Émulation de consoles de jeux

Sur l'exemple de la console Nintendo GameBoy®

Plan de la présentation

- Qu'est-ce qu'un émulateur?
- Quel est le besoin?
- La Nintendo GameBoy, rapide cours de (pré)histoire...
- Émuler, mais comment?
- Chargement d'une „ROM“
- Émulation de la mémoire vive
- Émulation du CPU
- Émulation du LCD

- **Qu'est-ce qu'un émulateur?**

L'émulation est la capacité d'un système informatique à imiter le comportement d'un autre système. L'exemple le plus flagrant est l'utilisation d'émulateurs de consoles de jeux afin de pouvoir exécuter sur un ordinateur PC des logiciels (jeux...) créés pour lesdites consoles.

- **Quel est le besoin?**

  1) Préservation d'anciens logiciels
  Depuis l'apparition des ordinateurs, de nombreux logiciels ont été développés. Au fil du temps, les plateformes matérielles deviennent obsolètes, les systèmes changent... et les logiciels deviennent inutilisables. Le besoin consiste donc à pouvoir lancer nos logiciels préférés, en leurs „faisant croire“ qu'ils se trouvent non pas sur votre PC dernier cri, mais sur leur système natif.

  2) Facilitation du développement sur certaines plateformes
  Un usage de niche de l'émulation consiste aussi à émuler des plateformes peu comodes à manier lors des processus de développement des applications (par ex. développement et test d'applications pour automates industriels sur émulateur, avant upload sur de vrais équipements).

- **La Nintendo GameBoy, rapide cours de (pré)histoire...**

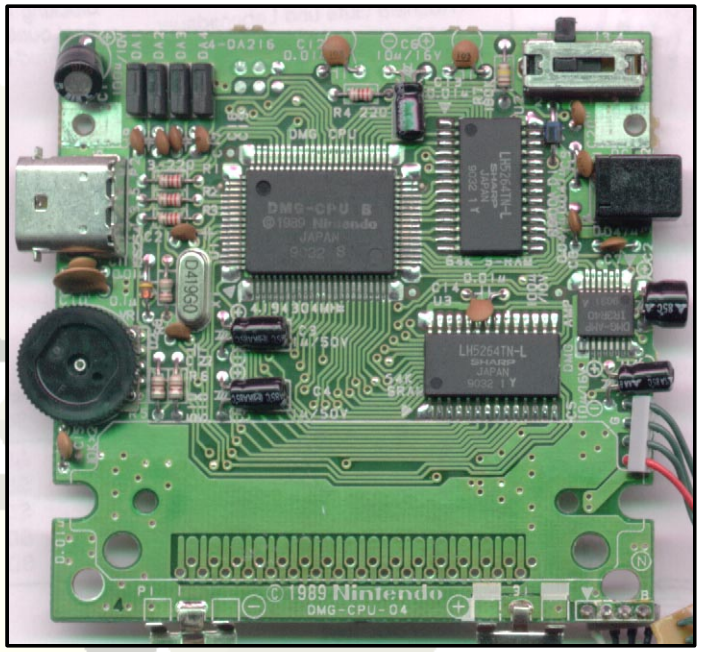
  Afin d'illustrer cette présentation, nous prendrons comme exemple l'émulation d'un système Nintendo GameBoy.

  Commercialisée en 1989, la console GameBoy a connue un franc succès auprès des jeunes adepts de jeux vidéo. En effet, de nombreux titres ont été disponibles pour ce système, qui a éduqué toute une génération de geeks. Aujourd'hui, à l'ère des gigahertz, il est devenu impossible de rejouer aux jeux de notre adolescence...

Une GameBoy du Paléolithique, retrouvée lors d'une fouille archéologique... ;-)

- **Émuler, mais comment?**

  Une console de jeux n'est ni plus ni moins qu'un ordinateur miniaturisé (système embarqué). La Nintendo GameBoy n'est pas une exception. Celle-ci est basée sur une version custom du processeur Zilog Z80, cadencée à 4 Mhz. Elle est dôtée d'une mémoire vive de 8KiB, et exploite un écran LCD 160x144 avec quatre nuances de gris.



