VOYAGE AU COEUR DE L'EMULATION : DESTINATION GAME BOY

SNOWCAMP

MARC-ALEXANDRE BLANCHARD - @MARC_ALX

US005184830A

United States Patent [19]
Okada et al.

[11] **Patent Number: 5,184,830**
[45] **Date of Patent: Feb. 9, 1993**

[54] **COMPACT HAND-HELD VIDEO GAME SYSTEM**

[75] Inventors: **Satoru Okada; Shin Kojo**, both of Kyoto, Japan

[73] Assignee: **Nintendo Company Limited**, Kyoto, Japan

[21] Appl. No.: **899,179**

[22] Filed: **Jun. 15, 1992**

Related U.S. Application Data

[63] Continuation of Ser. No. 462,400, Jan. 8, 1990, abandoned.

[30] **Foreign Application Priority Data**

Apr. 20, 1989 [JP] Japan 1-101028
Oct. 1, 1989 [JP] Japan 1-4452

[51] **Int. Cl.**[5] **A63F 9/22**
[52] **U.S. Cl.** ..... **273/433**; 273/434; 273/435; 273/85 G
[58] **Field of Search** ..... 273/433, 434, 435, 437, 273/85 R, 85 G, DIG. 28; 364/410

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,359,222 11/1982 Smith, III et al. ..... 273/85 G
4,395,760 7/1983 Soski et al. 364/410
4,438,926 3/1984 Yokoi et al. ..... 273/85 G
4,572,509 2/1986 Sitrick ..... 273/85 G
4,589,659 5/1986 Yokoi et al. ..... 273/1 GC
4,729,563 3/1988 Yokoi ..... 273/1 E
4,745,478 5/1988 Nakagawa 356/181
4,783,812 11/1988 Kaneoka 381/61
4,815,733 3/1989 Yokoi ..... 273/1 E
4,865,321 9/1989 Nakagawa et al. ..... 273/85 G
4,890,832 1/1990 Komaki 273/435

FOREIGN PATENT DOCUMENTS

58-136192 9/1983 Japan .
57989 9/1984 Japan .
60-21784 2/1985 Japan .
2033763 5/1980 United Kingdom .
8302566 8/1983 World Int. Prop. O. ..... 273/85 G

OTHER PUBLICATIONS

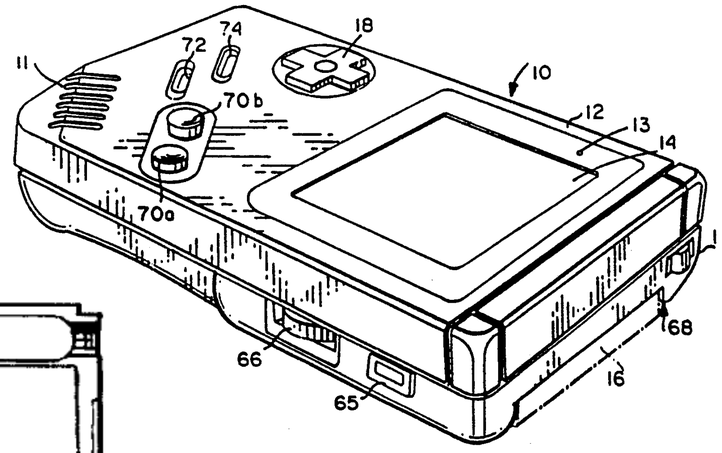
Worley, Joyce "Spitball Sparky", Electronic Games, Nov. 1984, p. 86.

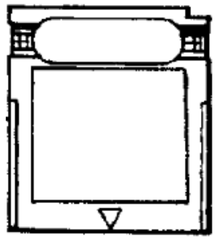
Primary Examiner—Jessica J. Harrison
Attorney, Agent, or Firm—Nixon & Vanderhye

[57] **ABSTRACT**

A hand-held electronic game machine for use with attachable/detachable memory game packs wherein the game machine includes a case of a size which may be held by a hand and capable of being sandwiched by both hands with a first switch disposed at a position such that during a game it can be operated by one thumb on a front surface of the case, a second switch disposed at a position such that during a game it can be operated by the other thumb on the first surface of the case and a third operation switch means provided in a region of said front surface where imaginary loci of both thumbs intersect with each other on the front surface, and wherein the game machine can be connected with others for simultaneous multiple player competition.

21 Claims, 12 Drawing Sheets





GAME BOY *le / la*

Console de jeu portable de Nintendo

Date de sortie : **1989**

Dernier jeu sorti : **2001**

Fin d'exploitation : 2003

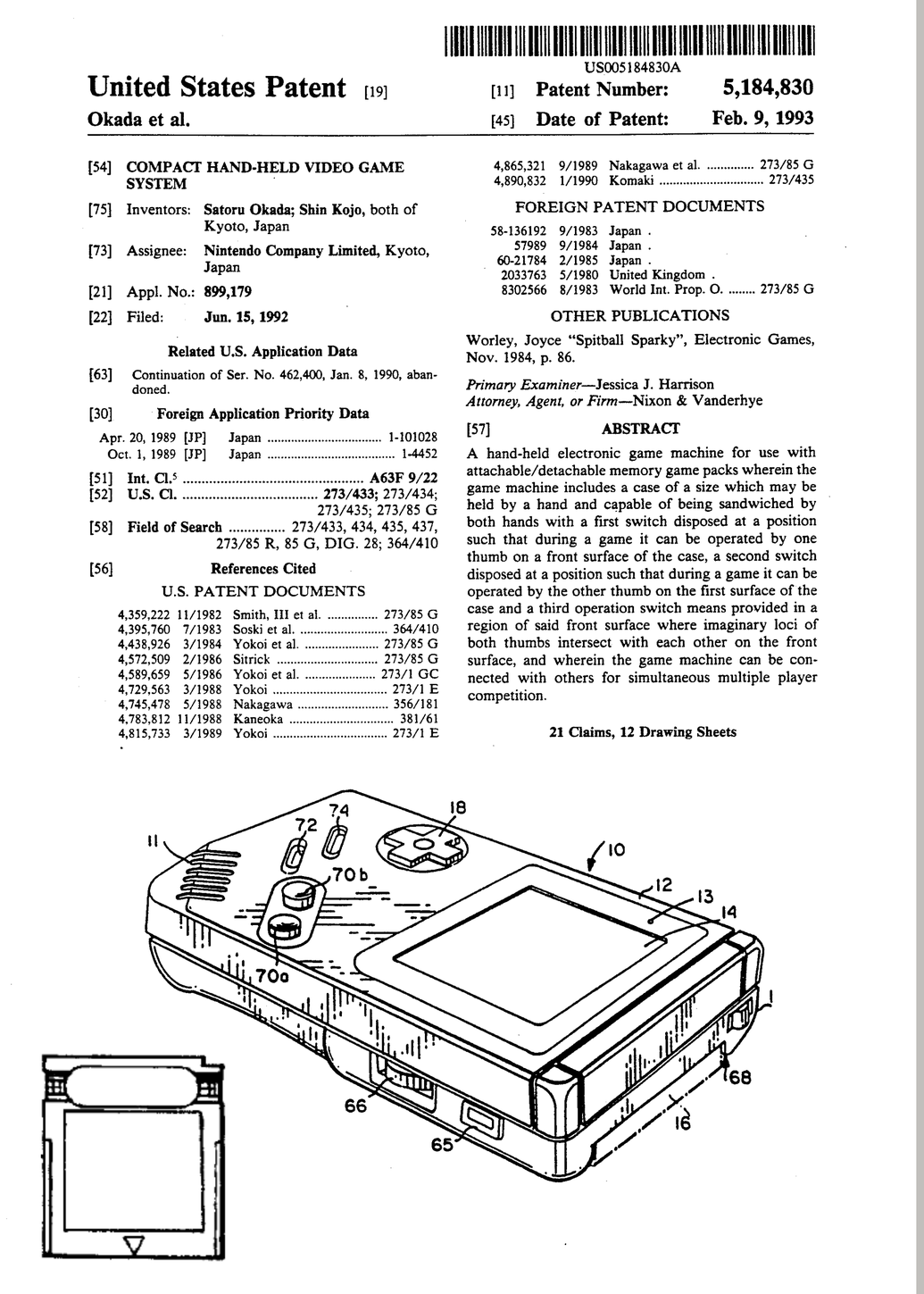
横井軍平 (Yokoi Gunpei)
Père de la Game Boy

GAME BOY

GAME BOY pocket

GAME BOY COLOR

GAME BOY ADVANCE



US005184830A

United States Patent [19]
Okada et al.

[11] **Patent Number: 5,184,830**
[45] **Date of Patent: Feb. 9, 1993**

[54] **COMPACT HAND-HELD VIDEO GAME SYSTEM**

[75] Inventors: **Satoru Okada; Shin Kojo**, both of Kyoto, Japan

[73] Assignee: **Nintendo Company Limited**, Kyoto, Japan

[21] Appl. No.: **899,179**

[22] Filed: **Jun. 15, 1992**

Related U.S. Application Data

[63] Continuation of Ser. No. 462,400, Jan. 8, 1990, abandoned.

[30] **Foreign Application Priority Data**

Apr. 20, 1989 [JP] Japan 1-101028
Oct. 1, 1989 [JP] Japan 1-4452

[51] **Int. Cl.**[5] **A63F 9/22**
[52] **U.S. Cl.** ..... **273/433**; 273/434; 273/435; 273/85 G
[58] **Field of Search** ..... 273/433, 434, 435, 437, 273/85 R, 85 G, DIG. 28; 364/410

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,359,222 11/1982 Smith, III et al. ..... 273/85 G
4,395,760 7/1983 Soski et al. 364/410
4,438,926 3/1984 Yokoi et al. ..... 273/85 G
4,572,509 2/1986 Sitrick ..... 273/85 G
4,589,659 5/1986 Yokoi et al. ..... 273/1 GC
4,729,563 3/1988 Yokoi ..... 273/1 E
4,745,478 5/1988 Nakagawa 356/181
4,783,812 11/1988 Kaneoka 381/61
4,815,733 3/1989 Yokoi ..... 273/1 E
4,865,321 9/1989 Nakagawa et al. ..... 273/85 G
4,890,832 1/1990 Komaki 273/435

FOREIGN PATENT DOCUMENTS

58-136192 9/1983 Japan .
57989 9/1984 Japan .
60-21784 2/1985 Japan .
2033763 5/1980 United Kingdom .
8302566 8/1983 World Int. Prop. O. ..... 273/85 G

OTHER PUBLICATIONS

Worley, Joyce "Spitball Sparky", Electronic Games, Nov. 1984, p. 86.

Primary Examiner—Jessica J. Harrison
Attorney, Agent, or Firm—Nixon & Vanderhye

[57] **ABSTRACT**

A hand-held electronic game machine for use with attachable/detachable memory game packs wherein the game machine includes a case of a size which may be held by a hand and capable of being sandwiched by both hands with a first switch disposed at a position such that during a game it can be operated by one thumb on a front surface of the case, a second switch disposed at a position such that during a game it can be operated by the other thumb on the first surface of the case and a third operation switch means provided in a region of said front surface where imaginary loci of both thumbs intersect with each other on the front surface, and wherein the game machine can be connected with others for simultaneous multiple player competition.

21 Claims, 12 Drawing Sheets

GAME BOY le / la

Console de jeu portable de Nintendo

Date de sortie : **1989**

Dernier jeu sorti : **2001**

Fin d'exploitation : 2003

Fonctionne avec des cartouches de jeu interchangeables :

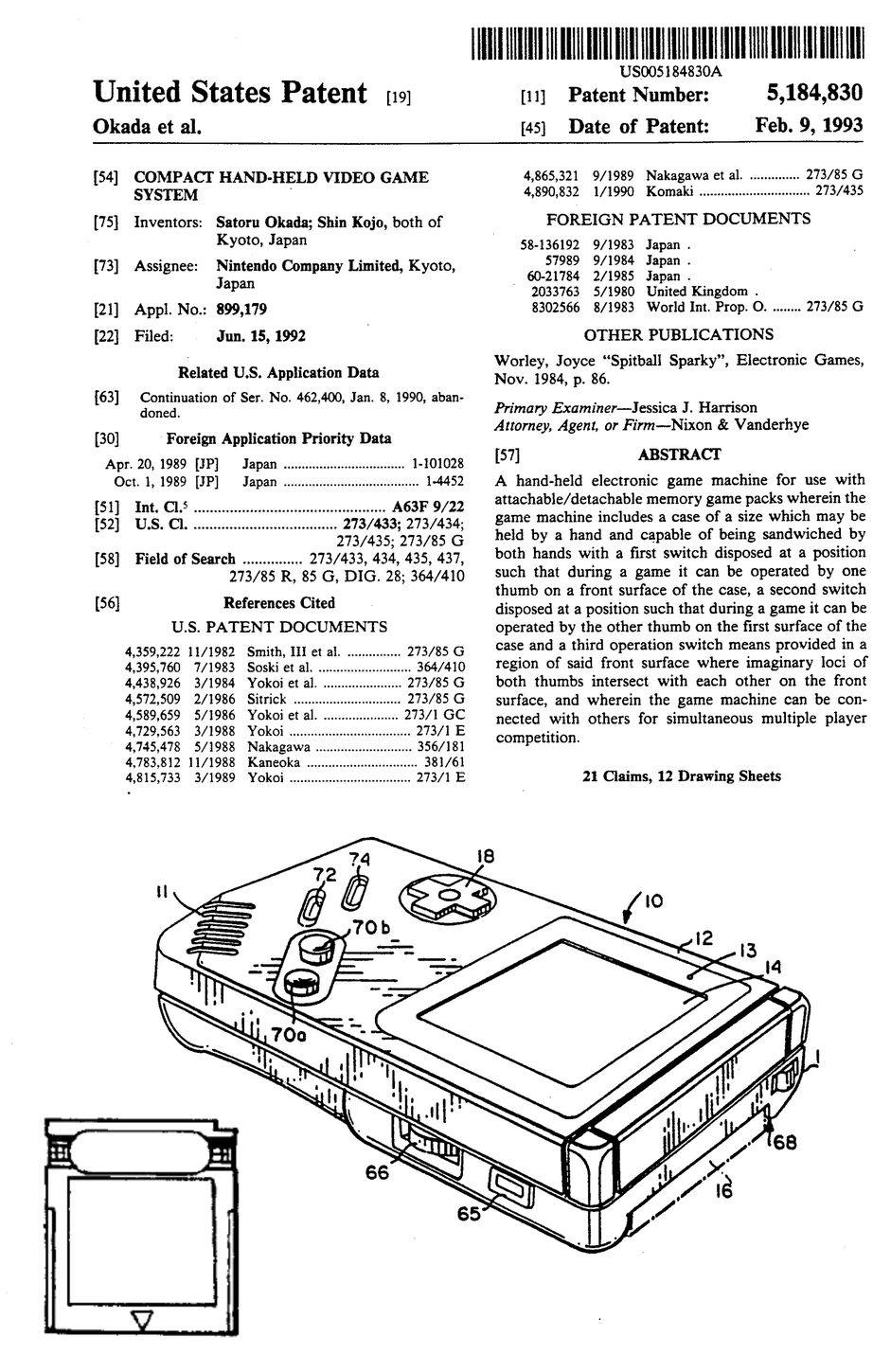
450 jeux uniques (800+ avec la série Color)

+ de 200 millions de jeux vendus

Nintendo GAME BOY
DMG-KYJ
MADE IN JAPAN

DMG-DECN-02
80-26

PINBALL
GAME BOY



US005184830A

United States Patent [19]
Okada et al.

[11] **Patent Number: 5,184,830**
[45] **Date of Patent: Feb. 9, 1993**

[54] **COMPACT HAND-HELD VIDEO GAME SYSTEM**

[75] Inventors: **Satoru Okada; Shin Kojo**, both of Kyoto, Japan

[73] Assignee: **Nintendo Company Limited**, Kyoto, Japan

[21] Appl. No.: **899,179**

[22] Filed: **Jun. 15, 1992**

Related U.S. Application Data

[63] Continuation of Ser. No. 462,400, Jan. 8, 1990, abandoned.

[30] **Foreign Application Priority Data**

Apr. 20, 1989 [JP] Japan 1-101028
Oct. 1, 1989 [JP] Japan 1-4452

[51] **Int. Cl.**[5] **A63F 9/22**
[52] **U.S. Cl.** ..... **273/433**; 273/434; 273/435; 273/85 G
[58] **Field of Search** ..... 273/433, 434, 435, 437, 273/85 R, 85 G, DIG. 28; 364/410

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,359,222 11/1982 Smith, III et al. ..... 273/85 G
4,395,760 7/1983 Soski et al. 364/410
4,438,926 3/1984 Yokoi et al. ..... 273/85 G
4,572,509 2/1986 Sitrick ..... 273/85 G
4,589,659 5/1986 Yokoi et al. ..... 273/1 GC
4,729,563 3/1988 Yokoi ..... 273/1 E
4,745,478 5/1988 Nakagawa 356/181
4,783,812 11/1988 Kaneoka 381/61
4,815,733 3/1989 Yokoi ..... 273/1 E
4,865,321 9/1989 Nakagawa et al. ..... 273/85 G
4,890,832 1/1990 Komaki 273/435

FOREIGN PATENT DOCUMENTS

58-136192 9/1983 Japan .
57989 9/1984 Japan .
60-21784 2/1985 Japan .
2033763 5/1980 United Kingdom .
8302566 8/1983 World Int. Prop. O. ..... 273/85 G

OTHER PUBLICATIONS

Worley, Joyce "Spitball Sparky", Electronic Games, Nov. 1984, p. 86.

Primary Examiner—Jessica J. Harrison
Attorney, Agent, or Firm—Nixon & Vanderhye

[57] **ABSTRACT**

A hand-held electronic game machine for use with attachable/detachable memory game packs wherein the game machine includes a case of a size which may be held by a hand and capable of being sandwiched by both hands with a first switch disposed at a position such that during a game it can be operated by one thumb on a front surface of the case, a second switch disposed at a position such that during a game it can be operated by the other thumb on the first surface of the case and a third operation switch means provided in a region of said front surface where imaginary loci of both thumbs intersect with each other on the front surface, and wherein the game machine can be connected with others for simultaneous multiple player competition.

21 Claims, 12 Drawing Sheets

GAME BOY le / la

Console de jeu portable de Nintendo

Date de sortie : **1989**

Dernier jeu sorti : **2001**

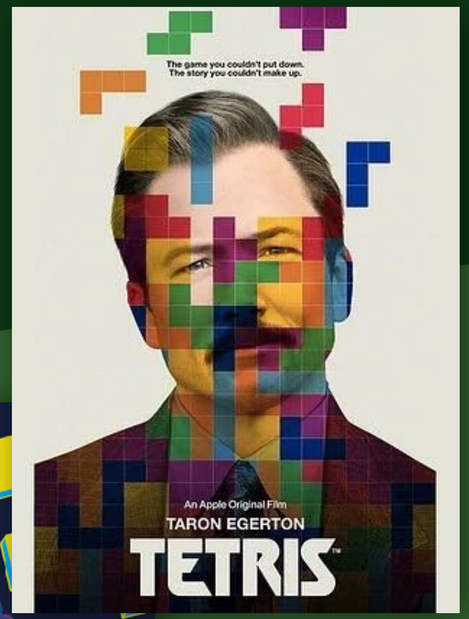
Fin d'exploitation : 2003

Fonctionne avec des cartouches de jeu interchangeables :

450 jeux uniques (800+ avec la série Color)

+ de 200 millions de jeux vendus

Vendue à plus de **100 millions** d'exemplaires

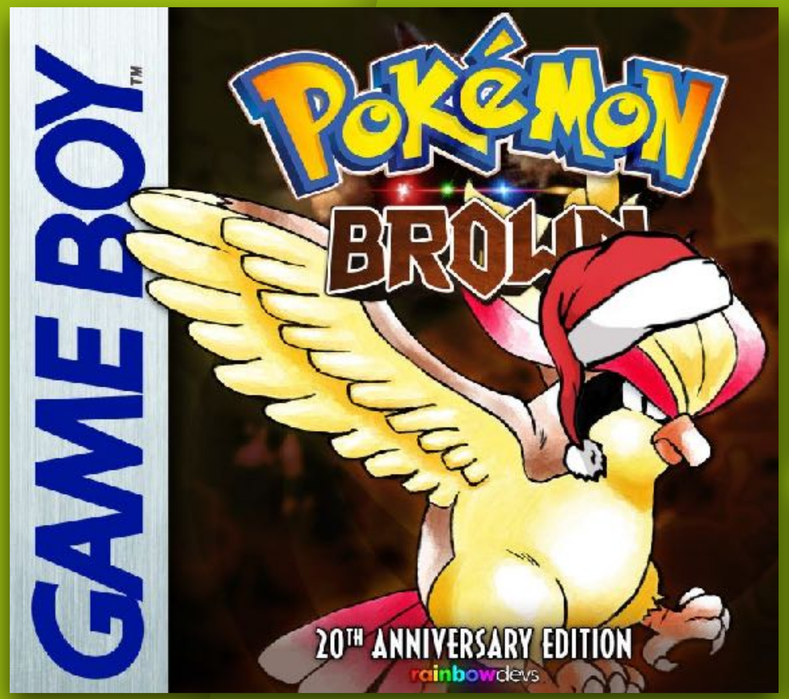


MALCOLM IN THE MIDDLE

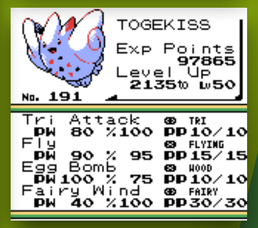
Russian cosmonaut **Aleksandr A. Serebrov** *(b. 15 February 1944) became* **the first person to play a videogame in space** when he packed a **Game Boy and** his personal copy of **Tetris** *(Nintendo, 1989) for his trip to the MIR Space Station* **in 1993**

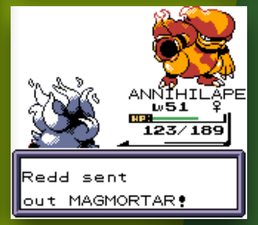
ANGÈLE - Nonante-Cinq

FRIENDS

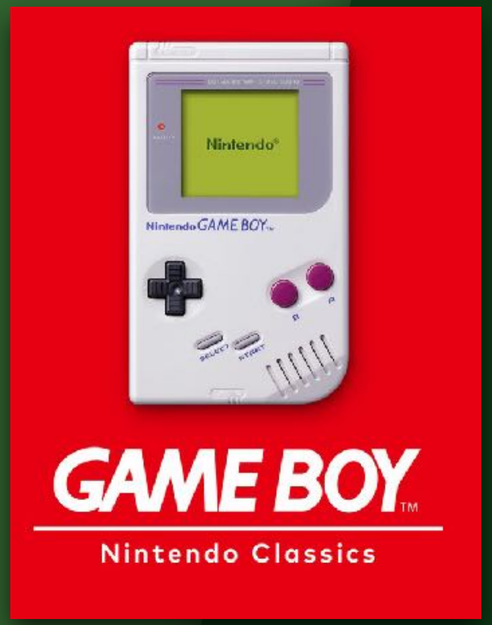

GAME BOY
Pokémon
BROWN
20TH ANNIVERSARY EDITION
rainbowdevs

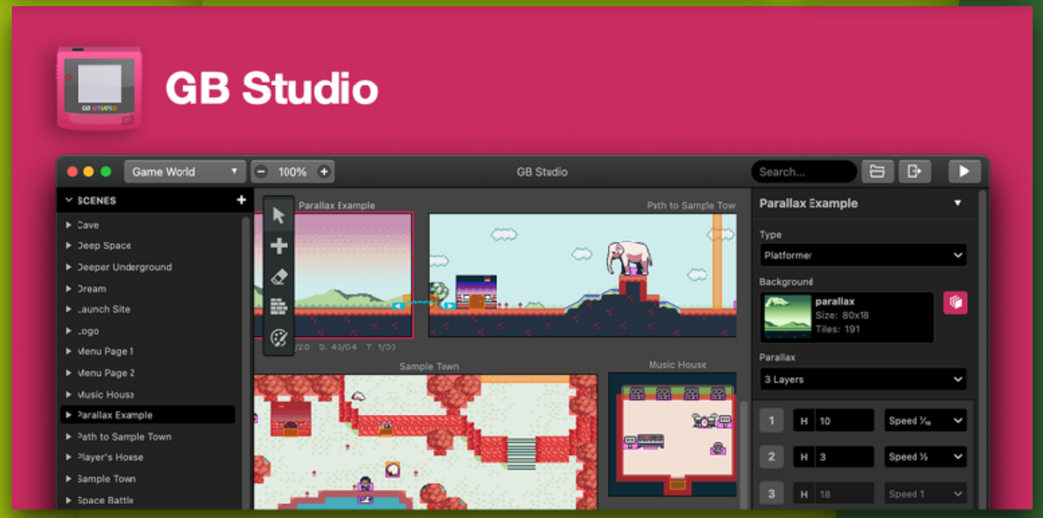
Pokémon
BROWN
20th ANNIVERSARY EDITION
2023-2024 RAINBOWDEVS


TOGEKISS
Exp Points
97865
Level Up
2135 to Lv50
No. 191
Tri Attack
PW 80 %100
PP 10/10
Fly
PW 90 % 95
PP 15/15
Egg Bomb
PW 100 % 75
PP 10/10
Fairy Wind
PW 40 %100
PP 30/30


ANNIHILAPE
Lv51
HP
123/189
Redd sent
out MAGMORTAR!

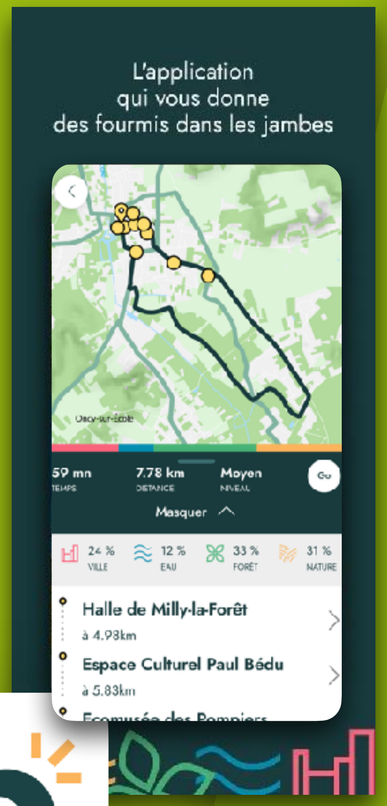
NINTENDO SWITCH
THE LEGEND OF
ZELDA
Link's Awakening
7


Nintendo
Nintendo GAME BOY
GAME BOY
Nintendo Classics


GB Studio
Game World
100%
GB Studio
SCENES
Cave
Deep Space
Deeper Underground
Dream
Launch Site
Logo
Menu Page 1
Menu Page 2
Music House
Parallax Example
Path to Sample Town
Player's House
Sample Town
Space Battle
Parallax Example
Sample Town
Music House
Parallax Example
Type
Platformer
Background
parallax
Size: 80x18
Tiles: 191
Parallax
3 Layers

TEDx Buffalo
16:30
Chiptune: Pushing the Limits Using Constraints: Dan Behrens(Danimal Cannon) at TEDxBuffalo
TEDx Talks
103.2 k vues • il y a 12 ans

Smart/Origin
GÉO-VALORISATION DE VOS DONNÉES MÉTIER



bougeott
(iOS/android)

YouTube FR

Rechercher

I was a developer of Shantae for the Nintendo Switch !

Modern Vintage Gamer

908 k abonnés

Rejoindre

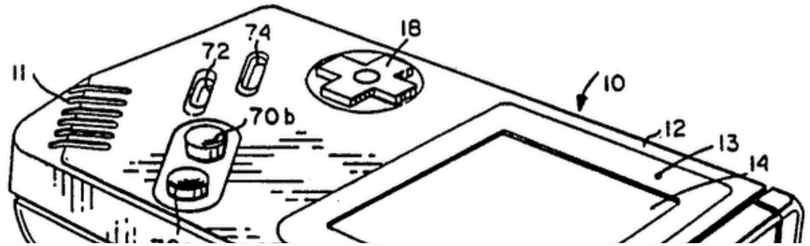
S'abonner

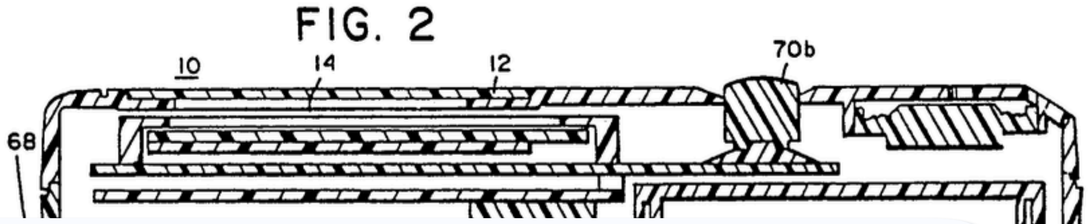
34 k

I was a developer of Shantae for the Nintendo Switch !

