

ZINBO

MAGAZYN RETROKULTURALNY

SPECCY.PL





Zin80

MAGAZYN
UŻYTKOWNIKÓW
MIKROKOMPUTERÓW
Z PROCESOREM Z80

WYDAWANY PRZEZ SERWIS

SPECCY.PL

REDAKCJA:

TYGRYS – REDNACZ

AUTORZY ARTYKUŁÓW:
ANDY REMIC, CHICADII,
DALTHON, DANIEL KOZMINSKI,
PEAR, SACHY, SIR DAVID,
TYGRYS, VOYAGER

OKŁADKA:
DKL

PROJEKT LOGOTYPU:
DKL

KOREKTA:
ENDER, MAT/ESI

SKŁAD:
FAUST (T2E.PL)

KONTAKT:
TYGRYS@SPECCY.PL

WWW:
SPECCY.PL

3... 2... 1...

Pisanie wstępniaka powinno być przyjemnością, wieńczącą koniec prac nad wydaniem kolejnego numeru pisma. Nie jestem do końca o tym przekonany, ponieważ ja nad nim siedzę już trzecią godzinę, pisząc chyba z dziesiątą wersję. Nie wiem czy Borek czy też MPS mieli z tym takie same problemy, oby nie. ;-)

Ten rok był dla mnie dość ciężki, głównie jeżeli chodzi o natłok obowiązków służbowych. To głównie przez to drugi numer ukazuje się na początku 2020. Najważniejsze jest to, że jednak się ukazuje! Sześćdziesiąt stron wypełnionych tekstami dotyczącymi komputerów z Z80 oraz oprogramowania dla nich, to moim zdaniem niezły wyczyn! Dłuższy czas wydawania numeru, to też okazja do tego, aby poprawić rzeczy, które kulały podczas naszego debiutu. Pędziliśmy z nim, aby wyrobić się kilka dni przed Speccy.pl party 2019.1, przez co nie było czasu na dokładną korektę już złożonych tekstów. Tym razem jest inaczej, było więcej czasu na poprawki, a dzięki wysiłkowi Endera i Mata, którzy wyłapali wiele literówek, jakość tekstów jest o wiele wyższa. Za skład po raz kolejny wziął się Faust. To dzięki niemu pismo wygląda atrakcyjnie i kolorowo!

W tym numerze teksty są już bardziej różnicowane, niż w poprzednim. Owszem, jest dużo opisów gier, ale nie stanowią już większości numeru. Są teksty publicystyczne – relacja ze Speccy.pl party 2018.2, którą podzielił się Dalthon, jest też tekst o powstawaniu filmu *Memoirs of a Spectrum Addict* Andy'ego Remica. Są też ciekawostki z dziedziny sprzętu – Pear napisał krótki tekst o Enterprise – komputerze niezbyt popularnym w Polsce, za to posiadającym duże możliwości graficzne. Kolejny artykuł opisuje sposób urucho-

mienia CP/M na stacji dysków do Atari. Tak – LDW Super 2000 jest konstrukcją opartą o Z80! Są i poradniki dotyczące programowania Z80. Sachy podzielił się wiedzą, w jaki sposób zacząć pisać programy (dema) na ZX Spectrum w assemblerze, a drugi tekst jest mojego autorstwa, opisuje w nim podstawy kodowania na Z80 w taki sposób, aby były zrozumiałe dla osób znających podstawy programowania w BASI-Cu (szczególnie dla ZX Spectrum). Całość została recenzowana przez kilka losowo wybranych osób, a ich uwagi uwzględnione w tekście. O tym, że BASIC dla Sama Coupé jest prosty i do tego mega elastyczny, możecie przeczytać w artykule Sir Davida, naszego eksperta od tych komputerów. Retrospecs to narzędzie do konwersji obrazów/filmów do formatów dla komputerów ośmiobitowych, z pewnością znajdzie zastosowanie wśród amatorów grafiki! Jak już wspomniałem – opisów gier również nie zabraknie, bo jako oprogramowania, powstaje ich najwięcej. Recenzji doczekały się Mister Kung Fu, Aliens Neoplasma. VOyager przygotował drugą część przeglądu unikalnych gier na ZX Spectrum.

Nieustannie zapraszamy każdego, kto chce podzielić się swoją wiedzą i napisać tekst do Zin80. Jestem pewien, że każdy może to zrobić, wystarczy odrobina dobrej woli. Jeżeli będzie potrzebna pomoc, możecie na nas liczyć. ;-)

Teraz powinno paść pytanie – kiedy ukazuje się kolejny numer? Odpowiedź jest prosta od Speccy.pl party 2020.1, które odbędzie się na początku kwietnia, dzielą nas tylko trzy miesiące i wierzę, że jednak do tego czasu uda się złożyć kolejny numer.

Życzę przyjemnej lektury!

TYGRYS

W NUMERZE :

SPECCY.PL 2018.2 RAPORT	3
ASSEMBLER Z80 DLA PRAWIE KAŻDEGO	8
PIERWSZE KROKI SPECTRUMOWEGO KODERA	15
SAM BASIC	23
ENTERPRISE	26
CP/M NA STACJI DLA ATARI	28
RETROSPECS	33
MISTER KUNG-FU	37
ALIENS: NEOPLASMA	40
UNIKALNE GRY NA ZX SPECTRUM	43
MEMOIRS OF SPECTRUM ADDICT	53

SPECCY.PL

PARTY 2018.2



dalthon

RAPPORT

RAPPORT

RAPPORT

Speccy.pl party to impreza przeznaczona dla wszystkich fanów komputerów Sir Clive'a Sinclaira, która od samego początku była planowana jako dwie edycje w roku. Niestety, długo, bo aż do 20 października 2018 roku nie udawało się zorganizować edycji z numerem 2! Po dużej (i jak zawsze udanej!) kwietniowej odświeżce, kolejna była pierwotnie zaplanowana na mniejszą, skupioną przede wszystkim na integracji środowiska.

CIACH...

I w tym momencie (pisząc i robiąc korektę) doszedłem do wniosku, że szkoda miejsca na tak suchą i drętą relację (której próbkę macie powyżej) z party. Przypomniały mi się opowieści z imprez jakie ukazywały się w różnych DiscMagach w ubiegłym wieku. Były pełne emocji i czytający taki tekst albo ciepło wspominali tamte miłe chwile (jeśli je oczywiście pamiętali!)), albo zgrzytali zębami żałując, że ich tam nie było. Ponieważ pierwsze wydanie Zin80 cieszyło się dużym zainteresowaniem także wśród fanów innych marek niż Sinclair, zdecydowałem się od początku napisać tę relację – tak by, przede wszystkim zachęcała do odwiedzenia kolejnych edycji Specy.pl party!

DROGA PRZEZ MEKĘ (NA WŁASNE ŻYCZENIE;)

Robiąc produkcję na party (jak zwykle na ostatnią chwilę) kolejny raz zarwałem nockę. Na całe szczęście nasze rodzime TGV (czyt. Pendolino) jeździ częściej niż dwa razy dziennie do Warszawy, dzięki czemu udało mi się wsiąść do pociągu prawie w samo południe. Trzy i pół godziny minęło bardzo szybko, a to za sprawą snu na jaki mogłem sobie tym razem pozwolić podczas podróży, czego z kolei nie mogłem zrobić jadąc na party 2017.1 i 2018.1, kiedy próbowałem dociec dlaczego moja produkcja nie działa jak należy, i co z tym zrobić. Jako że w stolicy byłem przed 16.00, to znając już drogę na Bohomolca cieszyłem się z faktu, że przed 17.00 ponownie zobaczę znajome twarze. Cóż, jak to w życiu bywa, zamiast zaufać rutynie i doświadczeniu, i iść znaną trasą, zachciało mi się skorzystać z nawigacji w nowym telefonie... Jak się łatwo domyślić, zamiast skrócić, wydłużyłem sobie czas wędrówki i na party place dotarłem o 18.00! Zaufałem technice i dzięki temu zrobiłem dodatkowe kilometry pieszo, co może nie jest złe dla zdrowia, ale zważywszy, że byłem wystawiony na opady deszczu, nie jestem już tego taki pewny.

CEL OSIĄGNIĘTY: BOHOMOLCA 15

Przechodząc przez drzwi znanego już domu, w którym odbywało się party, delikatnie mówiąc nie tryskałem entuzjazmem, ale zmieniło się to w ułamku sekundy, gdy tylko zobaczyłem znajome twarze, a było ich bardzo dużo. Z zaplanowanej mocno kameralnej imprezki, zrobiła się impreza pełną gębą!

Pojawiło się tylu ludzi, że przechodząc miejscami było naprawdę ciasno. Partyzanci stali w mniejszych i większych grupach, podziwiali zgromadzone sprzęty, oglądali produkcje na odbornikach TV czy też prowadzili ożywione dyskusje, które były niczym w porównaniu do zgietku, jaki panował w specjalnym miejscu na werandzie i wcale nie dlatego, że padał tam wtedy deszcz:).

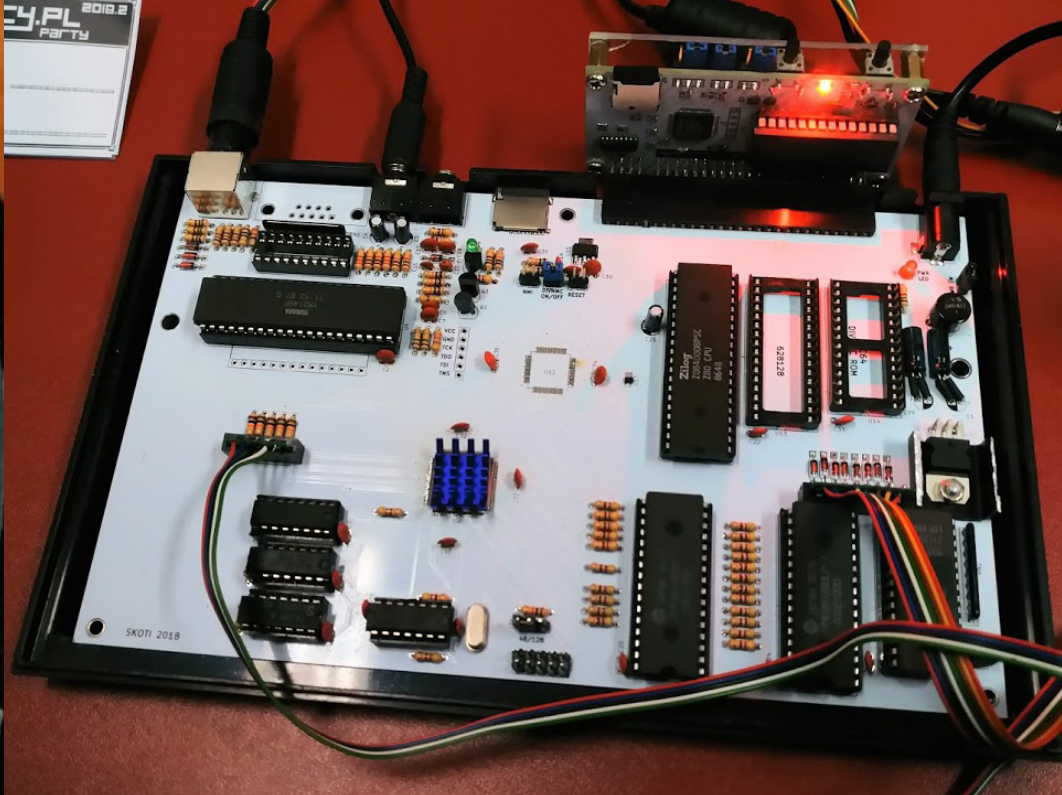
ATRAKCJE, ATRAKCJE, ATRAKCJE

Pomimo że z założenia impreza miała mieć charakter głównie integracyjny, to atrakcjami można by było obdzielić kilka konkurencyjnych party!



▲ JESZCZE NIE WSZYSCY SIĘ POJAWILI...
▼ TIMEX RZĄDZI, TIMEX RADZI, TIMEX NIGDY CIĘ NIE ZDRADZI!





▲ NAJNOWSZY KLON ZX'A: SPIDER

▼ BIGSCREEN TO PODSTAWA!



▼ NIESPODZIEWANY KONCERT KRAFTWERK



Przedpremierowo mieliśmy okazję obejrzeć fragment filmu dokumentalnego *Gry, użytki - co dla Ciebie* przygotowanego przez ekipę serwisu Loading.

Zainteresowani mogli wysłuchać ciekawej historii firmy Timex – a jakby tego było mało, Zoon nie ograniczył się tylko do wygłoszenia przygotowanej prezentacji! Zaprezentował prawie wszystkie maszyny jakie pojawiły się z logo Timex: nie tylko komputery, ale również stacje dysków, magnetofony, drukarki, czy całą masę przystawek! TimexPorn normalnie! :) Co innego o nich słyszeć, a co innego je zobaczyć i dotknąć – takie okazje nie zdarzają się codziennie!

Jeżeli już wspominać o hardware, to warto dodać, że na party było tego więcej. Poza standardowymi maszynkami trafił się: Sam Coupé, „laptopowe” przenośne ZX'y, czy spectrumowe klony – w tym najświeższy Spider!

Phonex natomiast zaprezentował swój najnowszy program: kompresor obrazka, który... dekompresuje się w trakcie ładowania go z taśmy! Szok i niedowierzanie ile jeszcze rzeczy można wycisnąć z wiekowego ZX, na widok których, twórcy tej maszynki mogliby wykrzyknąć tylko jedno: „Nie! To niemożliwe!”

COMPETITION

Każdy uczestnik party, poza rzecz jasna identyfikatorem, otrzymywał na wejściu tajemniczy kod. Jego zastosowanie stało się jasne, gdy przyszła pora na konkursy demoscenowe. Pozwalał on załogować się (za pomocą telefonu) na specjalną stronę i zagłosować na prace, które były pokazywane na big-screenie! Przyznam, że jak na imprezę nastawioną z założenia na integrację, było ich zaskakująco dużo, bo aż jedenaście. Dzięki głosowaniu on-line, wyniki można było poznać dosłownie chwilę po zakończeniu konkursów, a ich uczestnicy, mogli zostać obdarowani nagrodami.

BYLE DO PÓŁNOCY

Po tych wszystkich atrakcjach można było powrócić do przerwanych dyskusji, napić się piwa w doborowym towarzystwie, a także (co jest tradycją tego miejsca) zjeść coś smacznego z grilla.

Wszystko co dobre, niestety szybko się kończy i nim się wszyscy obejrzeli było już moooooocno po północy i trzeba było kończyć imprezę. :/

Tygrys po raz kolejny przygotował wspaniałe party i żałować mogą tylko ci, którym nie udało się na nie dotrzeć. Warto wspomnieć, że impreza była transmitowana on-line jak przystało na XXI wiek. ;)

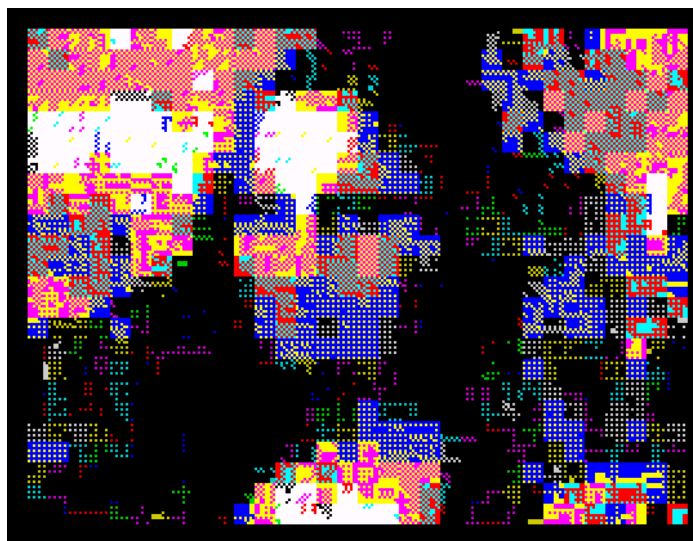
AFTER PARTY

Część partowiczów nie mogła się ze sobą rozstać, więc udała się na wcześniej ustalone pozycje. ;)

Również organizatorzy, którzy to dopiero teraz mogli się zrelaksować – bo gdy wszyscy się dobrze bawili na party, oni byli w „pracy” – dali się namówić na imprezowanie. Niestety relacja z tej części imprezy pozostanie jedynie w formie przekazu ustnego, a że była tak udana, to niektórzy do domu nie trafili w niedzielę. :D



WYNIKI ZMAGAŃ TYCH KTÓRYM CHCIAŁO SIĘ COŚ ZROBIĆ WYGLĄDAJĄ NASTĘPUJĄCO:

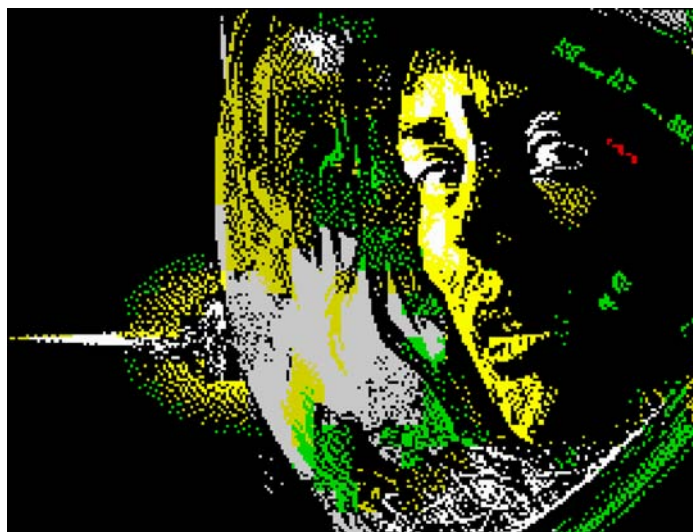


1. 63 PKT. **ZX TRIBUTE TO MONA** - DALTHON ▲
2. 46 PKT. **WE NEED BEN BACK** - VOYAGER

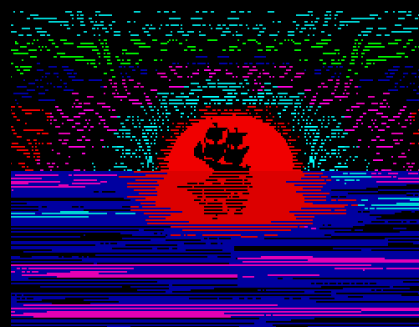
GFX:

1. 68 PKT. **SHIP IN THE BLOOD** - ROGAL ►
2. 61 PKT. **NO TIME** - ZX FREEQ
3. 48 PKT. **SPECCY** - MANDI
4. 46 PKT. **JAZZ** - DAKTI

GFX COPY:



1. 77 PKT. **ISOLATION** - TOOLOUDTOOWIDE ▲
2. 71 PKT. **DRAGON** - TOOLOUDTOOWIDE
3. 65 PKT. **ASSASSINATION** - DALTHON
4. 56 PKT. **MÓZG ROZJEBANY** - DAKTI
5. 47 PKT. **W SRACZU** - ROGAL



Ship In The Blood

Rogal
ZX Spectrum
(256 x 192, 16 colors)

I miejsce na
Specy.pl Party 2018.2
w kategorii GFX Compo

ASSEMBLER Z80

OD POSTAW DLA PRAWIE KAŻDEGO

TYGRYS

DLA KOGO JEST TEN TEKST?

Kursów programowania w assemblerze Z80 jest wiele, czy to w postaci książek, czy też poświęconych temu blogów oraz kanałów na YouTube. Ten kurs jest inny. Zakładam, że osoba, która chce nauczyć się assemblera umie już programować. Założyłem, że znanym językiem programowania jest BASIC - najprostszy do opanowania z języków „retro”. Zakładam również, że czytelnik wie w jaki sposób są zapisywane liczby w systemie binarnym i szesnastkowym oraz czym są bity. W niniejszym artykule stosuję pewne uproszczenia, które nie przytłoczą czytelnika nadmiarową wiedzą. Pominięte rzeczy zostaną omówione w kolejnych częściach kursu. Zakładam również, że platformą, na której czytelnik będzie szlifować swoją wiedzę z assemblera będzie ZX Spectrum.

WSZYSTKO CO NIEZBĘDNE ABY ZACZAĆ

Jeżeli ktoś chce poczuć się jak w latach osiemdziesiątych, może uczyć się na oryginalnym sprzęcie, korzystając z wielu dostępnych kompilatorów jak *Gens* czy *ZEUS Assembler*. Znacznie wygodniejsze, w obecnych czasach, jest używanie oprogramowania dla współczesnych systemów. Proponuję użycie kompilatora *Pasmo*, dzięki któremu nie trzeba się martwić o tak zwany „loader”. *Pasmo* generuje plik typu .tap z programem wraz z dołączonym loaderem.

Program piszemy w swoim ulubionym edytorze tekstu. Ja do tego używam Notepad++. Plik z listingiem programu może mieć dowolne rozszerzenie, lecz ja stosuję .asm co sprawia, że tworzę pliki nazwane program.asm. Do stworzenia szkieletu takiego pliku proponuję użyć wzorca jak poniżej. Po zapisaniu pliku na dysku, należy go następnie skompilować. Zakładam, że czytelnik wie czym jest linia poleceń i umie się nią posługiwać.

Kompilacja następuje poprzez:

```
pasmo --tapbas program.asm program.tap
```

zaś najprostszy program wzorec wygląda następująco:

```
;-----  
org 32768      ; ORG to dyrektywa kompilatora,  
               ; mówiąca pod jaki adres  
               ; należy skompilować program  
start          ; to jest etykieta, musi  
               ; mieć nazwę inną niż  
               ; instrukcje asm i dyrektywy  
               ; kompilatora  
  
;  
; tutaj trzeba umieścić główny program  
;  
RET           ; o RET będzie w dalszej części  
               ; artykułu tutaj jest miejsce na  
               ; dane lub inne procedury
```

```
end start      ; dzięki temu  
               ; działa --tapbas w pasmo
```

```
;-----
```

Wynikiem wykonania programu *Pasmo*, o ile nie wystąpiły błędy kompilacji, będzie *program.tap*. Należy go uruchomić pod dowolnym emulatorem ZX Spectrum. Moje ulubione emulatory to *ZX Spin* i *Fuse*.

UNIWERSALNE LD

To polecenie, które jest najczęściej używaną instrukcją i służy do ładowania danych, dlatego warto jest je poznać na samym początku. Jest to skrót od LOAD (ang. ładuj). Składnia jest następująca:

```
LD    cel, źródło
```

Aby przypisać/przesać liczbę do celu (rejestru), po prostu się ją przypisuje, a ten sposób adresowania nazywa się natychmiastowym, zapis w BASICu wygląda następująco:

```
LET a = 10
```

co przekłada się na instrukcję w assemblerze w kilku wariantach zapisu liczb:

```
LD a, 10      ; zapis dziesiętny  
LD a, $0A     ; można też stosować  
               ; zapis szesnastkowy,  
               ; wielkość liter nie ma znaczenia  
LD a, %00001010 ; ale można też dwójkowy
```

Aby zaznaczyć, że chcemy załadować lub zapisać daną z/do pamięci, stosuje się nawiasy, które otaczają podany argument.

```
LD a,(2000)   ; załaduj do A zawartość komórki  
               ; pamięci o adresie 2000  
LD (5000),a   ; Zapisz do pamięci pod adres  
               ; 5000 wartość rejestru A
```

Powyższy zapis realizuje to samo co w BASICu:

```
LET a=PEEK 2000  
POKE 5000,a
```

REJESTRY 8 BITOWE

Dostępne są następujące rejestry ogólnego przeznaczenia: **A, B, C, D, E, H, L**. Rejestr możemy traktować jak zmienną w BASICu i dowolnie ją interpretować. Może to być aktualna liczba żyć w grze, liczba pocisków do wykorzystania, dzielnik, kolor znaku, kod ASCII litery „A”, liczba powtórzeń w pętli itp. Wyprzedzę trochę temat i dodam, że jest jeszcze jeden „magiczny” rejestr 8-bitowy i jest on zapisywany jako **(HL)**. Ma bardzo specyficzną cechę - jego wartością jest zawartość komórki pamięci - o czym świadczą nawiasy - wskazywana przez 16-bitowy rejestr **HL**.

Najczęściej używanym podczas programowania rejestrem jest **A** zwany również akumulatorem. To w nim lądują wyniki działań arytmetycznych, logicznych, itp. Wielokrotnie w tekście będę pokazywać wyjątkowość Akumulatora.

Najprostszą rzeczą jaką można zrobić, jest przypisanie wartości do rejestru. Wartość może być przypisana bezpośrednio do rejestru lub też przesłana z innego.

W BASICu zapisalibyśmy to następująco:

```
LET a=10 : LET b=a : LET h=b
```

W assemblerze wygląda to tak (zapis małymi lub dużymi literami nie ma znaczenia, ale instrukcje zapisują dużymi literami, argumenty małymi):

```
LD a,10
LD b,a
LD h,b
```

Z (HL) sprawa wygląda trochę inaczej, chociaż można go również stosować następująco:

```
LD d,(hl)
LD (hl),d
LD (hl),10      ; prześlij wartość 10 do komórki
                 ; pamięci na której
                 ; adres wskazuje
                 ; wartość rejestru HL
```

to w BASICu nie ma dobrego porównania do (HL), więc trzeba się odwołać do operacji na pamięci:

```
LET d = PEEK hl
POKE hl,d
POKE hl,10
```

Jednak z (HL) należy postępować ostrożnie, gdyż zmiana wartości rejestrów H lub L spowoduje zmianę komórki pamięci wskazywanej przez tę parę rejestrów (temat par rejestrów zostanie poruszony niebawem). Zatem LD 1, (hl) wykonywać należy tylko wtedy, kiedy wiemy dobrze co chcemy takim przypisaniem osiągnąć.

Jest jeszcze jeden rejestr, który jest używany bardzo często, aczkolwiek zwykle nie odwołujemy się do niego bezpośrednio (poza jednym wyjątkiem, o którym napiszę w kolejnej części artykułu). Jest to rejestr F. Jest on używany przez procesor do składowania na pojedynczych bitach dodatkowych informacji o wyniku operacji na danych oraz warunkowego wykonania kodu zależnie od tych wyników. Najczęściej wykorzystywanymi flagami (znacznikami) są:

Z (zero) - bit mówiący, że efektem ostatniego działania jest zero,

C (Carry - przeniesienie) - bit, w którym zapisuje się nadmiarowy bit z działań arytmetycznych oraz operacji na bitach.

Z tym rejestrem związane są następujące, dedykowane instrukcje:

SCF - Set Carry Flag - ustawia flagę C na wartość 1

CCF - Complement Carry Flag - ustawia flagę C na jej odwrotność (zamienia 0 na 1 albo 1 na 0)

Nie ma dedykowanej instrukcji zerującej wartość Carry. Jeżeli wartość znacznika Z nie jest w danym momencie istotna, Carry można zresetować wykorzystując właściwości instrukcji logicznej OR (o której niebawem) bez zmiany wartości rejestru A:

```
OR A
```

OPERACJE NA DANYCH 8-bitowych

Można dodawać, odejmować, porównywać, operować wartościami arytmetyki boole'owskiej. Pamiętać należy, że wartość operacji arytmetycznej (prawie) zawsze zapisywana jest do rejestru A. Oto kilka przykładów, które zastąpią 1000 słów.

Instrukcją dodawania jest ADD A, argument. Argumentem może być dowolny rejestr 8 bitowy oraz liczba ośmiobitowa. Oto przykłady:

```
LET a=10
LET b=20
```

```
LET a=a+20
```

```
LET a=a+b
```

i odpowiednio w assemblerze:

```
LD a,10
LD b,20
ADD a,20      ; bezpośrednie dodawanie
               ; do A liczby 8bit
ADD a,b       ; oraz dodanie wartości
               ; z innego rejestru
```

A co jeżeli mamy w A potrzebną wartość, a chcemy dodać wartości innych rejestrów? Musimy na chwilę przechować daną w A w innym rejestrze (lub zapisać ją na stosie, o czym już niedługo) w następujący sposób:

Prosty zapis w BASICu:

```
LET a=120: LET b=10 : LET e=30
LET b=b+e
```

Przekłada się na zapis w assemblerze:

```
LD a,120
LD b,10
LD e,30      ; nie ma instrukcji ADD b,e
               ; dlatego też musimy w dodawanie
               ; zaangażować rejestr A
LD c,a       ; w rejestrze C
               ; przechowujemy wartość A
```

```
LD a,b       ; właściwe dodawanie
ADD a,e
LD b,a
LD a,c       ; odzyskanie
               ; wartości rej. A
```

Jeżeli wartość dodawania przekroczy wartość 8-bitową, czyli będzie większa niż 255, wtedy flaga „Carry” z rejestru F posłuży jako dodatkowy 9. bit. Informację tę można wykorzystać na kilka sposobów. Jedną z nich jest wykorzystanie w instrukcjach warunkowych, drugą zaś potraktowanie wartości z „Carry” jako bitu pomocniczego podczas działań na rejestrach. Dodawania z uwzględnieniem wartości Carry realizuje się za pomocą instrukcji:

```
ADC A,rejestr/liczba
```

Można to wykorzystać do dodawania liczb 16-bitowych przy pomocy 8 bitowych rejestrów, gdzie pomocny jest bit przeniesienia Carry:

```
LD h,0
LD d,200     ; łącznie w rej. h i d jest 200
LD a,100
ADD a,d      ; 200+100 daje łącznie 300, zatem
               ; to jest więcej niż 255.
               ; Następuje przepełnienie
               ; rejestru 8bit, do Carry
               ; trafia 1 (zapasowy 9. bit),
               ; a do A wartość 300-256 = 44
LD d,a       ; tę wartość zapisujemy
               ; z powrotem do D
LD a,0       ; A = 0
ADC a,h      ; ...i dodajemy do A
               ; zawartość H oraz Carry
LD h,a       ; przepisujemy wynik - w
               ; tym wypadku 1 - do H
```

Odejmowanie realizowane jest za pomocą instrukcji SUB argument. Argumentami mogą być, tak jak dla dodawania, rejestry oraz liczba ośmiobitowa. Przykład w BASICu:

```
LET c=30
LET a=a-c
```

Oraz odpowiadający fragment w assemblerze:

```
LD c,30
```

```
SUB c      ; alternatywny zapis
           ; to SUB a,c
```

Odejmowanie dla liczb ośmiobitowych z uwzględnieniem bitu przeniesienia „Carry” to SBC A, rejestr. Oto przykład w asm:

```
LD a,50
LD c,40
SCF      ; ustawienie „Carry” na wartość 1
SBC a,c  ; A = A - c - Carry ->
           ; a = 50 - 40 - 1
           ; wyniku otrzymamy 9,
           ; zapisane w rejestrze A
```

W wypadku, kiedy „Carry” jest wyzerowane, SBC zadziała tak samo jak SUB.

Czasami, a nawet bardzo często jest tak, że do rejestru, lub komórki pamięci wskazywanej przez (HL), chcemy dodać lub odjąć wartość 1. Do tego przeznaczone są instrukcje INC i DEC.

```
LET h=h+1
LET c=c-1
```

a w assemblerze odpowiednio:

```
INC h
DEC c
```

A zapis w BASICu:

```
POKE hl,1+PEEK hl
```

to odpowiednik assemblerowego:

```
INC (hl)
```

Do porównania dwóch wartości służy rozkaz CP wartość. Wartość jest zawsze porównywana do tego, co jest w akumulatorze:

```
LET a=10
LET b=20
```

```
LD a,10
LD b,20
CP b
```

CP wewnętrznie wykonuje operację odejmowania argumentów tak samo jak SUB tylko nie zapisuje jej wyniku w A, a jedynie ustawia flagi. W efekcie gdy porównywane wartości są równe to wtedy flaga **Z** w rejestrze **F** zawiera wartość 1, w przeciwnym przypadku 0. Jeżeli wartość w A jest mniejsza niż porównywana, to flaga **C** jest ustawiana na 1, jeżeli zaś A jest równe lub większe od porównywalnej wartości, wtedy flaga **C** jest resetowana (przyjmuje wartość 0). Przyda się to później do instrukcji warunkowych.

Innymi istotnymi instrukcjami są operacje logiczne AND, OR i XOR. Jest to podstawa arytmetyki binarnej, ale z racji tego, że w BASICu ZX Spectrum AND i OR działały jedynie na wartościach logicznych, a nie bitach, pozwoliłem sobie na krótkie wyjaśnienie, w jaki sposób działa każda z nich.

AND wartość to jest logiczne „i” dla bitów czyli wartość bitu 1 będzie tylko wtedy kiedy oba „równoległe” bity będą miały wartość 1. W przeciwnym razie jest 0. Oto szybki przykład:

```
LD a, %11001100
LD b, %10101010
AND b
```

sprawi że w rejestrze A pojawi się wartość:

```
%10001000
```

OR wartość to logiczne „lub” dla bitów czyli wartość bitu 1 będzie wtedy jeśli którykolwiek z „równoległych” bitów będzie miał wartość 1. W przeciwnym razie jest 0.

```
LD a, %11001100
LD b, %10101010
OR b
```

w rezultacie wyniesie:

```
%11101110
```

XOR wartość to „alternatywa wykluczająca” czyli wartość bitu 1 będzie tylko wtedy kiedy „równoległe” bity będą miały przeciwstawne wartości. W przeciwnym razie jest 0.

```
LD a, %11001100
LD b, %10101010
XOR b
```

otrzymamy wynik:

```
%01100110
```

Wszystkie wymienione operacje logiczne ustawiają w rejestrze stanów **F** flagi - **Z** zależnie od otrzymanego wyniku, zaś **C** jest zawsze zerowana.

REJESTRY 16 BITOWE

To rejestry, które mogą przechowywać wartości 16-bitowe czyli zakres liczbowy od 0 do 65535 (\$0000...\$FFFF szesnastkowo). Do rejestrów 16-bitowych zaliczają się **SP**, **IX** i **IY**.

Powstają one również z połączenia znanych już rejestrów ośmiobitowych: **BC**, **DE** i **HL**. Zakres stosowania rejestrów 16-bitowych jest mniejszy niż 8-bitowych, niemniej są bardzo pomocne - bez nich operowanie danymi w pamięci byłoby znacznie mniej wygodne.

