art-connecteur-ard010-18114.htm |
| Connecteur ARD008 femelle longues broches | 48003 | 0,42 € | 0,50 € | 2 | 1,00 € | http://www.gotronic.fr/art-connecteur-ard008-18113.htm |
| Connecteur ARD006 femelle longues broches | 48002 | 0,42 € | 0,50 € | 1 | 0,50 € | http://www.gotronic.fr/art-connecteur-ard006-18112.htm |
| Lot de 5 câbles Grove 5 cm | 31422 | 1,75 € | 2,10 € | 1 | 2,10 € | http://www.gotronic.fr/art-lot-de-5-cables-grove-5-cm-19055.htm |
| Connecteur Grove coudé | 31234 | 0,13 € | 0,15 € | 1 | 0,15 € | http://www.gotronic.fr/art-connecteur-grove-coude-20572.htm |

| Total | 74,00 € |
|---|---|

**Projet ASI 3 : Jeu sur écran LCD contrôlé par joystick avec Arduino**
DARCHEN Gautier, GICQUEL Enora, HUAT Alexandre, JUDIC Romain

Page 1

# VIII — Cahier de suivi

Bien évidemment, tout le travail n'a pas été fourni exclusivement durant les séances de projet, mais durant les périodes entre deux séances. Le cahier de suivi ici dressé a pour but de retracer notre progression au fur et à mesure du projet. Les travaux indiqués correspondent ici aux tâches réalisées durant la séance de projet et à terminer pour la séance suivante.

## Séance 1

Date : 21/10/15
Travail réalisé :

- *Gautier DARCHEN, Romain JUDIC* : calcul des dimensions du boîtier et réalisation du châssis sous SolidWorks
- *Gautier DARCHEN* : essais d'affichage des briques pour le jeu
- *Enora GICQUEL* : tests d'affichage de l'écran et programmation du casse-brique, réflexion en pseudo-code sur le code du casse-brique
- *Alexandre HUAT* : bon de commande, calcul des dimensions du boîtier, tests sur un joystick avec Arduino

## Séance 2

Date : 18/11/15
Travail réalisé :

- *Gautier DARCHEN, Romain JUDIC* : suite et fin de la conception du boîtier 3D, adaptation selon les contraintes dues à l'impression 3D
- *Enora GICQUEL* : suite de la programmation du casse-brique
- *Gautier DARCHEN, Enora GICQUEL, Romain JUDIC* : calcul de coordonnées diverses necessaires à la programmation du casse-brique
- *Alexandre HUAT :* Suite et fin des tests sur joystick, réflexion quant à la conception du PCB à réaliser

## Séance 3.1

Date : 2/12/15
Travail réalisé :

- *Alexandre HUAT* : Conception du PCB sous KiCad
- *Enora GICQUEL, Gautier DARCHEN, Romain JUDIC* : Suite de la programmation du jeu de casse-brique, gestion des mouvements de la barre du jeu commandée par le joystick

## Séance 3.2

Date : 9/12/15
Travail réalisé :

- *Enora GICQUEL, Gautier DARCHEN, Romain JUDIC :* Gestion de la collision de la balle avec les briques, et du mouvement de la balle dans le code du jeu et poursuite du code (réglages de petits problèmes liés au code)
- *Alexandre HUAT* : Fin de la conception du PCB, selon les contraintes imposées par les techniques de création (rendre les pistes le plus large possible de sorte à ronger le moins de cuivre possible...), création du menu du jeu

## Séance 4

Date : 16/12/15
Travail réalisé :

- *Gautier DARCHEN, Alexandre HUAT* : perçage du PCB et soudage des composants.
- *Enora GICQUEL, Romain JUDIC* : Suite du codage du jeu, gestion des trois niveaux possibles (selon la vitesse de la balle)

# Conclusion

Au terme des 4 séances et de la période impartie pour la réalisation du projet, notre groupe a produit comme prévu initialement un jeu vidéo portable sous forme d'une mini-console équipée du jeu du casse-brique et d'un menu de navigation au démarrage.