Modern Vintage Gamer

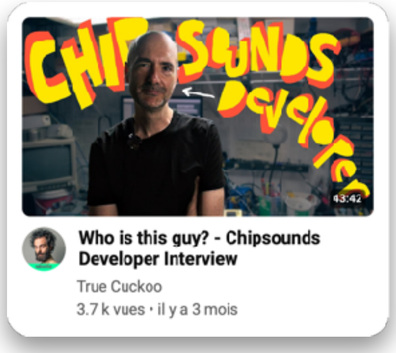
378.8 k vues • il y a 5 ans



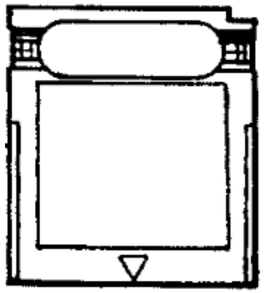


Émulation : « Reproduire le comportement physique d'un matériel par un logiciel, afin de le rendre compatible avec les logiciels d'époque » ≠ **Simulation**

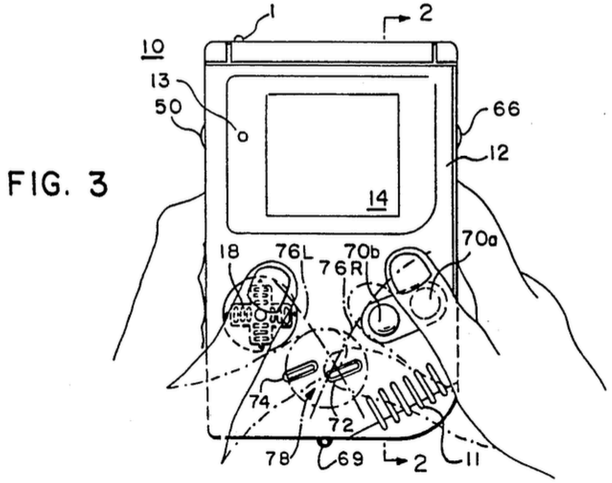
Rétro-engénierie : « Méthodes pour comprendre et expliquer comment un système accomplit une tâche sans connaissance de son fonctionnement exact »



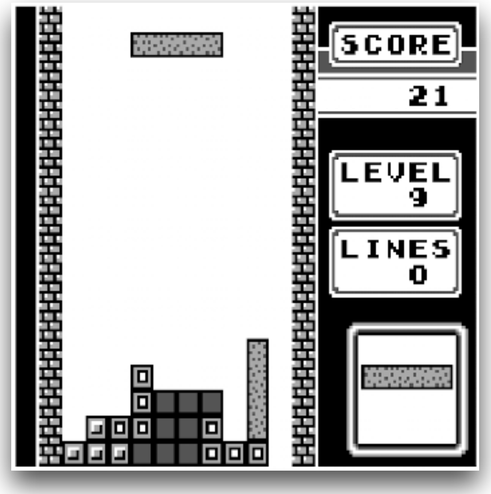
ROM : **R**ead-**O**nly **M**emory, par métonymie est utilisé pour désigner le fichier résultant de l'extraction d'un support physique (cartouche, disque…).



Jeu sur cartouche



Console



Jeu jouable

Extraction « Dump »

ROM

Rétro-ingénierie

Émulateur

Objection!
CAPCOM

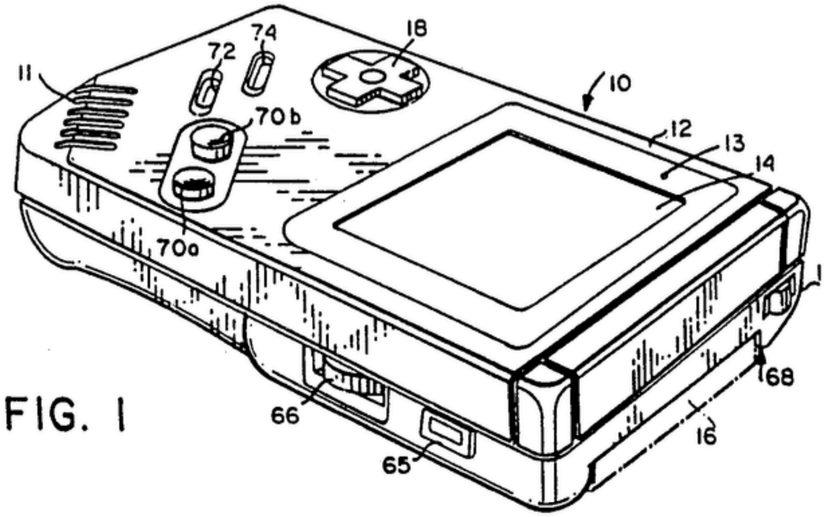


FIG. 1

FIG. 2

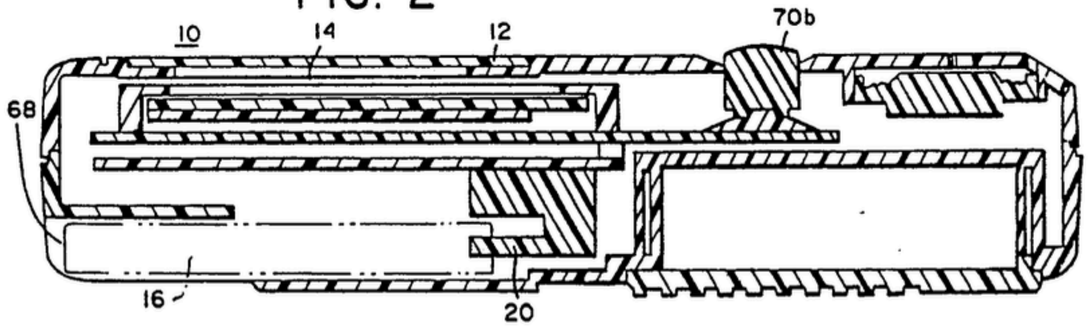
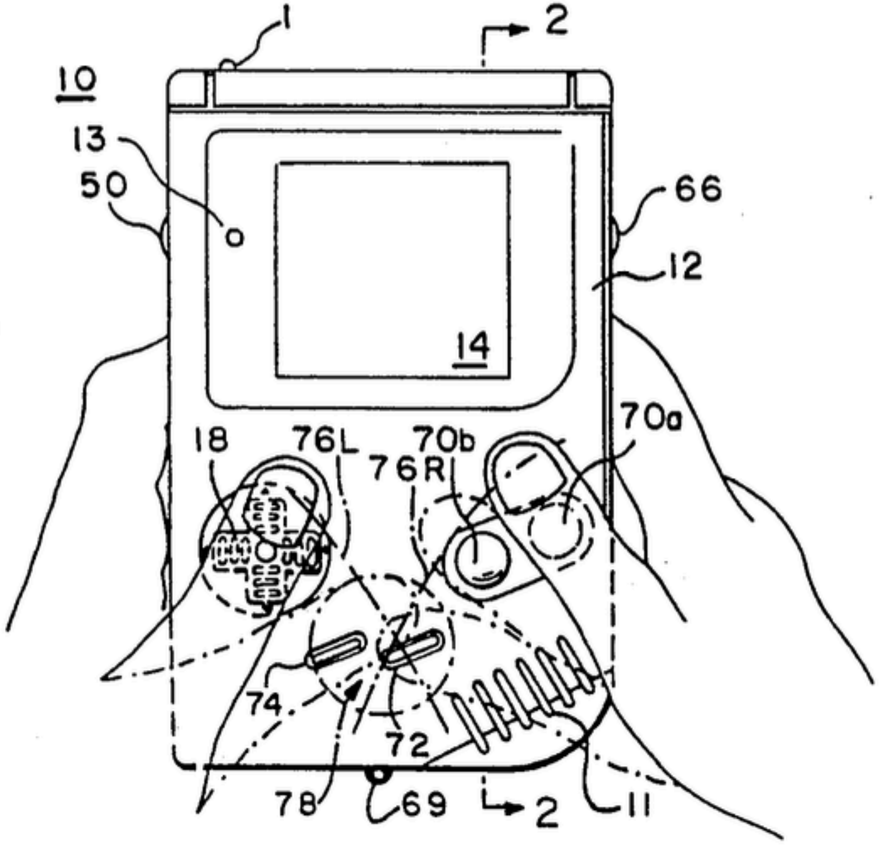


FIG. 3



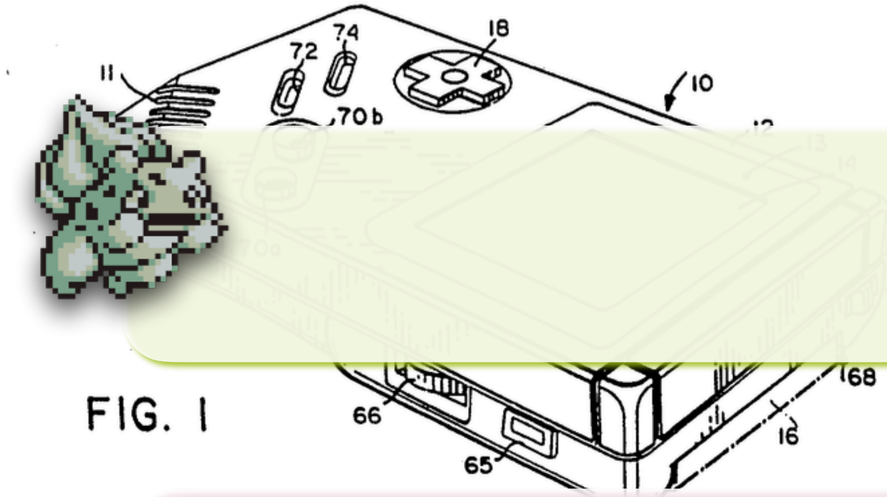


FIG. 1

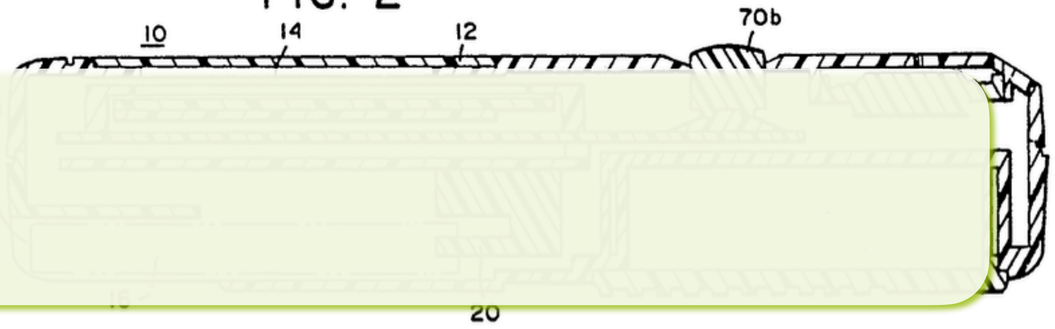


FIG. 2

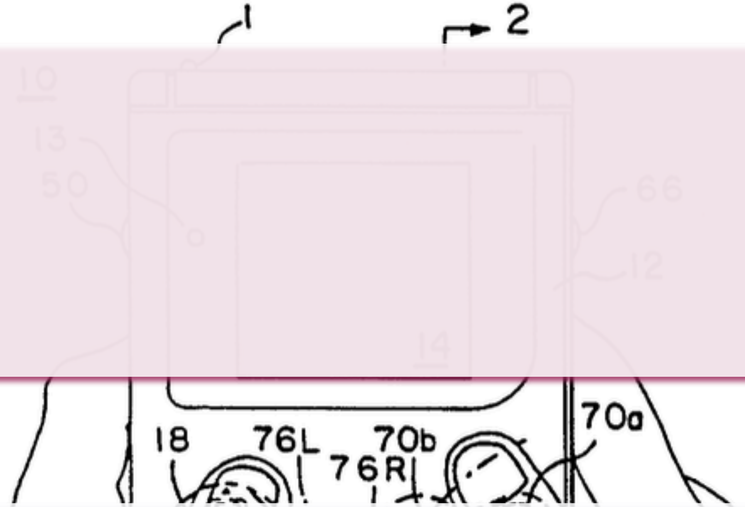
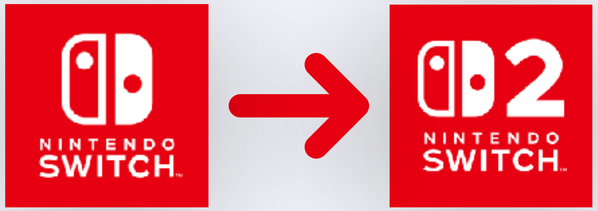
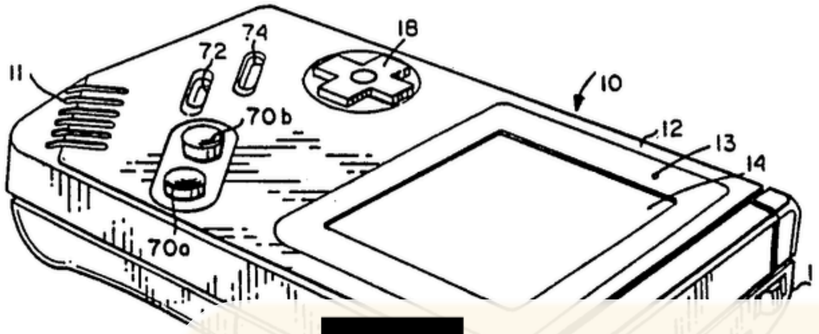


FIG. 3




11
72
74
18
10
70b
12
13
14
70a

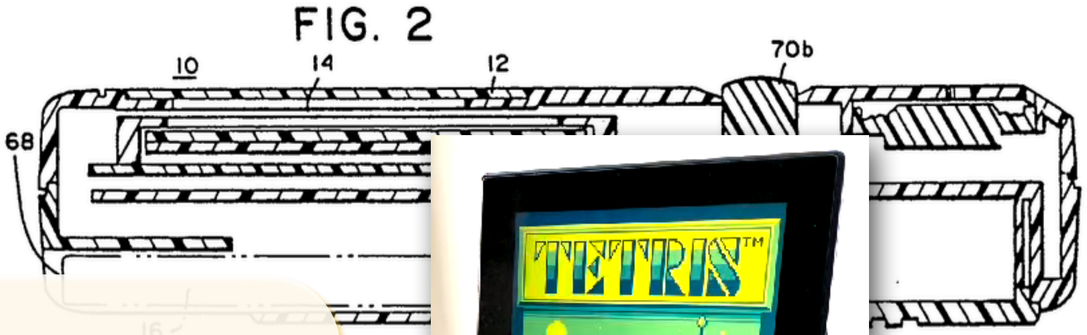


FIG. 2
10
14
12
70b
68

TETRIS™
1PLAYER
2PLAYER
©1989 Nintendo

fpga

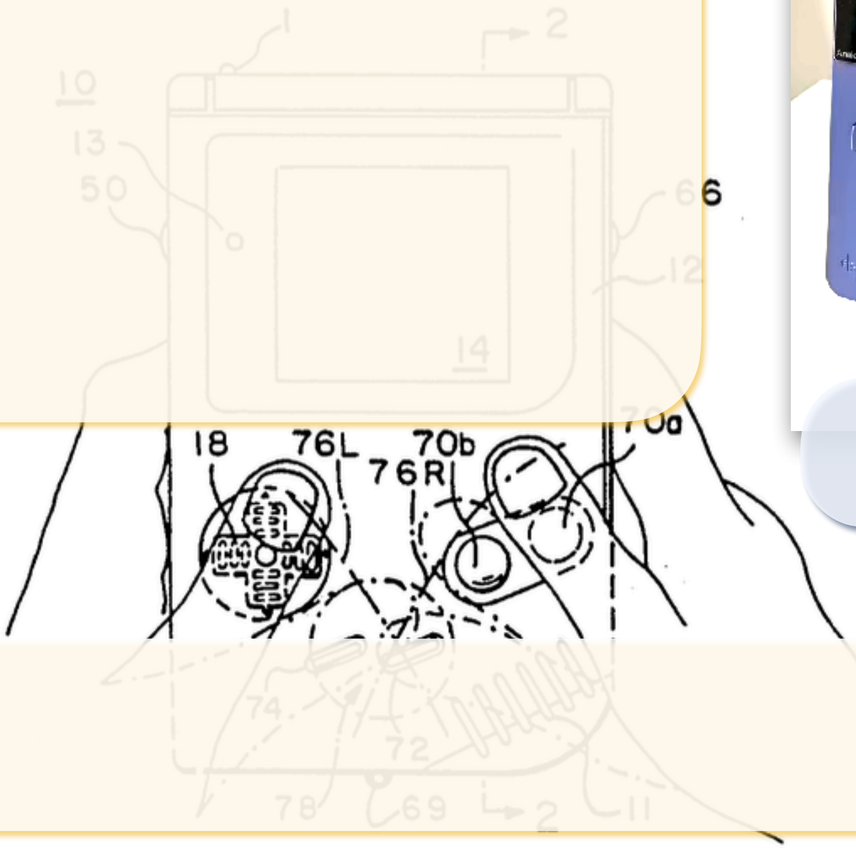
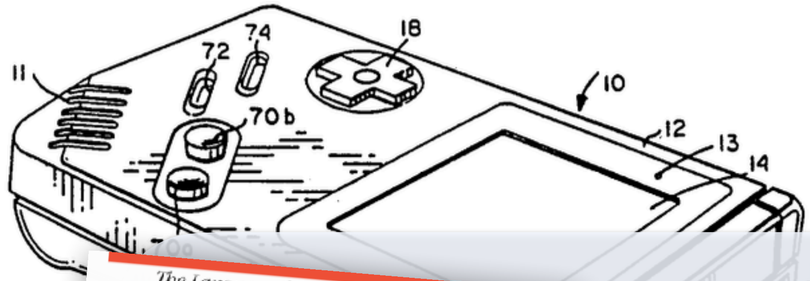
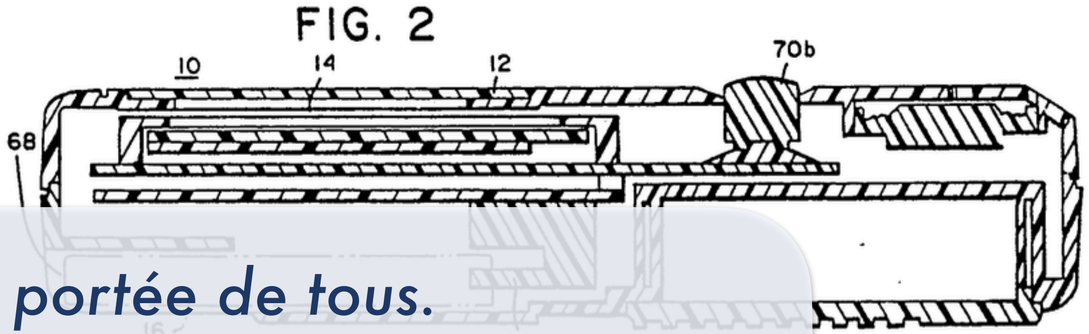
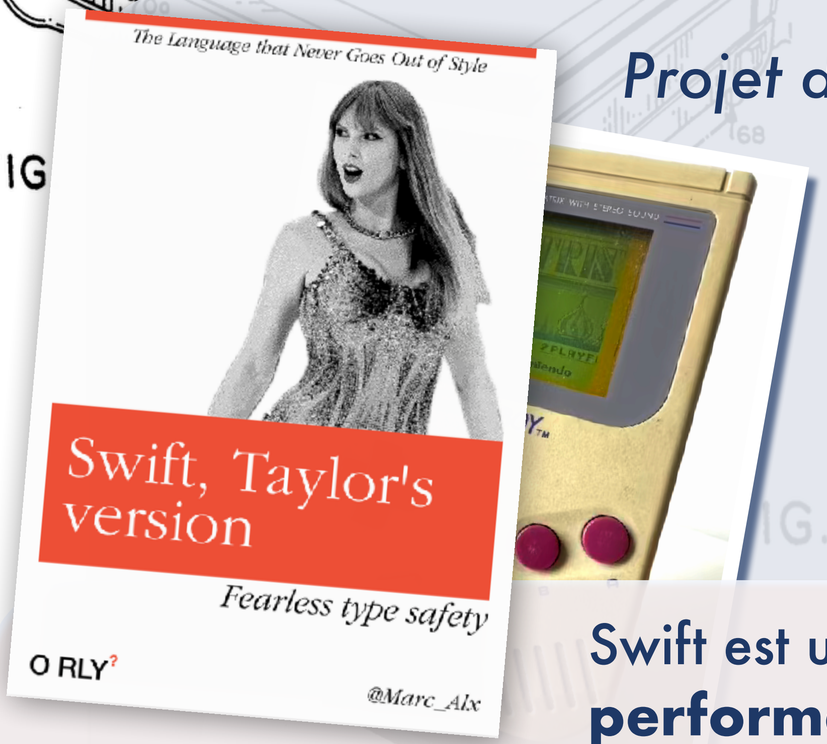


FIG. 3
18
76L
70b
76R
70a







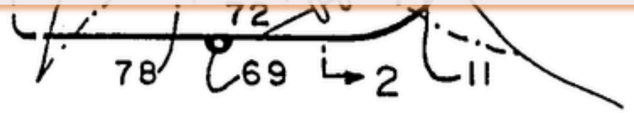
Projet d'émulation à la portée de tous.

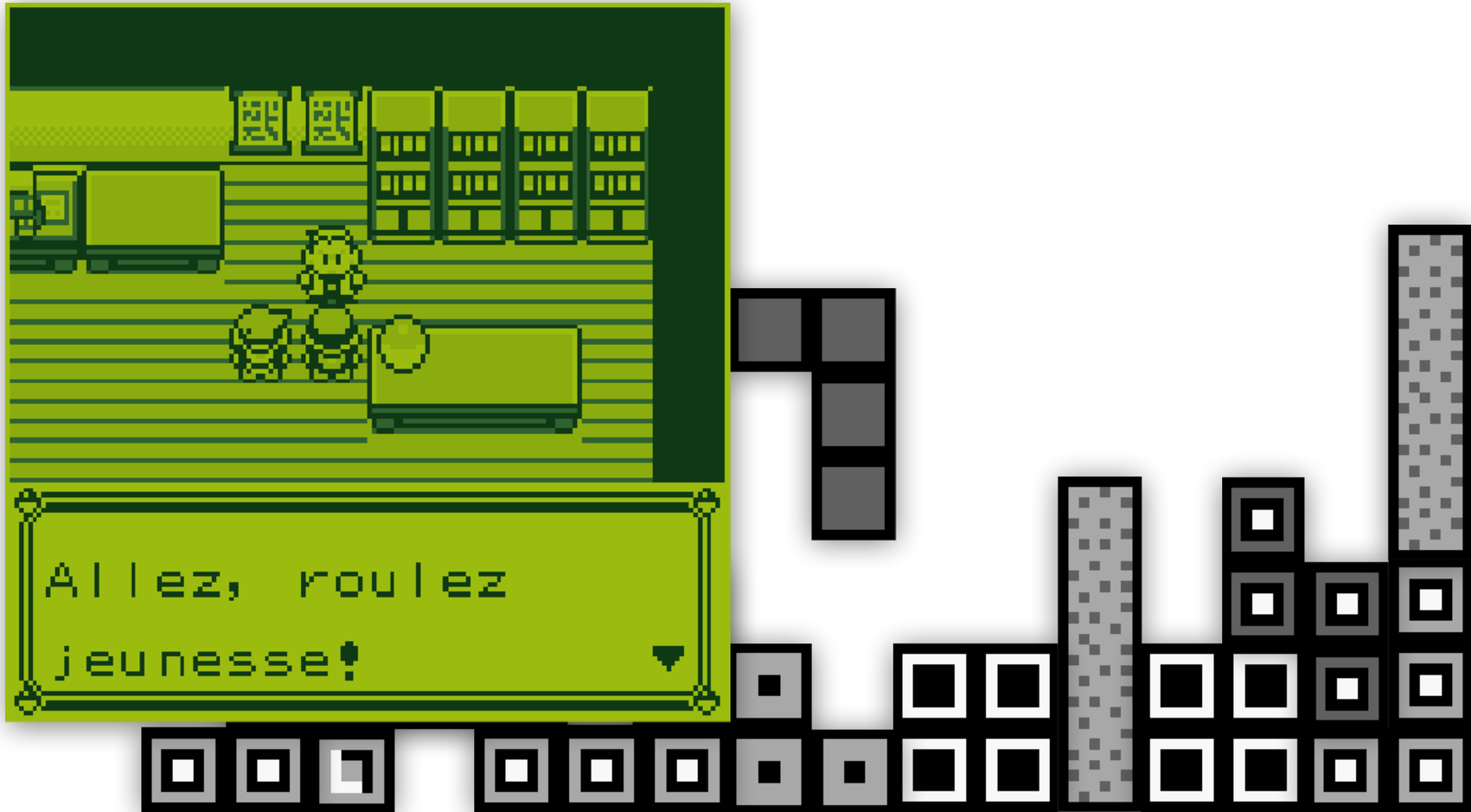
Durée (à peu près™) : **1 à 2 mois**

Fortement documentée *pour son fonctionnement de base*
Communauté très présente

Swift est un langage **open-source d'** **multi-plateforme, performant, sécurisé et moderne**.

Son écosystème riche fournit les principaux blocs pour le rendu graphique, la partie sonore, *l'UI*...



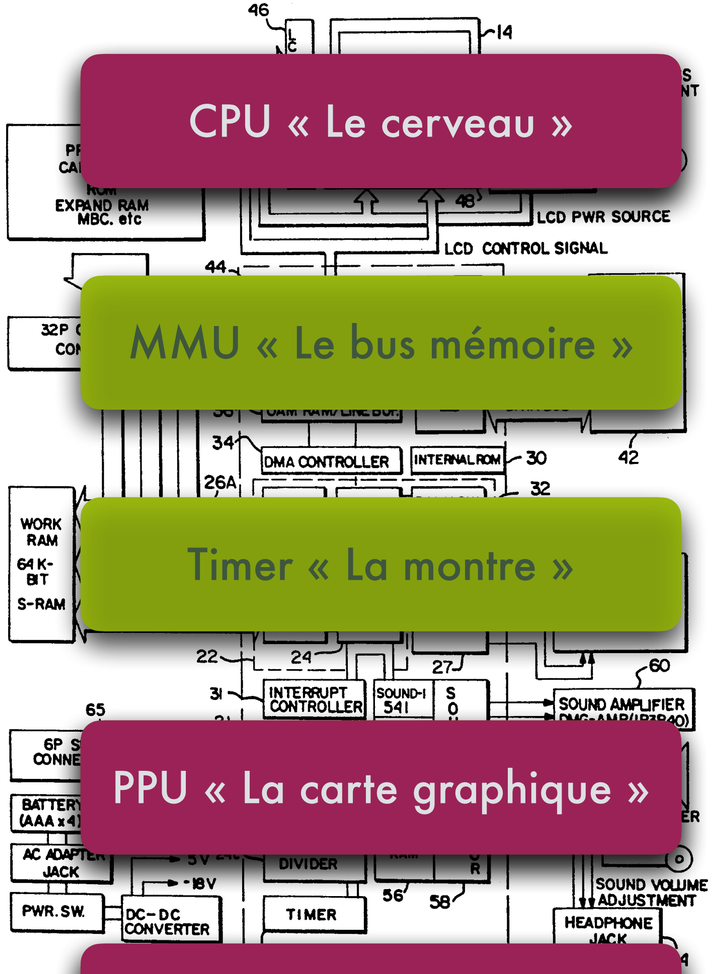

Allez, roulez
jeunesse!