L'émulation consistera donc simplement à reproduire le comportement de chacun des composants électroniques de la console, via un logiciel appellé „émulateur“. Nous verrons plus loin quelques extraits de code C, issus de l'émulateur zBoy.

En photo, la carte mère d'une GameBoy (1989).

- **Chargement d'une „ROM“**

  Les jeux de la console GameBoy sont stockés sur des supports électroniques à mémoire morte – des cartridges. Les cartridges les plus simples sont composés uniquement d'un module mémoire, sur lequel est gravé le code machine du jeu. Par abus de langage, on désigne par le terme „ROM“ les fichiers contenant une copie du code machine d'un jeu, extrait depuis un cartridge.

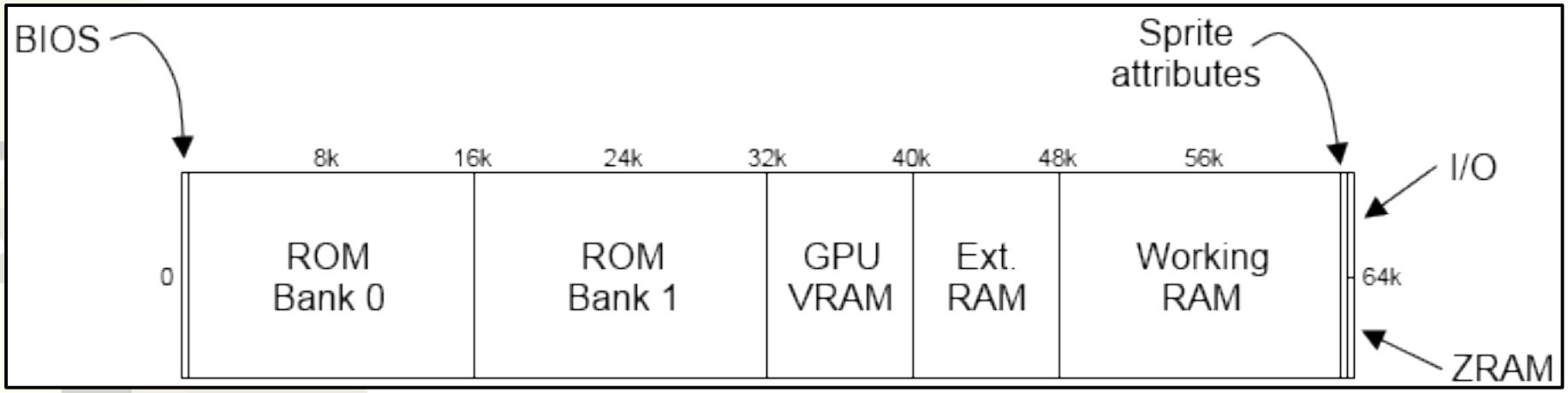
Afin de charger le contenu du cartridge „virtuel“, l'émulateur devra lire le fichier „ROM“ contenant le jeu, et le charger en mémoire, par exemple dans un simple tableau d'entier 8bits non signés:

```
FILE *RomHandler;
uint8 MemoryROM[4194304];
unsigned int RomSize;

RomHandler = fopen(Filename, "rb");
if (RomHandler != NULL) {
    fseek(RomHandler, 0, SEEK_END); /* seek to the end of the file */
    RomSize = ftell(RomHandler);    /* get current file pointer (ROM's size) */
    if (RomSize > 4194304) {
        fclose(RomHandler);
        printf("Erreur de chargement");
      } else {
        fseek(RomHandler, 0, SEEK_SET); /* seek back to beginning of file */
        fread(MemoryROM, RomSize, 1, RomHandler);
        fclose(RomHandler);
    }
  } else {
    printf("Erreur de chargement");
}
```

