

## 1 Wersje konsol

Pierwotnym wydaniem konsoli był Family Computer z 1983 roku, w skróconej formie nazywany "Famicom". Famicom oficjalnie nigdy nie został wydany poza Japonią. Nintendo przygotowując się do wydania międzynarodowego swojej konsoli do grania podjęło decyzję o przeprojektowaniu wyglądu sprzętu w celu dostosowania się do zachodnich odbiorców. Wynikiem tych prac był Nintendo Entertainment System, wydany w 1985 roku w USA, rok później na terenie Europy oraz w 1987 roku w pozostałych lokalizacjach w Europie oraz Australii[?].



Rysunek 1: Family Computer wraz z kontrolerami[1]

Rysunek 2: Nintendo Entertainment System NES-001 wraz z kontrolerem[2]

Odświeżone wersje konsol zostały wydane w 1993 roku. Nintendo znacznie zunifikowało wygląd zewnętrznego konsoli, zachowując kompatybilność kartridzy dla poszczególnych regionów.



Rysunek 3: "New Famicom" wraz z kontrolerem[4]  
Rysunek 4: Nintendo Entertainment System NES-101 wraz z kontrolerem[3]

Obok oryginalnych sprzętów od Nintendo równocześnie egzystowały konsole podrabiane na masową skalę, w większości były to produkty kompatybilne z japońskim Famicomem. Dokładna historia nie jest znana; zebrane informacje pozwalają na stwierdzenie, iż pierwsze podróbki zaczęły pojawiać się pod koniec lat osiemdziesiątych oraz zostały wyprodukowane na Tajwanie. W latach dziewięćdziesiątych następowała miniutaryzacja sprzętów oraz minimalizacja kosztów, co spowodowało stopniowo powiększane przenoszenie produkcji do Chin oraz drastyczny spadek jakości wykonania konsol. Kreatywność "piratów" doprowadziła do powstania sprzętów będących czymś innym niż zwyczajna konsola:

- Konsole z dwoma gniazdami na kartridże - jedno dla gier z Famicoma, drugie dla gier z NES,
- Konsole z dwoma gniazdami na kartridże - jedno standardowo na zewnątrz obudowy, drugie wewnątrz obudowy z fabrycznie zamontowaną grą. Dzięki temu rozwiązaniu gdy w zewnętrznym gnieździe nie było zamontowanego kartridża to uruchamiał się kartridż wbudowany w konsolę,
- Konsola-klawiatura, udająca komputer - zawsze dołączano kartridż z "programami multimedialnymi" które korzystały z klawiatury. Była tu pewna inspiracja oficjalnym Family Basic,
- Przenośna konsola z wbudowanym wyświetlaczem,
- Sprzęty "Plug 'n' Play" o wymyślnych kształtach, często były one pozbawione gniazda na kartridże, zatem wbudowane gry musiały wystarczyć.



Rysunek 5: Generation NEX - klon z dwoma gniazdami na kartridże, górne jest dla gier Famicom, boczne dla gier NES[5]



Rysunek 6: Daryar DY-400-656 - konsoła z wbudowaną grą 400 in 1[6]



Rysunek 7: GLK-2004 - popularna konsola-klawiatura[7]



Rysunek 8: Game Axe Color - jeden z pierwszych handheldów (2000 rok) odwierzający gry z Famicoma[8]

Klony konsoli Nintendo są produkowane po dzień dzisiejszy, tyle że częściowo wyszły ze szarej strefy - "Wielkie N" nie ma podstaw prawnych do sądzenia się z firmami gdyż wygasły patenty (TODO: sprawdzić dokładnie jak to wygląda, lecę z głowy z tego co kiedyś czytałem). Konstrukcje są również uwspółczesniane poprzez zastosowanie gniazda HDMI czy obsługę gier z kilku platform jednocześnie.



Rysunek 9: RetroUSB AVS - konsola wysokiej jakości do której stworzenia użyto podzespołów z oryginalnych NES[9]



Rysunek 10: Hyperkin RetroN 5 - sprzęt obsługujący ogromną ilość platform jednocześnie, również kartridże do NES i Famicoma[10]



Rysunek 11: NES 620 Games - niskobudżetowy, współczesny klon konsoli NES z wbudowanymi grami, bez zewnętrznego gniazda na kartridże[11]

Nieoficjalne sprzęty zyskały popularność w biedniejszych krajach w których Nintendo nie dystrybuowało swoich produktów. "Marki" konsol różniły

się między poszczególnymi państwami, niektóre kraje miały nawet "oficjalnych dystrybutorów tychże podróbek. Miały również miejsce sytuacje, gdzie jeden i ten sam model / typ konsoli był sprzedawany pod kilkoma nazwami.[?]