« Représenter ce qui existe »

L'écran

La carte mère

La « manette »



CPU « Le cerveau »

MMU « Le bus mémoire »

Timer « La montre »

PPU « La carte graphique »

APU « Le processeur audio »

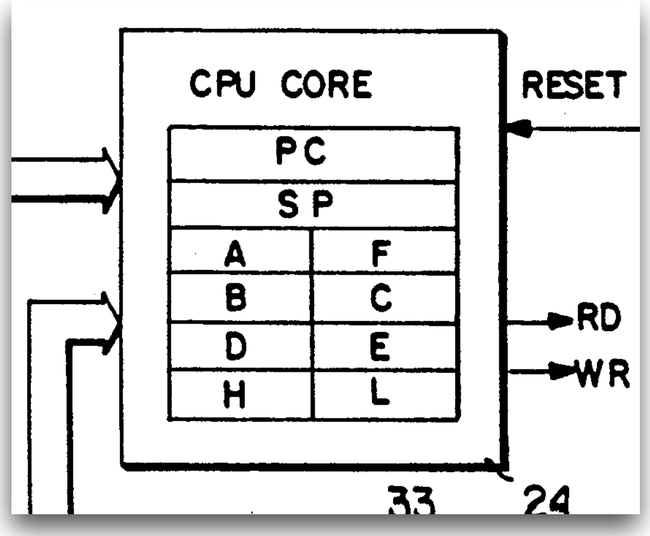
La communication série

Les cartouches de jeu

Abordés dans ce talk

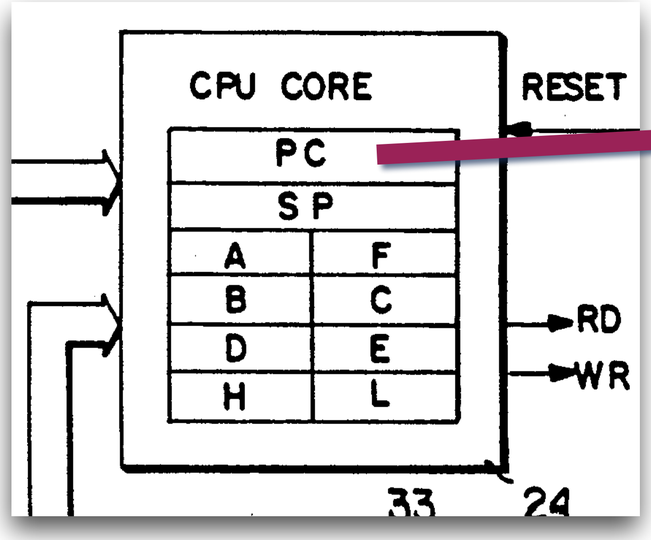
« Here comes troubles »

« Le cerveau »



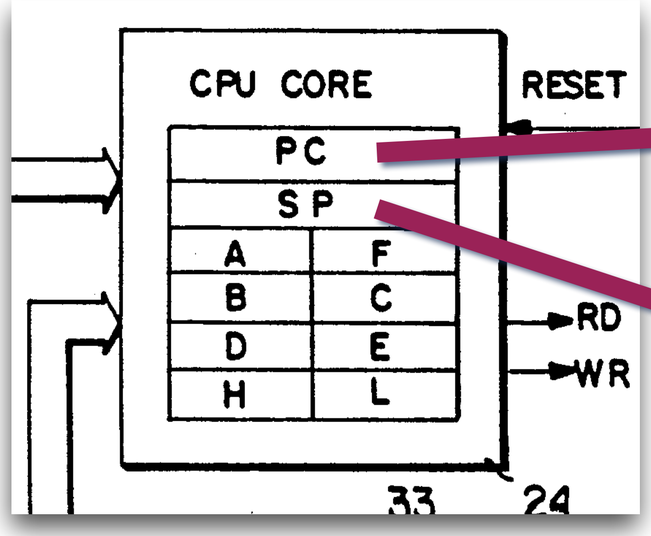
Concept		

« *Le cerveau* »



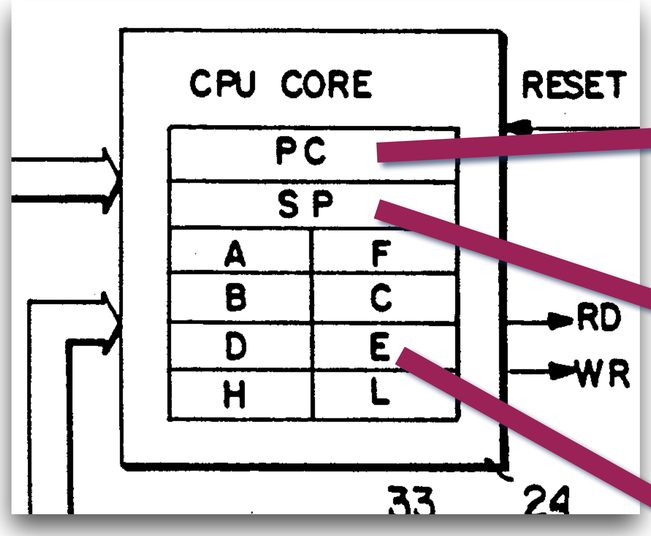
Concept		
PC	**Adresse de la prochaine instruction** à exécuter	Ce que je suis en train de faire

« *Le cerveau* »



Concept		
PC	**Adresse de la prochaine instruction** à exécuter	Ce que je suis en train de faire
SP	**Haut de la pile** : stocke les arguments de fonction	Mémoire moyen terme

« *Le cerveau* »



Concept		
PC	**Adresse de la prochaine instruction** à exécuter	Ce que je suis en train de faire
SP	**Haut de la pile** : stocke les arguments de fonction	Mémoire moyen terme
Registres	« Variables globales », ex **A : Accumulateur, F : Flags** résultats d'opérations arithmétiques.	Mémoire immédiate

Registers.swift

```swift
class Registers: Component,Describable {
  private var _AF:EnhancedShort = EnhancedShort(

  //accumulator
  public var A:Byte {
    get { return self._AF.msb }
    set { self._AF.msb = newValue}
  }
  //flags
  public var F:Byte {
    get { return self._AF.lsb }
    set { self._AF.lsb = newValue &
  }
  //AF
  public var AF:Short {
    get { return self._AF.value }
    set { self._AF.value = newValue & 0xFFF0 }
  }
  ...
}
```

Types.swift

```swift
///Byte is shorthand for UInt8
public typealias Byte = UInt8

///Short is shorthand ife UInt16
public typealias Short = UInt16
///enhanced short
public typealias EnhancedShort = TwoByte

/// two bytes wrapper
public struct TwoBytesWrapper {
    ///most significant bytes
    public var msb:Byte

    ///least significant bytes
    public var lsb:Byte
}
```

Arithmetics.swift

```swift
/// 8bit/byte mask to extract/check bit
enum ByteMask: Byte {
    case Bit_7 = 0b1000_0000
    case Bit_6 = 0b0100_0000
    case Bit_5 = 0b0010_0000
    case Bit_4 = 0b0001_0000
    case Bit_3 = 0b0000_1000
    case Bit_2 = 0b0000_0100
    case Bit_1 = 0b0000_0010
    case Bit_0 = 0b0000_0001
}

/// true if bit identified by mask is 1 in val
func isBitSet(_ mask:ShortMask,_ val:Short) -> Bool {
    return (val & mask.rawValue) > 0
}

/// swap msb and lsb in input
func swap_lsb_msb(_ val:Byte) -> Byte {
    return (val << 4) | (val >> 4)
}
```

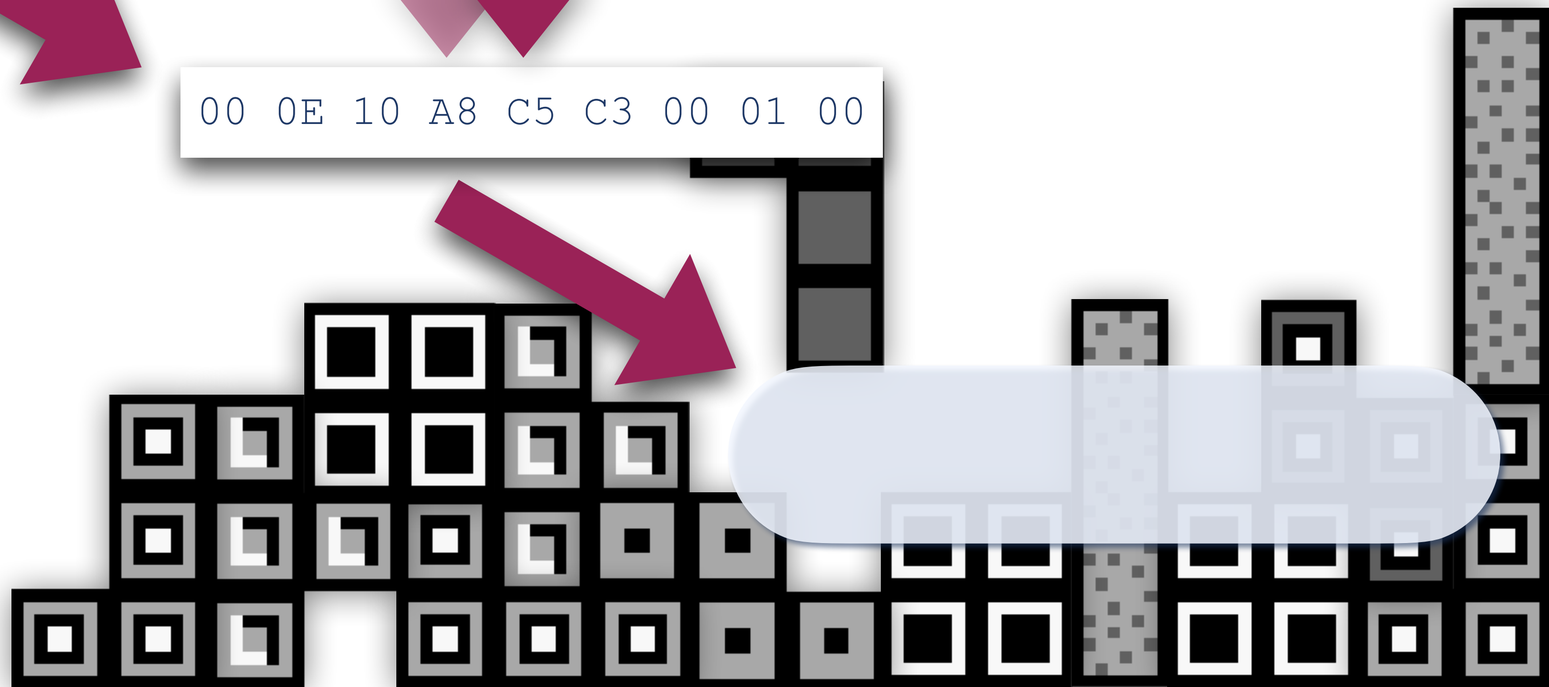
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NOP 1 4 - - - -	LD BC, n16 3 12 - - - -	LD [BC], A 1 8 - - - -	INC BC 1 8 - - - -	INC B 1 4 Z 0 H -	DEC B 1 4 Z 1 H -	LD B, n8 2 8 - - - -	RLCA 1 4 0 0 0 C	LD [a16], SP 3 20 - - - -	ADD HL, BC 1 8 - 0 H C	LD A, [BC] 1 8 - - - -	DEC BC 1 8 - - - -	INC C 1 4 Z 0 H -	DEC C 1 4 Z 1 H -	LD C, n8 2 8 - - - -	RRCA 1 4 0 0 0 C
1x	STOP n8 2 4 - - - -	LD DE, n16 3 12 - - - -	LD [DE], A 1 8 - - - -	INC DE 1 8 - - - -	INC D 1 4 Z 0 H -	DEC D 1 4 Z 1 H -	LD D, n8 2 8 - - - -	RLA 1 4 0 0 0 C	JR e8 2 12 - - - -	ADD HL, DE 1 8 - 0 H C	LD A, [DE] 1 8 - - - -	DEC DE 1 8 - - - -	INC E 1 4 Z 0 H -	DEC E 1 4 Z 1 H -	LD E, n8 2 8 - - - -	RRA 1 4 0 0 0 C
2x	JR NZ, e8 2 12/8 - - - -	LD HL, n16 3 12	LD [HL+], A 1 8	INC HL 1 8	INC H 1 4	DEC H 1 4	LD H, n8 2 8	DAA 1 4 Z - 0 C	JR Z, e8 2 12/8 - - - -	ADD HL, HL 1 8 - 0 H C	LD A, [HL+] 1 8 - - - -	DEC HL 1 8 - - - -	INC L 1 4 Z 0 H -	DEC L 1 4 Z 1 H -	LD L, n8 2 8 - - - -	CPL 1 4 - 1 1 -
3x	JR NC, e8 2 12/8 - - - -							SCF 1 4 - 0 0 1	JR C, e8 2 12/8 - - - -	ADD HL, SP 1 8 - 0 H C	LD A, [HL-] 1 8 - - - -	DEC SP 1 8 - - - -	INC A 1 4 Z 0 H -	DEC A 1 4 Z 1 H -	LD A, n8 2 8 - - - -	CCF 1 4 - 0 0 C
4x	LD B, B 1 4 - - - -							LD B, A 1 4 - - - -	LD C, B 1 4 - - - -	LD C, C 1 4 - - - -	LD C, D 1 4 - - - -	LD C, E 1 4 - - - -	LD C, H 1 4 - - - -	LD C, L 1 4 - - - -	LD C, [HL] 1 8 - - - -	LD C, A 1 4 - - - -
5x	LD D, B 1 4 - - - -							LD D, A 1 4 - - - -	LD E, B 1 4 - - - -	LD E, C 1 4 - - - -	LD E, D 1 4 - - - -	LD E, E 1 4 - - - -	LD E, H 1 4 - - - -	LD E, L 1 4 - - - -	LD E, [HL] 1 8 - - - -	LD E, A 1 4 - - - -
6x	LD H, B 1 4 - - - -							LD H, A 1 4 - - - -	LD L, B 1 4 - - - -	LD L, C 1 4 - - - -	LD L, D 1 4 - - - -	LD L, E 1 4 - - - -	LD L, H 1 4 - - - -	LD L, L 1 4 - - - -	LD L, [HL] 1 8 - - - -	LD L, A 1 4 - - - -
7x	LD [HL], B 1 8 - - - -							LD [HL], A 1 8 - - - -	LD A, B 1 4 - - - -	LD A, C 1 4 - - - -	LD A, D 1 4 - - - -	LD A, E 1 4 - - - -	LD A, H 1 4 - - - -	LD A, L 1 4 - - - -	LD A, [HL] 1 8 - - - -	LD A, A 1 4 - - - -
8x	ADD A, B 1 4 Z 0 H C							ADD A, A 1 4 Z 0 H C	ADC A, B 1 4 Z 0 H C	ADC A, C 1 4 Z 0 H C	ADC A, D 1 4 Z 0 H C	ADC A, E 1 4 Z 0 H C	ADC A, H 1 4 Z 0 H C	ADC A, L 1 4 Z 0 H C	ADC A, [HL] 1 8 Z 0 H C	ADC A, A 1 4 Z 0 H C
9x	SUB A, B 1 4 Z 1 H C	SUB A, C 1 4 Z 1 H C	SUB A, D 1 4 Z 1 H C	SUB A, E 1 4 Z 1 H C	SUB A, H 1 4 Z 1 H C	SUB A, L 1 4 Z 1 H C	SUB A, [HL] 1 8 Z 1 H C	SUB A, A 1 4 1 1 0 0	SBC A, B 1 4 Z 1 H C	SBC A, C 1 4 Z 1 H C	SBC A, D 1 4 Z 1 H C	SBC A, E 1 4 Z 1 H C	SBC A, H 1 4 Z 1 H C	SBC A, L 1 4 Z 1 H C	SBC A, [HL] 1 8 Z 1 H C	SBC A, A 1 4 Z 1 H -
Ax	AND A, B 1 4 Z 0 1 0	AND A, C 1 4 Z 0 1 0	AND A, D 1 4 Z 0 1 0	AND A, E 1 4 Z 0 1 0	AND A, H 1 4 Z 0 1 0	AND A, L 1 4 Z 0 1 0	AND A, [HL] 1 8 Z 0 1 0	AND A, A 1 4 Z 0 1 0	XOR A, B 1 4 Z 0 0 0	XOR A, C 1 4 Z 0 0 0	XOR A, D 1 4 Z 0 0 0	XOR A, E 1 4 Z 0 0 0	XOR A, H 1 4 Z 0 0 0	XOR A, L 1 4 Z 0 0 0	XOR A, [HL] 1 8 Z 0 0 0	XOR A, A 1 4 1 0 0 0
Bx	OR A, B 1 4 Z 0 0 0	OR A, C 1 4 Z 0 0 0	OR A, D 1 4 Z 0 0 0	OR A, E 1 4 Z 0 0 0	OR A, H 1 4 Z 0 0 0	OR A, L 1 4 Z 0 0 0	OR A, [HL] 1 8 Z 0 0 0	OR A, A 1 4 Z 0 0 0	CP A, B 1 4 Z 1 H C	CP A, C 1 4 Z 1 H C	CP A, D 1 4 Z 1 H C	CP A, E 1 4 Z 1 H C	CP A, H 1 4 Z 1 H C	CP A, L 1 4 Z 1 H C	CP A, [HL] 1 8 Z 1 H C	CP A, A 1 4 1 1 0 0
Cx	RET NZ 1 20/8 - - - -	POP BC 1 12 - - - -	JP NZ, a16 3 16/12 - - - -	JP a16 3 16 - - - -	CALL NZ, a16 3 24/12 - - - -	PUSH BC 1 16 - - - -	ADD A, n8 2 8 Z 0 H C	RST $00 1 16 - - - -	RET Z 1 20/8 - - - -	RET 1 16 - - - -	JP Z, a16 3 16/12 - - - -	PREFIX 1 4 - - - -	CALL Z, a16 3 24/12 - - - -	CALL a16 3 24 - - - -	ADC A, n8 2 8 Z 0 H C	RST $08 1 16 - - - -
Dx	RET NC 1 20/8 - - - -	POP DE 1 12 - - - -	JP NC, a16 3 16/12 - - - -	—	CALL NC, a16 3 24/12 - - - -	PUSH DE 1 16 - - - -	SUB A, n8 2 8 Z 1 H C	RST $10 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C, a16 3 16/12 - - - -	—	CALL C, a16 3 24/12 - - - -	—	SBC A, n8 2 8 Z 1 H C	RST $18 1 16 - - - -
Ex	LDH [a8], A 2 12 - - - -	POP HL 1 12 - - - -	LDH [C], A 1 8 - - - -	—	—	PUSH HL 1 16 - - - -	AND A, n8 2 8 Z 0 1 0	RST $20 1 16 - - - -	ADD SP, e8 2 16 0 0 H C	JP HL 1 4 - - - -	LD [a16], A 3 16 - - - -	—	—	—	XOR A, n8 2 8 Z 0 0 0	RST $28 1 16 - - - -
Fx	LDH A, [a8] 2 12 - - - -	POP AF 1 12 Z N H C	LDH A, [C] 1 8 - - - -	DI 1 4 - - - -	—	PUSH AF 1 16 - - - -	OR A, n8 2 8 Z 0 0 0	RST $30 1 16 - - - -	LD HL, SP + e8 2 12 0 0 H C	LD SP, HL 1 8 - - - -	LD A, [a16] 3 16 - - - -	EI 1 4 - - - -	—	—	CP A, n8 2 8 Z 1 H C	RST $38 1 16 - - - -

- Misc / control instructions
- Jumps / calls
- 8-bit load instructions
- 16-bit load instructions
- 8-bit arithmetic / logical instructions
- 16-bit arithmetic / logical instructions
- 8-bit shift, rotate and bit instructions

```swift
private func fetch() -> (OperationCode,[Instruction]) {
    let opCode = self.readIncrPC()
    if opCode == GBConstants.ExtendedInstructionSetOpcode {
        return ((true, self.readIncrPC()),extendedInstructionSet)
    }
    return ((false,opCode),standardInstructionSet)
}
```

PC +1

00 0E 10 A8 C5 C3 00 01 00



	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NOP 1 4 - - - -	LD BC, n16 3 12 - - - -	LD [BC], A 1 8 - - - -	INC BC 1 8 - - - -	INC B 1 4 Z 0 H -	DEC B 1 4 Z 1 H -	LD B, n8 2 8 - - - -	RLCA 1 4 0 0 0 C	LD [a16], SP 3 20 - - - -	ADD HL, BC 1 8 - 0 H C	LD A, [BC] 1 8 - - - -	DEC BC 1 8 - - - -	INC C 1 4 Z 0 H -	DEC C 1 4 Z 1 H -	LD C, n8 2 8 - - - -	RRCA 1 4 0 0 0 C
1x	STOP n8 2 4 - - - -	LD DE, n16 3 12 - - - -	LD [DE], A 1 8 - - - -	INC DE 1 8 - - - -	INC D 1 4 Z 0 H -	DEC D 1 4 Z 1 H -	LD D, n8 2 8 - - - -	RLA 1 4 0 0 0 C	JR e8 2 12 - - - -	ADD HL, DE 1 8 - 0 H C	LD A, [DE] 1 8 - - - -	DEC DE 1 8 - - - -	INC E 1 4 Z 0 H -	DEC E 1 4 Z 1 H -	LD E, n8 2 8 - - - -	RRA 1 4 0 0 0 C
2x	JR NZ, e8 2 12/8 - - - -	LD HL, n16 3 12 - - - -	LD [HL+], A 1 8 - - - -	INC HL 1 8 - - - -	INC H 1 4 Z 0 H -	DEC H 1 4 Z 1 H -	LD H, n8 2 8 - - - -	DAA 1 4 Z - 0 C	JR Z, e8 2 12/8 - - - -	ADD HL, HL 1 8 - 0 H C	LD A, [HL+] 1 8 - - - -	DEC HL 1 8 - - - -	INC L 1 4 Z 0 H -	DEC L 1 4 Z 1 H -	LD L, n8 2 8 - - - -	CPL 1 4 - 1 1 -
3x	JR NC, e8 2 12/8 - - - -	LD SP, n16 3 12 - - - -	LD [HL-], A 1 8 - - - -	INC SP 1 8 - - - -	INC [HL] 1 12 Z 0 H -	DEC [HL] 1 12 Z 1 H -	LD [HL], n8 2 12 - - - -	SCF 1 4 - 0 0 1	JR C, e8 2 12/8 - - - -	ADD HL, SP 1 8 - 0 H C	LD A, [HL-] 1 8 - - - -	DEC SP 1 8 - - - -	INC A 1 4 Z 0 H -	DEC A 1 4 Z 1 H -	LD A, n8 2 8 - - - -	CCF 1 4 - 0 0 C
4x	LD B, B 1 4 - - - -	LD B, C 1 4 - - - -	LD B, D 1 4 - - - -	LD B, E 1 4 - - - -	LD B, H 1 4 - - - -	LD B, L 1 4 - - - -	LD B, [HL] 1 8 - - - -	LD B, A 1 4 - - - -	LD C, B 1 4 - - - -	LD C, C 1 4 - - - -	LD C, D 1 4 - - - -	LD C, E 1 4 - - - -	LD C, H 1 4 - - - -	LD C, L 1 4 - - - -	LD C, [HL] 1 8 - - - -	LD C, A 1 4 - - - -
5x	LD D, B 1 4 - - - -	LD D, C 1 4 - - - -	LD D, D 1 4 - - - -	LD D, E 1 4 - - - -	LD D, H 1 4 - - - -	LD D, L 1 4 - - - -	LD D, [HL] 1 8 - - - -	LD D, A 1 4 - - - -	LD E, B 1 4 - - - -	LD E, C 1 4 - - - -	LD E, D 1 4 - - - -	LD E, E 1 4 - - - -	LD E, H 1 4 - - - -	LD E, L 1 4 - - - -	LD E, [HL] 1 8 - - - -	LD E, A 1 4 - - - -
6x	LD H, B 1 4 - - - -	LD H, C 1 4 - - - -	LD H, D 1 4 - - - -	LD H, E 1 4 - - - -	LD H, H 1 4 - - - -	LD H, L 1 4 - - - -	LD H, [HL] 1 8 - - - -	LD H, A 1 4 - - - -	LD L, B 1 4 - - - -	LD L, C 1 4 - - - -	LD L, D 1 4 - - - -	LD L, E 1 4 - - - -	LD L, H 1 4 - - - -	LD L, L 1 4 - - - -	LD L, [HL] 1 8 - - - -	LD L, A 1 4 - - - -
7x	LD [HL], B 1 8 - - - -	LD [HL], C 1 8 - - - -	LD [HL], D 1 8 - - - -	LD [HL], E 1 8 - - - -	LD [HL], H 1 8 - - - -	LD [HL], L 1 8 - - - -	HALT 1 4 - - - -	LD [HL], A 1 8 - - - -	LD A, B 1 4 - - - -	LD A, C 1 4 - - - -	LD A, D 1 4 - - - -	LD A, E 1 4 - - - -	LD A, H 1 4 - - - -	LD A, L 1 4 - - - -	LD A, [HL] 1 8 - - - -	LD A, A 1 4 - - - -
8x	ADD A, B 1 4 Z 0 H C	ADD A, C 1 4 Z 0 H C	ADD A, D 1 4 Z 0 H C	ADD A, E 1 4 Z 0 H C	ADD A, H 1 4 Z 0 H C	ADD A, L 1 4 Z 0 H C	ADD A, [HL] 1 8 Z 0 H C	ADD A, A 1 4 Z 0 H C	ADC A, B 1 4 Z 0 H C	ADC A, C 1 4 Z 0 H C	ADC A, D 1 4 Z 0 H C	ADC A, E 1 4 Z 0 H C	ADC A, H 1 4 Z 0 H C	ADC A, L 1 4 Z 0 H C	ADC A, [HL] 1 8 Z 0 H C	ADC A, A 1 4 Z 0 H C
9x	SUB A, B 1 4 Z 1 H C	SUB A, C 1 4 Z 1 H C	SUB A, D 1 4 Z 1 H C	SUB A, E 1 4 Z 1 H C	SUB A, H 1 4 Z 1 H C	SUB A, L 1 4 Z 1 H C	SUB A, [HL] 1 8 Z 1 H C	SUB A, A 1 4 1 1 0 0	SBC A, B 1 4 Z 1 H C	SBC A, C 1 4 Z 1 H C	SBC A, D 1 4 Z 1 H C	SBC A, E 1 4 Z 1 H C	SBC A, H 1 4 Z 1 H C	SBC A, L 1 4 Z 1 H C	SBC A, [HL] 1 8 Z 1 H C	SBC A, A 1 4 Z 1 H -
Ax	AND A, B 1 4 Z 0 1 0	AND A, C 1 4 Z 0 1 0	AND A, D 1 4 Z 0 1 0	AND A, E 1 4 Z 0 1 0	AND A, H 1 4 Z 0 1 0	AND A, L 1 4 Z 0 1 0	AND A, [HL] 1 8 Z 0 1 0	AND A, A 1 4 Z 0 1 0	XOR A, B 1 4 Z 0 0 0	XOR A, C 1 4 Z 0 0 0	XOR A, D 1 4 Z 0 0 0	XOR A, E 1 4 Z 0 0 0	XOR A, H 1 4 Z 0 0 0	XOR A, L 1 4 Z 0 0 0	XOR A, [HL] 1 8 Z 0 0 0	XOR A, A 1 4 1 0 0 0
Bx	OR A, B 1 4 Z 0 0 0	OR A, C 1 4 Z 0 0 0	OR A, D 1 4 Z 0 0 0	OR A, E 1 4 Z 0 0 0	OR A, H 1 4 Z 0 0 0	OR A, L 1 4 Z 0 0 0	OR A, [HL] 1 8 Z 0 0 0	OR A, A 1 4 Z 0 0 0	CP A, B 1 4 Z 1 H C	CP A, C 1 4 Z 1 H C	CP A, D 1 4 Z 1 H C	CP A, E 1 4 Z 1 H C	CP A, H 1 4 Z 1 H C	CP A, L 1 4 Z 1 H C	CP A, [HL] 1 8 Z 1 H C	CP A, A 1 4 1 1 0 0
Cx	RET NZ 1 20/8 - - - -	POP BC 1 12 - - - -	JP NZ, a16 3 16/12 - - - -	JP a16 3 16 - - - -	CALL NZ, a16 3 24/12 - - - -	PUSH BC 1 16 - - - -	ADD A, n8 2 8 Z 0 H C	RST $00 1 16 - - - -	RET Z 1 20/8 - - - -	RET 1 16 - - - -	JP Z, a16 3 16/12 - - - -	PREFIX 1 4 - - - -	CALL Z, a16 3 24/12 - - - -	CALL a16 3 24 - - - -	ADC A, n8 2 8 Z 0 H C	RST $08 1 16 - - - -
Dx	RET NC 1 20/8 - - - -	POP DE 1 12 - - - -	JP NC, a16 3 16/12 - - - -	—	CALL NC, a16 3 24/12 - - - -	PUSH DE 1 16 - - - -	SUB A, n8 2 8 Z 1 H C	RST $10 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C, a16 3 16/12 - - - -	—	CALL C, a16 3 24/12 - - - -	—	SBC A, n8 2 8 Z 1 H C	RST $18 1 16 - - - -
Ex	LDH [a8], A 2 12 - - - -	POP HL 1 12 - - - -	LDH [C], A 1 8 - - - -	—	—	PUSH HL 1 16 - - - -	AND A, n8 2 8 Z 0 1 0	RST $20 1 16 - - - -	ADD SP, e8 2 16 0 0 H C	JP HL 1 4 - - - -	LD [a16], A 3 16 - - - -	—	—	—	XOR A, n8 2 8 Z 0 0 0	RST $28 1 16 - - - -
Fx	LDH A, [a8] 2 12 - - - -	POP AF 1 12 Z N H C	LDH A, [C] 1 8 - - - -	DI 1 4 - - - -	—	PUSH AF 1 16 - - - -	OR A, n8 2 8 Z 0 0 0	RST $30 1 16 - - - -	LD HL, SP + e8 2 12 0 0 H C	LD SP, HL 1 8 - - - -	LD A, [a16] 3 16 - - - -	EI 1 4 - - - -	—	—	CP A, n8 2 8 Z 1 H C	RST $38 1 16 - - - -

CPU.swift

```swift
private func execute(instruction:Instruction) -> Int {
    switch(instruction.length) {
    case InstructionLength.OneByte:
        instruction.execute()
        break
    case InstructionLength.TwoBytes:
        let arg = self.readIncrPC()
        instruction.execute(arg)
        break
    case InstructionLength.ThreeBytes:
        let lsb = self.readIncrPC()
        let msb = self.readIncrPC()
        let arg = EnhancedShort(lsb,msb)
        instruction.execute(arg)
        break
    }
    //return cycle overhead
    return self.willCycleOverhead

}
```

PC + 0 (argument)

XOR A, B
1 4
Z 0 0 0

00 0E 10 A8 C5 C3 00 01 00

CPUCore.swift

```swift
func xor_a_b() -> Void { self.xor_a(self.registers.B) }
func xor_a_c() -> Void { self.xor_a(self.registers.C) }
func xor_a_d() -> Void { self.xor_a(self.registers.D) }
func xor_a_e() -> Void { self.xor_a(self.regi
func xor_a_h() -> Void { self.xor_a(self.regi
func xor_a_l() -> Void { self.xor_a(self.regi
func xor_a_hlp() -> Void { self.xor_a(mmu.rea
func xor_a_a() -> Void { self.xor_a(self.regi
```

```swift
/// xor val with A then stores result in A
internal func xor_a(_ val: Byte) {
    self.registers.A ^= val
    self.registers.conditionalSet(cond: self.registers.A == 0, flag: .ZERO)
    self.registers.clearFlags(.NEGATIVE,.HALF_CARRY,.CARRY)
}
```

« Les tests en boîte noire »

dmg-acid2

dmg-acid2 is a test for developers of Game Boy (DMG) emulators to verify their emulation of the Game Boy's Pixel Processing Unit (PPU).

