Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Licence en Informatique

Thème

Etude des méthodes et techniques de l'émulation

Réalisé par :

- **BENABADJI Adil**
- **DIB Abdelkrim**

Présenté le 9 Juin 2014 devant la commission d'examination composée de MM.

- *BENMANSOUR F.* *(Examinatrice)*
- *BENTAALLAH A.* *(Examinateur)*
- *BRIKCI NIGASSA A.* *(Encadreur)*

Année universitaire : 2013-2014

Remerciements

Tout d'abord nous remercions notre encadreur M. BRIKCI NIGASSA pour son aide et ses précieux conseils qui nous ont permis de mener à bien nos recherches, et de bien structurer notre document.

Nous remercions aussi les nombreuses personnes qui ont mis à disposition sur Internet les documents nécessaires à l'étude de l'émulation.

Nous remercions également les membres du jury, M. BENTAALLAH et M^{me} BENMANSOUR, pour avoir accepté de juger ce modeste travail.

Dédicaces

Nous dédions notre travail à toute personne s'intéressant à l'émulation et à la compilation, ainsi qu'à tous les humains, même les méchants, ces méchants hackers qui ont réussi à bidouiller les machines et à créer des émulateurs. Grâce à eux, le domaine s'est développé et démocratisé.

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Pour faire fonctionner un programme qui n'est pas à la base ciblé pour une tierce machine, on a souvent recours à l'émulation. Mais de nos jours, un smartphone peut avoir un processeur cadencé jusqu'à 2.2 GHz par cœur et reste incapable d'émuler convenablement une machine cadencée à 200 MHz, c'est-à-dire que bien qu'une machine soit dix fois plus rapide qu'une autre, elle reste incapable de bien l'émuler. Pourquoi ? Que faut-il faire ?

La première solution qui vient à l'esprit pour résoudre ce problème est de changer la façon de faire, et justement, ce travail consiste à tester différentes approches pour émuler une machine dans le but d'augmenter ses performances. Certes la technologie s'améliore de jour en jour, et si hier les smartphones avaient des processeurs à 1 GHz, aujourd'hui à 2 GHz, alors demain on aura certainement des smartphones avec un processeur cadencé à 3 GHz qui seront capables d'émuler plus de machines, alors pourquoi ne pas simplement patienter ?

On assiste ces derniers temps à la propagation et la démocratisation du DIY (Do It Yourself) (ex: Arduino) et des ordinateurs bon marché (ex: Raspberry Pi), qui sont souvent utilisés comme Media Center car ils ne sont pas assez puissants pour faire tourner autre chose. Il serait intéressant de les utiliser comme une collection de vieilles consoles par exemple. En d'autres termes, malgré les avancées technologiques, les appareils à processeurs de faibles puissances existerons et continueront d'exister, et le fait qu'ils ne soient pas maintenus, pas utilisés pour développer des logiciels, pas utilisé à leurs pleines puissances, rend l'émulation incontournable, non seulement pour faire revivre les vieilles machines, mais aussi pour apporter à l'appareil une bibliothèque de logiciels et de jeux plus intéressante.

Première partie :
Synthèse bibliographique

Chapitre 1 : Définition et histoire de l'émulation

1.1 Qu'est-ce qu'un émulateur ?

La définition standard de l'émulation est : « la disposition qui porte à égaler ou à surpasser quelqu'un [1] ou quelque chose ». Un émulateur est celui qui émule quelqu'un ou quelque chose.

En informatique, l'émulation consiste à émuler le comportement d'un élément matériel par un logiciel ou un autre matériel, ou émuler le comportement d'un élément logiciel par un autre matériel ou logiciel [2]. En d'autres termes, un émulateur fournit une couche d'abstraction permettant la portabilité des programmes vers des machines qui ne sont pas à la base destinée à l'exécuter.

1.2 Fonctionnement d'un émulateur

Dans ce mémoire, le type d'émulation abordé est l'émulation par logiciel. Chaque ordinateur possède plusieurs composants qui forment un circuit électronique. Le but de l'émulateur est de retranscrire ce circuit. Pour cela, le comportement de chacune de ses composantes doit être réécrit dans un algorithme. Ces programmes assemblés se comportent comme la machine d'origine, ces composants émulés sont « virtuels ».

1.3 Que peut-on émuler ?

Selon la thèse de Church-Turing: «Les problèmes résolubles par une machine de Turing Universelle sont exactement les problèmes résolubles par un algorithme ou par une méthode concrète de calcul», ou encore: « Si un algorithme existe pour faire un calcul, alors ce même calcul peut aussi être calculé par une machine de Turing » [3]. Ce qui peut vouloir dire que les calculs faits par le matériel d'une machine peuvent être calculés avec un langage de programmation. Par contre, pour qu'une machine de Turing Universelle puisse émuler une autre machine de Turing, il faut qu'elle ait suffisamment de mémoire, le reste n'est qu'une question de rapidité d'exécution et de synchronisation.

1.4 Les machines virtuelles

La différence entre les machines virtuelles et les émulateurs est le fait qu'une machine virtuelle simule le minimum de composants d'un ordinateur hôte. Pour qu'un système d'exploitation invité puisse fonctionner correctement, on laisse la plupart des opérations se dérouler sur le matériel réel pour assurer la performance [4]. La virtualisation est donc

normalement plus rapide que l'émulation mais le système réel doit avoir une architecture identique au système invité.

1.5 Histoire de l'émulation

Selon la définition d'un émulateur, l'émulation a toujours existé. En effet, le simple fait de porter un programme vers un autre environnement peut être assimilé à une émulation.

Le premier émulateur défini tel quel a vu le jour en 1962 par IBM [5]. Un utilisateur réussit à rendre les programmes d'IBM 1401 compatibles avec son IBM 705 [6]. IBM comprend l'importance de la compatibilité descendante et fait des recherches pour une solution logicielle permettant de simuler le comportement de ses sept machines les plus populaires de l'époque, mais le résultat n'était pas satisfaisant.

IBM ne lâche pas l'affaire et comprend que pour avoir des performances acceptables, il faut qu'il y ait une certaine ressemblance dans l'architecture des deux machines. Dans l'équipe de recherche, un certain Larry Moss parvient à mettre au point une solution n'affichant aucune perte de performance, le premier émulateur était né [6].

Chapitre 2 : Types d'émulation

2.1 Emulation de bas niveau (LLE : Low Level Emulation)

L'émulation LLE repose sur une idée: traduire le jeu d'instructions d'une architecture source vers celui d'une architecture cible, souvent appelée traduction du code ou 'binary translation' [7].

Afin de parvenir à émuler un système en LLE, il faut avoir une idée bien approfondie sur le fonctionnement de la machine source. Ensuite, le choix de la ou les méthodes à utiliser est indispensable. En effet, dans la mesure où chaque processeur dispose d'un environnement, ou d'un contexte extérieur différent, les possibilités d'émulation sont donc aussi variées et différentes.

2.1.1 L'interprétation linéaire

L'interprétation linéaire d'un programme lit instruction par instruction, décode au fur et à mesure les « opcodes » lus. Elle traduit le code compilé d'une machine source vers un code qui sera compilé et exécuté dans la machine cible.

```
while (emulation_en_marche)
{
        switch (memoire [compteur_ordinal++])
        {
                case OPCODE1:
                        opcode1();
                        break;
                case OPCODE2:
                        opcode2();
                        break;
                …
                case OPCODEn:
                        opcoden();
                        break;
        }
}
```

Figure 0.1 : principe de l'interprétation. Adapté de Figure 3. de [2]

Le principal avantage de cette méthode est qu'elle est facile à implémenter, indépendante de la cible et utile pour le débogage des programmes de la machine source. Par contre, l'exécution des programmes est lente et gourmande en temps CPU à cause du traitement effectué pour chaque « opcode » lu ou relus pendant l'exécution.

2.1.2 La recompilation

La recompilation part du principe que si toutes les architectures de processeur ont bien des jeux d'instructions différents, la plupart des instructions ont des instructions (ou combinaisons d'instructions) équivalentes d'une architecture à l'autre [8].

Cette méthode a pour but de corriger les défauts de l'interprétation, c'est-à-dire éviter de répéter le traitement déjà fait. Par contre, l'implémentation de l'émulateur reste dépendante de l'architecture de la machine cible et fait souvent appel au langage assembleur.

La recompilation peut être faite de deux façons:

2.1.2.1 Recompilation statique

La recompilation statique n'est rien d'autre que la compilation, seulement le code source est écrit en langage machine de la machine à émuler (source) et le langage compilé doit être le langage machine de la machine cible [9]. D'ailleurs, certaines étapes de la compilation n'y sont pas. En d'autres termes, c'est la phase génération du code et optimisation de la compilation.