- **Émulation de la mémoire vive**

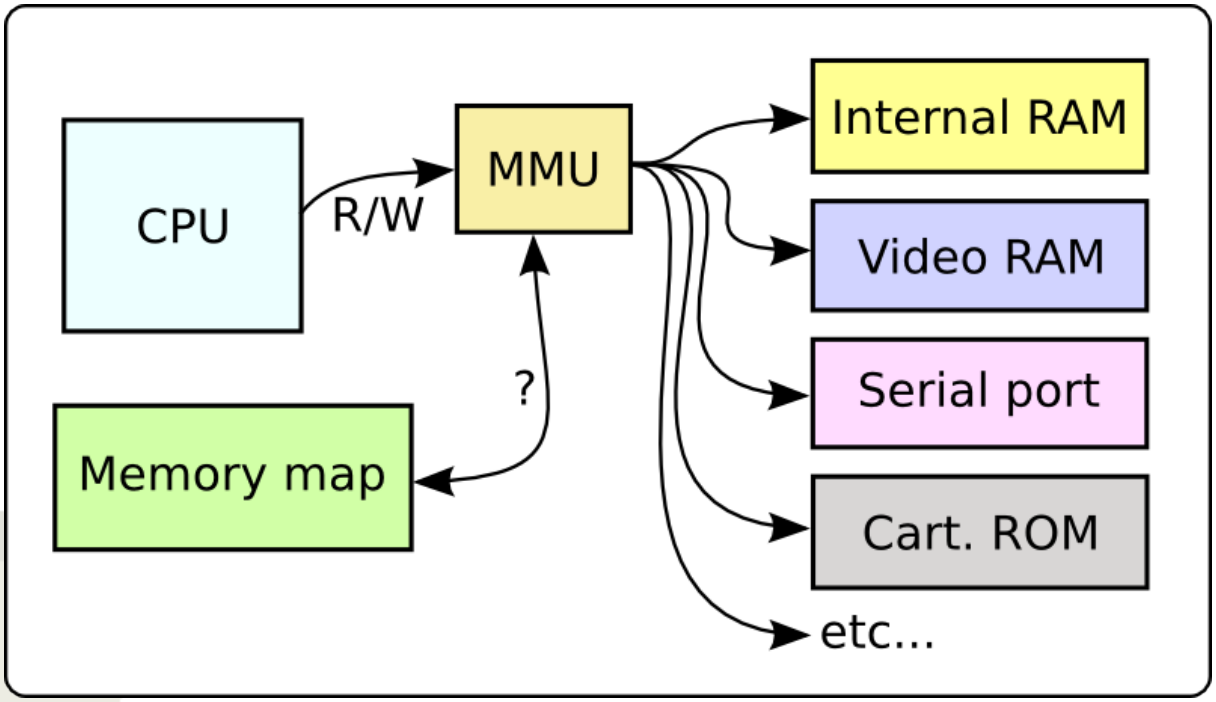
  La gestion de la mémoire vive d'une console GameBoy est assez particulière. La console utilise une notion de „memory map“. Lorsque le processeur fait appel à une adresse mémoire, cet appel est en réalité interprété par un controlleur mémoire (MMU), qui redirige la commande de lecture/écriture vers sa destination réelle.

  Le processeur Zilog 80 utilise des adresses 16bits, en conséquence la memory map est de 65536 octets.



L'émulation de la mémoire vive (ou plutôt, de la MMU) de la console GameBoy nécessite d'intercepter les appels CPU, et appliquer des actions différentes en fonction de l'adresse visée.

Un moyen simple consiste à émuler la mémoire vive réelle via un tableau d'entiers, et fournir au bloc d'émulation CPU des fonctions de lecture/écriture qui appliqueront la memory map.



