# Family Web Computer

～Web上でレトロハードを楽しむ～

2014.10.11 ぺちぱな天

@MIRROR_

# Family Web Computer

## - Agenda -

| | |
|---|---|
| Introduction | // はじめに |
| Subject | // 動作・開発環境 |
| Working! | // ちょっとお試し |
| Resolved...? | // まとめ |
| Question and answer | // 質疑応答 |

# Family Web Computer

## - Introduction -

自己紹介

@MIRROR_ 

とりあえず、プログラマやってます。

BASIC ＞マシン語(SC61860)＞C言語＞
C++＞VisualBasic＞Java＞JavaScript＞
VisualBasic.NET＞php...＞

# Family Web Computer

## - Introduction -

### 配布資料について

fwc-src-20141011.zip … ソースコード

簡単なサンプルからゲームとしての流れを一通り作成したプロジェクトのmakefile及びソースコードをまとめています。
コンパイルは、中のREADME.txtを参照してください。

fwc-bin-20141011.zip … 実行モジュール

上記ソースコードのコンパイル済み実行モジュールをまとめています。
エミュレータ上で動作します。

# Family Web Computer

## - Introduction -

突然ですが、

# 質問です！

# Family Web Computer

## - Introduction -

# “Family Computer”

って知っていますか？

# Family Web Computer

## - Introduction -


1983.7.15
任天堂より発売の
ロムカセット交換式家庭用
ゲーム機
いわゆるファミコン

# Family Web Computer

## - Introduction -

AV仕様
ファミリーコンピュータ
（Newファミコン）

1993.12.1
SUPER FAMICOMの
発売後に登場

ビデオ出力に対応
コントローラが変更

カセットのイジェクト
レバーは廃止

# Family Web Computer

## - Introduction -

どんなゲームがあったの？
・アクション
・シューティング
・パズル

アーケードの移植から
オリジナル作品まで様々

# Family Web Computer

## - Introduction -

今でも遊べるの？

旧ファミコン … アナログ地上波が受信可能な
テレビが必要。
ビデオ出力は要改造。

新ファミコン … ビデオ入力対応のテレビでOK。
音声はモノラル出力。

# Family Web Computer

## - Introduction -

そもそもスペックは？

| | | |
|---|---|---|
| CPU | RP2A03 (MOS 6502 カスタム 1.79MHz) | |
| Memory | RAM | 2KB |
| | ROM | 32KB |
| | Video | 2KB |
| Sound | 矩形波2音、三角波1音、<br>ノイズ1音、⊿PCM1音 | |
| Sprite | 8×8 or 8x16 | 64枚 |
| BG | 256x240 ２画面分 | １枚 |

# Family Web Computer

## - Introduction -

そもそもスペックは？

| | | |
|---|---|---|
| CPU | RP2A03 (MOS 6502 カスタム 1.79MHz) | |
| Memory | RAM | 2KB |
| | ROM | 32KB |
| | Video | 2KB |