La mini-console dans ce qu'elle implique (à savoir pouvoir tout simplement jouer une partie du jeu proposé) est assez aboutie par rapport à la promesse que nous nous étions faite, néanmoins il est toujours possible d'y ajouter quelques modifications dans le code pour y ajouter des jeux par exemple.
De plus, dans l'optique d'une reprise du projet les années suivantes, il est toujours possible de modifier la partie électronique car le boîtier est assez large à l'intérieur et de l'espace est encore libre pour ajouter des pièces supplémentaires.

Un tel projet a été pour nous une opportunité de mettre en pratique une partie des cours du semestre, à savoir notamment l'utilisation d'un micro-contrôleur, la génération logicielle d'un PCB et la gestion d'un potentiomètre.

De plus, nous avons pu faire une nouvelle expérience du travail en équipe : la répartition des tâches, la gestion du temps, la cohabitation sur de mêmes travaux... Les compétences de chacun nous ont permis d'appréhender ce projet de la meilleure manière.

# Annexe

## Code Arduino du projet

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <Wire.h>
#include <SD.h>

#define TFT_DC 9
#define TFT_CS 10

#define SD_CS 4

#define TS_MINX 150
#define TS_MINY 130
#define TS_MAXX 3800
#define TS_MAXY 4000

#define JOYST_X A1
#define JOYST_Y A0
#define JOYST_MINX 206
#define JOYST_MAXX 798
#define JOYST_MINY 203
#define JOYST_MAXY 797
#define BP_EVENT (xJoyst==1023)
#define JOYST_XMILIEU (JOYST_MAXX+JOYST_MINX)/2
#define JOYST_PLAGEX (JOYST_MAXX-JOYST_XMILIEU)/2


#define COULEUR_FOND ILI9341_BLACK
#define COULEUR_BALLE ILI9341_RED
#define COULEUR_PADDLE ILI9341_WHITE
#define COULEUR_FOND_AFFICHAGE ILI9341_BLUE

#define LIGNES 10
#define COLONNES 8
#define HAUTEUR_BRIQUE 12
#define BANDE_AFFICHAGE 24
#define ESPACE_NOIR 24
#define RAYON_BALLE 6
#define ADD_CONTACT_ZONE 5
#define LARGEUR_PADDLE 60
#define HAUTEUR_PADDLE 8
#define FACTEUR_ACCELERATION 1.005

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

// Variables globales
int largeur;
int hauteur;
int hauteurJeu;
int largeurBrique;
```

```
int xBalle;
int yBalle;
int xPaddle;
int yPaddle;
int xJoyst;
int yJoyst;

float vxBalle;
float vyBalle;

int briques[LIGNES][COLONNES];
uint16_t couleurs[18];

boolean fin;

#define NB_VIE_INIT 3
int vieRestante;

int score;

int choixMenu;
#define MENU 0
#define JOUER 1
#define NIVEAU 2
#define BONUS 3
int choixNiveau;

boolean carteSDPresente;

void setup() {
  Serial.begin(9600);

  carteSDPresente = SD.begin(SD_CS);

  tft.begin();
  tft.fillScreen(COULEUR_FOND);
  initialisationDimension();
  choixMenu = MENU;
  choixNiveau = 2;
}

// Fonction de bouclage
void loop(){
  if (choixMenu==MENU || choixMenu==NIVEAU) {
   choisirMenu();
  }
  else if (choixMenu==JOUER) {
   tft.fillScreen(COULEUR_FOND);
   tft.fillRect(0, hauteurJeu, largeur, BANDE_AFFICHAGE, ILI9341_BLUE);
   afficherBriques();
   casseBrique();
  }
  else if (choixMenu==BONUS) {
   bonus();
  }
}
```