Rysunek 12: Micro Genius IQ-502 - wersja konsoli sygnowana marką producenta[12]



Rysunek 13: Pegasus IQ-502 - wersja znaczona na rynek rosyjski, Dendy to Micro Genius IQ-502 przeznaczona na rynek polski, Pegasus to marka firmy Bobmark.[13]

Wszystkie trzy powyższe sprzęty to w rzeczywistości jeden projekt firmy Micro Genius, różniący się logotypami na konsoli oraz padach.

## 2 Specyfikacja konsoli

Procesor (CPU): Ricoh 2A03 (NTSC) / Ricoh 2A07 (PAL), ośmiorozmiotowy mikroprocesor będący modyfikacją MOS 6502. Różnice 2A03 / 2A07 względem produktu MOS Technology są następujące[?]:

- Deaktywowany tryb dziesiętny (BCD),
- Wbudowany generator dźwiękowy,
- Obsługa kontrolerów poprzez porty \$4016 oraz \$4017,
- DMA.

Mimo iż jest to 8-bitowy procesor, można zaadresować 64 kilobajty pamięci. To, jak podzielona jest ta przestrzeń adresowa przedstawia poniższa tabela (podane wartości są zapisane w systemie szesnastkowym):

| Adresy          | Rozmiar | Zastosowanie   |
|-----------------|---------|--|
| \$0000 - \$07FF | \$0800  | Wewnętrzna pamięć RAM (dwa kilobajty)                        |
| \$0800 - \$1FFF | \$0800  | Duble danych spod adresów \$0000 - \$07FF (co dwa kilobajty) |
| \$2000 - \$2007 | \$0008  | Rejestry kontrolujące PPU                                    |
| \$2008 - \$3FFF | \$1FF8  | Duble danych spod adresów \$2000 - \$2007 (co osiem bajtów)  |
| \$4000 - \$4017 | \$0018  | Rejestry kontrolujące APU oraz rejesty wejścia / wyjścia     |
| \$4018 - \$401F | \$0008  | Dezaktywowany tryb samotestowania                            |
| \$4020 - \$FFFF | \$BFE0  | Przestrzeń do dowolnej dyspozycji dla kartridża              |

Adresy \$4020 - \$FFFF są zaadresowane zgodnie z możliwościami kartridża (patrz Mapper) [?]

Układ graficzny (PPU): Ricoh 2C02, ośmiorozmiotowy układ opracowany przez Nintendo specjalnie dla tej konsoli. Mimo iż jest to 8-bitowy procesor, można zaadresować 16 kilobajtów pamięci. To, jak podzielona jest ta przestrzeń adresowa przedstawia poniższa tabela (podane wartości są zapisane w systemie szesnastkowym):

| Adresy          | Rozmiar | Zastosowanie                              |
|-----------------|---------|---|
| \$0000 - \$0FFF | \$1000  | Pattern Table 0                           |
| \$1000 - \$1FFF | \$1000  | Pattern Table 1                           |
| \$2000 - \$23FF | \$0400  | Nametable 0                               |
| \$2400 - \$27FF | \$0400  | Nametable 1                               |
| \$2800 - \$2BFF | \$0400  | Nametable 2                               |
| \$2C00 - \$2FFF | \$0400  | Nametable 3                               |
| \$3000 - \$3EFF | \$0F00  | Dubel danych spod adresów \$2000 - \$2EFF |
| \$3F00 - \$3F1F | \$0020  | Paleta barw                               |
| \$3F20 - \$3FFF | \$00E0  | Duble palety barw (co 32 bajty)           |

Dostęp do wyżej wymienionych adresów odbywa się poprzez adresy CPU: \$2006 i \$2007. Gniazdo kartridzy: 60-pin (Famicom), 72-pin (NES)

## 2.1 Rejestry kontrolujące PPU

Adresy \$2000 - \$2007 oraz \$4014 w CPU mają specjalne znaczenie - poprzez nie programista / program może wpływać na działanie układu graficznego konsoli. Zwyczajowe nazwy tych adresów przedstawia poniższa tabela [?]:

| Adres hex | Nazwa     |
|-----------|-----------|
| \$2000    | PPUCTRL   |
| \$2001    | PPUMASK   |
| \$2002    | PPUSTATUS |
| \$2003    | OAMADDR   |
| \$2004    | OAMDATA   |
| \$2005    | PPUSCROLL |
| \$2006    | PPUADDR   |
| \$2007    | PPUDATA   |
| \$4014    | OAMDMA    |

Nazw tych powszechnie używa się w kodzie programów zamiast adresów bezpośrednich oraz w dyskusjach na temat programowania konsol NES / Famicom. Taką zasadę stosuję również w tym dokumencie.

