

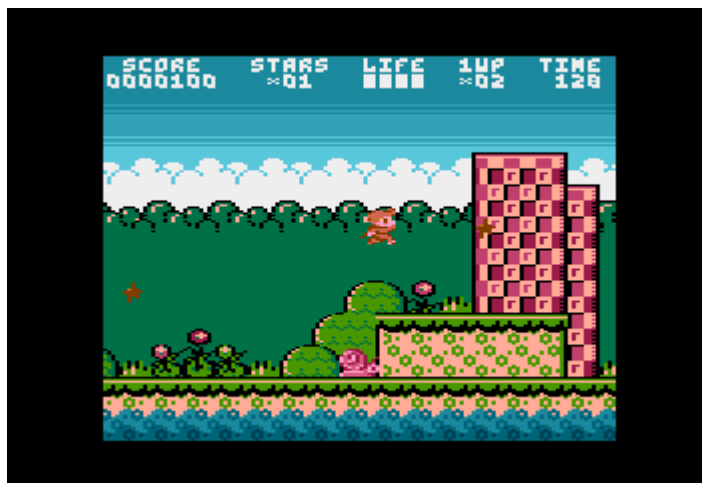
Shanti77 Sprite Multiplexer

Wstęp

Autorem poniżej opisywanego multipleksera jest Janusz „Shanti77” Chabowski. Silnik w prezentowanej wersji pozwala wyświetlać do 16 animowanych duszków sprzętowych o rozmiarze 8x16 pikseli w 3 kolorach z możliwością zmiany koloru i kształtu. Maksymalna dostępna liczba kształtów ograniczona jest do 128.

Przykładem wykorzystania takiej koncepcji multipleksera są gry „The Last Squadron”, „Crownland”.

„Crownland” jako pierwszy współcześnie zaczął wykorzystywać taki silnik w grze, jednak Piotr „Probe” Wiśniewski nie upublicznił jego źródeł.



<https://atarionline.pl/v01/index.php?ct=katalog&sub=C&tg=Crownland#Crownland>

W demie „Rigoletto” z końca lat 80-ych także można dopatrzyć się realizacji podobnego pomysłu na zwiększenie liczby wyświetlanych obiektów.



<https://atarionline.pl/demoscena/R/Rigoletto.xex>

Układ GTIA w Atari XE/XL pozwala na wyświetlenie w linii maksymalnie 4-ech duchów i 4-ech pocisków. Jeśli połączyć duchy w pary uzyskamy dwa 3-kolorowe duchy i dwa 3-kolorowe pociski.

Zwiększenie liczby wyświetlanych obiektów polega na naprzemiennym wyświetleniu par duchów w kolejnych ramkach obrazu, jest to widoczne w postaci migania.

W konsoli Nintendo NES występuje podobny efekt kiedy liczba wyświetlanych obiektów w linii przekracza dopuszczalny limit.

Obsługa silnika

Cały kod silnika, razem z tablicami mieści się w 4KB pamięci. Kod jest w postaci relokowalnej, do prawidłowego działania wymagane jest aby podczas linkowania ustawić dla niego adres na początek 2KB bloku pamięci.

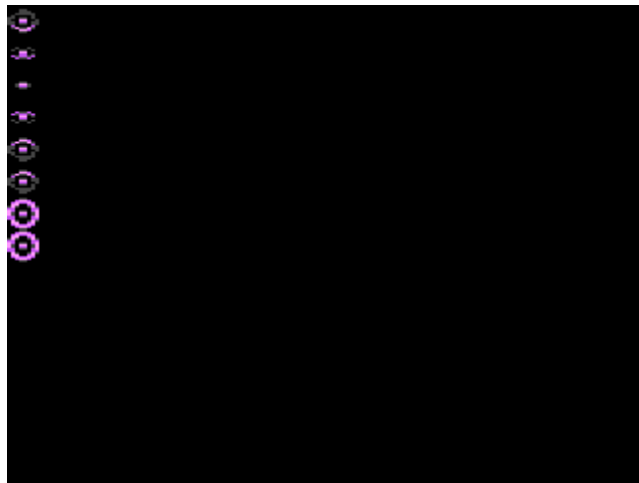
```
.align $0800  
.link 'engine.obx'
```

Kształty dla obiektów można przygotować załączonym programem 'convert\mic2shp.exe', który zamienia wybrane kolumny obrazka MIC na odpowiedni kod assemblera z kształtem duszków.

Program generuje dane zawsze dla ośmiu klatek: 'shp0', 'shp1', 'shp2', 'shp3', 'shp4', 'shp5', 'shp6', 'shp7', 'shp8'. W dwóch wariantach '_01' czyli duch 0+1, oraz '_23' czyli duchy 2+3.

```
.local      shp0  
_01  
    lda #0  
    sta sprites+$500+0,x  
    sta sprites+$500+1,x  
    sta sprites+$400+14,x  
    ...  
    sta sprites+$400+6,x  
    sta sprites+$400+7,x  
    lda #220  
    sta sprites+$400+8,x  
    sta sprites+$400+9,x  
  
    jmp multi.ret01  
_23  
    lda #0  
    sta sprites+$700+0,x  
    sta sprites+$700+1,x  
    sta sprites+$600+14,x  
    ...  
    sta sprites+$600+6,x  
    sta sprites+$600+7,x  
    lda #220  
    sta sprites+$600+8,x  
    sta sprites+$600+9,x  
  
    jmp multi.ret23  
.endl
```

Dane kształtów w pliku MIC (40 bajtów na linię) muszą być ułożone w postaci kolumn o szerokości 8 pikseli i wysokości maksymalnie 128 pikseli (8 klatek po 16 pikseli wysokości).