La méthode semble être efficace car contrairement aux autres méthodes, la traduction s'effectue durant la recompilation; cela donne un gain de performance. Cependant, les programmes compatibles avec cette technique sont des cas particuliers dans le sens où ils ne sont pas automodifiables (absence de pagination). La seule solution pour résoudre le problème des programmes qui utilisent la pagination est de déterminer à l'avance le déroulement du programme, ce qui est impossible.

2.1.2.2 Recompilation dynamique

La recompilation dynamique est un mélange entre l'interprétation et la recompilation statique. Elle permet de recompiler des parties d'un programme durant son exécution.

Chaque bloc d'instruction est compilé une fois pour toute, et est réutilisé plus tard si jamais l'exécution revient sur ce bloc. Elle est souvent nommée « dynarec » [7]. Le concept ressemble à celui de la compilation à la volée (JIT : Just In Time).

Cette méthode garde l'équilibre entre la performance et la compatibilité des programmes. Les « opcodes » lus sont recompilés au fur et à mesure et si jamais une pagination s'effectue l'émulateur invalide le bloc et recompile les nouveaux « opcodes » évitant ainsi le plantage.

2.2 Emulation de haut niveau (HLE : High Level Emulation)

Le fonctionnement d'un émulateur dépend de l'environnement d'exécution du programme. Dans ce type d'émulation, la machine à émuler (source) a la même architecture que la machine l'émulant (cible). Cependant, elles diffèrent dans la plateforme exécutant les programmes. Afin de parvenir à émuler la machine source, la machine cible substitue les bibliothèques utilisées dans la plateforme source. Exemple : « Wine » substitue les bibliothèques de Windows (.dll), ainsi les fonctions telles que celles de l'API graphique DirectX sont remplacées par des fonctions équivalentes OpenGL.

L'émulation se fait très rapidement vu que le programme en question est exécuté nativement. Mais cela peut causer des problèmes de synchronisation. Autre avantage, comme l'émulation ne se fait qu'au niveau des fonctions et des API, pas besoin au concepteur de l'émulateur de connaitre le fonctionnement du matériel.

Deuxième partie : Application

Chapitre 3 : Présentation de la NES

3.1 Introduction

Afin de parvenir à émuler et à comparer les différentes approches, la NES (Nintendo Entertainment System) a été choisie. Crée par Nintendo, c'est une console sortie en 1983 au Japon sous le nom de Famicom et en 1985 aux Etats-Unis, elle n'est arrivée qu'en 1986 en Europe [10].

Figure 0.1 : La consolle NES et de sa manette.

La NES est dotée d'un processeur 8-bit, un CPU 2A03 qui n'est rien d'autre qu'une variante du 6502. Il est assez puissant pour faire fonctionner les programmes mais reste incapable de générer les graphismes. Afin d'y remédier, une autre puce est mise à disposition, la PPU 2C02, responsable du calcul et de l'affichage des graphismes. Les deux puces ont été fabriquées par Ricoh. La NES contient 2 Ko de RAM, 16 Ko de VRAM et un espace de stockage spécial d'une capacité de 256 octets nommé SPR-RAM [11]. Quant aux programmes, ils sont stockés dans une ROM dans la cartouche.

3.2 CPU (2A03)

Cadencé à 1,7897725 MHz pour la version NTSC et 1,773447 MHz pour la version PAL, la différence avec le 6502 est l'absence du mode décimal dans le 2A03 [11]. Le CPU de la NES peut aussi servir comme pAPU (pseudo Audio Process Unit) permettant de générer et de traiter le son.

3.2.1 Les registres

Comme tout processeur, le 2A03 a des registres dont la taille est variable selon le registre :

- *Compteur Ordinale* : registre 16 bits, contient l'adresse de la prochaine instruction à exécuter, sa valeur varie dans la plage d'adresse $0000~$FFFF.
- *Pointeur de Pile* : registre 8 bits, sert à stocker l'index de la tête de pile, varie dans l'intervalle $00~$FF.
- *Registre d'Etat* : registre 8 bits, stocke certaines informations sur le résultat de la dernière instruction effectuée sous forme de drapeaux. Il peut aussi être utilisé pour contrôler certaines opérations.
- *Registre Accumulateur* : c'est un registre général de 8 bits, utilisé pour stocker le résultat des instructions utilisant l'unité arithmétique et logique du processeur.
- *Registre X et Registre Y* : deux registres généraux de taille 8 bits, souvent utilisés comme index pour certains modes d'adressage, ils peuvent aussi être utilisés pour le contrôle de boucles.

3.2.2 L'accès à la mémoire

Pour avoir accès à la mémoire, le CPU utilise des bus: un bus d'adresse de 16 bits, ce qui implique que le CPU peut adresser jusqu'à 2^{16} adresses, c'est-à-dire qu'il peut adresser une mémoire de 64 Ko, de l'adresse $0000 à $FFFF; un bus de 8 bits pour transporter les données; et un bus de contrôle pour informer le composant (RAM, ROM, registres,…) si l'opération en cours est une lecture ou une écriture.

La mémoire à laquelle peut accéder le CPU est divisée principalement en trois parties: la ROM (stockée dans la cartouche); la RAM du CPU et les registres d'entrée/sortie. La RAM se situe de l'adresse $0000 à $0800, celle-ci a une plage $0000~$00FF nommée « page zéro » permettant un accès plus rapide, et une plage $00FF~$01FF définissant le segment de pile [12]. Les données situées dans $0800~$1FFF sont des miroirs de la plage $0000~$0800 reflétée trois fois, c'est une sorte d'alias, si par exemple une donnée est écrite à l'adresse $0000, elle est aussi écrite à l'adresse $0800, $1000 et $1800. Les registres d'entrées/sorties se trouvent dans $2000~$401F. Les mémoires se trouvant dans $2007~$3FFF reflètent les registres $2000~$2007 tous les 8 octets. La SRAM est une

mémoire permettant la sauvegarde, elle se trouve dans la cartouche dans la plage $6000~$8000 et est optionnelle. La ROM, aussi appelée PRG-ROM, est la mémoire où est stocké le programme (le jeu). Le CPU de la NES peut adresser jusqu'à 32 Ko de PRG-ROM, de l'adresse $8000 à $FFFF. Cette plage est divisée en deux pages d'une taille de 16 Ko chacune.

L'accès à la mémoire se fait par un « mapper ». Un mapper se trouve dans la cartouche et peut différer d'un jeu à l'autre. C'est un circuit, qui relie les composants, permettant ainsi d'augmenter la mémoire. Il est aussi responsable de la pagination.

3.2.3 Les interruptions

Les interruptions sont des évènements que le processeur doit gérer. La NES a trois types d'interruptions :

- *NMI (Non-Maskable Interrupt)* : Ce sont des interruptions générées par la PPU dès qu'un VBlank se produit à la fin de chaque frame (voir p.24). Après génération, le processeur interrompt le cycle courant de l'exécution et exécute une sous-routine dont l'adresse est stockée dans $FFFA et $FFFB ($FFFA → les 8 premiers bits, $FFFB → les 8 derniers).
- *IRQ (Interrupt Request)* : Elles peuvent être masquées. Elles sont générées soit par le matériel soit par une instruction BRK. Dès qu'une IRQ se produit, le processeur fait un saut à l'adresse se trouvant dans $FFFE et $FFFF
- *Reset* : Elles se produisent au lancement de la machine ou dès que l'utilisateur appuie sur le bouton de réinitialisation « Reset ». Le CPU, fait un saut à l'adresse contenue dans $FFFC et $FFFD.

3.2.4 Les modes d'adressage

Le CPU de la NES compte 56 instructions différentes et 151 opcodes [11]. Grâce aux modes d'adressage, une instruction peut avoir différentes implémentations pour passer les paramètres. Le 2A03 a en tout 12 modes d'adressage.

Exemple pour l'instruction INC (Increment) sur la mémoire dont l'adresse est $0075, sachant que le registre X est égal à 1 :

Tableau 0.1 L'instuction INC et ses différents modes d'adressage

Mode d'adressage	Mnémonique	Hexadécimal
Adressage page zéro	INC $75	$E6 $75
Adressage indexé page zéro	INC $74, X	$F6 $74
Adressage absolu	INC $0075	$EE $75 $00
Adressage absolu indexé	INC $0074, X	$FE $74 $00

3.3 PPU (2C02)

La NES contient une puce dédiée pour la gestion des graphismes appelée PPU (Picture Processing Unit). Cette puce est la 2C02. Fabriquée par Ricoh, elle peut afficher des images d'une résolution de 256x240 pixels en 50 Hz (50 images/seconde) pour la version PAL, et 256x232 en 60 Hz pour la version NTSC [13]. La PPU dispose d'une mémoire vidéo dédiée nommée VRAM, d'une taille de 16 Ko, mais aussi d'une mémoire SPR-RAM pour stocker les sprites de 256 octets.