## 2.2 Rejestry kontrolujące APU

Adresy \$4000 - \$4013, \$4015 oraz \$4017 w CPU mają specjalne znaczenie - poprzez nie programista / program może wpływać na działanie układu

dźwiękowego konsoli. Zwyczajowe nazwy tych adresów przedstawia poniższa tabela [?]:

| Adres hex | Nazwa      |
|-----------|------------|
| \$4000    | PL1_VOL    |
| \$4001    | PL1_SWEEP  |
| \$4002    | PL1_LO     |
| \$4003    | PL1_HI     |
| \$4004    | PL2_VOL    |
| \$4005    | PL2_SWEEP  |
| \$4006    | PL2_LO     |
| \$4007    | PL2_HI     |
| \$4008    | TRI_LINEAR |
| \$400A    | TRI_LO     |
| \$400B    | TRI_HI     |
| \$400C    | NOISE_VOL  |
| \$400E    | NOISE_LO   |
| \$400F    | NOISE_HI   |
| \$400E    | DMC_FREQ   |
| \$400F    | DMC_RAW    |
| \$4010    | DMC_START  |
| \$4011    | DMC_LEN    |
| \$4012    | SND_CHN    |

Nazwy te zostały zaczerpnięte z biblioteki audio do NES / Famicom - FamiTone2 [?]. Używam ich również w kodzie źródłowym programu oraz w tym dokumencie. (TODO: omówienie każdego rejestru, jeszcze nie kodowałem tego to nie znam szczegółów programowych)

### 3 Projekt

W tej sekcji omówię krok po kroku jak powstawał mój własny projekt programu na platformę NES. Zacznę od pojedynczych, prostych czynności aż po pewnego rodzaju pełnoprawny szkielet takiego projektu zgodny z dobrymi praktykami programowania NES. Następnie zostaną omówione krytyczne fragmenty kodu z punktu widzenia całego projektu oraz własne rozwiązania często spotykanych zagadnień.

## 3.0 Przygotowania do projektu

Dotychczas miałem minimalne doświadczenie z programowaniem pod NES, przerobiłem popularny kurs traktujący o tym - Nerdy Nights [?], niestety z miernym zrozumieniem nie potrafiąc zrobić czegoś swojego. Problem był też w dialekcie asemblera użytego w tym kursie (NESASM3 [?]) o którym niewiele dyskutuje się w sieci a nie widziałem sensu w nauce kolejnej odmiany asemblera jak i tak za wiele nie umiem zaprogramować. Tym razem miałem zamiar zmodyfikować podejście do tematu. Pierwsza sprawa to do głębnie poznać możliwości używanych narzędzi, w bardzo zaawansowanym stopniu opanować narzędzia wbudowane w emulator po to, aby umieć łatwo namierzać błędy oraz móc eksperymentować - robiąc modyfikacje w kodzie a następnie obserwować co się zmieniło, jak to działa. Kolejna rzecz to użyć wiedzę programistyczną zdobytą od ostatniej próby nauki, ustanowić pewnego rodzaju dobre wzorce pisania kodu, podzielić projekt na pliki - poprzednio wszystko było w jednym, wielkim pliku źródłowym.

Następna sprawa - dokumentacje, źródła wiedzy i tak dalej. Najobszerniejszym źródłem wiedzy na temat NES jest strona nesdev - wiki [15] oraz forum [?]. Mocno techniczne źródła wiedzy, warto nadmienić, nie sposób zrozumieć wszystko tak od razu ale wtedy zawsze miał ratować mnie emulator z narzędziami metodą prób i błędów. Podobna uwaga tyczy się dialekty asemblera CA65 którego miałem zamiar używać, dokumentacja projektu jest minimalna i ogólna. O tej decyzji zadecydowała jego popularność, możliwe najszysze uzyskania pomocy w razie problemów.

Generalnie miało być możliwie najprofesjonalniej.