Wcześniej wspominałem już o (**HL**) jako 8-bitowym rejestrze pamięci. To nic innego jak złożenie do siebie dwóch rejestrów **H** i **L**. Co to właściwie oznacza? Przykład w BASICu będzie najlepszy!

```
LET h=8
LET l=45
LET HL=256*h + l : REM co daje wartość 2093
```

zatem:

```
LD h,8
LD l,45
```

jest odpowiednikiem:

```
LD hl,2093
```

Powyższe dotyczy także par **BC** i **DE**.

BC i **DE** mogą służyć jako rejestry pamięci, tak jak to ma się z **HL**. Jednak dane można przesyłać wyłącznie do i z **A**:

```
LD a,(bc)
LD a,(de)
LD (de),a
LD (bc),a
```

aby przesłać do innego rejestru, trzeba najpierw posłużyć się **A**:

```
LD a,(de)
LD c,a
```

Możliwe jest też bezpośrednie przesłanie zawartości rejestru do pamięci, co dla operacji na 8 bitach nie było możliwe:

```
LD de,(1000)
LD (2000),de
```

OPERACJE NA DANYCH 16-bitowych

Przy arytmetyce 16-bitowej, podstawowym rejestrem jest **HL**, w większości pełniący taką funkcję jak **A** dla arytmetyki 8-bitowej.

Zatem:

```
ADD hl,hl
```

```
ADD hl,de
ADD hl,bc
ADD hl,sp
```

Niestety nie da się dodać do HL liczby podanej bezpośrednio. Musi się ona znaleźć w którymś z rejestrów BC lub DE. Na przykład:

```
LD de,10305
ADD hl,de      ; co jest równoważne
                ; z LET hl=hl+10305
```

Dla rejestrów 16-bitowych niedostępne jest odejmowanie bez znacznika/flagi C. Można je zrealizować poprzez dodawanie liczb ujemnych, na przykład:

```
LD hl,10000
LD bc,-2000
ADD hl,bc      ; co jest równoważne
                ; z LET hl=hl-bc
```

Z „Carry“ można już:

```
SBC hl,bc
SBC hl,de
```

Działa również inkrementacja i dekrementacja:

```
INC hl
DEC hl
```

Działania te nie ustawiają znacznika Z więc nie dowiemy się bezpośrednio, czy wartość rejestru osiągnęła 0.

Z80 nie posiada arytmetyki bitowej dla 16-bitów.

Przesył informacji między 16-bitowymi rejestrami nie jest możliwy. Trzeba go realizować za pomocą 8-bitowych prześłań. BASICowy program:

```
LET hl=1000
LET bc=hl
```

realizuje się w następujący sposób:

```
LD hl,1000
ld b,h
ld c,l
```

Konstruktorzy Z80 przewidzieli pewną sytuację, mianowicie pozwolili na szybką zamianę (a nie skopiowanie) wartości rejestrów między HL i DE. Instrukcja ma postać:

```
LD hl,1000
LD de,500
EX de,hl
```

a jej działanie można zobrazować następująco:

```
LET hl=1000
LET de=500
LET temp=hl
LET hl=de
LET de=temp
```

Ma to zastosowanie na przykład przy dodawaniu wartości dla DE i BC. Oto przykład:

```
LD hl,1000
LD de,4000
LD bc,300

                ; ADD de,bc nie istnieje, w takim
                ; przypadku można to
                ; zrealizować w następujący sposób,
                ; zachowując wartość rejestru HL

EX de,hl
ADD hl,bc
EX de,hl
```

Wspomniane wcześniej rejestry IX i IY to rejestry indeksowe. Oznacza to, że mogą wskazywać pewien obszar pamięci, z którego odczytywać dane można z przesunięciem, zwanym indeksem.

Oto przykład:

```
LD ix,1000
LD a,(ix+10)
```

gdzie indeks mogą stanowić liczby od -128 do 127.

Nie ma możliwości użycia jakiegokolwiek rejestru 8bit jako indeksu.

W BASICu wyglądałoby następująco:

```
LET ix=1000
LET a=PEEK (ix+10):REM czyli odczyt
z pamięci o adresie 1010
```

Wartości indeksowe mogą być używane podobnie jak to miało miejsce z (HL):

```
LD b,(ix+10)
ADD a,(ix+110)
SUB (iy-10)
SBC a,(ix+0)
OR (ix-12)
```

STOS

SP to niezwykle rejestr - to rejestr stosu. Wskazuje miejsce w pamięci, które służy jako magazyn na tymczasowe dane. Wartości SP można ustawiać bezpośrednio wpisując dowolny adres pamięci, ale również kopiować z rejestrów HL, IX i IY.

```
LD sp,65545
LD sp,hl
LD sp,ix
```

Dozwolone jest używanie stosu w dodawaniu:

```
ADD hl,sp      ; również IX i IY
                ; wartość dodawania znajdzie się
                ; w docelowym rejestrze.
```

Najważniejszą cechą stosu jest możliwość chwilowego zapamiętywania wartości rejestrów. Do odkładania wartości służy instrukcja PUSH rejestr_16bit, gdzie spodziewanym argumentem są: AF, BC, DE, HL, IX i IY. Transfer wartości ze stosu do rejestru zajmuje się instrukcja POP rejestr_16bit przyjmując dokładnie takie same argumenty jak PUSH. Należy pamiętać o tym, że stos działa wg zasady LIFO (ang. last in first out - ostatni wchodzi, pierwszy wychodzi) - jeśli chcemy tylko odtworzyć dane to kolejność ściągania danych ze stosu musi być odwrotna do kolejności ich odkładania. Zmiana kolejności powoduje, że dane trafiają do innego rejestru docelowego niż były przy odkładaniu. Użycie stosu do tymczasowego przechowywania danych jest następujące:

```
PUSH de      ; rejestr DE zostaje
              ; odłożony jako pierwszy

PUSH bc
PUSH af      ; AF jest odkładany jako
              ; ostatni... jakieś operacje
              ; zmieniające dane w
              ; rejestrach

POP af       ; ...i
              ; „odzyskiwany” jako pierwszy

POP bc
POP de      ; DE zostaje pobrany jako ostatni
```

PUSH i POP mogą być też używane jako wygodny sposób zamiany wartości między rejestrami. Stos pełnić tu będzie rolę tymczasowego miejsca do przechowywania danych. Dla przykładu, zamieńmy miejscami wartości rejestrów HL i BC:

```
PUSH hl      ; wartość HL na stos
PUSH bc      ; BC na stos
POP hl       ; do HL „leci” wartość z BC
POP bc       ; a do BC trafia wartość z HL
```


SKOKI DO PODPROGRAMÓW I BEZWARUNKOWE

Podprogramy i procedury niezwykle zwiększają czytelność programu. Dzięki nim możemy powtarzać pewne czynności wielokrotnie bez potrzeby duplikowania kodu. Instrukcją przejścia do podprogramu z możliwością powrotu jest CALL adres_bezwzględny a powrotem z procedury jest RET (od angielskiego RETURN). Wykonanie skoku do podprogramu (CALL) zapisuje przed skokiem adres powrotu - czyli adres następnego rozkazu po CALL - na stosie a wykonanie RET zdejmuje ze stosu pierwszą wartość i „wraca” pod ten adres. Należy w związku z tym pamiętać, żeby w podprogramie nie pozostawić nadmiarowych danych na stosie albo nie zdjąć przypadkiem adresu powrotu, bo w takim wypadku RET nie wróci z podprogramu tylko skoczy w losowe miejsce, co zwykle kończy się zawieszeniem albo restartem.

Oto przykład:

```
CALL init      ; wywołania procedury
               ; „inicjalizacja”
LD de,100      ; od tego miejsca kontynuowane
               ; będzie wykonywanie
               ; programu po wykonaniu RET
.
.
init
RET
```

Należy unikać następujących sytuacji:

```
               ; NIE RÓB TAK, CHYBA ŻE
               ; WIESZ CO ROBISZ
CALL procedura ; wywołanie procedury
RET
procedura
LD bc,230
PUSH bc        ; odłożenie na stosie wartości
               ; pary rejestrów BC
RET            ; a próba powrotu z procedury
               ; skończy się skokiem pod
               ; adres pamięci 230 NIE RÓB TAK,
               ; CHYBA ŻE WIESZ CO ROBISZ
```

Powyższy przykład prezentuje jak w najprostszy sposób zaimplementować procedurę, a następnie z niej wrócić. CALL jako parametr przyjmuje adres bezwzględny. I tu pomocne są etykiety które zwalniają nas z wyliczania tego adresu - wykona to za nas kompilator wykorzystując wspomniane etykiety.

JR przesunięcie_względne to skrót od „Jump Relative”, czyli skok bez powrotu, do adresu relatywnego. Relatywność polega na tym, że do adresu następnej instrukcji po JR, dodaje się wartość podaną jako argument do instrukcji. Przyjmuje wartości od -128 do 127. Stosując etykiety nie musimy o tym pamiętać. W przypadku kiedy skok miałby być dalszy niż dozwolone wartości, kompilator poinformuje nas o błędzie. JR ma zatem zastosowanie w krótkich procedurach:

```
nieDalejNiz127bajtow
LD a,10
ADD a,a
JR nieDalejNiz127bajtow
```

JP adres_bezwzględny to skrót od Jump. Przekazuje działanie do adresu wskazanego przez argument, bez możliwości łatwego powrotu za pomocą RET. Stosuje się go głównie tam, gdzie odległość jest zbyt duża dla JR:

```
JP 50000      ; skok do programu
               ; o podanym adresie
```

SKOKI WARUNKOWE

Instrukcje skoków do podprogramów oraz adresów relatywnych nabierają sensu kiedy możemy to zrobić warunkowo. Warunki zależą od stanu flag w rejestrze F, omawianym nieco wcześniej. Najczęściej sprawdza się stan flag Z i C. Składnia polecenia warunkowego jest następująca:

POLECENIE WARUNEK (, argumenty)

Wyjaśnienie znajduje się w poniższym przykładzie:

```
CALL Z, podprogram ; instrukcja wykona się
                   ; tylko wtedy, kiedy
                   ; flaga Z ma wartość 1
JP NZ, adres        ; instrukcja wykona się
                   ; tylko wtedy, kiedy
                   ; flaga Z ma wartość 0
JR C, adres         ; instrukcja wykona się
                   ; tylko wtedy, kiedy
                   ; flaga C ma wartość 1
RET NC              ; instrukcja wykona się
                   ; tylko wtedy, kiedy
                   ; flaga C ma wartość 0
```

Jak widać w przykładach warunek z „N” na początku sprawdza czy dana flaga (C,Z) ma wartość 0 a brak „N” sprawdza wartość 1.

Oczywiście, istnieją również WARUNKI oparte o inne flagi, ale nie są one już tak często używane, dlatego też pomijam je w niniejszym artykule.

PĘTLE

Są najczęściej wykorzystywaną konstrukcją w programach. Wykorzystuje się w nich skoki warunkowe. Najprostsza pętla znana z BASIC to:

```
FOR a = 0 TO 10 : NEXT a
```

Można zapisać w asm jako:

```
LD a,0
petla
INC a
CP 10          ; jeżeli liczby nie są równe
               ; flaga Z jest resetowana
               ; (ustawiana na 0)
JR NZ,petla
```

Z rejestrem B związana jest instrukcja DJNZ przesunięcie_względne wręcz stworzona do pętli! Jest to instrukcja złożona (a zatem jako całość wykonuje się znacznie szybciej niż każda instrukcja z osobna). Do rejestru B przypisuje się liczbę powtórzeń, co obrazuje przykład w BASICu:

```
1 LET a = 1
2 LET b = 10
3 LET a = a + b
4 LET b = b - 1
5 IF b<>0 THEN GO TO 3
```

to w asm wygląda to tak:

```
LD a,1
LD b,10
petla
ADD a,b
DJNZ petla
```

Należy pamiętać, aby długość procedury wewnątrz pętli nie przekraczała 128 bajtów.

W przypadku, kiedy chcemy więcej niż 255 powtórzeń, wykorzystujemy rejestry 16-bitowe. Konstrukcja typowej pętli wygląda następująco:

```
LD de,1000
```

```

petla _ 2
    DEC de          ; nie modyfikuje Z,
                   ; zatem trzeba „ręcznie”
                   ; sprawdzić czy
                   ; wartość rejestru DE jest zerowa

    LD a,d
    OR e            ; OR ustawi flagę Z
                   ; tylko wtedy jeśli wszystkie
                   ; bity A i E są zerowe
                   ; czyli A i E zawierają 0

    JR nz,petla _ 2

```

DYREKTYWY KOMPILATORA

Brakuje jeszcze znajomości kilku dyrektyw używanych przez kompilator. Najczęściej stosowane będą:

- **ciąg_znaków** - podany ciąg znaków nie będący słowem kluczowym traktuje jako etykietę, która może reprezentować adres pamięci
- **\$** - znak dolara - aktualny adres pamięci w kompilowanym programie
- **ORG adres** - określa adres, pod który program ma być skompilowany
- **DB wartość** - zapisuje podaną wartość jako 8-bitową [byte]
- **DW wartość** - zapisuje w pamięci wartość 16-bitową [word]
- **DEFM "ciąg znaków"** lub **DB "ciąg znaków"** - zapisuje napis [ciąg znaków] do pamięci.
- **label EQU wartość** - przypisuje etykietce label podaną wartość

PIERWSZY PROGRAM!

Z taką wiedzą możemy już napisać swój pierwszy program, który robi więcej niż typowy „hello world”. Napiszmy swoje pierwsze demo na ZX Spectrum! Na początek jednak minimum informacji na temat architektury komputera, dzięki czemu zrozumienie programu stanie się łatwiejsze. ZX Spectrum 48K ma „na pokładzie” 64KB pamięci. Pierwsze 16 kilobajtów zajmuje pamięć ROM - niemodyfikowalna, w której zapisany jest system oraz procedury pomocnicze, z których będziemy korzystać. Od adresu 16384 zaczyna się pamięć RAM. Obszar pomiędzy 16384 a 23295 zajmuje pamięć obrazu, przy czym od 22528 do 23295 przechowywane są atrybuty - informacje o kolorach. Za pamięcią obrazu aż do 23760 znajdują się różnego rodzaju obszary danych zarezerwowane przez system. Pamięć od 16384 aż do 32767 to obszar, do którego dostęp ma układ generujący obraz - ULA a dostęp do RAM jest współdzielony z procesorem. Współdzielony dostęp powoduje, że procesor (a co za tym idzie wykonywanie programu) jest wstrzymywany co jakiś czas, do czasu, kiedy ULA skończy swoją pracę. Pamięć powyżej 32768 to pamięć, do której dostęp ma tylko procesor. Jest to najlepsze miejsce dla naszego programu. Inny powód jest taki że łatwo zapamiętać jego początek, w zapisie szesnastkowym zapisuje się go jako \$8000.

```

    ORG 32768        ; adres kompilacji
                   ; adres pamięci
                   ; atrybutów jest stały,
                   ; dla wygody
                   ; stwórzmy etykietę,
                   ; w której zapamiętamy ten adres

    ATRYBUTY _ START EQU 22528
    start          ; etykieta mówiąca o miejscu,

```

```

                   ; w którym zaczyna się program.
                   ; Część inicjalizacyjna programu
                   ; inicjalizacja kanału dla
                   ; napisów, wykorzystujemy
                   ; do tego procedury z ROM.
                   ; Póki co nie zagłębiamy się
                   ; dokładnie jak i co jest robione

    LD a,2           ; 2 = górny ekran
    CALL 5633        ; wykonanie podprogramu
                   ; otwarcia kanału
                   ; wypełnijmy ekran
                   ; treścią - najlepiej jakiś napis

    LD b,37          ; powtórzmy 37 razy

```

```

petla _ tekstu
    PUSH bc          ; odłożenie na stosie
                   ; rejestru BC
                   ; w szczególności
                   ; aktualnej wartości
                   ; B - licznika powtórzeń

    CALL wypisz _ tekst ; wywołanie podprogramu
                   ; rysującego napis

    POP bc           ; przywrócenie wartości
                   ; BC ze stosu

    DJNZ petla _ tekstu ; powtórzmy operację
                   ; inicjalizacja kolorów,
                   ; zamalujmy cały ekran różnymi
                   ; kolorami tak aby
                   ; było co później modyfikować

    LD hl,ATRYBUTY _ START ; adres
                   ; początkowy atrybutów

    LD e,1           ; początkowy kolor
    LD c,24          ; liczba linii Y

```

```

init _ kol _ petla0
    LD a,e           ; kopiujemy aktualny kolor
    LD b,32          ; liczba powtórzeń równa ilości
                   ; znaków w poziomie

```

```

init _ kol _ petla1
    LD (hl),a        ; zapisanie koloru
                   ; pod wskazany adres

    INC hl           ; zwiększenie adresu
                   ; pamięci atrybutów

    INC a            ; zwiększenie numeru koloru
    AND 127          ; pozbycie się ostatniego
                   ; bitu FLASH.

    djnz init _ kol _ petla1 ; powtórzenie
                   ; operacji dla linii

    INC e            ; zwiększenie koloru początkowego
    DEC c            ; zmniejszenie licznika linii Y
                   ; DEC modyfikuje flagi
                   ; w rejestrze F

    JR nz,init _ kol _ petla0
                   ; i skok, jeżeli jeszcze nie
                   ; osiągnęliśmy wartości 0

```

```

; główna część naszego dema - migajmy kolorami
; w sposób profesjonalny!

```

```

glowna _ petla
    LD hl,ATRYBUTY _ START ; adres pamięci atrybutów
    LD bc,768             ; liczba powtórzeń - wielkość
                   ; obszaru atrybutów

```

```

petla_kolory
    LD a,(hl)      ; do A wartość koloru
    INC a          ; zwiększ o 1
    AND 127        ; obetnij FLASH, jeżeli kolor
                  ; osiągnie wartość 128
                  ; to po obcięciu, licznik
                  ; przyjmie wartość 0

    LD (hl),a      ; zapisanie nowej wartości
    INC hl         ; zwiększenie wskaźnika
                  ; pamięci atrybutów
    DEC bc         ; zmniejszenie licznika
    LD a,c          ;
    OR b           ; test na wartość 0
    JR nz,petla_kolory ; powtórz jeżeli nie 0
    HALT          ; oczekiwanie na przerwanie,
                  ; domyślnie jest IM 1
                  ; a jego obsługa
                  ; znajduje się w ROM

    JR glowna_petla ; przeskoczenie
                  ; do początku pętli głównej
                  ; procedura wypisująca
                  ; tekst na ekranie

wypisz_tekst
    LD de,tekst    ; adres początkowy tekstu do DE
    LD bc,koniec_tekstu - tekst
                  ; do BC - długość tekstu
                  ; jego wartość jest równa
                  ; końcowi pamięci z tekstem
                  ; minus jego początek.

    CALL 8252      ; wywołanie procedury w ROM
                  ; do „drukowania” ciągu znaków

    RET           ; powrót z podprogramu

tekst             ; etykieta, pod którą znajdzie
                  ; się adres początku tekstu

    DEFM "[####] KOLOR [####]" ; tekst ;)

koniec_tekstu    ; etykieta wskazująca
                  ; na koniec tekstu

    end start     ; dzięki temu działa
                  ; --tapbas w pasmo

```

W pierwszej części kursu czytelnik miał okazję zapoznać się z rejestrami, sposobem ich adresowania oraz instrukcjami przypisania/wymiany danych, podstawowymi działaniami na liczbach, realizacją pętli oraz skoków warunkowych i bezwarunkowych do podprogramów. To tylko tyle, lub też aż tyle, aby napisać prosty program który już coś robi.

W dalszych odcinkach kursu będzie mowa o kolejnych operacjach na danych ośmiobitowych takich jak: operacje na bitach, przesunięcia, operacjach na blokach danych, operacjach wejścia/wyjścia oraz rejestrach alternatywnych. Rozszerzymy również wiedzę na tematy już poruszane w tym odcinku, a których nie podałem aby nie przytłoczyć czytelnika.

LINKI

PASMO [HTTP://PASMO.SPECCY.ORG/](http://PASMO.SPECCY.ORG/)

ZX SPIN [HTTPS://SITES.GOOGLE.COM/SITE/ULAPLUS/HOME/ZX-SPIN-AND-BASIN](https://sites.google.com/site/ulaplus/home/zx-spin-and-basin)

FUSE [HTTP://FUSE-EMULATOR.SOURCEFORGE.NET/](http://FUSE-EMULATOR.SOURCEFORGE.NET/)

SPECYFIKACJA ZX SPECTRUM

[HTTPS://WWW.WORLDOFSPECTRUM.ORG/FAQ/REFERENCE/REFERENCE.HTM](https://www.worldofspectrum.org/faq/reference/reference.htm)



Tabela z podsumowaniem

REJESTRY 8-BIT	A, B, C, D, E, H, L, „(HL)”
REJESTR F	flaga Z (Zero) - 1 jeżeli wynik operacji wynosi 0, 0 jeśli różny od zera flaga C (Carry) - bit przeniesienia, ustawiany kiedy nastąpi nadmiar w działaniu arytmetycznym CCF SCF
REJESTRY 16-BIT	SP, IX, IY, BC, DE, HL
OPERACJE 8-BIT	dana = liczba 8bit, rejestr 8-bit LD rejestr, dana ADD A,dana ADC A,dana SUB dana SBC A,dana INC rejestr DEC rejestr AND dana OR dana XOR dana
OPERACJE 16-BIT	dana: LD rejestr 16bit, liczba ADD HL, rejestr 16bit SBC HL, rejestr 16bit INC rejestr 16bit DEC rejestr 16bit EX DE,HL
STOS	rejestr = BC,DE,HL,IX,IY rejestr2 = rejestr, AF LD SP, adres LD SP,ii gdzie ii=HL IX IY PUSH rejestr2 POP rejestr2
SKOKI	CALL adres CALL warunek, adres LD B,licznik : DJNZ przesunięcie_względne JP adres JP warunek, adres JR przesunięcie_względne JR warunek, przesunięcie_względne RET RET warunek

PIERWSZE KROKI SPECTRUMOWEGO KODERA

CZYLI ROBIMY WŁASNE INTRO NA KOLEJNE SPECCY.PL PARTY!

NA POCZĄTEK SPOSTRZEŻENIE:
SPECTRUMOWCY TO SZCZĘŚCIARZE!
W ŚWIECIE POPULARNYCH WŚRÓD
SCENOWCÓW MIKROKOMPUTERÓW
8-BIT NIE MA CHYBA INNEJ
TAK PROSTEJ I CZYTELNEJ, A JEDNAK
DOŚĆ MOCNEJ ARCHITEKTURY,
JAKĄ JEST ZX48K WRAZ Z AY'EM.

SACHY / WANTED TEAM ^ RESISTANCE ^ LAMERS

Zarówno Atari XL/XE jak i Commodorki wymagają wczytania się w ich układy specjalizowane (ANTIC/GTIA/POKEY na A8, VIC-II/SID na C64) oraz poznania ciekawych, ale nietrywialnych tricków programowych na DisplayList (A8) bądź cyklowania rejestrów \$D0xx na C64. Na ZX jest o wiele prościej, bo czegoś takiego robić nie trzeba. Do tego procesor 6502 (i 6510) choć popularny i niewątpliwie ciekawy – ubogi jest zarówno w użyteczne rejestry (sztuk 3), jak i taktowanie (Atari XE/XL 1.77MHz »PAL«, C64 niecały 1MHz, dokładnie 0.985MHz w wersji PAL). Przy takich konkurentach Z80 z 7 rejestrami 8-bit (plus zestaw alternatywny) i dwoma 16-bit (IX/IY) oraz taktowaniem 3.5MHz to spora bestia. Z doświadczenia głównie amigowego kodera, jak i prób zakodowania czegoś na A8/C64 mogę śmiało stwierdzić, że ZX48k+AY to najprostsza i najlepsza platforma, by rozpocząć karierę demoscenowego klepacza kodu.

W założeniu poniższy poradnik ma doprowadzić czytelnika do zakodowania pierwszej demoscenowej produkcji, z grafiką bitmapową, muzyką i jakimś typowym efektem na ekranie. Postaramy się też stworzyć podstawę pod kilkuczęściową „prodkę” (jest to na Speccy naprawdę proste) oraz lekko zsynchronizować z muzyką to, co dzieje się na ekranie. I wszystko to przy założeniu – jak najmniej szczegółów (które zazwyczaj



DEMO THX ZX! BY RESISTANCE SPECCY.PL PARTY 2018.1

zaciemniają sprawę), a jak najwięcej efektu. Nie zamierzam wyjaśniać dokładnie wszystkich instrukcji Z80 (bo sam znam ledwie część i ciągle myślę przypisane im rejestry), całego mechanizmu przerwań, czy możliwości narzędzi typu assembler pasmo. Po prostu będziemy stosować działające części programu, bez specjalnego wgłębiania się w nie – na to przydzie pora, kiedy już nabierzemy pewnego doświadczenia. Teraz byłoby to po prostu za trudne i przeszkadzałoby ogarnąć niezbędne podstawy (a o to w tym chodzi). Dlatego też będziemy wspomagać się dostępnymi narzędziami i opublikowanym w sieci kodem – czy to z różnych stron WWW, czy też ze skarbicy wiedzy forum speccy.pl. Tak naprawdę wszyscy koderzy zaczynają od odpalenia dostępnych przykładów i ich modyfikacji. Jest to najprostsza droga, by nie tylko zacząć robić coś sensownego i zaznajomić się z tematem, ale i po to, by nie zniechęcić się aktualnie bezużytecznymi detalami. Muszę przyznać, że ZX zainteresowałem się dopiero niedawno (kilka lat temu i to pobieżnie), a w czasach *Bajtka* przeskakiwałem *Klan Spectrum* czy *Amstrada*, ponieważ uważałem, że nie są mi potrzebne (w moim mieście był jeden Gumiak, część to XL/XE, czy C=, a resztę stanowiły same A500). Z80 po-

znałem również wyrywkowo – na potrzeby programowania... centrali alarmowej opartej o Hitachi HD64180 (protoplasta Z180 czyli Z80+MMU+IO). Niemniej jednak 2 maile od Tygrysa opisujące jak zasemblować .tap i jak zorganizowany jest ekran, pozwoliły mi w 2-3 tygodnie poskładać małe demko na Specy.Pi Party 2018.1 (<http://tiny.cc/fksniz>) i podobną drogą spróbuję przeprowadzić wszystkich czytających ten tekst. No to zaczynamy!

Od asemblera do pliku .tap

Skoro chcemy napisać program działający na ZX, to potrzebujemy narzędzia, które przetłumaczy napisany przez nas kod programu (z angielskiego zwany „źródłowym”, czyli początkowym/wyjściowym – tekstowy zapis programu) na numeryczną („binarną” – bitowo-bajtową) formę, którą to jedynie procesor Z80 może „zrozumieć” i wykonać. Scalak akceptuje wyłącznie ciągi kilku (zazwyczaj od 1 do 4) bajtów, które rozkłada sobie na bity i na ich podstawie wie co należy wykonać w danej chwili, skąd pobrać potrzebne dane i gdzie umieścić wynik. To wszystko jest dokładnie „wszyste w procesor”, to jest jego język i tak naprawdę to jest jedyne co potrafi nasz CPU – pobrać z pamięci kolejne bajty, rozpoznać co one znaczą i wykonać na tej podstawie zaplanowane wcześniej operacje. Tymi operacjami mogą być np. obliczenia (dodawanie, odejmowanie, operacje AND/OR/itp.) lub przenoszenie czy zmiana danych w pamięci (np. narysowanie czegoś na ekranie). Może też wysłać jakieś dane np. do układu AY by coś zagrać, albo odczytać stan portu joysticka, ale jest to w zasadzie to samo, co odczyt lub zapis pewnego adresu w pamięci, czyli miejsca w „przestrzeni adresowej” – do tego wrócimy później. Na razie jednak najważniejsze jest to, że narzędzie zwane asemblerem (znów z angielskiego „to assemble”, czyli składać coś razem/zbierać w całość/montować) odczyta sobie nasz plik tekstowy z programem źródłowym, przetłumaczy go instrukcja po instrukcji na bajty i z tych bajtów poskłada nasz docelowy plik wykonywalny, czyli nasze intro/demo/grę. Dodatkowo, na Spectrum nasz działający kod binarny (nasze „zasemblowane”, czyli przetłumaczone na bajty intro) wymaga małego programu w BASICu, który załaduje naszą produkcję gdzie trzeba do pamięci i uruchomi ją. Taki programik nazywamy loaderem. Jako, że będziemy używać popularnego na scenie ZX asemblera *Pasmo* – jesteśmy w dobrej sytuacji, bo nie dość, że przetłumaczy nasz kod i wygeneruje „binarkę”, to jeszcze sam doklei do naszego kodu odpowiedni loader. Tak więc za jednym zamachem otrzymamy w pełni gotowego do uruchomienia „exeka”. Całą sprawę załatwimy jedną instrukcją:

```
PASMO --TAPBAS ZINBO.ASM ZINBO.TAP
```

Należy pamiętać, by podać właściwą ścieżkę do naszego kodu źródłowego (tzw. „źródłówki”), by asembler *Pasmo* nie miał problemów ze znalezieniem go – w naszym przypadku będzie to plik „zinbo.asm”, w którym zapiszemy nasze intro. Dla ułatwienia zalecam na początku trzymać wszystkie pliki (assembler, kod źródłowy, później grafiki czy muzykę) w tym samym katalogu – nie będzie problemów ze ścieżkami do plików. Gdy powstanie gotowy „TAP”, możemy go od razu odpalić na emulatorze (w celach testowych) lub na „prawdziwym sprzęcie” (real hardware), czyli na naszym ulubionym Spectrumie (wiadomo, zalecane są ZX48k/+ z AY lub szary ZX+2). Emulator ma oczywiście ogromne plusy – łatwość i szybkość uruchamiania – np. asembtację i uruchamianie można spiąć sobie pod jedną opcją w ulubionym edytorze lub pliku .bat, można też łatwo debuggować kod w przypadku nieprawidłowo-

wego działania, itp. Minusem jednak jest pewna „niedokładność” emulacji – tylko prawdziwy HW prawdę nam pokaże, czy nasz kod działa w 100% poprawnie (timingi!), kolory się zgadzają, itp. Tego typu problemy zazwyczaj dotyczą już daleko bardziej zaawansowanych przypadków i na 99% emulator wystarczy do naszych potrzeb (zachęcam jednak do okazjonalnego testowania swojego kodu na prawdziwym sprzęcie, kodowanie dem to czasem dość niewdzięczne i zaskakujące zajęcie – lepiej wiedzieć wcześniej, że nasze długo pisane demo uruchomi się bez problemów na compo-maszynie na kolejnym świetnym party).

Skoro wiemy już jak „zbudować” nasz kod (do formy wykonywalnej na ZX) i uruchamiać (na emulatorze lub ZX Spectrum), czas wreszcie coś napisać. Zaczniemy od rzeczy bardzo prostych, ale już niezbędnych w naszej produkcji. Na początek najlepiej zrobić kilka ćwiczeń/testów/eksperymentów, by mieć pewność, że nasze „środowisko developerskie” (czyli narzędzia: edytory tekstu, asembler, emulator) działają sprawnie i bez problemów oraz zacząć tworzyć ogólną strukturę zapisu naszego projektu. W praktyce jest tak, że po napisaniu pierwszego w miarę kompletnego intra/dema, mamy już gotowy szablon do tworzenia kolejnych. Sporo elementów zostaje takich samych (player do muzyki, obsługa przerwań, procedurki czyszczenia ekranu, stawiania punktów, wyświetlania obrazków, itp), a inne po prostu zastępujemy nową treścią.

Na początek parę słów o pamięciach ROM/RAM i przestrzeni adresowej

Zanim skupimy się na kodzie, który coś robi, musimy wybrać miejsce w pamięci, w którym będzie on leżał i przygotować ogólną formę naszego projektu. Jak już wiemy – kod wykonywalny (czyli nasz program) to instrukcje Z80 w postaci bajtów (tak samo zresztą jak muzyka czy grafika – wszystko to bajty w pliku, czy w pamięci). Z tego wybranego przez nas miejsca w pamięci Z80 będzie pobierał kolejne bajty, rozpoznawał je i wykonywał. W tym miejscu będzie też przechowywał opracowane przez siebie dane (wyniki obliczeń), czy pobierał np. dane do wyświetlenia na ekranie. ZX Spectrum jest genialnie prosty pod tym względem. Jak wiemy ma 48kB RAM i 16kB ROM, czyli w sumie 64kB. Ta liczba nie jest przypadkowa. 64kB pamięci to oczywiście 64*1kB, czyli 64*1024 bajty – w sumie 65536 bajtów (wiem, trywialne, ale warto to rozpisać). Mamy więc 65536 1-bajtowych komórek pamięci ponumerowanych od 0 do 65535 (to w normalnym, „szkolnym” systemie dziesiętnym). Możemy też to samo zapisać w bardziej wygodny sposób, czyli w tzw. „heksie”, czyli kodzie „szesnastkowym” (heksadecymalnym) i nasz zakres możemy rozpisać od zera (0 = \$0000) do naszego dziesiętnego maxa: 65535 = \$FFFF. Wiem, że BASIC w Spectrum uwielbia korzystać z „normalnego” zapisu dziesiętnego, ale przy pisaniu w asemblerze lepiej sprawdza się ten „heksadecymalny”. To kwestia przyzwyczajenia (przyjdzie z czasem), ale czytelniej jest układać kolejne dane o długości 16, 32, 64, 128 czy 256 bajtów od adresu \$4000 czy \$FC00 niż od 16384 czy 64512 – a to dokładnie te same adresy komórek pamięci, tylko inaczej zapisane. Po prostu w złotych czasach Spectrum nie było jeszcze takich wygod i pożytecznych narzędzi jak... kalkulator z Windows, który można przełączyć w tryb programisty i który sam dokona nam zamiany DEC/HEX/BIN i to w każdą stronę, zależnie od potrzeb. Nikt już dziś nie przelicza tych systemów na kartce, inna sprawa, że po kilku użyciach najważniejsze adresy komórek w formie heksadecymalnej czy binarnej same zapadają w pamięć i nieraz łatwiej pamiętać ich heksadecymalną

formę niż „normalną dziesiętną“. Tutaj polecam uruchomienie wspomnianego kalkulatora w Windows, wybranie w menu wersji „Programisty“ (oczywiście!) i chwilę zabawy w przełączenie się na system BIN (a zależnie od wersji, nieraz kalkulator inteligentnie ograniczy nam możliwe cyferki do 0 i 1), wklepanie %10100110 i sprawdzenie jak to wygląda w DEC czy HEX. Tak samo, warto zmienić system na HEX i wbić \$00A6. Jak widać, to to samo. I w ten sposób w niecałą minutę opanowaliśmy magię podstaw asemblera – przeliczanie adresów komórek, czy wartości danych do wygodniejszych form. Łatwiej będzie ustawić np. 5. bit w rejestrze zapisując do niego wartość %00100000 (bity numerujemy od prawej strony i liczymy od zera, dlatego 5. bit jest... szóstym od prawej) niż liczbę dziesiętną 32. Przykładowo gdy będziemy chcieli dodatkowo ustawić bit 2. (%00000100), by uzyskać binarny „wzorek“ %00100100 – mniej wygodnie jest do 32 dodawać 4 i używać liczby 36, niż od razu rozpoznawalnego niemal graficznie, dość ładnego %00100100. A takie „binarne wzorki“ mogą przydać się nam np. do wypełnienia ekranu jakiś ciekawym szlaczkiem czy tłem pod efekt. Tutaj liczba w postaci %00100100 działa lepiej wizualnie na wyobraźnię i zapis jest czytelniejszy niż po prostu ładowanie do rejestru liczby 36, nieprawdaż? Zapewniam, że z czasem (dość szybko) przeliczanie HEX/BIN staje się proste i robi się to w pamięci, dzieląc każdy bajt na pół (po 4 bity), a te z kolei już zapamiętują się same. Kolejna korzyść jest taka, że gdy raz opanujemy sztukę konwersji systemów na kalkulatorze Windy – zastosujemy ją w każdym asemblerze, na każdy chyba procesor (jak również w językach C/C++ i pewnie wielu innych). Jeszcze na szybko spójrzmy na nasz obszar dostępnej pamięci (tzw. „przestrzeń adresowa“ czyli numeracja dostępnej pamięci ROM i RAM) to 0-65535 czyli \$0000-\$FFFF a więc (dzieląc dla ułatwienia 4-kami bitów) %0000-0000-0000-0000 do %1111-1111-1111-1111 i już widzimy, że wszystko to dokładnie to samo, i w zależności od tego jak nam się podoba możemy używać dowolnych form zapisu, a nawet mieszać je, dodawać, odejmować, katować binarnymi operacjami logicznymi – wszystko w zależności od potrzeb i własnego uznania. Kalkulatora z Windy nie należy się wstydzić, jest 21 wiek – skoro używamy peceta czy jabłka do kodowania na Spectrum, a emulatora do uruchamiania naszego kodu, to niech Windows przyda się również i w tym godnym celu.