Download the ROM, or build using RGBDS:

```
git clone --recurse-submodules https://github.com/mattcurrie/dmg-acid2
cd dmg-acid2 && make
```

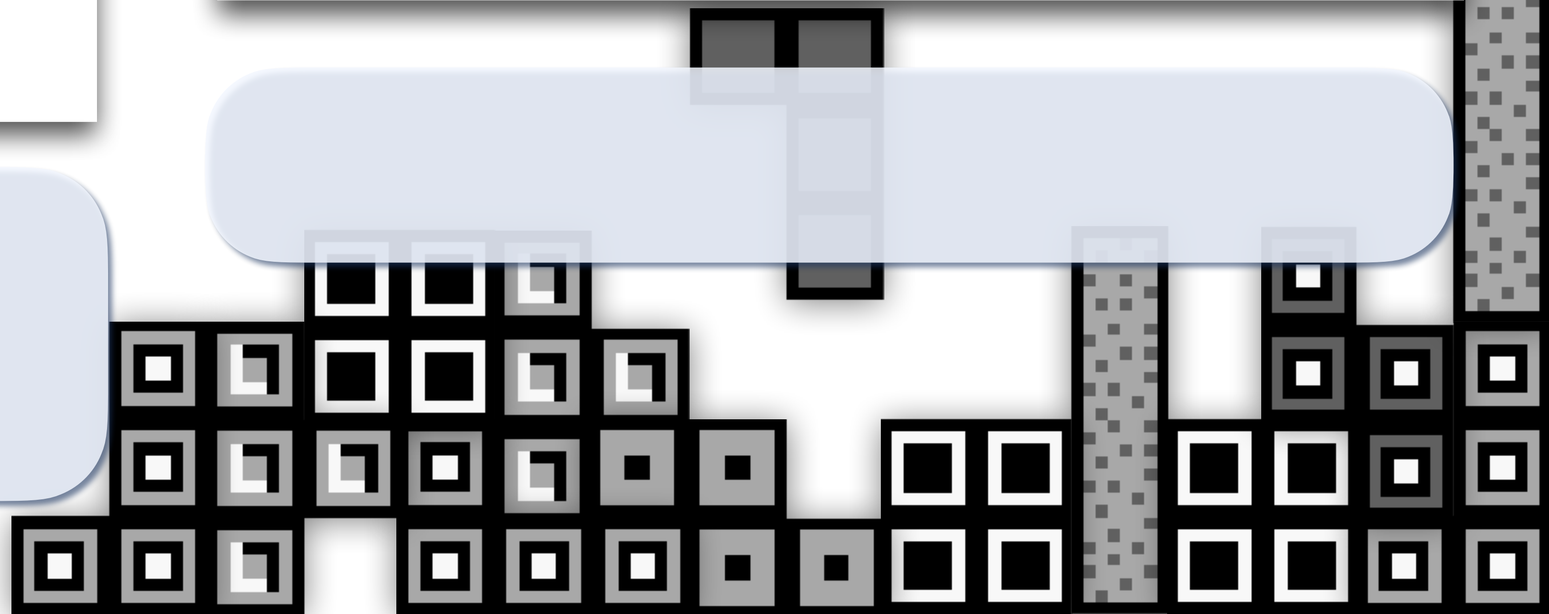
For testing the Game Boy Color you can try cgb-acid2.

Reference Image

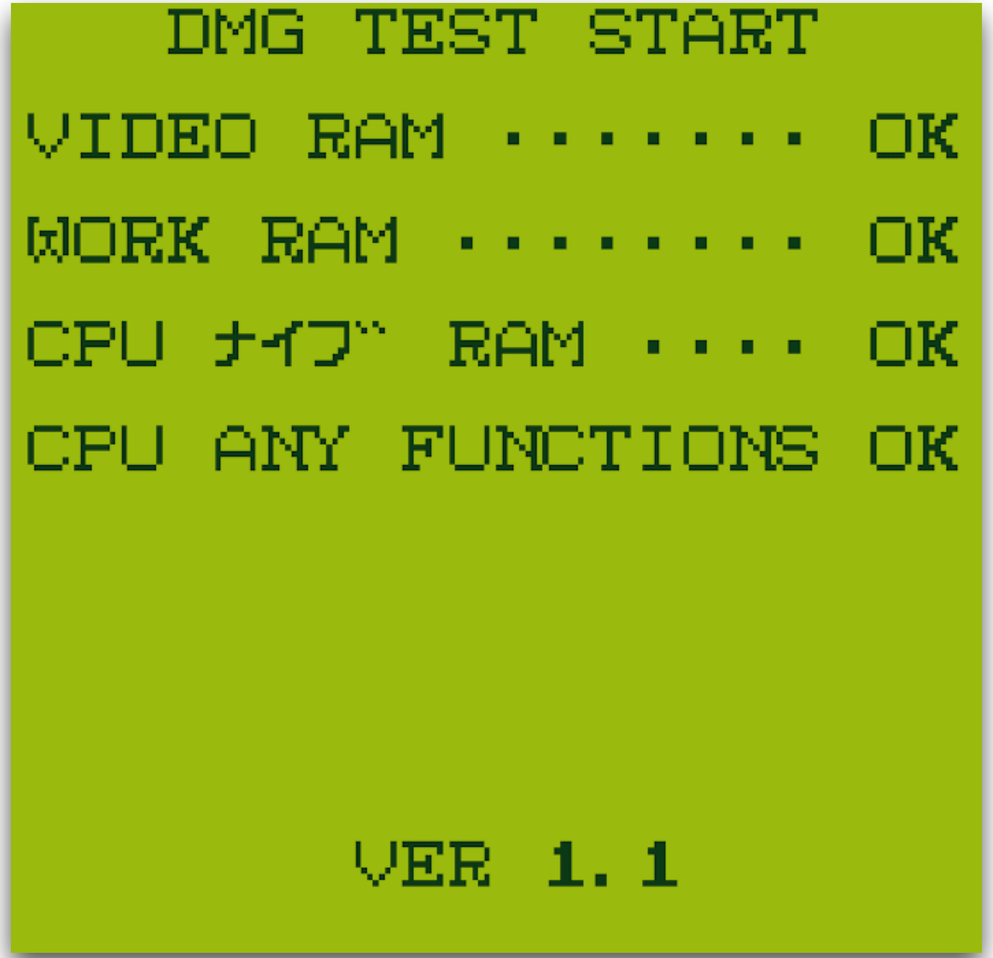
An accurate DMG emulator should generate output identical to the image below:

README

```
Blargg's Gameboy hardware test ROMs. Originally hosted at http://blargg.parodius.com/gb-tests/
before parodious.com went down.
New official location: http://blargg.8bitalley.com/parodius/gb-tests/
```

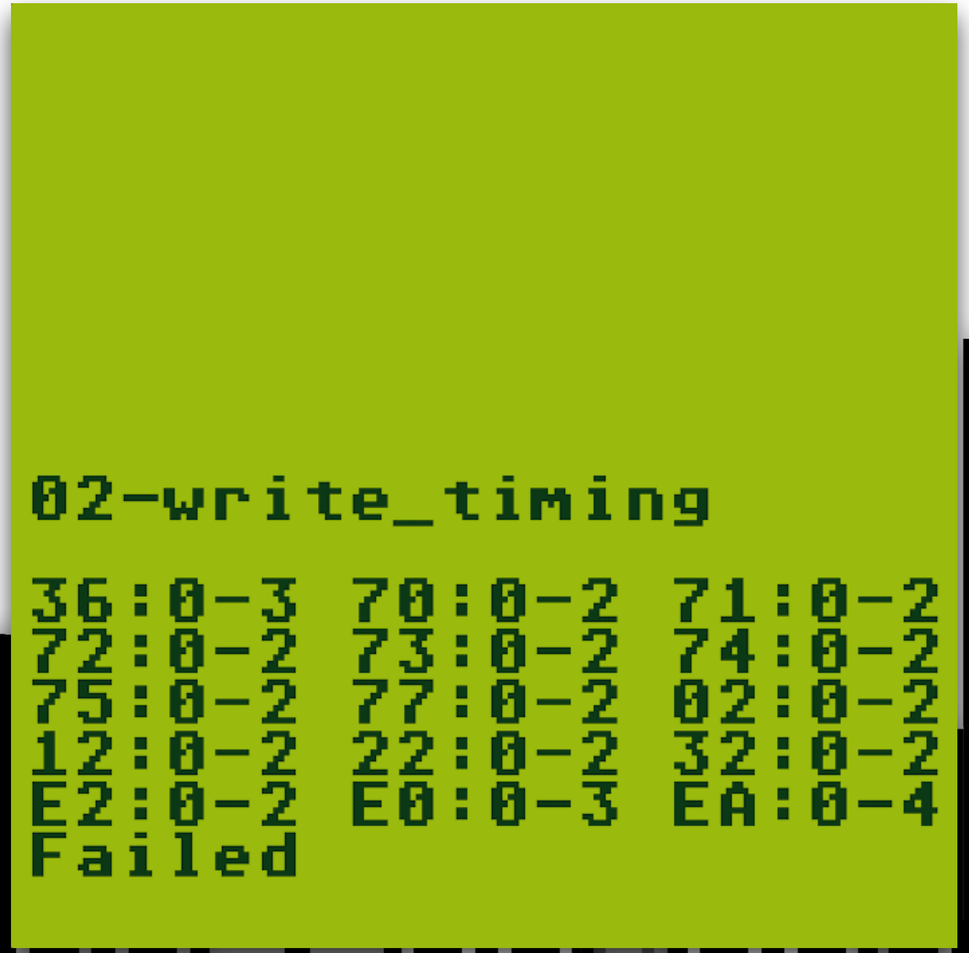


« Les tests en boite noire »



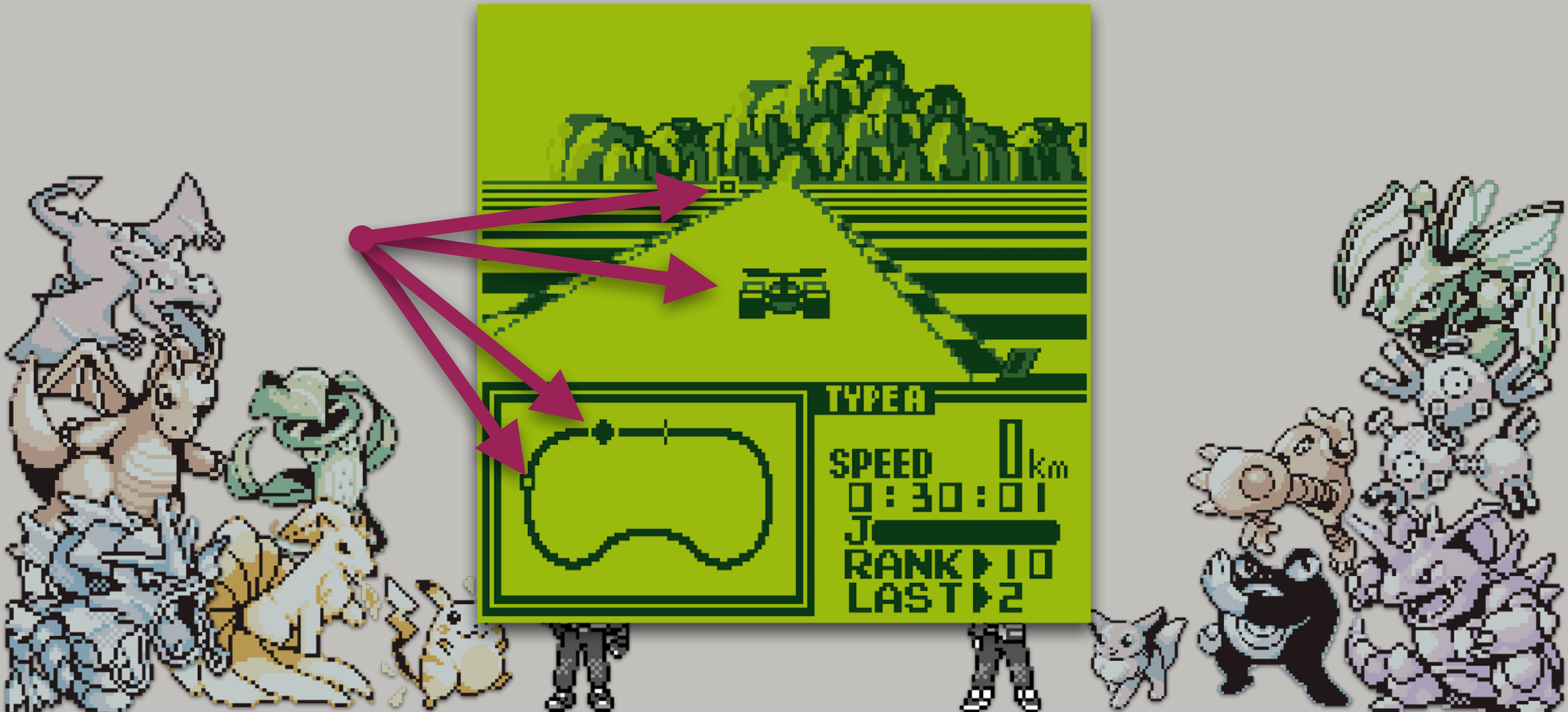
Test ROM Nintendo : aging.gb

Test ROM communautaire

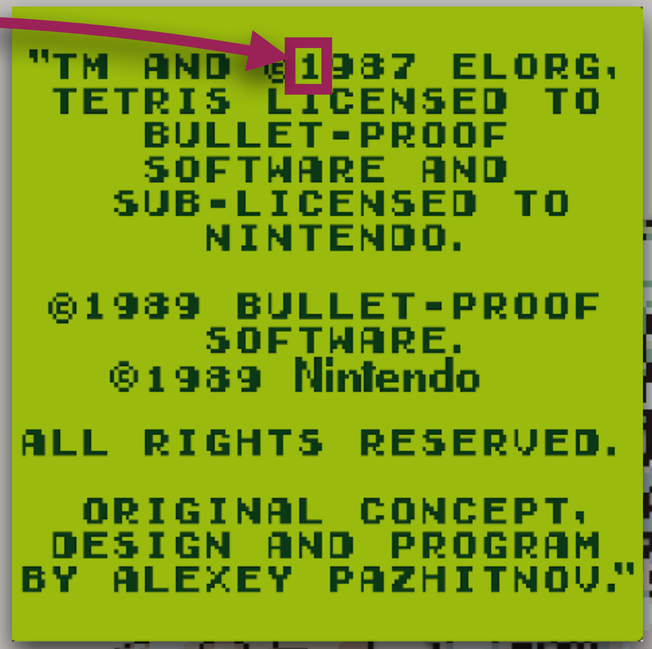


« Let there be light »

TYPE A
SPEED 0 km
0:22:00
J
RANK ▶10
LAST ▶2


TYPE A
SPEED 0 km
0:30:01
RANK ▶10
LAST ▶2

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x8000:	00	00	3c	3c	66	66	66	66	66	66	66	66	3c	3c	00	00
0x8010:	00	00	18	18	38	38	18	18	18	18	18	18	3c	3c	00	00
0x8020:	00	00	3c	3c	4e	4e	0e	0e	3c	3c	70	70	7e	7e	00	00
0x8030:	00	00	7c	7c	0e	0e	3c	3c	0e	0e	0e	0e	7c	7c	00	00
0x8040:	00	00	3c	3c	6c	6c	4c	4c	4e	4e	7e	7e	0c	0c	00	00
0x8050:	00	00	7c	7c	60	60	7c	7c	0e	0e	4e	4e	3c	3c	00	00
0x8060:	00	00	3c	3c	60	60	7c	7c	66	66	66	66	3c	3c	00	00
0x8070:	00	00	7e	7e	06	06	0c	0c	18	18	38	38	38	38	00	00
0x8080:	00	00	3c	3c	4e	4e	3c	3c	4e	4e	4e	4e	3c	3c	00	00
0x8090:	00	00	3c	3c	4e	4e	4e	4e	3e	3e	0e	0e	3c	3c	00	00
0x80a0:	00	00	3c	3c	4e	4e	4e	4e	7e	7e	4e	4e	4e	4e	00	00
0x80b0:	00	00	7c	7c	66	66	7c	7c	66	66	66	66	7c	7c	00	00
0x80c0:	00	00	3c	3c	66	66	60	60	60	60	66	66	3c	3c	00	00
0x80d0:	00	00	7c	7c	4e	4e	4e	4e	4e	4e	4e	4e	7c	7c	00	00
0x80e0:	00	00	7e	7e	60	60	7c	7c	60	60	60	60	7e	7e	00	00
0x80f0:	00	00	7e	7e	60	60	60	60	7c	7c	60	60	60	60	00	00



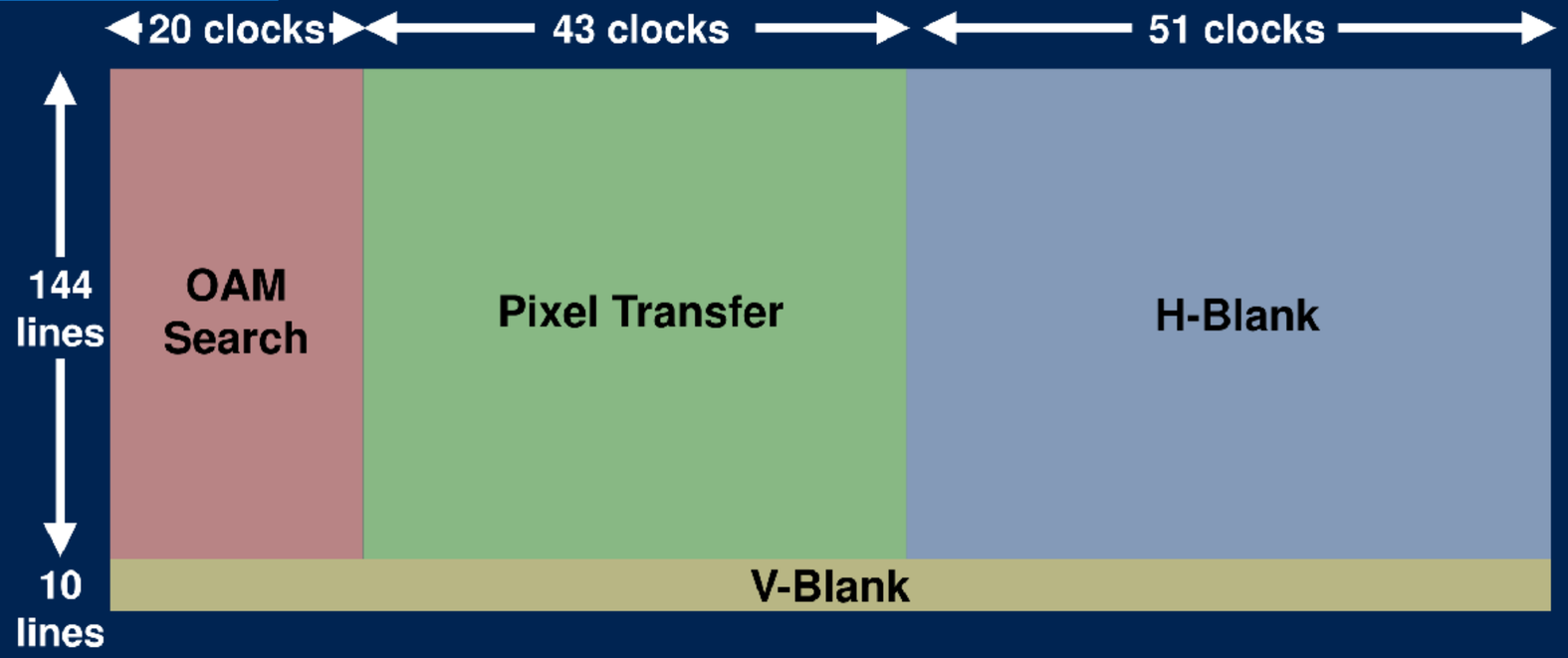
Exemple approximatif

			abcdefgh		ijklmnop		ia jb kc ld me nf og ph
$3C	$7E	=	00111100	+	01111110	=	00 10 11 11 11 11 10 00
$42	$42		01000010		01000010		00 11 00 00 00 00 11 00
$42	$42		01000010		01000010		00 11 00 00 00 00 11 00
$42	$42		01000010		01000010		00 11 00 00 00 00 11 00
$7E	$5E		01111110		01011110		00 11 01 11 11 11 11 00
$7E	$0A		01111110		00001010		00 01 01 01 11 01 11 00
$7C	$56		01111100		01010110		00 11 01 11 01 11 10 00
$38	$7C		00111000		01111100		00 10 11 11 11 10 00 00
			abcdefgh		ijklmnop		ia jb kc ld me nf og ph

Sample tile data

```
public struct ColorPalette {
    private var values:[Color]

    public subscript(colorIndex:Int) -> Color {
        get {
            return self.values[colorIndex]
        }
        set {
            self.values[colorIndex] = newValue
        }
    }
}

/// from two bytes and a ByteMask identifies color to use from palette
/// following the bit blending rule the GB uses
private func decodeColor(palette:ColorPalette, b1:Byte, b2:Byte, at:ByteMask) -> (Int,Color) {
  var color = 0
  if(isBitSet(at, b1)) {
      color += 1
  }
  if(isBitSet(at, b2)) {
      color += 2
  }
  return (color,palette[color])
}
```

PPU Timing



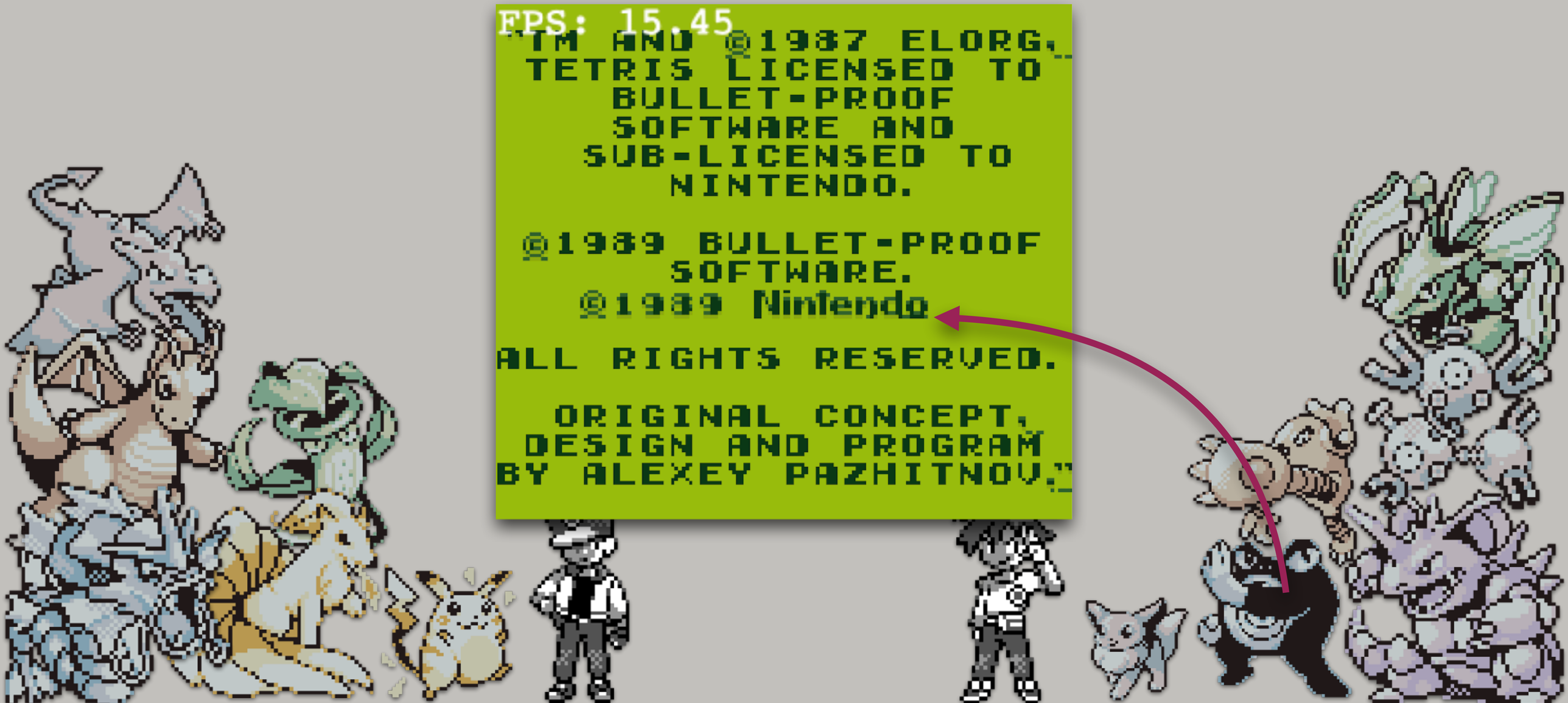
114 x 154 = 17,556 clocks per screen

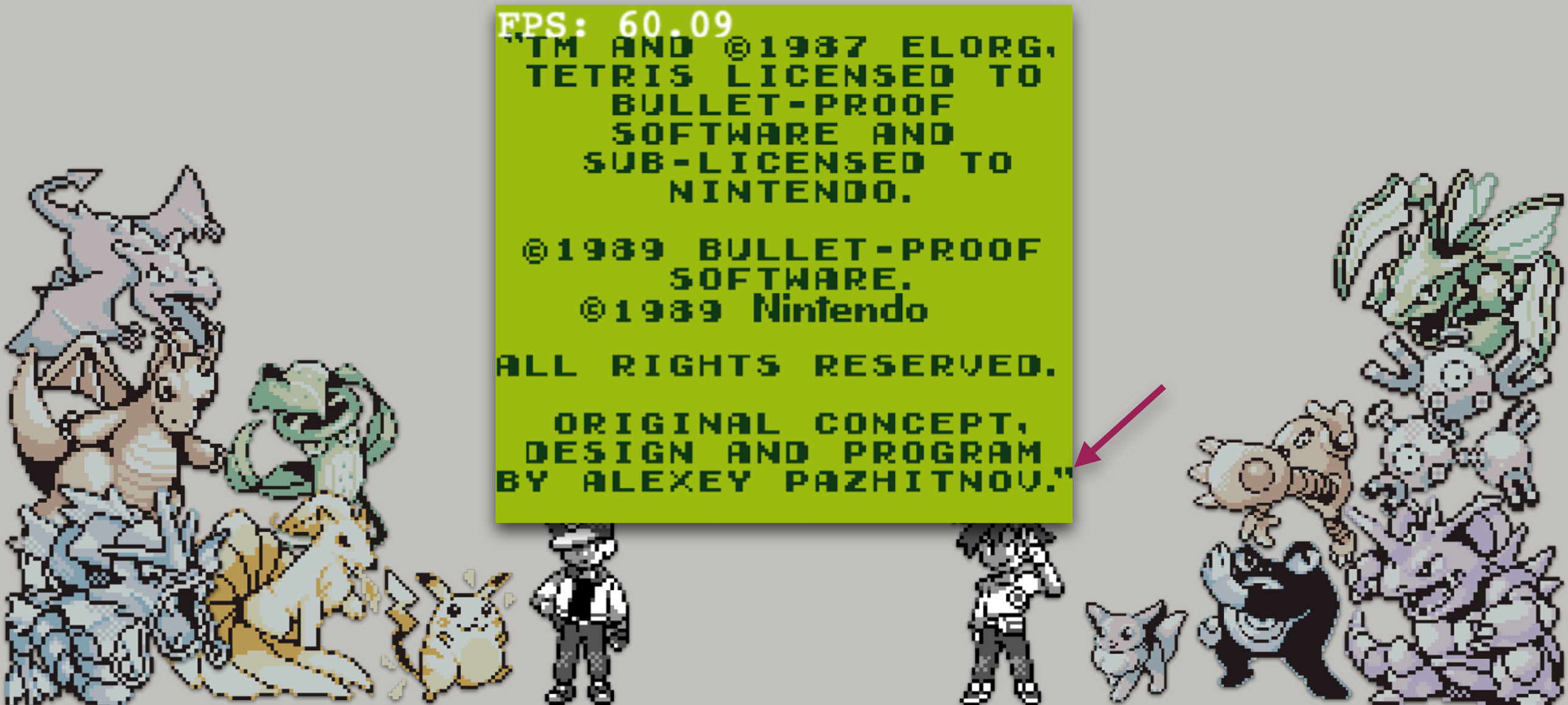
1,048,576 / 17,556 = 59.7 Hz refresh rate