3.3.1 Les registres

Les registres de la PPU sont les mémoires auxquelles le CPU a accès à travers les adresses $2000~$2007 et $4014 [14]. Ils remplissent chacun un rôle. D'ailleurs c'est grâce à ces registres que la CPU et la PPU communiquent.

Les registres $2000 et $2001 s'appellent des registres de contrôle. Afin de les utiliser, le développeur n'a droit qu'à l'écriture sur ces deux registres. Ils contiennent des flags contrôlant ainsi le déroulement du programme et l'affichage. Par exemple, si le bit 7 du registre $2000 est mis à 0, cela aura comme effet de désactiver les interruptions NMI. Quant au registre $2002, c'est le registre d'état du PPU. Disponible seulement en lecture, les programmes l'utilisent souvent pour éviter les erreurs. Le registre $2005 sert à changer les coordonnées de la caméra. Les registres restants servent à écrire ou à lire dans les mémoires VRAM et SPR-RAM.

3.3.2 Direct Memory Access (DMA)

Afin de parvenir à transférer des informations à la mémoire SPR-RAM, deux solutions existent. La première est de transférer les informations octet par octet en passant par les étapes suivantes :

1- Ecrire les 8 premiers bits de l'adresse dans le registre $2003 ;
2- Ecrire les 8 derniers bits dans le même registre ;
3- Ecrire l'octet à transférer dans le registre $2004.

Cette méthode est lourde s'il y a plusieurs octets à transférer. Pour y remédier, la deuxième solution consiste à utiliser le Direct Memory Access (DMA). Cette technique copie directement 256 octets de données se trouvant dans la RAM vers la SPR-RAM. Pour l'utiliser, le programme écrit un octet dans le registre $4014, cet octet est alors multiplié par $100 et pris comme adresse de début des données à copier.

3.3.3 VRAM et SPR-RAM

Comme pour le CPU, la PPU a un bus d'adresse de 16 bits permettant d'adresser jusqu'à 64 Ko de VRAM. Dans la NES, la VRAM a une taille de 16 Ko et est structurée. Ainsi, aux adresses $0000~$2000 se trouvent 2 pages nommées « tables des graphismes », où sont stockés les « tiles ». La plage $2000~$3F00 contient les informations nécessaires pour dessiner l'arrière-plan du jeu. Elle est divisée en 4 groupes se trouvant tous dans $2000~$3000. Le reste ($3000~$3F00) n'est qu'un miroir reflétant la précédente plage. Chaque groupe contient $3C0 octets consacrés aux « tables de nommage » suivis de $40 octets pour la « table d'attribution ». La NES ne peut utiliser que deux groupes vu la quantité de mémoire disponible. Pour en utiliser plus, une mémoire supplémentaire doit être fournie à la console par le biais de la cartouche. A l'adresse $3F00~$3F1F sont stockées les palettes utilisées pour colorer les « tiles » se trouvant dans la « table des graphismes » afin de pouvoir les afficher. Les mémoires dont l'adresse est dans la plage $3F20~$3FFF ne sont que des miroirs de la précédente plage. La plage restante $4000~$FFFF est un miroir de toute la plage $0000~$3FFF.

La mémoire SPR-RAM est une mémoire de 256 octets utilisée pour contenir les informations relatives aux sprites. Chaque sprite prend 4 octets, ainsi la SPR-RAM peut contenir jusqu'à 64 sprites.

3.3.4 Palette des couleurs

La NES dispose de 64 couleurs mais dont seulement 52 couleurs sont différentes. En d'autres termes, seuls les 6 premiers bits sont utilisés pour représenter une couleur. De plus, la machine ne peut pas afficher toutes les couleurs en même temps. Les couleurs utilisées sont stockées dans $3F00~$3F1F. Les 16 premières valeurs représentent la palette utilisée par les tiles de l'arrière-plan, nommée « palette de l'arrière-plan ». Les 16 restantes représentent la palette utilisée pour les sprites, nommée « palette des sprites » [15]. Les couleurs ne sont pas définies par un système tel que RGB, YUV,… mais par des index de couleurs internes à la machine.

La valeur de l'adresse $3F00 représente la couleur de l'arrière-plan. Les adresses $3F04, $3F08, $3F0C, $3F10, $3F14, $3F18 et $3F1C sont des miroirs reflétant la case mémoire $3F00, c'est-à-dire que la valeur est dupliquée tous les 4 octets [14]. Mes couleurs stockées dans ces cases mémoires sont prises par les tiles comme étant une couleur transparente.

3.3.5 Table des graphismes

L'arrière-plan comme les sprites sont composés de tiles. Sur la NES, un tile est un carré de 8x8 pixels. Les tables de graphismes, aussi appelées « pattern tables », ont pour rôle de stocker ces tiles qui sont définis dans la plage $0000-$1FFF [14]. Chaque table de graphismes a une taille de $1000 octets soit 4 Ko. Ainsi la NES peut adresser 2 tables de graphismes. Ces tables se trouvent généralement dans une mémoire dans la cartouche, parfois appelée CHR-ROM (Character ROM).

Chaque pixel dans un tile est représenté par deux bits, ces deux bits sont les bits de poids faible sur les quatre utilisés pour identifier la couleur du pixel dans la palette d'attribution.

Chaque tile contient 8x8 pixels, c'est-à-dire 64 pixels. Dans la mémoire, il occupe 128 bits, c'est-à-dire 16 octets. Ainsi, une table des graphismes peut contenir jusqu'à 256 tiles.

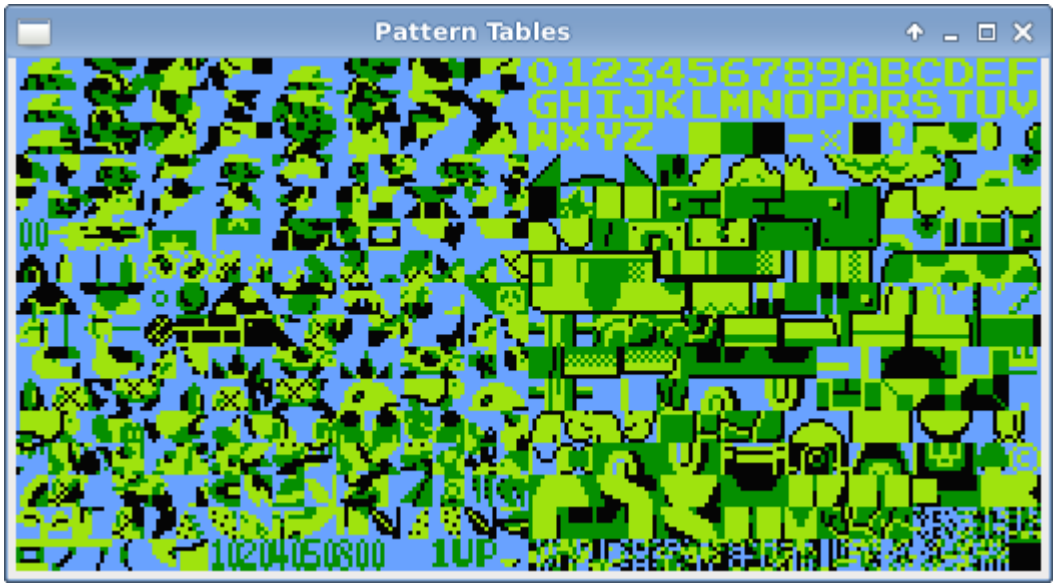

Figure 0.2 : Exemple de table des graphismes issue de Super Mario Bros.

Les données sont structurées de sorte qu'il puisse y avoir deux couches. La première est représentée par les 8 premiers octets contenant le bit de poids faible de chaque pixel. Les 8 octets restants contiennent le deuxième bit de poids faible du pixel.

Tableau 0.2 Différentes combinaisons représentant un pixel dans un tile.

Couche 2	Couche 1	Bits de poids faible du pixel	Couleur selon la palette
0	0	00	Couleur 0
0	1	01	Couleur 1
1	0	10	Couleur 2
1	1	11	Couleur 3

3.3.6 Table de nommage et table d'attribution

Une table de nommage est une matrice 32x30. Chaque cellule pointe vers un tile stocké dans la table des graphismes. La table de nommage couvre ainsi 256x240 pixels. La PPU peut adresser jusqu'à quatre tables de nommage, mais en réalité dans la NES, la

VRAM ne peut en contenir que deux. Pour en utiliser plus, une mémoire supplémentaire doit être mise à disposition dans la cartouche.