Skoro wiemy już jak adresować (numerować) komórki w pamięci, czas poznać te najważniejsze i najprzydatniejsze adresy (jak pisałem na początku ZX48k jest genialny w swej prostocie i jest ich naprawdę kilka, w porównaniu do dziesiątek lub setek na innych komputerach). Trzeba będzie też poznać najciekawsze obszary pamięci, czyli następujące po sobie komórki pamięci, które do czegoś konkretnego służą (np. nasza ulubiona tzw. „pamięć ekranu“). Takie obszary opisujemy przez podanie jego początku i końca, czyli przez podanie adresu (numeru) jego pierwszej i ostatniej komórki w pamięci. Przykładem niech będzie tutaj pamięć ROM naszego ZX, w której mieści się cały BASIC, procedury restartu, tablice (w tym fonty, które widzimy pisząc na ekranie), obsługa ekranu, magnetofonu, drukarki, klawiatury, głośniczka, przerwań, kalkulator zmienoprzecinkowy, itp. To wszystko (i więcej) zajmuje pierwsze 16kB, czyli od adresu 0 do 16383, a czytelniej: \$0000-\$3FFF. Może \$3FFF (ostatni bajt ROM) nie wygląda teraz jeszcze super czytelnie, ale następujący po nim bajt to 16384 oznacza to samo co \$4000, a to z kolei początek pamięci RAM. \$4000 na pewno łatwiej zapamiętać niż wspomniany adres 16384. Co więcej, jeśli \$0000-\$3FFF to 16kB ROM a od \$4000 kolejne 48kB (do \$FFFF) to RAM (w pewnym uproszczeniu) zapiszę to jako ROM:\$0000-\$3FFF, RAM:\$4000-\$FFFF. Teraz mamy

już konkretny pogląd jak wygląda rozkład pamięci (architektura pamięci) w ZX Spectrum i cieszymy się jak klarowny i łatwy do zapamiętania on jest. Tym sposobem raczej nie popełnimy błędów, próbując zapisać coś do ROM, bo wiemy, że „poniżej“ \$4000 zapisać się nie da (bo to ROM), a „powyżej“ \$FFFF też nie, bo to jest już poza dostępną pamięcią (Z80 nie ogranicza adresu \$FFFF+1 – asembler zinterpretuje to jako przepełnienie licznika, czyli \$0000).

Dobra, wiemy już, że Z80 może zapisywać i odczytywać komórki pamięci od \$0000 do \$FFFF, wiemy, że od adresu zero do niemal \$4000 jest ROM (dokładnie do \$3FFF) i rzadko kiedy będziemy tam zaglądać (procedury z ROMu są przydatne, ale zazwyczaj za wolne do scenowego użytku). Gdzie więc można bezpiecznie coś poskładać (ang. assemble). Cóż, jest wiele miejsc, czasem można nawet coś schować w pamięci ekranu, ale na początek najlepszym miejscem będzie początek górnych 32kB, czyli połowa obszaru obsługiwanego przez procesor: adres 32768, czyli w czytelniejszym zapisie: \$8000. Prawda, że łatwiejsze do zapamiętania? Po prostu „połowa pamięci“, „górne 32kB“ (bo „dolne“ jest od \$0000 do \$8000-1 czyli \$7FFF). Mnożąc ten adres x2 dostajemy 65536, ale pamiętamy, że numerujemy adresy od zera, odejmując więc 1 mamy 65535 = \$FFFF, czyli koniec dostępnego obszaru adresowego Z80 i wszystko się zgadza. Być może nudne jest to, że powtarzam podobne sprawy kolejny raz, ale robię to specjalnie, by przyzwyczaić czytelnika do swobodnego operowania adresami w różnych systemach oraz pokazać, jak na ZX48k wszystko pięknie i prosto pasuje do siebie.

Powoli zmierzam już do konkretów, ale przed napisaniem „coś już robiącego“ kodu wymienię jeszcze tylko dwa ważne adresy. Pierwszy adres to początek pamięci ekranu, czyli początek takiego obszaru RAM, że gdy coś tam zapiszemy (dowolny bajt, czy nawet bit) to będzie to widać na ekranie (jako zapalony piksel lub kilka pikseli). Drugi adres to początek „bufora atrybutów ekranu“. Słowo, co to jest bufor: to taki obszar w pamięci wybrany przez inżyniera lub koder, żeby służył mu jako obszar roboczy. Czasem jest pod jedną konkretną rzecz, czasem jest to wygodne miejsce do wykonywania wielu różnych potrzebnych operacji – zazwyczaj przechowywane są tam i opracowywane (obliczane, zmieniane, kopiowane) jakieś akurat potrzebne dane. Koder sam definiuje sobie gdzie taki bufor umieści (gdzie będzie początek) i jak duży on będzie (od którego do którego adresu). Bufor jest w zasadzie pewnym pomysłem na wykorzystanie pamięci. Z kolei twórcy danego sprzętu (systemu, architektury) mogą zarezerwować konkretne obszary pamięci (pod zawsze tymi samymi, czyli stałymi adresami) na obszary robocze, czyli bufor dla drukarki, ekranu graficznego (nasza „pamięć ekranu“), czy atrybutów kolorów. Jak wiemy, ZX Spectrum ma zazwyczaj dostępne 2 kolory (kolor tła i kolor punktu) w widocznym obszarze ekranu w kwadraciku 8x8 pikseli. Te kwadraciki 8x8 pikseli leżą jeden za drugim, ułożone od lewego górnego rogu ekranu aż do dolnego prawego. Dzięki takiemu ułożeniu czasami atrybuty kolorów wykorzystuje się jako ogromne (8x8!) „piksele“ i realizuje na nich np. plazmy, duże scrollle i podobne efekty. Ich ogromną zaletą jest ułożenie ich właśnie w taki ciągły (jeden za drugim) sposób, bo już „normalne piksele“ (jak wiadomo jest ich 192 linie poziome po 256 pikseli w każdej) aż tak prosto ułożone nie są, ale do tego wrócimy innym razem. Na razie wykorzystamy wspomniane atrybuty kolorów jako „mega-piksele“ do nauki i eksperymentów oraz pierwszych efektów graficznych. Tak więc początek pamięci ekranu to adres \$4000 (dec: 16384), a początek pamięci/bufora atrybutów to adres \$5800 (dec: 22528). Jak widać heksadecymalne

adresy (te zaczynające się od znaku dolara '\$') są dość łatwe do zapamiętania, ale skoro są one zawsze stałe, można im przypisać bardziej czytelne „pisane” odpowiedniki. Ucząc się o ZX Spectrum zapisałem sobie je tak jak poniżej, dodając przydatne informacje, by nie szukać ich ponownie po dokumentacji, czy internecie:

```
GFX_SCR_START:
EQU $4000          ;16384: 2048*3, 6144 BYTES
```

```
GFX_SCR_ATTR:
EQU $5800          ;22528: 32X24, 768 BYTES
```

Na początku linii nadałem im czytelne dla mnie i łatwe do zapamiętania „pseudonimy”: GFX_SCR_START i GFX_SCR_ATTR ale każdy może nazwać je po swojemu (np. Ekran_Start_Adr i Attrib_Start_Adr). Nie ma to znaczenia, ważne by łatwo je zapamiętać i używać zamiast adresów liczbowych. Za „pseudonimami” widzimy komendę („dyrektywę”, czyli rozkaz dla narzędzia kompilatora/asemblera) „equ” i oznacza to, że od tej pory asembler gdy zobaczy nasz nadany „pseudonim” będzie rozumiał to tak samo, jakby był tam adres zapisany liczbowo. Dalej jest średnik, czyli znak, że od tego miejsca, do końca linii wszystko jest komentarzem – można tam pisać informacje wyjaśniające o co chodzi w danej linii programu, albo też inne dodatkowe informacje. W moim zapisie umieściłem sobie przypomnienie jak zbudowany („zorganizowany”) jest ekran i bufor atrybutów oraz ile zajmują. Te linie będę zawsze kopiował do każdego kolejnego kodu intra czy dema, wpisałem więc tam sobie takie „przypominaczki”, by zaoszczędzić ponownego szukania, gdy zechcę coś nowego zakodować na ZX po dłuższej przerwie. Można też nic tam nie wpisywać, jeśli linijka kodu jest w 100% oczywista (wtedy średnik też nie jest potrzebny).

Skoro mamy już jakieś fajne i użyteczne definicje, dodajmy do nich też definicję początkowego adresu naszego programu. Ustaliliśmy, że dobrym miejscem na początek naszego programu jest adres \$8000. To ważny adres i używany co najmniej 2 razy. Nasz kod z biegiem czasu rozrośnie się do setek, a nawet tysięcy linii. Narzędzie asembლujące chce wiedzieć gdzie nasz program ma się zacząć i gdzie skończyć, żeby nie musiał się przepracowywać (wykonywać niepotrzebną i nadmiarową pracę tłumaczenia kodu). Dlatego dodałem sobie etykietę (czyli „pseudonim” adresu) określającą mój początek przyszłego kodu (wybrany adres początkowy), dodałem go również do wcześniejszej listy definicji adresów:

```
GFX_SCR_START:    EQU $4000    ; 16384:
                                   ; 2048*3,
                                   ; 6144 BYTES
GFX_SCR_ATTR:     EQU $5800    ; 22528:
                                   ; 32X24,
                                   ; 768 BYTES
DEMO_CODE_MEM:    EQU $8000    ; MY DEMO
                                   ; STARTS HERE
```

To są pomocne rzeczy opisujące co gdzie jest w pamięci, jednak na razie nie produkują one jeszcze żadnego kodu wykonywalnego (żadnego bajtu w pamięci) – to są pomocnicze dyrektywy, które ułatwią nam pracę z własnym kodem w przyszłości. Teraz jednak wykorzystamy zdefiniowany właśnie adres początku naszego kodu do faktycznego umieszczenia czegoś w pamięci (pierwszego rozkazu procesora Z80) oraz do zbudowania pierwszej poprawnej i w pełni wartościowej struktury naszego wykonywalnego programu:

```

;-----
GFX_SCR_START:    EQU $4000    ; 16384:
                                   ; 2048*3,
                                   ; 6144 BYTES
GFX_SCR_ATTR:     EQU $5800    ; 22528:
                                   ; 32X24,
                                   ; 768 BYTES
DEMO_CODE_MEM:    EQU $8000    ; MY DEMO
                                   ; STARTS HERE
ORG DEMO_CODE_MEM
                                RET
END DEMO_CODE_MEM
;-----

```

Kopiujemy linie zawarte między obszarami wyznaczonymi liniami ;----- do pliku o nazwie „zinbo.asm” i zapisujemy ten plik w tym samym folderze co nasz asembler „pasm0.exe”. Teraz uruchamiamy „dosową linię poleceń”, czyli stary dobry „cmd.exe”, przechodzimy w nim do folderu, w którym znajduje się pasmo.exe i nasz „zinbo.asm” (ZINBO – tak nazwiemy nasze intro) oraz wydajemy poznaną wcześniej komendę (u mnie wygląda to tak):

```
PASMO --TAPBAS ZINBO.ASM ZINBO.TAP
```

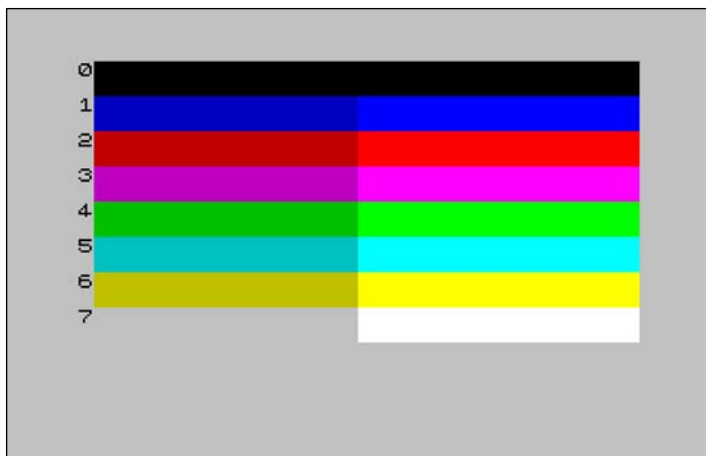
Jeśli pojawią się błędy to asembler poda nam, w której linii – trzeba to naprawić (może coś źle zapisaaliśmy lub skopiowaliśmy). Jeśli nic nie napisze to oznacza, że wszystko jest ok. Przechodzimy do naszego folderu i tam powinien znajdować się plik „zinbo.tap”. Mój ma 122 bajty, z czego jeden bajt to właściwy już rozkaz procesora Z80: „ret” (od RETURN) a reszta to loader BASICa do załadowania i uruchomienia naszego programu. Uruchamiamy dowolny emulator (polecam ZX Spin) ładujemy nasz „zinbo.tap” i... niby nic się nie dzieje (pozornie). Jednak na górze widzimy typową nazwę uruchamianego na Spectrum pliku (tutaj „Bytes: zinbo.tap”), na dole „0 OK, 40:1”, czyli nic nie wybuchło, a zatem pełen (choć mało spektakularny) sukces. Żeby jednak nie ściemniać, że to fajne, dodajmy coś sensownego, czyli coś widocznego na ekranie. Do podanego wcześniej kodu dopiszmy parę rozkazów jak poniżej, zapiszmy kod, zasembლujmy (ta sama komenda w cmd.exe) i odpalmy w emulatorze. I co?

```

;-----
GFX_SCR_START:    EQU $4000    ; 16384:
                                   ; 2048*3,
                                   ; 6144 BYTES
GFX_SCR_ATTR:     EQU $5800    ; 22528:
                                   ; 32X24,
                                   ; 768 BYTES
DEMO_CODE_MEM:    EQU $8000    ; MY DEMO
                                   ; STARTS HERE
ORG DEMO_CODE_MEM
DEMO_START:
                                LD A,%10010001
                                LD (GFX_SCR_ATTR),A
DEMO_STOP:
                                RET
END DEMO_CODE_MEM
;-----

```

Niby nic, a jednak – mamy nie tylko działający poprawnie program napisany w asemblerze Z80 na ZX Spectrum, ale i pierwszy efekt wizualny – a stąd jak wiadomo prosta droga



na szczyty demosceny. No może nie taka prosta i nie od razu na szczyty, ale pierwszy krok został zrobiony i teraz należy po prostu dodawać kolejne. Wracając jednak do kodu – w pierwszym programie dyrektywy (komendy dla narzędzia jakim jest assembler) „ORG adres” i „END adres” utworzyły nam strukturę programu wykonywalnego, informując assembler pod jakim adresem ma zacząć układać wykonywalny kod dla procesora Z80 i gdzie ten kod zakończyć. Zdefiniowany adres początku programu w tym przykładzie zastąpiliśmy definicją DEMO_CODE_MEM. Korzystać z tego taka, że jeśli z jakichś powodów chcielibyśmy, by nasz kod leżał w pamięci np. pod adresem \$C000, to wystarczy tylko w jednym miejscu zmienić wartość \$8000 na \$C000 i sprawa załatwiona, bez konieczności poprawiania obu miejsc (ORG i END). To jednak nadal są tylko dyrektywy, czyli pomocnicze wskazówki dla narzędzia asambującego. Tak naprawdę jedyną instrukcją wykonywalną w naszym pierwszym programie był „ret” (RETURN), czyli powrót z podprogramu, a tym podprogramem był dla „systemu ZX Spectrum” – właśnie nasz mały programik. Tak więc ZX Spectrum uruchomił nasz (pod)program, ale że jedyną instrukcją (choć ważną i poważną) był RET, to nasz program sam się zakończył (bo nic innego nie miał wykonać) i elegancko wrócił do programu nadrzędnego, czyli tego, który go uruchomił (tutaj: do BASICa), oddając mu możliwość działania. Kolejna wersja naszego programu to już konkretne wykorzystanie architektury sprzętowej ZX48k, czyli działanie na jego możliwościach graficznych. Etykieta DEMO_START to tylko tradycyjny zapis porządkowy – by ładnie zacząć. Kolejne dwie instrukcje „LD” (od „LOAD”) to rozkazy procesora Z80, służące to przetrzymywania danych. Czasem dane przetrzymuje się między rejestrami, czasem między pamięcią a rejestrem, a czasem poprzez załadowanie rejestru daną z pamięci i przetrzymanie tej danej do innego miejsca w pamięci – realizuje się przesyłanie danych między dwoma adresami. Tutaj pierwsze LD nakazuje procesorowi najpierw załadowanie liczby (wartości) %10010001 do jego własnego rejestru wewnętrznego nazywanego A – od Accumulator, bo rejestr ten jest najczęściej wykorzystywany w wielu operacjach, a spora grupa z nich może być wykonana tylko z rejestrem A. Z80 ma też inne rejestry (B, C, D, E, H, L, F i nie tylko), ale to właśnie Akumulator jest tym „najważniejszym” i najbardziej użytecznym. Tak więc pierwszy rozkaz LD załadował liczbę (wartość) do rejestru A i na tym zakończył działanie. Procesor pobrał więc z pamięci następną w kolejności instrukcję, a ta kazała mu liczbę z rejestru A przepisać do pamięci pod adresem GFX_SCR_ATTR (= \$5800), czyli do pierwszego bajtu bufora atrybutów ekranu. W momencie, kiedy wartość %10010001 znalazła się w tym bajcie, na ekranie pojawił się kwadrat 8x8 pikseli w czerwonym (lub niebieskim) kolorze. Skąd te kolory? Otóż kolory wynikają z tego w jaki

sposób ułożone są bity w każdym bajcie atrybutów kolorów i co one oznaczają dla układu ULA, który na podstawie tych bitów wyświetla obraz i jego kolory. Spectrum ma ustaloną przez swoich twórców i niezmienną paletę kolorów. Jest ich niby 16, ale w zasadzie to 15, a tak naprawdę to 8... plus 7. Skąd taki bałagan? Cóż, kwestia interpretacji pewnych faktów. Jest bowiem 8 rozróżnialnych kolorów podstawowych i najlepiej pokazuje je obrazek (◀).

Podstawowe 8 kolorów ma również swój jaśniejszy odpowiednik, czyli wersję „bright”. Czym różni się wersja bright od podstawowej najlepiej pokazuje przypadek koloru o numerze 7. W wersji podstawowej kolorem o numerze 7 jest kolor szary, a w wersji „rozjaśnionej” jest to kolor biały. Licząc 8 kolorów podstawowych + 8 rozjaśnionych możemy powiedzieć, że Spectrum może wyświetlić 16 kolorów. Jesteśmy blisko prawdy, jednak kolor czarny (numer 0) wygląda tak samo w trybie normalnym i jasnym, stąd więc bardziej prawdziwa liczba to 15. Ta liczba ma jeszcze dodatkowe ograniczenia (1 z 8 kolorów do wyboru w polu 8x8), ale żeby to zrozumieć, trzeba przyrzeć się dokładnie pojedynczemu bajtowi Atrybutu Koloru. Poniższy obrazek pokazuje dokładnie jakie znaczenie mają wszystkie bity w takim bajcie:

7	6	5	4	3	2	1	0
F	B	P2	P1	P0	I2	I1	I0

- BIT F** – URUCHAMIA FLASH MODE, CZYLI MIGOTANIE POLA
- BIT B** – WYBIERA BRIGHTNESS, CZYLI JAŚNIEJSZĄ WERSJĘ KOLORÓW W POLU
- BITY P2 DO P0** – BITY NUMERU KOLORU DLA PAPER, CZYLI KOLORU TŁA
- BITY I2 DO I0** – BITY NUMERU KOLORU DLA INK, CZYLI RYSOWANYCH PIKSELI W TYM POLU

Od prawej strony widzimy dwie grupy po 3 bity każda. Grupa I (INK) to kolor pisaka, czyli w tym kolorze pojawi się postawiony w naszym polu 8x8 dowolny punkt. Grupa P (PAPER) to nic innego jak kolor tła. Jeśli nasze pole 8x8 będzie wyczyszczone z pikseli (wszystkie bity w tym obszarze będą ustawione na zero) to całe pole będzie w kolorze tła. Oczywiście od razu widać, że ustawiając ten sam numer koloru dla grupy I i P – dostaniemy kolor tła, niezależnie od tego, czy wszystkie piksele będą wyczyszczone (bity w obszarze wyzerowane), czy nie. Bez sensu? Niekoniecznie. Czasem korzysta się z tego triku, by ukryć zawartość obszaru ekranu i odstąpić go w odpowiedniej chwili lub w odpowiednio ciekawy sposób. Zawartość pola 8x8 można więc wypełnić wcześniej (np. kawałkiem obrazka), a dopiero w odpowiednim momencie do bajtu atrybutu koloru należy wpisać właściwą wartość koloru pikseli i staną się one widoczne – bardzo przydatna sprawa. Co to jest „numer koloru”? Tu sprawa też jest dość prosta. Grupy bitów dla INK i PAPER mogą zawierać binarny zapis liczb od 0 do 7, bo na 3. bitach zmieszczą się tylko takie liczby (%000=0, %001=1, %010=2, %011=3, %100=4, %101=5, %110=6, %111=7). A teraz spójrzmy na wcześniejszy obrazek przedstawiający wszystkie dostępne standardowe kolory. Po lewej stronie widzimy liczby 0-7 i są to numery konkretnych kolorów. Tu widać już związek między grupami bitów INK i PAPER. Gdy w miejsca tych bitów wstawimy konkretne numery, czyli przyporządkowane kolorom trójki bitów (od %000 do %111) – nasze pole 8x8 będzie miało kolor tła w kolorze takim jak bity P2,P1,P0 a każdy postawiony w nim punkt – w kolorze zapisanym na bitach I2,I1,I0. Wiedząc to, warto wrócić do naszego przykładu i pozmienić wartości bitów w linijce kodu:

```
LD A,%10010001
```

np. na:

```
LD A,%10100110 ;ROZBIJAJAC NA BITY: %10,  
;%100=4, CZYLI ZIELONY,  
;%110=6, CZYLI ZOLTY
```

Odpalamy nowy kod, obserwujemy różnice, robimy własne eksperymenty i wiemy już o co chodzi. Pozostały do wyjaśnienia 2 pozostałe bity, te najbardziej z lewej: FLASH i BRIGHTNESS. Ten pierwszy powoduje efekt migania (ang. flash), czasowo zamieniając miejscami kolory tła i pikseli, a bit BRIGHTNESS (jasność) powoduje, że kolory w naszym polu 8x8 będą brane z 2. kolumny naszej rozpiski kolorów, czyli właśnie z tego jaśniejszego zestawu kolorów. Proponuję spróbować odpalić jeszcze raz nasz kod zmieniając wartości tych 2. bitów. Gdy to zrobimy, mamy już pełną wiedzę jak na ZX Spectrum wyświetla się kolory i pora na kilka eksperymentów z kodem. Umówmy się, że podane poniżej przykłady będzie wklejać pomiędzy etykiety DEMO_START i DEMO_STOP, to zaoszczędzi nam przepisywania części kodu, który na razie się nie zmienia. Ok, podmieńmy kod na poniższy:

DEMO_START:

```
LD A,%10010001  
LD (GFX_SCR_ATTR),A ;TOP-LEFT  
LD A,%01100110  
LD (GFX_SCR_ATTR+32+3),A ;BRIGHTNESS  
;FLAG ON  
  
LD A,%10100110  
LD (GFX_SCR_ATTR+32+4),A ;FLASH  
;FLAG ON
```

DEMO_STOP:

Odpalamy i co widzimy? Nasz pierwszy zestaw instrukcji (LD/LD) wpisuje wartość bajtu atrybutu koloru w początek bufora atrybutów, czym ustawia migający kwadracik 8x8 w lewym górnym rogu ekranu. Kolejne dwie instrukcje LD robią coś podobnego, tyle że do początku bufora atrybutów dodają +32+3. Co to jest? Otóż jest to prosty i wygodny sposób obliczania adresu takiego bajtu atrybutu, do którego dokładnie chcemy coś wpisać. Z budowy ekranu i jego bufora atrybutów wiemy, że każdy bajt opisuje pole 8x8, czyli na takie pola podzielony jest ekran, jeśli chodzi o możliwość ustawienia tam kolorów. Rozdzielczość ZXa to 256 punktów w poziomie i 192 w pionie. Skoro bajt koloru opisuje pole 8x8, to ilość pikseli w poziomie, jak i w pionie dzielimy przez 8. Daje nam to 32 pola w poziomie ($256/8=32$) i 24 w pionie ($192/8=24$). Mamy więc „ekran atrybutowy” o rozdzielczości 32x24. Jeżeli chcemy ustawić bajt atrybutu w 2. „linii ekranu atrybutowego”, to wystarczy pominać pierwsze 32 bajty bufora i znajdujemy się na początku szukanej 2. linii, w jej pierwszym bajcie. Jeżeli chcemy znaleźć któryś z 32 „kwadratów kolorów” w tej linii, to wystarczy do tego dodać liczbę, której wartość wskaże nam, o który bajt od lewej strony ekranu nam chodzi. Należy więc dodać liczbę z zakresu 0-31 (bo pamiętamy, że numerujemy wszystko od zera), a dodając 32 do początku bufora jesteśmy w zerowym bajcie drugiej linii. Co się stanie jeśli dodamy 32 lub 33? Po prostu trafimy na początkowe bajty kolejnej „linii ekranu atrybutowego” (polecam sprawdzić samemu). Tu jednak najważniejsze jest żeby zobaczyć, jak proste jest posługiwanie się adresem zapisanym pod GFX_SCR_ATTR jako początkiem, a dalej można już łatwo odszukiwać konkretną pozycję dodając pozycję Y

(przemnożoną przez 32) i X (już w pojedynczych bajtach) do adresu początkowego bufora atrybutów. W taki właśnie sposób postawimy nasze kwadraciki zarówno na końcu bufora, jak i w jego środku:

DEMO_START:

```
LD A,%10010001  
LD (GFX_SCR_ATTR),A ;TOP-LEFT  
LD A,%01100110  
LD (GFX_SCR_ATTR+32+3),A ;BRIGHTNESS  
;FLAG ON  
  
LD A,%10100110  
LD (GFX_SCR_ATTR+32+4),A ;FLASH  
;FLAG ON  
  
LD A,%10010001  
LD (GFX_SCR_ATTR+32*23+31),A ;BOTTOM-RIGHT  
LD A,%10010001  
LD (GFX_SCR_ATTR+32*11+15),A ;MIDDLE POINT
```

MY_PAUSE:

JP MY_PAUSE

DEMO_STOP:

Jak widać, żeby obliczyć dokładne pozycje, posłużyłem się normalnym zapisem działań matematycznych oraz systemem dziesiętnym – bo tak było mi łatwiej i jest czytelniej, niż gdybym zapisał wszystko heksadecymalnie. Program asemblerujący sam wykonuje zapisane działania zanim przetłumaczy kod i wiedząc to możemy łatwo obliczać potrzebne wartości czy adresy – posługując się zapisem szkolnej matematyki w obrębie czterech działań. Dodatkowo, daje to naprawdę o wiele czytelniejszy kod. Wracając do takiego zapisu po kilku tygodniach od razu załapiemy o co nam chodziło, w przeciwieństwie np. do zapisów takich jak poniżej:

```
LD (23295),A  
LD ($5AFF),A  
LD (GFX_SCR_ATTR+767),A  
LD ($5800+$2FF),A
```

Wszystkie te zapisy oznaczają to samo, czyli:

```
LD (GFX_SCR_ATTR + 32*23+31),A ;BOTTOM-  
;RIGHT
```

Pamiętajmy, że wszystko liczymy od „offsetu zero”, czyli od odległości „zera bajtów od początku”, bo pierwszy bajt każdej linii leży w zerowej odległości od lewego brzegu ekranu, a pierwsza „linia atrybutowa” leży w zerowej odległości od zerowego bajtu bufora atrybutów, a więc pod „zerowym offsetem” od początku bufora. Offset – to odległość lub inaczej przesunięcie np. w bajtach od jakiegoś początku. Tu to początek bufora atrybutów, w innym przypadku np. w tablicy wartości sinusów, itp. To po prostu „odstęp od początku”. Jeśli wynosi zero, to jest dokładnie taki sam jak adres początkowy, z tym, że później łatwiej jest podawać np. offset +5, offset +31, offset \$255, niż dodane już konkretne wartości. Później wszystko będzie jasne, gdy zaczniemy działać w pętlach na buforach, teraz zapamiętajmy, że jest to po prostu odległość od początku.

Wracając do przykładu, wyjaśnić trzeba jeszcze dwie pominięte dotąd rzeczy. Fragment kodu:

```
LD A,%01100110  
LD (GFX_SCR_ATTR+32+3),A ;BRIGHTNESS  
;FLAG ON
```



```
LD A,%10100110
LD (GFX_SCR_ATTR+32+4),A      ;FLASH
                                ;FLAG ON
```

pokazuje użycie flag (czyli inaczej bitów) jasności i migania w bajcie atrybutu koloru. Tutaj jako offset od początku bufora atrybutów wybrałem specjalnie napis „Bytes“ by (na literkach „es“) pokazać zarówno kolor INK, jak i PAPER oraz dwie wspomniane flagi. Pierwsza para instrukcji pokazuje użycie bitu BRIGHTNESS, czyli te same kolory wyświetlane są w wersji jaśniejszej. Druga para pokazuje jak działa bit FLASH, z wyłączonym BRIGHTNESS i specjalnie ułożyłem je obok siebie, by łatwo to można było zobaczyć. Oczywiście nic nie stoi na przeszkodzie, by włączyć zarówno BRIGHTNESS jak i FLASH dla tego samego obszaru 8x8, obie flagi działają niezależnie od siebie, polecam poeksperymentować. Ostatnią rzeczą do wyjaśnienia z załączonego przykładu jest mała pętla:

```
MY_PAUSE:
JP MY_PAUSE
```

Jako, że po skończeniu naszego programu i po etykiecie „DEMO_STOP“ występuje rozkaz RET – nasz program po wykonaniu się oddaje kontrolę do środowiska BASICa. Ten z kolei informuje nas, że wszystko wykonało się poprawnie, wypisując:

```
0 OK 40:1
```

i tym tekstem zamazuje nasz ostatni obszar 8x8, w którym stawialiśmy kolorowy kwadracik w prawym dolnym rogu ekranu. Aby dało się go jednak obejrzeć, w tym wyjątkowym przypadku uniemożliwiłem naszemu programowi wykonanie instrukcji RET, poprzez wprowadzenie go w „niekończącą się pętlę“. Żeby to zrobić, nadałem mu etykietę MY_PAUSE i rozkazem JP (od JUMP) nakazałem mu skakać do... samego siebie. Procesor Z80 widzi moją etykietę MY_PAUSE jako adres w pamięci i w tej chwili zupełnie jest nieistotne jaki – grunt, że jest on rozpoznawalny dla procesora i stały. Instrukcja skoku może skoczyć w dowolny obszar pamięci, czyli pod dowolny adres (tu określony przez etykietę MY_PAUSE podaną za rozkazem JP) i nakazać Z80 wykonywanie kolejnych bajtów kodu właśnie spod tego wskazanego adresu. Nasz przykład jest prosty, ale w innym przypadku mogła by to być np. kolejna część dema do uruchomienia. Tutaj jednak nakazałem rozkazowi JP skakanie pod własny adres, czyli pod adres w pamięci, gdzie leży ten właśnie rozkaz JUMP. Dzięki temu Z80 będzie kręcił się w kółko (aż do resetu lub nadejścia przerwania), a my możemy spokojnie obejrzeć dolną część ekranu – nie nadpisaną przez BASIC. Ten trick przydaje się częściej niż się wydaje, polecam jego zapamiętanie.

Powyżej omówiliśmy w zasadzie wszystko co trzeba wiedzieć o atrybutach kolorów. Jest to całkiem fajny mechanizm i wykorzystując go można wiele zdziałać na Spectrum, np. wyświetlając w ciekawszy sposób obrazki, tworząc scrollle (tak!), plazmy i inne kolorowe efekty, aż po... ukrywanie części danych potrzebnych później w pamięci ekranu.