### 3.1 Hello World

Wiedząc że jest perfekcyjnie opracowany kawałek kodu inicjujący konsolę do pełni poprawnego działania po prostu go skopiowałem, zbudowałem plik ROM a następnie analizowałem krok po kroku jak on działa w emulatorze i co tam się dzieje pod "maską" konsoli. Ten kod wygląda następująco:

```
reset:  
    sei  
    cld  
    ldx #$40  
    stx $4017  
    ldx #$ff  
    txs  
    inx  
    stx $2000
```

```

    stx $2001
    stx $4010

    bit $2002

@vblankwait1:
    bit $2002
    bpl @vblankwait1

    txa
@clrmem:
    sta $000,x
    sta $100,x
    sta $200,x
    sta $300,x
    sta $400,x
    sta $500,x
    sta $600,x
    sta $700,x
    inx
    bne @clrmem

@vblankwait2:
    bit $2002
    bpl @vblankwait2

```

Miałem już wiedzę na temat najczęściej używanych mnemoników typu (`lda` / `ldx`, `sta` / `stx`, `inx`, `txs`, `txa`) gdyż były proste do zrozumienia, idea etykiet również była jasna, orientacja na temat instrukcji skoków warunkowych też była ale szczegółów nie znałem. Co do fragmentów kodu to część pod etykietą `@clrmem` była jasna - zerowanie całego powierzchni RAM (adresy \$0000 - \$07FF). Pętla kończyła się gdy rejestr X się "przekręcił" na zero. Pętle z etykietami `vblankwait` były po to, żeby konsola mogła poprawnie wyświetlać grafikę. Mnemonik `cld` wyłączał tryb dziesiętny którego w sumie i tak nie ma w tej wersji procesora.

W celu zrozumienia całości w stu procentach zacząłem wertować internet oraz debugować ten kod. Opoznałem `sei` to instrukcja wyłączająca odnotowywanie przez CPU przerwań niemaskowalnych. To znaczy że procesor nie będzie reagował na przerwania, dzięki czemu nie będzie zakłócał wykonania naszego kodu inicjującego. Przerwania nadal nadal się "wewnętrznie", cyklicznie występują! Instrukcja `cld` ma sens gdyż debuggery ogólnie działające na kodzie

napisanym pod MOS 6502 (nie konkretnie pod NES) opierają się na tym, iż ta konkretna flaga jest ustawiona. Wartość #\$40 to właściwie ustawienie bitu numer sześć. Ta wartość jest wysyłana pod port \$4017 CPU w celu wyłączenia przerwań generowanych przez dodatkowe podzespoły rozszerzające możliwości kartridża. Następnie odbywa się inicjacja stosu - wartość \$FF jest ładowana do rejestru X, mnemonik *tcs* kopiuje tę wartość z rejestru X do rejestru wskaźnika stosu. Proszę zauważać że to dolna część wskaźnika stosu - górną jest ustawiona na wartość \$01. Zatem w tym przypadku góra stosu to adres \$01FF. Trzy kolejne zapisy wartości zero pod adresy \$2000, \$2001 oraz \$4010 to wyłączenie wszelkich flag którymi te trzy rejestrów zarządzają. Najważniejsze z nich to dezaktywacja NMI - znaczy to tyle że przestanie wykonywać się to przerwanie (rejestr \$2000), wyłączenie renderowania ekranu (\$2001), wyłączenie przerwania DMC którego dedykowanym zastosowaniem jest prawidłowe odgrywanie digitalizowanych próbek dźwiękowych (\$4010).

Z pętlami *vblankwait* sprawa okazuje się bardziej skomplikowana. Trzeba odczekać dwie klatki obrazu aby układ graficzny ustabilizował się oraz zaczął pracować w pełni prawidłowo PPU (TODO: dokładne wskazanie omówienia co tam się dzieje). Mamy wyłączone NMI ale jest jeszcze jedna metoda na odliczenie klatek obrazu. PPU zawsze sygnalizuje moment w którym zakończył rysowanie klatki obrazu (rozpoczęcie wygaszanie pionowe obrazu) poprzez ustawienie najwyższego bitu w rejestrze flag \$2002. A ten można sprawdzić mnemonikiem *bit*. Odpowiednio *bit* ma specyficzne działanie: wykonuje operację bitową "akumulator AND operand instrukcji *bit*", następnie "zapomina" wynik za to ustawia flagę Z (flaga zero) jeśli wynik operacji AND był równy zero. Oprócz tego zawsze kopiuje bit szósty operandu do flagi V (flaga przepełnienia), a siódmy do flagi N (flaga wartości ujemnej). W akumulatorze mamy w tej chwili wartość zero - w jego przypadku akurat mamy gwarancję że po uruchomieniu konsoli / resecie zawsze będzie to wartość zero a wcześniej w kodzie nigdzie nie manipulowaliśmy tym rejestrów. Przykład powinien rozjaśnić:

W rejestrze A mamy wartość zero.