Les tables de nommage sont situées à partir des adresses $2000, $2400, $2800 et $2C00. Chaque table prend $3C0 octets (soit 960 octets). Les données sont organisées de façon à ce que le remplissage des pointeurs de tile se fasse de gauche à droite, de haut en bas. C'est-à-dire pour accéder au tile de la ligne 11, colonne 15 de la table #0 on applique la formule :

*((Ligne * 32) + colonne) + adresse début de la table = adresse du tile en question*

Donc on aura : *((11 * 32) + 15) + $2000 = $216F*

Les $40 octets (64 octets) suivant chaque table de nommage représentent la table d'attribution. Chaque table d'attribution est associée avec une table de nommage servant à stocker les informations manquantes pour l'affichage du tile, soit les bits de poids fort des tiles pour identifier les couleurs dans la palette de l'arrière-plan.

Chaque octet de la table d'attribution représente un groupe de 4x4 tiles. Chaque deux bits dans un octet représentent un groupe de 2x2 tiles. Donc, un octet aura une forme 33221100.

Groupe 0		Groupe 1	
Tile $0	Tile $1	Tile $4	Tile $5
Tile $2	Tile $3	Tile $6	Tile $7
Groupe 2		Groupe 3	
Tile $8	Tile $9	Tile $C	Tile $D
Tile $A	Tile B	Tile $E	Tile $F

Figure 0.3 : Groupe de 4x4 tiles.

3.3.7 Fonctionnement des miroirs

Les tables de nommage non utilisées sont des miroirs des autres tables. Il existe quatre types de miroirs [13], les tables #1, #2, #3 et #4 représentent les tables à l'adresse $2000, $2400, $2800 et $2C00 respectivement:

- *Miroir horizontal*: les tables #1 et #2 sont identiques et représentent la première table de nommage, et les tables #3 et #4 représentent la deuxième table qui est utilisée par les jeux à scrolling horizontal. Exemple : *Super Mario Bros.*

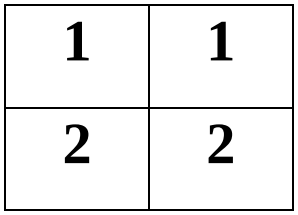

Figure 0.4 : Miroir horizontal.

- *Miroir vertical*: les tables #1 et #3 représentent la première table, les tables #2 et #4 représentent la deuxième, exemple de jeux utilisant ce type de miroir. Exemple: *Donkey Kong*.

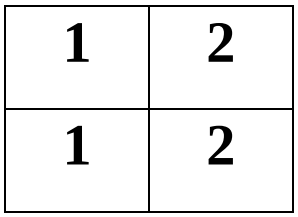

Figure 0.5 : Miroir vertical.

- *Miroir à un seul écran*: les tables #1, #2, #3 et #4 reflètent toutes la même table de nommage. Rares sont les jeux qui l'utilisent.

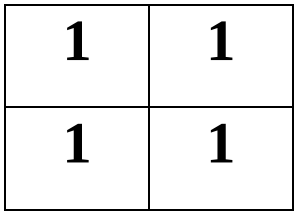

Figure 0.6 : Miroir à un seul écran.

- *Miroir à quatre écrans*: les tables représentent chacune une table de nommage différente. Pour utiliser ce mode, 2 Ko de mémoire additionnelle doivent être fournis dans la cartouche.

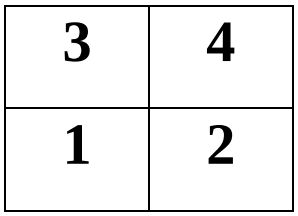

Figure 0.7 : Miroir à quatre écrans.

3.3.8 Sprites

Les animations sur la NES sont faites par le défilement (scrolling) de l'arrière-plan et les mouvements des sprites dans l'écran. L'arrière-plan n'est dessiné qu'une seule fois, le reste est une question de scrolling. Les sprites sont les éléments dynamiques dans l'affichage. La dimension des sprites peut-être 8x8 pixels ou 8x16 pixels [14], et comme pour l'arrière-plan, ils sont composés de tiles stockés dans la table des graphismes. Les informations relatives aux sprites sont situées dans SPR-RAM (voir p.19). Un sprite prend 4 octets dans la mémoire qui contiennent les informations sur ses coordonnées à l'écran, l'index du tile à utiliser, les 2 bits de poids fort pour les couleurs utilisées par le tile (ces couleurs sont issues de la palette des sprites), etc.

Les sprites sont dessinés selon leurs positions dans la mémoire SPR-RAM. Le sprite #0, disponible dans [$00-$03], a la priorité la plus haute. Ainsi, la PPU dessine le sprite #63, ensuite le sprite #62,… jusqu'au sprite #0 assurant la visibilité des sprites à haute priorité.

3.3.9 Scanline, VBlank et HBlank

Dans le téléviseur, les images sont affichées sur l'écran du tube cathodique (CRT : Cathode Ray Tube). Celui-ci dessine les pixels ligne par ligne, de gauche à droite, de haut en bas. Une ligne de pixels est appelée scanline [14]. Une fois la scanline dessinée, le canon à électron doit aller à la prochaine ligne en retournant à gauche pour dessiner la nouvelle ligne : cette période s'appelle HBlank (Horizontal Blank). Quant au VBlank (Vertical Blank), c'est la période nécessaire pour revenir en haut une fois l'écran complété.

Dans la NES, une frame est constituée de 262 scanlines, mais seulement 240 sont affichées à l'écran pour la version PAL, et 232 pour la version NTSC [16].

3.3.10 Scrolling

Le scrolling consiste à faire défiler l'arrière-plan. Cela se fait sur deux axes seulement : horizontalement et verticalement. L'utilisation du scrolling s'effectue par l'écriture successive de deux valeurs dans le registre $2005 représentants le scroll vertical et le scroll horizontal. Ces deux valeurs peuvent être vues comme les coordonnées x et y de la caméra.

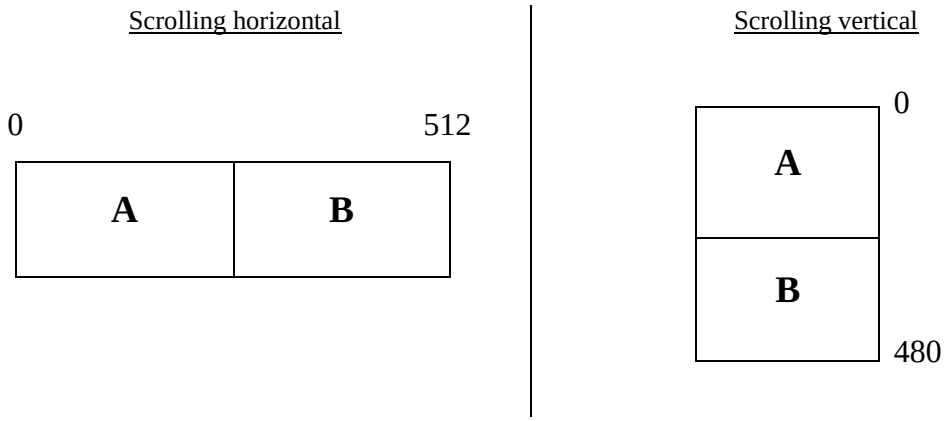

Figure 0.8 : Scrolling vertical et horizontal.

La table de nommage « A » est spécifiée via les deux bits de poids faible du registre $2000, et « B » est la table de nommage qui vient après (cela dépend du miroir utilisé).

Table de nommage #2 ($2800)	Table de nommage #3 ($2C00)
Table de nommage #0 ($2000)	Table de nommage #1 ($2400)

Figure 0.9 : Les tables de nommage utilisées pour l'arrière-plan.

L'affichage d'une frame se fait pixel par pixel, de gauche à droite, de haut en bas. Ecrire dans le registre $2005 en plein milieu du traitement de la frame peut créer un effet d'écran partagé. Certains jeux, notamment les jeux de course, utilisent cela pour donner un effet 3D à la route, d'autres, comme *Super Mario Bros.*, l'utilisent pour la barre d'état.