Rozkaz HALT, czyli prosta synchronizacja wyświetlania ekranu

Na wszystkich znanych mi komputerach 8/16-bit, żeby wyświetlić coś poprawnie na ekranie, trzeba zsynchronizować się z wyświetlaniem na ekranie, czyli z wiązką elektronów strzelającą w kineskop naszego ulubionego telewizora RUBIN... Hm, no może nie do końca jest to prawda, ale zasada pozostała ta

sama. Ekran wyświetla się od górnego lewego rogu – do prawego dolnego. Gdy to się dokona, następuje „czas powrotu“ naszego niby działka elektronowego do góry, by zacząć wyświetlanie od nowa. Sprawa jest taka, że gdy będziemy zmieniali zawartość ekranu w czasie jego wyświetlania – powstaną widoczne „błędy“. Aby tego uniknąć, trzeba jakoś ominąć nałożenie się w czasie i miejscu rysowania i wyświetlania. Sposobem, żeby to zrobić jest modyfikacja zawartości ekranu albo przed rozpoczęciem wyświetlania danego fragmentu, albo po jego wyświetleniu. Oczywiście są techniki podwójnego buforowania, gdy na jednym fragmencie pamięci rysujemy nowy ekran, a w tym czasie wyświetlamy inny fragment pamięci, jednak na ZX48 bufor ekranu jest jeden i w ustalonym miejscu i trzeba sobie z tym jakoś radzić. Można oczywiście wszystko tworzyć na boku (w innym obszarze pamięci) i przepisywać gotowca do pamięci ekranu, ale nie dość, że zabiera to sporo pamięci, to i samo przepisanie nowej zawartości zabiera bardzo dużo czasu procesora i zazwyczaj mija się z celem. My spróbujemy po prostu unikać konfliktów z wyświetlaniem i rozkaz HALT bardzo nam w tym pomoże. HALT to instrukcja, która poniekąd zatrzymuje procesor (a dokładniej wykonuje instrukcje NOP), aż do nadejścia przerwania lub resetu. Brrr... przerwania – czyli wieje grozą?! Nie, na szczęście nie tutaj. W tym odcinku nie będziemy za dużo zajmowali się przerwaniem, a kiedy przyjdzie czas – okaże się, że wystarczy skorzystać z „gotowca“ i przerwania będą pracować dla nas (np. grając muzykę) w zasadzie w sposób niezauważalny. Tutaj jednak chodzi nam o synchronizację z wyświetlaniem i wystarczy nam do tego 2-3 instrukcje. HALT – wstrzyma nam wykonanie naszego programu aż do czasu, gdy przyjdzie przerwanie, np. przerwanie od układu ULA, które pojawia się co „ramkę“. Co to jest „ramka“? Otóż w uproszczeniu jest to „czas na wyświetlanie ekranu“, „czas do wykorzystania przez program, by skonstruować kolejny ekran“, itp. Termin „ramka“ pochodzi oczywiście od angielskiego „frame“. Telewizyjny system PAL wyświetla obraz 50 razy na sekundę i jeśli w każdym z kolejno wyświetlanych obrazach przesuniemy np. postać w grze o jeden piksel w lewo – nasze oko zarejestruje płynny ruch postaci w grze. To coś podobnego do kolejnych klatek animacji, z tym, że my nie chcemy mieć gotowych narysowanych klatek, a sami programowo je stworzyć. Jeżeli zrobimy to 50 razy na sekundę – będziemy mieli wrażenie płynnego ruchu: scrolla, duszka, plazmy, itp. Taka pojedyncza klatka nazywana jest właśnie ramką, w którą wpisujemy naszą zawartość. Jeśli rysując nową klatkę wyrobimy się w „czasie ramki“ – wszystko na ekranie będzie płynne i eleganckie. Jeśli nasz program będzie działał zbyt długo – Spectrum zacznie wyświetlać już kolejną i nastąpi „zerwanie ramki“ – będziemy widzieć jakieś niechciane „śmieci“, czy przesunięcia na ekranie. W Spectrum każdy początek nowej ramki „zwiastowany“ jest przerwaniem z układu ULA. I to właśnie przerwanie uruchomi nam ponownie Z80 zatrzymany przez instrukcję HALT. Dalej nasz kod będzie się wykonywał (rysował po ekranie), aż wykona co zaplanowaliśmy i znów zatrzymamy go instrukcją HALT, która przytrzyma procesor, aż kolejne przerwanie pchnie nam Z80 do przodu i tak w kółko... Dzięki temu mamy tanią i wydajną synchronizację z wyświetlaniem, możemy bezpiecznie działać na ekranie (o ile wyrobimy się w ramce) i mieć płynne efekty w naszym demku – o to właśnie chodziło. Aby jednak przerwanie mogło wybudzić procesor z instrukcji HALT, musimy jednorazowo upewnić się, że przerwania są odblokowane. Służy do tego instrukcja EI (ang. Enable Interrupts), która włącza przerwania wyłączając instrukcją DI (ang. Disable Interrupts). Skoro nie wiemy, czy przerwania są aktualnie włączone – dajemy EI i po kłopotcie.

Sprawdźmy to na przykładzie.

```
DEMO_STOP:
EI                ; ENABLE INTERRUPTS
LD A,%10010001    ; LET A HAVE SOME
                  ; COLOR ATTRIBUTE
                  ; BYTE
LD HL,GFX_SCR_ATTR ; COLOR ATTRIBUTE
                  ; BUFFER START

MY_SYNCHRO:
HALT              ; HALT - LET'S WAIT
                  ; TO START A NEW
                  ; FRAME

LD (HL),A         ; PUT ATTRIBUTE BYTE FROM
                  ; REG. A TO
                  ; THE ATTRIBUTE BUFFER
INC HL            ; +1 TO GET THE NEXT
                  ; ATTRIBUTE BYTE ADDRESS
JP MY_SYNCHRO     ; JUMP TO HALT,
                  ; TO WAIT FOR START OF
                  ; A NEXT FRAME

DEMO_STOP:
```

Na początku upewniamy się, że dzięki rozkazowi EI przerwanie na pewno obudzi Z80 ze stanu czekania, w które wprowadzi go co ramkę rozkaz HALT. Następnie przygotowujemy sobie wszystko do działania naszej pętli. Do rejestru A wpisujemy znaną z poprzednich przykładów zawartość bajtu atrybutu kolorów. Robimy to w tym miejscu, przed wejściem w pętlę, żeby zainicjować potrzebną nam wartość i by nie powtarzać tej czynności w środku pętli. Następnie do pary rejestrów HL (czyli do 8-bitowego rejestru H i również 8-bitowego rejestru L) wpisujemy adres początku bufora atrybutów. Z80 pozwala zapisywać niektóre rejestry 8-bitowe jako pary – od razu pełnym, 16-bitowym adresem. Takie pary to B i C (jako BC), D i E (jako DE) oraz H i L (jako HL). Dlatego adres GFX_SCR_ATTR czyli \$5800 wrzucamy jedną instrukcją do „połączonego” rejestru HL i od tej pory możemy działać na HL, by zaadresować dokładnie dowolny punkt w pamięci ZX Spectrum. Skoro przygotowania skończone, przyjrzymy się właściwej pętli (czyli MY_SYNCHRO). Rozkaz HALT poczeka na początek ramki i gdy nadejdzie przerwanie – puści Z80 do wykonania kolejnych instrukcji. Następny rozkaz LD prześle bajt z rejestru A do miejsca w pamięci, które aktualnie wskazuje para HL. Jak pamiętamy, HL wskazuje na początek bufora atrybutów kolorów, a więc na pole 8x8 w lewym górnym rogu ekranu. Wpisując tam bajt, wyświetlimy w tym miejscu kolorowy kwadracik. Kolejna instrukcja to INC (ang. Increment) i służy do zwiększania o 1 zawartości używanego z nią rejestru. Tutaj jest to HL, więc do HL zostanie dodane 1 ($\text{new_HL} = \text{old_HL} + 1$). Jaki jest cel tej instrukcji? Otóż tutaj przygotowujemy sobie już dane do kolejnego wykonania naszej pętli. Dodając 1 do HL przesuwamy adres w HL na kolejny bajt, czyli drugie od lewej pole 8x8. Robimy to, by kolejne wykonanie pętli (po kolejnym wykonaniu HALT) wpisało wartość bajtu z kolorami do kolejnej lokacji w buforze atrybutów kolorów. Tak więc INC HL pracuje nam tutaj niejako „na zapas”, czy „do przodu”. Tak robi się bardzo często, to po prostu wygodna forma tworzenia pętli w assemblerze i warto to zapamiętać. Po rozkazie INC następuje JP, czyli właściwy rozkaz „zapętlaający” nasz program, skaczący do HALT, gdzie cała pętla zaczyna się od nowa. Po uruchomieniu tego kodu zobaczymy jak punkt (8x8) po punkcie (8x8) wypełnia się cały ekran. To, że możemy zobaczyć jak wypełnia się cały ekran zawdzięczamy właśnie rozkazowi HALT i naszej synchronizacji. Dzięki

temu w ciągu jednej sekundy wykona się 50 pętli i 50 punktów zostanie wypełnionych. Cały ekran to 768 pól do wypełnienia (32×24), więc dzięki HALT ekran wypełni się po ok. 15 sekundach (bo $768/50=14.76$). Gdyby nie było HALT, procesor Z80 wypełniłby wszystkie pola w ciągu jednej ramki, czyli za szybko dla naszego oka. Polecam wyłączenie HALT i spróbowanie samemu oraz wpisywanie różnych wartości do rejestru A – zmieniać to bajt atrybutu na inny, a więc kolejne pole pojawi się w innym kolorze. Najprostszy przykład to zamienić rejestr A w instrukcji „LD (HL),A” na L, czyli „LD (HL),L”. Wykorzystajmy ten pomysł i dodatkowo narysujmy coś na ekranie – tym razem jakieś piksele. Aby to zrobić, wykorzystamy drugą parę rejestrów (BC) i tam zamieścimy adres początku nie bufora atrybutów kolorów, a już właściwego bufora ekranu. Jak pamiętamy – cokolwiek wpisujemy do obszaru pamięci zaczynającego się pod tym adresem (i dużego na 6144 bajty) – zostanie pokazane na ekranie. Zaczniemy od rysowania pikseli na ekranie:

```
DEMO_START:
EI                ; ENABLE INTERRUPTS
LD A,%10110111    ; LET A HAVE SOME
                  ; BITMAP PATTERN
LD BC,GFX_SCR_START ; PIXEL SCREEN
                  ; BUFFER START

MY_SYNCHRO:
HALT              ; HALT - LET'S WAIT
                  ; TO START A NEW
                  ; FRAME
LD (BC),A         ; PUT A BYTE FROM
                  ; REG. A TO THE
                  ; SCREEN
INC BC            ; +1 TO GET THE NEXT
                  ; SCREEN BYTE ADDRESS
JP MY_SYNCHRO     ; JUMP TO HALT, TO
                  ; WAIT FOR THE NEXT
                  ; START OF A FRAME

DEMO_STOP:
```

W tym przykładzie do rejestru A wpisuję bajcik zawierający jakiś wzorek binarny, który chcę wyświetlić w każdym bajcie ekranu (polecam spróbowanie innych, np. \$ff czy \$6F). Dalej, podobnie jak w poprzednim przykładzie, wykorzystuję parę rejestrów BC, by umieścić tam adres bufora, ale tym razem bufora ekranu (\$4000). Dalej pętla działa analogicznie, jak w przykładzie z atrybutami kolorów – adresy w BC się zwiększają, kolejne wartości trafiają w kolejne bajty ekranu. Przy okazji warto zaobserwować trochę dziwne wyświetlanie. Chociaż zapisujemy kolejne bajty (bo korzystamy z INC BC), to po zapisaniu całej pierwszej linii, zamiast zapisywać drugą – nasze bajty pojawiają się 8 linii niżej. To niestety wada ZX Spectrum, tak z pewnych względów skonstruowany jest ekran Specca i trzeba będzie się do tego przyzwyczaić. Na razie warto tylko o tym wspomnieć, w detalach zajmiemy się w kolejnym odcinku. Tutaj jednak postaramy się połączyć dwa powyższe przykłady i wykonamy obie operacje zapisu (do bufora kolorów i na ekran) niemal równolegle, czyli w sposób, w jaki często w demach w tym samym czasie na ekranie leci scroll, rusza się logo a na środku ekranu wiją się sinusowe plot'y. Może nasz przykład jest bardzo prosty, ale pokazuje sposób, w jaki może to być zrealizowane.

```
DEMO_START:
EI                ; ENABLE INTERRUPTS
LD A,%10110111    ; LET A HAVE SOME
                  ; BITMAP PATTERN
```

```

LD BC,GFX_SCR_START ; PIXEL SCREEN
                        ; BUFFER START
LD HL,GFX_SCR_ATTR   ; COLOR ATTRIBUTE
                        ; BUFFER START
MY_SYNCHRO:
HALT                  ; HALT - LET'S WAIT
                        ; FOR START OF
                        ; A NEXT FRAME

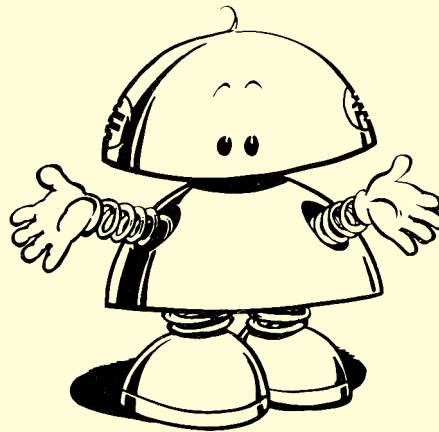
; DRAW PIXELS ON THE SCREEN
LD (BC),A             ; PUT A BYTE FROM REG.
                        ; A TO THE SCREEN
INC BC                ; +1 TO GET THE NEXT
                        ; SCREEN BYTE ADDRESS
                        ; USE ATTRIBUTES
LD (HL),L             ; PUT A FAKE ATTR.
                        ; BYTE FROM REG. L
                        ; TO THE ATTRUBUTE BUFFER
INC HL                ; +1 TO GET THE
                        ; NEXT ATTRIBUTE
                        ; BYTE ADDRESS
JP MY_SYNCHRO         ; JUMP TO HALT, TO
                        ; WAIT FOR START OF
                        ; A NEXT FRAME

DEMO_STOP:

```

Jak widać, po przygotowaniu sobie adresów buforów przed wejściem w pętlę, najpierw rysuję piksele na ekranie a potem, niezależnie od tej pierwszej czynności, wypełniam bufor atrybutów. Podzieliłem więc zadania do wykonania na niezależne bloki (małe w przykładzie, ale w pełni funkcjonalne) i wykorzystując jedną pętlę realizuję zadania jakby dla dwóch. Jako że obie części zdążą wykonać się w tej samej ramce – wykonają się jakby w tym samym czasie i „równolegle” (takie powstaje złudzenie, choć tak naprawdę wykonywane są przecież jedno po drugim). Specjalnie użyłem też osobnych grup rejestrów, gdyż utrzymywanie potrzebnych danych w rejestrach w kolejnych obrotach pętli to dobry sposób na przyspieszanie kodu, ale rzadko kiedy to się uda (można to jednak z góry zaplanować). Zazwyczaj trzeba rejestry zapisywać w pamięci, zrzucić na stos lub korzystać z alternatywnego zestawu rejestrów, jednak to rzeczy do nauczania się później. Na razie, tak skonstruowany program pozwoli czytelnikowi zacząć samodzielne próby z kodem Z80 i ekranem ZX Spectrum po to, by opanować wyświetlanie zarówno pikseli jak i kolorów, a jak wiadomo – w demach o to głównie przecież chodzi!

Tym przykładem zakończę ten artykuł. Jeśli spodobał się, pojawiły się pytania, a kontynuacja kursu na tym poziomie ma dla kogoś sens – proszę o uwagi na mój adres mailowy (sachozol@op.pl). Polecam jak najwięcej własnych eksperymentów z kodem, tworzenie własnych pętli i wyrzucanie ich wyników na ekran. Właśnie graficzny sposób sprawdzania wyników swojego kodu daje najszybsze rezultaty i najwięcej satysfakcji (bez konieczności nauki obsługi debbugera, itp). Kursów samego asemblera Z80 jest mnóstwo, mam nadzieję że tym tekstem pomogłem zasypać przepaść (jaka zazwyczaj jawi się początkującemu koderowi) między suchą nauką asemblera a „namacalnymi” wynikami pracy programu w postaci nawet tak minimalnych efektów graficznych jak powyżej. Praktycznie każdy wielki koder zaczynał od tego typu „wprawek” i przejście tego etapu nauki jest po prostu krokiem w stronę rzeczy bardziej zaawansowanych. Kiedy jednak pozna się te podstawy, kolejne kroki będą łatwiejsze.



SAM BASIC

CZYLI BASIC DOSKONAŁY CZĘŚĆ 1

SIR DAVID

Jedną z cech odróżniających komputery szesnastobitowe od ich ośmiobitowych poprzedników był brak w ich ROMie interpretera języka BASIC. Nie wiem, co było powodem zmiany podejścia producentów do tej kwestii, bo wszystkie popularne mikrokomputery ośmiobitowe miały jakiś (lepszy lub gorszy) BASIC. Być może chodziło o uproszczenie obsługi dla mniej „technicznych” użytkowników, na wzór konsoli do gier? Zainteressowani zawsze mogli wgrać sobie język programowania z zewnątrz. Komputery szesnastobitowe miały szybsze pamięci zewnętrzne, np. wbudowane stacje dyskiek, więc język programowania można było załadować w miarę szybko. Tyle, że to już nie to samo. Wbudowany interpreter zmuszał użytkownika do wpisywania komend z klawiatury, choćby do załadowania programu (w ZX spectrum za pomocą LOAD ""), a to już mogło stanowić impuls do zainteresowania się, jak brzmią inne komendy i jak to wszystko działa. Nie zdziwiłbym się, gdyby okazało się, że procent użytkowników np. Amigi



▲ GRA STRATEGICZNA "ORE WARZ II", NAPISANA W CAŁOŚCI W BASICU.

próbujących programowania był mniejszy, niż w przypadku osób zaczynających swoją przygodę z komputerami w czasach ośmiobitowych.

Wbudowane interpretery języka BASIC na różnych platformach potrafiły się dość mocno różnić. Język ogólnie był

```

9900 DEF PROC SETUP
9905 DIM BG$(8,8): DIM CH$(8,8): DIM P$(4,4,131): DIM SP$(4,21)
9910 LET AD=90163: FOR Z=1 TO 4: FOR F=1 TO 3: LET P$(Z,F)=MEM$(AD TO AD+130),AD=AD+
    131: NEXT F: NEXT Z
9915 FOR Z=1 TO 4: LET P$(Z,4)=MEM$(96623 TO 96753): NEXT Z
9920 LET AD=96754: FOR Z=1 TO 3: LET SP$(Z)=MEM$(AD TO AD+20),AD=AD+21: NEXT Z
9925 LET SP$(4)=MEM$(97874 TO 97894)
9990 END PROC
10000 DEF PROC SCREENON
10005 LET AD=SCR1,AD1=SCR2
10010 FOR Z=1 TO 24: POKE AD, MEM$(AD1 TO AD1+511)
10015 LET AD1=AD1+1024,AD=AD+1024
10020 NEXT Z
10025 LET AD1=AD1-512,AD=AD-512
10030 FOR Z=1 TO 24: POKE AD, MEM$(AD1 TO AD1+511)
10035 LET AD1=AD1-1024,AD=AD-1024
10040 NEXT Z: SCREEN 2: CLS: SCREEN 1
10045 END PROC
10050 DEF PROC MARSET

```

```

9900 DEF PROC SETUP
9905 DIM BG$(8,8)
    DIM CH$(8,8)
    DIM P$(4,4,131)
    DIM SP$(4,21)
9910 LET AD=90163
    FOR Z=1 TO 4
        FOR F=1 TO 3
            LET P$(Z,F)=MEM$(AD TO AD+130),AD=AD+131
        NEXT F
    NEXT Z
9915 FOR Z=1 TO 4
    LET P$(Z,4)=MEM$(96623 TO 96753)
NEXT Z
9920 LET AD=96754
    FOR Z=1 TO 3
        LET SP$(Z)=MEM$(AD TO AD+20),AD=AD+21
    NEXT Z
9925 LET SP$(4)=MEM$(97874 TO 97894)

```

▲ FRAGMENT KODU GRY "TWISTER" WYŚWIETLONY W FORMACIE LIST FORMAT 0 (Z LEWEJ) I LIST FORMAT 2 (Z PRAWEJ).

ten sam, ale liczba dostępnych rozkazów, ich składnia (szczególnie w przypadku operacji taśmowo/dyskowych) i obsługa edytora już nie, co potrafiło znacząco utrudnić przesiadkę między różnymi maszynami. Chyba najbardziej specyficzny edytor miały komputery z rodziny Sinclaira. Wszystkie pozostałe posiadały edytory ekranowe, czyli pisać dało się w dowolnym miejscu ekranu, natomiast ZX Spectrum (oraz jego poprzednicy i klony) – edytor liniowy. Tu ekran jest podzielony na listing programu na górze i linię edycji na dole. Dodatkowo wpisywanie komend nie odbywa się litera po literze, a wszystkie komendy wprowadzane są w całości, po naciśnięciu jednego klawisza lub kombinacji klawiszy. Ma to na pewno swoje zalety, jak brak błędów w pisowni czy też podgląd na wszystkie dostępne komendy. Ma też wady, bo taki zestaw musiał zostać ograniczony po to, aby dało się go przypisać określonej liczbie klawiszy. Z tego powodu rozszerzenia języka, jak na przykład Betabasic, zmieniają sposób wprowadzania komend

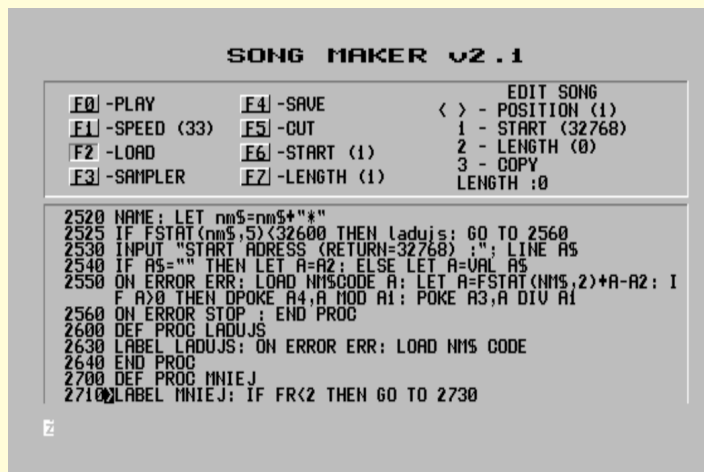
siadka ze Spectrum na SAMa jest dziecinnie prosta – dostajemy podobny (ale znacznie lepszy!) edytor liniowy i tę samą składnię wszystkich znanych ze Spectrum komend. Ich ilość i możliwości wydają się wręcz oszałamiające i nieograniczone. W efekcie powstała spora liczba ciekawych gier i programów napisanych właśnie w SAM BASICu, a ich jakość i szybkość działania potrafi zaskoczyć. Według mnie to właśnie SAM BASIC jest najlepszym wyborem dla każdego, kto lubi programować w BASICu, zna już nieco ten z ZX Spectrum i chciałby spróbować czegoś więcej.

SAM Coupé posiada znakomitą instrukcję użytkownika („SAM Coupé Users' Manual”), gdzie w bardzo przyjemny i przystępny sposób opisany jest między innymi jego BASIC. Dostępna jest jedynie w języku angielskim, jednak nawet osoby z podstawową znajomością tego języka powinny dać radę. Ja poznawałem SAM BASIC w czasach, gdy dopiero zaczynałem się uczyć angielskiego i mimo to szybko zacząłem pisać długie i dość zaawansowane programy. Korzystając z własnych doświadczeń, postaram się przybliżyć najciekawsze możliwości tego języka.

W pierwszej części artykułu zajmiemy się edytorem. Ogólnie jest to dobrze znany ze Spectrum edytor liniowy. Piszemy na dole ekranu i po naciśnięciu RETURN polecenie wykonuje się lub jeżeli nadamy numer linii, ta przeskoczy do górnej części ekranu jako fragment programu. Podstawowa różnica jest taka, że komendy trzeba wpisywać po literce (choć jest kilka, które można wywołać w całości kombinacją klawiszy, np. naciśnięcie SYMBOL+P napisze PRINT). Dla uproszczenia i przyspieszenia pisania, komendy zawierające spację można pisać jednym ciągiem, np. GOTO zamiast GO TO. Po wprowadzeniu linii do programu brakujące spacje zostaną dodane. Chcąc przywołać linię do edycji, należy nacisnąć klawisz EDIT (w emulatorze na PC to prawy Alt). Można przywołać od razu konkretną linię, wpisując jej numer i wtedy naciskając EDIT. Po edytowanej linii można poruszać się we wszystkich kierunkach, również góra/dół (czego nie dało się zrobić w Spectrum). Nie muszę chyba mówić, że to znacznie przyspiesza edycję długich linii.

ZX Spectrum nie posiada chyba żadnych komend wspomagających pracę edytora. W SAMie jest ich oczywiście wiele:

AUTO – wydanie tego polecenia rozpocznie automatyczną numerację linii pisanego programu. Samo AUTO bez parametrów spowoduje rozpoczęcie numeracji z parametrami domyślnymi, czyli od linii 10 z krokiem również 10. Jeżeli istnieją już jakieś linie, to numerowanie zacznie się od numeru bieżącej linii wskazywanej przez znacznik plus 10. Dodanie jednego parametru, np. AUTO 150 rozpocznie numerację od wskazanego



▲ JEŻELI ZATRZYMYMY DZIAŁANIE PROGRAMU KORZYSTAJĄCEGO Z OKIEN TEKSTOWYCH, TO LISTING POJAWI SIĘ JEDYNNIE W BIEŻĄCYM OKNIE.

na literowy. Edytor liniowy również ma swoje wady, gdyż wprowadzanie lub edycja długich linii, zajmujących kilka wierszy, spowalnia go w bardzo znaczącym stopniu.

Jeżeli chodzi o możliwości samego języka, czyli liczbę i rodzaj dostępnych komend, to z najpopularniejszej czwórki: ZX Spectrum, Commodore, Atari, Amstrad, zdecydowanie wyróżniał się ten ostatni. Dopiero SAM Coupé, który pojawił się parę lat później, potrafił mu dorównać. SAM BASIC to w rzeczywistości rozbudowany Betabasic z ZX Spectrum, z którego się wywodzi za sprawą tego samego autora. Dzięki temu prze-

numeru. Dodanie dwóch parametrów, czyli np. AUTO 150,20 zmieni dodatkowo krok na 20. Chcąc zatrzymać automatyczną numerację, należy wydać jakiegokolwiek polecenie bez numeru linii, np. STOP.

RENUM – zmiana numeracji linii w istniejącym programie. Analogicznie jak w AUTO, wydanie polecenia RENUM bez parametrów zmieni numerację linii w całym programie zaczynając od 10 z krokiem 10. Przy okazji uaktualnione zostaną wszystkie komendy odwołujące się do numerów linii, czyli np. GO TO, GO SUB, RUN, czy RESTORE. Komenda RENUM ma kilka możliwych parametrów i tak np. RENUM 20 TO 50 zmieni numerację tylko części programu, o numerach od 20 do 50. W dalszym ciągu zacznie od linii 10 z krokiem 10. Chcąc zmienić krok, należy użyć parametru STEP n, natomiast początek zmienia się parametrem LINE n. W pełnej postaci, ze wszystkimi możliwymi parametrami komenda może wyglądać np. tak: RENUM 20 TO 50 LINE 5 STEP 15. Należy pamiętać, że zmieniając numerację tylko części programu, nowe numery linii muszą zmieścić się w tym samym miejscu programu, co dotychczasowe. Nie można w ten sposób przenieść części programu w inne miejsce.

DELETE – usuwanie części programu. Domyślny format to przykładowo DELETE 10 TO 50 – usuwa linie od 10 do 50. Parametry można też pomijać, i tak np. DELETE TO 100 usunie wszystko od początku do linii 100, DELETE 100 TO usunie wszystko od linii 100 do końca, a DELETE TO usunie cały program. W odróżnieniu od komendy NEW, DELETE TO zachowa jednak wszystkie obecne w pamięci zmienne.

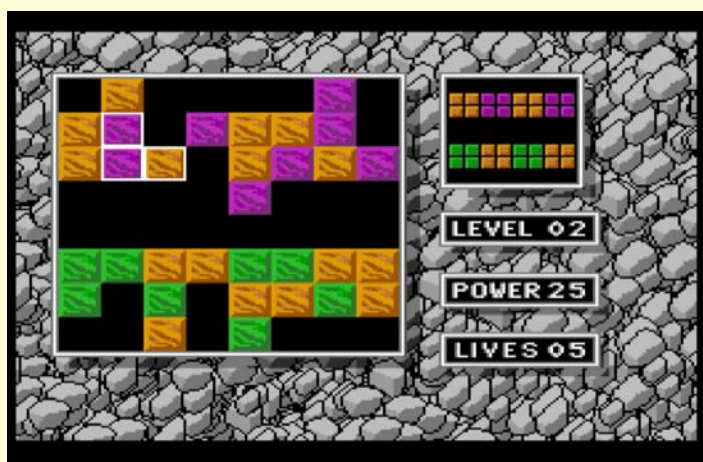
LIST – w normalnym użyciu komenda ta wyświetla kod



▲ GRA LOGICZNA „SLYDER” ZOSTAŁA NIEMAL W CAŁOŚCI NAPISANA W BASICU. JEDYNYM WYJĄTKIEM JEST ODTWARZACZ MUZYKI, SWOJĄ DROGĄ BARDZO DOBREJ.

programu. Jeżeli jest wywołana z parametrem, np. LIST 100, to wyświetla kod od wskazanej linii. Tak jest w ZX Spectrum, w SAMie można dodatkowo wpisać numer, do którego ma być wyświetlony kod, czyli np. LIST 100 TO 200 albo LIST TO 200 (od początku do linii 200). Jednak w SAMie komenda LIST ma dodatkowe zastosowanie – może być użyta do zmiany sposobu wyświetlania całego kodu programu.

LIST FORMAT 1 (albo 2) – po wydaniu komendy w tej postaci wygląd całego programu ulegnie diametralnej zmianie w ten sposób, że wszystkie polecenia w linii, które dotąd były oddzielone dwukropkiem, będą teraz widoczne w kolejnych liniach (ale bez dodatkowych numerów linii). Dwukropki oddzielające komendy nie będą wyświetlane, a pisząc nową linię, w momencie wpisania dwukropka będziemy przenoszeni do kolejnej linii. Te dwukropki w rzeczywistości tam będą, ale nie będzie ich widać. Dodatkowo wszelkie pętle (np. FOR), procedury (np. DEF PROC) czy warunki (np. IF) będą wy-



▲ GRA LOGICZNA „TWISTER”, TO KOLEJNA GRA NAPISANA W BASICU Z MUZYKĄ ODTWARZANĄ W KODZIE MASZYNOWYM.

świetlane w ten sposób, że linie w ich „środku” będą miały dodatkowe wcięcia z lewej strony. Parametr 1 lub 2 oznacza ilość spacji tego wcięcia. Wszystko to będzie się działo automatycznie, znacznie poprawiając czytelność programu. Chcąc przywrócić wygląd standardowy, należy wydać polecenie LIST FORMAT 0.

CSIZE szerokość, wysokość – wygląd kodu programu można też zmodyfikować zmieniając wielkość czcionek. Jak można się domyślić, powyższa komenda definiuje rozmiar znaku w punktach. Jednak wbrew oczekiwaniom nie umożliwia płynnego skalowania wielkości czcionek. Szerokość może mieć jedynie wartość 8 lub 6, ale wartość 6 ma wpływ na szerokość znaku jedynie w trybie graficznym 3, czyli o wysokiej rozdzielczości. Wysokość natomiast może mieć wartości od 6 do 32, jednak powoduje jedynie dodanie dodatkowych pustych linii pomiędzy wierszami. Wartości od 16 do 32 dodatkowo podwójnie zwiększają wysokości znaku.

WINDOW lewo, prawo, góra, dół – jest ostatnią komendą, którą można zmienić wygląd edytora (choć bardziej przydatną w kodzie programu). Służy ona do utworzenia okna tekstowego o wielkości mniejszej niż cały ekran. Parametrami są tu numery skrajnych kolumn i wierszy nowego okna. Od momentu wydania tej komendy tekst może być pisany tylko w utworzonym oknie. Lewy górny róg okna ma od tej chwili współrzędne tekstowe 0,0.

Jaki ma to wpływ na edytor? Otóż wprowadzenie komendy WINDOW w trakcie pisania programu spowoduje, że listing będzie ograniczony do zdefiniowanego okna. Ponadto, jeżeli zatrzymamy działanie programu korzystającego z okien tekstowych, to listing pojawi się jedynie w bieżącym oknie, a wszystko poza nim pozostanie nietknięte. Aby przywrócić stan początkowy wystarczy wydać komendę WINDOW bez parametrów.

Okno tekstowe nie ogranicza działania instrukcji graficznych (PLOT, DRAW itd), które w dalszym ciągu operują na całym ekranie. Rozkaz CLS również czyści cały ekran, chyba że będzie wydany z parametrem różnym niż 0, wtedy wyczyści tylko okno tekstowe.

Na tym zakończę opis edytora. Na podstawie własnych doświadczeń mogę z czystym sumieniem powiedzieć, że poznanie możliwości SAM BASICa to fascynująca przygoda dla użytkownika ZX Spectrum, do której gorąco zachęcam. ■

PRZYDATNE LINKI:

WWW.SIMCOUPE.ORG – EMULATOR SIMCOUPE

FTP.NVG.NTNU.NO/PUB/SAM-COUPE/DOCS/MANUALS/ – DOKUMENTACJA SAMA, W TYM USERS' MANUAL



ENTERPRISE

**MIKROKOMPUTER ENTERPRISE JEST W POLSCE NIEMAL
NIEZNANY. SKONSTRUOWANY W WIELKIEJ BRYTANII PRZEZ FIRMĘ
INTELLIGENT SOFTWARE ZNACZNIE PRZEWYŻSZAŁ SWOIMI
MOŻLIWOŚCIAMI ISTNIEJĄCE WÓWCZAS KOMPUTERY DOMOWE.**

PEAR

Konstrukcja komputera opiera się na dwóch układach zaprojektowanych specjalnie dla niego, nazwanych od imion ich twórców Nick i Dave (Nick Toop i Dave Woodfield). Pierwszy z układów obsługuje grafikę, zaś drugi dźwięk i pamięć.

