



**SUPINFO**  
International University

INSTITUTE OF INFORMATION TECHNOLOGY

## ***Class Group Project Architecture***

Delivery

---

Louis CHEVALIER & Valentin ORBAN  
Documentation

Version 1.0

Last update: 06/06/2013

Use: Students

Author: Samuel CUELLA

**Conditions d'utilisations :** SUPINFO International University vous permet de partager ce document. Vous êtes libre de :

- Partager — reproduire, distribuer et communiquer ce document
- Remixeur — modifier ce document

**A condition de respecter les règles suivantes :**

Indication obligatoire de la paternité — Vous devez obligatoirement préciser l'origine « SUPINFO » du document au début de celui-ci de la même manière qu'indiqué par SUPINFO International University – Notamment en laissant obligatoirement la première et la dernière page du document, mais pas d'une manière qui suggérerait que SUPINFO International University vous soutiennent ou approuvent votre utilisation du document, surtout si vous le modifiez. Dans ce dernier cas, il vous faudra obligatoirement supprimer le texte « SUPINFO Official Document » en tête de page et préciser notamment la page indiquant votre identité et les modifications principales apportées.

En dehors de ces dispositions, aucune autre modification de la première et de la dernière page du document n'est autorisée.

**NOTE IMPORTANTE :** Ce document est mis à disposition selon le contrat CC-BY-NC-SA Creative Commons disponible en ligne <http://creativecommons.org/licenses> ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA modifié en ce sens que la première et la dernière page du document ne peuvent être supprimées en cas de reproduction, distribution, communication ou modification. Vous pouvez donc reproduire, remixer, arranger et adapter ce document à des fins non commerciales tant que vous respectez les règles de paternité et que les nouveaux documents sont protégés selon des termes identiques. Les autorisations au-delà du champ de cette licence peuvent être obtenues à [support@supinfo.com](mailto:support@supinfo.com).

© SUPINFO International University – EDUCINVEST - Rue Ducale, 29 - 1000 Brussels Belgium . [www.supinfo.com](http://www.supinfo.com)

### Table of contents



<b>1</b>	<b>GROUP SUMMARY .....</b>	<b>4</b>
1.1	<i>GROUP MEMBERS .....</i>	<i>4</i>
<b>2</b>	<b>PROJECT REPORT.....</b>	<b>5</b>
<b>3</b>	<b>SOLUTION MANUAL .....</b>	<b>5</b>
<b>4</b>	<b>TECHNICAL DOCUMENTATION .....</b>	<b>5</b>

### 1 GROUP SUMMARY

#### 1.1 GROUP MEMBERS

Campus: **Troyes**

Class: **A.Sc.2**

ID Open Campus	Last Name	First Name	Photo
214594	CHEVALIER	LOUIS	
215037	ORBAN	VALENTIN	

## 2 PROJECT REPORT

---

Louis Chevalier – Chef de projet  
Valentin Orban - Développeur

## 3 SOLUTION MANUAL

---

*Note: Use the part as a documentation for users/administrators/developers as specified in the “Project Guidelines” available on courses.*

*Language: English/French, at your option.*

## 4 TECHNICAL DOCUMENTATION

---

### 4.1 DEVELOPING ON NES

---

Cette première partie a pour but de présenter les outils utilisés, où les trouver, et de présenter les restrictions techniques autour desquelles le projet doit être construit.

- Nous utilisons le langage C. Pour compiler, nous utilisons cc65 (<http://cc65.github.io/cc65/>)
- Pour éditer les sprites, YY-chr (<http://www.romhacking.net/utilities/119/>)
- En émulateur, FCEUX (<http://www.fceux.com/web/download.html>)

Au niveau du développement même, pour des soucis d’espace de stockage sur une ROM, certains concepts ne sont pas utilisés, comme le retour des fonctions ou l’usage de paramètres. Toutes les fonctions sont déclarées ainsi : void fonction(void) ;

Pour les même raisons, seules des variables globales sont utilisées.

La NES est une console 8bits, aussi toutes les valeurs sont du type unsigned char.

Le fichier reset.s est récupéré de <https://nesdoug.com/>, lien donné par le sujet. Il permet d’organiser la ROM. Il a aussi été modifié pour importer des variables de nephoid.c, afin de les altérer dans le label NMI, dont le code est exécuté à chaque rendu graphique du hardware (indépendant du code). L’interruption exécutant le code NMI est appelée 60 fois par seconde. En utilisant cela, on peut créer une horloge 60fps pour une boucle de jeu.