PPU.swift

```swift
private func scanline() -> Void {

  // ... BG+WIN

  //OBJ are enabled
  if(self.configuration.isOBJEnabled && ios.readLCDControlFlag(.OBJ_ENABLE)) {
    //tiles to consider for line
    let tiles = self.listObjTilesByDrawingOrder(line: ly, withHeight: th)
    //pre-fetch obj palettes
    let obp0 = ios.LCD_OBP0
    let obp1 = ios.LCD_OBP1

    //for each tile (horizontal priority is handled by previous ordering)
    for tile in tiles {
      //obj palette considering obj palette flag
      let objPalette = ColorPalette(paletteData: tile.useObjPalette1 ? obp1 : obp0,
                                    reference: self.configuration.palette)
      //tile line considering y flip flag
      let tileLine = tile.isYFlipped ? th - ly - tile.viewportYPos- 1
                                     : ly - tile.viewportYPos

      //tile address is linear following tile index
      let tileAddress:Short = self.getAddressAt(index: tile.Index * GBConstants.TileLength)

      //loop through tile bits to draw
      for curBit in 0...(nbBitsToDraw-1)
      {
         //destx is absolute to tile pos
         let destx:Byte = effectivex + curBit
         let tileBit = tile.isXFlipped ? curBit : tw - curBit - 1
         let (colorIndex, effectiveColor) = self.decodeColor(palette: objPalette,
                                                             b1: self.mmu[lineAddr],
                                                             b2: self.mmu[lineAddr+1],
                                                             at: tileBit)

          //color 0 for a tile means transparent (do not draw)
          if(colorIndex != 0){
              //draw pixel
              self.drawPixelAt(x: destx, y: desty, withColor: effectiveColor)
          }
      }
    }
  }
}
```

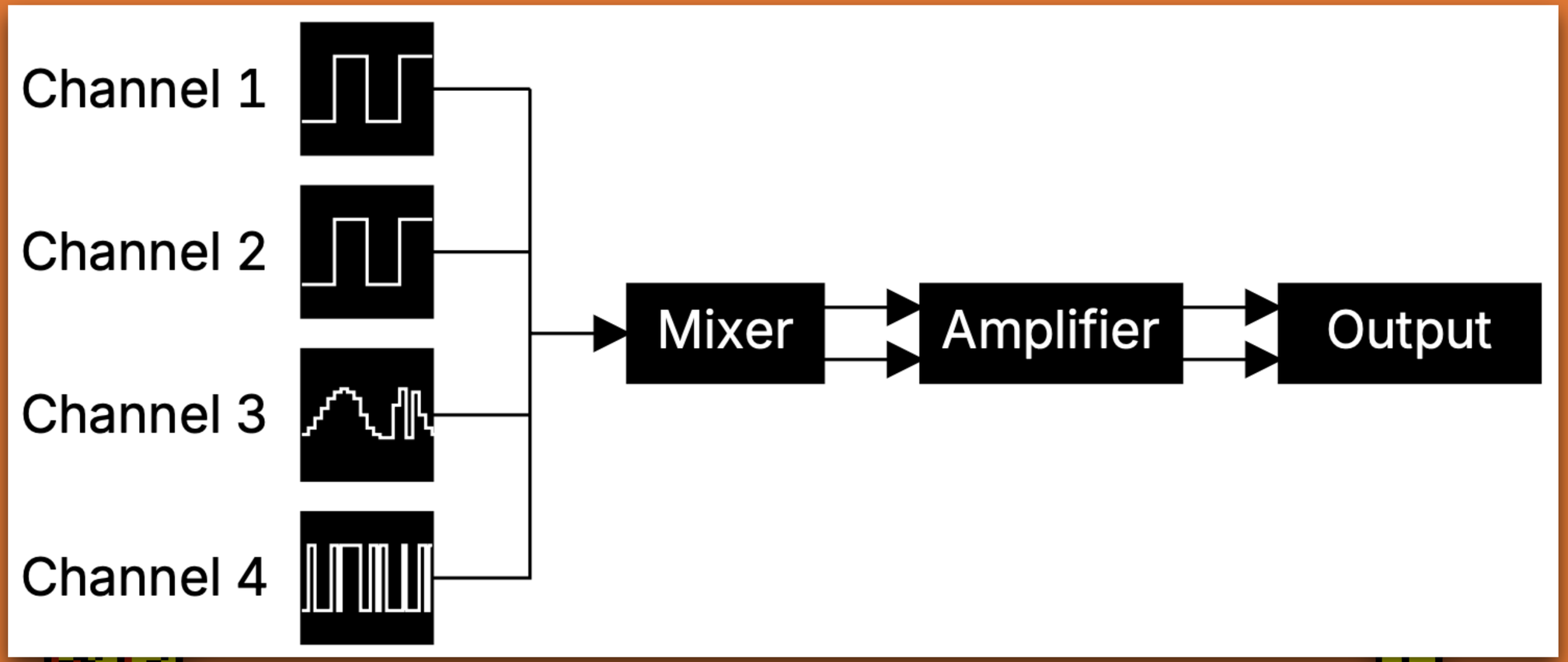
FPS: 17.65


FPS: 15.45
TM AND ©1987 ELORG.
TETRIS LICENSED TO
BULLET-PROOF
SOFTWARE AND
SUB-LICENSED TO
NINTENDO.
©1989 BULLET-PROOF
SOFTWARE.
©1989 Nintendo
ALL RIGHTS RESERVED.
ORIGINAL CONCEPT,
DESIGN AND PROGRAM
BY ALEXEY PAZHITNOV.


FPS: 60.09
"TM AND ©1937 ELORG.
TETRIS LICENSED TO
BULLET-PROOF
SOFTWARE AND
SUB-LICENSED TO
NINTENDO.
©1989 BULLET-PROOF
SOFTWARE.
©1989 Nintendo
ALL RIGHTS RESERVED.
ORIGINAL CONCEPT,
DESIGN AND PROGRAM
BY ALEXEY PAZHITNOV."

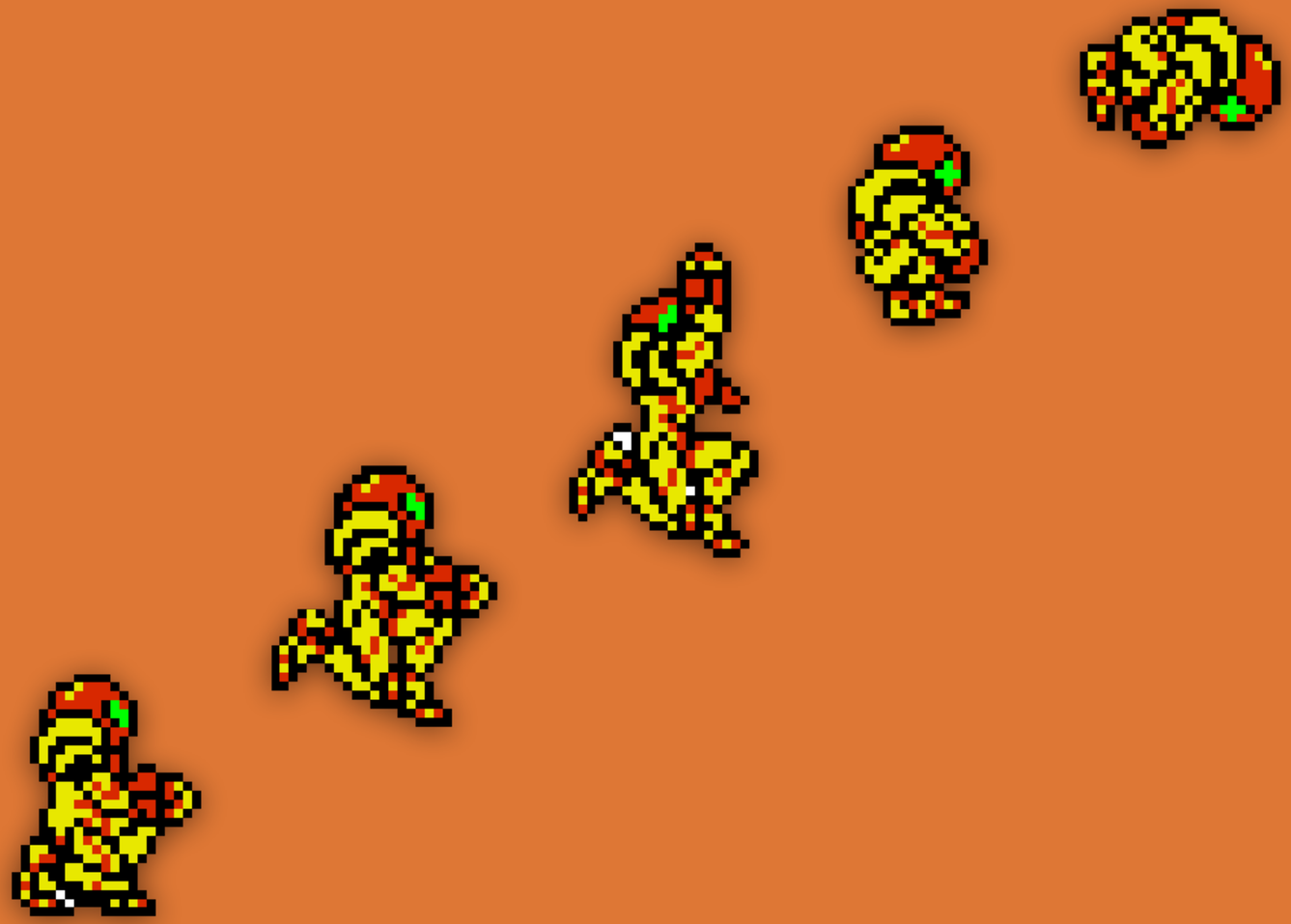
Concept	Vidéo 📽️	Audio 🔊
Unité	Frame	
Composé de	Pixels	
Composantes	RGB / RGBA	
Exprimé en	Frame rate	
Exemple	60-30 FPS	

Concept	Vidéo 📽️	Audio 🔊
Unité	Frame	Buffer
Composé de	Pixels	Samples
Composantes	RGB / RGBA	L, R Channels
Exprimé en	Frame rate	Sample rate
Exemple	60-30 FPS	44.1-48 kHz



Source : https://gbdev.io/pandocs/Audio.html

Exemple d'usage avec Tetris



Channels.swift

```swift
public class Noise: AudioChannelWithEnvelope, NoiseChannel {
    ...
    public func tick(_ masterCycles: Int, _ frameCycles: Int) {
        if(self.frequencyTimer == 0){
            //reload noise timer
            self.frequencyTimer = self.clockDivisor << self.clockShift
            //compute LFSR bit to apply (Not XOR between bit 0 and 1)
            let xor:Short = ~(((self.LFSR & 0b10) >> 1) ^ (self.LFSR & 0b01))
            //store xor at corresponding bit according to noise width
            if(self.hasNoiseShortWidth){
                self.LFSR = clear(.Bit_7, self.LFSR)
                self.LFSR |= xor << 7
            }
            else {
                self.LFSR = clear(.Bit_15, self.LFSR)
                self.LFSR |= xor << 15
            }
            //shift LFSR by 1
            self.LFSR >>= 1
        }
        else{
            self.frequencyTimer -= 1
        }
    }
    ...
    /// returns channel amplitude according to current wave pattern
    public override var amplitude:Byte {
        get {
            if(self.enabled){
                //amplitude is equal to LFSR bit 0 value. 0 no volume, 1 volume
                return Byte(self.LFSR & 0b0000_0000_0000_0001) * self.volume
            }
            else {
                return 0
            }
        }
    }
}
```

APU.swift

```swift
public class APU: Component, Clockable, APUProxy {
  /// return L and R sample by mixing each channel amplitude
  func sample() -> AudioSample {
    let panning = self.mmu.getAPUChannelPanning()
    let volume  = self.mmu.getMasterVolume()

    //sample to build
    var leftSample:Float  = 0;
    var rightSample:Float = 0;

    //apply panning

    //CH1
    if(self.configuration.isChannel1Enabled){
        if(panning.CH1_L){
            leftSample += self.channel1.analogAmplitude
        }
        if(panning.CH1_R){
            rightSample += self.channel1.analogAmplitude
        }
    }
```

```swift
    //return sample by applying master volume
    // divide each sample by 4, as we have summed for 4 channels
    // divide volume by 7, as volume is stored on 3 bits
    return (L: ((leftSample  / 4.0) * Float(volume.L)) / 7.0,
            R: ((rightSample / 4.0) * Float(volume.R)) / 7.0)
    }
}
```

TETRIS™
1PLAYER
2PLAYER
©1989 Nintendo

$$\frac{4\,048\,576 \text{ Hz} / 60 \text{ images}}{4 \text{ ticks minimum par action}} = \mathbf{17\,476 \text{ M ticks / image}}$$

Motherboard.swift

```swift
public class Motherboard: Clockable {
    ...
    public func insert(cartridge:Cartridge)

    public func powerOn()

    public func powerOff()

    public func tick(_ masterCycles:Int, _ frameCycles:Int) {
        self.cycles = self.cycles &+ GBConstants.MCycleLength
    }

    public func update() {
        if(self.isOn && self.hasCartridgeInserted) {
            while(tmpCycles < GBConstants.MCyclesPerFrame){
                self.cpu.tryHandleInterrupts()
                self.cpu.tick(self.cycles, tmpCycles)
                self.timer.tick(self.cycles, tmpCycles)
                self.mmu.tick(self.cycles, tmpCycles)
                self.ppu.tick(self.cycles, tmpCycles)
                self.apu.tick(self.cycles, tmpCycles)
                self.tick(self.cycles, tmpCycles)
                tmpCycles += GBConstants.MCycleLength
            }
        }
    }
}
```

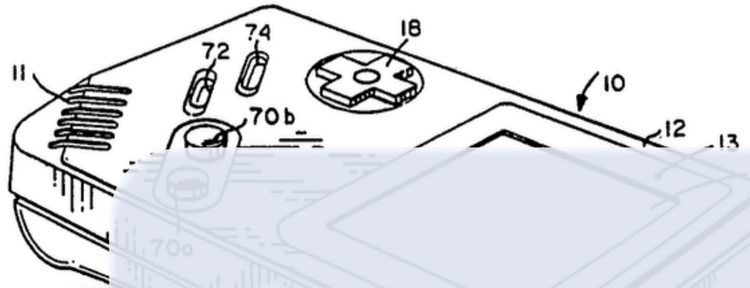
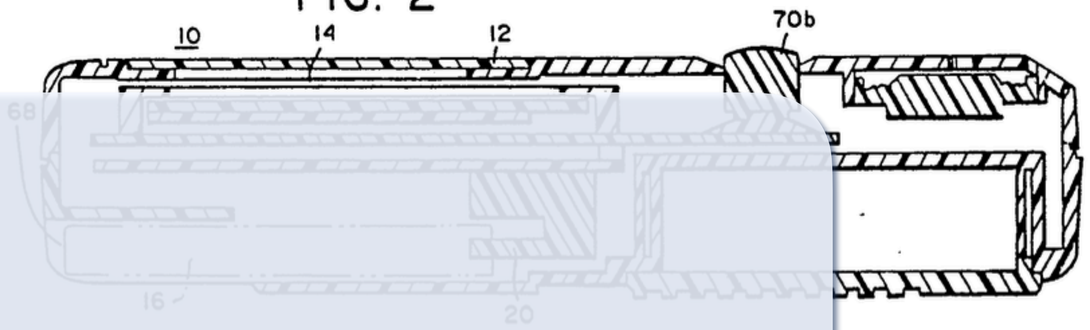


FIG. 2



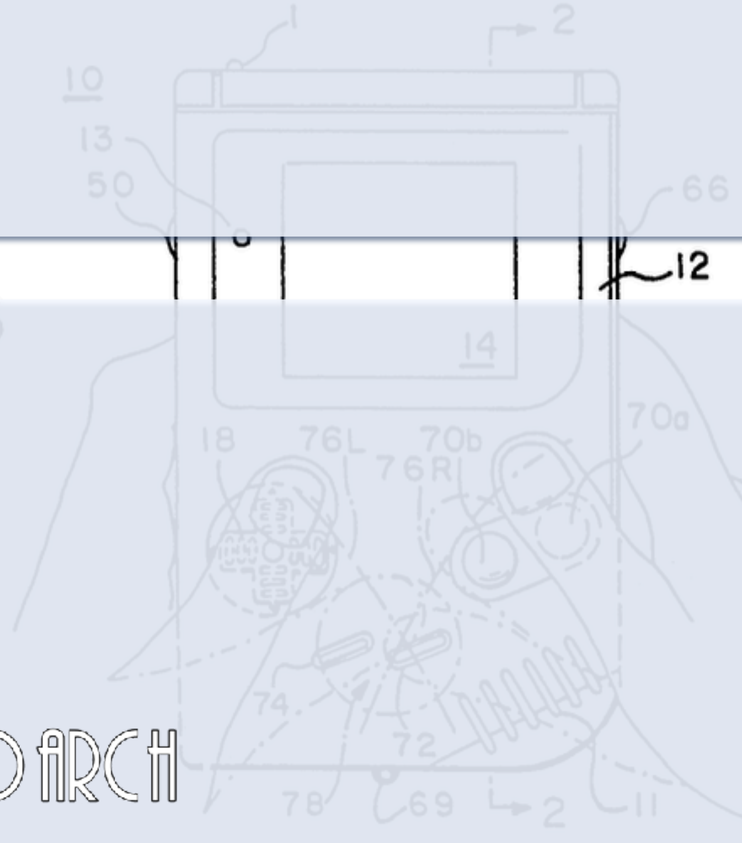
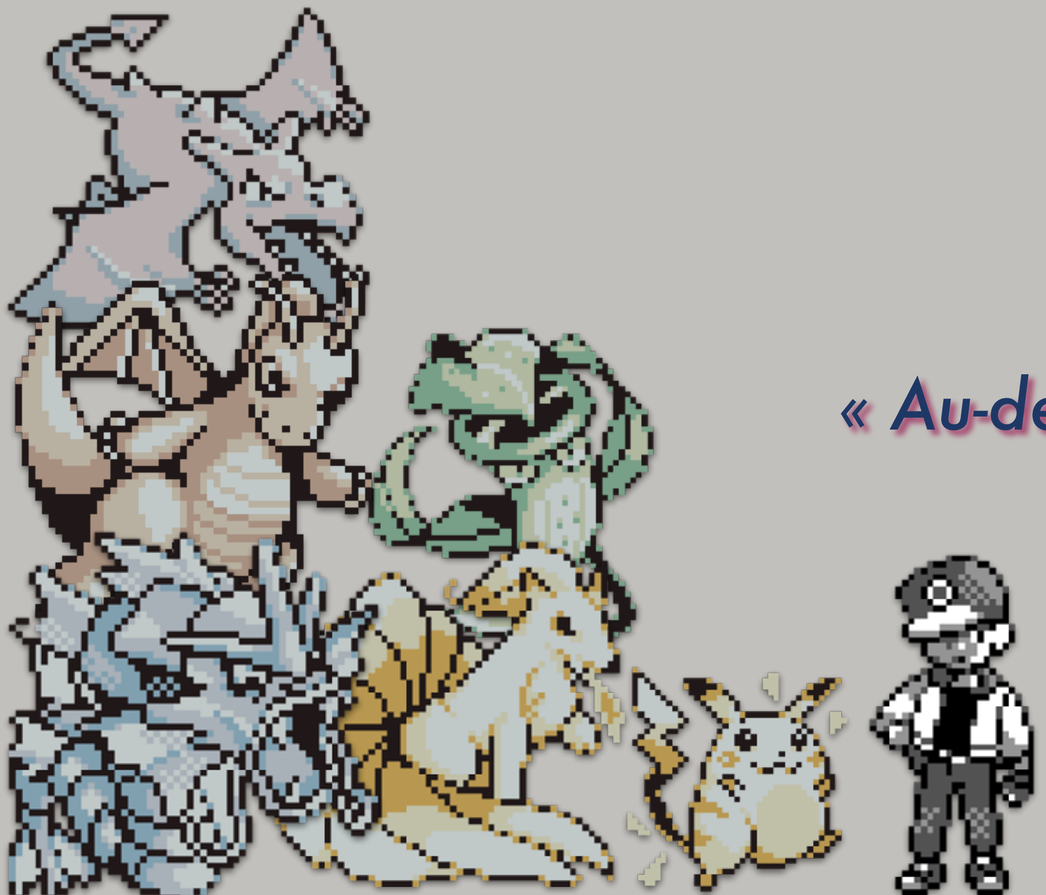
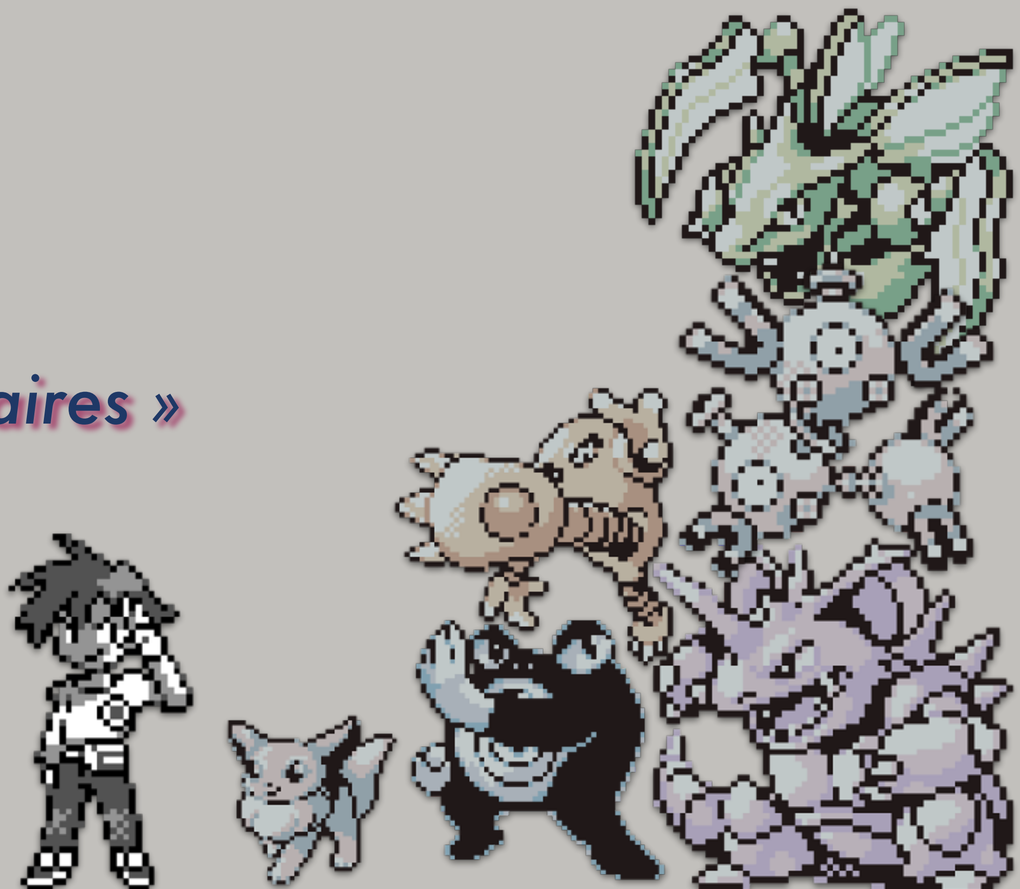


FIG. 3

RETRO ARCH



« Au-delà des tests unitaires »



SAMEBOY

A Friendly and Powerful Game Boy Emulator

Home Features Downloads Changes Bug Tracker GitHub Sponsor

About SameBoy

SameBoy is a user friendly, powerful and open source Game Boy, Game Boy Color and Super Game Boy emulator for macOS, iOS, watchOS, Windows and Unix-like platforms. SameBoy is extremely accurate and includes a wide range of both powerful debugging features and user-facing features, making it ideal for both casual players and developers. Of course, SameBoy also has every feature one would expect from an emulator – from save states to scaling filters.

Download SameBoy 1.0.2 for macOS

SameBoy for iOS is available on the App Store

Pokémon Gold

Game Boy Color

« It's a long way to the top... »

À titre d'exemple	GBKit	
Statut	Fonctionnel	
CPU	~1000 SLOC	
PPU	~600 SLOC	
APU	~1000 SLOC	

« It's a long way to the top... »

À titre d'exemple	GBKit	SameBoy
Statut	Fonctionnel	L'état de l'art
CPU	~1000 SLOC	~2000 SLOC
PPU	~600 SLOC	~2500 SLOC
APU	~1000 SLOC	~2100 SLOC

Gestion des cas particuliers

WIN
BG
SCX, SCY
Viewport
Tile map
Enable
Highlight
FF40 — LCDC: LCD control
LCDC is the main LCD Control register. Its bits toggle what elements are displayed on the screen and how.
Bit | Name | Usage notes
7 | LCD and PPU enable | 0=Off, 1=On
6 | Window tile map area | 0=9800-9BFF, 1=9C00-9FFF
5 | Window enable | 0=Off, 1=On
4 | BG and Window tile data area | 0=8800-97FF, 1=8000-8FFF
3 | BG tile map area | 0=9800-9BFF, 1=9C00-9FFF
2 | OBJ size | 0=8×8, 1=8×16
Corresponding Tile IDs
Block | VRAM Address | Objects | BG/Win if LCDC.4=1 | BG/Win if LCDC.4=0
0 | $8000–$87FF | 0–127 | 0–127
1 | $8800–$8FFF | 128–255 | 128–255 | 128–255 (or -128–-1)
2 | $9000–$97FF | (Can't use) | 0–127
Tiles are always indexed using an 8-bit integer, but the addressing method may differ. The "$8000 method" uses $8000 as its base pointer and uses an unsigned addressing, meaning that tiles 0-127 are in block 0, and tiles 128-255 are in block 1. The "$8800 method" uses $9000 as its base pointer and uses a signed addressing, meaning that tiles 0-127 are in block 2, and tiles -128 to -1 are in block 1, or to put it differently, "$8800 addressing" takes tiles 0-127 from block 2 and tiles 128-255 from block 1. (You can notice that block 1 is shared by both addressing methods)
Sample tile data
Et il y en a plein des cas particuliers
OAM DMA Transfer
FF46 — DMA: OAM DMA source address & start
Object Attribute Memory (OAM)
The Game Boy PPU can display up to 40 movable objects (or sprites), each 8×8 or 8×16 pixels.
Because of a limitation of hardware, only ten objects can be displayed per scanline. Object tiles have the same format as BG tiles, but they are taken from tile blocks 0 and 1 located at $8000-8FFF and have unsigned numbering.
Object attributes reside in the object attribute memory (OAM) at $FE00-FE9F. (This corresponds to the sprite attribute table on a TMS9918 VDP.) Each of the 40 entries consists of four bytes with the following meanings:
OBJ

« *Et les émulateurs officiels ?* »

Super GAME BOY

SUPER NINTENDO ENTERTAINMENT SYSTEM

GAME BOY PLAYER

NINTENDO GAMECUBE™

GAME BOY ADVANCE

README

tetris

Disassembly of the original Tetris for Game Boy. See tetris.asm.

It doesn't build yet -- the GBDis assembly syntax is not quite the same as RGBDS's, which is the standard today (used in pokered, for instance).

You might be able to build it with some ancient 90s assembler on Jeff Frohwein's site or something, but so far I've only used it to see what's going on in Tetris, not to mod it, so I haven't tried.