Chcemy wykonać operację \textit{bit} na wartości #\$80 (%10000000 binarnie).  
Zatem wynik operacji:

bit     #\$80

będzie następujący:

%00000000

AND %10000000

%00000000

a w rezultacie:

1. flaga Z zostanie ustawiona,
2. flaga V zostanie wyzerowana,
3. flaga N zostanie ustawiona.

Powyższe omówienie instrukcji *bit* to sytuacja tożsama z tą, gdy właśnie PPU dało znak że skończyło rysować klatkę. Instrukcja *tpl* to jeden ze skoków warunkowych - dla *tpl* skok do etykiety podanej jako operand ma miejsce, gdy flaga wartości ujemnej (V) jest wyzerowana. Zatem program będzie "kręcił się w miejscu" dopóki flaga jest ujemna. Przestanie to robić gdy opkod *bit* ustawi flagę N, a to się stanie gdy PPU skończy rysować klatkę oraz ustawi najwyższy bit w rejestrze \$2002. W ten sposób należy rozumieć zaprezentowany kod. Działa to poprawnie także dlatego, iż po odczycie rejestru \$2002 najwyższy bit zostanie wyczyszczony, zatem nie ma mowy o sytuacji gdzie PPU ustawiło tę flagę jakimś cudem wcześniej a teraz jest w trakcie rysowania kolejnej klatki.

Instrukcja *bne* zamykająca pętlę czyszczącą RAM polega na fladze zero (Z). W przypadku tego skoku warunkowego skok ma miejsce gdy ta flaga jest wyzerowana / nieustawiona. Z tego wynika że wykonanie zostanie przepuszczone za tę instrukcję gdy flaga Z ustawi się, a ustawi się w momencie inkrementacji przekręcając rejestr X (z wartości *FFna00*).

Pętla *vblankwait2* ma działanie analogiczne do oczekiwania na pierwsze rozpoczęcie wygaszania obrazu.

Ten kod już wymagał bardzo dużej wiedzy żeby zrozumieć co jak jest wykonane oraz czemu tak, a to nadal nie uruchomi się nawet na emulatorze. W sekcji kodu potrzeba dodatkowych linii treści a do tego warto jakoś pokazać iż wszystko zostało zainicjowane poprawnie:

```

lda #01100000
sta $2001

lda #$16
sta $0000
Stop:
jmp Stop

.segment "VECTORS"
.word _INT_Reset

```

- 3.2 Drugi ekran z tłem, przewijanie ekranu**
- 3.3 Podprogramy / procedury**
- 3.4 Liczenie czasu**
- 3.5 Animacja grafiki**
- 3.6 Podział projektu na pliki**
- 3.7 Obsługa kontrolera**
- 3.8 Implementacja trybów programu**

## **4 Informacje o platformie**

- 4.1 Zero Page**
- 4.2 NMI**
- 4.3 PPU**
- 4.4 Mapper**
- 4.5 Pattern Table**
- 4.6 Nametable**
- 4.7 Paleta barw**