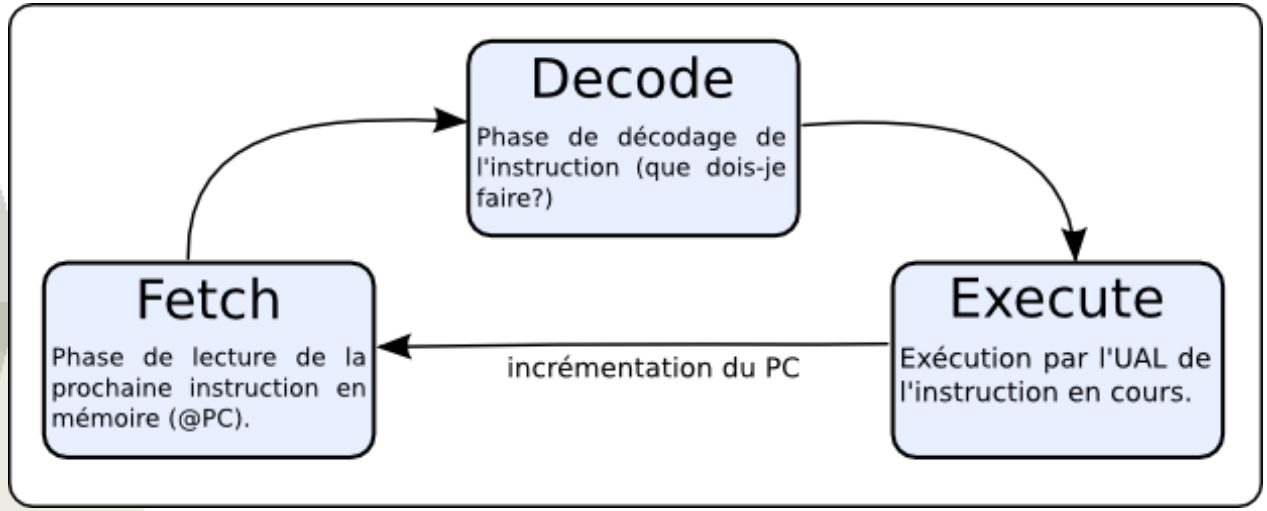
Voici un exemple de fonction émulant un ordre de lecture MMU. Attention: certains cartridges utilisent du bank switching, l'exemple n'est donc vrai que pour des ROMs <= 16K).

```
uint8_t MemoryInternalRAM[0xE000];    /* Internal RAM Area  [8KiB] */
uint8_t SpriteOAM[0xFEA0];            /* Sprite OAM memory */
uint8_t VideoRAM[0xA000];             /* Video RAM [8KiB] */
uint8_t MemoryInternalHiRAM[0x10000]; /* Internal RAM (high area + IE register) */
uint8_t IoRegisters[0xFF4C];

uint8_t MemoryRead(unsigned int ReadAddr) {
  if (ReadAddr < 0x4000) {                                  /* ROM bank #0 /*
    return(MemoryROM[ReadAddr]);
  } else if ((ReadAddr >= 0xC000) && (ReadAddr < 0xE000)) {   /* Internal 8KiB RAM /*
    return(MemoryInternalRAM[ReadAddr]);
  } else if ((ReadAddr >= 0xE000) && (ReadAddr < 0xFE00)) {   /* RAM mirror /*
    return(MemoryInternalRAM[ReadAddr - 8192]);
  } else if ((ReadAddr >= 0xFE00) && (ReadAddr < 0xFEA0)) {   /* Sprite OAM memory /*
    return(SpriteOAM[ReadAddr]);
  } else if ((ReadAddr >= 0xFF80) && (ReadAddr <= 0xFFFF)) {  /* Hi RAM area /*
    return(MemoryInternalHiRAM[ReadAddr]);
  } else if ((ReadAddr >= 0xFF00) && (ReadAddr <= 0xFF4B)) {  /* I/O registers /*
    return(IoRegisters[ReadAddr]);
  } else if ((ReadAddr >= 0x8000) && (ReadAddr < 0xA000)) {   /* Video RAM (8KiB) /*
    return(VideoRAM[ReadAddr]);
  } else {
    printf("Erreur MMU: Adresse inconnue");
  }
}
```

- **Émulation du CPU**

  Le processeur d'une console GameBoy est une variante du Zilog 80. Ce processeur dispose de 8 registres 8 bits (A, B, C, D, E, F, H, L), et deux registres 16 bits: le compteur ordinal (PC) et le pointeur de pile (SP), ainsi que d'un jeu de 500 instructions.

  Le processeur lit séquenciellement les instructions chargées en mémoire, les reconnait et exécute, de manière tout à fait classique (cycle d'instruction fetch / decode / execute).