```
void initialisationDimension() {
  largeur = tft.width();
  hauteur = tft.height();
  hauteurJeu = hauteur - BANDE_AFFICHAGE;
  largeurBrique =  largeur/COLONNES;

  couleurs[0]=ILI9341_NAVY;
  couleurs[1]=ILI9341_DARKGREEN;
  couleurs[2]=ILI9341_DARKCYAN;
  couleurs[3]=ILI9341_MAROON;
  couleurs[4]=ILI9341_PURPLE;
  couleurs[5]=ILI9341_OLIVE;
  couleurs[6]=ILI9341_LIGHTGREY;
  couleurs[7]=ILI9341_DARKGREY;
  couleurs[8]=ILI9341_BLUE;
  couleurs[9]=ILI9341_GREEN;
  couleurs[10]=ILI9341_CYAN;
  couleurs[11]=ILI9341_RED;
  couleurs[12]=ILI9341_MAGENTA;
  couleurs[13]=ILI9341_YELLOW;
  couleurs[14]=ILI9341_WHITE;
  couleurs[15]=ILI9341_ORANGE;
  couleurs[16]=ILI9341_GREENYELLOW;
  couleurs[17]=ILI9341_PINK;
}

void afficherMenu(int choix_temp) {
  // on met le texte à l'endroit
  tft.setRotation(2);
  tft.setCursor(0,0);

  // en-tete de menu
  uint16_t c = ILI9341_RED;
  uint16_t surbrille = ILI9341_WHITE;
  tft.setTextColor(ILI9341_CYAN);
  tft.setTextSize(5);
  tft.println();
  tft.println("  MENU");
  tft.setTextWrap(false);
  tft.println  ("--------");

  tft.setTextSize(3);
  // choix JOUER
  tft.print("\n");
  setSurbrillance(choix_temp,JOUER,c,surbrille);
  tft.println("JOUER");

  // choix NIVEAU
  tft.setTextColor(c,COULEUR_FOND);
  tft.print("\n  ");
  setSurbrillance(choix_temp,NIVEAU,c,surbrille);
  tft.print("NIVEAU: "); tft.println(choixNiveau);

  // choix BONUS
  tft.setTextColor(c,COULEUR_FOND);
  tft.print("\n   ");
```

```
  setSurbrillance(choix_temp,BONUS,c,surbrille);
  tft.println("CREDITS");

  // on remet l'affichage à l'endroit
  tft.setRotation(0);
}

void setSurbrillance(int choix_temp, int txt_menu, uint16_t c, uint16_t
surbrille) {
  if (choix_temp==txt_menu)
   tft.setTextColor(c,surbrille);
  else
   tft.setTextColor(c,COULEUR_FOND);
}

void choisirMenu() {
  tft.fillScreen(COULEUR_FOND);
  int choix_temp = 0;
  do {
   afficherMenu(choix_temp);
   xJoyst = analogRead(JOYST_X);
   yJoyst = analogRead(JOYST_Y);

   if (yJoyst<300 && choix_temp<BONUS)
   choix_temp++;
   else if (yJoyst>700 && choix_temp>JOUER)
   choix_temp--;

   if (choix_temp==NIVEAU)
   choisirNiveau();
// delay(10);
  }
  while (!BP_EVENT);
  choixMenu = choix_temp;
}


void choisirNiveau() {
  if (xJoyst<300 && choixNiveau>1)
   choixNiveau--;
  else if (xJoyst>700 && choixNiveau<3)
   choixNiveau++;
}

// affiche les crédits
void bonus() {
  tft.setRotation(2);
  tft.setCursor(0,0);
  tft.fillScreen(COULEUR_FOND);
  tft.setTextColor(ILI9341_RED);
  tft.setTextSize(3);
  tft.println("\n    Jeu de\ncasse-briques\navec joystick\n");
  tft.setTextColor(ILI9341_CYAN);
  tft.setTextSize(2);
  tft.println("\n   Gautier Darchen");
  tft.println("\n    Enora GICQUEL");
  tft.println("\n   Alexandre HUAT");
```

```
  tft.println("\n    Romain JUDIC");
  delay(3000);
  choixMenu = MENU;
}

// Initialisation des positions de la balle et du paddle
void initialisationPosition() {
  xBalle = largeur / 2;
  yBalle = hauteur / 3;
  xPaddle = largeur / 2;
  yPaddle = hauteur / 30;

  vxBalle = 0;
  vyBalle = -2 * choixNiveau;
}

// Fonctions dessinant les vies
void initVie(int nb) {
  for (int i = 0; i < nb; i++) {
   dessinerVie(10 + i * 20, hauteur - 10, false);
  }
}