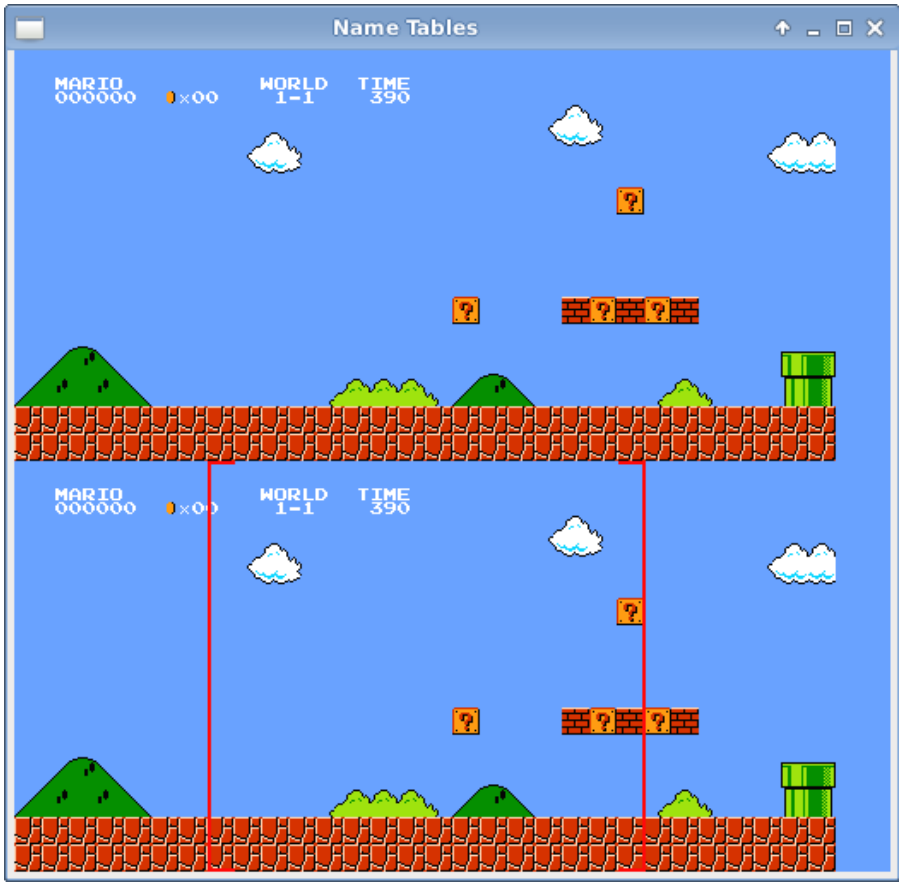

Figure 0.10 : Exemple de scrolling horizontal utilisé dans Super Mario Bros.

Figure 0.11 : Scrolling selon la scanline 32 dans Super Mario Bros.

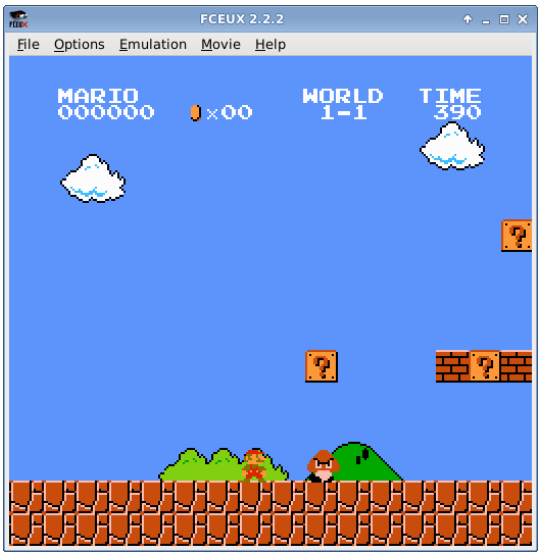

Figure 0.12 : Scrolling selon toutes les scanlines dans Super Mario Bros.

3.4 pAPU (Pseudo-Audio Processing Unit)

La NES n'a pas de puce dédiée pour générer les sons; c'est une partie du CPU qui prend cela en charge d'où le nom pAPU (Pseudo-Audio Processing Unit : pseudo unité de traitement audio). Les registres de la pAPU sont adressés dans la plage $4000~$4013, et les adresses $4015 et $4017 [14]. Elle dispose de 4 voix de synthèse FM et une voix pour le son numérique (PCM). En sortie, les sons seront convertis numériques en son analogique.

3.5 Mapper

Les communications entre les différents composants et mémoires se font à travers un circuit appelé le « mapper ». Le mapper a le même rôle que celui des chipsets des cartes mères actuelles; il se charge de fournir le bus à prendre pour véhiculer les données. Comme il est situé dans la cartouche, certains concepteurs modifient les mappers fournis par Nintendo pour avoir plusieurs pages de PRG-ROM et CHR-ROM, augmentant ainsi la capacité de stockage. Même Nintendo modifie ses propres mappers et les propose aux développeurs. En effet, un mapper peut faire des opérations de « swap », permettant ainsi de changer les pages mises à disposition pour le programme. Voici quelques exemples des mappers les plus utilisés:

- *UNROM*: connu comme le 'mapper #2' [13], c'est l'un des premiers mappers utilisés dans les cartouches de jeux de la NES. La ROM est divisée principalement en deux parties : CHR-ROM et PRG-ROM. UNROM ne se charge de faire le swap que pour la partie PRG, le nombre maximum de pages de 16 Ko qu'il peut supporter est de 8.
- *CNROM*: ou le 'mapper #3' [13]. contrairement au UNROM, CNROM n'est capable de faire le swap qu'avec la partie CHR-ROM. Les jeux équipés par ce mapper ont généralement des graphismes plus sophistiqués.

3.6 Contrôleurs

L'interaction de l'utilisateur avec la machine se fait à travers les contrôleurs. La NES a deux ports d'entrée pour les périphériques. Il existe plusieurs types d'appareils qui peuvent servir comme périphériques d'entrée. Généralement, les jeux utilisent les joypads. L'état des touches des joypads est disponible via les adresses $4016 et $4017

[14]. Le joypad fonctionne avec quatre boutons, *A*, *B Start*, et *Select*, et une croix directionnelle.

Figure 0.13 Manette de la NES.

Chapitre 4 : Implémentation de l'émulateur

4.1 Choix du langage de programmation

Sachant que n'importe quel langage évolué aurait fait l'affaire pour créer un émulateur. Le but de notre travail est de trouver le moyen le plus efficace de créer un émulateur rapide et fidèle au fonctionnement originel de la machine source, indépendant de l'environnement d'exécution.

Notre choix s'est porté sur le langage C++ parce qu'il offre la possibilité de programmer en orienté objet, ce qui permet une gestion agile du projet, aussi parce qu'il dispose d'une gestion de mémoire bas niveau, utile pour les optimisations, mais surtout, c'est un langage compilé et exécuté nativement, permettant de rester indépendants des machines virtuelles comme Java.

4.2 Fonctionnement du programme

Notre machine cible est un ordinateur équipé d'un processeur d'une architecture x86 différente de celle de la NES, donc l'émulation s'effectue en LLE (Low Level Emulation). La conception de notre émulateur suit la composition matérielle de la NES. Ainsi, les registres et les mémoires de tailles 8 bits et 16 bits sont remplacés par des variables de types `uint8` et `uint16` respectivement, définies comme suit :

```
typedef unsigned char uint8;
typedef unsigned short uint16;
```

Quant aux processeurs et chipsets, ils sont définis par des classes :

4.2.1 CPU

La classe CPU contient des attributs représentant les registres du 2A03, la file d'attente des interruptions, ainsi que des méthodes utiles pour la traduction du code 2A03 (6502).

```
...
private:
	/// Registres du 2A03 (6502)
	uint8 A, X, Y; // registres généraux
	uint8 SP; // registre pointeur de pile
	uint8 status; // registre d'état
```

```
        uint16 PC; // registre compteur ordinal (Program Counter)
        ...
};
```

Dans notre mémoire, nous n'avons traité qu'une seule technique : l'interprétation. Celle-ci est implémentée de deux façons :