## **5 Narzędzia**

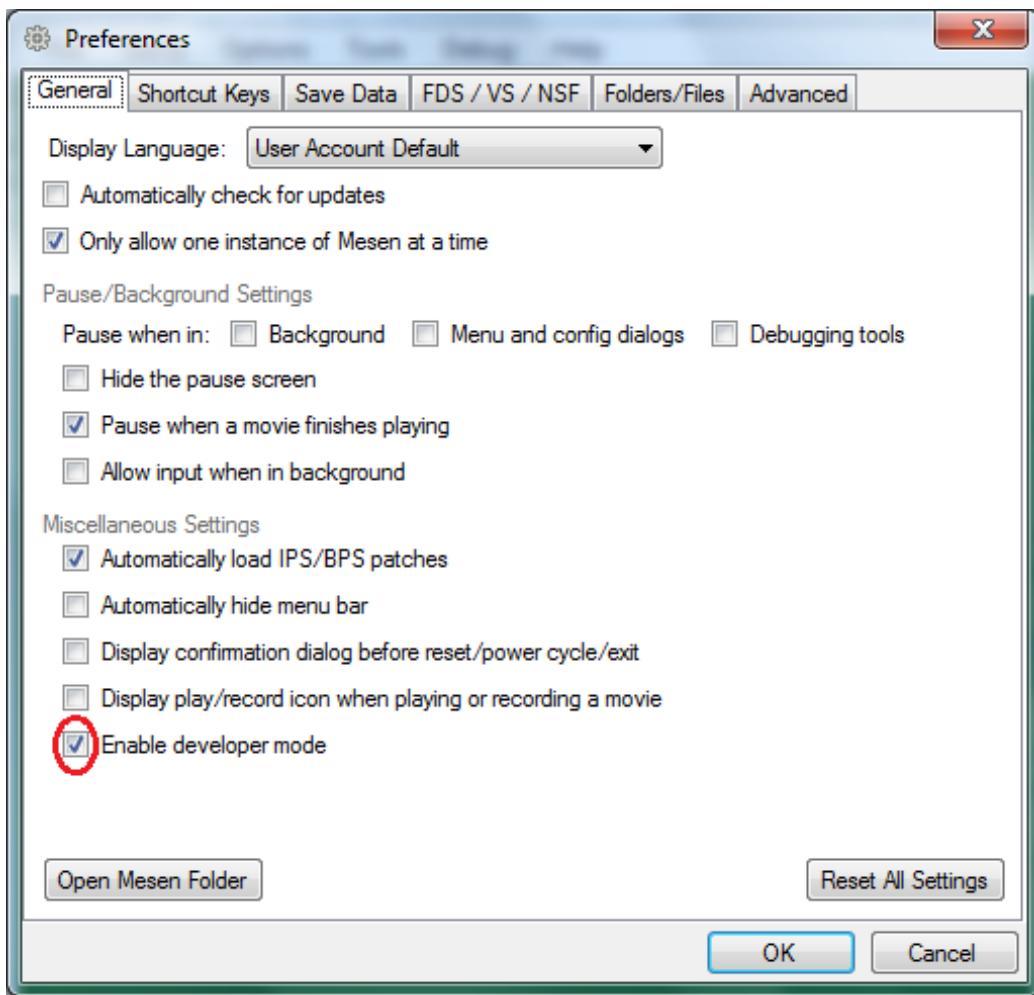
### **5.1 CA65**

Strona projektu: <https://cc65.github.io/> CA65 jest częścią większego pakietu zwanego CC65. Jest on zbiorem narzędzi do komplikacji kodu napisanego w języku C na sprzętę z procesorem MOS 6502. Za to CA65 jest asemblerem pod ten sam procesor. W przypadku NES język C stanowi problem wydajnościowy, gdyż plik wynikowy który powstaje po komplikacji kodu w C jest znacznie wolniejszy niż odpowiednik asemblerowy; warto pamiętać iż ta konsola jest wielokrotnie mniej wydajna niż współczesne sprzęty. Stąd moja decyzja o pisaniu kodu tylko i wyłącznie w asemblerze.

(UWAGI: zestaw kompilator plus linker zasługuje na większe wyróżnienie od niżej opisanych programów pomocniczych do grafiki czy muzyki; do przemyślenia miejsce w dokumencie)