**8bit CPU**

アドレス空間は16bitなので、通常で64KBのメモリアクセスが可能。
64KB中、Low側がRAMとメモリマップドI/O、High側はROM。

**注意点**

インデックスレジスタは8bitなのでテーブルは256バイト単位が基本。
RAMが少ないので、省メモリを心がけます。

# Family Web Computer

## - Introduction -

### そもそもスペックは？

**Sprite**
任意の場所に表示可能。ただし、横に8キャラクタまで。
色は3色＋透明のパレット4本から選択。

**BG**
８ｘ8のキャラクタをパターンネームテーブルに並べて表示。
色は4色パレット4本から2ｘ2キャラクタ単位に選択。

**Bitmap方式ではなく、８ｘ8のキャラクタ単位で表現するのが基本です！**

| | | |
|---|---|---|
| Sprite | 8×8 or 8x16 | 64枚 |
| BG | 256x240 | ２画面分　１枚 |

# Family Web Computer

## - Introduction -

自分でゲームを作ることはできる？

- アセンブラやC言語がある
- キャラクタ変換ツールがある
- 結構詳細な技術資料もある
- 実機がなくてもエミュレータが揃っている

制約が多いので、根気が必要ですが
十分いけます。

# Family Web Computer

## - Introduction -

自分でゲームを作ることはできる？

今回は、
C言語で単純なゲームを作る
手順を紹介していきます！

なお今回は、Windows中心の紹介になりますのであしからず。

# Family Web Computer

- Introduction -

# 準備する物

# Family Web Computer

## - Subject -

開発環境として必要なものは

- コンパイラ、アセンブラ
- グラフィックの編集ツール
- BGMやSEの編集ツール
- 技術資料
- 動作確認用のエミュレータ

# Family Web Computer

## - Subject -

開発環境はコチラ

コンパイラ、アセンブラ

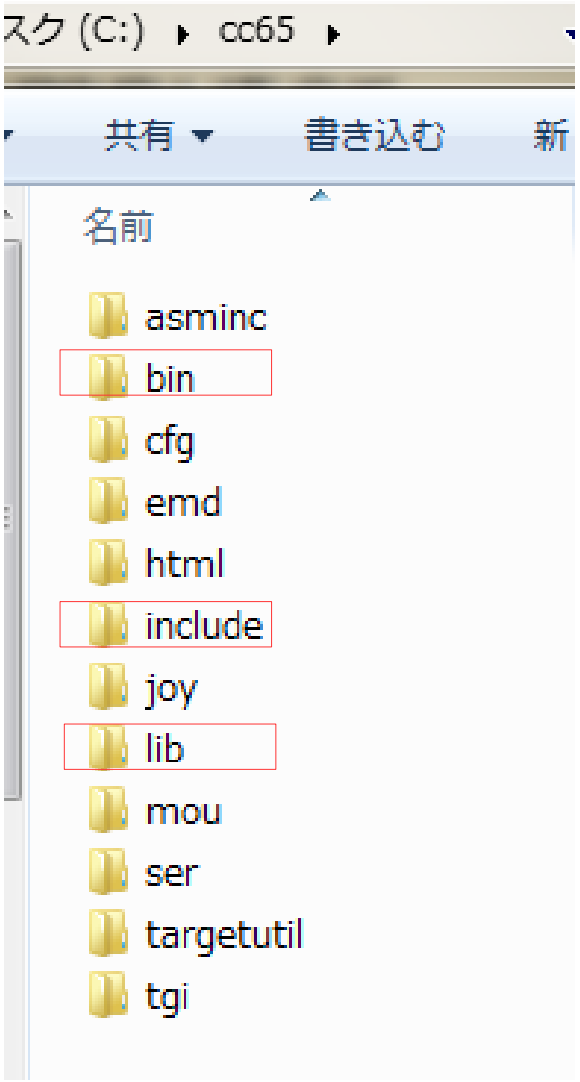
cc65 -the 6502 C complier

<http://cc65.github.io/cc65/>

Windows版の他、
ソースコードがあるためビルドできれば
LinuxやMacでもイケル…はず。
他にもnesasmとかWDC-SDKなどがあります。

# Family Web Computer

## - Subject -




通常のCコンパイラと同様の構成です。

cc65/bin
… コンパイラ、アセンブラ等
実行モジュール類。

cc65/include
… cc65附属の標準関数等の
ヘッダファイル群。

cc65/lib
… cc65附属の標準関数等の
ライブラリ群。

コンパイラは、
msys等のmakeから呼び出して使えます。

# Family Web Computer

## - Subject -

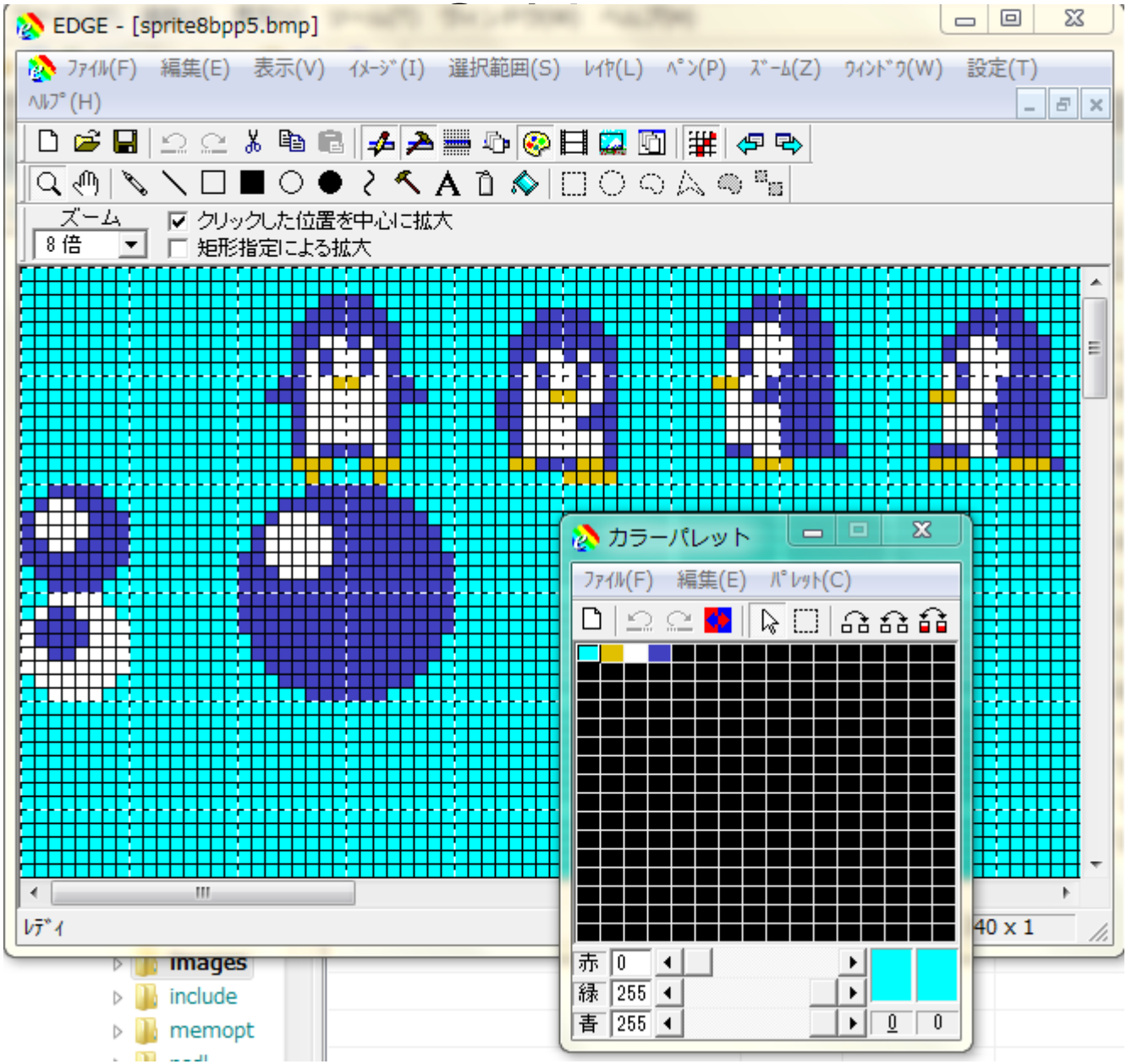
グラフィックの編集ツール

EDGE

<http://takabosoft.com/edge>
256色のパレットが操作できるペイントソフト

2pp(4色パレットカラー)のビットマップデータが必要ですが、後でツールを使って変換します。
とにかく、4色パレット分で絵が描ければOKです。

# Family Web Computer



# Family Web Computer

## - Subject -

グラフィックの編集ツール

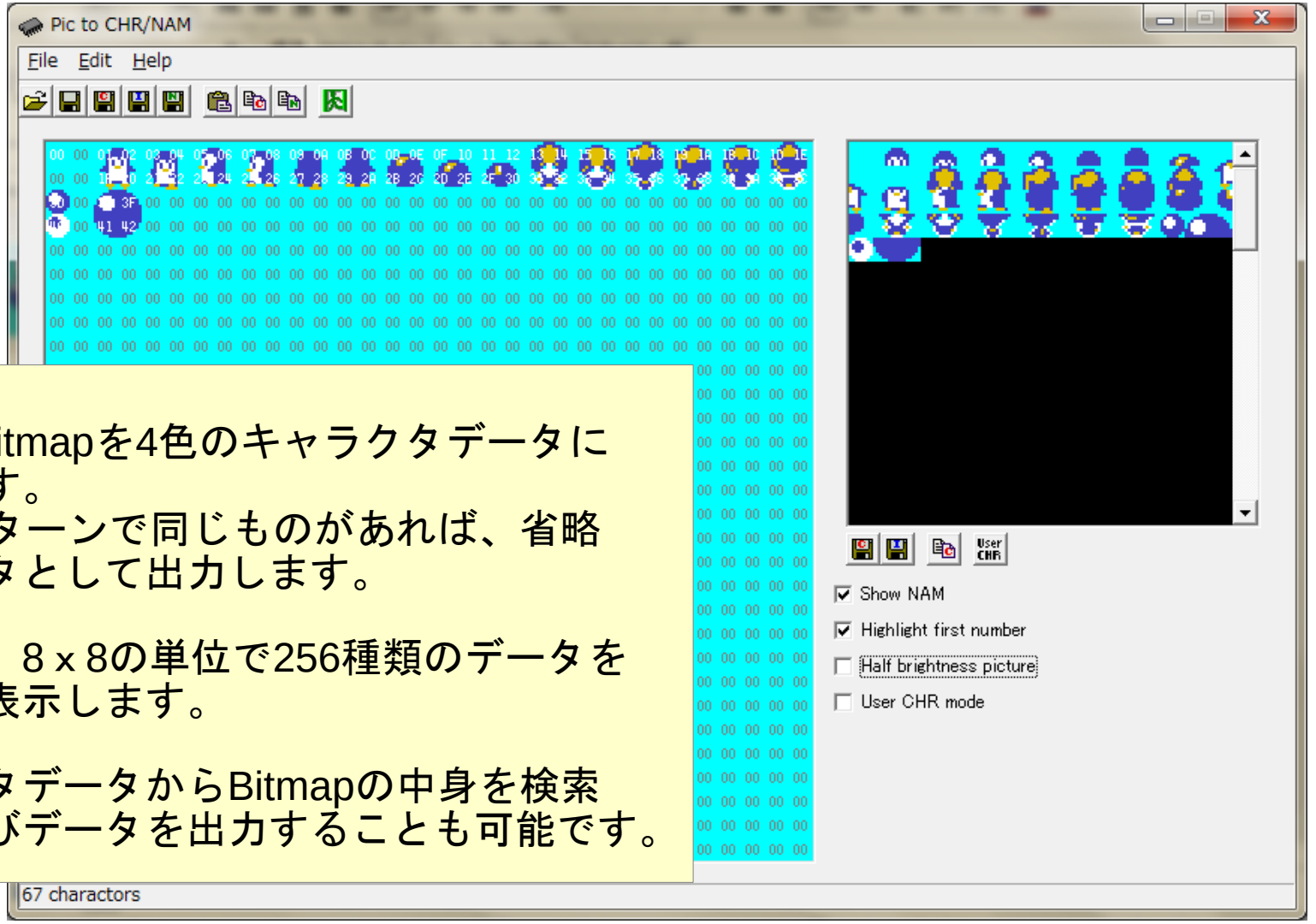
PicToCHR、YY-CHR

<http://www.geocities.co.jp/Playtown-Denei/4503/>
パターンテーブル、ネームテーブル、
キャラクタテーブルの作成には必須となるソフト

ここで素材として使えるフォーマットに変換します。
Windows版のみです。LinuxやMacでは該当するアプリを
別途開発する必要があります。

# Family Web Computer

## - Subject -




PicToCHR

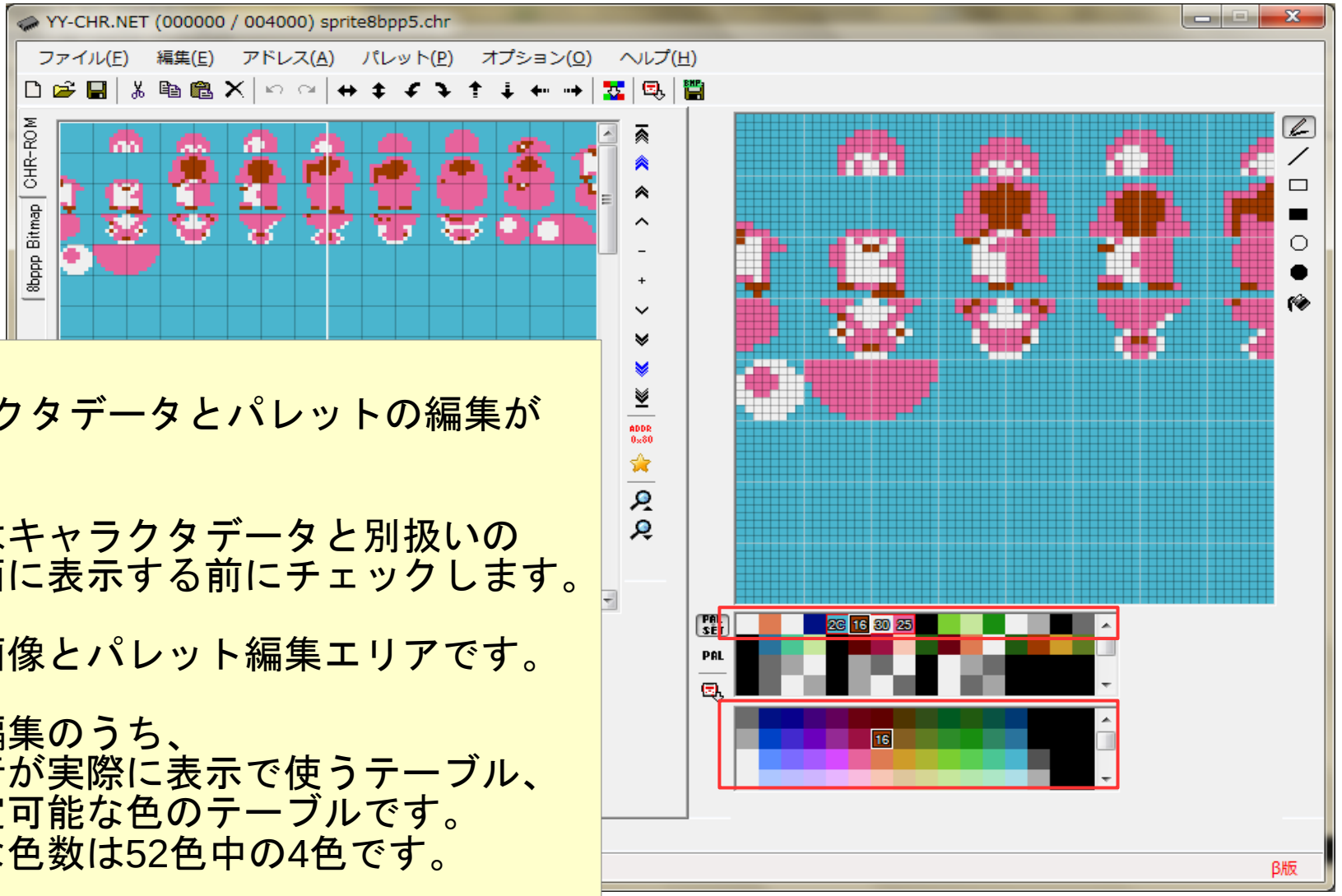
256色のBitmapを4色のキャラクタデータに変換します。
8 x 8のパターンで同じものがあれば、省略したデータとして出力します。