- *Première méthode*: l'opcode est récupéré de la mémoire grâce à l'objet `mapper` passé en paramètre à la méthode `interpret()`. Ensuite, l'instruction lui correspondant est définie à travers le tableau `opdata`. Enfin, les instructions substituants l'instruction du CPU de la NES sont choisies grâce au bloc `switch` et sont exécutées.

```
/* Exécute un opcode (ou une interruption) et retourne le nombre de
cycles écoulés */
const uint8 CPU::interpret(Mapper* mapper)
{
        // Verifier l'existence d'une interruption
        if (!m_interruptions.vide())
        {
                switch (m_interruptions.teteDeListe())
                {
                        case NMI:
                                ...
                                break;

                        case IRQ:
                                ...
                                break;

                        case RESET:
                                ...
                                break;
                }
                m_interruptions.effacerTeteDeListe();
                return 0;
        }

        uint8 opcode = 0;
        // Récupérer et décoder l'opcode
        opcode = mapper->cpuLire(PC);

        // Le switch qui interprète le code
        switch (opdata[opcode].instruction)
        {
                // *******
                // * ADC *
                // *******
                case INS_ADC:
                        // - Add with carry -
                        // Instructions subtituant l'instruction ADC
```

```
                        ...
                        break;
                ...
                // *******
                // * INC *
                // *******
                case INS_INC:
                        // - INCrement memory -
                        {
                        /* récupérer l'adresse absolue à partir de
                           PC+1 et le mode d'adressage de
                           l'instruction */
                        uint16 addr = adresseAbsolue(PC + 1,
                                                     opdata[opcode].mode,
                                                     mapper);
                        uint8 temp = mapper->cpuLire(addr)+1;
                        // Mise à jour du bit de signe
                        if (temp & 0x80) // 0x80 <=> bit 7
                                status |= STATUS_SIGN;
                        else
                                status &= ~STATUS_SIGN;
                        // Mise à jour du bit d'état zéro
                        if (temp == 0)
                                status |= STATUS_ZERO;
                        else
                                status &= ~STATUS_ZERO;
                        mapper->cpuEcrire(addr, temp);
                        }
                        /* Mettre PC à l'adresse de la prochaine
                           instruction à éxécuter */
                        PC += opdata[opcode].taille;
                        break;
                ...
        }
        return opdata[opcode].cycles;
}
```

Le tableau `opdata` sert à fournir des informations relatives à l'opcode, comme l'instruction correspondante à l'opcode, étant donné qu'une instruction peut avoir plusieurs opcodes.

- *Deuxième méthode*: Cette méthode ressemble à la recompilation dynamique. Chaque instruction du 2A03 a une méthode dans la classe CPU la substituant prenant comme paramètres l'opcode et le mapper. L'interprétation se fait en vérifiant si l'opcode disponible à l'adresse `PC` a déjà été traité ou pas. Pour cela, le tableau `m_cache` est mis à disposition. Ce tableau stocke l'adresse de la fonction correspondant à l'instruction de l'opcode, et un booléen `finBloc` renvoyant *vrai* si la prochaine instruction est une instruction de branchement. Si l'opcode à l'adresse `PC` a déjà été traité, le programme exécute l'instruction

lui correspondant ainsi que toutes les instructions qui suivent jusqu'à ce que le booléen `finBloc` soit *vrai*. Sinon, le programme remplit le tableau avec les pointeurs des méthodes et les exécute.

```
const uint16 CPU::interpret2(Mapper* mapper)
{
	// Vérifier l'existence d'une interruption
	...

	// Vérifier si les opcodes ont déjà été interprétés
	if (m_cache[PC].instruction == 0)
	{
		uint16 tmpPC = PC;
		uint8 opcode;

		do
		{
			opcode = mapper->cpuLire(tmpPC);

			m_cache[tmpPC].instruction =
m_instructionMap[opdata[opcode].instruction];
			m_cache[tmpPC].finBloc = false;

			tmpPC += opdata[opcode].size;

		} while (opdata[opcode].instruction != INS_JMP
			&& opdata[opcode].instruction != INS_JSR
			&& opdata[opcode].instruction != INS_RTS

			&& opdata[opcode].instruction != INS_BCC
			&& opdata[opcode].instruction != INS_BCS
			&& opdata[opcode].instruction != INS_BEQ
			&& opdata[opcode].instruction != INS_BMI
			&& opdata[opcode].instruction != INS_BNE
			&& opdata[opcode].instruction != INS_BPL
			&& opdata[opcode].instruction != INS_BVC
			&& opdata[opcode].instruction != INS_BVS

			&& opdata[opcode].instruction != INS_BRK
			&& opdata[opcode].instruction != INS_RTI);

		m_cache[tmpPC - opdata[opcode].size].finBloc = true;
	}

	uint16 cycles = 0;
	do
	{
		uint8 opcode = mapper->cpuLire(PC);
		cycles += (this->*(m_cache[PC].instruction))(opcode,
                                                     mapper);
	} while ((m_cache[PC].finBlock == false) && (PC < 0x10000));

	return cycles;
}
```

Le tableau `m_instructionMap` renvoie l'adresse de la méthode correspondante à l'instruction passée comme index.

4.2.2 PPU

La classe `PPU` joue le rôle d'un compteur de cycles et ajoute une interruption au CPU lors de la période VBlank. Elle modifie aussi ses registres ($2000, $2001, $2002) au moment voulu.

```
        ...
private:
        uint8 m_control1, m_control2; // registres $2000 et $2001
        uint8 m_status; // registres $2002
        uint8 m_latch_low, m_latch_high; /* utile pour les registres
                                            $2003 et $2006 */
        uint8 m_scrollX, m_scrollY; // registre $2005

        bool m_bVblank; // vrai si en periode de VBlank

        uint16 m_scanline; /* la ligne courante que la PPU est
                              entrain de dessiner */
        uint16 m_cycles; /* Le nombre de cycles PPU passé durant la
                            frame courante*/
        ...

};
```

La classe a une méthode `emule()` qui doit être appelée après chaque exécution de code venant du CPU afin de synchroniser le CPU et la PPU. Le paramètre `cpuCycles` est le nombre de cycles des instructions exécutées par le CPU entre le dernier appel à la méthode `emule()` et le nouveau.

```
void PPU::emule(Mapper* mapper, CPU* cpu, uint16 cpuCycles)
{
        // update cycles
        m_cycles += cpuCycles * CYCLEPPU_PAR_CYCLECPU;


        // update scanline
        if (m_cycles >= CYCLES_PAR_HBLANK)
        {
                m_scanline += m_cycles / CYCLES_PAR_HBLANK;
                m_cycles = m_cycles % CYCLES_PAR_HBLANK;
                if (m_scanline > 261)
                        m_scanline = 0;
                ...
        }

        // Periode de Vblank
        if (m_scanline > 239)
        {
```

```
                // Début du Vblank
                if (!m_bVblank)
                {
                        ...
                        m_bVblank = true;
                        // Emission de l'interruption NMI
                        if (m_control1 & PPU_REG_NMI_ON_VBLANK)
                                cpu->ajoutInterrupt(Interrupt::NMI);
                }
        }
        else
        {
                // Fin du Vblank
                if (m_bVblank)
                {
                        m_bVblank = false;
                        ...
                }
                ...
        }
}
```

Un cycle du CPU équivaut à trois cycles de la PPU, exprimé par la constante `CYCLEPPU_PAR_CYCLECPU`.

La constante `CYCLES_PAR_SCANLINE` définit le nombre de cycles PPU pour dessiner une scanline (= 341).

4.2.3 Mapper

Il existe différents mappers pour la NES. Comme tous ces mappers jouent le même rôle, à savoir lire et écrire des données sur différentes mémoires, la classe `Mapper` est une classe abstraite, ainsi toutes les classes définissant un mapper héritent de cette classe.

4.2.4 Mapper0

La classe `Mapper0` hérite de la classe `Mapper`. Cette classe représente le Mapper #0 utilisé dans la plupart des jeux sortis avec la NES. Simple et basique, il n'utilise pas de pagination, il ne peut gérer que 32 Ko de PRG-ROM et 8 Ko de CHR-ROM.

4.2.5 Cartridge

Afin de charger le jeu (la ROM) dans la mémoire et créer le mapper utilisé dans le jeu, l'émulateur doit passer par des opérations, et ces opérations sont effectuées par la classe `Cartridge`, représentant la cartouche.

4.2.6 NES

Cette class contient tous les objets composant l'émulateur.

Chapitre 5 : Etude comparative des techniques utilisées

5.1 Mesures de performances

Afin de comparer les deux méthodes que nous avons implémentées, nous avons dû choisir entre deux techniques de mesure :

- La comparaison du nombre d'instructions exécutés par l'émulateur.
- La mesure du nombre de frame traitées.

Nous avons préféré utiliser la deuxième, car il est évident qu'elle est la plus significative. En effet, les optimisations effectuées fausseraient la mesure du nombre d'instructions en ne considérant pas celles qui ont été sautées.

Les performances ont été mesurées à partir d'un ordinateur personnel ayant les caractéristiques suivantes :

- Processeur : Intel® Pentium® E2140 @ 1.7 Ghz
- RAM : 2Go DDR2
- Système d'exploitation : Windows 7 Edition Integrale SP1

L'émulateur a été compilé avec Microsoft Visual Studio 2013 et est lancé 10 fois sous les mêmes conditions. Le résultat est la moyenne des frames obtenues après chaque exécution, sachant qu'une exécution dure 10 secondes.