Origin

This disassembly is basically Jeff Frohwein's; all I've done is regenerate it and put it online with instructions.

`res/TETRIS.TAB` is Jeff Frohwein's original table from June 8, 1996. I used it to generate the initial disassembly from the Tetris ROM with MD5 `982ed5d2b12a0377eb14bcdc4123744e`.

Generating tetris.asm from the table file

Get the `tetris.gb` ROM somewhere and put it alongside `TETRIS.TAB` in the `res` folder.

Download GBDis V1.3 (gbdis13.zip) from Jeff Frohwein's site. It's a DOS binary, but it comes with its Turbo Pascal source code in a file called `DIS.PAS`. Here's the header at the top:

```
{   Nintendo Gameboy Disassembler V1.3  }
{             by                        }
{         Jeff Frohwein                 }
{   Written in Turbo Pascal V6.0        }
{   Started: 25 Nov 93                  }
{   Last edited: 9 Jun 96               }
```

README MIT license

mgbdis

A Game Boy ROM disassembler.

Features

- Generates assembly code compatible with RGBDS
- Supports ROMs with multiple banks
- Supports .sym files to define labels, code, data, text and image blocks
- Outputs a makefile to rebuild the ROM
- Uses defines from hardware.inc for hardware registers (source)
- Slow on large ROMs

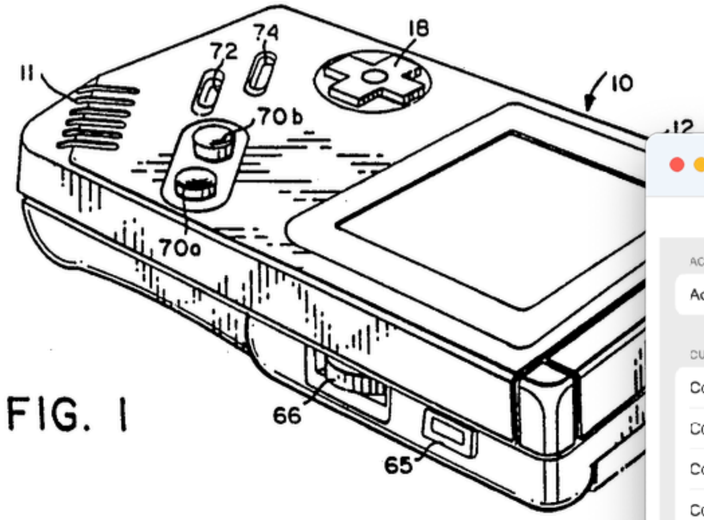

11
72
74
18
10
70b
70a
66
65

FIG. 1

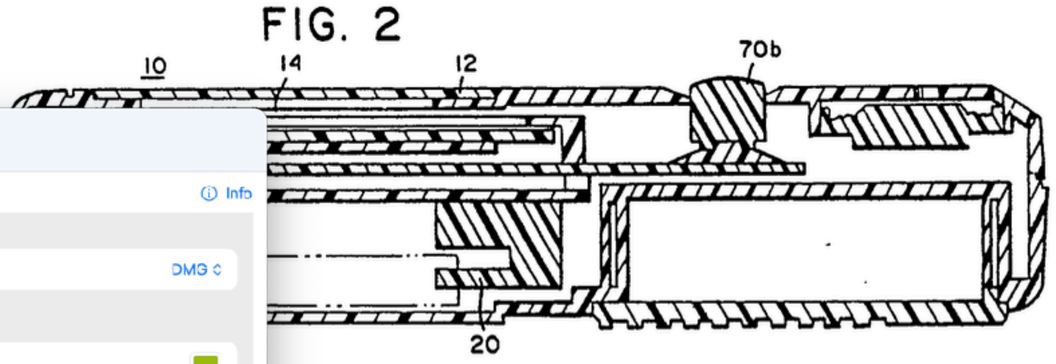
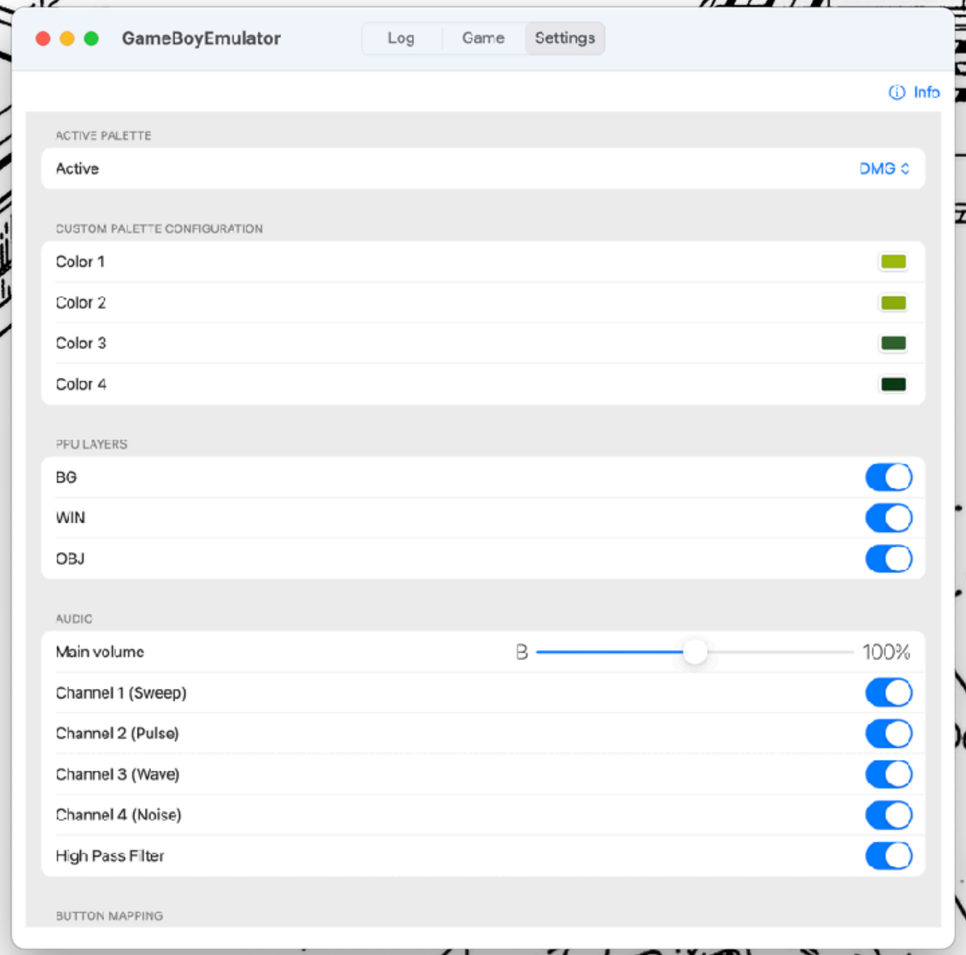
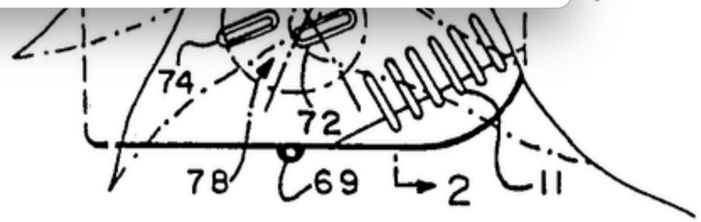
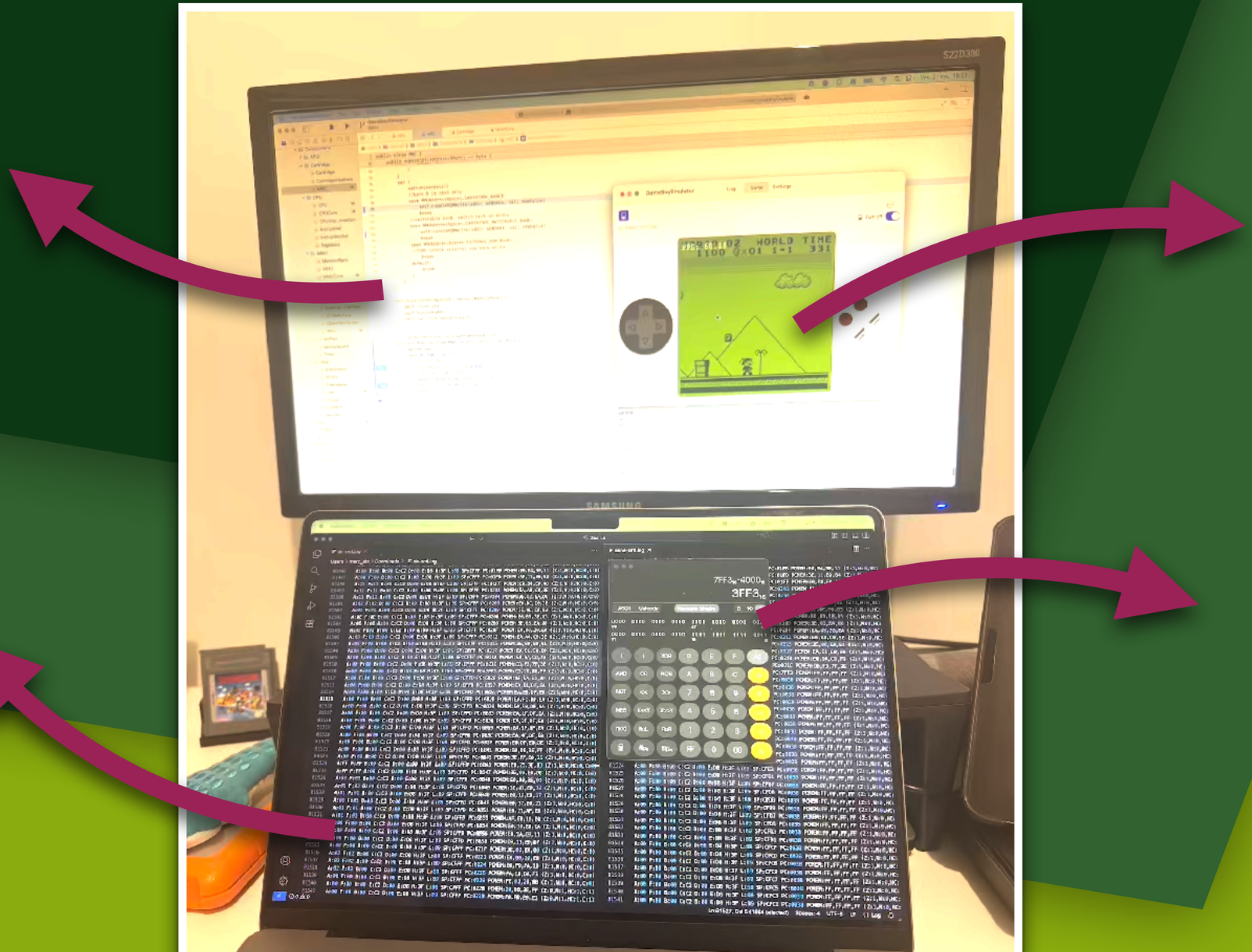

10
14
12
70b
20

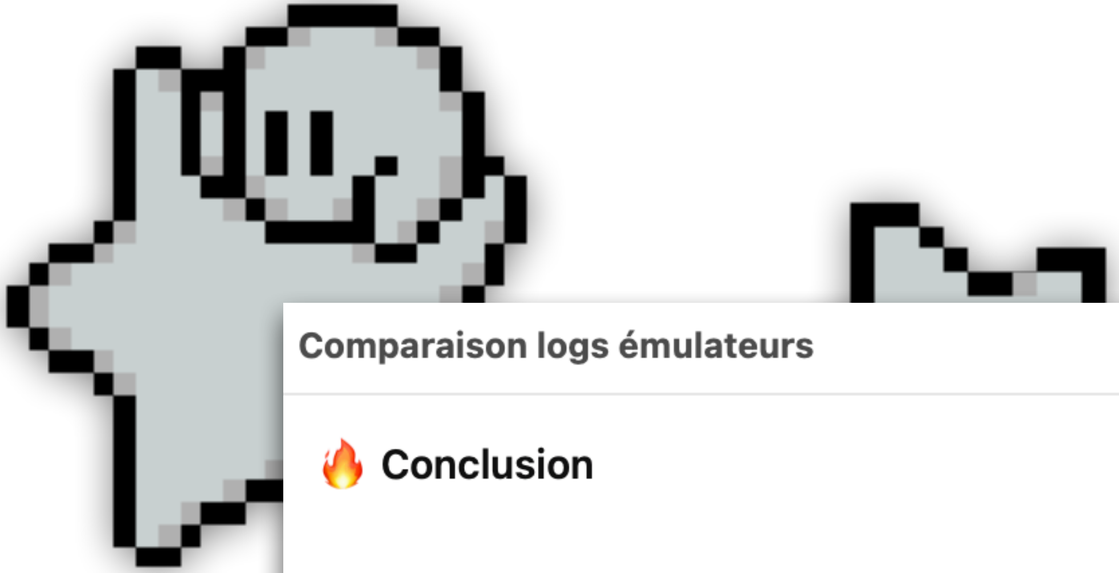
FIG. 2


GameBoyEmulator
Log
Game
Settings
Info
ACTIVE PALETTE
Active
DMG
CUSTOM PALETTE CONFIGURATION
Color 1
Color 2
Color 3
Color 4
PPU LAYERS
BG
WIN
OBJ
AUDIO
Main volume
100%
Channel 1 (Sweep)
Channel 2 (Pulse)
Channel 3 (Wave)
Channel 4 (Noise)
High Pass Filter
BUTTON MAPPING


74
72
78
69
2
11

GOOD LUCK


S22D300
SAMSUNG
WORLD TIME



Comparaison logs émulateurs

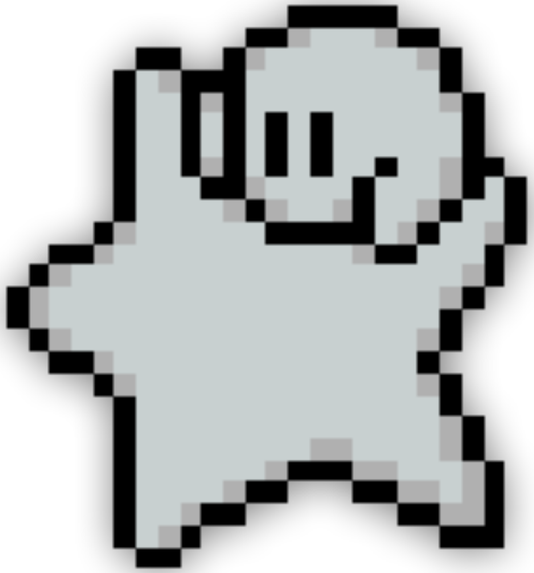
🔥 Conclusion

Ton SP va bien.
C'est ton CPU qui panique parce qu'il exécute du 0xFF à cause du MBC mal implémenté → RST 38 en boucle → PILE QUI SE DÉVERSE.

Fixe ton banking et tu vas retrouver les mêmes logs que l'émulateur "fonctionnel".

Si tu veux, envoie ton code MBC ou ta fonction `read(addr)` / `write(addr)` pour qu'on l'examine.

Exemple ChatGPT - Modèle 5



r/PSP

The best way to play Pokémon

42 upvotes · 14 commentaires

-_-Redd-_- · 7 h

My main problem with GBA emulation on PSP is the audio, its so choppy. Thankfully, I have a 3DS for my Nintendo handheld needs

Répondre 8

stmb_arisu AO · 7 h
PSP-3000

Personally my audio has been fine but maybe I'm oblivious to it.

2

-_-Redd-_- · 7 h

Its night and day bro when you know how the audio is supposed to sound like

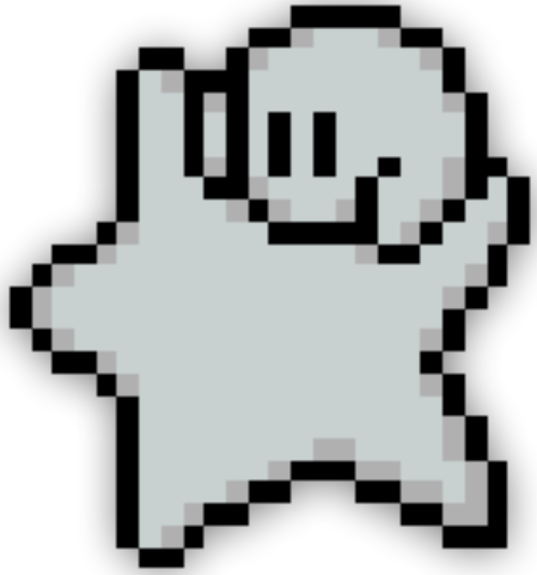
3

stmb_arisu AO · 7 h
PSP-3000

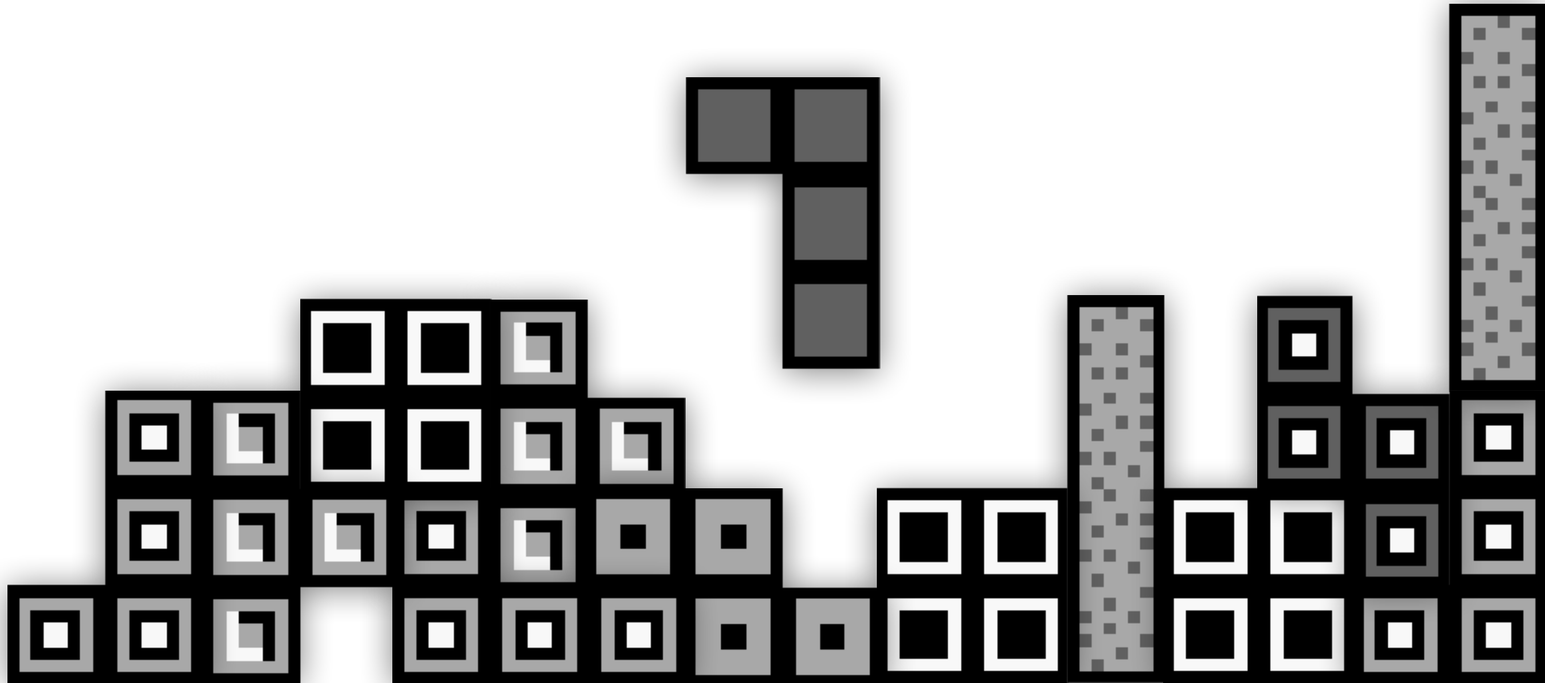
I guess I just don't have an ear for it.

1

reddit



« Au-delà du Game Dev »



README Apache-2.0 license

GBKit

Swift package that provides emulation back-end to make your own Game Boy emulator.

Here's a simple emulator (front-end) usage : https://github.com/MarcAlx/gb

Status Games that don't rely on advance CPU tricks or MBC special feature works, e.g Super Mario Land, Kirby Dream Land, Tetris, Tennis, F1 Race, Pokémon...

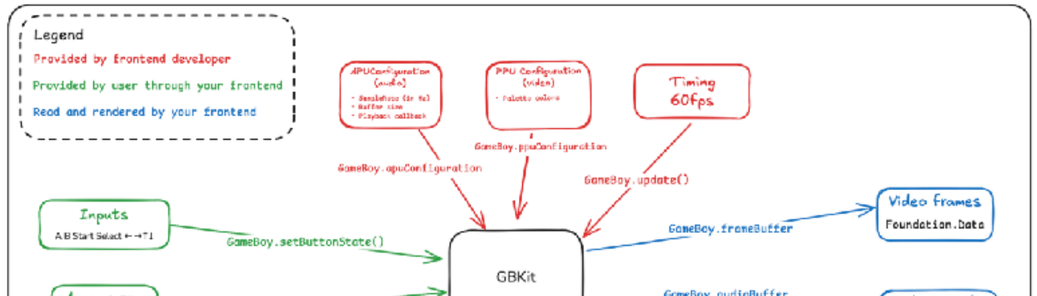
Under active development (full MBC-2-3-5-6-7 support on the way! now limited to RAM feature, ETA: when its done!). For now limited to GB, GBC support planned.

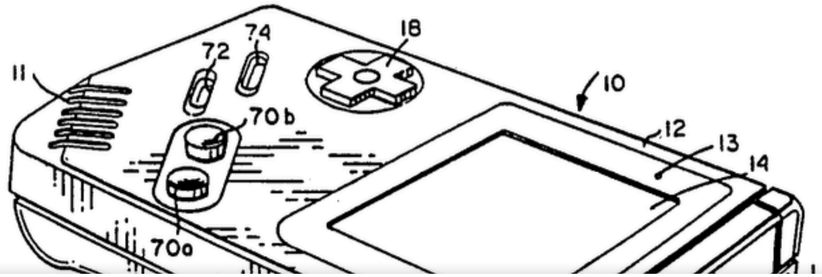
Miletones

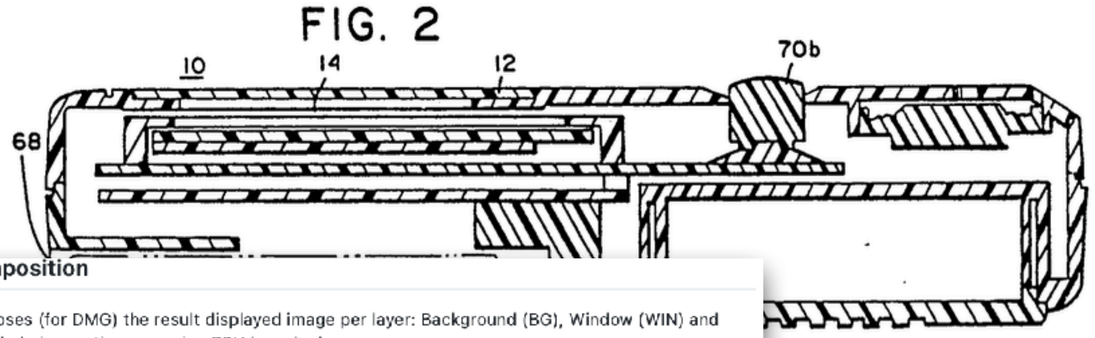
- CPU (Done)
- PPU (Done)
- APU (Done, not all obscure behaviors implemented)
- MBC handling (No MBC and MBC1: OK, MBC2,MBC3, MBC5 ROM/RAM Ok, full MBC-2-3-5-6-7 support in development, see status table below)
- Finish unit tests (TODO)
- Move from `XCTest` to new Swift `Testing` (Next)
- GBC Support (Later)

▶ MBC support status

Architecture







Improve README.md #6

Edit | Code

Open | MarcAlx wants to merge 5 commits into `mattcurrie:master` from `MarcAlx:dev/doc`

Conversation 1 | Commits 5 | Checks 0 | Files changed 8 | +39 −0

MarcAlx commented on May 23, 2024

Hi Matt,

Thank you for your wonderful tool that's an unmissable checkpoint in writing a DMG/GBC emulator.

I've made this PR to add a new failure example that I've found easy to miss (misaligned footer text caused by SCX not being handled).

Then I complete the README.md by decomposing per layer the final image.

Screenshots have been made using my own emulator.

Feel free to give me some feedback.

PS As I'm not working on a GBC emulator yet, I may not be able to produce the same result for GBC.

Marc_Alx and others added 5 commits last year

- [doc] add a new failure exemple that illustrates the SCX register not… — 2105004
- [doc] Improve readme by decomposing result image per layer (BG/WIN/OBJ). — 3acac88
- Update README.md — Verified — a2e3a43
- [doc] replace asset (not well cropped) — a9f0940
- [doc] fix typo in sub section title — 22dcca7

mattcurrie commented on May 27, 2024 — Owner

Thanks for this! These look to be some good improvements to the documentation.

I'll try to have a good look through it this week.

Reviewers: No reviews
Still in progress? Convert to draft

Assignees: No one assigned

Labels: None yet

Projects: None yet

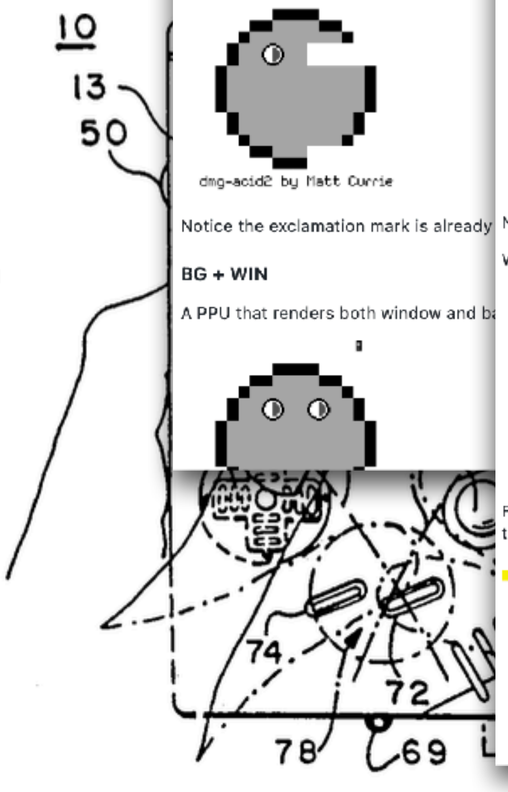
Milestone: No milestone

Development: Successfully merging this pull request may close these issues. None yet

Notifications — Customize
Unsubscribe
You're receiving notifications because you authored the thread.

2 participants

Allow edits by maintainers



Per layer decomposition

This section decomposes (for DMG) the result displayed image per layer: Background (BG), Window (WIN) and Tiles (OBJ), this may help in creating a passing PPU layer by layer.

BG

A ppu that renders only the backgound

dmg-acid2 by Matt Currie

Notice the exclamation mark is already

BG + WIN

A PPU that renders both window and b

BG + WIN + OBJ

The following image highlights in red the OBJ part of the final targeted image:

dmg-acid2 by Matt Currie

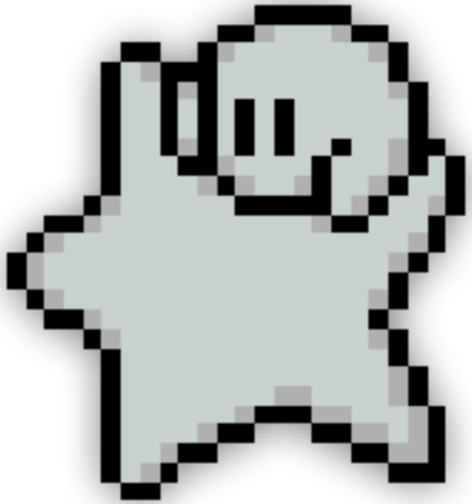
Notice that obj tiles are transparent and let below layers appear.

Wrapping it all, the following image highlights both WIN (in blue) and OBJ (in red) for the final image:

dmg-acid2 by Matt Currie

For context the following image highlights in yellow where PPU is disabled (LCDC bit 7 being reset) thus masking the hair:

dmg-acid2 by Matt Currie



r/EmuDev · -3 m.
Marc_Alx

Audio emulation 101: Game Boy example

Disclaimer: this post is not a full breakdown, it's just a compilation of information I wish I had before starting. I will focus on Game Boy as it's the only system I emulate.

After achieving audio emulation on my Game Boy emulator I think it's a good time to summarize some key things I've understood in the process.