画面では、8 x 8の単位で256種類のデータを指定して表示します。

キャラクタデータからBitmapの中身を検索して、並びデータを出力することも可能です。

# Family Web Computer

## - Subject -




YY-CHR

4色キャラクタデータとパレットの編集ができます。

パレットはキャラクタデータと別扱いのため、画面に表示する前にチェックします。

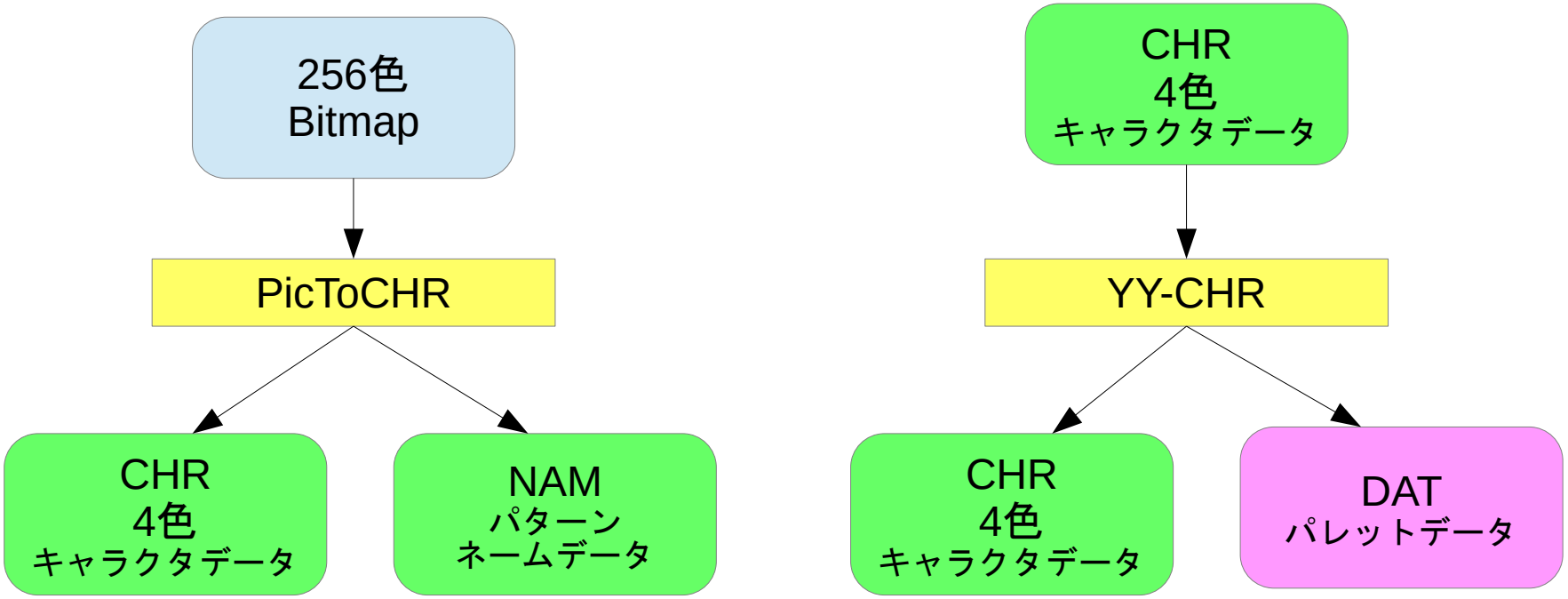
画面右は画像とパレット編集エリアです。

パレット編集のうち、
上段の１行が実際に表示で使うテーブル、
下段は指定可能な色のテーブルです。
利用可能な色数は52色中の4色です。

# Family Web Computer

## - Subject -

グラフィックデータ変換、編集の流れ




CHRデータはそのままキャラクタROMデータへ、
他はプログラムのデータとして管理します。

# Family Web Computer

## - Subject -

BGMやSEの編集ツール

Ｓ．Ｗ．　Ｈｏｍｅｐａｇｅ

NES Sound Driver & Library (NSD.Lib)
<http://shaw.la.coocan.jp/nsdl/>

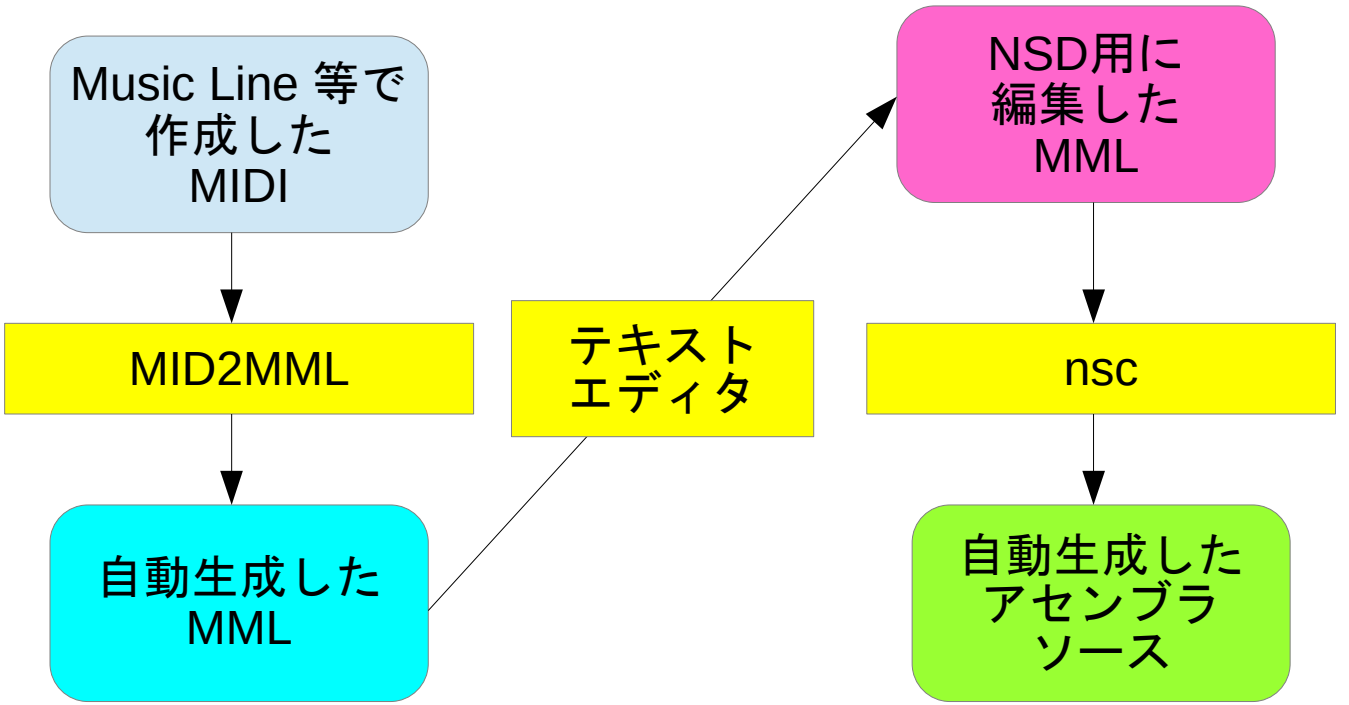
C言語からも呼び出せるライブラリで、拡張音源もバッチリ。

データはMML(Music Macro Language)で記述し、
同梱のMMLコンパイラでアセンブラソースに変換します。

# Family Web Computer

## - Subject -

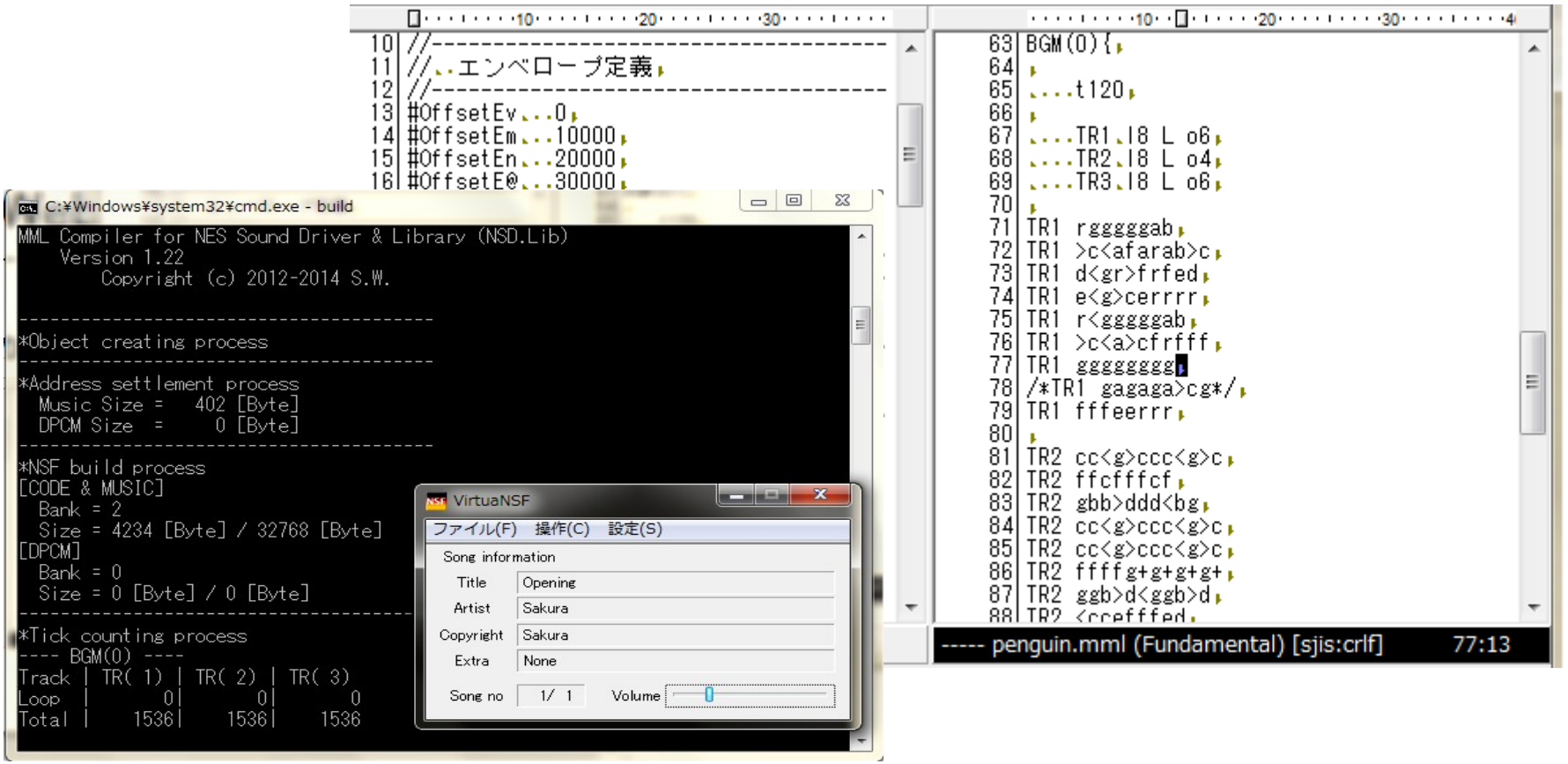
### BGMやSEの編集の流れ



# Family Web Computer

## - Subject -

BGMやSEの編集の流れ



# Family Web Computer

## - Subject -

技術資料

Nesdev

<http://wiki.nesdev.com/w/index.php/Nesdev>
英語ですが、物凄く充実してます。

NES研究室

<http://hp.vector.co.jp/authors/VA042397/nes/>
グラフィックの仕組み等、丁寧な解説があります。

cc65@wiki

<http://www34.atwiki.jp/cc65/>
雛形ソースやヘッダファイルとサンプルコードがあります。

# Family Web Computer

## - Subject -

動作確認用のエミュレータ

VirtuaNES

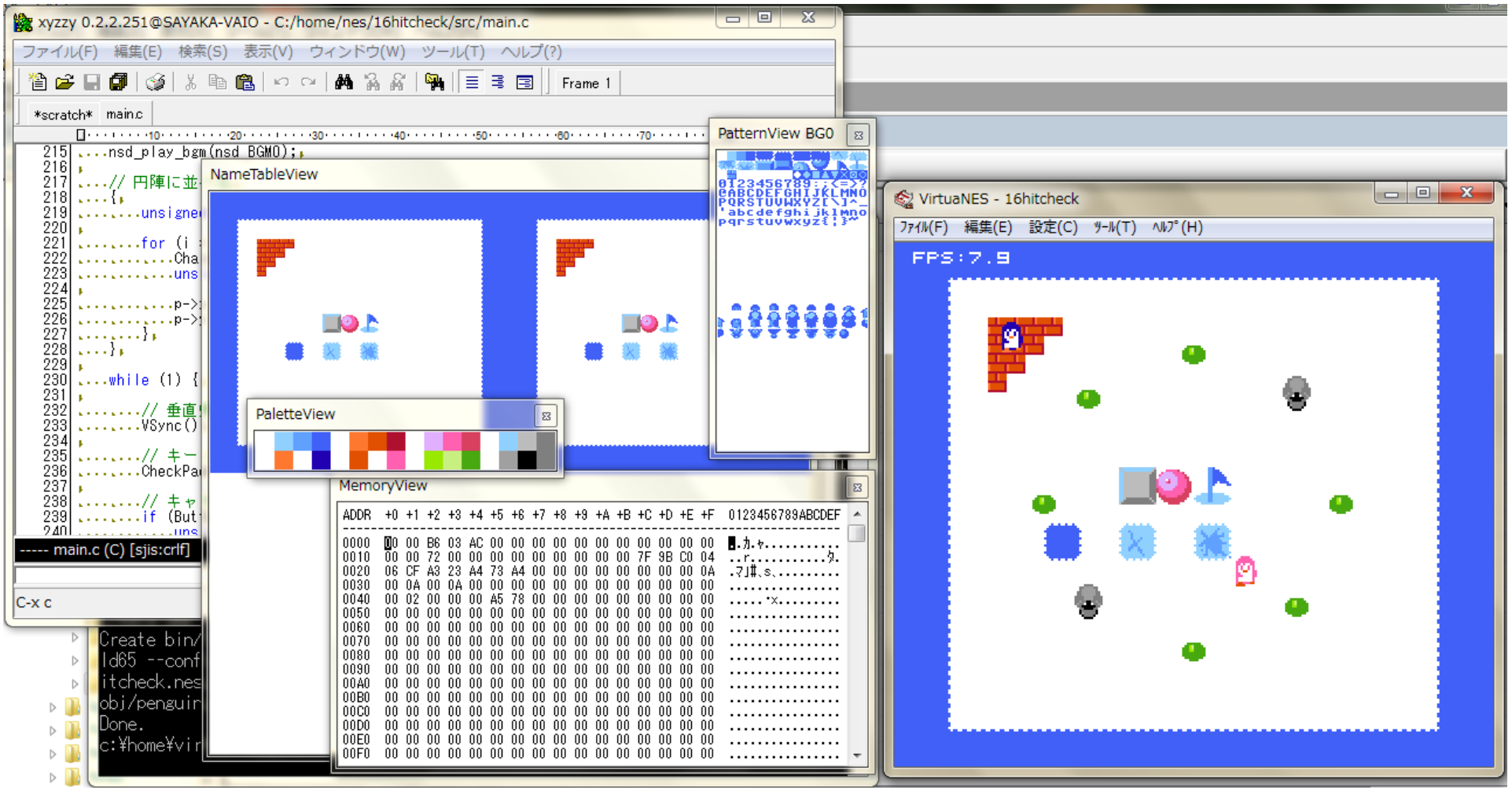
<http://virtuanes.s1.xrea.com/>

メモリビュアーがついているのでデバッガ代わり。

特定のメモリに書き出したり、
マップファイルとにらめっこして値の変化を確認します。

# Family Web Computer

## - Subject -



# Family Web Computer

## - Subject -

JavaScript製エミュレータも公開されています！

CycloaJS <http://ledyba.org/utl/CycloaJS/>

JSNES　<https://fir.sh/projects/jsnes/>

ここまでくれば、立派なWebアプリに！