L'émulation des registres du processeur peut être facilement réalisée en utilisant une structure mémoire:

```
struct CpuRegisters {
  uint8_t A;  /* Accumulateur */
  uint8_t B;
  uint8_t C;
  uint8_t D;
  uint8_t E;
  uint8_t F;
  uint8_t H;
  uint8_t L;
  int PC;     /* Program Counter */
  int SP;     /* Stack Pointer */
};
struct CpuRegisters Register;
```

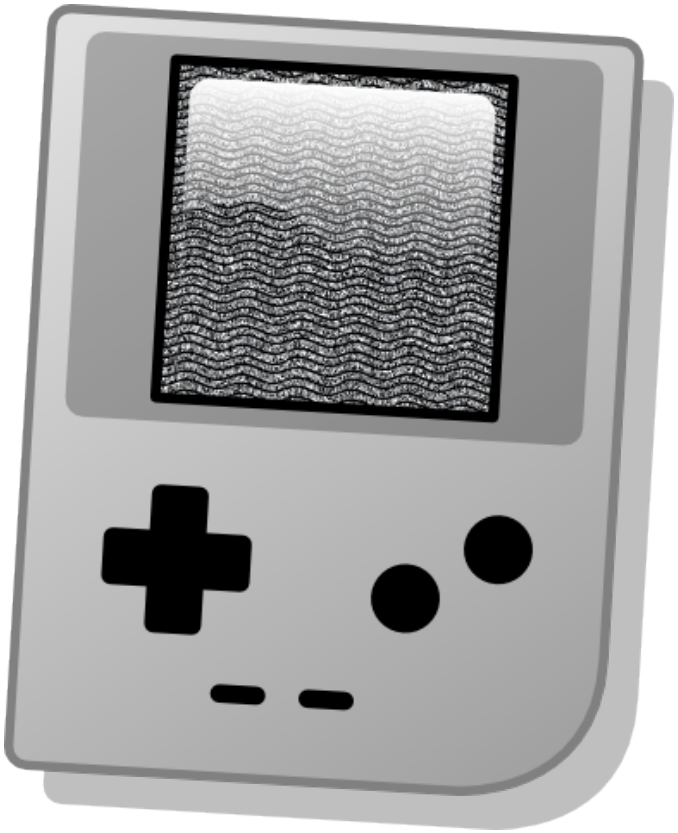
Les différentes instructions CPU émulées utiliseront cette structure, lorsque le logiciel (jeu) en aura besoin.

La phase de décodage consiste à reconnaitre l'instruction, pour l'exécuter ensuite. J'ai choisi d'implémenter cela via un switch() disposant de autant de cas, qu'il y a d'instructions possibles. Voici un extrait de code assurant l'émulation de la phase de décodage et d'exécution (pour l'instruction LD C,A):