First, audio (as for the Game Boy) is way less documented and straightforward than CPU or Graphics (PPU for Game Boy). As always reading the doc and reference is key to understand what's happening.

1. Video vs. Audio

Usually when making an emulator everyone understands what video memory is and how an image should be produced from that, but for audio it's usually obscure.

Important notion: Back-end VS Front-end, back-end usually refers to your emulation logic where front-end refers to how your emulation is interacted with (both input and output). Do not merge these two notions, separate them in your code, *i.e do not put SDL code in your APU; make two modules.*

Let's make a comparison between audio and video, from a front-end perspective

Lire la suite

64 · 11

reddit

SNOWCAMP

jeu. 15 janv. 2026

10:10 › 10:55

FR Voyage au coeur de l'émulation : destination Game Boy

Hors-piste

Conférence (45m)

BEGINNER

Talk summary

Venez découvrir comment, au détour d'une vidéo YouTube sur l'émulation et d'un « mais ça, je peux le faire » , j'ai embarqué dans un voyage au croisement de l'informatique, de l'électronique, du gaming, de l'audio et même du droit.

On se dirige tout droit vers la nostalgie et pas n'importe laquelle : celle des gros pixels, des sons 8 bits, des cartouches de jeux, des piles AA chipées dans la télécommande du salon, des tétrominos, des 151 premiers Pokémon : de la Nintendo Game Boy !

Ce talk, c'est le récit d'un side project très nerdy : développer un émulateur Game Boy. Du premier pixel affiché aux premières notes entendues, vous découvrirez une multitude de défis techniques : compréhension du matériel, reverse engineering, audio/video processing, optimisation des performances, qualité logicielle et techniques de déboggage. Au-delà du fun et de la nostalgie, ce projet montre à quel point l'émulation est un formidable terrain d'apprentissage pour tout développeur.

Voici quelques-unes des questions auxquelles on va répondre :

- Par où commence-t-on un projet pareil ?
- Quelles sont les différentes techniques modernes d'émulation ?
- Comment architecturer un émulateur ?
- Ça prend combien de temps d'émuler une console ?
- Comment fait-on pour émuler le son ?
- Comment sait-on qu'un émulateur est fiable ?
- Comment débogguer un programme qui ne renvoie pas d'erreur ?
- Et surtout : c'est légal tout ça ?

GAME BOY COLOR

Vers un autre projet d'émulation ?

Pokémon mini

Playdate

CGI

open

KAIZEN SOLUTIONS

aws

CRITEO

MOODY'S

clever cloud

AVISTO

KLS GROUP Le French Logistique

kelkoogroup

zenika

VISEO POSITIVE DIGITAL MAKERS

alma

salesforce

sopra steria

PeriScop

enalean software engineering is now

HACK YOUR JOB

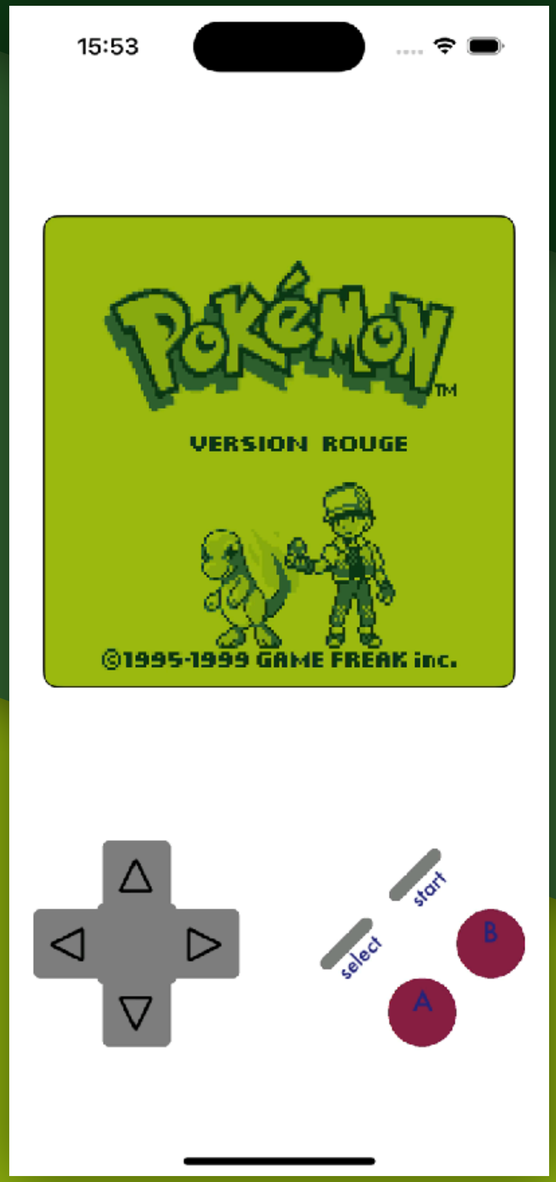
PingID

118 M Game Boy &
Game Boy Color
79 M Game Boy Advance
197 M total
1049 Game Boy Games
576 Game Boy Color Games
33c3
WORKS FOR ME

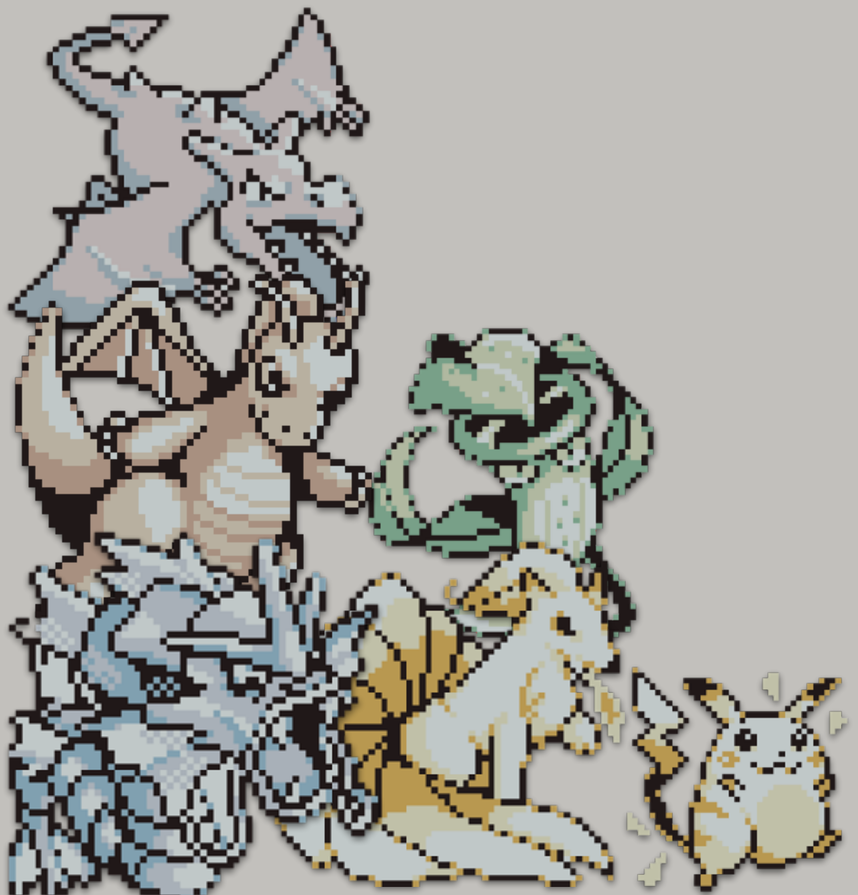
L'Histoire de
NINTENDO
Florent GORGES
Volume 4
L'incroyable histoire de la Game Boy
omake books

GDC
Selling
Classic
Games
00:57
[GDC]
"It's Just Emulation!": The Challenge of Selling Old Games
GDC Festival of Gaming
215.3 k vues • il y a 10 ans

reddit


15:53
Pokémon
VERSION ROUGE
©1995-1999 GAME FREAK inc.
start
select
B
A

« True last boss? »



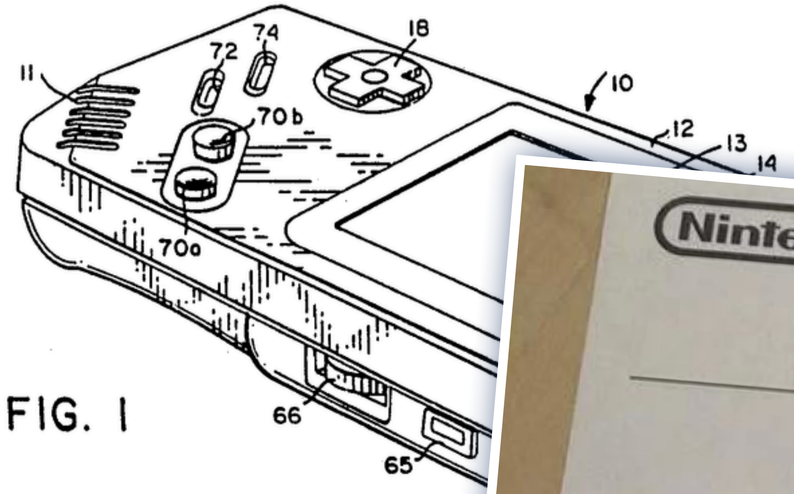


FIG. 1

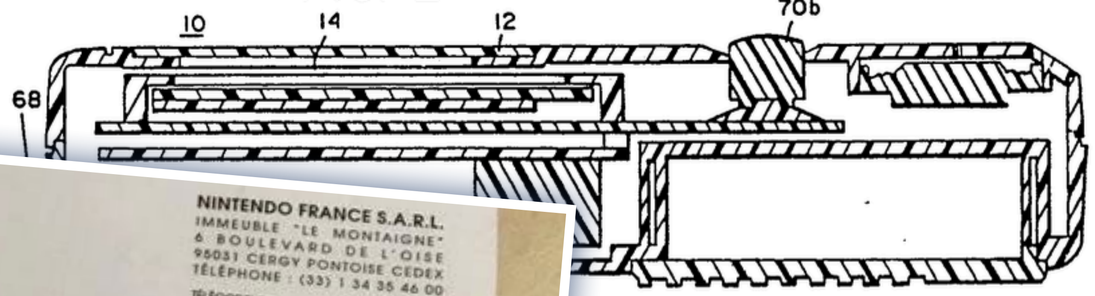


FIG. 2

Nintendo

NINTENDO FRANCE S.A.R.L.
IMMEUBLE "LE MONTAIGNE"
6 BOULEVARD DE L'OISE
95031 CERGY PONTOISE CEDEX
TÉLÉPHONE : (33) 1 34 35 46 00

TÉLÉCOPIEUR
Direction commerciale : (33) 1 34 35 46 36
Administration des ventes : (33) 1 34 24 90 85
Direction financière : (33) 1 34 35 46 35
Direction marketing : (33) 1 34 35 46 67

GUILLAUME GETE

[redacted]

Cergy, le 14 mars 2016

Bonjour,

Merci d'apprécier notre société et nos jeux vidéo.

Nous avons bien reçu votre courrier que nous avons lu avec le plus grand intérêt, et vous remercions de l'intérêt que vous portez à notre société.

Il faut dire : UN GAMEBOY
C'est masculin.

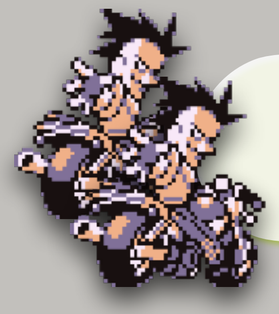
Mais ce n'est pas faux de dire la gameboy. Qui veut dire « la console gameboy »

Vous souhaitant d'agréables parties de jeu Nintendo,

Cordialement.

NINTENDO FRANCE

Se lancer dans un projet d'émulation



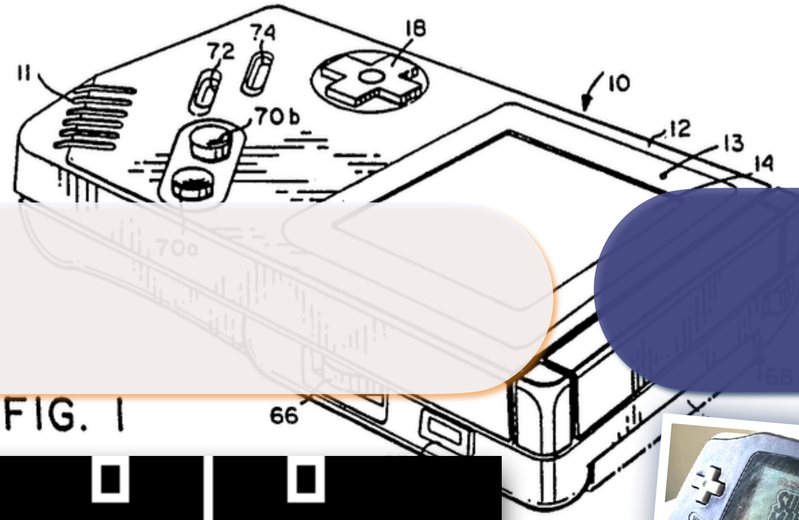


FIG. 1

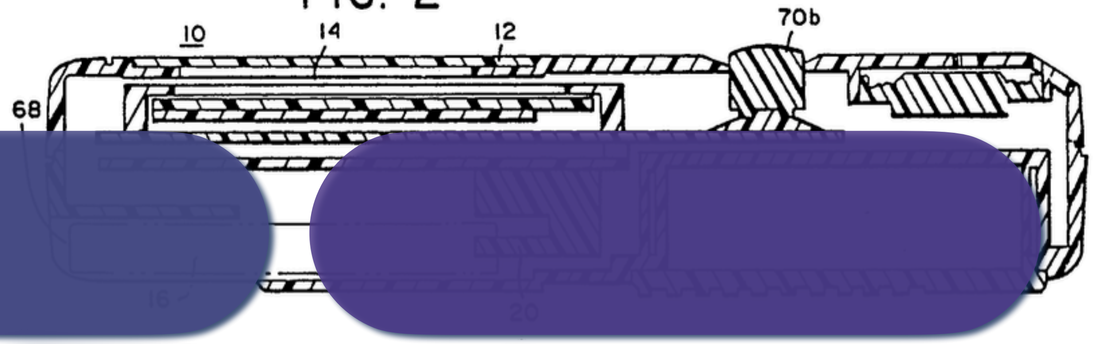


FIG. 2

FIG. 3

GAME BOY pocket

Nintendo
SUPER MARIO 3
POWER
GAME BOY ADVANCE

GAME BOY ADVANCE SP

SUPER MARIO 3

GAMEBOY

Hello!
ミシンではじめてのカタチ。
グラマーでかわいい。
だけど実力はすごいゾ！
世界で初めて！
ゲームボーイ接続型の
新・コンピュータミシン
GAME BOY COLOR

PERIPHERAL VISION
INSIDE THE GAME BOY'S ACCESSORIES
Nintendo GAME BOY
Daniel S. Baxter

GAME BOY
game link
universal game link cable set

Nintendo
GAME BOY
GAME BOY printer
Nintendo
POWER
GAME BOY
printer
DER DRUCKER ZUR GAME BOY CAMERA!
L'IMPRIMANTE POUR VOTRE GAME BOY CAMERA!
DE PRINTER VOOR JE GAME BOY CAMERA!
¡LA IMPRESORA PARA TU GAME BOY CAMERA!
Nintendo

Vue d'ensemble des projets d'émulation (consoles)

Approche	Docs officielles / Retro-ingénierie / Communautaire			Retro-ingénierie Communautaire	
Durée à peu près™	1 à 2 Semaines	1 mois	3 à 4 mois	1 à 2 ans	??
Techniques	Classique / JIT / FPGA			JIT + Translation layer	
Technos	Tous les langages	Tous les langages	ASM / C++ / Rust		
Plateformes	Chip 8	NES Game Boy	SNES GBA PS1	PS2 Xbox GameCube	PS3 Xbox 360
		80's	90's	2000	2005 ➜ 2010

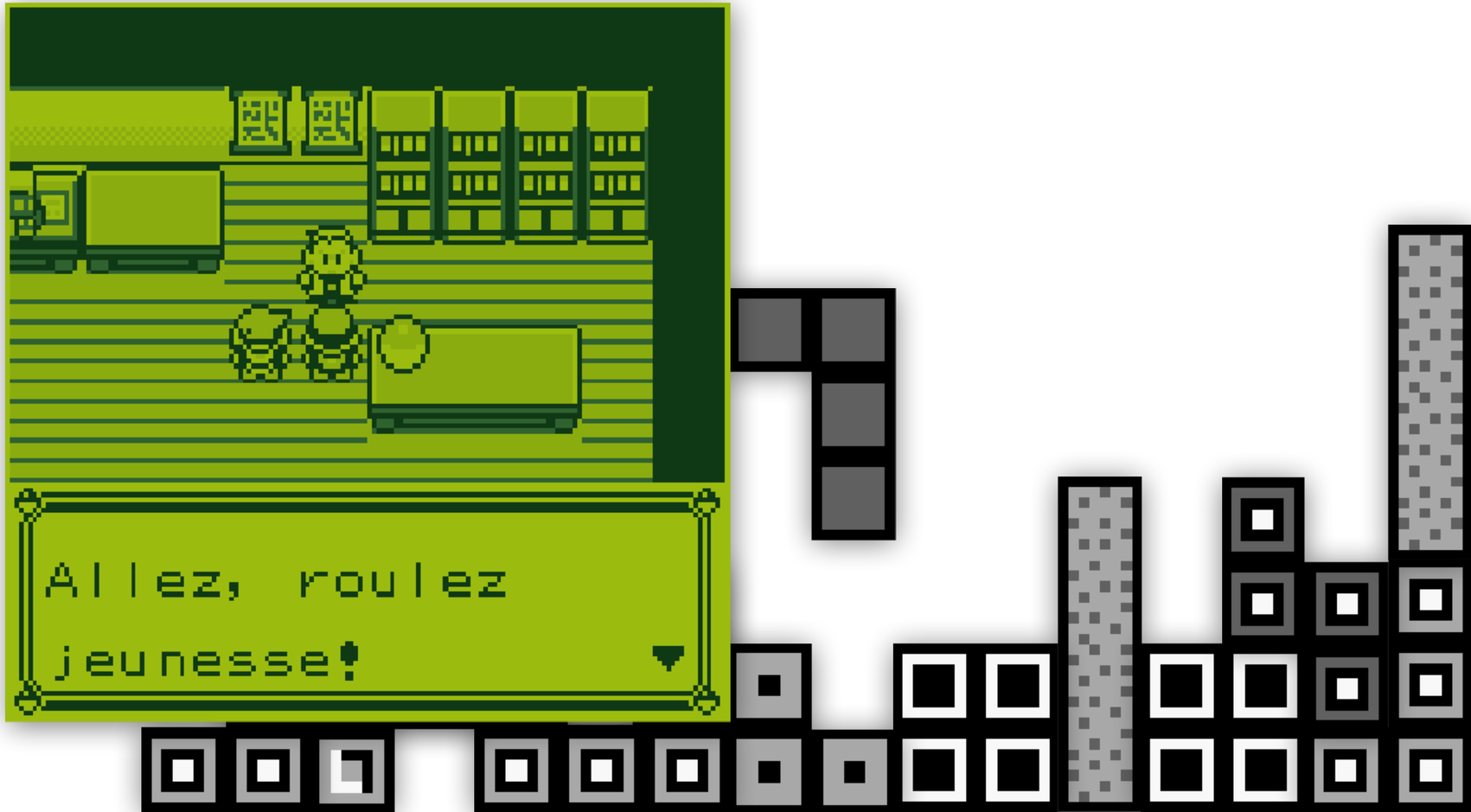
Console 8 bits : *GAME BOY* / Nintendo ENTERTAINMENT SYSTEM

Projet d'émulation à la portée de tous

Durée (à peu près™) : **1 à 2 mois**

Fortement documentée *pour son fonctionnement de base.*
Communauté très présente

Réalisable avec **n'importe quel langage** */ technique*


Allez, roulez
jeunesse!

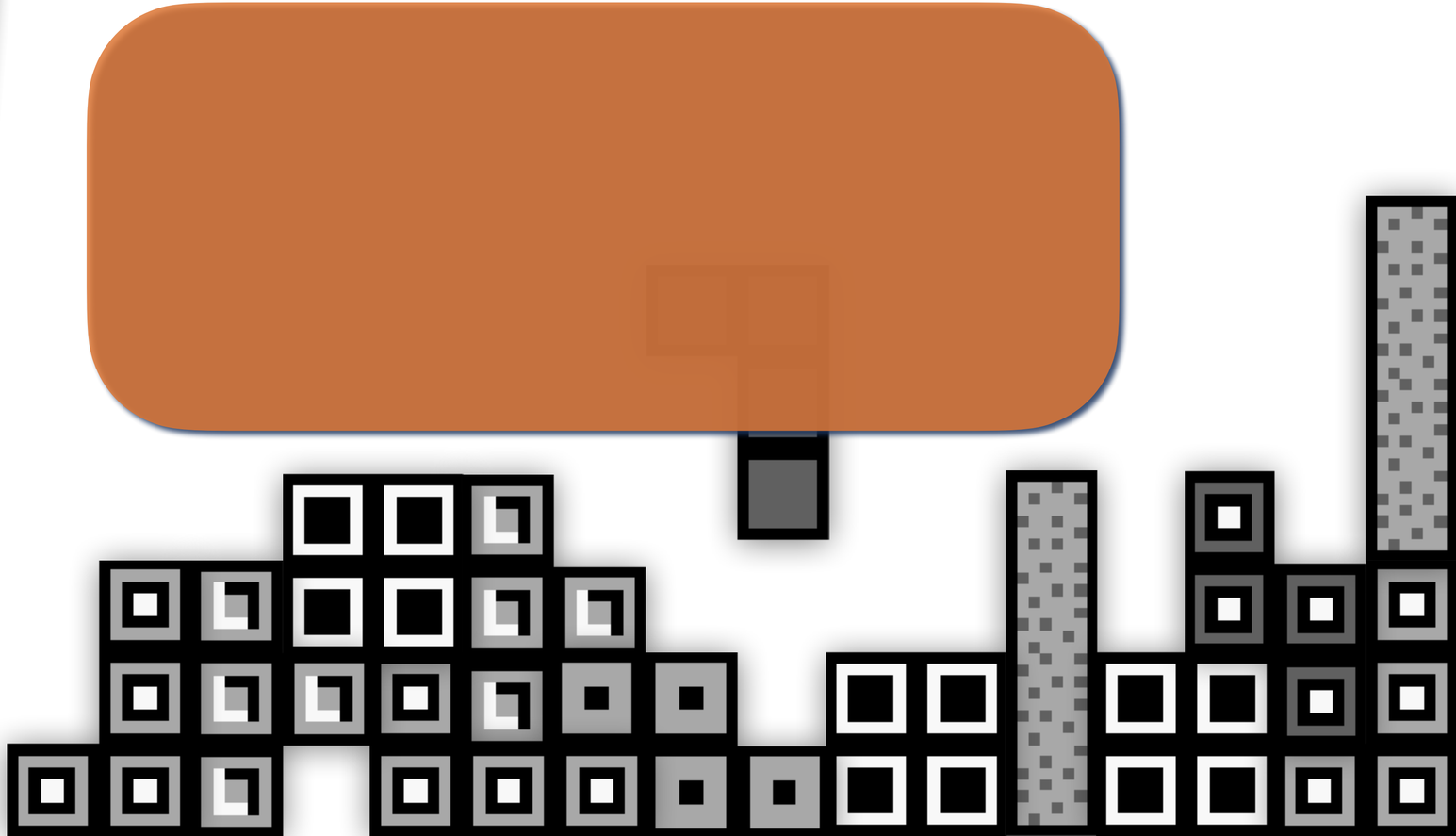
« Décoder le header des cartouches de jeu »

Chaque ROM dispose d'un header avec des informations sur le jeu.

Console

```
[INFO] top-level - Title: TETRIS
Manufacturer code:
Licensee: Nintendo
Destination: GBKit.DestinationCode.JAPAN_OVERSEAS
Version: 1

Cartridge type: GBKit.CartridgeType.ROM_ONLY
ROM: 32KiB (2 banks)
RAM: 0KiB (0 banks)
CGB support: unspecified

Nintendo logo in headers: true

Header checksum: 10 (computed: 10, equals: true)
Global checksum: 5823 (computed: 5823, equals: true)
```

Nintendo GAME BOY™
THIS SIDE OUT / CE CÔTÉ À L'EXTÉRIEUR



Start	End	Description
0000	3FFF	16 KiB ROM bank 00
4000	7FFF	16 KiB ROM Bank 01–NN
8000	9FFF	8 KiB Video RAM (VRAM)
A000	BFFF	8 KiB External RAM
C000	CFFF	4 KiB Work RAM (WRAM)
D000	DFFF	4 KiB Work RAM (WRAM)
E000	FDFF	Echo RAM (mirror of C000–DDFF)
FE00	FE9F	Object attribute memory (OAM)
FEA0	FEFF	Not Usable
FF00	FF7F	I/O Registers
FF80	FFFE	High RAM (HRAM)
FFFF	FFFF	Interrupt Enable register (IE)

Source : https://gbdev.io/pandocs/Memory_Map.html

MMU.swift

```swift
public class MMU:Component, Clockable {
  ...
    public subscript(address:Short) -> Byte {
        get {
          switch address {
            ...
            case MMUAddressSpaces.CARTRIDGE_ROM:
                return self.currentCartridge[address]
            case MMUAddressSpaces.EXTERNAL_RAM_BANK:
                return self.currentCartridge[address]
            //prohibited area, always return 0
            case MMUAddressSpaces.PROHIBITED_AREA:
                return 0x00
            //set ram value
            default:
                return self.ram[address]
            }
        }
        set {
            switch address {
            ...
            case MMUAddressSpaces.CARTRIDGE_ROM:
                self.currentCartridge[address] = newValue
            case MMUAddressSpaces.EXTERNAL_RAM_BANK:
                self.currentCartridge[address] = newValue
            //prohibited area cannot be set
            case MMUAddressSpaces.PROHIBITED_AREA:
                break
            //default to ram
            default:
                self.ram[address] = newValue
            }
        }
  ...
}
```

Les interruptions

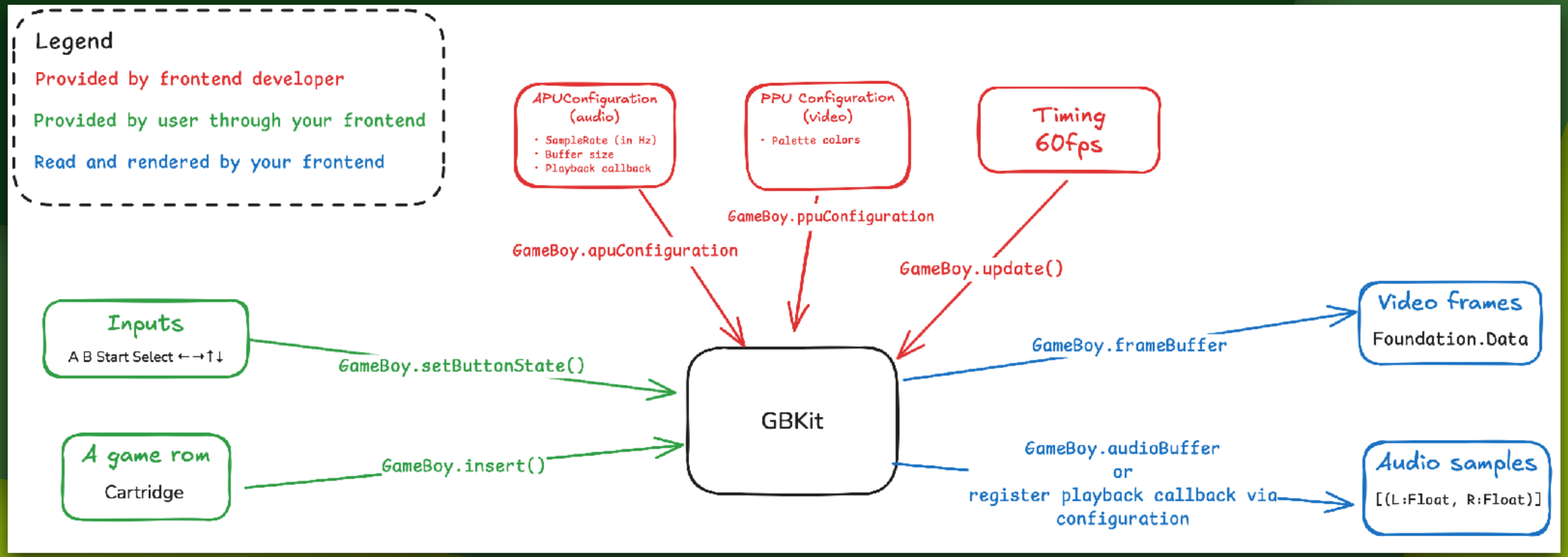
*Différents « **hook** » pour permettre l'interactivité d'un jeu, lors d'une interruption le jeu se met en pause execute le code à une adresse précise et reprend son cours.*