Le jeu choisi pour notre benchmark est *Super Mario Bros*.

Le code utilisé pour le benchmark est comme suit :

```
int main(int argc, char* argv[])
{
	NES nes;
	nes.charger("/roms/Super Mario Bros. (Japan, USA).nes");

	CPU* cpu = nes.getCPU();
	PPU* ppu = nes.getPPU();
	Mapper* mapper = nes.getMapper();

	unsigned long int instructions = 0;
	unsigned long int frames = 0;

	/* obtenir le temps écoulé depuis le lancement du programme
	   en ms */
	Uint32 tempsDebut = SDL_GetTicks();
```

```
	// tant que 10 secondes ne sont pas écoulées
	while (SDL_GetTicks() - tempsDebut < 10000)
	{

		// Frame
		for (int i = 0; i < 263; i++)
		{
			uint16 cycles;
			for (cycles = 0;
		 cycles < CYCLES_PER_HBLANK / PPUCYCLE_PER_CPUCYCLE;)
			{
				// Pour la méthode 1
				cycles += cpu->interpret(mapper);
				// Pour la method 2
				// cycles += cpu->interpret2(mapper);
				instructions++;
				ppu->emuler(mapper, cpu, cycles);
			}
		}

		frames++;
	}

	std::cout << "instructions:\t " << instructions
		  << std::endl;
	std::cout << "frames:\t " << frames << std::endl;
	std::cout << std::endl;

	system("pause");

	return 0;
}
```

5.2 Résultats et discussion

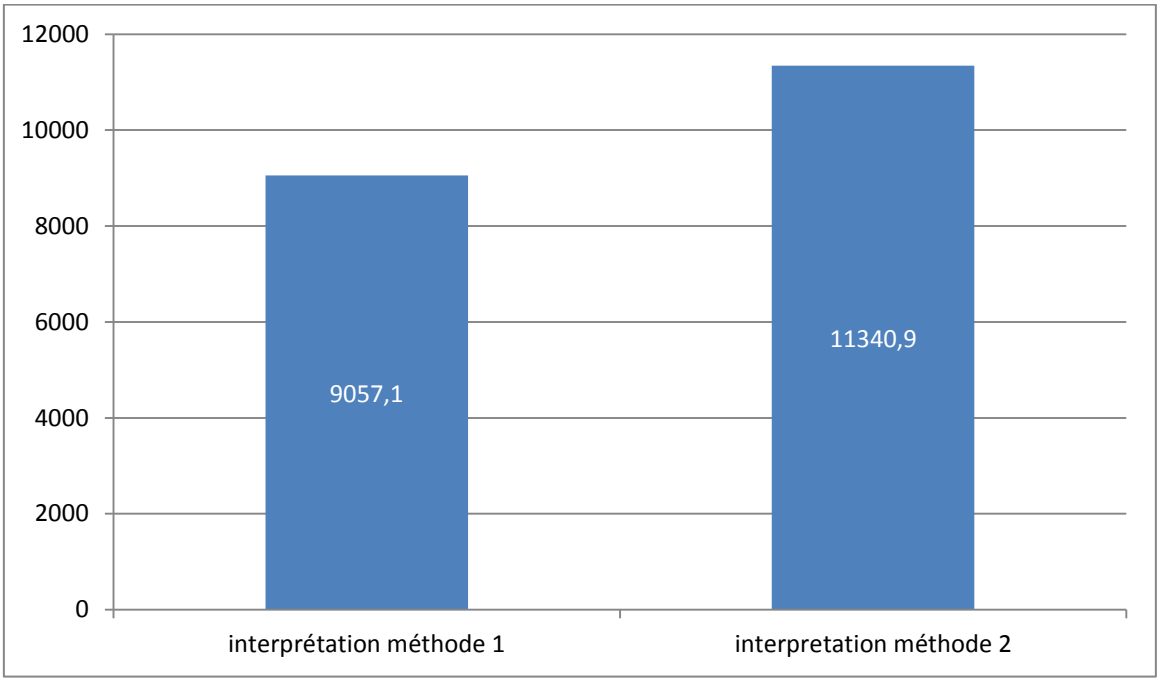

Figure 0.1 Graphe comparatif montrant le nombre de frames obtenu avec chaque méthode utilisée.

Le graphe montre qu'en utilisant la deuxième méthode de l'interprétation. Le nombre de frames obtenu est 25% plus élevé que celui obtenu avec la première méthode. Ceci est dû principalement au prefetcher exécuté dans le processeur de la machine cible.

En effet, dans la deuxième méthode le pointeur de la fonction à exécuter est directement retrouvé grâce au tableau `m_cache`. Cependant, lors de l'accès à une case dans un tableau, une copie est faite de la case en question mais aussi de toutes les cases voisines. Ainsi, puisque les pointeurs de fonction à exécuter sont contigus, l'exécution est plus rapide.

Conclusion et perspectives

Ce mémoire est à mi-chemin entre la théorie et la pratique. Il nous a permis de présenter les bases théoriques du fonctionnement d'un émulateur en en exposant les différents types. Toutefois, même s'il est possible de faire fonctionner quelques jeux avec l'application réalisée dans la partie pratique, elle n'est pas destinée à une utilisation en dehors du benchmark.

L'étude comparative réalisée montre bien qu'une technique peut avoir plusieurs méthodes d'implémentation qui diffèrent significativement dans les résultats obtenus.

Notre application n'a traité que la technique LLE avec la méthode d'interprétation. En perspectives, les autres techniques restent à étudier. C'est ce que nous avons entamé en effectuant des recherches sur la recompilation. Nous avons notamment été intéressés par l'API LLVM, qui permet de créer du code exécutable durant l'exécution de l'émulateur. Une étude complète sortirait du cadre de ce projet, auquel le temps imparti est limité, d'autant plus que l'utilisation de cette API est peu documentée, mais pourrait faire l'objet de recherche plus poussées.

Bibliographie

[1] «Synonyme émulation français,» [En ligne] : http://dictionnaire.reverso.net/francais-synonymes/%C3%A9mulation.

[2] V. M. d. Barrio, Study of the techniques for emulation programming, Computer Science Engineering – FIB UPC, 2001.

[3] W. contributors, «Church–Turing thesis,» Wikipedia, The Free Encyclopedia., 6 mai 2014. [En ligne] : http://en.wikipedia.org/w/index.php?title=Church%E2%80%93Turing_thesis&oldid=607372653.

[4] «The difference between virtualization and emulation - Oxford University,» [En ligne] : http://jpc.sourceforge.net/oldsite/Emulation.html.

[5] «EmuTimeLine Part 2 1962 - 1988,» [En ligne] : http://www.zophar.net/articles/art_14-2.html.

[6] «Histoire de l'émulation,» [En ligne] : http://www.tomshardware.fr/articles/Emulation-tuto,2-809-2.html.

[7] M.König, «Dynamic Recompilation - Freqently Asked Questions,» [En ligne] : http://web.archive.org/web/20030201234728/http://www.dynarec.com/~mike/drfaq.html.

[8] «Théorie de l'émulation la recompilation dynamique (dynarec),» [En ligne] : http://blogs.wefrag.com/mrhelmut/2012/05/09/theorie-de-lemulation-la-recompilation-dynamique-dynarec/.

[9] «Création d'un émulateur - Choix de la technique,» [En ligne] : https://github.com/franckverrot/EmulationResources/blob/master/vieux_tutoriaux/general/iii%20-%20Cre%CC%81ation%20d%27un%20e%CC%81mulateur%20-%20Choix%20de%20la%20technique%20d%27e%CC%81mulation.html.

[10] W. contributors, «Nintendo Entertainment System,» Wikipedia, The Free Encyclopedia, 23 mai 2014 . [En ligne] : http://en.wikipedia.org/w/index.php?title=Nintendo_Entertainment_System&oldid=609858187.

[11] CRISPYSIX, «nesdoc1.txt,» [En ligne] : http://nesdev.com/nesdoc1.txt.

[12] A. J. Jacobs, «6502 Architecture,» [En ligne] : http://www.obelisk.demon.co.uk/6502/architecture.html.

[13] P. Diskin, «NESDoc.pdf,» [En ligne] : http://nesdev.com/NESDoc.pdf.

[14] CRISPYSIX, «nesdoc2.txt,» [En ligne] : http://nesdev.com/nesdoc2.txt.

[15] «The NES Picture Processing Unit (PPU),» [En ligne] : http://badderhacksnet.ipage.com/badderhacks/index.php?option=com_content&view=article&id=270:the-nes-picture-processing-unit-ppu&catid=14:dr-floppy&Itemid=7.