L’affichage de la NES est séparé en 4 écrans, un seul visible à la fois. Nous n’utiliserons, pour le casse-brique, qu’un seul écran. Au niveau du background (système d’affichage par blocs de 8\*8px immobiles), l’adresse du bloc du coin haut gauche est \$2020 (\$2000 est une ligne non représentée sur l’écran physique). Chaque ligne possède 32 blocs, et un écran compte 30 lignes.

L’affichage des sprites (système d’affichage de blocs 8\*8px au pixel prêt) est restreint par le nombre de sprites affichables, 8 par ligne simultanément. Si plus, les moins prioritaires (on ne s’attardera pas sur la notion de priorité des sprites car le jeu ne dépasse jamais les 8 sprites par ligne) ne sont pas affichés. Les coordonnées du pixel du coin haut gauche sont (0,8) ((0,0) étant sur la ligne

absente de l'écran physique). Un sprite nécessite 4 octets d'information : sa position Y, son emplacement dans la mémoire (on donnera juste l'offset auquel le sprite est défini dans le .chr), ses attributs (flipping vertical/horizontal et quelle palette de couleur utiliser) et enfin sa position X.

Le fonctionnement du hardware impose d'avoir une routine à effectuer à chaque frame du jeu dans le code : attendre le blanc vertical (temps de latence entre deux rendus), écrire à l'écran, remettre à zéro le scrolling).

La NES possède 56 couleurs. On définit une palette de colorsets, qui sont les ensembles de couleurs disponibles pour le rendu. On définit également une Attribute Table, qui détermine quel colorset est utilisé par quel partie de l'écran.

Une adresse particulière (\$4016) permet de récupérer les états des boutons de la manette 1. L'adresse suivante correspond à la manette 2. Après avoir demandé la lecture des états (en écrivant 1 puis 0) on peut les lire un par un (l'information est portée par le bit de poids faible).

## 4.2 INPUT HANDLING

Pour la gestion des inputs, la NES possède deux registres : \$4016 et \$4017 (\$ signifie adresse exprimée en hexadécimal). Le premier registre pour la manette 1, le second pour la manette 2. Lorsqu'on souhaite récupérer les informations d'une manette à un instant t, on procède en 2 étapes : écrire dans le registre correspondant les valeurs 1 puis 0 pour indiquer que nous sommes prêts à lire le registre. Ensuite, on récupère les statuts des boutons un par un. Le bit contenant l'information est celui de poids faible. On va donc faire un décalage vers la gauche de ce bit avant de le stocker dans un octet en mémoire.

Dans le code, on déclare inputStatus, comme un caractère non signé (codé sur 8 bits avec des valeurs comprises entre 0 et 255). Une fois les statuts récupérés avec la méthode décrite précédemment, on peut définir les instructions à effectuer en fonction des inputs (dans notre cas, inverser l'état de pause si START est enfoncée, lancer la balle pour A, et déplacer le paddle avec les flèches gauches et droite. Les statuts des boutons B, SELECT, haut et bas sont récupérés également, dans le cas où de nouvelles mécaniques seraient implémentées.

## 4.3 COLLISION DETECTION

Pour détecter si deux éléments sont en collision, on effectue des comparaisons logiques sur leur coordonnées. Les portions de codes concernées dans nephoid.c sont relativement claires quant à la détection de cette collision.

La détection de la collision de deux sprites (foreground) ne pose donc pas de problème. Techniquement, la NES ne peut afficher que 8 sprites par lignes, il faut donc représenter les briques via arrière-plan.

Pour détecter une collision entre une brique (qui est un élément du background) et la balle (foreground) nous avons opté pour la sauvegarde de toutes les coordonnées possibles d'une brique. En effet, notre jeu est conçu pour utiliser une surface de 8\*8 briques, sur la partie supérieure de l'écran. On imagine donc une grille 8\*8 de coordonnées d'arrière-plan (représentées par des adresses, dont celle du coin haut gauche est \$2020). Une seconde matrice sera utilisée pour préciser si une brique i est affichée ou non. Cette matrice est variable, elle sera mise à jour quand la balle détruira des briques. Enfin, pour détecter les collisions, on va créer une dernière matrice de coordonnées, calquée sur le foreground (si les briques étaient représentées sur le foreground, cette matrice contiendrait leurs coordonnées exactes). Les coordonnées du coin haut gauche sont (0, 8).

On va utiliser cette matrice pour déterminer s'il y a collision avec la balle.

### 4.4 CROSS-COMPILING

---

Le sujet demande de documenter le cross-compiling. Il est pourtant inexistant puisqu'il n'y a qu'une seule plateforme cible, la NES. On utilise cc65 pour compiler, et obtenir une ROM au format .nes.