作成過程ではWindowsがないと辛いのですが、
公開するときはプラットフォームを選びません！

今回は、JSNES上でサンプルを動かします！

# Family Web Computer

- Subject -

材料が揃ったら、
早速

# Let's C Language !

# Family Web Computer

- Subject -

今回配布のソースコードに同梱している
Makefileは、
msysやVirtuaNESに依存しています。

実行前に環境変数の設定が必要です。

ビルドしてみたい方は
README.txtを参照してください。

# Family Web Computer

## - Subject -

C言語で「こんにちは」

- cc65でmain()書いてCUI的なアプリ作れます。

```
// http://www34.atwiki.jp/cc65/pages/14.html
#include <conio.h>
int main (void)
{
    clrscr();
    cprintf("hello cc65");
    while(1)
    {
    }
    return 0;
}
```

# Family Web Computer

## - Subject -

C言語で「こんにちは」

- Makefileを準備してビルド！

```
C:\home\nes\01hello>make
cl65 -t nes -I c:\cc65/include --create-dep obj/main.o.dep -c -o obj/main.o src/
main.c
Create bin/01hello.nes
ld65 -t nes -L c:\cc65/lib -m bin/01hello.nes.map -o bin/01hello.nes --obj  obj/
main.o  --lib nes.lib atmos.lib
Done.
```

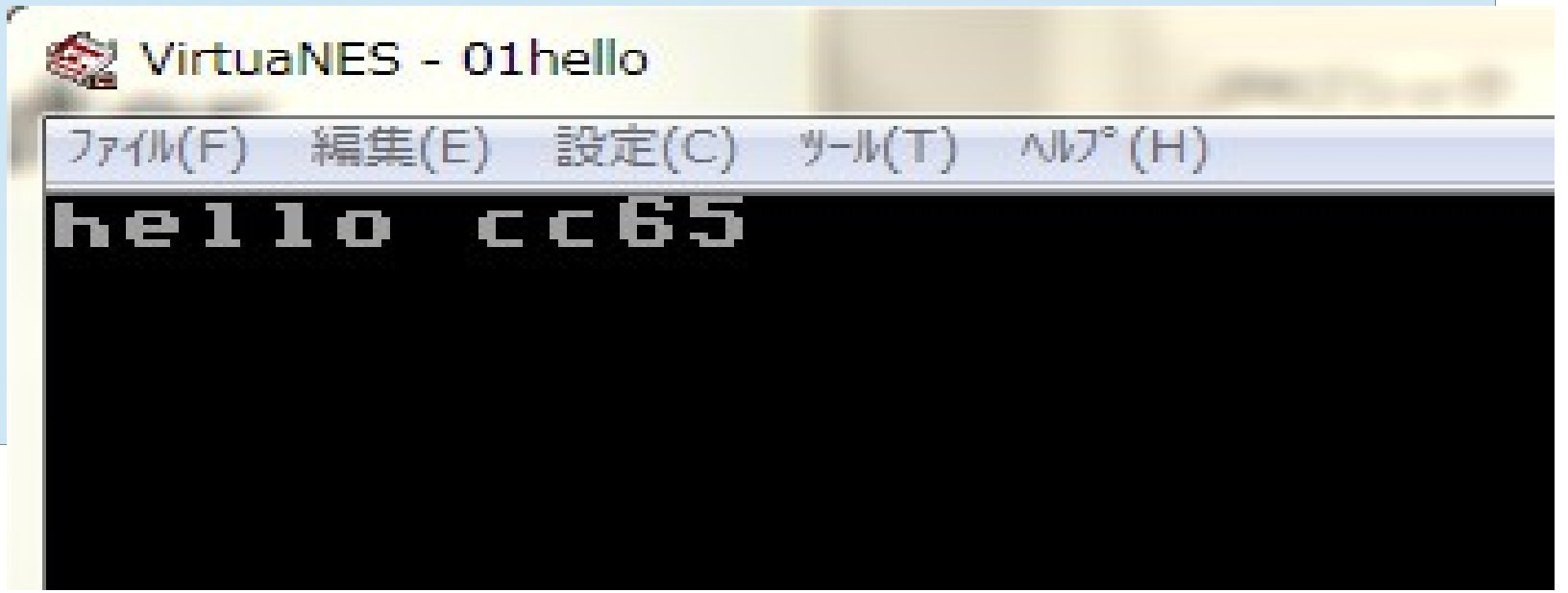
# Family Web Computer

## - Subject -

C言語で「こんにちは」

- エミュレータで動作確認

```
C:\home\nes\01hello>make run
c:\opt\virtualnes\VirtuaNES.exe bin/01hello.nes
```

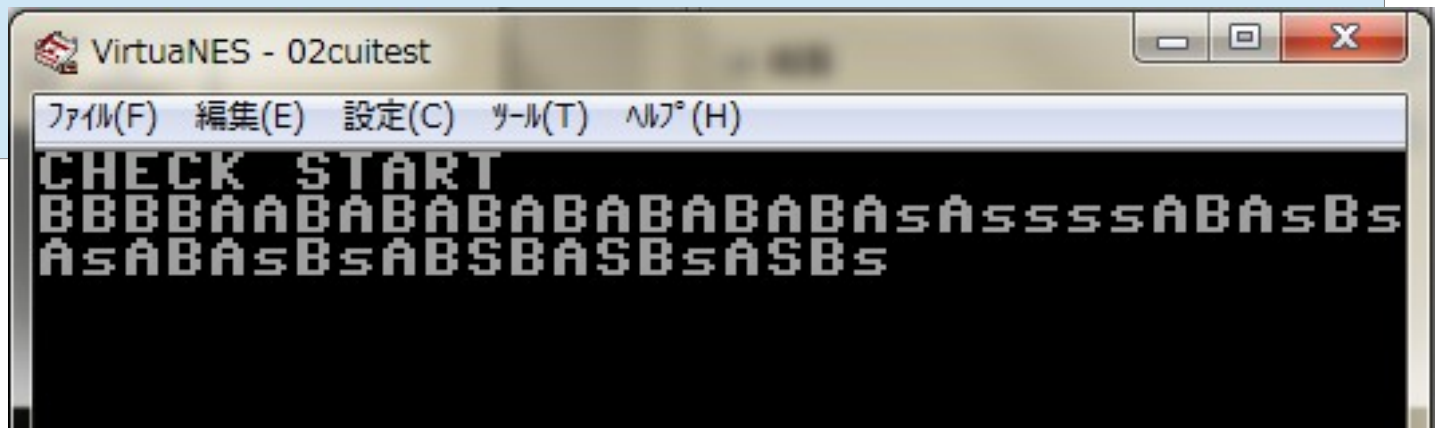


# Family Web Computer

## - Subject -

main()書いて、CUIっぽいアプリができます。

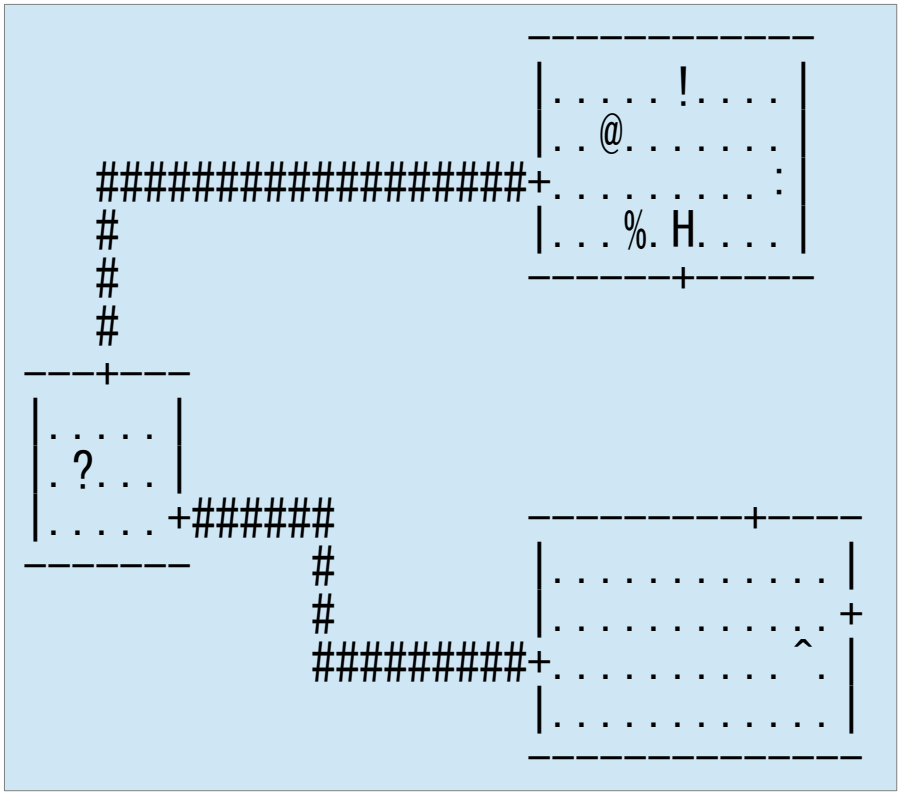
```
C:\home\nes\02cuitest>make run
cl65 -t nes -I c:\cc65/include --create-dep obj/main.o.dep -c -o obj/main.o src/
main.c
Create bin/02cuitest.nes
ld65 -t nes -L c:\cc65/lib -m bin/02cuitest.nes.map -o bin/02cuitest.nes --obj
obj/main.o  --lib nes.lib atmos.lib
Done.
c:\opt\virtualnes\VirtuaNES.exe bin/02cuitest.nes
```