[16] «FCEUX Help,» [En ligne] : http://www.fceux.com/web/help/fceux.html?PPU.html.

Glossaire

Terme	**Définition**
6502	Le 6502 est un microprocesseur 8-bit conçu par MOS Technology en 1975, utilisé notamment dans Apple II. Aussi il fut adapté dans la Nintendo Entertainment System par Ricoh pour donner le 2A03.
API	Application Programming Interface. C'est un ensemble de bibliothèques offrant aux développeurs la possibilité de réutiliser des classes, fonctions et méthodes déjà définies.
BCD	Binary Coded Decimal. C'est un système de numérotation. Chaque nombre est représenté en chiffres décimaux et chacun de ces chiffres est codé sur 4 bits.
Benchmark	Un benchmark est un banc d'essai permettant de mesurer les performances d'un programme pour le comparer à d'autres.
Binary Translation	Binary Translation ou traduction de code consiste à produire une séquence de code pour le processeur cible équivalente aux opérations effectuées par un bloc d'instructions sur la machine source.
Bus	Dans une machine, on appelle un Bus tous support de communication entre le matériel de cette machine. Généralement, il existe trois types de bus : bus de contrôle, bus d'adresses et bus de données.
Cartouche	Une cartouche est un étui contenant une mémoire morte (ROM), stockant le code du jeu vidéo. Différents matériels nécessaires pour l'exécution du programme peuvent parfois y être trouvés.
CHR-ROM	Character ROM. C'est une partie de la ROM contenant les graphismes du jeu.
Compilation	En informatique, la compilation est tout processus se chargeant de transformer un programme écrit en langage évolué en un autre langage, en passant par deux phases, la phase d'analyse et la phase de génération de code.
CPU	Central Processing Unit ou unité central de traitement. C'est un composant de l'ordinateur permettant d'exécuter les instructions du code machine, utile pour le fonctionnement du programme.

DMA	Direct Memory Access. C'est un circuit électronique capable de transférer les données de, et vers, la mémoire sans que le processeur intervienne.
DynaRec	Dynamic Recompilation ou recompilation dynamique. C'est le processus qui permet de générer un code natif à la machine cible à partir d'un code binaire d'un programme d'une autre machine durant l'exécution de ce dernier.
Frame	La frame est l'image générée par la puce graphique et affichée sur l'écran, c'est une composition de tous les éléments de l'affichage (arrière-plan, sprites,…).
HBlank	Horizontal Blank. C'est l'intervalle de temps qui sépare l'affichage de deux lignes qui se suivent.
HLE	High Level Emulation. C'est une technique d'émulation qui consiste à remplacer la couche entre le matériel et le programme à émuler.
JIT	Just In Time, ou recompilation à la volée. C'est une sorte de compilation où le code source est écrit en langage haut niveau par rapport au langage généré. La compilation se fait durant l'exécution du programme.
LLE	Low Level Emulation. C'est une approche dans l'émulation. Elle consiste à reconstruire la machine à émuler par logiciel ou par matériel.
Mapper	Le mapper est une puce disponible dans la cartouche, permettant de gérer le transport de données entre différents composants de la machine.
NES	Nintendo Entertainment System est une console de jeu de troisième génération fabriquée par Nintendo, sortie en 1983.
NTSC	National Television System Committee est un standard des télévisions en couleur, créé par les américains en 1953. Il est adapté en format vidéo 525 lignes à une fréquence de 30 images par seconde.
OPCODE	Un opcode (OPeration CODE), parfois référencé par Byte Code, est une portion d'une instruction du langage machine qui définit l'opération à effectuer.

Pagination	La pagination permet de diviser l'espace de la mémoire vive en un ensemble de pages ayant une taille identique et faire la correspondance entre les mémoires réelle et virtuelle. En d'autres termes, elle permet d'augmenter la mémoire vive.
PAL	Phase Alternating Line est un standard de télévision en couleur conçu par les allemands en 1960. Ce standard permet d'afficher des images de 652 lignes en 25 Hz.
Pile	La pile est un espace dans la mémoire RAM. Elle a une structure spéciale, LIFO (Last In First Out). Elle est utilisée généralement pour stocker des données temporaires.
PPU	Picture Processing Unit est la puce responsable de gérer le coté graphique dans la machine. Elle a le même rôle que les cartes graphiques des ordinateurs récents.
Prefetcher	Les prefetchers dans un processeur sont les optimisations faites lors de l'exécution de certaines instructions.
PRG-ROM	Program ROM est une partie de la ROM contenant la partie logique du programme.
RAM	Random Access Memory est une mémoire volatile, a un temps d'accès rapide, utilisée pour stocker les données nécessaire au fonctionnement des programmes
Registre	Le registre est la mémoire la plus rapide disponible dans la machine. Elle se situe généralement dans les processeurs. Elle sert à stocker des informations sur le résultat des opérations, ou des informations nécessaires dans le déroulement du programme.
Ricoh	Ricoh est une entreprise japonaise fondé en 1936. Elle a été le fournisseur principal du CPU (2A03) et PPU (2C02) de la Nintendo Entertainment System.
ROM	Read Only Memory comme son nom l'indique c'est une mémoire disponible en lecture seulement.
Sprite	Un sprite est un ensemble de tiles ayant des coordonnés indépendantes de l'arrière-plan.

SPR-RAM	SPRite Random Access Memory. C'est une petite mémoire de 256 bits utilisée pour stocker les informations nécessaires pour afficher les sprites tel que leur position sur l'écran, le tile à utiliser, etc.
SRAM	Save RAM est une mémoire vive situé dans la cartouche utilisée pour la sauvegarde.
Tile	Un tile est un élément de base dans l'affichage, c'est grâce à lui que les graphismes sont générés.
VBlank	Vertical Blank est le lapse de temps entre la fin d'une frame et le début d'une autre.
VRAM	Video Random Access Memory est une mémoire vive servant à stocker les éléments essentiels à l'affichage. Généralement la VRAM a un accès plus rapide que la RAM.
WINE	Même si WINE est l'acronyme de Wine Is Not an Emulator, Wine est un émulateur permettant d'exécuter des programmes créés pour Windows sous Unix, Linux, BSD, OS X.

Résumé

Pour exécuter certaines tâches, les ordinateurs lisent un fichier binaire (programme). Cependant, la lecture d'un programme peut différer d'un support à un autre, rendant le programme incompatible.

L'émulation consiste à traduire le programme (émulation haut niveau), ou à simuler la machine compatible avec le programme en question (émulation bas niveau), afin de le faire fonctionner sur d'autres types de supports.

Dans notre projet, nous décrivons et testons différentes approches pour émuler un programme. Nous avons choisi comme exemple la Nintendo Entertainment System comme machine à émuler, Super Mario Bros. comme programme et un PC comme machine cible.

Mots-clés: émulation, compilation, simulation, compatibilité.

الخلاصة

حتى يتمكن الحاسوب من تشغيل برنامج معين يتوجب عليه تنفيذ مجموعة من الأوامر المدرجة داخل ما يعرف بالملفات الثنائية، لكن وبطبيعة الحال الأجهزة الحاسبة تتنوع وتختلف مما بجعل للأمر الواحد عدت صيغ ثنائية. كل صيغة تقابلها مجموعة من الحاسبات التي تفهمها. هنا تكمن مشكلة عدم التوافق. تقنية المحاكات وجدت لحل هذه المشكلة عن طريق ترجمة البرنامج وإعادة صياغته حتى نتمكن من نقله من جهاز الى أخر وهذه أعلى مستويات المحاكات، هناك أيضا طريقة أخرى لتنفيذ المحاكات من خلال إعادة بناء الحاسب المتوافق مع البرنامج المراد تشغيله ليكون بمثابة منصة وهمية وهذا أدنى مستوى للمحاكات حيث أننا نتطرق الى تفصيل الجهاز المراد محاكاته.

مشروعنا تمثل في اختبار مجموعة من طرق المحاكات لجهاز الألعاب "Nintendo Entertainment System" على الحاسوب الشخصي واختير برنامج "Super Mario Bros" كموضوع للاختبار.

كلمات البحث :محاكاة، تحويل برمجي، توافق

Abstract

In order to execute some tasks, computers read a binary file (program). However, reading a program can differ from a platform to another, making the program incompatible.

The emulation is the translation of a program (high level emulation), or the simulation of the compatible machine with the program (low level emulation), to run the program on others types of platforms.

In our project, we describe and test differents approaches to emulate a program. We choosed as an example the Nintendo Entertainment System as a machine to emulate, Super Mario Bros. as the program to emulate and a PC as a target machine.

Keywords: emulation, compilation, simulation, compatibility.