// Fonction lançant le casse-brique
void casseBrique() {
  initVie(NB_VIE_INIT);
  vieRestante = NB_VIE_INIT;
  initialisationPosition();
  score = 0;
  majScore();
  fin = false;
  while (!fin) {
   affichage(true);
   mouvementBalle();
   xJoyst = analogRead(JOYST_X);
   mouvementPaddle();
   affichage(false);
   delay(20);
  }
  afficherFinPartie();
  delay(5000);
  choixMenu = MENU;
}

// Fonction définissant la position du paddle on focntion de
l'utilisation du joystick
void mouvementPaddle() {
  // en dessous d'un certain seuil, on considère que le joystick ne
bouge pas
  if (abs(xJoyst-JOYST_XMILIEU)>JOYST_PLAGEX/5) {
   // vitesse du paddle proportionnelle à l'amplitude du mvmnt du
joystick
   xPaddle = xPaddle-5*(xJoyst-JOYST_XMILIEU)/(JOYST_PLAGEX);
   if (xPaddle-LARGEUR_PADDLE/2 < 0) {
         xPaddle = LARGEUR_PADDLE/2;
   }
   else if (xPaddle+LARGEUR_PADDLE/2 > largeur) {
```

```
          xPaddle = largeur-LARGEUR_PADDLE/2;
    }
   }
}

// Fonction définissant la position de la balle après mouvement
void mouvementBalle() {
  yBalle = yBalle+vyBalle;
  xBalle = xBalle+vxBalle;

  // si la balle arrive sur le paddle
  if (yBalle-RAYON_BALLE <= yPaddle+HAUTEUR_PADDLE && vyBalle < 0
   && xBalle-ADD_CONTACT_ZONE <= xPaddle+LARGEUR_PADDLE/2
   && xBalle+ADD_CONTACT_ZONE >= xPaddle-LARGEUR_PADDLE/2) {
   paddleCollision();
   yBalle = yPaddle+HAUTEUR_PADDLE+RAYON_BALLE+1;
  }

  // si la balle touche le haut de l'ecran
  else if (yBalle+RAYON_BALLE >= hauteurJeu) {
   verticalCollision();
   yBalle = hauteurJeu - RAYON_BALLE - 1;
  }

  // si la balle touche le bas de l'ecran
  else if (yBalle - RAYON_BALLE <= 0) {
   vieRestante--;
   dessinerVie(10 + vieRestante * 20, hauteur - 10, true);
   if (vieRestante == 0) {
   fin = true;
   }
   else {
   initialisationPosition();
   }
  }

  // si la balle touche un cote de l'ecran
  if (xBalle - RAYON_BALLE <= 0){
   horizontalCollision();
   xBalle = RAYON_BALLE;
  }
  else if (xBalle + RAYON_BALLE >= largeur) {
   horizontalCollision();
   xBalle = largeur - RAYON_BALLE;
  }

  // collision briques
  int l, l1, l2, c, c1, c2;
  int deltaYRadius, deltaXRadius;
  if (vyBalle > 0 ) {
   deltaYRadius = RAYON_BALLE;
  }
  else {
   deltaYRadius = -RAYON_BALLE;
  }
  if (vxBalle > 0 ) {
   deltaXRadius = RAYON_BALLE;
```