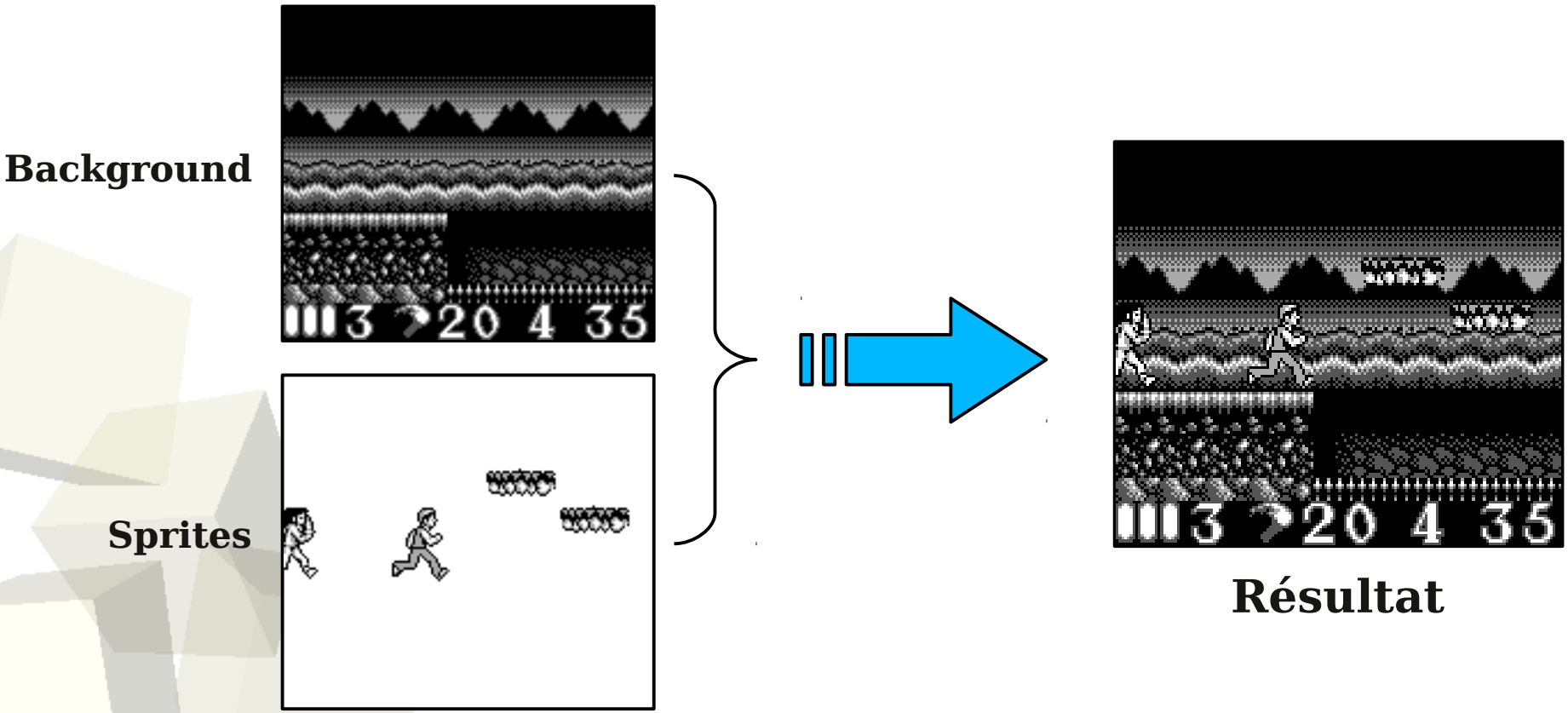
```
uint8_t CpuInstruction;          /* holds current CPU instruction (bytecode) */
CpuInstruction = MemoryRead(Register.PC);   /* Fetch the next opcode */

switch (CpuInstruction) {
  case 0x00:
    /* emulation de l'instruction 0x00 */
    break;
  case 0x01:
    /* emulation de l'instruction 0x01 */
    break;
  case 0x02:
    /* emulation de l'instruction 0x02 */
    break;
  /*  etc...  */
  case 0x4F:   /*  LD C,A */
      Register.C = Register.A;
      Register.PC += 1;
      break;
  /*  etc...  */
}
```

- **Émulation du LCD**

  L'émulation de la mémoire vive et du processeur ne suffit pas à faire un émulateur fonctionnel. Sans émulation du controlleur graphique, rien ne s'affichera à l'écran!