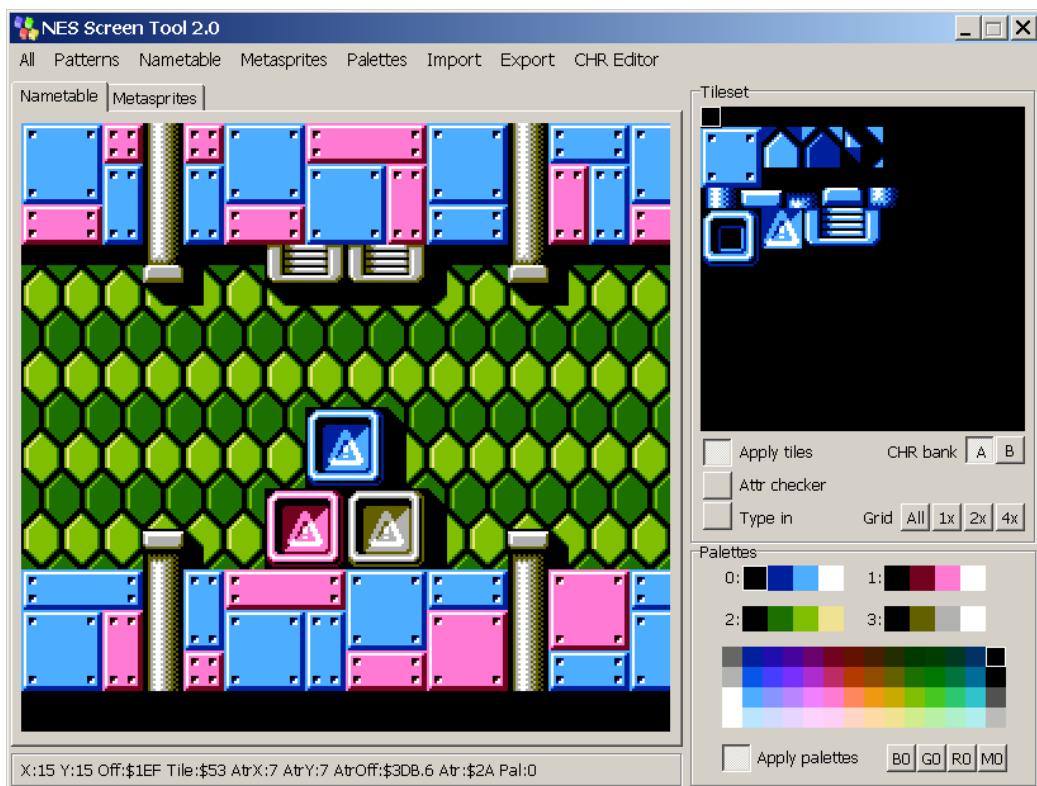
## 5.2 Mesen

Emulator znacząco wspiera proces tworzenia oprogramowania pod NES, gdyż zawiera wiele narzędzi pozwalających na zbadanie zachowania naszego programu. Aby mieć dostęp do narzędzi deweloperskich, musimy w menu *Options -> Preferences* zaznaczyć checkbox "Enable developer mode". Od tej pory w menu jest widoczna nowa pozycja - *debug*.



Rysunek 15: Przełącznik opcji deweloperskich w emulatorze Mesen.

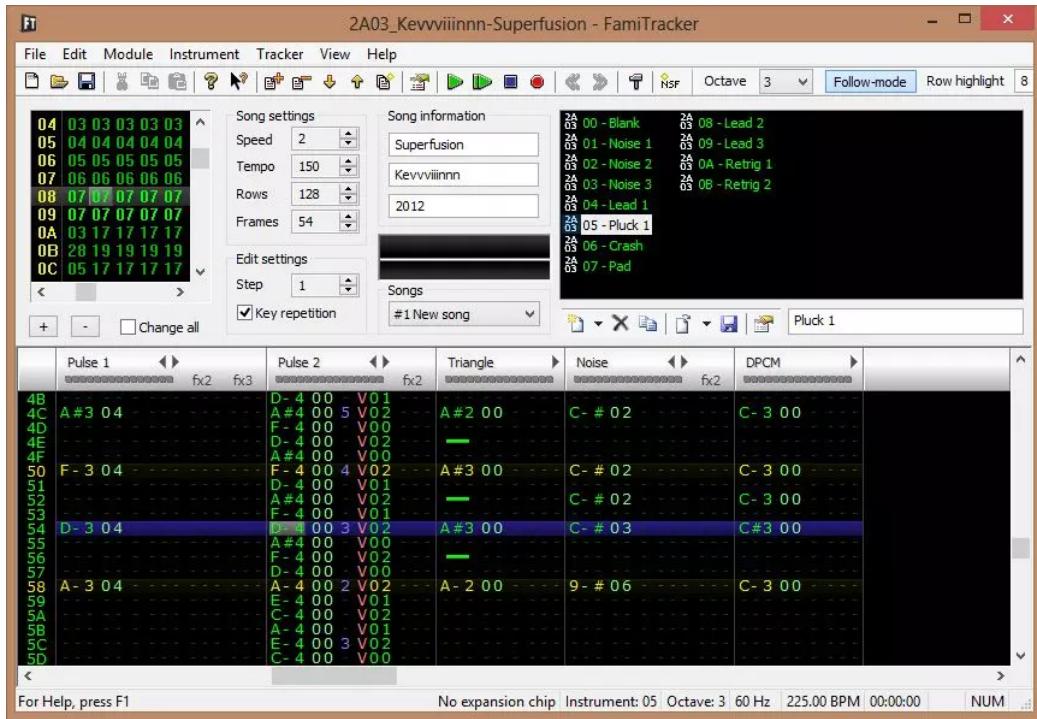
### 5.3 NES Screen Tool



Rysunek 16: Ekran główny aplikacji [?]

NES Screen Tool to program wspomagający tworzenie grafiki zgodnej z platformą NES / Famicom. Aplikacja pozwala na szczegółowe tworzenie grafik per pixel jak i bardziej ogólne działania jak eksport przygotowanych danych do formatu zrozumiałego przez konsolę, gotowego do zastosowania w kodzie programu. Strona domowa [?].