Prototyp po raz pierwszy został zaprezentowany prasie 14 września 1983 roku, ale przez kolejnych kilkanaście miesięcy nie trafił do klientów z powodu problemów z jakością i opóźnień w dostawach układów ASIC.

Ostatecznie Enterprise 64 pojawił się w sklepach dopiero na 10 dni przed gwiazdką roku 1984. Model 128 trafił do sprzedaży w maju 1985 roku. Do tego czasu konkurencja zdążyła już podzielić między siebie rynek komputerów domowych. To opóźnienie było bezpośrednią przyczyną upadku firmy.

Wyprodukowano 80 tys. komputerów znanych w Wielkiej Brytanii, Francji, Hiszpanii, Holandii pod nazwami Samurai, Oscar, Elan, Flan, a ostatecznie Enterprise. W Niemczech występował również pod nazwą Mephisto. To kolejny powód, który ograniczał rozpoznawalność marki.

W maju 1987 roku partia 20 tys. sztuk trafiła na Węgry dystrybuowana przez firmę Videoton i to tam właśnie Enterprise był i nadal jest najbardziej popularny.

Wygodna klawiatura produkowana w standardach QWERTY i QWERTZ, posiada charakterystyczny joystick zamiast klawiszy kursora oraz 8 klawiszy funkcyjnych. Z tyłu komputera znajduje się duży przycisk Reset oraz wyłącznik zasilania.

Komputer wyposażono w procesor Z80A taktowany zegarem 4 MHz. W 32 KB ROM znajdują się system operacyjny EXOS oraz edytor tekstu. Interpreter BASIC znajduje się w kartridżu z dodat-

kowym 16 KB ROM. Na płycie głównej zainstalowane jest 64 KB pamięci RAM. W modelu 128 dodatkowy RAM zamontowany jest jako moduł wewnątrz obudowy.

Pamięć w Enterprise podzielona jest na bloki po 16 KB każdy. Stronicowaniem pamięci zajmuje się układ Dave i maksymalnie może obsłużyć 256 stron co daje możliwość zaadresowania do 4 MB.

Nick oferuje 2 tryby tekstowe 40 i 80 kolumn w wierszu, 2 tryby graficzne 640 i 320 pikseli w linii oraz tryb grafiki atrybutowej. Maksymalna rozdzielczość wynosi 640x512 pikseli. W najniższej rozdzielczości 80x256 dostępnych jest 256 kolorów. W wyższych rozdzielczościach można korzystać z palet 16, 4 lub 2 kolorów.

Najciekawszą właściwością jest możliwość podzielenia ekranu na sekcje i praca na każdej z nich w różnych trybach graficznych. Rozdzielczość pionowa każdej sekcji może być większa niż wyświetlane na ekranie okno.

Układ Dave udostępnia 3-kanałowy generator dźwięku z funkcjami obwiedni, filtrów, modulacji pierścieniowej oraz distortion. Czwarty kanał to generator szumu. Każdy z kanałów można dowolnie przypisać do kanałów stereo (lewy, prawy lub oba).

Do współpracy ze światem zewnętrznym komputer został wyposażony w złącze krawędziowe, port kartridża, dwa porty joysticków, interfejs drukarki Centronics, interfejs szeregowy RS-423, wyjście RF dla telewizora, wyjście RGB dla monitora oraz złącza do współpracy z magnetofonem. Niestety, poza magnetofonem, wszystkie pozostałe gniazda to złącza krawędziowe. Aby podłączyć jakiegokolwiek

osprzęt potrzebny jest oryginalny kabel lub odpowiednia przejściówka.

Podczas kampanii reklamowej producent obiecywał szeroki wybór akcesoriów dla Enterprise. Niestety z powodu kłopotów finansowych w sprzedaży pojawiło się tylko kilka z nich, inne nie wyszły poza etap prototypu, a nawet poza etap projektu. Dostępne w sprzedaży były interfejsy EXDOS dla napędów dyskietek 3,5" oraz 5,25" oraz stacje dyskietek z wbudowanym interfejsem, podłączane na sztywno do złącza krawędziowego, znajdującego się z prawej strony komputera.

Z oryginalnych akcesoriów pojawiły się jeszcze mysz oraz sprzętowy emulator ZX Spectrum 48, który umożliwia uruchomienie niemal 80% oprogramowania tego komputera na Enterprise.

Dostarczany razem z komputerem IS-BASIC jest dość rozbudowany, ale niezbyt szybki. W dużym stopniu zgodny z ANSI BASIC pozwala na programowanie strukturalne. Umożliwia definiowanie własnych procedur i funkcji z parametrami i zmiennymi lokalnymi. Posiada rozbudowane instrukcje do obsługi grafiki i dźwięku. Jego unikalną cechą jest możliwość uruchamiania kilku programów jednocześnie i wymiany danych pomiędzy nimi.

Poza BASIC dostępne były również inne języki programowania jak Forth, Lisp, czy Pascal.

Dostępny system IS-DOS, kompatybilny z CP/M, pozwalał na uruchamianie bogatej bazy istniejącego oprogramowania.

W roku 2014 pojawił się wielozadaniowy, okienkowy system SymbOS w wersji dla Enterprise (wcześniej znany dla Amstrada CPC i komputerów MSX).

DANE TECHNICZNE:

CPU:

ZILOG Z80A, 4 MHZ

PAMIĘĆ:

32 KB EXOS ROM, 16 KB BASIC CARTRIDGE
64 LUB 128 KB RAM, ROZSZERZALNA DO 4 MB

ROM:

32 KB ZAWIERAJĄCY SAM BASIC.

GRAFIKA:

16 TRYBÓW GRAFICZNYCH,
256 KOLORÓW
ROZDZIELCZOŚĆ GRAFIKI DO 672 X 256
(Z PRZEPLOTEM 672X512)
TRYB TEKSTOWY DO 84 X 28
(Z PRZEPLOTEM 84X56)
MOŻLIWOŚĆ SPRZĘTOWEGO NAKŁADANIA
NA OBRAZ ZEWNĘTRZNEGO SYGNAŁU
VIDEO W 16 KOLORACH
VIDEO CHIP "NICK"

DŹWIĘK:

8 OKTAW, 4 KANAŁY, STEREO
AUDIO CHIP "DAVE"

PRZYDATNE LINKI

FORUM: [HTTPS://ENTERPRISEFOREVER.COM/](https://enterpriseforever.com/)
WIKI: [HTTPS://WIKI.ENTERPRISEFOREVER.COM](https://wiki.enterpriseforever.com)

Enterprise to wygodny komputer dla wymagających użytkowników. Wciąż powstają dla niego nowe gry, programy użytkowe oraz nowe akcesoria. Dla tych którzy nigdy nie mieli styczności z Enterprise dostępne są emulatory na PC - EP128Emu i EP32.







Nothing compare to... ENTERPRISE



AMSTRAD C Schneider



C commodore 64



International Karate + Reloaded

System 3, 1988

http://ep128.hu/Ep_Games/Leiras/International_Karate_Plus_Reloaded.htm



CP/M na stacji do Atari

KAŻDY KOMPUTER JEST TAK DOBRY, JAK DOBRE JEST OPROGRAMOWANIE DLA NIEGO DOSTĘPNE. W PRZYPADKU 8-BITOWYCH ATARI UŻYTKOWNICY NA PRZEŁOMIE LAT 70. I 80. NIE MOGLI SPECJALNIE WYBIERAĆ WŚRÓD PROGRAMÓW UŻYTKOWYCH, KTÓRE WYRÓŻNIAŁY SIĘ NA CORAZ SZYBCIEJ ROZWIJAJĄCYM SIĘ RYNKU DOMOWYCH KOMPUTERÓW.

DANIEL KOZMIŃSKI

O wszem, dostępne na rynku Atari Word Processor (szybko zastąpiony AtariWriterem), VisiCalc, czy seria oprogramowania od Synapse Software (przede wszystkim SynCalc i SynFile+) mogły zaspokoić potrzeby mniej zaawansowanych użytkowników, ale posiadacze domowych komputerów Atari z zazdrością spoglądali na użytkowników „większych” maszyn, którzy dzięki temu, że ich komputery były zbudowane na podstawie mikroprocesorów Intel 8080 i Z80, mogli korzystać z bogatej biblioteki aplikacji przeznaczonej dla systemu CP/M.

Control Program/Monitor, później przemianowany na Control Program for Microcomputers, stworzony w 1974 r. przez Gary’ego Kildalla był, patrząc z dzisiejszego punktu widzenia, prostym systemem operacyjnym dla komputerów opartych na mikroprocesorze Intel 8080/85, a także Zilog Z80. Minimalne wymagania były niewielkie, wystarczył terminal używający zestawu znaków zgodnych z ASCII, 16 kilobajtów RAM i jedna stacja dysków, która potrafiła odczytać kod inicjalizujący (bo-



▲ OPROGRAMOWANIE DLA CP/M ORAZ DYSKIETKI SYSTEMOWE INDUS GT

otstrap) z pierwszego sektora dyskietki. System składał się z trzech zasadniczych elementów:

BIOS, czyli Basic Input/Output System,
BDOS – Basic Disk Operating System,
CCP – Console Command Processor.

Moduły te były ładowane do pamięci podczas inicjalizacji i pierwsze dwa z nich pozostawały w pamięci, a procesor linii poleceń mógł być nadpisany przez ładowaną przez niego aplikację i był przywracany, kiedy wywoływany program kończył swoje działanie.

Tak niewielkie wymagania, a także niska cena szybko po skutkowały stosunkowo dużym zainteresowaniem zarówno producentów sprzętu, jak i programistów, którzy przygotowali mnóstwo aplikacji, które szybko stały się standardem w danych dziedzinach. To właśnie na tej platformie powstał WordStar oraz – przez długi czas najlepszy – silnik bazodanowy dBase, a także zmora studentów informatyki Turbo Pascal. Wśród innych programów użytkowych można wymienić przodka Microsoft Excel, czyli Microsoft Multiplan, AutoCAD czy SuperCalc.

Tej klasy oprogramowania nie było dane zobaczyć na pozostałych popularnych platformach 8-bitowych, przez co wiodący producenci mikrokomputerów opartych na innych procesorach szybko rozpoczęli pracę nad sposobem udostępnienia bogatej biblioteki aplikacji dla ich maszyn.

I tak Commodore udostępniło moduł Commodore 64 CP/M cartridge zawierający procesor Z80. Dodatek ten niestety był wyjątkowo nieużywalny, z powodu niskiego – tylko 1 MHz – taktowania układu, a także z uwagi na 40-kolumnowy – jedyny dostępny – tryb wyświetlania obrazu. Nie jest możliwe uzyskanie, przynajmniej programowego, trybu 80-kolumn, przez co zdecydowana większość programów wygląda nieciekawie i korzystanie z nich jest wybitnie utrudnione. Nie ma również możliwości pracy na dwóch, przełączalnych częściach ekranu (jak to jest możliwe np. w przypadku ZX Spectrum). Na dodatek moduł działa poprawnie wyłącznie z pierwszymi modelami płyt głównych.

Popularność CP/M zauważył również Microsoft przygotowując kartę Z-80 SoftCart przeznaczoną dla popularnych ówczesnie na amerykańskim rynku komputerów Apple II. Warto przy okazji dodać, że to był pierwszy sprzęt w historii wyprodukowany przez obecnego giganta i przez kilka lat jego najbardziej dochodowy produkt. Karta ta była wyposażona w taktowany 2 MHz mikroprocesor Z80 oraz niezbędną logikę dopasowującą szynę procesora do złącza

Apple Bus. Wkrótce Microsoft zaoferował kolejną iterację swojego rozwiązania – tym razem z dodatkową pamięcią oraz 80-kolumnowym wyjściem obrazu – Premium Softcard IIe.

Atari już na początku lat 80-tych planowało udostępnienie użytkownikom sprzedawanych przez siebie komputerów możliwości korzystania z biblioteki oprogramowania dostępnego dla CP/M 2.2. Rozpoczęto wtedy, we współpracy z Add-on Computer Corp, projekt SweetPea, który miał się zmaterializować na półkach dystrybutorów na początku 1984r. w postaci zewnętrznego urządzenia podłączanego do magistrali SIO. Atari 1060 – ponieważ tak miała brzmieć handlowa nazwa – przypominało nieco stację dysków 1050, wyposażone miało być w 64 KB RAM oraz 2 KB pamięci wideo, wyjście na telewizor lub 80-kolumnowy monitor, oprogramowanie emulujące popularny terminal DEC VT52 oraz oczywiście Z80A taktowany zegarem 4 MHz.

Plany nieco się zmieniły, kiedy rozwinęto projekt modułu rozszerzeń do komputerów XL – Atari 1090. Wtedy to zdecydowano się na anulowanie projektu 1060, a doświadczenia zebrane podczas dotychczasowych prac wykorzystano przy nowym projekcie – Atari 1066, który miał być jedną z dostępnych dla tego modułu kart, oczywiście umożliwiającą uruchamianie systemu CP/M 2.2. Wkrótce jednak Atari zostało sprzedane Jackowi Tramielowi, którego jedną z pierwszych decyzji było anulowanie rozwoju komputerów oraz peryferiów dla linii 8-bitowej, przez co użytkownicy Atari musieli jeszcze chwilę poczekać na to, aby skorzystać z profesjonalnego oprogramowania – dała tę możliwość zewnętrzna firma Indus.

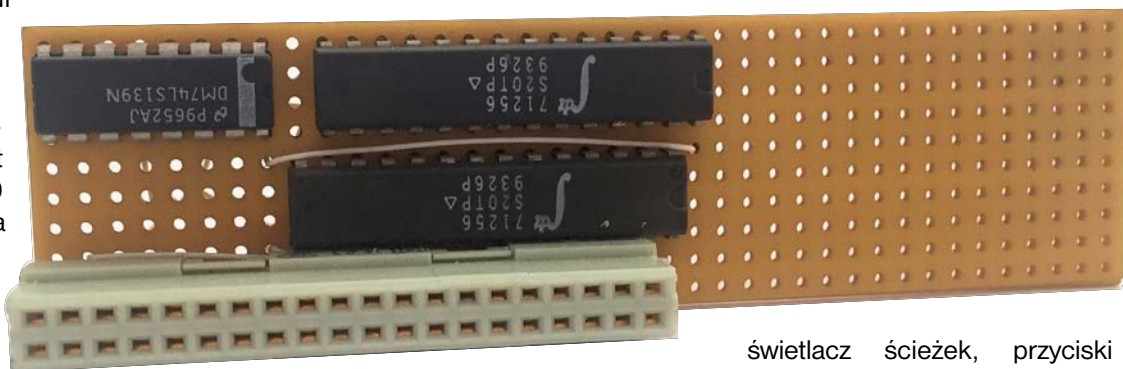
Indus Systems zaoferował użytkownikom Atari ówczesnie najbardziej zaawansowaną stację dysków – Indus GT. Wy-



▲ KOMUNIKAT POWITALNY INDUS CP/M



▲ TURBO PASCAL W PROGRAMOWYM TRYBIE 80. KOLUMN



światłacz ścieżek, przyciski służące do obsługi napędu, a także obsługa podwójnej gęstości były cechami, które stawiały ją na czele stawki urządzeń magazynujących dla Atari (dostępne były również wersje stacji dla C64 i Apple). GT zbudowany był na bazie mikroprocesora Z80, który był taktowany zegarem



◀ PROTOTYPOWY
SRAM CHARGER
ZAMONTOWANY
W LDW SUPER 2000

4 MHz, zawierał 4 KB ROM oraz 2 KB RAM. „Wygląda jak Ferrari, prowadzi się jak Rolls, a parkuje zgrabnie jak Beetle” – tak Indus Systems reklamował swój produkt w prasie komputerowej.

Czytelników nie interesują nawet tak wyszukane slogany, a faktyczne możliwości dostępne dla użytkowników – tu GT oferował (poza wcześniej wymienionymi) szybką transmisję 38,4 kbps, a nawet 68,2 kbps wraz z buforowaniem ścieżek po zainstalowaniu rozszerzenia pamięci o nazwie RAM Charger. Moduł ten oprócz oferowania szybkiego, wręcz błyskawicznego, ładowania danych dawał możliwość dostępu do ogromnej biblioteki oprogramowania dzięki obsłudze CP/M w wersji 2.2.

RAM Charger korzystał ze złącza rozszerzeń umieszczonego po prawej stronie płyty głównej urządzenia i zamieniał Atari XL/XE w inteligentny terminal komunikujący się z komputerem – którym stawał się Indus GT – poprzez złącze SIO. Komputer zapewniał wyjście obrazu na monitor oraz klawiaturę, a programy wykonywały się na stacji dysków.

Niestety Indus zakończyło działalność zanim prace nad modulem zostały zakończone. Future System – kolejne przedsiębiorstwo mające prawa do produktu sfinalizowało co prawda prace nad RAM Chargerem, firma nie była jednak specjalnie zainteresowana jego promocją zajmując się bardziej dochodowymi stacjami dysków do Atari 8-bit i ST.

Do Europy GT trafił dzięki Logical Design Works – importerowi komputerów Atari, a także producentowi akcesoriów i oprogramowania – pod nową nazwą LDW Super 2000. Produkt ten jest niemal dokładnym klonem Indus GT, a różnice dotyczą innego napędu – zastosowano mechanizm firmy Roc-tec, przeprojektowanej płyty głównej oraz układu zasilania. Oprogramowanie, które było oryginalnie przeznaczone do działania z Indus GT działa dokładnie tak samo z LDW Super 2000 – dotyczy to również RAM Chargera.

Moduł ten, dzięki zastosowaniu odwrotnej inżynierii, został odtworzony przez Polaków: Bartosza Trybusa we współpracy z Jerzym Sobolą i Januszem Dąbrowskim, a materiały pozwalające na zbudowanie własnego klona są dostępne

w Internecie, na stronie autora projektu. W miejsce pamięci dynamicznej zastosowano pamięć SRAM, co pozwoliło na uproszczenie produktu.

Korzystanie z CP/M jest możliwe dzięki terminalom pracującym w trybach 40- i 80-kolumnowych na standardowym Atari (tryb 80-kolumn jest realizowany programowo), a także dzięki nowemu terminalowi (TTerminal - Trub Terminal), który korzysta z rozszerzenia VBXE, zapewniając najlepszy komfort pracy.

Praca z CP/M 2.2 na 8-bitowym Atari niestety nie jest wyjątkowo luksusowa, że względu na powolną komunikację – i co za tym idzie niezbyt żwawe odświeżanie ekranu – komputera ze stacją dysków, niemniej użytkownikom otwierają się szeroko drzwi do biblioteki oprogramowania o jakości, której na Atari nigdy nie było.

Dzisiaj możemy się zastanawiać, czy Atari mogło być znacznie bardziej popularnym komputerem w latach 80-tych, gdyby ówczesnie istniała możliwość pracy z oprogramowaniem spełniającym rynkowe standardy. Implementacja CP/M zaproponowana przez Indus Systems była lepsza od tej, którą przygotowało Commodore dla C64 (możliwość pracy w 80-kolumnach, brak wymagań co do rewizji płyty głównej), a także oferowanej przez Amstrada dla ZX Spectrum +3 (tylko 64 kolumny, 80 kolumn dostępne poprzez przełączanie ekranów, droższe i mniej popularne dyskietki). ■

PRZYDATNE LINKI

INDUS GT (ATARIKI)

[HTTP://ATARIKI.KRAP.PL/INDEX.PHP/INDUS_GT](http://atariki.krap.pl/index.php/indus_gt)

LDW SUPER 2000 (ATARIKI)

[HTTP://ATARIKI.KRAP.PL/INDEX.PHP/LDW_SUPER_2000](http://atariki.krap.pl/index.php/ldw_super_2000)

RAM CHARGER (ATARIKI)

[HTTP://ATARIKI.KRAP.PL/INDEX.PHP/RAM_CHARGER](http://atariki.krap.pl/index.php/ram_charger)

INFORMACJE TECHNICZNE DOT. SRAM CHARGER (TRUB)

[HTTP://TRUB.ATARI8.INFO/INDEX.PHP?REF=CPM](http://trub.atari8.info/index.php?ref=cpm)

INSTRUKCJA KORZYSTANIA CP/M 2.2 Z ATARI XL/XE (ATARI.AREA)

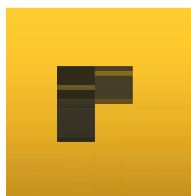
[HTTP://WWW.ATARI.ORG.PL/ARTYKUL/CP-M-2.2-DLA-ATARI-XL-XE/37](http://www.atari.org.pl/artykul/cp-m-2.2-dla-atari-xl-xe/37)



RETROSPECS

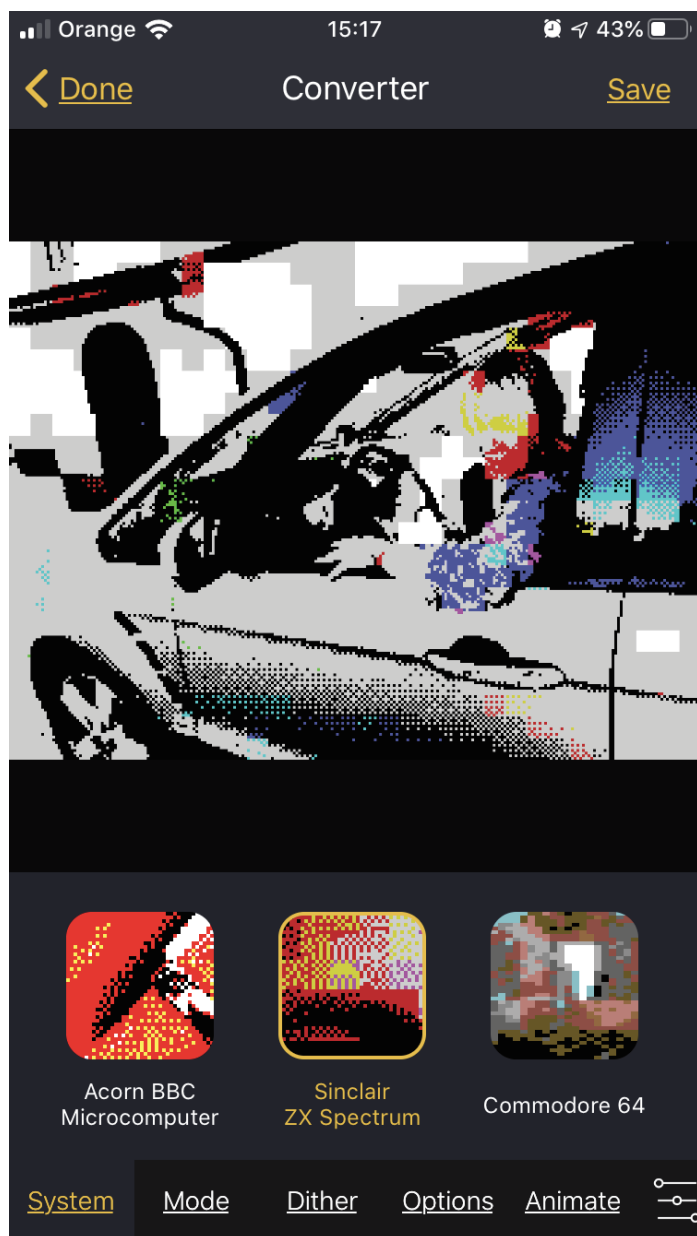
TO PIERWSZY KROK W ŁAŃCUCHU PRZEKSZTAŁCANIA ZDIGITALIZOWANYCH OBRAZÓW ORAZ W PEWNYM ZAKRESIE FILMÓW DO SPECYFIKI KONKRETNÝCH RETROSYSTEMÓW. PRZEJRZAŁEM MULTUM KONWERTERÓW CZY NARZĘDZI KONWERTUJĄCYCH WPISANYCH W PROGRAMY GRAFICZNE NA PC. NIE OWIJAJĄC W BAWĘŁNĘ POWIEM, ŻE POD WZGLĘDEM MOŻLIWOŚCI ORAZ INTUICYJNOŚCI DLA FOTOGRAFA CZY TEŻ FOTOGRAFIKA (TO WAŻNE) CHYBA ŻADEN Z TYCH, KTÓRE SPOTKAŁEM NIE DORASTA RETROSPECS DO PIĘT.

CHICADII

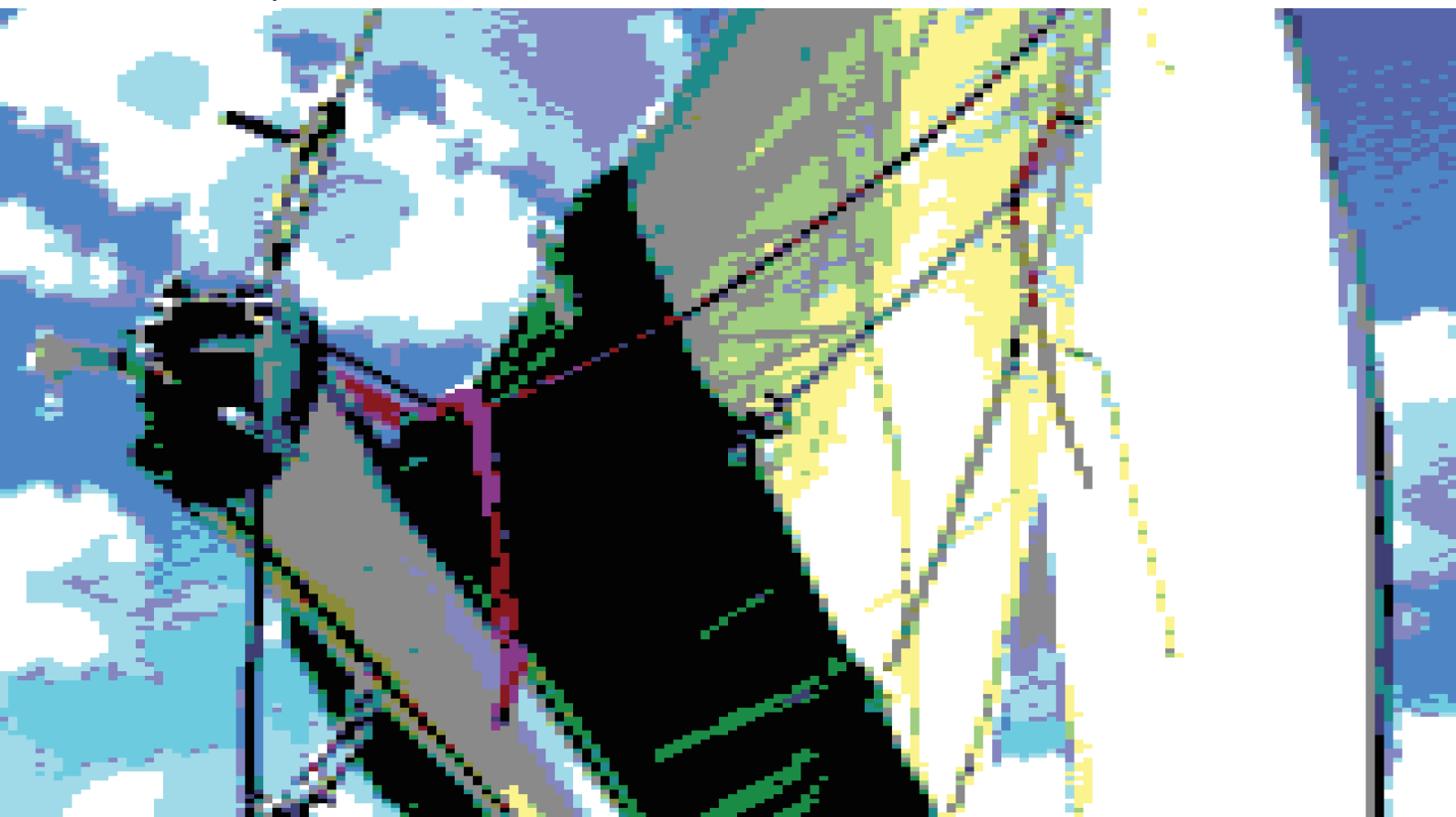


To jest artykuł o narzędziu, które ma za zadanie pozwolić porozumieć się istniejącym już w formie cyfrowej produkcjom świata zewnętrznego ze światem wnętrza starych systemów komputerowych. To narzędzie, które pozwala zobaczyć, jak wizualia naszego świata interpretują maszyny.

Pozwala również wykorzystać te produkcje do tworzenia zupełnie nowych rzeczy za pomocą technik późniejszego łączenia, przekształcania czy miksu. Celowo piszę tu o technikach, ponieważ - jak w każdej dziedzinie - istnieje wiele metod dotarcia do końcowego dzieła. W różny sposób także należy oceniać kunszt twórcy, ponieważ każda technika wymaga odmiennych umiejętności. Z tego punktu widzenia, dobrze wykorzystane narzędzia do ręcznego rysowania, jak w przypadku ZX Spectrum np. *Multipaint*, znajdują szczególne zastosowanie w rękach artystów czy osób wprawionych w rysunku



▼ KONWERSJA ZDJĘCIA DO FORMATU AMSTRAD CPC



ręcznym oraz w naszym przypadku – i tu dodatkowo oceniamy kunszt – oczywiście dodatkowo posiadających wiedzę na temat ograniczeń systemu, sprytnego stosowania atrybutów, kolorystyki itd. Tego typu osobom narzędzia do konwersji w zasadzie niemal nie są do niczego potrzebne. Nie należy jednak zapominać, że tak jak dziedziną tworzenia jest rysunek jest nią również fotografia. A ta z kolei – poza purystycznymi frakcjami typu „tylko analog” czy „zero ingerencji” – posiada całą paletę technik przetwarzania, przeinaczania, łączenia czy wyolbrzymiania – technik, których najsłynniejszym dzieckiem jest narzędzie pod nazwą *Photoshop*.

8-bitowce nie doczekały się dotychczas niestety – nad czym ubolewam – narzędzia, które w swych podwalinach miałyby filozofię obróbki i łączenia gotowych fragmentów fotografii czy innych elementów graficznych – z prostą obsługą i panelem warstw, kilkoma metodami ich nakładania, obrotu czy przekształcania z uwzględnieniem ograniczeń systemowych. Może kiedyś się doczekam. Istnieją pewne niezależne programy, które w ograniczony sposób to potrafią, ale to chyba w ogóle temat na inny artykuł.

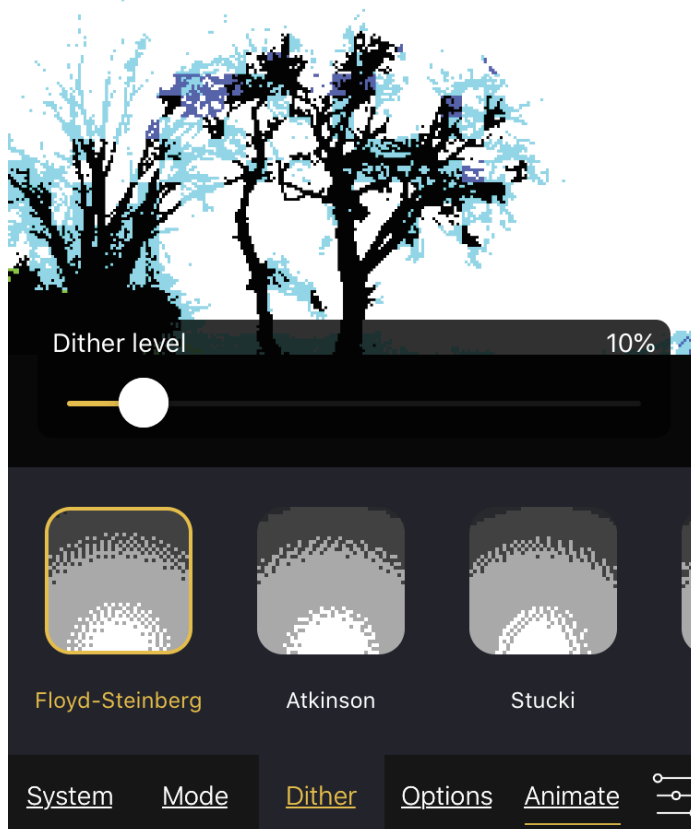
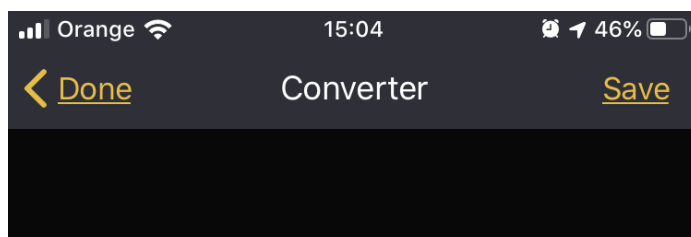
Narzędzie będące przedmiotem niniejszego artykułu – konwerter *Retrospecs* – dostępny póki co wyłącznie na iOS (w niezależnych wariantach na iPhone'a i iPad'a) – to pierwszy krok w łańcuchu przekształcania zdigitalizowanych obrazów oraz w pewnym zakresie filmów do specyfiki konkretnych retnosystemów. Jako człowiek dość mocno związany z fotografią, który posiada napis ZX wryty na lewej komorze serca – przegrzebałem i przejrzałem multum konwerterów czy narzędzi konwertujących wpisanych w programy graficzne na PC. Nie owijając w bawełnę powiem, że pod względem możliwości oraz intuicyjności dla fotografa czy też fotografika (to ważne) chyba żaden z tych, które spotkałem nie dorasta *Retrospecs* do pięt. Ale, ale – zaczniemy od początku.