# Family Web Computer

## - Subject -

コンソール画面だけでも、
Rogue みたいなゲームが作れそうです。



# Family Web Computer

- Subject -

準備はここまで！

# Family Web Computer

- Subject -

ようやくここからが

# 本番です！

# Family Web Computer

– Working! –

# ゲームを作るよ！

# Family Web Computer

– Working! –

今回は単純な

# 押し出しパズル

# Family Web Computer

## - Working! -

作るのは、

１．背景画面の表示
２．キャラクタの表示
３．キャラクタの操作
４．接触判定
５．ルールの実装

… あたりになります！

# Family Web Computer

– Working! –

# WARNING

# Family Web Computer

– Working! –

C言語で作成しますが、
標準ライブラリは
使いません。

# Family Web Computer

## - Working! -

メモリマップの状況を確認！

bin/01hello.nes.map

Segment list:

| Name | Start | End | Size | Align |
|---|---|---|---|---|
| CHARS | 000000 | 000FFF | 001000 | 00001 |
| HEADER | 000000 | 00000F | 000010 | 00001 |
| ZEROPAGE | 000002 | 00001B | 00001A | 00001 |
| DATA | 006000 | 00602F | 000030 | 00001 |
| BSS | 006030 | 006060 | 000031 | 00001 |
| STARTUP | 008000 | 00805C | 00005D | 00001 |
| INIT | 00805D | 00809E | 000042 | 00001 |
| CODE | 00809F | 0088B8 | 00081A | |
| RODATA | 0088B9 | 0089FF | 000147 | |
| VECTORS | 00FFF6 | 010001 | 00000C | |

変数の領域は
バックアップ
メモリの場所
になっている！

# Family Web Computer

## - Working! -

### メモリマップの状況を確認！

bin/01hello.nes.map

Segment list:

| Name | Start | End | Size | |
|---|---|---|---|---|
| CHARS | 000000 | 000FFF | 001000 | |
| HEADER | 000000 | 00000F | 000010 | |
| ZEROPAGE | 000002 | 00001B | 00001A | |
| DATA | 006000 | 00602F | 000030 | 00 |
| BSS | 006030 | 006060 | 000031 | 00 |
| STARTUP | 008000 | 00805C | 00005D | 00 |
| INIT | 00805D | 00809E | 000042 | 00001 |
| CODE | 00809F | 0088B8 | 00081A | 00001 |
| RODATA | 0088B9 | 0089FF | 000147 | 00001 |
| VECTORS | 00FFF6 | 010001 | 00000C | 00001 |

数行でもこのサイズ。
少しでもメモリを
稼ぎたいので、
今回は
標準ライブラリを
外します。

# Family Web Computer

## - Working! -

こんなことやります。

- メモリレイアウト定義を差し替え
- スタートアップを差し替え
- 割込み関数を用意

… 組み込み系の王道です！（USO）

# Family Web Computer

## – Working! –

## メモリレイアウト定義を差し替え (before)

cc65\cfg\nes.cfg (一部抜粋)

```
# standard 2k SRAM (-zeropage)
# $0100-$0200 cpu stack
# $0200-$0500 3 pages for ppu memory write buffer
# $0500-$0800 3 pages for cc65 parameter stack
SRAM:   file = "", start = $0500, size = __STACKSIZE__, define = yes;

# additional 8K SRAM Bank
# - data (run)
# - bss
# - heap
RAM:    file = "", start = $6000, size = $2000, define = yes;
```

内部RAMをスタック領域、
外部RAMを静的変数やヒープ領域に割り当て、
標準関数が使いやすい設定になっていた。

# Family Web Computer

## – Working! –

## メモリレイアウト定義を差し替え (after)

nes\03appbase\nes-mapper0.cfg (一部抜粋)

```
# standard 2k SRAM (-zeropage)
# $0100-$0200 cpu stack
# $0200-$0400 2 pages for cc65 parameter stack
# $0400-$0680 2.5 pages for data, bss, heap
# $0680-$0700 0.5 pages for ppu memory write buffer
# $0700-$0800 1 pages for ppu dma buffer
STACK:      start = $0200, size = __STACKSIZE__, type = rw, define = yes;
RAM:        start = $0400, size = $0280,         type = rw, define = yes;
CMDAREA:    start = $0680, size = $0080,         type = rw, define = yes;
DMAAREA:    start = $0700, size = $0100,         type = rw, define = yes;

# additional 8K SRAM Bank
SRAM:       start = $6000, size = $2000, type = rw, define = yes;
```

内部RAMにスタック領域、静的変数やヒープ領域に割り当て、
外部RAMを単純な拡張領域とした。
メモリサイズは小さいけど、速い内部RAMを変数で使います！

# Family Web Computer

## – Working! –

## スタートアップを差し替え

C:\home\nes\03appbase\src\startup.asm

```
.segment "STARTUP"
; リセット割り込み
.proc   Reset
    sei
    ldx  #$ff
    txs

    ; ソフトウェアスタック設定
    lda  #$ff
    sta  sp
    lda  #$03
    sta  sp + 1

    ; メイン関数呼び出し
    jsr  _NesMain
.endproc
```

電源投入/リセット時に呼ばれる割込み処理から
メイン関数を呼び出します。

標準関数のための initlib や donelib は呼びません。

関数呼び出し時のパラメータ変数や自動変数の
ためのスタック領域の開始だけは設定。

main() の名前のままだと
本来のスタートアップ処理がリンクされてしまう
ため、関数名を変更します。

関数名の前に「_」が付くのは、
アセンブラではC言語の識別子に必ず付加するお約束。

# Family Web Computer

## – Working! –

## 割込み関数を用意

C:\home\nes\03appbase\src\startup.asm

```
; VBlank割り込み
.proc       NMI
        ; レジスタ退避
        pha
        txa
        pha
        tya
        pha

        ; 割込発生時関数呼び出し
        jsr    _NMIProc

        ; レジスタ復帰
        pla
        tay
        pla
        tax
        pla
        rti
.endproc
```

```
; ベクタテーブル
.segment "VECTORS"
        .word       NMI
        .word       Reset
        .word       $0000
```

ベクタテーブルを用意して、
VBlank割込みとReset割込みに対応します。

Reset割込みは先のメイン関数呼び出し、
VBlank割込みは画像描画や定周期処理を行います。

VBlankはディスプレイの垂直走査線が一番下から
一番上に戻る期間を指します。

ファミコンの描画が綺麗にできる期間ですので、
ここから呼ばれる NMIProc() 内で描画処理を記述
します。

# Family Web Computer

– Working! –

標準関数を捨てたので、
ファミコンの機能は
メモリにマッピングされている
レジスタを直接読み書きして操作します。

今回は、cc65@wikiに掲載の
「kihon.h」の内容を利用しました。

ちょっとアレンジしたソースを
「nes.h」、「nes.c」として準備しています。

# Family Web Computer

– Working! –

それでは、

メイン処理の
中身を書いて行きます。

# Family Web Computer

## – Working! –

nes\03appbase\src\main.c

```
void NesMain()
{
	// PPUの初期化
	InitPPU();

	// パレット設定
	SetPalette(palettebg, 0);

	// 背景設定（パターンネームテーブル設定）
	FillNameTbl(0, 0, 32, 30, 0);
	SetBackground(0x21,0x06,"PHPer-na Ten! 2014", 18);

	// 背景属性設定
	FillAttribute(BG1_ATTR_HADR, BG1_ATTR_LADR, 0, 64);

	// DMA初期化
	InitDMA();

	// 背景の表示位置設定
	SetScroll(0, 0);

	// 描画開始(NMI割り込み有効)
	SetPPU(0x88, 0x1e);

	while (1) {
	}
}
```

標準関数は使いませんが、
ファミコンの機能を制御する関数を
呼び出して初期化しています。

最後の無限ループは関数を終わらせ
ないため。

本来は、キー入力チェックをして
キャラクタの動きを制御します。

printf()は使えませんので、
直接BGに書き込んでいます。

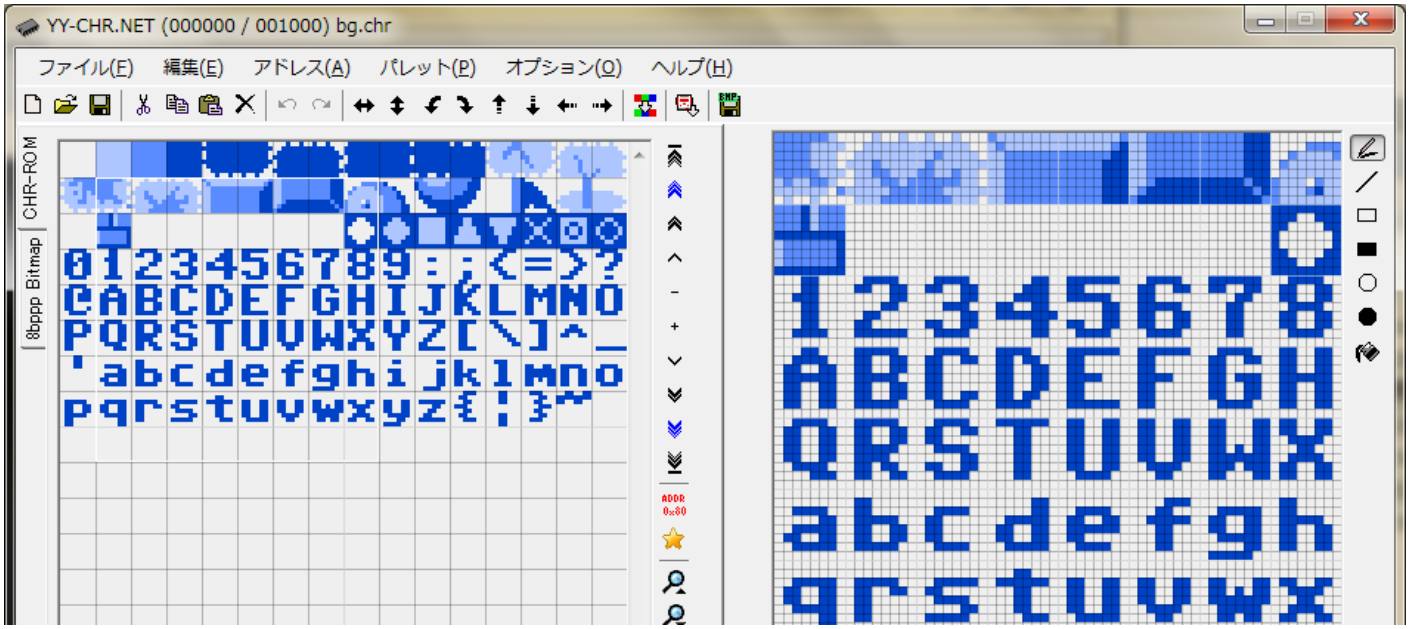
BGを表示するには、
キャラクタデータの準備の他に
パレット設定、
パターンネームテーブル設定、
背景属性設定が必要です。