## 5.4 FamiTracker



Rysunek 17: Ekran główny aplikacji [?]

FamiTracker to program pozwalający na tworzenie muzyki w stu procentach kompatybilnej z platformami NES / Famicom. Aplikacja należy do rodziny trackerów - programów komputerowych w których komponuje się muzykę wykorzystując połączenie zapisu nutowego oraz poleceń sterujących efektami [?]. Strona domowa [?].

## Literatura

- [1] <https://upload.wikimedia.org/wikipedia/commons/0/06/Nintendo-Famicom-Console-Set-FL.jpg>
- [2] <https://upload.wikimedia.org/wikipedia/commons/8/82/NES-Console-Set.jpg>
- [3] [https://upload.wikimedia.org/wikipedia/commons/f/f1/New\\_Famicom.jpg](https://upload.wikimedia.org/wikipedia/commons/f/f1/New_Famicom.jpg)
- [4] <https://upload.wikimedia.org/wikipedia/commons/e/e0/NES-101-Console-Set.jpg>

- [5] <https://levelupvideogames.com/used-messiah-generation-next-468309594161/>
- [6] <http://forum.pegasus-gry.com/index.php?topic=2315.0>
- [7] <https://arhn.eu/2018/04/historia-pegasusa-z-klawiatura-glk-2004-et-al/>
- [8] <https://www.ign.com/articles/2000/06/10/game-axe-color>
- [9] <https://upload.wikimedia.org/wikipedia/commons/7/7a/RetroUSB-AVS-Console-wController-FL.jpg>
- [10] [https://cdn10.bigcommerce.com/s-2l8ru96d/products/223/images/1135/ret5bk\\_54484.1567](https://cdn10.bigcommerce.com/s-2l8ru96d/products/223/images/1135/ret5bk_54484.1567)
- [11] <https://www.amazon.com/Controllers-Children-Birthday-Childhood-Memories/dp/B083RBYRYN>
- [12] <http://www.retrogamingmuseum.com/the-collection/micro-genius-nes-clone>
- [13] <https://archiwum.allegro.pl/oferta/konsola-pegasus-iq-502-pady-pudelko-plomba-i7530854084.html>
- [14] <https://youla.ru/moskva/hobbi-razvlecheniya/konsoli-igry/dendy-classic-2-novaia-5b6c9ae1cf689a85567f44a6>
- [15] [http://wiki.nesdev.com/w/index.php/Nesdev\\_Wiki](http://wiki.nesdev.com/w/index.php/Nesdev_Wiki)
- [16] <http://forums.nesdev.com/>
- [17] [https://en.wikipedia.org/wiki/Nintendo\\_Entertainment\\_System](https://en.wikipedia.org/wiki/Nintendo_Entertainment_System)
- [18] [https://en.wikipedia.org/wiki/Nintendo\\_Entertainment\\_System\\_hardware\\_clone](https://en.wikipedia.org/wiki/Nintendo_Entertainment_System_hardware_clone)
- [19] Patrick Diskin: *Nintendo Entertainment System Documentation, 2.1*  
2A03 Overview, <http://www.nesdev.com/NESDoc.pdf>
- [20] [http://wiki.nesdev.com/w/index.php/CPU\\_memory\\_map](http://wiki.nesdev.com/w/index.php/CPU_memory_map)
- [21] [http://wiki.nesdev.com/w/index.php/PPU\\_registers#Summary](http://wiki.nesdev.com/w/index.php/PPU_registers#Summary)
- [22] <http://wiki.nesdev.com/w/index.php/APU#Specification>
- [23] <http://forums.nesdev.com/viewtopic.php?t=7329>
- [24] <https://shiru.untergrund.net/software.shtml>

- [25] <https://shiru.untergrund.net/pic/nesst.png>
- [26] <http://famitracker.com/>
- [27] <https://blog.uptodown.com/wp-content/uploads/Famitracker-pianola.jpg.webp>
- [28] [https://pl.wikipedia.org/wiki/Tracker\\_\(oprogramowanie\\_muzyczne\)](https://pl.wikipedia.org/wiki/Tracker_(oprogramowanie_muzyczne))
- [29] [http://wiki.nesdev.com/w/index.php/Init\\_code](http://wiki.nesdev.com/w/index.php/Init_code)
- [30] <http://wiki.nesdev.com/w/index.php/File:Savtool-swatches.png>
- [31] <https://nerdy-nights.nes.science/>
- [32] <http://www.nespowerspak.com/nesasm/>