Autorem aplikacji jest John Parker, jej pierwsza wersja powstała i została opublikowana w 2014 roku, od tego czasu jest systematycznie rozwijana i wzbogacana o kolejne funkcje. Program istnieje w dwóch wersjach – darmowej podstawowej (wyposażonej w najważniejsze funkcje i udostępniającej kilka

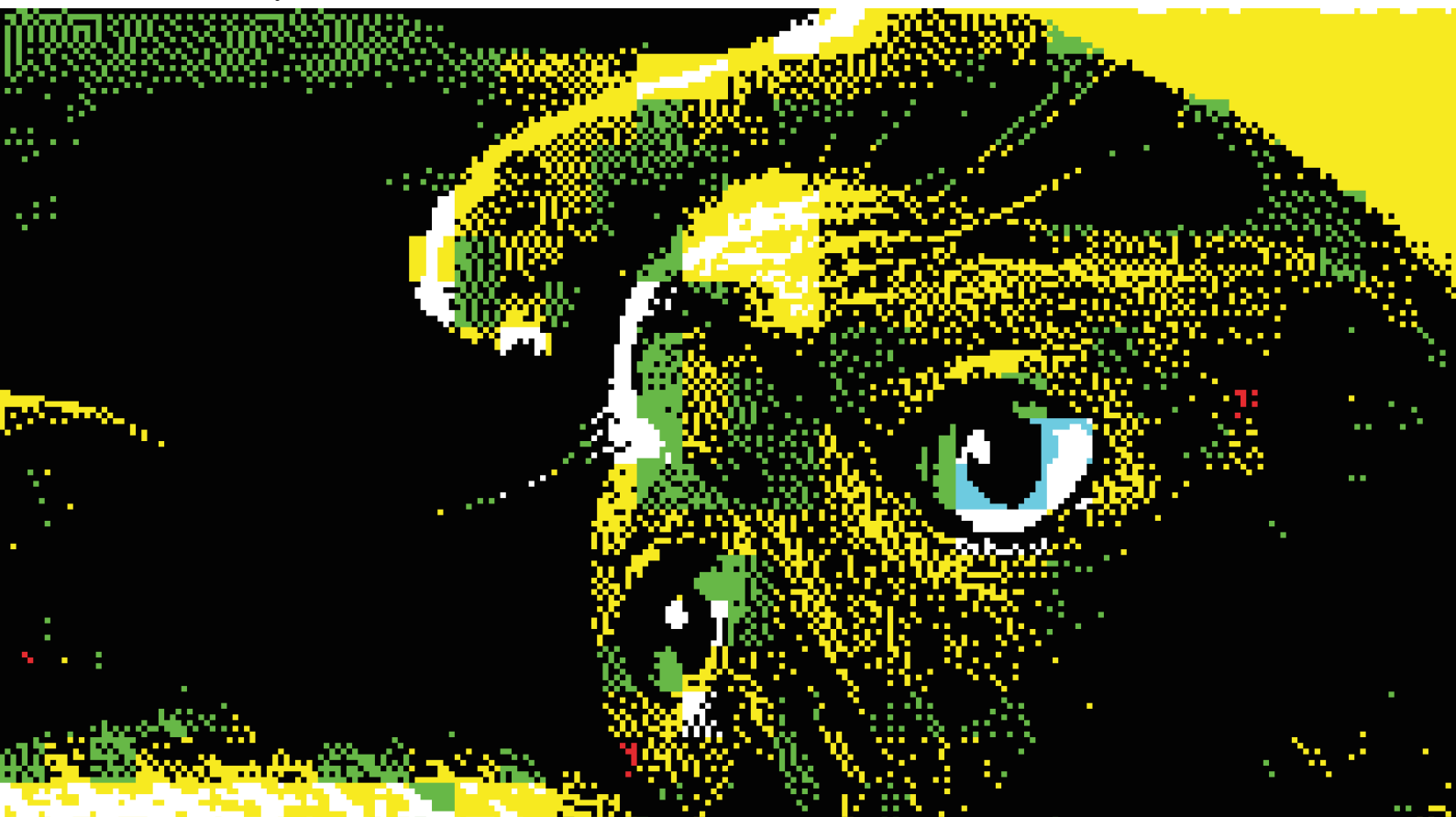
systemów, o czym poniżej) oraz płatnej (cena 18,99 zł), która w znaczny sposób poszerza paletę możliwości.

Z wersji darmowej w przypadku czytników naszego Zin80 (czyli sprzętu opartego na procesorze Z80) najbardziej zadowoleni będą użytkownicy Amstrada CPC, bowiem jest on jednym z 8 systemów udostępnionych w tej wersji. Nadmienić należy, że między innymi udostępnia ona także oparty o Z80 system Mattel Aquarius, ciekaw natomiast jestem ilu z czytników takowy system posiada. Prócz wspomnianych dwóch systemów, pozostałymi emulacjami konwersji w podstawowej wersji są: Atari 2600, Commodore PET, IBM CGA, Fujitsu FM-7, Acorn Archimedes oraz Nintendo Gameboy. Tu chciałbym zauważyć, że jestem prawie pewien, że we wcześniejszych wersjach aplikacji w podstawowej wersji udostępniony był również ZX Spectrum, natomiast chyba decyzją autora, niedawno został on przeniesiony do wersji rozszerzonej. Troszkę szkoda, ale cóż, jak mawiał James Bond: „Po co się babrać w przeszłości?”.

Zasadniczą ideą programu jest oczywiście przekształcenie zdigitalizowanego obrazu lub zdjęcia do formatu odpowiedniego dla wybranego z wymienionych wyżej systemów. Po zrobieniu zdjęcia lub wybraniu gotowego z biblioteki, dokonujemy wyboru systemu, a następnie jednego z trybów (o ile system posiada ich większą liczbę - np. dla Amstrada CPC udostępniono 6 trybów, różne wariacje Mode 0, 1 i 2). Kolejnym krokiem jest przystąpienie do obróbki tak zdefiniowanego obrazu docelowego. Na cele manipulacji obrazem w wersji bazowej do dyspozycji mamy: 5 algorytmów ditheringu (Floyd-Steinberg, Atkinson, Checkerboard, Bayer 8x8, Horizontal) oraz takie opcje edycji dla pliku wynikowego jak: kompensacja słabego światła, podbicie nasycenia, kontrastu, dodanie szumu, zepsucie atrybutów, itd. Dodatkowo program udostępnia możliwość globalnej korekty pliku źródłowego przed konwersją, w postaci możliwości regulacji nasycenia, jasności, kontrastu, ostrości oraz przesunięcia barwowego. Wszystko z natychmiastowym podglądem symulowanego efektu końcowego. Obrazek możemy wykadrować - w połączeniu z opcją powiększania do określonego fragmentu zdjęcia. Do decyzji pozostaje również, czy powinien on posiadać oryginalne proporcje, proporcje niestan-



▼ KONWERSJA ZDJĘCIA DO FORMATU ZX SPECTRUM



dardowe, czy proporcje, jakie posiada docelowy system. Na koniec pozostaje eksport do pliku png. Wersja podstawowa obsługuje również filmy i animacje – w tym przypadku chodzi o konwersję stylizacyjną, opartą o specyfikację danego systemu. Na animację można nałożyć dodatkowo parametry zmieniającego się szumu czy zniekształcenia atrybutów. Obsługę całości dopełnia prosta możliwość kadrowania – tak jak w przypadku zdjęcia. Całokształt można wyeksportować do formatu animowanego pliku gif. Efekty są świetne. Już podstawowa wersja zachęca do zabawy, w efekcie zakupu wersji rozszerzonej natomiast aplikacja zamienia się w prawdziwy kombajn. Ilość dostępnych systemów wzrasta do 51 (a znaczna część z nich posiada więcej niż jeden tryb) – łącznie z takimi egzotycznymi konstrukcjami jak np. Tatung Einstein czy Colecovision, wśród nich są również takie systemy jak Timex, ZX81, Sam Coupé czy Spectrum Next. Ilość algorytmów ditheringu – rośnie do 20.

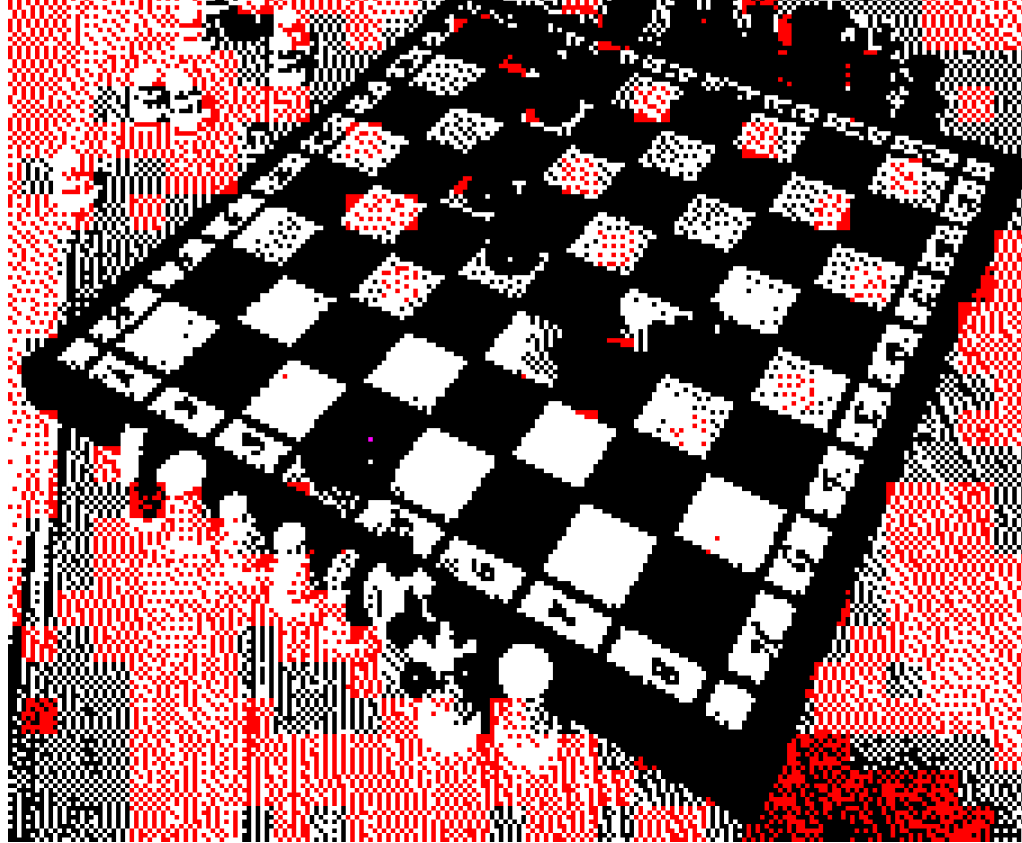
Pojawia się nowy moduł animacji – z opcją nakładania 13 predefiniowanych efektów animacyjnych na nieruchome zdjęcie. Opcje animacji udostępniają możliwość upscalingu wyniku, ustalenia jakości pliku wynikowego oraz możliwość stworzenia animacji typu ping-pong. Rozszerzają się również możliwości wyboru formatu eksportowanego pliku – teraz, oprócz animowanego gifa pojawia się również format mp4.

Moc pokazuje dział dotyczący tworzenia własnych systemów emulacyjnych, w którym otrzymujemy kopalnię możliwości. I tak, tworząc własny system decydujemy, czy ma być on oparty na atrybutach (ekran dzielony jest na komórki, które zawierają określoną liczbę kolorów), pixelach (gdzie każdy piksel może przyjmować dowolny kolor z palety), palecie (gdzie każdy piksel na ekranie może przybierać jeden z określonej liczby kolorów) czy znakach (gdzie ekran jest podzielony na komórki znakowe z kolorem tła oraz kolorem znaku).

Kolejnym krokiem jest konfiguracja systemu – dla systemu opartego na atrybutach opcjami są tu: określenie szerokości i wysokości atrybutów, liczbę kolorów atrybutów oraz liczba współdzielonych kolorów w obrębie atrybutu. Dla systemu opartego na palecie określamy ilość kolorów (maksymalnie 4), dla systemów opartych na znakach do wyboru mamy z kolei 32 rzeczywiste fonty takich systemów jak m.in. ZX80, Atari czy Amiga. Następny etap to dobór palety, gdzie określamy głębię koloru – tu do wyboru mamy dziesiątki rzeczywistych systemów (policzyłem! – 81) i ponad setkę pozasystemowych presetów. Można oczywiście również tworzyć własne zestawy kolorystyczne. Stworzoną paletę można zapisać oraz wyeksportować do pliku png. Z takiego też formatu do aplikacji można paletę zaimportować. Ostatnią z opcji tworzenia własnego systemu jest określenie szerokości i wysokości całego ekranu oraz wielkości pikseli.

W pełnej wersji pliki wynikowe mogą być wyeksportowane wg różnych wielkości do wyboru – od natywnej skokowo aż do powiększenia w okolicach 8000 pixeli na dłuższy bok.

W specjalny sposób potraktowany został ZX Spectrum – z okazji 35-lecia systemu autor postanowił dodać miłą funkcję – obrazki stworzone w programie można bezpośrednio ekspor-



▲ KONWERSJA ZDJĘCIA DO FORMATU ZX SPECTRUM

tować do natywnych plików *.tap oraz *.scr. Przetestowałem w realu, działa!

Praca i zabawa z aplikacją daje dużo przyjemności, choć warto skoncentrować się na kilku systemach, bo liczba możliwości i opcji w wersji rozszerzonej kusi, a eksperymentowanie może odciągać od pierwotnie przyjętego planu. Chyba najistotniejszą funkcją, jaka przychodzi mi do głowy, o którą w przyszłości warto by program rozszerzyć, to lokalna edycja pliku źródłowego. Chodzi mi o możliwość zmiany ekspozycji, nasycenia czy kontrastu wybranego fragmentu zdjęcia. Na ten moment – o ile jest to konieczne – trzeba wykonać to w zewnętrznym programie. No, ale program się rozwija – miejmy nadzieję, że autor zdecyduje się na wprowadzenie takiej funkcji. Dobrze byłoby również, gdyby aplikacja była dostępna na Androida.

Dodajmy opisane opcje i możliwości do najważniejszej cechy aplikacji, czyli tego, że jej środowiskiem jest wyposażony w aparat fotograficzny, podręczny smartfon. Dla fotografa czy fotografika już ta jedna rzecz zamiata całą długą sekwencję tworzenia zdjęcia w aparacie, kartowania, kabelkowania, kopiowania do folderów, wgrywania do przeciętnych programów edycyjnych itd. Pstryk, zdjęcie z poziomu aplikacji i od razu przystępujemy do konwersji. Z dbałością o szczegóły i szacunkiem do starych systemów. Ja używam i nie zamierzam przestać. ■

TYTUŁ APLIKACJI: RETROSPECS

SYSTEM: IOS, WERSJA IPHONE + OSOBNĄ WERSJĄ IPAD

ROK WYDANIA: 2014, PIERWSZA WERSJA, AKTUALNIE 2.12.2

DEVELOPER: JOHN PARKER

[HTTPS://8BITARTWORK.CO.UK](https://8bitartwork.co.uk)





MISTER

KUNG-FU

KTÓŻ NIE PAMIĘTA KLIMATU ARCADOWYCH CIEMNYCH RAJÓW LAT OSIEMDZIESIĄTYCH, EMOCJI JAKIE UDZIELAŁY SIĘ WE WNĘTRZACH ZATĘCHŁYCH DRZYMAŁOBUSÓW CZY OBSKURNYCH NOR Z ZAKRATOWANYMI OKNAMI. NIEKWESTIONOWANYM KRÓLEM TYCH MIEJSC BYŁ *KUNG FU MASTER*, PERŁKA JAPOŃSKIEGO MISTRZA COIN-UPÓW TAKASHI'EGO NISHIYAMY.

CHICADII

O powieść o poruszającej młodociane serca fabule, która opowiada o porwaniu pięknej i dobrej dziewczyny o imieniu Sylwia. Skutkuje to bolesną rozłąką z kochającym oraz nadszpiewanie i niebiblijnie wiernym Thomasem. Pięciu niegodziwców już wkrótce miało się przekonać, że zadzierać z nim nie warto. Upór, zręczność i inne nieziemskie umiejętności, sprawiły, że Thomas bez problemu pokonywał napotkanych po drodze wrogów, a wszystko po to, by odbić tą, którą kochał i by być z nią na zawsze. Opłaciło się, bo dzięki temu wrył się w pamięć wielu adeptów komputerowych sztuk walki jako **King Thomas - Kung Fu Mistrz**.

ŁARA!!!!!!

Doskonale pamiętam entuzjazm jaki towarzyszył mi podczas powrotów do domu, w którym do dyspozycji miałem własną maszynkę do gier (ZX Spectrum), która oferowała rozrywkę video na nieco innym poziomie niż ta znana z salonów gier. Mimo to i tak rozpaliała głowy jej posiadaczy i ich kolegów. Trochę trwało zanim dorwałem oficjalny port mojej ulubionej salonowej gry. Nie pamiętam już w jaki sposób stałem się



WZÓR WSZYSTKICH KLONÓW - ORYGINALNA WERSJA KUNG-FU MASTER ARCADE FIRMY IREM, KTÓRA KRÓLOWAŁA W SALONACH ROZRYWKI I W WOZACH DRZYMAŁY.

jej posiadaczem, ale koniec końców stary Kasprzak zaśpiewał charakterystyczną skrzeczącą melodię dekady przełomu roku 86, a ja z biciem serca oczekiwałem na wgranie się



mojego hitu. Za którymś tam razem w końcu nie wyskoczył: „R TAPE LOADING ERROR“ i.....

BŁŁEEEEEE! CO TO JEST?

Zamiast porywającej przygody jakiś fioletowo-zielony anemiczny koszmarek z pokrącaną chyba trzyklatkową animacją ciosu i paralitycznymi kukłami, pojawiającymi się to z lewa, to z prawa, w które nijak nie dawało się trafić. Muzyka podczas gry niespecjalnie różniła się od tej podczas jej wgrywania. Rozczarowanie było ogromne, nadzieje zaprzepaszczone, wydawca portu z 1986r. - U.S. Gold odważyło wtedy niezłego crapa - na pękniętą membranę, toż nawet Atari 2600 dostało lepszą konwersję... jak to możliwe? Tytuł szybko poszedł w odstawkę, a gołowąsowi piszacemu te słowa, uczącemu się sztuk walki pod blokiem z czerwonej książki o zapomnianym tytule, pozostały już tylko ryzykowne eskapady do salonu i zagrożenie prawym prostym centralnie w ryło od lokalnego niebieskiego ptactwa bez hodowcy.

I wydawałoby się, że to już koniec tej jakże charakterystycznej dla lat 80. osiedlowej historii bez happy endu. Było, minęło, po co się grzebać w przeszłości? Jak pokazują jednak ostatnie lata, tych co się grzebią jest jeszcze pokaźna ilość, a że czynią to w sposób kreatywny - zawsze jest szansa na coś nowego.

W zeszłym roku na portalu Proboard zainicjowana została akcja ZX-Dev M.I.A.-Remakes, w ramach której ochotnicy dokonują remake'ów, ulepszeń bądź portów tytułów na naszą trzymającą się coraz lepiej platformę, czyli ZX Spectrum. Jedną z inicjatyw było „nowe podejście do portu *Kung Fu Mastera*“ zainicjowane przez Eltona Bird'a - występującego m.in. dla tego tematu jako założyciel Uprising Games. W połowie



ZAMIAST POCHŁANIAJĄCEJ PRZYGODY JAKIŚ FIOLETOWO-ZIELONY ANEMICZNY KOSZMAREK Z POKRACZNĄ CHYBA TRZYKATKOWĄ ANIMACJĄ CIOSU (U.S. GOLD, 1984 - ORYGINALNY PORT GRY)

grudnia 2018 ukazał się nowy produkt - o tytule obarczonym wadą genetyczną z wiadomych względów (baaaad corpo!). *Mister Kung Fu*. „Ciekawe“, pomyślałam, raz - dwa ładując program do maszyny i...

ŁOOOŁ!

To jest to, na co czekali wszyscy w 1986 roku! Port, który przede wszystkim ma szacunek do oryginału i stara się go w jak najwierniejszy sposób odtworzyć i czyni to w świetnym stylu. Od pierwszego screenu z listem wprowadzającym oraz ekranem rozpoczynającym akcję gry widać, w jaki tytuł gramy. *Mister Kung Fu* jest niesamowicie szybki i płynny - Uprising Games deklaruje prędkość 50 klatek na sekundę i naprawdę - czuć to w każdym momencie. Tempo rozgrywki zbliżone jest do oryginału - fale wrogów atakują to z lewej, to



z prawej strony i rzeczywiście w chwili zagapienia, momentalnie zostajemy pochwyceni w żelazne kleszcze łap, odbierających energię, a wtedy potrzeba dużej zręczności, aby się uwolnić.

Dla nieznających oryginału (jest ktoś taki?) - misją gry jest uwolnienie z rąk Pięciu Synów Diabła ukochanej głównego bohatera o imieniu Sylwia. W tym celu poruszamy się postacią karateki, idąc korytarzem w jednym kierunku, pokonując równocześnie hordy wrogów, atakujących go to z przodu, to z tyłu. Do dyspozycji mamy pięć wiernie odtworzonych pięter, każde z bossem na końcu, po pokonaniu którego przechodzi się na kolejne piętro, aż do samego końca rozgrywki. A jest ona urozmaicona, w ramach postępów pojawiają się nowi wrogowie - z obrębu gatunku homo sapiens, jak również przedstawiciele fauny niższej, w tym magicznej.

Graficznie gra wiernie nawiązuje do oryginału - niemal wszystkie postaci wykonane są w sposób do niego zbliżony, stanowią reprezentację tego co widzieliśmy w salonie gier. Mało tego - swoim zachowaniem NPC-e również reagują bardzo podobnie - nożownicy czasami wycofują się, gdy chcemy do nich podejść, karty wyskakują w górę robiąc salto, da się je również przeskoczyć, smoki zioną, nietoperze fruwać, kule wybuchają itd. Każdy kto uczył się ruchów na pierwotnym, poczuje się w trakcie rozgrywki jak ryba w wodzie. Cały repertuar ciosów został zachowany i odtworzony, mamy do dyspozycji kopnięcia i uderzenia ciosem, w przysiadzie, stojąc i w wyskoku. Poprzez sprytnie narysowane tła autorzy niemal zminimalizowali występowanie colour clasha - jest on praktycznie niezauważalny.

Udzwiękowanie stoi na wysokim poziomie, szybki i dynamiczny soundtrack (dostępny w wersji 128K) - będący oczywiście bardzo dobrym coverem oryginału - podkreśla atmosferę i sprawia, że chce się grać i pracować, pracować i grać. Dźwięki ciosów zostały przyzwoicie odtworzone i w zasadzie jedyne czego mi tak naprawdę brakuje, to odpowiednika charakterystycznego dla tej gry odgłosu „Łar!“, rozlegającego się w oryginale przy wyskoku z kopnięciem.

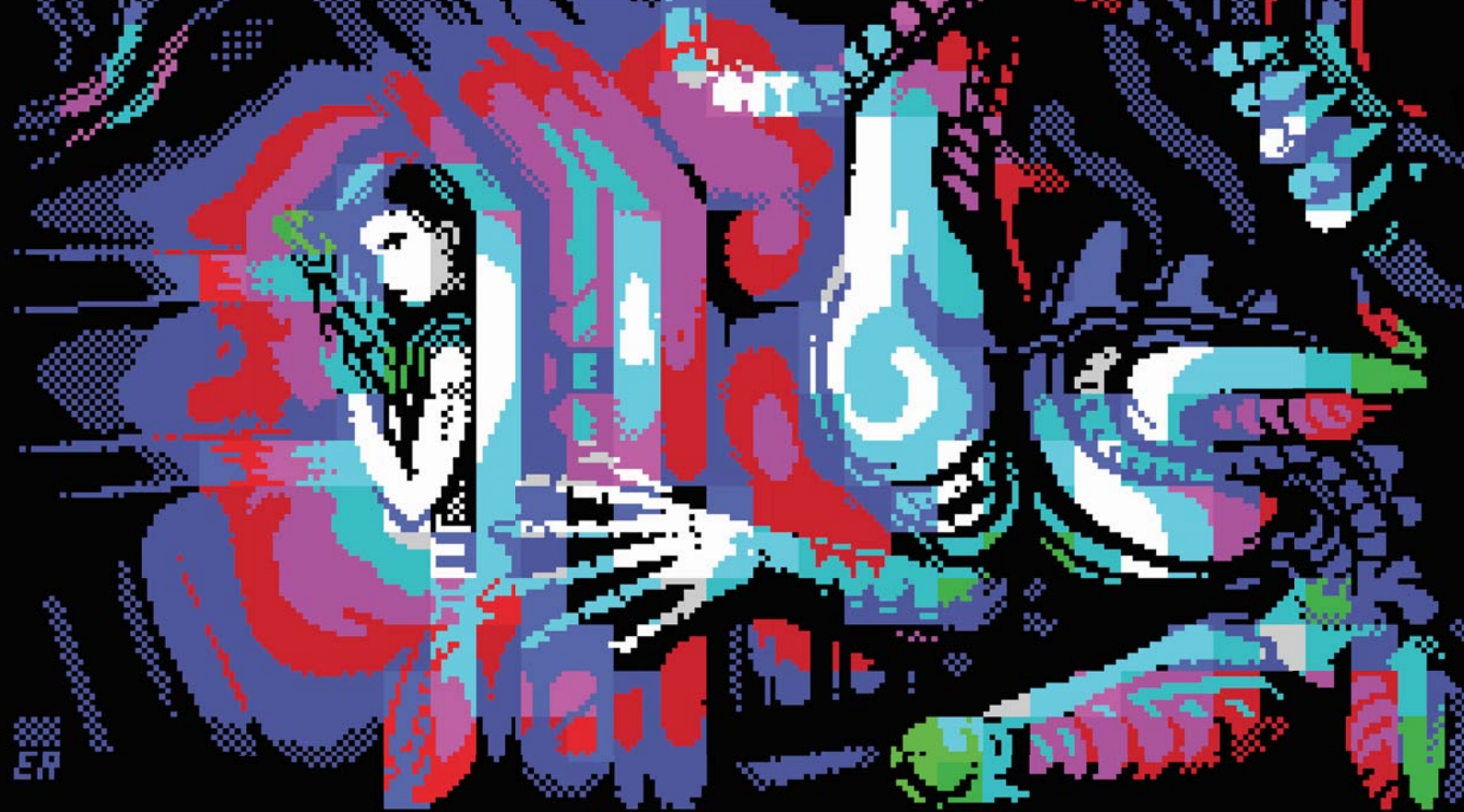
A skoro już mowa o tym, co mogłoby być lepsze - walki z bossami są NIECO prostsze niż w oryginale, bardziej skomplikowane w zasadzie jest dotarcie do najgorszych złoczyńców niż ich pokonanie - ten element chyba mógłby być w jakiś sposób podciągnięty. Piszę „chyba“ i „w jakiś sposób“, bo mimo to jednak balans gry został dobrze dobrany (pod warunkiem oczywiście, że gra się w duchu czasów oryginalnych - bez korzystania z save'a w emulatorze lub na DIVMMC). Nie jest prosto ukończyć tę grę - dalsze poziomy, do których dotarcie jest dość czasochłonne, wymagają zręczności i nauki, więc graczy mających zamiar przejść grę w starym stylu, czeka całkiem niezłe, wielogodzinne, a może i dłuższe wyzwanie.

Z małych bugów - głowa Thomasa wchodzi na panel z punktami podczas wchodzenia po schodach. Z małych marzeń - można by się może pokusić o dodanie oryginalnej animacji początkowej - opowiadającej o porwaniu Sylwii, a także mikroanimacji pomiędzy poziomami - może w Spectrumie zabrakło autorom już na to miejsca w pamięci, ale ponieważ plik gry zajmuje 48KB może udałoby się to zrobić w jakiejś skorygowanej, rozszerzonej wersji? Dużo tych „może“. Tak czy inaczej to naprawdę są drobiazgi.

Easy to start, hard to master jak mawia przysłowie. A raczej mister, Mister Kung Fu. Uprising Games - czapki z głów, dziękuję za mistrzowską konwersję. Jestem przekonany, że gdyby taki tytuł pojawił się w latach osiemdziesiątych byłby już wtedy jedną z najlepszych gier na Spectrums, zajmując dumnie miejsce w gatunku obok *Renegade*. Tak to miało wyglądać od początku, lecz cóż - takim to tytułem staje się z końcem 2018. Ratuj Sylwię kto może!



Tytuł: **Mister Kung Fu**
Rok wydania: **2018**
Developer: **Uprising Games**
Kod: **Elton Bird**
Grafika: **Andrew McDonnell, Elton Bird**
Audio: **Elton Bird**
Dodatkowa grafika: **Tony Pastor**
<http://uprising-games.com/>



ALIENS

NEOPLASMA

POMIMO WYSTĘPOWANIA WIELU GIER ZE SŁOWEM ALIENS W TYTULE NA ZX SPECTRUM, W ZASADZIE NIE ODNOSZĄ SIĘ ONE DO SCOTTOWSKIEGO UNIWERSUM. DOPIERO 2019 ROK PRZYNIOŚŁ NOWĄ PRODUKCJĘ, KTÓRA EWIDENTNIE WIĄŻE SIĘ ZE ŚWIATEM GIGEROWSKICH KSEOMORFÓW.



CHICADII

Przelom lat 2018 i 2019 rozpoczął się dla spectrumowej braci z przytupem, a to wszystko dzięki inicjatywie ZX DEV M.I.A. Remakes, która za wyzwanie postawiła sobie stworzenie nowych tytułów, jakie albo nigdy nie powstały na ZX Spectrum (stąd skrót od *Missing In Action*), albo byłyby ulepszonymi remake'ami wcześniejszych produkcji. ZX DEV spotkało się z dużym odzewem

twórców, co zaowocowało łącznie około 20 produkcjami, przy czym kilka z nich to absolutne perełki. Nie inaczej jest z omawianym tutaj tytułem **Aliens: Neoplasma**. To nowe podejście rosyjskiej grupy Sanchez (znanej m.in. z wybitnej **Castlevanii: Spectral Interlude** czy świetnej konwersji **Mighty Final Fight**) do szanowanego i kultowego uniwersum **Obcego** Ridleya Scotta.

Ksenomorf gościł już wcześniej na platformie ZX Spectrum – jakżeby nie, podobnie zresztą jak większość block-

busterów filmowych lat 80. Licencjonowany **Aliens** (Mind Games) z 1984 roku, to taktyczna gra z dość ubogą grafiką, ale docenianą przez niektórych za klaustrofobiczną rozgrywkę. Druga część – **Aliens** (Electric Dreams) – również na licencji – wyszła najpierw w wersji angielskiej w 1986r. – jako ciekawe podejście do FPP, oraz następnie w wersji amerykańskiej w 1987r. – proponującej kilka typów rozgrywki, przez co była w zasadzie kompletnie inną grą (niestety nieudaną).

Pomimo występowania wielu innych gier ze słowem Aliens w tytule, w zasadzie nie odnoszą się do Scottowskiego uniwersum. Dopiero 2019 rok przyniósł nową produkcję, która ewidentnie wiąże się ze światem Gigerowskich ksenomorfów.

Akcja **Aliens: Neoplasma** rzuca nas na pokład potężnego statku transportowego – Achillesa – mającego za zadanie przetransportowanie próbek oraz pasażerów stanowiących wcześniej mieszkańców planetoidy LV-426 – tak, tej samej, na której Ripley rozpoczęła swój koszmar z obcymi kilkadziesiąt lat wcześniej – na Ziemię. Fabuła toczy się pomiędzy wydarzeniami z filmu **Obcy**, umiejscowionych w 2122 roku a tymi, które to z kolei znamy z drugiej części filmu (Dla porządku na 10 lat przed ponownym wybudzeniem Ripley, ale 57 lat po wydarzeniach z części pierwszej). Jak więc widać, w międzyczasie sporo się działo...

Główna bohaterka gry – Ashley Smith – budzi się więc jako jedyna ze stanu hibernacji w 2169 roku – po 2 latach i 4 miesiącach lotu – i powoli odkrywa, że coś nie poszło zgodnie z planem... Brzmi znajomo? Oczywiście. I tak będzie dalej...

Gra jest zręcznościówką labiryntowo – platformową dziejącą się na pokładzie Achillesa. Rozgrywkę toczymy patrząc na całość z boku. Jak to w tego typu produkcjach bywa, mamy za zadanie rozeznać się, co się dookoła dzieje, odnaleźć przejścia w labiryntach oraz dojść tam gdzie fabuła poprowadzi uważając na czyhające niebezpieczeństwa.

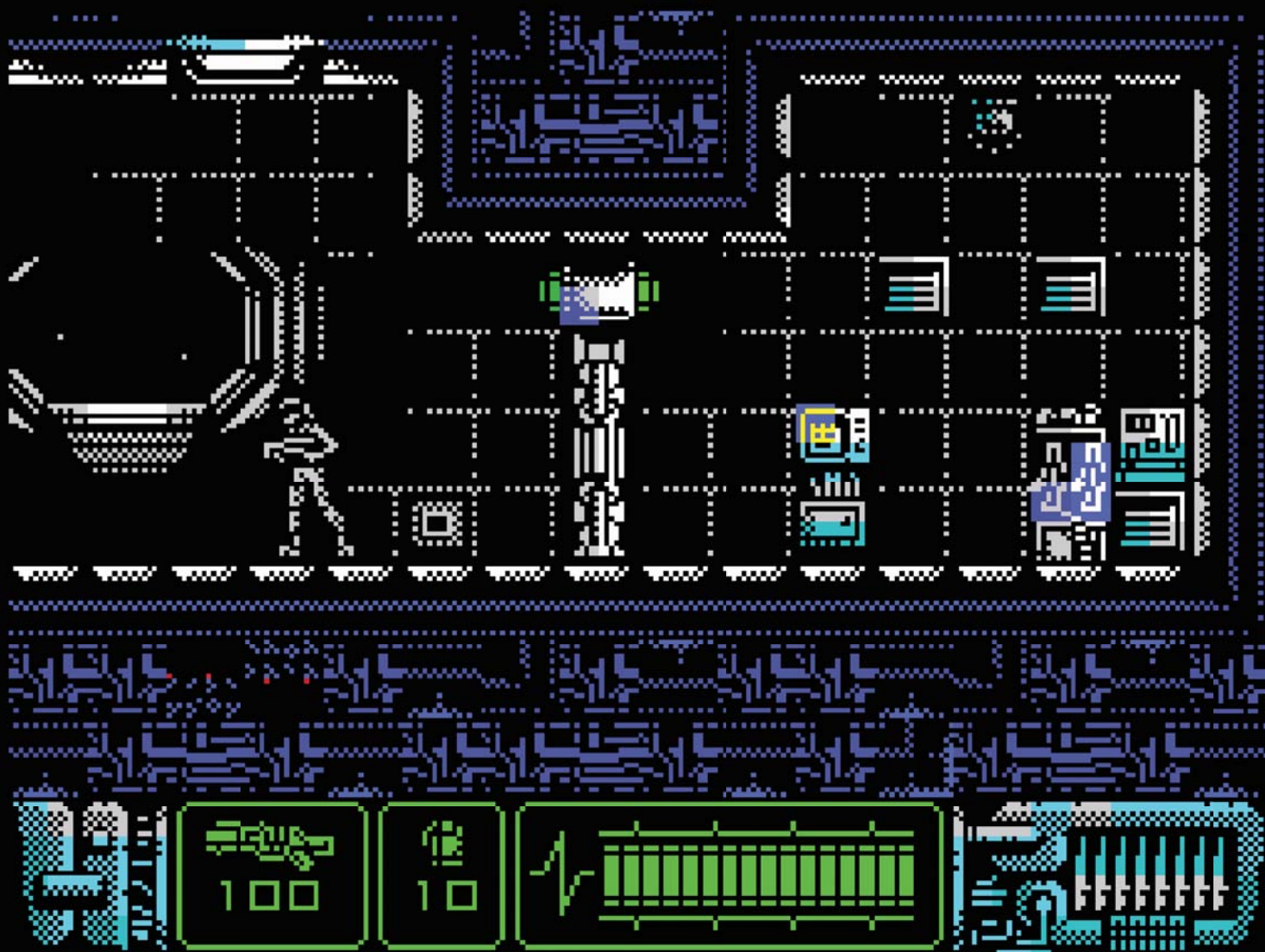
Postać Ashley wyposażona została we wszystkie niezbędne umiejętności – pokład przemierzamy biegając, wspinając się po drabinach, przeskakując przez wyrwy w konstrukcjach oraz schylając się w klaustrofobicznych korytarzach. Fabułę posuwa do przodu komunikacja z inteligentnym systemem nawigacyjnym statku, odpowiedzialnym również za sterowanie systemami przejść i drzwi.