# Family Web Computer

## – Working! –

PHPer□na Ten□ 2014

文字化けに見えるところは、
文字以外のデータが
入っているから！




参照：nes\03appbase

# Family Web Computer

## - Working! -

### メモリマップの状況を再確認！

bin/03appbase.nes.map

Segment list:

| Name | Start | End | Size | |
|---|---|---|---|---|
| CHARS | 000000 | 001FFF | 002000 | 00 |
| HEADER | 000000 | 00000F | 000010 | 000 |
| ZEROPAGE | 000002 | 00001B | 00001A | 000 |
| BSS | 000400 | 000407 | 000008 | 00001 |
| STARTUP | 008000 | 00801C | 00001D | 00001 |
| CODE | 00801D | 0082FB | 0002DF | 00001 |
| RODATA | 0082FC | 008326 | 00002B | 00001 |
| VECTORS | 00FFFA | 00FFFF | 000006 | 00001 |

だいぶスッキリになりました！

# Family Web Computer

## – Working! –

### nes.h (抜粋)

```
// コントローラ確認
#define CheckPad() \
{ \
	unsigned char i, j; \
	unsigned char p, c; \
	unsigned char ret; \
	*JOYPAD = 1; \
	*JOYPAD = 0; \
	padother= 0; \
	for(i = 0; i < 2; i++) { \
		p = padinfo[i][BTN_STAT_PRESS]; \
		c = 0; \
		for(j = 0; j < 8; j++) { \
			ret = *(JOYPAD + i); \
			c <<= 1; \
			c |= ret & 0x01; \
			padother |= ret & 0x0f; \
		} \
		padinfo[i][BTN_STAT_PRESS] =  c; \
		padinfo[i][BTN_STAT_DOWN ] =  c & ~p; \
		padinfo[i][BTN_STAT_UP   ] = ~c &  p; \
	} \
}

// ボタン押しっぱなし　player:0=1P , 1=2P
#define ButtonPress(player, btn) \
	(padinfo[player][BTN_STAT_PRESS] & btn)

// ボタン押す
#define ButtonDown(player, btn) \
	(padinfo[player][BTN_STAT_DOWN] & btn)

// ボタン離す
#define ButtonUp(player, btn) \
	(padinfo[player][BTN_STAT_UP] & btn)

// マイク・光線銃など
#define ControlOther(btn) \
	(padother & btn)

// PPUコントロールレジスタの初期化
#define InitPPU() \
	*REGIST_PPU_CTRL1 = 0x00; \
	*REGIST_PPU_CTRL2 = 0x00;

// PPUコントロールレジスタの設定
#define SetPPU(flag1, flag2) \
	*REGIST_PPU_CTRL1 = flag1; \
	*REGIST_PPU_CTRL2 = flag2;

// スクロール設定
#define SetScroll(x, y) \
	scroll_x = x, scroll_y = y; \
	*REGIST_SCROLL = scroll_x; \
	*REGIST_SCROLL = scroll_y;

// スクロール値の再設定
#define ResetScroll() \
	*REGIST_SCROLL = scroll_x; \
	*REGIST_SCROLL = scroll_y;

// 背景設定（繰り返し）
#define FillBackground(address1, address2, val, cnt) \
{ \
	unsigned char i; \
	*REGIST_ADR = address1; \
	*REGIST_ADR = address2; \
	for (i = 0; i < cnt; i++) *REGIST_GRA = val; \
}
```

# Family Web Computer

– Working! –

## nes.hの中身は
## ほとんどマクロ関数です。

普通のC言語の使い方ではありません。

基本はメモリマップドI/O操作と
姑息な速度稼ぎです。
気をつけないと、
同じ処理があちこちに散らばります！

# Family Web Computer

– Working! –

と、いうわけで…

C言語を

# 高級アセンブラ

として使って行きます。

8ビット以上の演算が楽！

自動変数が使えてメモリ管理が楽！

# Family Web Computer

– Working! –

＊注意＊
本来はmain()が呼ばれる前に初期化や
初期値のロードが行われますが

スタートアップを必要最低限にしているので
**グローバル変数は**
**初期化されません！**

# Family Web Computer

– Working! –

使用するメモリは、

自分で初期化します！

値が変わらない定数は、

“const”付けて

ROM領域に配置します！

# Family Web Computer

## - Working! -

malloc()は使えないので、
どうしてもメモリが欲しい場合は、
外部RAM領域を直接使います。

アセンブラより遅いので、
すぐに処理落ちします…が、
機能のシェイプアップで乗り越えましょう！

# Family Web Computer

## - Working! -

基本的な書き方が分かったところで、

# 背景画面処理の
# 中身を書いて行きます。

# Family Web Computer

## - Working! -

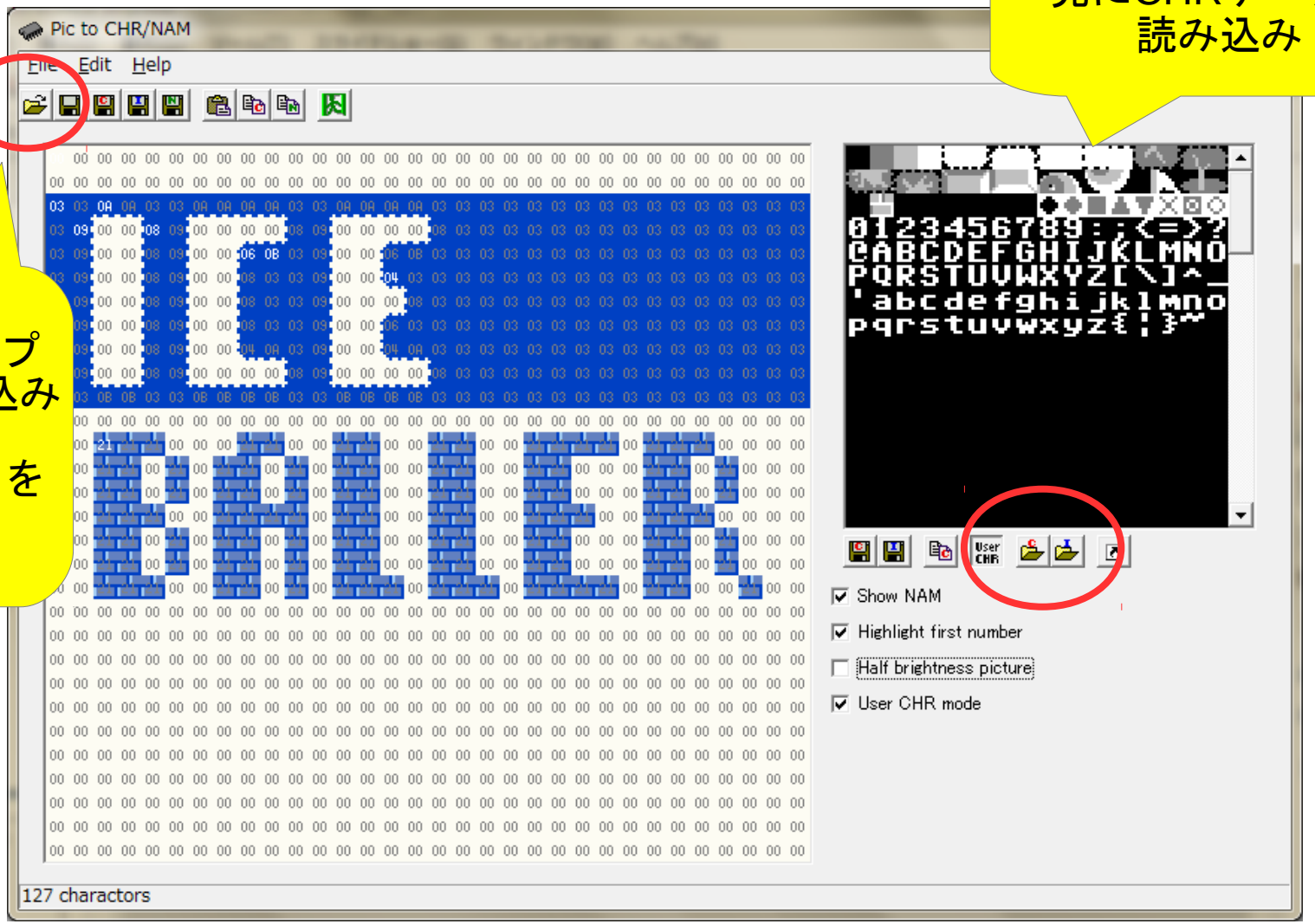
BG表示

８ｘ８のキャラクタを組み合わせて
背景画像を作り出します！

手っ取り速いのは、
PicToCHRでネームテーブルと言われる
キャラクタの配置情報を作り出して
そのままVRAMに転送します。

# Family Web Computer

## – Working! –




先にCHRデータを
読み込み

後から
ビットマップ
画像を読み込み

NAMデータを
保存

# Family Web Computer

## – Working! –

nes\04bgtest01\src\logo_ice.c （抜粋）

```
/*
 * logo_ice.c
 * Wed Oct 01 02:01:24 2014
 */
const unsigned char logo_ice[] = {
0x03,0x03,0x0a,0x0a,0x03,0x03,0x0a,0x0a,0x0a,0x0a,0x03,0x03,0x0a,0x0a,0x0a,0x0a,0x03,0x03,0x03,0x03,0x03,0x03,
0x03,0x09,0x00,0x00,0x08,0x09,0x00,0x00,0x00,0x00,0x08,0x09,0x00,0x00,0x00,0x00,0x08,0x03,0x03,0x03,0x03,0x03,
0x03,0x09,0x00,0x00,0x08,0x09,0x00,0x00,0x06,0x0b,0x03,0x09,0x00,0x00,0x06,0x0b,0x03,0x03,0x03,0x03,0x03,0x03,
0x03,0x09,0x00,0x00,0x08,0x09,0x00,0x00,0x08,0x03,0x03,0x09,0x00,0x00,0x04,0x03,0x03,0x03,0x03,0x03,0x03,0x03,
0x03,0x09,0x00,0x00,0x08,0x09,0x00,0x00,0x08,0x03,0x03,0x09,0x00,0x00,0x00,0x08,0x03,0x03,0x03,0x03,0x03,0x03,
0x03,0x09,0x00,0x00,0x08,0x09,0x00,0x00,0x08,0x03,0x03,0x09,0x00,0x00,0x06,0x03,0x03,0x03,0x03,0x03,0x03,0x03,
0x03,0x09,0x00,0x00,0x08,0x09,0x00,0x00,0x04,0x0a,0x03,0x09,0x00,0x00,0x04,0x0a,0x03,0x03,0x03,0x03,0x03,0x03,
0x03,0x09,0x00,0x00,0x08,0x09,0x00,0x00,0x00,0x00,0x08,0x09,0x00,0x00,0x00,0x00,0x08,0x03,0x03,0x03,0x03,0x03,
0x03,0x03,0x0b,0x0b,0x03,0x03,0x0b,0x0b,0x0b,0x0b,0x03,0x03,0x0b,0x0b,0x0b,0x0b,0x03,0x03,0x03,0x03,0x03,0x03,
};
```

今回は自作ツールでバイナリ
ファイルをCの配列データ化

参照：nes\04bgtest01

# Family Web Computer

## – Working! –

nes\04bgtest01\src\main.c （抜粋）

```
// メイン処理
void NesMain() {

    // PPUの初期化
    InitPPU();

    // パレット設定
    SetPalette(palettebg, 0);

    // 背景設定（パターンネームテーブル設定）
    FillNameTbl(0, 0, 32, 30, 0);
    SetNameTbl(logo_ice   , 0,  4, logo_ice_W   , logo_ice_H   );
    SetNameTbl(logo_baller, 2, 14, logo_baller_W, logo_baller_H);

    // 背景属性設定
    FillAttribute(BG1_ATTR_HADR, BG1_ATTR_LADR, 0, 64);
```