```
  }
  else {
   deltaXRadius = -RAYON_BALLE;
  }
  if (yBalle + deltaYRadius > hauteurJeu - ESPACE_NOIR - LIGNES *
HAUTEUR_BRIQUE && yBalle + deltaYRadius < hauteurJeu - ESPACE_NOIR) {
   // collision verticale
   l = (hauteurJeu - ESPACE_NOIR - (yBalle + deltaYRadius) ) /
HAUTEUR_BRIQUE;
   c1 = (xBalle - ADD_CONTACT_ZONE) / largeurBrique;
   c2 = (xBalle + ADD_CONTACT_ZONE) / largeurBrique;
   if (briques[l][c1] == 1 || briques[l][c2] == 1) {
   if (c1 != c2 && briques[l][c1] == 1 && briques[l][c2] == 1) {
         score += 20 * choixNiveau;
   }
   else {
         score += 10 * choixNiveau;
   }
   briques[l][c1] = 0;
   briques[l][c2] = 0;
   if (vyBalle > 0 ) {
         yBalle = hauteurJeu - ESPACE_NOIR - (l+1) * HAUTEUR_BRIQUE -
RAYON_BALLE;
   }
   else {
         yBalle = hauteurJeu - ESPACE_NOIR - l * HAUTEUR_BRIQUE +
RAYON_BALLE;
   }
   verticalCollision();
   tft.fillRect(c1 * largeurBrique, hauteurJeu - ESPACE_NOIR - (l+1) *
HAUTEUR_BRIQUE, largeurBrique, HAUTEUR_BRIQUE, COULEUR_FOND);
   tft.fillRect(c2 * largeurBrique, hauteurJeu - ESPACE_NOIR - (l+1) *
HAUTEUR_BRIQUE, largeurBrique, HAUTEUR_BRIQUE, COULEUR_FOND);
   majScore();
   if (finPartie()) {
         choixNiveau++;
         afficherBriques();
         initialisationPosition();
   }
   }

   // collision horizontale
   l1 = (hauteurJeu - ESPACE_NOIR - yBalle - ADD_CONTACT_ZONE) /
HAUTEUR_BRIQUE;
   l2 = (hauteurJeu - ESPACE_NOIR - yBalle + ADD_CONTACT_ZONE) /
HAUTEUR_BRIQUE;
   c = (xBalle + deltaXRadius) / largeurBrique;
   if (briques[l1][c] == 1 || briques[l2][c] == 1) {
   if (l1 != l2 && briques[l1][c] == 1 && briques[l2][c] == 1) {
         score += 20 * choixNiveau;
   }
   else {
         score += 10 * choixNiveau;
   }
   briques[l1][c] = 0;
   briques[l2][c] = 0;
   if (vxBalle > 0 ) {
```

```
            xBalle = c * largeurBrique - RAYON_BALLE;
    }
    else {
            xBalle = (c+1) * largeurBrique + RAYON_BALLE;
    }
    horizontalCollision();
    tft.fillRect(c * largeurBrique, hauteurJeu - ESPACE_NOIR - (l1+1) *
HAUTEUR_BRIQUE, largeurBrique, HAUTEUR_BRIQUE, COULEUR_FOND);
    tft.fillRect(c * largeurBrique, hauteurJeu - ESPACE_NOIR - (l2+1) *
HAUTEUR_BRIQUE, largeurBrique, HAUTEUR_BRIQUE, COULEUR_FOND);
    majScore();
    if (finPartie()) {
            choixNiveau++;
            afficherBriques();
            initialisationPosition();
    }
    }
   }

   if (vyBalle < 1 && vyBalle > -1) {
    if (vyBalle < 0) {
    vyBalle = -1;
    }
    else {
    vyBalle = 1;
    }
   }
}

// Fonction gerant les collisions de la balle avec le paddle
void paddleCollision() {
   vyBalle = -vyBalle * FACTEUR_ACCELERATION;
   double oldAngle = atan(vxBalle/vyBalle);
   double radius = vyBalle/cos(oldAngle);
   double newAngle =
oldAngle+((xBalle-xPaddle)/(LARGEUR_PADDLE/2.0))*(M_PI/2-M_PI/6);
   vxBalle = (sin(newAngle)*radius);
   vyBalle = (cos(newAngle)*radius);
}

// Fonction gerant les collisions verticales de la balle
void verticalCollision() {
   vyBalle = -vyBalle;
   if (abs(vyBalle * FACTEUR_ACCELERATION) >= 1) {
    vyBalle = vyBalle * FACTEUR_ACCELERATION;
   }
}