Program wywołujemy z dwoma parametrami: nazwa pliku MIC, numer kolumny.

```
mic2shp.exe plik.mic 0 >shape0.asm
```

Wynik zapisywany jest na konsoli, aby go zapisać do pliku musimy dokonać przekierowania '>fileout'.

Przykład zapisania 6-u kolumn do kolejnych plików 'shape0.asm' .. 'shape5.asm'

```
mic2shp.exe plik.mic 0 >shape0.asm  
mic2shp.exe plik.mic 1 >shape1.asm  
mic2shp.exe plik.mic 2 >shape2.asm  
mic2shp.exe plik.mic 3 >shape3.asm  
mic2shp.exe plik.mic 4 >shape4.asm  
mic2shp.exe plik.mic 5 >shape5.asm
```



Dodawanie kształtów do silnika realizujemy przez wywołanie makra 'addShape' z dwoma parametrami: numer kształtu, adres danych.

```
addShape 0, shp0
addShape 1, shp1
addShape 2, shp2
addShape 3, shp3
addShape 4, shp4
addShape 5, shp5
addShape 6, shp6
addShape 7, shp7
```

Dodawanie obiektów do silnika realizujemy przez wywołanie makra 'addSprite' z ośmioma parametrami: numer obiektu, color0, color1, numer kształtu, pozycja pozioma, pozycja pionowa, prędkość animacji, maska dla licznika klatek animacji

```
addSprite 0 $64 $18 12 40 32 7 255
addSprite 1 $24 $e8 0 56 40 3 255
addSprite 2 $04 $38 3 72 48 1 255
addSprite 3 $04 $78 15 84 56 3 255
addSprite 4 $a4 $a8 0 96 64 3 255
addSprite 5 $34 $38 18 104 72 1 255
addSprite 6 $04 $68 2 112 82 3 255
addSprite 7 $04 $08 0 124 94 3 255
addSprite 8 $4c $f8 11 134 108 3 255
```

maska dla licznika klatek animacji = %11111111 (DEC 255) oznacza tylko 1 klatkę animacji
maska dla licznika klatek animacji = %11111000 (DEC 31) oznacza 8 klatek animacji

Wywołanie silnika, inicjalizacja

```
multi.init_engine #vbl
```

Jako parametr przekazujemy adres naszego programu przerwania VBL, które musi kończyć się rozkazem 'PLR' zwracającym stan rejestrów A-X-Y oraz 'RTI' kończącym przerwanie. W najkrótszej wersji nasz VBL wyglądać może tak:

```
vbl      plr
         rti
```

Główna pętla:

```
loop     lda:cmp:req 20

         jsr multi.show_sprites

         jsr multi.animuj

         ...
         sta sprite_x
         ...
         sta sprite_y

         jmp loop
```

Jeśli nie chcemy automatycznie animować obiektów możemy zrezygnować z linii 'jsr multi.animuj'

Mapa pamięci

Po zainicjowaniu silnika do dyspozycji dostajemy całą przestrzeń pamięci, ROM zostaje zastąpiony przez RAM.

Tablice po 16 bajtów każda z parametrami dla obiektów

```
sprite_x      ;pozycja X obiektu
sprite_y      ;pozycja Y obiektu
sprite_shape  ;kształt obiektu
sprite_c0     ;kolor 0 obiektu
sprite_c1     ;kolor 1 obiektu
sprite_anim   ;liczba klatek animacji obiektu
sprite_anim_speed ;szybkość animacji obiektu
```

Jeśli pozycja pozioma 'SPRITE_X' = 0 wówczas taki obiekt traktowany jest jako wyłączony.

Dodatkowe 32 bajtowe tablice z informacją o starszym adresie zestawu znaków 'charset', wartości dla rejestru koloru 'tcolor'. Wartości z tych tablic obowiązują dla kolejnych wierszy ekranu.

```
charset
tcolor
```

Domyślnie rejestrem do którego zapisywane są wartości z 'tcolor' jest \$d01e, aby zmiany odnosiły się do konkretnego rejestru koloru należy go ustawić z pomocą etykiety 'creg'.

```
lda <$d01a      ; ustawienie rejestru koloru $d01a
sta creg+1
```

Program dla ANTIC-a

W programie dla ANTICA musimy ustanowić przerwanie DLI co wiersz

```
dlist      dta $f0
           :26 dta $40|$04|$80|$10,a(scr+##48)
           dta $41,a(dlist)
```

Program rozpoczyna się 8 pustymi liniami z włączonym przerwaniem DLI (\$70 + \$80 = \$f0)

Zmiana początku programu, dodanie innej liczby pustych linii spowoduje błędne działanie silnika.

Preferowaną szerokością obrazu jest wąski obraz ze względu na największą liczbę wolnych cykli dla CPU. Można stosować inne szerokości jednak trzeba się liczyć z częstszym występowaniem 'glitchy' czyli urwanych fragmentów obiektów. Efekt taki spowodowany jest zbyt późną zmianą rejestrów w linii.