配列データをVRAMに転送！
背景属性は4ｘ4の区画毎に
パレットの指定を行います。

参照：nes\04bgtest01

# Family Web Computer

## – Working! –


参照：nes\04bgtest01

# Family Web Computer

## – Working! –

nes\04bgtest02\src\main.c (抜粋)

```
        while (1) {

                // 割込み終了待ち
                Sync();

                // BG転送コマンド準備
                ResetCMD();

                // BG転送コマンドデータ設定
                if (frame == 0) {
                        SetCMD((32-18)/2, 22, "PRESS START BUTTON", 18);

                } else if (frame == 128) {
                        SetCMD((32-18)/2, 22, "                  ", 18);
                }

                // BG転送データ設定終了
                SetEndCMD();

                ++frame;
```


後から画面の一部を
書き換えることもできます。
最高32キャラクタぐらいまで。
VBlank期間以外では
画面が崩れるので注意！

参照：nes\04bgtest02

# Family Web Computer

## – Working! –

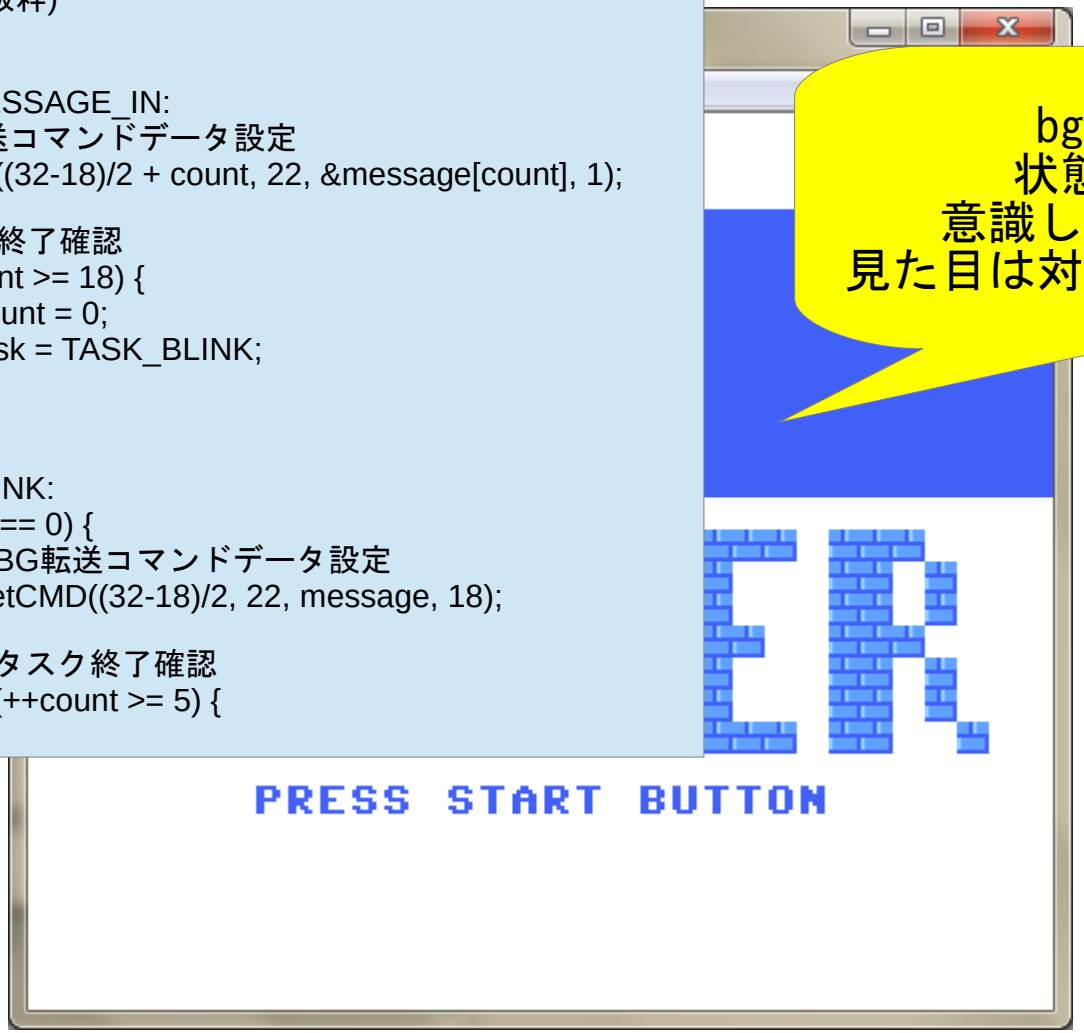
nes\04bgtest03\src\main.c (抜粋)

```
switch (task) {
case TASK_MESSAGE_IN:
        // BG転送コマンドデータ設定
        SetCMD((32-18)/2 + count, 22, &message[count], 1);

        // タスク終了確認
        if (++count >= 18) {
                count = 0;
                task = TASK_BLINK;
        }
        break;

case TASK_BLINK:
        if (frame == 0) {
                // BG転送コマンドデータ設定
                SetCMD((32-18)/2, 22, message, 18);

                // タスク終了確認
                if (++count >= 5) {
```




bgtest03は、
状態変化管理を
意識したバージョン。
見た目は対して変わりません。

参照：nes\04bgtest03

# Family Web Computer

## – Working! –

nes\04bgtest04\src\main.c (抜粋)

```
switch (task) {
case TASK_SCROLL_IN:
        // 背景の表示位置設定
        SetScroll(0, count);

        // タスク終了確認
        if (++count >= SCREEN_H - 1) {
                count = 255;
                task = TASK_WAIT;
        }
        break;

case TASK_WAIT:
        // 背景の表示位置設定
        SetScroll(0, SCREEN_H - 1);

        // タスク終了確認
        if (!count--) {
                count = SCREEN_H - 1;
                task = TASK_SCROLL_OUT;
```


bgtest04は、
上下方向にスクロールします。

参照：nes\04bgtest04

# Family Web Computer

## - Working! -

タイトルなどは、
ネームテーブルの情報を
そのまま配置する事が多いのですが、

ゲーム中の画面は、
マップ情報を用意して
そこから画面を作りだします！

# Family Web Computer

## – Working! –

nes\05map\src\map.c （抜粋）

```
// マップ
const unsigned char map01[] = {
        _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,
        _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,
        _,  _,  _,  _,  _,  _,  _,  _,  _,  0,  _,  _,
        _,  _,  _,  _,  B,  _,  _,  _,  0,  _,  0,  _,
        _,  _,  _,  B,  _,  B,  _,  _,  _,  0,  _,  _,
        _,  _,  B,  _,  _,  _,  B,  _,  _,  _,  _,  _,
        _,  _,  _,  _,  _,  _,  _,  B,  _,  _,  _,  _,
        _,  _,  _,  _,  _,  _,  _,  _,  B,  _,  _,  _,
        _,  _,  _,  _,  _,  _,  _,  _,  _,  B,  _,  _,
        _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,
        _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,  _,
};
```

今回は、12×11の区画に分け、
記号で区画のデータを
管理します！

参照：nes\05map

# Family Web Computer

## – Working! –


プログラムで
記号に合わせて
背景キャラを配置します。

```
// マップチップ
const unsigned char mapchips[][4]  = {
        {0x00,0x00,0x00,0x00}, // 歩けるところ
        {0x18,0x19,0x1a,0x1b}, // 転がすボール
        {0x1c,0x1d,0x1e,0x1f}, // ゴールの旗
        {0x14,0x15,0x16,0x17}, // 壊せない壁
        {0x21,0x21,0x21,0x21}, // 壊せるブロック
        {0x21,0x21,0x21,0x21}, // 動かせるブロック
};
```

記号を数値に変換して、
配列から2ｘ2のデータを配置してます。

参照：nes\05map

# Family Web Computer

– Working! –

次は、

キャラクタ表示と
操作を一気に行きます。

# Family Web Computer

– Working! –

スプライト表示

BG同様、８ｘ８のキャラクタを組み合わせて表示します。
好きな位置に表示できるのが強みです。

スプライトの領域に座標とキャラクタ番号を指定すれば、すぐに表示できます。

# Family Web Computer

## - Working! -

アニメーションはキャラクタの番号を変えるだけでできます。

# Family Web Computer

## – Working! –

nes\07anime\src\chara.c (抜粋)

```
// アニメーションテーブル
const char animetable[][4][2] = {
        {{4, 0x00}, {5,0x00}, {4, 0x00}, {5, 0x40}}, // 上
        {{2, 0x40}, {3,0x40}, {2, 0x40}, {3, 0x40}}, // 左
        {{0, 0x00}, {1,0x00}, {0, 0x00}, {1, 0x40}}, // 下
        {{2, 0x00}, {3,0x00}, {2, 0x00}, {3, 0x00}}, // 右
};

// アニメーションの選択
const char *anime = animetable[p->dir][frame];
const char *chara = charatable[anime[0]];
unsigned char flag = anime[1] | charapal[p->chr];

// DMA転送予約設定
SetDMA(p->x-8, p->y-12, chara[0], flag);
SetDMA(p->x  , p->y-12, chara[1], flag);
SetDMA(p->x-8, p->y- 4, chara[2], flag);
SetDMA(p->x  , p->y- 4, chara[3], flag);
```

表示座標、キャラクタ番号、
パレットと反転を含めた
フラグ情報を設定してます。

表示座標を変化させることで
キャラクタの移動が
できます。

# Family Web Computer

## - Working! -

キャラクタの移動

コントローラ情報を読み込み、
キャラクタの現在位置から移動方向に向けて
加減算します。

横方向の座標値は0～255なので
気をつけないと画面の反対側から現れます。

# Family Web Computer

## – Working! –

nes\07anime\src\main.c (抜粋)

```
// コントローラ確認
CheckPad();

char vx = 0;
char vy = 0;

// キー入力による移動
if (ButtonPress(i, BTN_RIGHT)) { vx++; p->dir = 1; }
if (ButtonPress(i, BTN_LEFT )) { vx--; p->dir = 3; }
if (ButtonPress(i, BTN_UP   )) { vy--; p->dir = 0; }
if (ButtonPress(i, BTN_DOWN )) { vy++; p->dir = 2; }

// 外圧による移動
vx += p->dx; p->dx = 0;
vy += p->dy; p->dy = 0;

p->x += vx;
p->y += vy;
```

方向ボタンの組み合わせで
斜めもOK!