Przeszkadzają oczywiście tytułowi obcy – są ich dwa rodzaje – pełnowymiarowe ksenomorfy, z którymi kontakt kończy się natychmiastową śmiercią oraz mniejsze, wcześniejsze pająkowate stadia, które próbują zagnieździć się w ciele Ashley, odbierając jej energię. Dla swej obrony dzielna pani porucznik w trakcie wędrówki znajdzie karabin oraz granaty, którymi będzie przecierać szlak. Ilość obcych nie jest przytłacza-

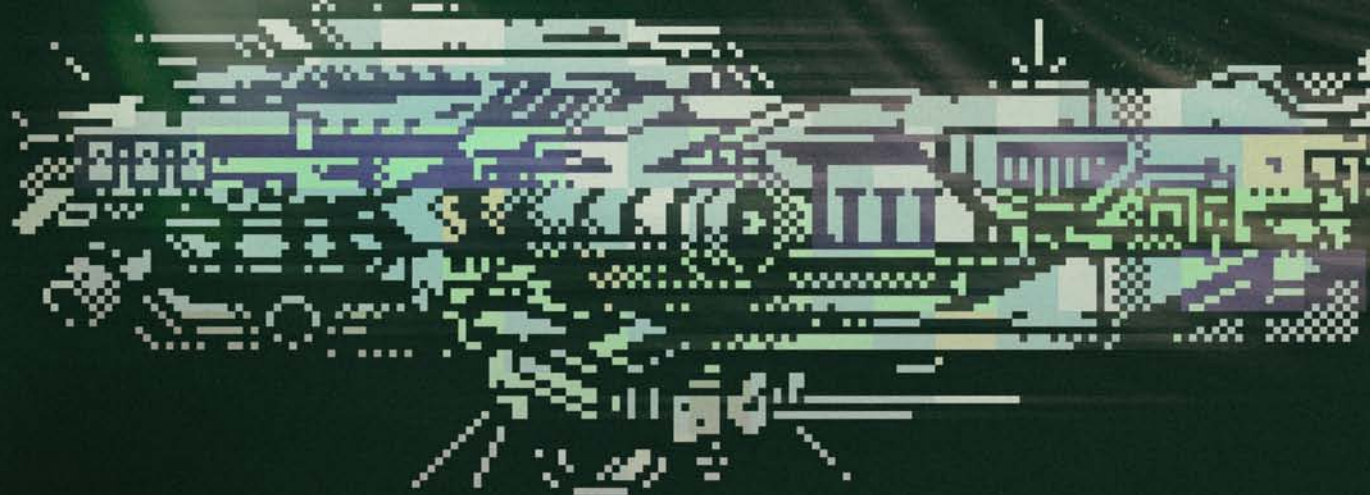
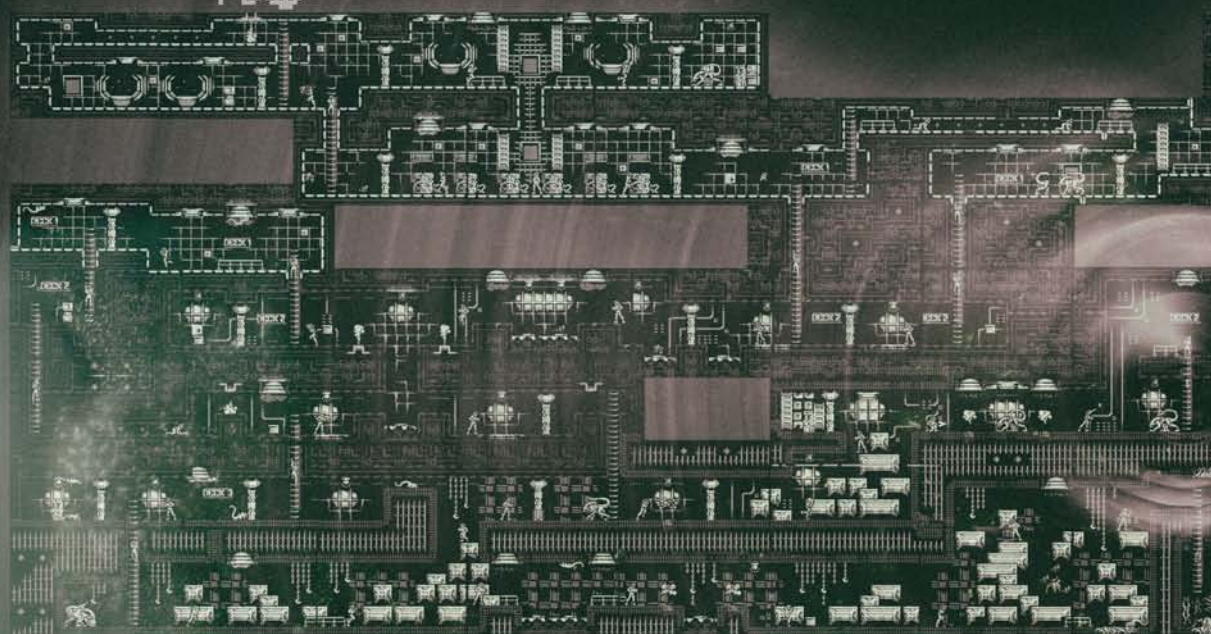
jąca, ale ciągle należy o nich pamiętać. Nieco trudne jest celowanie karabinem, szczególnie w młode obce, które są szybkie, ale dobrze służy to potęgowaniu poczucia niebezpieczeństwa w grze.

Dobrze wyważony został poziom rozgrywki, dokonano tego poprzez wdrożenie nowoczesnego podejścia do utrzymania płynności gamplay'u, które czynią ją przystępniejszą w stosunku do gier sprzed 30-40 lat. Autorzy zastosowali m.in. automatyczne checkpointy w postaci terminali komunikacyjnych, a także rozstawili po całym statku terminale uzbrojenia, pozwalające uzupełnić amunicję oraz terminale medyczne – umożliwiające podłączenie skołatanego ciała i nerwów.

Mapa gry – a więc i obszar rozrywki – jest całkiem pokaźna – składa się na nią 99 scrollowanych screenów zrealizowanych – jak mnie fachowo pouczył Tygrys – (thx!) w technice metatiles. Gra daje dwa możliwe zakończenia, w zależności od dokonanych wyborów. Długość gry jest według mnie adekwatna do dzisiejszych czasów – jej ukończenie będzie oczywiście zależało od tego, czy posługiwać się będziemy wyłącznie mechaniką przygotowaną przez autorów – a ta może dać w sumie ładnych parę

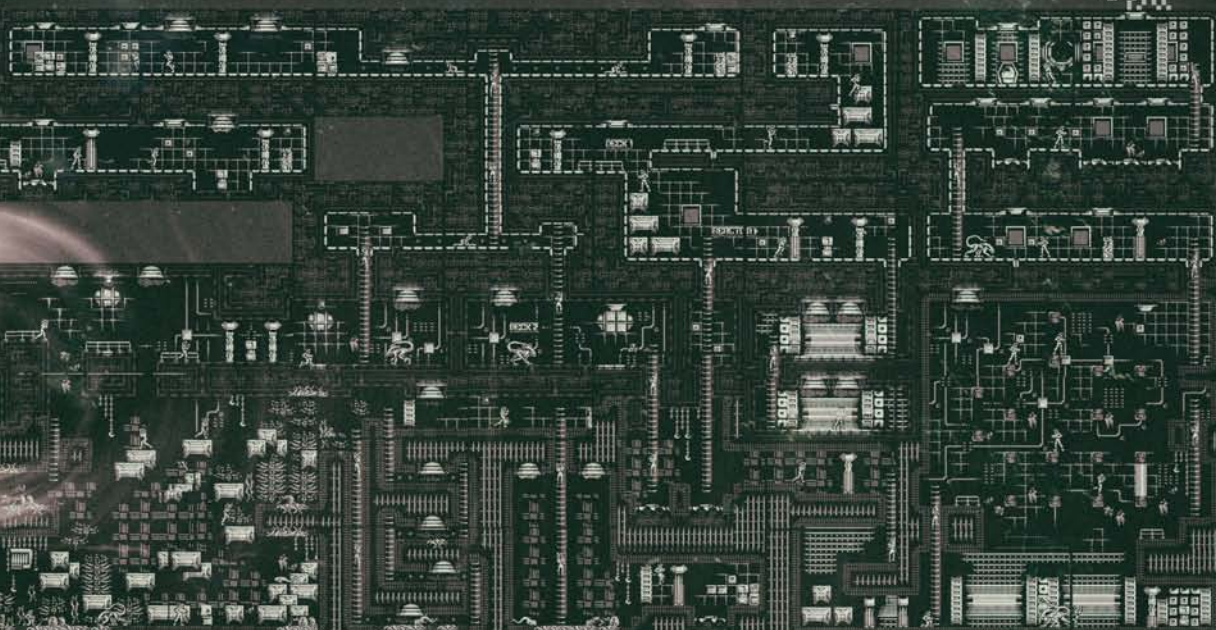


ALIENS





NEOPLASMA



ACHILLES



NAVIGATION
SYSTEM
SYSTEM TIME: 2169
1st of March 25 minutes
18 hours

OBJECT:
MULTIPURPOSE spaceship
"Achilles"

FLIGHT TIME:
2 years, 4 months

CREW:
15 persons

CARGO:
ore samples

created by chicaniii

godzin zabawy, czy też pomożemy sobie zapisywaniem stanu rozgrywki w emulatorze bądź DIVMMC... Wydaje mi się, że w dobie natłoku i wielkości różnorodnych produkcji – nie tylko komputerowych – takie podejście jest słuszne.

Parę słów o technikaliach – grafika jest różnorodna, zrealizowana kolorystycznie w ciemnych tonacjach, bardzo dobrze dobrana. Świetnie został oddany klimat statku kosmicznego – bogaty drugi plan z urządzeniami, oknami – szczególnie w górnej części statku. Nastrój wzbogacają animacje – migających lamp, wybuchów pary czy rotujących platform – dzięki temu statek faktycznie ożywa. Na oddzielną wzmiankę zasługują sprite'y obcych – przede wszystkim duże osobniki zostały wykonane kapitalnie, nie pozostawiając wątpliwości już na pierwszy rzut oka, jakiego uniwersum rzecz dotyczy. Animacje wszystkich postaci są bardzo płynne i szybkie, bez migotania czy lagów. Nieźle i szybko czyta się przerywniki tekstowe, które przewijają się błyskawicznie. Jest szybko i nowocześnie. Można się skupić na rozgrywce.

Grze towarzyszy zapętlony utwór muzyczny zrealizowany oczywiście na AY – jest on profesjonalnie wykonany, szybki i dynamiczny – tu mam w zasadzie jedną uwagę, że może lepiej brzmiałoby coś nieco bardziej niepokojącego w odbiorze i wolniejszego, ale to może być i kwestią gustu. Inną sprawą jest, że po początkowym nieśmiałym poruszaniu się – i nabraniu pewności w rozgrywce, daje się jej tempem dogonić tempo soundtracku, więc może ona w sumie do tego dopingować.

Summa summarum – gra jest świetna, to znów jedna z najlepszych produkcji na ZX Spectrum – ever – a wykonana kilka miesięcy temu... W ciekawych czasach przyszło nam żyć. Mam nadzieję, że idea ZX DEV M.I.A Remakes będzie kontynuowana, bo produkcje które dzięki niej powstały są naprawdę wyjątkowe. **Aliens: Neoplasma** jest oficjalnym zwycięzcą tego współzawodnictwa i każdym swoim aspektem udowadnia, że nie jest to przypadek. Kto nie zagra ten trąba. ■

ACHILLES NAVIGATION SYSTEM

Tytuł: **Aliens: Neoplasma**

Platforma: **ZX Spectrum 128KB**

Rok wydania: **2019**

Developer: **Sanchez crew**

Kod, design, dialogi: **Alexander Udotov**

Grafika i scenariusz: **Evgeniy Rogulin**

Audio: **Oleg Nikitin**

<https://zxonline.net/game/alien-game/>

UNIKALNE GRY DLA ZX Spectrum

PEWNE NAZWISKA W TYM ZESTAWIENIU BĘDĄ POWTÓRZENIEM Z PIERWSZEJ CZĘŚCI. NIC DZIWNEGO, NIEKTÓRE ZASKAKUJĄCE PROJEKTY MOGŁY POWSTAĆ TYLKO NA ZX SPECTRUM. OBOK KLASYKI LAT 80., OCZYWIŚCIE ZDOMINOWANEJ PRZEZ BRYTYJSKIE STUDIA TWORZĄCE GRY, TYM RAZEM MAMY TEŻ KILKA PEREŁEK Z ZUPEŁNIE INNYCH REJONÓW, NIEISTNIEJĄCEJ JUŻ CZECHOSŁOWACJI I POLSKIEJ RZECZPOSPOLITEJ LUDOWEJ. O ILE PRODUKCJA CZECHÓW JEST WYJĄTKOWO AMBITNA, TO PRODUKT Z POLSKI BAZUJE JUŻ NA ZUPEŁNIE INNYCH ZAŁOŻENIACH, JEDNAK NALEŻY O NIM CHOĆBY WSPOMNIEĆ. PONOWNIE, W ZESTAWIENIU UWZGLĘDNIAM TYTUŁY WYDANE W LATACH 80. I 90., UZNAJĄC NOWE TYSIĄCLECIE ZA OKRES POWROTU DO ZABAWY NA ZX SPECTRUM, KTÓRY NIE MIAŁ JUŻ ZNACZENIA Z PUNKTU WIDZENIA WYDAWCÓW (NO CHYBA, ŻE MÓWIMY O WYDANIACH KOLEKCJONERSKI, W ZGRABNYM PUDEŁKU I NA KASECIE, JEDNAK TO WCIĄŻ HOMEBREW). Z DRUGIEJ STRONY, CZYM TAK NAPRAWDĘ RÓŻNIŁO SIĘ WIELE OFICJALNYCH WYDAŃ GIER Z UBIEGŁEGO WIEKU, OD TEGO Z CZYM MAMY DO CZYNIEŃ DZIŚ?

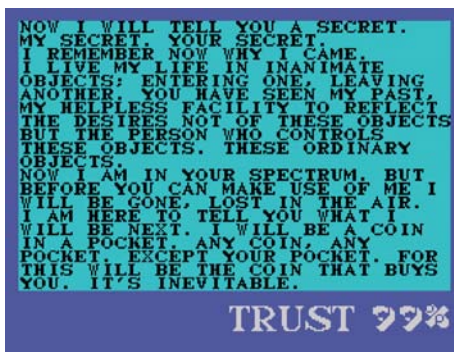
The Great Space Race (1984, Legend)



Brytyjskie studio Legend przed premierą *The Great Space Race* miało już w swoim dorobku jeden przebój - *Valhallę* - fabularną grę w klimacie fantasy, w której większość rozgrywki prowadziła się w okienku tekstowym. *The Great Space Race* zbudowano na bazie podobnego zestawu narzędzi (powiedzmy silnika zbudowanego w znacznej części w BASIC-u), jednak ten kosmiczny twór nazwano już „prawdziwym, komputerowym filmem”, a sama rozgrywka bardziej przypominała proste gry strategiczne. O ile oprawa graficzna i opowieść rzeczywiście ma filmowy charakter - przeloty pomiędzy stacjami kosmicznymi ilustrowane są animacjami, podobnie jak walki i naprawy pojazdów, to już sama mechanika rozgrywki jest bardzo uproszczona. Po wyborze pilotów i wyposażeniu ich pojazdów, możemy jedynie decydować o kierunku lotów (mających na celu przewożenie towaru), atakowaniu przeciwników i płaceniu podatków oraz naprawach pojazdów. Niekiedy trafi się okazja, by przejąć ładunek, jakiegoś dryfującego w przestrzeni pojazdu, ale takie próby obarczone są ryzykiem wyboru odpowiedniego kodu do bomb, za-

instalowanych na takich statkach (te da się zdobyć podczas podróży). Koniec gry następuje w momencie utraty całej floty. Z tej okazji gra częstuje nas porcją statystyk. Na fali fascynacji *Gwiezdnymi wojnami* niewątpliwie *The Great Space Race* przyciągał do monitorów. Z dzisiejszej perspektywy to prosta strategia (która będzie toczyć się i bez naszej interakcji), za to bogato ilustrowana animacją.

iD (1986, Nu Wave Software/CRL Group)



Tekstowa pogawędka ze starożytnym bytem, który trafił na trzecią planetę Układu Słonecznego i opanował ZX Spectrum, a teraz próbuje poskładać się w jedną całość. Cała rozgrywka sprowadza się do zdobycia zaufania tytułowego Id, poprzez trwającą nawet kilka godzin rozmowę. Za stworzeniem tej gry stoi Mel Croucher (właściciel Automata UK Ltd, współtwórca: *Automonopoli*, *Castle Master*, *Chambers of Death*, *Creepy Dungeons*, *Crusoe*, *Deus Ex Machina*, *Drunk Policeman* i *The Egg*) oraz Colin Jones (pracował przy: *Grange Hill*, *Paradise in Microdot*, *Rockstar Ate My Hamster*, *Sarlmoor* i *Slightly Magic*). W ostatecznym rozrachunku dostajemy dosyć zaskakującą miksturę roz-

mowy z Elizą, nacechowaną odrobiną historii i mistycyzmu, która po zakończeniu rozgrywki wypłyje na ekran wszystkie fakty, jakie próbaliśmy jej wpoić.

The Incredible Shrinking Fireman (1986, Mastertronic)



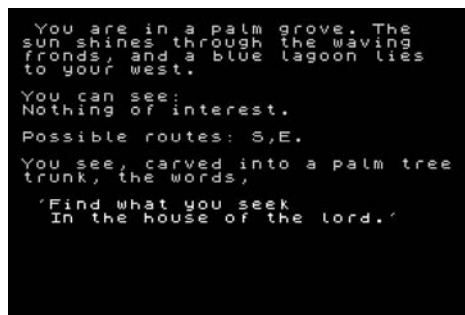
Zręcznościowa przygodówka, wymagająca główkowania przy przenoszeniu z miejsca na miejsce kilkunastu przedmiotów, rozrzuconych po kilkudziesięciu planszach remizy strażackiej. Cel gry to przywrócenie naturalnych rozmiarów strażakowi nazwanemu Shuffling Sid the Fearless Fireman, który ucierpiał na gabarytach wpadając podczas gaszenia pożaru do maszyny zmniejszającej. Jedyny ratunek to odnalezienie wszystkich części maszyny rozciągającej i odwrócenie procesu. Całość sprowadza się do chodzenia i skakania po wspomnianej remizie i omijania ruchomych przeszkód. Generalnie *The Incredible Shrinking Fireman* to ubogi krewny *Jet Set Willy* oraz serii *Magic Knight* (również wydanych przez Mastertronic), który nie porywał ani zagadkami, ani dynamiką rozgrywki. Grę napisał Andy Michell, a za jej oprawę graficzną odpowiada David Kidd.

Invasion of the Body Snatchas! (1983, Design Design Software/Crystal Computing)



Duet złożony z Simona Brattela i Neila Mottersheada na tworzeniu dynamicznych gier z grafiką wektorową dobrze się zna (zobaczcie *Dark Star*), więc nie jest zaskoczeniem stworzenie przez tych samych panów nieoficjalnego, ultraszybkiego portu *Defendera*. Za zmienionym tytułem (*Invasion of the Body Snatchas!*) i nieco zmodyfikowaną w stosunku do pierwowzoru fabułą kryje się jedna z najbardziej dynamicznych strzelanin z płynnie przesuwającym ekranem, jaka powstała dla 48-kilobajowego ZX Spectrum. Gra obsługuje Fuller Boxa, przez co możemy usłyszeć efekty dźwiękowe wydobywające się z AY-ka oraz zagrać joystickiem (sterowanie klawiaturą trudno opanować). W zasadzie rozgrywka w *Body Snatchas!* dokładnie kopiuje zasady zabawy w *Defendera*, z jego charakterystycznym radarem widocznym w górnej części ekranu i ratowaniem ludzików (płci pięknej) z objęć żarłoczych kosmitów. Również ciągnące się przez cały ekran ślady pocisków i efektowne wybuchy skopiowano z pierwowzoru. Ta gra traci na powabie jedynie wtedy, gdy zatrzymamy się na chwilę w miejscu.

The Island (1983, Virgin Games Ltd.)



Virgin na początku kariery ZX Spectrum nie był szczególnie znanym dystrybutorem gier. Większość wydanych przez Virgin Games Ltd. w pierwszej połowie lat

Jet-Story (1992, Ultrasoft)



Pochodząca zza naszej południowej granicy produkcja, na pierwszy rzut oka przypomina *Cybernoida*. Detale zdradzają jednak, że to znacznie późniejsza gra, oferująca też nieco bardziej złożoną mechanikę rozgrywki i skrzące się od kolorów plansze. Autorzy *Jet-Story*, pochodzący z ówczesnej Czechosłowacji Miroslav Fiedler i Frantisek Fuka mają na swoim koncie wiele gier tekstowych, tworzonych na lokalny rynek (w tym przygody Indiany Jonesa), jak i sporo narzędzi (edytory czcionek, edytor tekstu). *Jet-Story* może kojarzyć się z wcześniejszym tytułem tego samego duetu *Planet of Shades* z 1986 roku. Sam *Jet-Story* świetnie maskuje color clash, obudowując plansze kolorowym tłem i zmieniając barwy tylko obiektów poruszających się na tle elementów do zebrania lub zniszczenia. Sama rozgrywka to przedzieranie się przez złożone tunele, pełne lewitujących i osadzonych na skałach przeciwników. Na nasz pojazd działa grawitacja, a uzbrojenie, osłony i paliwo trzeba od czasu do czasu uzupełniać, sięgając do zasobów rozrzuconych po pokaznej mapie gry.

80. gier do tekstówki i proste strategię. Za trzy z nich odpowiada Martyn Charles Davis, który w przypadku *The Island* zbudował fabułę wokół poszukiwania skarbów na wyspach. To typowa gra tekstowa, pozbawiona ilustracji, za to odtwarzająca dźwięki odpowiadające niektórym wydarzeniom. Oprócz przechadzania się po wyspach, oferuje również krótkie zręcznościowe przerywniki, jak ucieczka spod palm, czy nawigacja wśród raf koralowych. Łatwo pomylić *The Island* Martina Davisa z innymi grami o tym samym tytule. W 1983 roku własną grę *The Island* wydaje Crystal Dynamics (ZX Spectrum), jak i wydane przez The Guild i M. Hagana jeszcze dwie inne wyspy dla ZX Spectrum. Żeby było jeszcze trudniej trafić na właściwy tytuł, w Polsce powstała kolejna gra pod tym samym tytułem, to *The Island* Piotra Kuciapskiego, wydany przez L.K. Avalon w 1996 roku.

Jason's Gem (1985, Mastertronic)

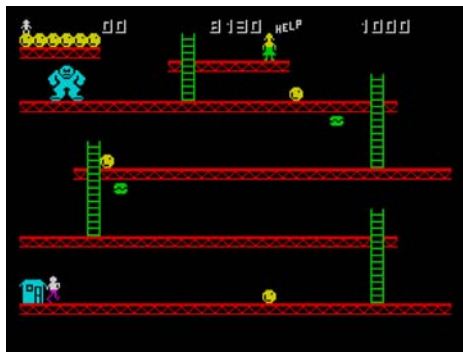
Prosta, ale zgrabnie zaprojektowana przez Simona Whyte'a gra, który prowa-

dzi nas do świata podróży kosmicznych i poszukiwań zaginionych klejnotów. W skórze głównego bohatera, Jasona,



musimy najpierw przebić się przez kilka warstw skał i posadzić na lądowisku statek kosmiczny, a potem już w kombinacji kosmicznej przedzierać się przez groty. Wszystko to trzeba zrealizować pod presją niknącego w oczach paska punktów bonusowych, skacząc od teleportu do teleportu, które prowadzą wprost do klejnotu. Nie jest to może produkcja najwyższych lotów, ale zarówno wykonanie gry, jak i oprawa graficzna, czy animacja obiektów nie pozwala się zbyt jej cześcić.

Killer Kong (1983, Blaby Computer Games)



Gary Capewell, twórca tej gry, ma na swoim koncie wiele nieoficjalnych konwersji przebojów z automatów arcade (od *Pac-Mana* po *Mr. Do!*). W tym zestawie inspiracji salonami gier wideo nie mogło zabraknąć *Donkey Konga*, który pojawił się na ekranach dwa lata wcześniej. *Killer Kong* cierpi jednak na zbyt wiele uproszczeń i toporną rozgrywkę. Zabrakło ikonicznego młota do pojedynkowania się z beczkami, za to dodano do mechaniki gry opcję zjadania hamburgerów (podnoszących punktację) i pięć różniących się układem plansz. Do panteonu najlepszych konwersji *Donkey Konga*, *Killer Kong* nie ma szans trafić.

King Arthur's Quest (1984, Five Ways Software Ltd/Hill MacGibbon)

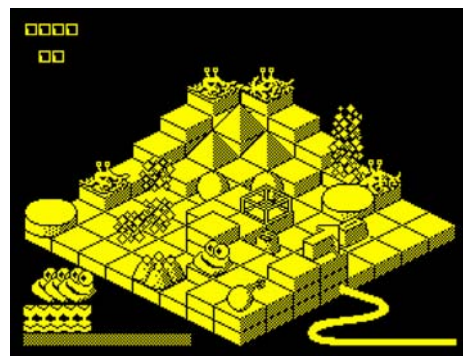


Five Ways Software odpowiada m.in. za stworzenie: *Strike Force Cobra*, *Run for Gold*, czy *Fire on Water* oraz jednej gry, zrealizowanej w podobny sposób jak *King Arthur's Quest* - *Aztec: Hunt for the Sun-God*. Mity arturiańskie musiały, prędzej czy później, zainspirować twórców gier do napisania własnej wersji wydarzeń, zilustrowanej na ekranie komputera. *King Arthur's Quest* to tytuł, który łączy elementy gry tekstowej z *Dungeon crawlerem* - całe sterowanie sprowadza się do wyboru kilku klawiszy (zgrabnie opisanych na dołączonej do kasety nakładce), biega-

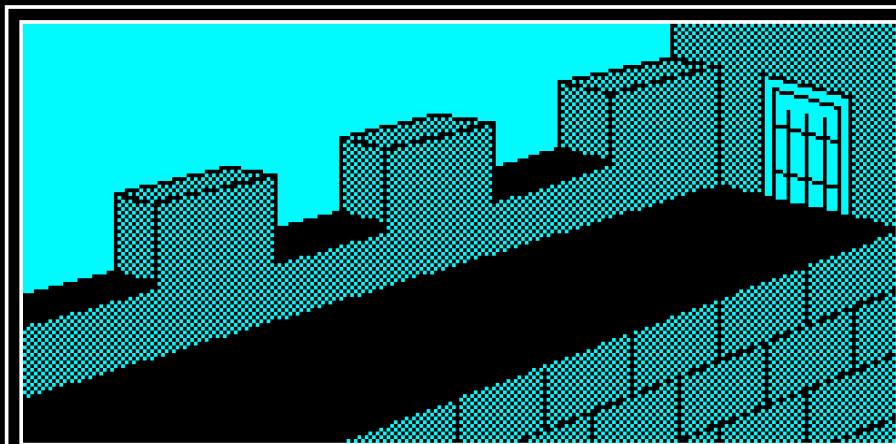
nia po labiryntach, zbierania i używania przedmiotów oraz rzucania czarów, co wymagało więcej zręczności niż pisania (jak w typowych grach tekstowych). Zupełnie nieszablonowy jest za to widok, który obserwujemy podczas gry - to obraz z perspektywy pierwszej osoby, zrealizowany podobnie jak w większości *dungeon crawlerów*, uwzględniający odległość przedmiotów i postaci.

Kirel (1986, Zegfried Kurz/Addictive Games Ltd)

Na pierwszy rzut oka *Kirel* to jedna z wielu gier powstałych na fali popularności *Knight Lore'a* i *Marble Madness*. Nic bardziej mylnego. Pomimo widoku izometrycznego *Kirelowi* bliżej do *Sentinela* i innych gier strategicznych, niż do typowych zręcznościówek, czy przygodówek arcade. Otóż tytułowy *Kirel* to stworzenie zdolne do pochłaniania pojedynczych klocków, z których zbudowane są plansze gry. Może je połykać, przenosić i zostawiać w innych miejscach. Sam



nie porusza się zbyt sprawnie - wskoczy zaledwie o poziom wyżej lub zeskoczy w dół o jeden stopień (no chyba, że ma ze sobą jeden klocek). W bieganiu po planszy przeszkadzają mu poruszające się losowo stworzenia - można je blokować lub połykać (o ile dysponujemy zasobem ciastek). Zetknięcie z nimi skracą pasek energii *Kirela*. Celem gry jest dotarcie do znajdujących się na planszy bomb i rozbrojenie ich. I tak przez 70 zróżnicowanych poziomów. Dodajmy jeszcze, że dla wygody możemy obracać obraz o 90/180/270 stopni. Również



southwest tower.
What now ? n
You are by an iron gate, which
is to your north. Looking west
over the wall, you can see the
Altivian mountains. Below is the
courtyard.
What now ? examine gate

Karyssia: Queen of Diamonds (1987, Incentive Software Ltd.)

Studio Incentive Software stworzyło m.in. *The Graphic Adventure Creator*, 3D *Construction Kit* oraz kilka znakomitych gier z wektorową grafiką: *Drillera*, *Total Eclipse* i *Castle Mastera*. *Karyssia* to jedna z wielu gier zrealizowanych za pomocą *The Graphic Adventure Creator* przez Darrena i Roberta Shacklady'ego. Wspólnie stworzyli opowieść o złej królowej *Karyssii*, którą mamy zgładzić na żądanie prawowitej dziedziczki królestwa - *Loranin*. Nie jest to może szczególnie odkrywcza pod względem fabuły tekstówka, jednak jeden element wyróżnia *Karyssię* - sposób prezentacji grafiki, rysowanej w rzucie izometrycznym.

w trybie pauzy Kirel ujawnia swoje położenie, gdybyśmy zagubili go na planszy. To jedna z najciekawszych, zapomnianych od dekady strategii rozgrywanych w czasie rzeczywistym. Jeśli nam się poszczęści i zdobędziemy elementy konstrukcyjne na planszy, będziemy mogli budować mosty.

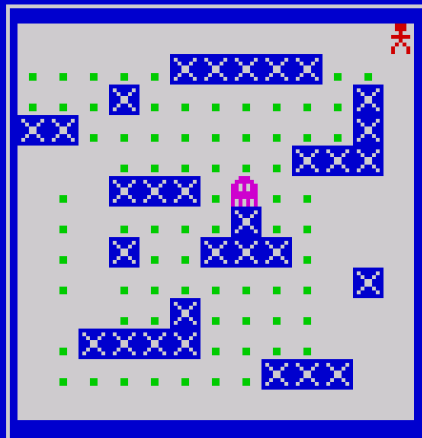
Knot in 3D (1983, New Generation Software)



Malcolm Evans rozpoznawany jest jako prekursor gier 3D dla ZX80/81 - w tym niezapomnianego *Monster Maze 3D*. Jego kariera związana z grami dla ZX Spectrum również oscylowała wokół tworzenia wrażenia, iż świat widziany na ekranie komputera jest trójwymiarowy, jak w *Corridors of Genon* i *3D Tunnel*. Zbiór najważniejszych gier dla ZX Spectrum Evansa z trójwymiarowym widokiem uzupełnia *Knot in 3D* - dziwaczne wyścigi przypominające *Trona*, z widokiem z perspektywy pierwszej osoby, rozgrywane się na planszach o wymiarach 16x16x16 pól. Owszem, trzeba sporo wyobraźni, by poruszać się po tym malowanym w trakcie rozgrywki labiryncie. Prawdopodobnie była to najgorsza z trzech gier z trójwymiarową grafiką Evansa, jednak zrekompensował nam to rok później *Trashmanem*.