Exemple :

- *Appuis sur une touche (JOYPAD)*
- *Fin de dessin d'une image (VBLANK)*
- *Arrivée à échéance d'un timer (TIMER)*
- *Communication via le port série (SERIAL)*

Techniquement la console :

- *Vérifie que l'interruption est activée par le jeu*
- *Sauvegarde l'état actuel du programme (PC)*
- *« jump » à l'adresse associée à l'interruption*
- *Une fois exécutée, le jeu reprend son court*


Legend
Provided by frontend developer
Provided by user through your frontend
Read and rendered by your frontend
APUConfiguration (audio)
• SampleRate (in Hz)
• Buffer size
• Playback callback
PPU Configuration (video)
• Palette colors
Timing 60fps
GameBoy.ppuConfiguration
GameBoy.apuConfiguration
GameBoy.update()
Inputs
A B Start Select ←→↑↓
GameBoy.setButtonState()
GameBoy.frameBuffer
Video frames
Foundation.Data
GBKit
A game rom
Cartridge
GameBoy.insert()
GameBoy.audioBuffer
or
register playback callback via configuration
Audio samples
[(L:Float, R:Float)]

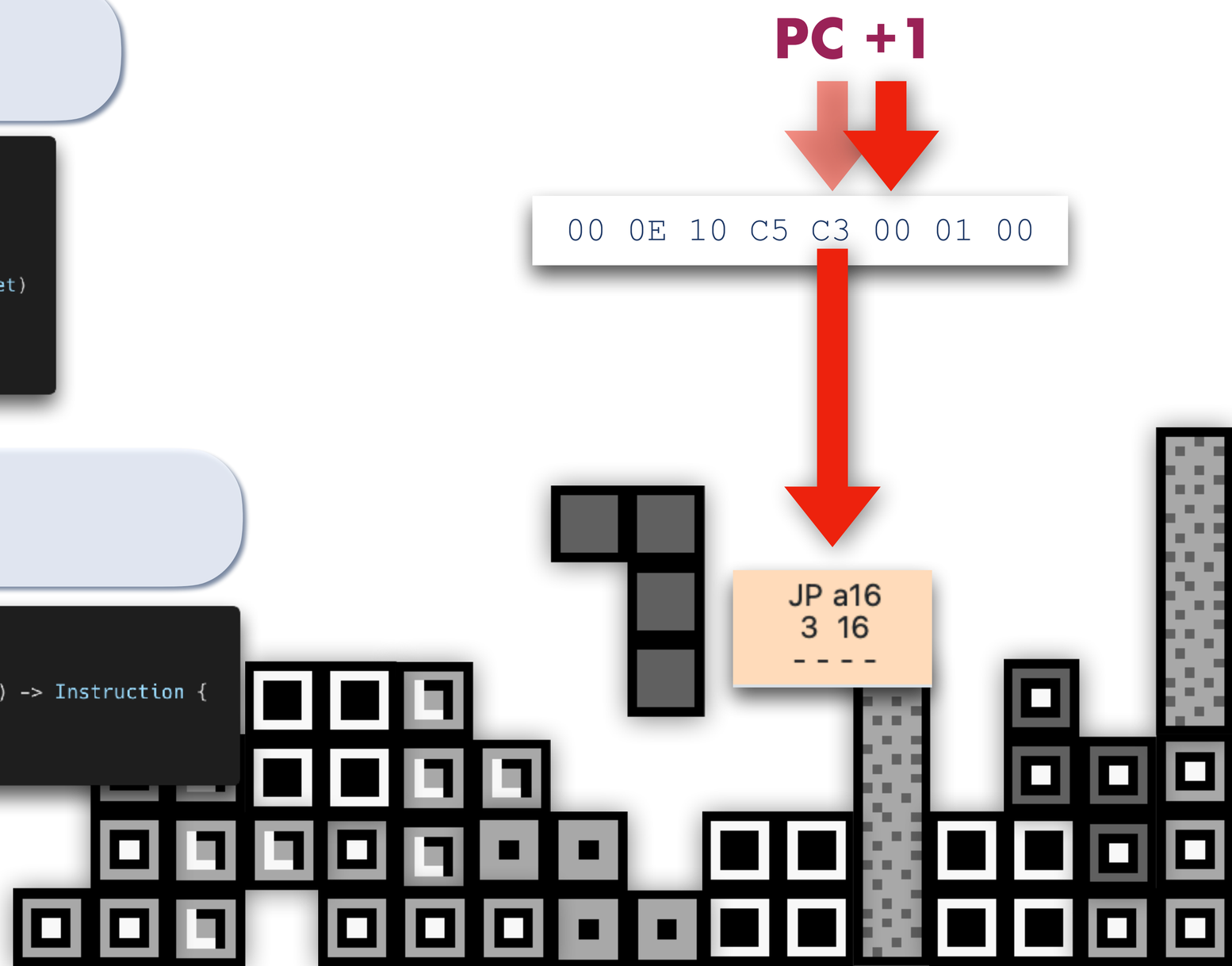
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	RLC B 2 8 Z 0 0 C	RLC C 2 8 Z 0 0 C	RLC D 2 8 Z 0 0 C	RLC E 2 8 Z 0 0 C	RLC H 2 8 Z 0 0 C	RLC L 2 8 Z 0 0 C	RLC [HL] 2 16 Z 0 0 C	RLC A 2 8 Z 0 0 C	RRC B 2 8 Z 0 0 C	RRC C 2 8 Z 0 0 C	RRC D 2 8 Z 0 0 C	RRC E 2 8 Z 0 0 C	RRC H 2 8 Z 0 0 C	RRC L 2 8 Z 0 0 C	RRC [HL] 2 16 Z 0 0 C	RRC A 2 8 Z 0 0 C
1x	RL B 2 8 Z 0 0 C	RL C 2 8 Z 0 0 C	RL D 2 8 Z 0 0 C	RL E 2 8 Z 0 0 C	RL H 2 8 Z 0 0 C	RL L 2 8 Z 0 0 C	RL [HL] 2 16 Z 0 0 C	RL A 2 8 Z 0 0 C	RR B 2 8 Z 0 0 C	RR C 2 8 Z 0 0 C	RR D 2 8 Z 0 0 C	RR E 2 8 Z 0 0 C	RR H 2 8 Z 0 0 C	RR L 2 8 Z 0 0 C	RR [HL] 2 16 Z 0 0 C	RR A 2 8 Z 0 0 C
2x	SLA B 2 8 Z 0 0 C	SLA C 2 8 Z 0 0 C	SLA D 2 8 Z 0 0 C	SLA E 2 8 Z 0 0 C	SLA H 2 8 Z 0 0 C	SLA L 2 8 Z 0 0 C	SLA [HL] 2 16 Z 0 0 C	SLA A 2 8 Z 0 0 C	SRA B 2 8 Z 0 0 C	SRA C 2 8 Z 0 0 C	SRA D 2 8 Z 0 0 C	SRA E 2 8 Z 0 0 C	SRA H 2 8 Z 0 0 C	SRA L 2 8 Z 0 0 C	SRA [HL] 2 16 Z 0 0 C	SRA A 2 8 Z 0 0 C
3x	SWAP B 2 8 Z 0 0 0	SWAP C 2 8 Z 0 0 0	SWAP D 2 8 Z 0 0 0	SWAP E 2 8 Z 0 0 0	SWAP H 2 8 Z 0 0 0	SWAP L 2 8 Z 0 0 0	SWAP [HL] 2 16 Z 0 0 0	SWAP A 2 8 Z 0 0 0	SRL B 2 8 Z 0 0 C	SRL C 2 8 Z 0 0 C	SRL D 2 8 Z 0 0 C	SRL E 2 8 Z 0 0 C	SRL H 2 8 Z 0 0 C	SRL L 2 8 Z 0 0 C	SRL [HL] 2 16 Z 0 0 C	SRL A 2 8 Z 0 0 C
4x	BIT 0, B 2 8 Z 0 1 -	BIT 0, C 2 8 Z 0 1 -	BIT 0, D 2 8 Z 0 1 -	BIT 0, E 2 8 Z 0 1 -	BIT 0, H 2 8 Z 0 1 -	BIT 0, L 2 8 Z 0 1 -	BIT 0, [HL] 2 12 Z 0 1 -	BIT 0, A 2 8 Z 0 1 -	BIT 1, B 2 8 Z 0 1 -	BIT 1, C 2 8 Z 0 1 -	BIT 1, D 2 8 Z 0 1 -	BIT 1, E 2 8 Z 0 1 -	BIT 1, H 2 8 Z 0 1 -	BIT 1, L 2 8 Z 0 1 -	BIT 1, [HL] 2 12 Z 0 1 -	BIT 1, A 2 8 Z 0 1 -
5x	BIT 2, B 2 8 Z 0 1 -	BIT 2, C 2 8 Z 0 1 -	BIT 2, D 2 8 Z 0 1 -	BIT 2, E 2 8 Z 0 1 -	BIT 2, H 2 8 Z 0 1 -	BIT 2, L 2 8 Z 0 1 -	BIT 2, [HL] 2 12 Z 0 1 -	BIT 2, A 2 8 Z 0 1 -	BIT 3, B 2 8 Z 0 1 -	BIT 3, C 2 8 Z 0 1 -	BIT 3, D 2 8 Z 0 1 -	BIT 3, E 2 8 Z 0 1 -	BIT 3, H 2 8 Z 0 1 -	BIT 3, L 2 8 Z 0 1 -	BIT 3, [HL] 2 12 Z 0 1 -	BIT 3, A 2 8 Z 0 1 -
6x	BIT 4, B 2 8 Z 0 1 -	BIT 4, C 2 8 Z 0 1 -	BIT 4, D 2 8 Z 0 1 -	BIT 4, E 2 8 Z 0 1 -	BIT 4, H 2 8 Z 0 1 -	BIT 4, L 2 8 Z 0 1 -	BIT 4, [HL] 2 12 Z 0 1 -	BIT 4, A 2 8 Z 0 1 -	BIT 5, B 2 8 Z 0 1 -	BIT 5, C 2 8 Z 0 1 -	BIT 5, D 2 8 Z 0 1 -	BIT 5, E 2 8 Z 0 1 -	BIT 5, H 2 8 Z 0 1 -	BIT 5, L 2 8 Z 0 1 -	BIT 5, [HL] 2 12 Z 0 1 -	BIT 5, A 2 8 Z 0 1 -
7x	BIT 6, B 2 8 Z 0 1 -	BIT 6, C 2 8 Z 0 1 -	BIT 6, D 2 8 Z 0 1 -	BIT 6, E 2 8 Z 0 1 -	BIT 6, H 2 8 Z 0 1 -	BIT 6, L 2 8 Z 0 1 -	BIT 6, [HL] 2 12 Z 0 1 -	BIT 6, A 2 8 Z 0 1 -	BIT 7, B 2 8 Z 0 1 -	BIT 7, C 2 8 Z 0 1 -	BIT 7, D 2 8 Z 0 1 -	BIT 7, E 2 8 Z 0 1 -	BIT 7, H 2 8 Z 0 1 -	BIT 7, L 2 8 Z 0 1 -	BIT 7, [HL] 2 12 Z 0 1 -	BIT 7, A 2 8 Z 0 1 -
8x	RES 0, B 2 8 - - - -	RES 0, C 2 8 - - - -	RES 0, D 2 8 - - - -	RES 0, E 2 8 - - - -	RES 0, H 2 8 - - - -	RES 0, L 2 8 - - - -	RES 0, [HL] 2 16 - - - -	RES 0, A 2 8 - - - -	RES 1, B 2 8 - - - -	RES 1, C 2 8 - - - -	RES 1, D 2 8 - - - -	RES 1, E 2 8 - - - -	RES 1, H 2 8 - - - -	RES 1, L 2 8 - - - -	RES 1, [HL] 2 16 - - - -	RES 1, A 2 8 - - - -
9x	RES 2, B 2 8 - - - -	RES 2, C 2 8 - - - -	RES 2, D 2 8 - - - -	RES 2, E 2 8 - - - -	RES 2, H 2 8 - - - -	RES 2, L 2 8 - - - -	RES 2, [HL] 2 16 - - - -	RES 2, A 2 8 - - - -	RES 3, B 2 8 - - - -	RES 3, C 2 8 - - - -	RES 3, D 2 8 - - - -	RES 3, E 2 8 - - - -	RES 3, H 2 8 - - - -	RES 3, L 2 8 - - - -	RES 3, [HL] 2 16 - - - -	RES 3, A 2 8 - - - -
Ax	RES 4, B 2 8 - - - -	RES 4, C 2 8 - - - -	RES 4, D 2 8 - - - -	RES 4, E 2 8 - - - -	RES 4, H 2 8 - - - -	RES 4, L 2 8 - - - -	RES 4, [HL] 2 16 - - - -	RES 4, A 2 8 - - - -	RES 5, B 2 8 - - - -	RES 5, C 2 8 - - - -	RES 5, D 2 8 - - - -	RES 5, E 2 8 - - - -	RES 5, H 2 8 - - - -	RES 5, L 2 8 - - - -	RES 5, [HL] 2 16 - - - -	RES 5, A 2 8 - - - -
Bx	RES 6, B 2 8 - - - -	RES 6, C 2 8 - - - -	RES 6, D 2 8 - - - -	RES 6, E 2 8 - - - -	RES 6, H 2 8 - - - -	RES 6, L 2 8 - - - -	RES 6, [HL] 2 16 - - - -	RES 6, A 2 8 - - - -	RES 7, B 2 8 - - - -	RES 7, C 2 8 - - - -	RES 7, D 2 8 - - - -	RES 7, E 2 8 - - - -	RES 7, H 2 8 - - - -	RES 7, L 2 8 - - - -	RES 7, [HL] 2 16 - - - -	RES 7, A 2 8 - - - -
Cx	SET 0, B 2 8 - - - -	SET 0, C 2 8 - - - -	SET 0, D 2 8 - - - -	SET 0, E 2 8 - - - -	SET 0, H 2 8 - - - -	SET 0, L 2 8 - - - -	SET 0, [HL] 2 16 - - - -	SET 0, A 2 8 - - - -	SET 1, B 2 8 - - - -	SET 1, C 2 8 - - - -	SET 1, D 2 8 - - - -	SET 1, E 2 8 - - - -	SET 1, H 2 8 - - - -	SET 1, L 2 8 - - - -	SET 1, [HL] 2 16 - - - -	SET 1, A 2 8 - - - -
Dx	SET 2, B 2 8 - - - -	SET 2, C 2 8 - - - -	SET 2, D 2 8 - - - -	SET 2, E 2 8 - - - -	SET 2, H 2 8 - - - -	SET 2, L 2 8 - - - -	SET 2, [HL] 2 16 - - - -	SET 2, A 2 8 - - - -	SET 3, B 2 8 - - - -	SET 3, C 2 8 - - - -	SET 3, D 2 8 - - - -	SET 3, E 2 8 - - - -	SET 3, H 2 8 - - - -	SET 3, L 2 8 - - - -	SET 3, [HL] 2 16 - - - -	SET 3, A 2 8 - - - -
Ex	SET 4, B 2 8 - - - -	SET 4, C 2 8 - - - -	SET 4, D 2 8 - - - -	SET 4, E 2 8 - - - -	SET 4, H 2 8 - - - -	SET 4, L 2 8 - - - -	SET 4, [HL] 2 16 - - - -	SET 4, A 2 8 - - - -	SET 5, B 2 8 - - - -	SET 5, C 2 8 - - - -	SET 5, D 2 8 - - - -	SET 5, E 2 8 - - - -	SET 5, H 2 8 - - - -	SET 5, L 2 8 - - - -	SET 5, [HL] 2 16 - - - -	SET 5, A 2 8 - - - -
Fx	SET 6, B 2 8 - - - -	SET 6, C 2 8 - - - -	SET 6, D 2 8 - - - -	SET 6, E 2 8 - - - -	SET 6, H 2 8 - - - -	SET 6, L 2 8 - - - -	SET 6, [HL] 2 16 - - - -	SET 6, A 2 8 - - - -	SET 7, B 2 8 - - - -	SET 7, C 2 8 - - - -	SET 7, D 2 8 - - - -	SET 7, E 2 8 - - - -	SET 7, H 2 8 - - - -	SET 7, L 2 8 - - - -	SET 7, [HL] 2 16 - - - -	SET 7, A 2 8 - - - -

CPU.swift

```swift
private func fetch() -> (OperationCode,[Instruction]) {
    let opCode = self.readIncrPC()
    if opCode == GBConstants.ExtendedInstructionSetOpcode {
        return ((true, self.readIncrPC()),extendedInstructionSet)
    }
    return ((false,opCode),standardInstructionSet)
}
```

CPU.swift

```swift
private func decode(opCode:OperationCode,instr:[Instruction]) -> Instruction {
    return instr[Int(opCode.code)]
}
```



CPU.swift

```swift
private func execute(instruction:Instruction) -> Int {
    switch(instruction.length) {
    case InstructionLength.OneByte:
        instruction.execute()
        break
    case InstructionLength.TwoBytes:
        let arg = self.readIncrPC()
        instruction.execute(arg)
        break
    case InstructionLength.ThreeBytes:
        let lsb = self.readIncrPC()
        let msb = self.readIncrPC()
        let arg = EnhancedShort(lsb,msb)
        instruction.execute(arg)
        break
    }
    //return cycle overhead
    return self.willCycleOverhead(instruction) ? instruction.durationWithOverhead
                                               : instruction.duration
}
```

JP a16
3 16
- - - -

00 0E 10 C5 C3 00 01 00

PC + 2 (arguments)

CPU.swift

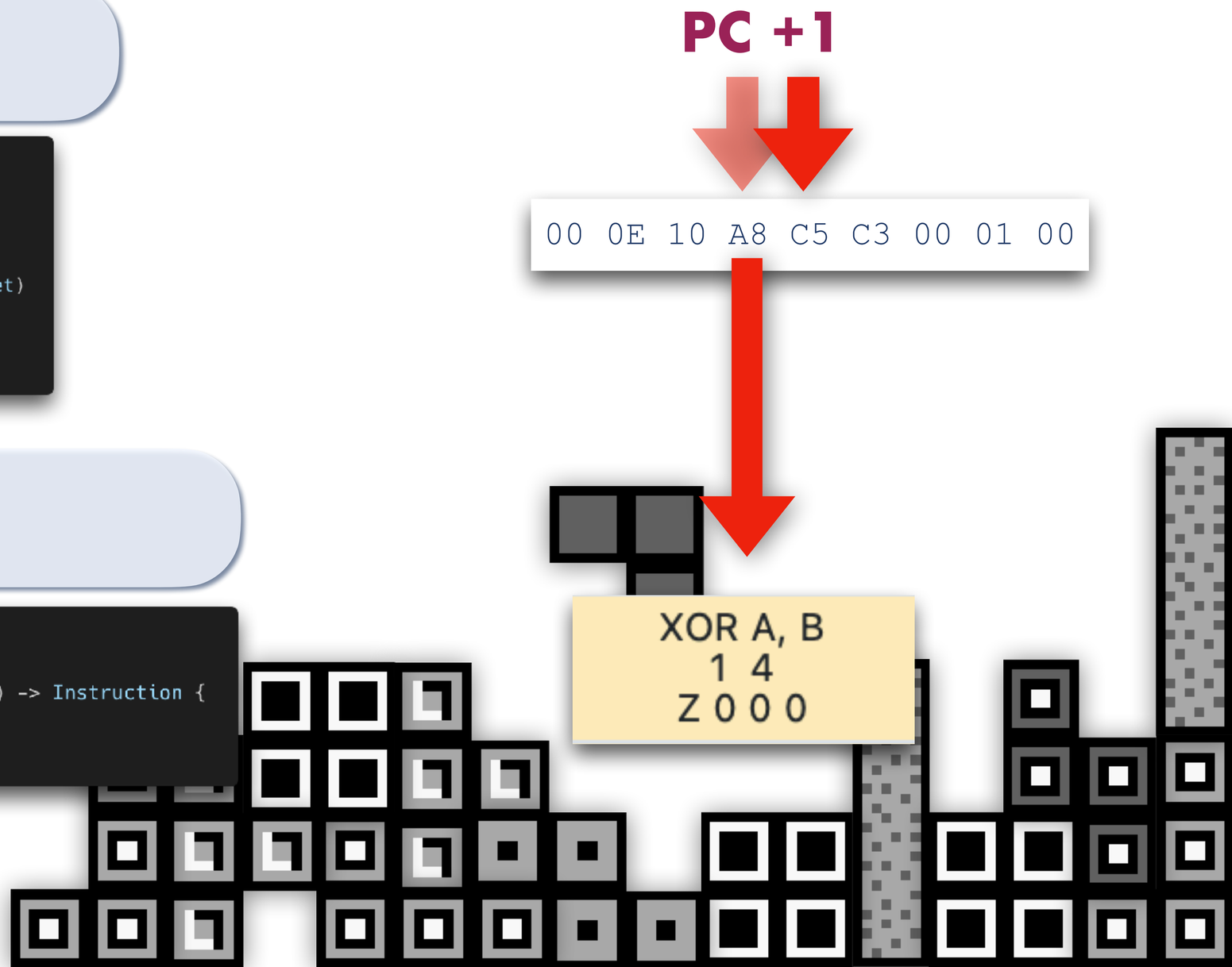
```swift
/// jump to address
internal func jumpTo(_ address:EnhancedShort) {
    self.registers.PC = address.value
}
```

CPU.swift

```swift
private func fetch() -> (OperationCode,[Instruction]) {
    let opCode = self.readIncrPC()
    if opCode == GBConstants.ExtendedInstructionSetOpcode {
        return ((true, self.readIncrPC()),extendedInstructionSet)
    }
    return ((false,opCode),standardInstructionSet)
}
```

CPU.swift

```swift
private func decode(opCode:OperationCode,instr:[Instruction]) -> Instruction {
    return instr[Int(opCode.code)]
}
```



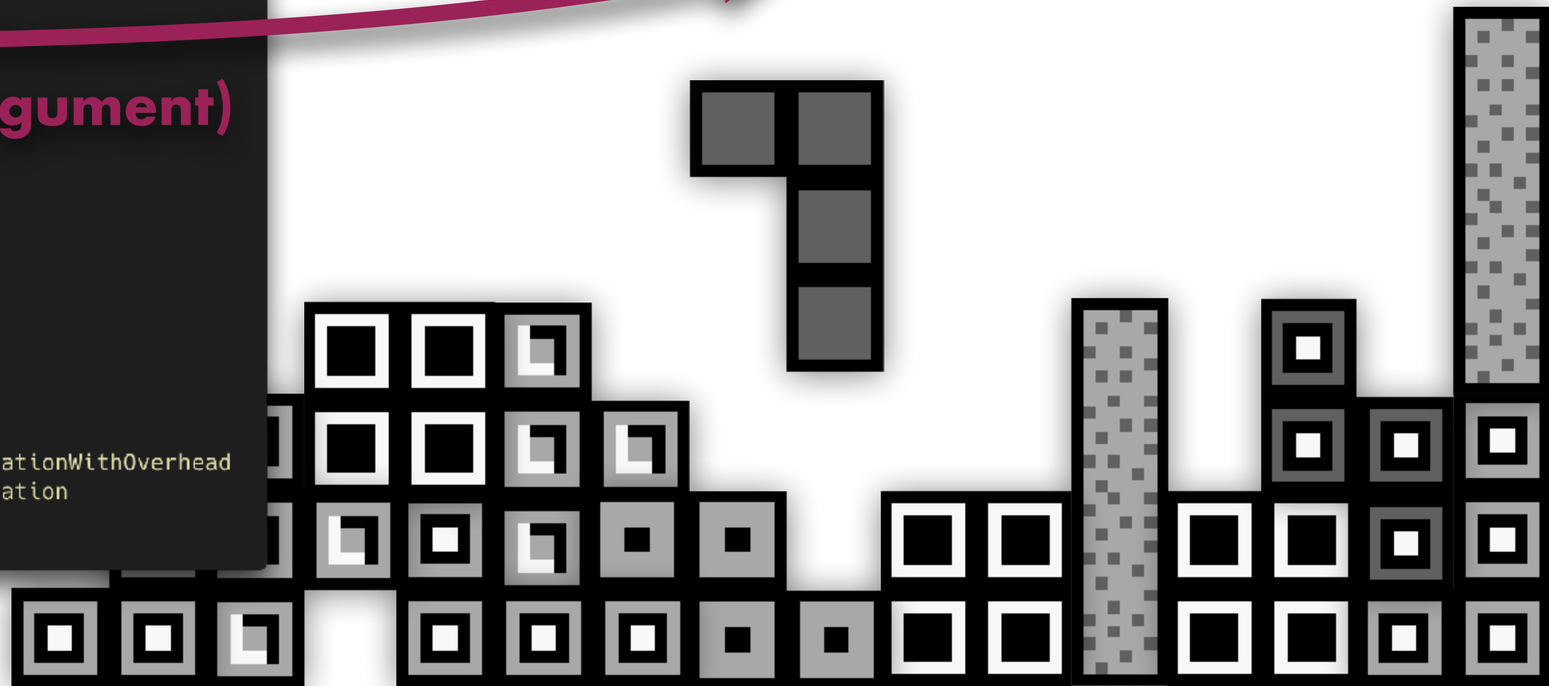
CPU.swift

```
private func execute(instruction:Instruction) -> Int {
    switch(instruction.length) {
    case InstructionLength.OneByte:
        instruction.execute()
        break
    case InstructionLength.TwoBytes:
        let arg = self.readIncrPC()
        instruction.execute(arg)
        break
    case InstructionLength.ThreeBytes:
        let lsb = self.readIncrPC()
        let msb = self.readIncrPC()
        let arg = EnhancedShort(lsb,msb)
        instruction.execute(arg)
        break
    }
    //return cycle overhead
    return self.willCycleOverhead(instruction) ? instruction.durationWithOverhead
                                               : instruction.duration
}
```

PC + 0 (argument)

XOR A, B
1 4
Z 0 0 0

00 0E 10 A8 C5 C3 00 01 00



XOR A (0xA8 → 0xAF)

XOR A, B
1 4
Z 0 0 0

```swift
/// xor val with A then stores result in A
internal func xor_a(_ val: Byte) {
    self.registers.A ^= val
    self.registers.conditionalSet(cond: self.registers.A == 0, flag: .ZERO)
    self.registers.clearFlags(.NEGATIVE,.HALF_CARRY,.CARRY)
}
```

CPUCore.swift

```swift
func xor_a_b() -> Void { self.xor_a(self.registers.B) }
func xor_a_c() -> Void { self.xor_a(self.registers.C) }
func xor_a_d() -> Void { self.xor_a(self.registers.D) }
func xor_a_e() -> Void { self.xor_a(self.registers.E) }
func xor_a_h() -> Void { self.xor_a(self.registers.H) }
func xor_a_l() -> Void { self.xor_a(self.registers.L) }
func xor_a_hlp() -> Void { self.xor_a(mmu
func xor_a_a() -> Void { self.xor_a(self.
```

CPUCore.swift

```swift
public class CPUImplementation: CPUCore, GameBoyInstructionSet {
    func asStandardInstructions() -> [Instruction] {
        return [
            ...
            Instruction(opCode: 0xA8, length: 1, name: "XOR A, B", duration:4,xor_a_b),
            Instruction(opCode: 0xA9, length: 1, name: "XOR A, C", duration:4,xor_a_c),
            Instruction(opCode: 0xAA, length: 1, name: "XOR A, D", duration:4,xor_a_d),
            ...
        ]
    }
}
```

PPU.swift

```swift
public func tick(_ masterCycles:Int, _ frameCycles:Int) -> Void {
    if(ly >= GBConstants.ScreenHeight)
    {
        newMode = LCDStatMode.VBLANK
        statInterruptTriggered = ios.readLCDStatFlag(.VBlankInterruptSource)
    }
    else if(self.lineSync < GBConstants.PIXEL_RENDER_TRIGGER)
    {
        newMode = LCDStatMode.OAM_SEARCH
        statInterruptTriggered = ios.readLCDStatFlag(.OAMInterruptSource)
    }
    else if(self.lineSync < GBConstants.HBLANK_TRIGGER)
    {
        newMode = LCDStatMode.PIXEL_TRANSFER
    }
    else if(self.lineSync < GBConstants.MCyclesPerScanline)
    {
        newMode = LCDStatMode.HBLANK
        statInterruptTriggered = ios.readLCDStatFlag(.HBlankInterruptSource)
    }

    if(curMode != newMode) {
        if(newMode == .PIXEL_TRANSFER){
            self.scanline()
        }
        else if(newMode == .VBLANK){
            self.interrupts.setInterruptFlagValue(.VBlank,true);
            self.commitFrame()
        }
        else if(curMode == .VBLANK && newMode == .OAM_SEARCH){
            self.prepareNextFrame()
        }
    }
}
```