ボタン以外の圧力を混ぜたり、
左右を入れ替えることも。

ここで計算しといた値を
表示のときに使います。

# Family Web Computer

– Working! –

## 移動ができれば接触判定！

ちょっとゲームらしくなってきます。

# Family Web Computer

## - Working! -

接触判定

本当は、細かく計算したいところです。


が、しかし、細かく計算しだすと
処理落ちしてきます… orz

# Family Web Computer

## – Working! –

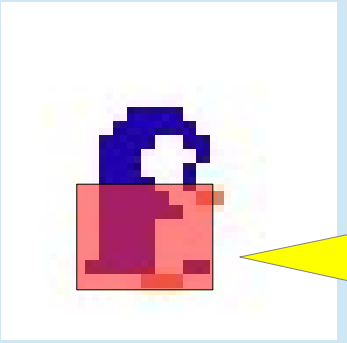
nes\08hitcheck01\main.c

```
// 総当たりで判定してみる #1
unsigned char x0 = p->x - 8;
unsigned char y0 = p->y - 8;
unsigned char x1 = p->x + 8;
unsigned char y1 = p->y + 8;
unsigned char j;

for (j = 0; j < 8; ++j) {
	Chara *e = &chara[j + 2];
	if (x0 <= e->x && e->x <= x1
		&& y0 <= e->y && e->y <= y1) {
		p->x = x;
		p->y = y;
		break;
	}
}
```

8体のキャラクタと
総当たりで
接触判定してます。

当たった場合は、
以前の位置に戻しています。



このあたりを
判定してます。

# Family Web Computer

## – Working! –

nes\08hitcheck02\main.c

```
// 総当たりで判定してみる #2
Chara *p = &chara[frame & 1];

基本は#1と同じですが、1Pと2Pを交互に判定してます。
```

nes\08hitcheck03\main.c

```
// 地形で判定してみる
ch = map_getmcp(x, y);
id = GET_SYMBOL_CHR(ch);
ch = GET_CHIP_ID(ch);

if (ch == MCP_FIELD && id <= MAP_SYMBOL_CHR(i)) {
    p->x += vx;
    p->y += vy;
}
```

マップ情報に
キャラクタの番号まで埋め込んで、
移動予定の場所に誰もいない場合は
移動可としてます。

判定は簡素ですが、精度は…

# Family Web Computer

– Working! –

基本的な処理が揃ったので、

## ルールの実装に行きます！

# Family Web Computer

## - Working! -

ルールの実装

- ゲームとしての「要」です。

- 本当は実装前に重要な点をまとめます。

- ルールは操作方法から画面構成、
  ゲームの進行など、多岐に渡ります。

# Family Web Computer

## - Working! -

今回は簡単なルールを決めて、
クリア条件とゲームの進行の実装に
焦点を絞ります。

# Family Web Computer

## - Working! -

ルール

１．固定画面
２．ボールを所定の場所に運べばクリア
３．壊せる壁、壊せない壁がある
４．当然、邪魔してくるキャラクタあり
５．時間制限あり

…ぐらいにしときます。

# Family Web Computer

## - Working! -

ルール

１．固定画面
２．ボールを所定の場所に運べばクリア
３．壊せる壁、壊せない壁がある
４．当然、邪魔してくるキャラクタあり
５．時間制限あり

…ぐらいにしときます。

# Family Web Computer

## - Working! -

ルールの実装

- ゲームの進行やキャラクタの管理に必要な物は「状態管理」です。

- 条件を満たすと別の状態に変化させ、最新の状態を画面に表示します。

- 複数の状態が管理できれば、よりリアルで複雑な表現が可能になります。

# Family Web Computer

## – Working! –

C:\home\nes\10rule01\src\main.c (抜粋)

```
    // ゲーム全体の制御
    switch (status) {
    default:
    case SCENE_STATUS_START:
        if (frame >= SCENE_START_WAIT) {
            status = SCENE_STATUS_PLAY;
            // 演奏開始
            nsd_play_bgm(BGM_PENGUIN);
        }
        break;

    case SCENE_STATUS_GOAL:
        if (frame >= SCENE_GOAL_WAIT) {
            status = SCENE_STATUS_CLEAR;
        }
        break;
```

ゲーム開始からゲーム中へ
時間で状態が変化します。

ゲーム中に変わった場合、
BGMの演奏を
開始しています。

ボールをゴールに運んだ場合、
一定時間後にシーンクリアの
状態へ変化します。

# Family Web Computer

## – Working! –

C:\home\nes\10rule01\src\main.c (抜粋)

```
// ボールの先を確認
x2 = p->x + forward_offset2[p->dir][0];
y2 = p->y + forward_offset2[p->dir][1];
mcp2 = map_getmcp(x2, y2);

if (mcp2 == MCP_GOAL) ball.status = STATUS_GOAL;
else if (mcp2 != MCP_FIELD) break;
else ball.status = STATUS_BUSY;

nsd_play_se(SE_MOVE);
map_move(x1, y1, p->dir);
ball.x = x1;
ball.y = y1;
ball.dir = p->dir;
```

ボールの移動先がゴールの場合
ボールの状態に
ゴールすることを
設定しています。

# Family Web Computer

## – Working! –

C:\home\nes\10rule01\src\main.c (抜粋)

```
	// ボールの移動
	if (ball.status) {
		nsd_play_se(SE_MOVE);
		map_move(ball.x, ball.y, ball.dir);

		// ゴール判定
		if (ball.status == STATUS_GOAL) {
			status = SCENE_STATUS_GOAL;
			frame = 0;
		}
		ball.status = 0;
	}
```

ボールの移動が完了した場合
ゲーム全体の状態を
変化させています。

# Family Web Computer

## - Working! -

nes\10rule01

プレイヤーがボールをAボタンで蹴って
旗の場所へ移動するとクリアになります。

nes\10rule02

NPCキャラの移動や攻撃を組み込んで
邪魔をするようになりました。

NPCキャラの制御も状態変化の管理です。
（今回のバージョンは
個別にタイマを持たせなかったのでイマイチですが。）

# Family Web Computer

– Working! –

最後に、

タイトルと
ゲーム本体を合体！

# Family Web Computer

## – Working! –

nes\11game\src\main.c (抜粋)

```
extern void __fastcall__ TitleMain(void);
extern void __fastcall__ GameMain(void);

// メイン処理
void NesMain()
{
	for (;;) {
		// タイトル呼び出し
		TitleMain();

		// ゲーム本体呼び出し
		GameMain();
	}
}

//NMI割り込み
void NMIProc(void)
{
```

以前のタイトルや
ゲーム本体のメイン関数は、
名前を変更して、
新しいメイン関数から呼び出します。

NMI割込みは共通なので、
main.cだけに残して、
タイトルとゲーム本体側は
削除します。

# Family Web Computer

## – Working! –


ゲームとしての
流れができたら
動かしてみよう！

参照：nes\11game

# Family Web Computer

## – Working! –


キャラクタ同士の判定がちょっと…

スライムに囲まれると…

フィールドの角にボールを蹴ると…

２Playerとの連携の仕組みが欲しい！

シーン数を増やしたい！

…などなど

問題はいっぱいありますが、

「カイゼン」していくと

おもしろいものに

仕上がっていきます！

# Family Web Computer

## - Resolved...? -

今時みたいなリアル方面のゲームは無理です。

が、単純明快なゲームが作れて、

Web上で公開できる環境が整ってきています。

# Family Web Computer

## - Resolved...? -

JavaScript製のエミュレータは実行速度が遅いのですが、

仮想デバイスを追加して

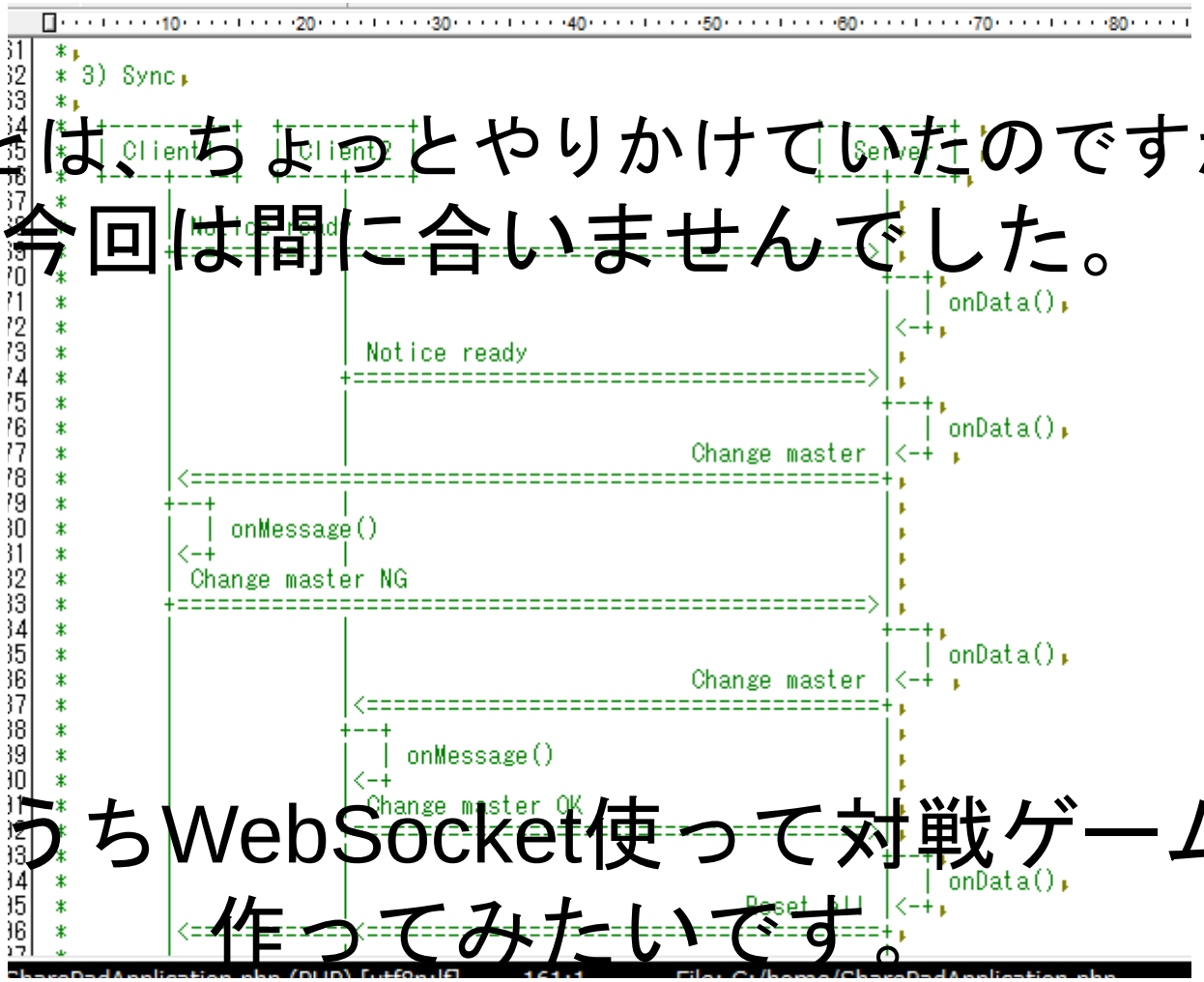
ネット越しにプレイできる環境や、

phpで動的に生成したデータと組み合わせるとか、

夢が広がります！

# Family Web Computer

## - Resolved...? -




ほんとは、ちょっとやりかけていたのですが…
今回は間に合いませんでした。

そのうちWebSocket使って対戦ゲームを
作ってみたいです。

# Family Web Computer

## - Resolved...? -

実機での動作環境も敷居が下がってきました！

テラネットワークシステムさんの「TNS-HFC」シリーズは、
NX(Nsf eXtend)アーキテクチャ仕様という
実機動作プラットホームです！
<http://www2s.biglobe.ne.jp/~tns/>

（今回はやっていませんが…）
SDカードに保存したプログラムを実機で動作させることも！

# Family Web Computer

## - Question and answer -

質疑応答

- 質問ある方、ドウゾ。

# Family Web Computer

Fin.

Thank you for your attention.