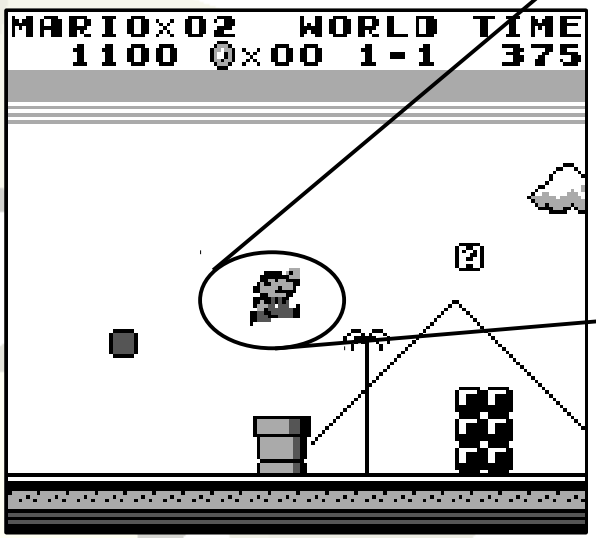


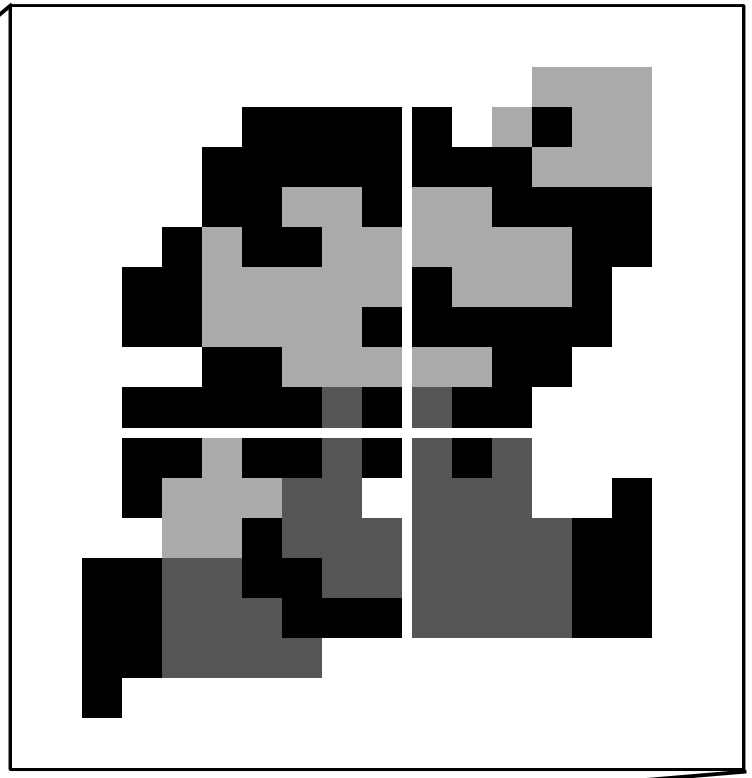
L'affichage d'un écran de GameBoy est composé de deux éléments: un fond d'écran (background), et des „sprites“ amovibles, placés à différents endroits de l'écran (leurs positions sont déclarées en temps réel dans la zone de mémoire OAM).



Chaque élément (sprites ou background) est lui-même composé de plusieures briques (tiles) de 8x8 pixels ou 8x16 pixels, stockées en mémoire vidéo.

L'émulateur devra donc charger chaque tile en mémoire, et l'appliquer à un écran virtuel (représenté par exemple sous forme d'un tableau 2d).





Notre protagoniste virtuel est ici construit à partir de 4 tiles de 8x8 pixels.

Une fois que notre „écran mémoire“ (tableau à deux dimensions) est prêt, il suffit de l'afficher à l'écran de notre machine physique. De nombreuses bibliothèques existent pour cela, ci-dessous se trouve un exemple de fonction d'affichage exploitant la bibliothèque SDL.

```
void RefreshScreen_NoScale(SDL_Surface *screen) {
  static unsigned int x, y;
  uint8_t *screenmem = (uint8_t *) screen->pixels;
  SDL_LockSurface(screen);
  for (x = 0; x <= 159; x++) {
    for (y = 0; y <= 143; y++) {
      screenmem[y * screen->pitch + x] = ScreenBuffer[x][y];
    }
  }
  SDL_UnlockSurface(screen);
  SDL_UpdateRect(screen, 0, 0, 160, 144);  /* Draw the buffer onscreen */
}
```

GAME OVER


TETRIS
1PLAYER 2PLAYER
©1989 Nintendo
Ferrari
GRAND PRIX
CHALLENGE
READY!!
GAME BOY
DONKEY KONG
PUSH START
©1994 Nintendo
LEVEL 05
VIRUS 12
:59
DARKWING DUCK
PRESS START
12:53
MARIO×01 WORLD TIME
109250 ×65 3-1 345