// Fonction gerant les collisions horizontales de la balle
void horizontalCollision() {
   vxBalle = -vxBalle;
   if (abs(vxBalle * FACTEUR_ACCELERATION) >= 1) {
    vxBalle = vxBalle * FACTEUR_ACCELERATION;
   }
}

// Fonction permettant de mettre à jour l'affichage du score
```

```
void majScore() {
  tft.fillRect(largeur - 2 * largeur / 3, hauteur - BANDE_AFFICHAGE, 2 *
largeur / 3, BANDE_AFFICHAGE, COULEUR_FOND_AFFICHAGE);

  // on met le texte à l'endroit
  tft.setRotation(2);
  tft.setCursor(0,0);

  // affichage score
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.print("Score ");
  tft.println(score);

  // on remet l'affichage à l'endroit
  tft.setRotation(0);
}

// Fonction dessinant un coeur aux coordonnés indiquées, si clean est
vrai
// il est dessiné de la couleur du fond afin de l'effacer
void dessinerVie(int x, int y, boolean clean) {
  uint16_t couleur;
  if (clean) {
   couleur = COULEUR_FOND_AFFICHAGE;
  }
  else {
   couleur = ILI9341_RED;
  }
  tft.fillCircle(x+6, y, 4, couleur);
  tft.fillCircle(x-2, y, 4, couleur);
  tft.fillTriangle(x-6, y, x+10, y, x+2, y-10, couleur);
}

// Fonction détectant si toutes les briques ont été détruites
boolean finPartie(){
  boolean res = true;
  int i,j;
  for (int i=0; i<LIGNES; i++){
   for (int j=0; j<COLONNES; j++){
   if (briques[i][j]==1){
         res = 0;
   }
   }
  }
  return res;
}

// Fonction affichant le score à la fin de la partie
void afficherFinPartie(){
  tft.fillRect(0,0,largeur,hauteur,ILI9341_YELLOW);
  // on met le texte à l'endroit
  tft.setRotation(2);
  tft.setCursor(0,0);

  tft.setTextColor(ILI9341_BLACK);
  tft.setTextSize(5);
```

```
  tft.println();
  tft.println(" Score :\n\n");
  tft.print("   ");
  tft.println(score);
  tft.setTextWrap(false);
}

// Fonction dessinant la balle et le paddle, si clean est vrai ils sont
// dessinés de la couleur du fond afin de les effacer
void affichage(boolean clean) {
  tft.fillCircle(xBalle, yBalle, RAYON_BALLE, (clean ? COULEUR_FOND :
COULEUR_BALLE));
  tft.fillRect(xPaddle-LARGEUR_PADDLE/2, yPaddle, LARGEUR_PADDLE,
HAUTEUR_PADDLE, (clean ? COULEUR_FOND : COULEUR_PADDLE));
}

// Fonction qui cree un tableau de briques de LIGNES*COLONNES cases
void creerBriques(){
  for (int i=0 ; i<LIGNES ; i++){
   for (int j=0 ; j<COLONNES ; j++){
   briques[i][j]=1;
   }
  }
}

// Fonction qui affiche les briques à l'écran
void afficherBriques(){
  uint16_t couleur;
  creerBriques();
  for (int i=0; i<LIGNES; i++){
   for (int j=0; j<COLONNES; j++){
   couleur = couleurCase();

tft.drawRect(j*largeurBrique,hauteurJeu-ESPACE_NOIR-(i+1)*HAUTEUR_BRIQUE
,largeurBrique,HAUTEUR_BRIQUE,COULEUR_FOND);

tft.fillRect(j*largeurBrique+1,hauteurJeu-ESPACE_NOIR-(i+1)*HAUTEUR_BRIQ
UE+1,largeurBrique-1,HAUTEUR_BRIQUE-1,couleur);
   }
  }
}


// Fonction qui retourne au hasard une des couleurs prédéfinies de
l'écran
uint16_t couleurCase(){
  int i=random(0,17);
  return couleurs[i];
}
```