Magic Meanies (1983, CDS Microsystems)

Ten tytuł to odpowiedź na zapotrzebowanie rynku, czyli nieoficjalna konwersja *Mr. Do!* dla ZX Spectrum. Zrealizowana z klasą, choć wyłącznie na podmienionym zestawie znaków, stąd i sztywna wielkość plansz: 30x20 pól. Gra oferuje większość z elementów mechaniki rozgrywki *Mr. Do!* takich jak: zbieranie diamentów, rzucanie jabłkowych bomb na przeciwników, nieodzowną kulkę, służącą do zestrzeliwania przeciwników oraz bonusy, pojawiające się czasami na planszy. *Magic Meanies* wprowadza pewnie zmiany do opowieści - zamiast klauna sterujemy



WYNIK: 64

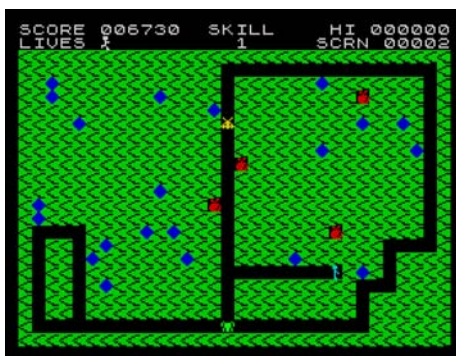
REKORD DNIA:

EWA 1000

GRA PORNO : KROLEWNA JEBACZKA

Królewna Jebaczka (1984, X-PROG)

Jedna z pierwszych, polskich gier wydanych dla ZX Spectrum. Zapewne zupełnie nieoficjalnie i sprzedawana na giełdach spod lady. Do dziś nie znamy autora, ukrywającego się za firmą X-PROG, za to wielu posiadaczy ZX Spectrum w latach 80. miało szansę zetknąć się z tą (jak określił ją autor) grą pornograficzną. I częściowo jest to prawda, bo sekcja „Instrukcje” zawiera tak naprawdę opowiadanie pornograficzne w klimacie fantasy, takie o złych duchach i dużej dawce nasienia. Sama rozgrywka jest już mniej nacechowana erotyzmem, przynajmniej w trzech pierwszych (powtarzanych do znudzenia) poziomach, gdzie zbieramy kropki w grze, przypominającej *Pac-Mana* i uciekamy do wyjścia. Ostatni poziom to okazja do sprawnego wycelowania penisa w tytułową królową. Reszta pozostaje w wyobraźni (nastoletniego) gracza. I przechodzi do historii polskich gier.



magiem, który rzuca w przeciwników szklaną kulą. Gra wydana na ZX Spectrum 16K może zaskoczyć jednym, autor sprytnie zaimplementował odtwarzanie muzyki na brzęczyku, w trakcie rozgrywki.

Mantronix (1986, Dominic Wood/Probe Software Ltd)

Zachwyt wykorzystaniem rzutu izometrycznego na ZX Spectrum w celu osią-

gnięcia efektu pseudo trójwymiarowego otoczenia, w połowie lat 80. osiągnął apogeum. Oprócz najbardziej rozpoznawalnych tytułów jak *Batman*, czy *Fairlight*, w tym okresie powstało także wiele gier, o których łatwo zapomnieć w natłoku izometrycznych wrażeń. Stworzony przez Dominica Wooda *Mantronix* to dynamiczna strzelanina zrealizowana w takim właśnie stylu. Fabułę można pominąć - to coś o tytułowym robocie wysłanym na planetę Zybor, w celu zgłazdzenia czworga kosmicznych zbrodni-



rzy. Szybka, wciągająca, pełna akcji, niekoniecznie wymagająca główkowa-
nia. W swoim czasie planowano port
Mantronixa na Amstrada i C64 (co wi-
dać było na reklamach Probe Software),
ostatecznie gra pozostała exclusive'em
dla ZX Spectrum. Sam Dominic Wood
przyczynił się jeszcze do stworzenia
portów: *1942*, *Berzerka* (*Cybotron*),
Pac-Mana (*Ghost's Revenge*), *Pengo*
(*Pengy*) oraz *Vlza* i *RoboCopa 3*.

Megapede (1983, Andrew Beale/Softek)



Na fali konwersji gier z automatów po-
wstał i ten klon *Centipede*, z drobnymi
modyfikacjami w rozgrywce i wyglądzie
(dostosowanym do możliwości ZX Spec-
trum 16K). Oczywiście strzelało się w tej
ultraszybkiej wersji *Centipede* do prze-
rośniętej stonogi, rozczłonkowały ją
na wiele niezależnie poruszających się
części, jednocześnie bacząc na pająki
i (tu nowość) skorpiony. Gra w całości
została napisana w kodzie maszyno-
wym, doczekała się również kontynuacji
ze stawonogiem w roli głównej, stworzo-
nej przez tego samego autora, Andrew
Beale'a i zatytułowanej tak samo jak
pierwowzór: *Milipede*.

Micro Mouse Goes De-Bugging (1983, Steve Hughes/MC Lothlorien Ltd)

Na początek drobne wyjaśnienie:
istnieje oficjalny port *Micro Mouse* dla
C64, jednak ze względu na charakter
rozgrywki, to gry diametralnie różniące
się pod względem listingów widocznych
na ekranie. O tym właśnie jest ta gra,
bawimy się w uzupełnianie brakujących
elementów programu w BASICu, wsta-
wiając na właściwe miejsca znaki roz-
rzucone w rogach ekranu. Po poprawie-
niu kodu możemy obejrzeć efekt jego
działania na ekranie. W wersji dla ZX
Spectrum ma się rozumieć operujemy
na ZX BASIC-u, w przypadku C64 - na
programach w BASIC 2.0. W 1989 roku
powstała jeszcze jedna gra, pod tym



samym tytułem, wydana przez Master-
tronic. Tym razem *Micro Mouse* napra-
wiała płyty główne maszyn. I ta gra do-
czekała się portów dla Amstrada i C64.

Molar Maul (1983, John Gibson/Imagine Software)



Prosta gra zręcznościowa o walce
z bakteriami panoszącymi się w jamie
ustnej. Do wykonania zadania dostajemy
porcję pasty do zębów, a nasi wrogo-
wie, gnieźdząc się początkowo na języ-
ku, atakują zęby. Jedyna rada - nacierać
pastą. Najpierw miejsca zaatakowane
przez bakterie, później te, które wyma-
gają doczyszczania. Za każdym razem
musimy dbać o nabieranie porcji pasty
i precyzyjne celowanie w zęby. Przejście
każdego z etapów wiąże się z wyczerpa-
niem tubki widocznej w górnej części
ekranu, przegrana to kilka zębów zeżar-
tych przez próchnicę. Ta w gruncie rze-
czy wciągająca zręcznościówka o szo-
rowaniu zębów przypomina nieco *Pssst*.
Zadziała nawet na ZX Spectrum 16K.

Moley Christmas (1987, Gremlin Graphics Software Ltd)

Nieco mniej znany epizod opowieści
o Montym Mole'u, zrealizowany przez
stałą ekipę Gremlina od przebojów
(Shawn Hollinworth, Peter M. Harap,
Chris Kerry). Złożona zaledwie z sze-
ściu plansz zręcznościówka, o uwiel-
bianym na ZX Spectrum kreciku, nie
jest szczególnie wymagająca, to raczej
produkt reklamowy, wydany na okładko-

wej taśmie, dołączonej do *Your Sinclair*.
To co go wyróżnia to muzyka, wsparta
motywem znanym z pierwowzoru (*Auf
Wiedersehen Monty*), zagranym na AY.
Moley Christmas to rekurencja w pełnej
krasie: fabuła o przygotowywaniu gry,
poczynając od zbierania kodu źródłowe-
go w biurach Gremlina, przygotowywa-
niu taśmy matki, dostarczaniu mastera
do biura *Your Sinclair*, przygotowania
gry do dystrybucji, aż po przeniesienie
finalnego produktu do kiosków. Redak-



cja *Your Sinclair* powiązała wdanie gry
z małym konkursem - kto pierwszy od-
czytał komunikat wyświetlany na końcu
gry i poinformował o tym redakcję, do-
stawał gratis 15 gier z biblioteki wydaw-
nictwa.

Monsters in Hell (1983, Softek)



Nieco uproszczona wersja *Lode Run-
nera* dla ZX Spectrum 16 KB, w której
nasz główny bohater prowadzi nierów-
ną walkę z wampirami. Po pierwsze nie
może dać się „wysać” (co kosztuje
życie), po drugie musi co pewien czas
uzupełniać zapasy świętej energii, sym-
bolizowanej przez krzyże. Cała zabawa
sprowadza się do wybijania dziur w pod-
łożu, w które wpadają potwory i zgrab-
nie roztrzaskują się o to, co leży na dnie.
Na niektórych twardzieli trzeba poświę-
cić nieco więcej wysiłku. Na samym
dole ekranu znajduje się piekło w swo-
jej ognistej krasie. My musimy również
uważać, by tam nie wpaść. Gra Martina
Levisa, znana także pod tytułem *Panic*,
jest wzorowana na *Space Panic* z 1980
roku.

Monty Is Innocent (1985, Gremlin Graphics Software Ltd)



Druga z sześciu części przygód Monty'ego Mole'a, tradycyjnie z potężnym labiryntem komnat i zestawem zadań do wykonania. Po *Wanted: Monty Mole* mogliśmy spodziewać się takiego obrotu akcji - w grze nie sterujemy postacią Monty'ego, a jego przyjacielem, Samem Stoatem, który ma uratować kreta z więzienia Scudmore. Ot dynamiczna gra zręcznościowa, wymagająca dużo biegania i zbierania przedmiotów. Na tle późniejszych przebojów Gremlina, to raczej jeden z odcinków serii niż wyróżniająca się (poza zmianą głównego bohatera) gra o Montym Mole'u.

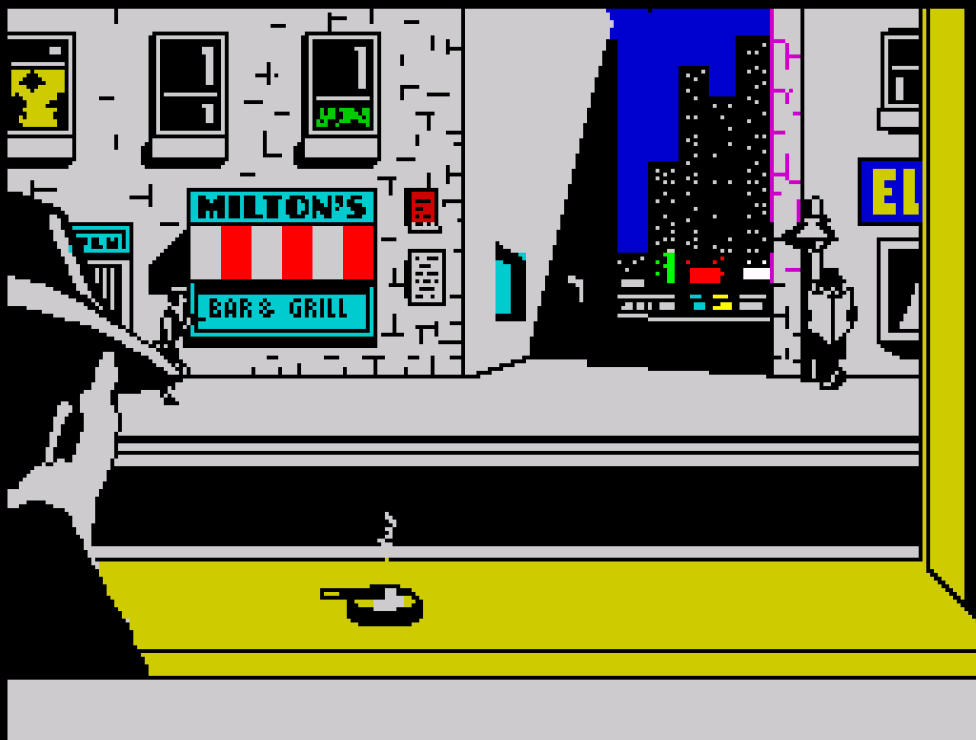
Moonlight Madness (1986, John F. Cain/Bubble Bus Software)



Aby uratować życie pewnego szalonego naukowca musimy przedzierać się przez jego posiadłość, złożoną z 43 pomieszczeń, odnaleźć 16 kluczy i kod do sejfów oraz wydobyć z wnętrza tego ostatniego pigułki. Zabawa opiera się na bieganiu z pomieszczenia do pomieszczenia, przechodzeniu przez kolorowe drzwi, naciskaniu przycisków, które otwierają mosty lub uruchamiają windy, unikaniu przeciwników i przeszkód oraz upadków z dużej wysokości. Johny F. Cain, twórca gry, powinien być dobrze znany posiadaczom ZX Spectrum - to on stoi za stworzeniem *Booty'ego* oraz

Mugsy (1984, Melbourne House)

Gra, której pierwsza część nigdy nie zaistniała na innej platformie niż ZX Spectrum, za to jej sequel trafił także na sąsiednie ekrany (Amstrad, C64). To pewnego rodzaju unikat i fenomen - komiksowy styl noir, świetnie wykorzystujący color clash, ilustruje opowieść o gangsterach, która tak naprawdę jest stosunkowo prostym menedżerem. Każda z tur w grze to jeden rok działalności tytułowego Mugsy'ego. I rok decyzji, na co przeznaczyć posiadane zasoby gotówki (łapówki, broń i amunicja, ochrona lokali). Gra stworzona przez Melbourne House nie bez powodu kojarzy się z innymi produkcjami australijskiego studia, jak chociażby *Hobbit*. Wykorzystano w niej autorskie narzędzie Melbourne Draw do tworzenia tych wszystkich wektorowych grafik, ilustrujących wydarzenia. Pomimo niezłego rozmachu wizualnego, cała interakcja gracza z Mugsym sprowadza się do wpisywania kilku liczb w każdej turze. Z jednym odstępstwem od reguły - gdy ktokolwiek zdecyduje się zlecić zabójstwo naszego bohatera, gra na chwilę zmienia swój dotychczasowy charakter na bardziej zręcznościowy.



*Potty Painter*a. *Moonlight Madness* trafił na okładową taśmę dołączoną do *Your Sinclair*a w 1991 roku, wraz z *Marsportem*, *Ninja Hamsterem* oraz *Wizard's Lair*. Krytycy zwrócili uwagę na wygórowaną cenę gry (7,95 Funt), jednak trudno nie dostrzec również muzyki, grającej na beeperze podczas całej rozgrywki, która to usprawiedliwia.

Mrs Mopp (1984, Tina Billet/Computasolve)

Niewątpliwie podstawowym celem, jaki przyświecał powstaniu *Mrs Mopp* była edukacja. Kogo? Krnąbrnych nastolatków, którzy nie potrafią utrzymać porządku w swoim pokoju, cały dzień spędzając przed komputerem. Tina Billet wyraźnie chciała coś powiedzieć wszystkim nabywcom *Mrs Mopp*. W grze sterujemy kobietą postacią,

będącą panią domu, mającą na głowie niekończące się porządki. Słowem sortowanie kolorowych przedmiotów, za pomocą odpowiednich przyrządów sprzątających i utylizacja śmieci.



Wszystko byłoby w porządku, gdyby nie dwie przeszkody - *Mrs Mopp* szybko się męczy (pomagają czary albo sherry), a nawałnica kolorowych śmieci czasami

może zablokować drogę do posprzątnia całego tego galimatiasu.

My Name is Uncle Groucho You Win a Fat Cigar (1983, Automata UK Ltd)



O mniej lub bardziej szalonych pomysłach Mela Crouchera już wspominałem, m.in. przy okazji gry *iD. Groucho* i przydługa nazwa gry również dobrze wpisuje się w profil wydawniczy studia Automata UK. W gruncie rzeczy *Groucho* to prosta gra tekstowa, utrzymana w formule quizu, który był jedynie pretekstem, do ogłoszenia konkursu na odgadnięcie właściwego bohatera gry, co prowadziło do zdobycia biletu na Concorde'a lecącego wprost do Hollywood. Wyniki zabawy ogłoszono 4 czerwca 1984 roku, zwycięzcą okazał się Phil Daley, który wracał do UK już na pokładzie Queen Elizabeth 2.

Ostron (1983, Andrew Glaister/Softek)



Gra opracowana przez Andrewa Glaistera na początku fali popularności ZX Spectrum, specjalnie dla modeli wyposażonych w 16 KB RAM. *Ostron* został wydany również pod tytułem *Joust* (to identyczna gra), bo de facto jest to po prostu próba przeniesienia hitu z automatów na ekran ośmiobitowca. Tytułowy *Ostron* to struś, na plecach którego mamy za zadanie toczyć boje z innymi, latającymi przeciwnikami. Do konfrontacji dochodzi w przypadku zderzenia się z oponentem. O zwycięstwie decyduje

to, kto ma wyżej uniesioną lancę. Aby nie było zbyt nudno, oprócz niebieskich przeciwników, w kolejnych etapach przyjdzie nam się zmierzyć z bardziej uciążliwymi wariantami, latających bojowników w innych barwach.

Out of the Shadows (1984, Mizar Computing)



Oryginalna gra fabularna, która oferuje nam na starcie do rozegrania kilka gotowych scenariuszy plus losowo generowane otoczenie, przeznaczone do eksploracji. Cała podróż sprowadza się do poruszania po miniaturowej mapce, pełnej budynków, lasów i przeciwników. Polecenia, zarówno te dotyczące ruchu, jak i walki wydawane są z klawiatury. Do większości z nich przewidziano skróty (np. A N co oznacza Attack North lub G NE, równoznaczne z Go North-East). Choć na pierwszy rzut oka *Out of the Shadows* przypomina gry tekstowe, cała rozgrywka toczy się w czasie rzeczywistym, co w praktyce sprowadza się do nerwowego wklepywania poleceń podczas walki. Na uwagę zasługuje oświetlenie - a raczej kąt widzenia postaci, dynamicznie rysowany podczas poruszania się po mapie. Coś jak w wydanym wiele lat później *Nox*. Gra rozpoznaje kilkanaście poleceń, możemy korzystać z ponad 50 przedmiotów i kilku czarów. Przewidziano nawet zakupy u lokalnych kupców.

Spectrum Safari (1983, A. J. Rushton/CDS Microsystem)

Jedna z tych gier, które najpierw trafiły do obiegu jako samodzielnie wydana produkcja, a nieco później doczekała się reedycji przez CDS Microsystem. Niewiele dobrego można o niej powiedzieć. *Spectrum Safari*, udający grę ekonomiczno-eksploracyjną z elementami przetrwania, to tak naprawdę zbieranie kilku prostych procedur w BASI-Cu, skłaniających nas do poszukiwania



misia Koala, przedzierania się przez labirynt złożony z krokodyli, czy kopania się z owcą. Dziwaczna, nierówna produkcja, w której czasami przyjdzie nam pohandlować, innym razem będziemy bawić się w kowboja strzelającego do węży albo w odtwarzanie sekwencji. Docelowo mamy wydostać się z wyspy pełnej jakże barwnych przygód. Na szczęście kontynuacja tej gry nigdy nie powstała.

Spellbound (1984, P.W. Norris/Beyond Software)



Nie należy mylić tej gry ze słynnym *Spellboundem* z 1995 roku, to zupełnie inny produkt. Zgrabny, choć nie rewelacyjny klon arcade'owego *Q*berta*. Tym razem pod wpływem czaru zostajemy przemienieni w ropuchę, by skakać po schodach i zapalać poszczególne elementy. I tak do skutku. Niewiele tu myślenia, to raczej czysta akcja, nakręcana przez napływające fale magicznych przeciwników i skracający się pasek czasu.

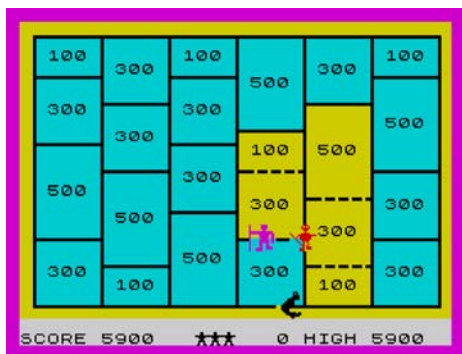
Styx (1983, Matthew Smith/Bug-Byte Software Ltd)

Pierwsza gra dla ZX Spectrum opracowana przez Matthewa Smitha, tego samego, który nieco później zasłynął dzięki *Manic Minerowi* i *Jet Set Willy*. *Styx* nie ma jednak zbyt wiele wspólnego z późniejszymi hitami. To prosta, zręcznościowa strzelanina, rozgrywająca się na jednym ekranie, podzielonym na trzy sekcje/obszary rozgrywki, przez które



przedzieramy się uzbrojeni w laser. Na krańcu ekranu oczekuje boss, a cała zabawa może nam zająć niewiele ponad minutę. Trochę za mało jak na komercyjny produkt.

Potty Painter (1983, John F. Cain/Rabbit Software Ltd)



Mało kto kojarzy dziś grę z automatów arcade, która zmuszała graczy do biegania wokół prostokątów i tym samym zdobywania kolejnych obszarów rozgrywki. Coś jak *Pac-Man*, tylko bez kropek do zbierania. Twór ten zwał się *Amidar* i posłużył jako inspiracja do stworzenia *Potty Paintera* - gry o podobnych zasadach, w której w skórze goryla uciekamy przed parą tubylców, kreśląc wzorki wokół prostokątów, tym samym zajmując kolejne części planszy. Jediną bronią, którą dysponujemy jest zamrażarka (łącznie trzy sztuki), umożliwiająca zatrzymanie przeciwników na 10 sekund. W kolejnych etapach doprowadzamy pluszaka do banana, malujemy ścieżki do wyczerpania farby i powtarzamy poziom pierwszy z większą liczbą oponentów na planszy. *Potty Painter* pomimo prostej oprawy graficznej, to gra równie wciągająca jak jej bliski kuzyn - *Oh Mummy*. Podobną do niej grę znajdziemy również na ZX Spectrum+ *User Guide Companion Cassette* - taśmie dołączonej do ZX Spectrum+. Tam znajduje się m.in. *Maze Game* stworzona przez Pete'a Cooke'a, tego samego, który odpowiada za tak wielkie dzieła jak: *Tau Ceti*, *Academy*, *Stunt*

Stonkers (1983, Imagine Software)

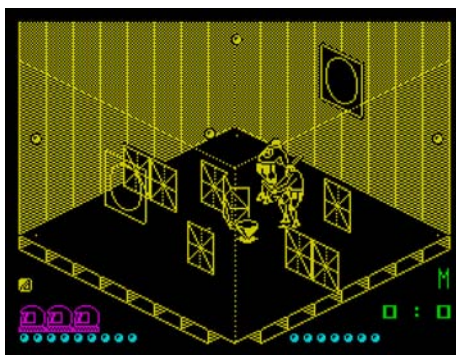
Być może będzie pewnym zaskoczeniem odkrycie pierwszej strategii rozgrywanej w czasie rzeczywistym niemal dekadę przed *Dune II*. Gra opracowana przez Johna Gibsona i Paula Lindale oferuje niemal wszystko to, co w RTS-ach najlepsze - jednostki bojowe i niezbędne do ich funkcjonowania zasoby, potyczki na czołgi, działa i piechurów. Plus podgląd większej mapy w razie konieczności. W praktyce to jednak twór zbyt powolny i mocno niegrywalny, za to (pomimo bugów, powodujących wykładanie się gry), całkiem zgrabnie opracowany wizualnie i imponujący tymi wszystkimi pojazdami i jednostkami poruszającymi się równolegle, w rytm naszych rozkazów.



INFANTRY DIVISION
SUPPLY STRENGTH...87
COMBAT STRENGTH...37
MOBILITY.....GOOD
AWAITING ORDERS

Car Racer, *Micronaut One*, czy opisany w poprzednim zinie *Earthlight*.

Play for Your Life (1988, Your Sinclair)



U schyłku swojej popularności ZX Spectrum doczekał się zmiany modelu dystrybucji gier. Po koniec lat 80. znaczącą rolę jako wydawca odgrywał *Your Sinclair*, dla którego tworzyły największe dotychczasowe studia (jak Ocean). Coraz chętniej bogaty zachód sięgał też po programistów ze wschodu. Tak było

i w przypadku *Play for Your Life*, opracowanego przez Serbów Dosko Dimitjevica i Dragoljuba Andjelkovica. *Play for Your Life* to coś w rodzaju oglądanego w rzucie izometrycznym tenisa, w którym rolę przeciwnika odgrywa komputer. Ta rozbita na 26 etapów (różniących się wystrojem, kolorem i układem przeszkód oraz liczbą piłek) rozgrywka nie opowiada jednak o zmaganiach sportowych. A przynajmniej nie jest to jedyna metoda na osiągnięcie celu - albo wbijamy przeciwnikowi trzy bramki, albo okładamy go kijem tak długo, aż wyzionie ducha (choć o to może być trudno, bo na polu boju przebywają roboty). Niewątpliwie *Play for Your Life* zasłużyło na uznanie za dobrze wykonaną oprawę graficzną.

... CDN

READY

Memoirs of a Spectrum Addict

ANDY REMIC - O POWODACH STWORZENIA FILMU
„WSPOMNIENIA UZALEŻNIONEGO OD SPECTRUM”
Z UDZIAŁEM: JOFFA SMIFFFA, JONA ROMERO,
MARKA R. JONESA, SIMONA BUTLERA I CLIVE'A TOWNSEDA.



OLIVER
FREY



RaM Films
spectrumaddict.com

Powody dla których zrealizowałem film dokumentalny *Wspomnienia uzależnionego od Spectrum* (2017) nie są tak oczywiste, jakby się to mogło wydawać. Pomysł miał zabrać mnie na kolejną przygodę, która obejmowałaby dziesięciolecia, przekraczała kontynenty, mogłaby sprowadzić na mnie wiele kłopotów, ale jednocześnie pozwoliłaby mi dotrzeć do przywileju spotkania wielu bohaterów mojego dzieciństwa. Było to jak spotkanie gwiazd rocka, innowatorów z mojej przeszłości, którzy doprowadzili mnie do mojego własnego uzależnienia od Spectrum. I dla mnie oczywistego, ponieważ jako dzieciak pragnąłem gier i spędzałem każdą chwilę na jawie grając w nie, programując na Spectrum, czytając o nim czasopisma lub rozmyślając o tym pięknym, czarnym, gumowo-klawiszowym „cosiu”. Jego niesamowity urok wynikający z prostoty formy, podkreślał klasyczny aluminiowy panel górny oraz mażnięcie kolorów w prawym dolnym rogu i niewyobrażalna ilość danych, które zdobiły każdy klawisz oraz przestrzeń pod i nad nim (Biedny Rick Dickinson! Jak później mi to wyjaśnił podczas wspaniałego wywiadu, umieszczenie tak dużej ilości informacji na tak małej klawiaturze stało się dla niego logistycznym koszmarem – ale to temat na kolejny wywiad i zupełnie inna historia...).

Mój przyjaciel z dzieciństwa, którego znałem od urodzenia (dorastaliśmy mieszkając obok siebie, był o miesiąc starszy ode mnie, więc razem dreptaaliśmy, przejmowaliśmy zjeżdżalnię w pokoju dziecięcym, bawiliśmy się razem na polach, w lasach, w strumieniach i opuszczonych fabrykach), nazywał się Darren Ralph – lub Ralphy. Kilka lat przed pomysłem stworzenia *Wspomnień uzależnionego od Spectrum*, Ralphy napisał pamiętnik z dzieciństwa, który czytałem z zapartym tchem, gdyż powodował u mnie przyływ dawnych wspomnień i sprawił, że dopadła mnie nostalgia za dawnymi czasami. Kiedy dorastasz i w twoim życiu pojawiają się dzieci, a do spłacenia są kolejne kredyty, stopniowo zapominasz o dzieciństwie, rozwodząc się nad codziennością, idąc w kierunku, który nieustannie przybliża cię do śmierci.

```

MOUNT CALIBARS GRANDDAUGHTER AND
GET 10 000 GOLD PIECES
THE BACKGROUND
ONE DAY YOU ARE WALKING THROUGH
YOUR HOME TOWN OF GOLDVILLE, WHEN
YOU SEE YOUR OLD FRIEND MOUNT C
ALIBAR. HE IS HURRYING ALONG, AND
YOU NOTICE THAT HE LOOKS WORRIE
D AND IS UNTIDILY DRESSED.
YOU HAIL HIM, AND HE COMES OVER.
YOU ASK HIM WHAT IS WRONG, AND HE
SAYS THAT HIS LITTLE GRANDDAUGH
TER HAS HAD A CURSE PUT ON HER,
AND THE ONLY WAY TO BREAK IT IS
TO KILL THE WITCH
YOU DECIDE TO HELP HIM AND YOU S
ET OFF ON AN ADVENTURE
YOU SET OFF NEXT MORNING BY BOAT
, AND REACH VOODOO ISLAND IN A CO
UPLE OF HOURS
AS YOU DOCK YOUR BOAT YOU FEEL S
UDDENLY EEEEEERIE

```

THE VOODOO PLOT

Jako, że byłem również pisarzem, po przeczytaniu słów Ralphy'ego, moje alter ego natychmiast pomyślało: „też chcę to zrobić”. No i zrobiłem. Pierwszą rzeczą, którą spisałem, było wspomnienie, które powróciło do mnie: „mam 12 lat, programuję swoją pierwszą grę w BASICu – *The Voodoo Plot* – trwa to cały dzień”. Produkcja miała blokową grafikę, wiele sposobów przejścia rozgrywki i plumkającą na beeperze muzykę. Zapisałem ją na kasecie magneto-fonowej WH Smiths C15 – nadal ją mam (nie, nie będę oferował wersji pod emulator, ha ha! – naprawdę jest taka zła).



ANDY REMIC JEST AUTOREM DWUDZIESTU TRZECH POWIEŚCI. MA DWÓJKĘ WSPANIAŁYCH DZIECI, PSA BORDER COLLIE O IMIENIU TILLY, MIESZKA W LINCOLNSHIRE W WIELKIEJ BRYTANII I LUBI: JAZDĘ ROWEREM GÓRSKIM, WSPINACZKĘ GÓRSKĄ, KICK BOXING, KRÓLIKI I WALKI NA PIŁY ŁAŃCUCHOWE. ZOSTAŁ NAZWANY "TARANTINO FANTASY", A JEGO DOKONANIA LITERACKIE PORÓWNUJĄ Z DAVIDEM GEMMELEM, MICHAIELEM MOORCOCKIEM I GEORGE M MARTINEM.

REMIC JEST TAKŻE NIEZALEŻNYM FILMOWCEM, A OPRÓCZ FILMÓW KRÓTKOMETRAŻOWYCH NAKRECIŁ TRZY PEŁNOMETRAŻÓWKI: IMPURITY (BARDZO NISKOBUDŻETOWY HORROR), MEMOIRS OF A SPECTRUM ADDICT I SPECTRUM ADDICT: LOAD FILM2, KTÓRE SĄ DOGŁĘBNYMI ROZPRAWAMI FILMOWYMI O STARYCH KOMPUTERACH 8-BITOWYCH PRODUKOWANYCH PRZEZ SINCLAIR RESEARCH.

AKTUALNE POWIEŚCI TO: SPIRAL, QUAKE, WARHEAD, WAR MACHINE, BIOHELL, HARDCORE, SIM, SERIAL KILLERS INCORPORATED, KELL'S LEGEND, SOUL STEALERS, VAMPIRE WARLORDS, CLONOWORLD, THEME PLANET, TOXICITY ORAZ ANTOLOGIA THE CLOCKWORK VAMPIRE CHRONICLES. JEGO NAJNOWSZE DZIEŁA TO: THE IRON WOLVES, THE WHITE TOWERS, THE DRAGON ENGINE AND TWILIGHT OF THE DRAGONS, WYDANE PRZEZ ANGRY ROBOT BOOKS ORAZ TRZY KSIĄŻKI WYDANE PRZEZ TOR: A SONG FOR NO MAN'S LAND, RETURN OF SOULS ORAZ THE IRON BEAST.

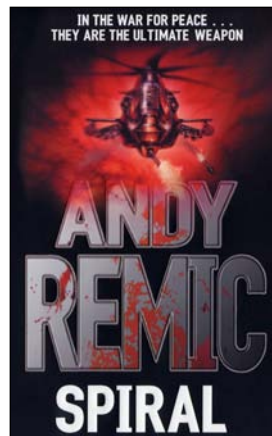
WIĘCEJ INFORMACJI ZNAJDZIESZ NA:

[HTTPS://ANDYREMIC.WORDPRESS.COM/](https://andyremic.wordpress.com/)

[HTTP://WWW.REMICMEDIA.COM/SPECCYADDICT/](http://www.remicmedia.com/speccyaddict/)

Potem do moich wspomnień dopisałem kilka długich scen, obejmujących wszystkie rzeczy związane ze Spectrum, od przekonania mojego taty, że będę wykorzystywał Spectrum przy odrabianiu prac domowych, do tego pierwszego niesamowitego Bożego Narodzenia spędzonego przy *Sabre Wulf* i *Knight Lore* oraz programowania moich własnych gier.

Potem zapisałem się na doktorat (głównie koncentrując się na angielskim, historii i opisywaniu życia dla potrzeb mojej pracy doktoranckiej), a to z kolei sprawiło, że spisałem jeszcze więcej wspomnień do mojego pamiętnika, a te związane ze Spectrum wylały się ze mnie, jak płynny kryształ z dzbanka połączonego prosto z moją przeszłością. Dla tych, którzy nie wiedzą, jestem także powieściopisarzem, moją pierwszą powieść *SPIRAL* opublikowało w 2003 roku wydawnictwo *Orbit Books* – pracowaliśmy nad nią od 2001 roku, kiedy to podpisałem z nimi umowę na trzy książki.



Dość często zdarza się, że gdy piszę powieść i zbliżam się do jej ukończenia, mój umysł bombardują pomysły na kolejną książkę. To dobra rzecz i sprawia, że książki

powstają płynnie, prawdopodobnie dlatego mam na koncie dwadzieścia trzy opublikowane powieści, przetłumaczone na sześć języków.

Jednakże w przypadku filmów było nieco inaczej. Nakręciłem kilka krótkich horrorów i filmów promujących moje książki, a ich tworzenie było świetną zabawą. Kiedy jednak zająłem się kręceniem pełnometrażowego horroru – *IMPURITY* – okazało się, że to bardzo długi, żmudny proces, przypominający wspinanie się po stromym zboczu, zamiast spaceru po łagodnym wzgórzu. Na głowie miałem nie tylko napisanie scenariusza, musiałem rekrutować i współpracować z aktorami, organizować sesje filmowe, porządkować plany filmowe, gromadzić rekwizyty, a każdy dzień zdjęć był zawsze bardzo napięty i stresujący, bo pojawiały się wyzwania, którym trzeba było sprostać. Podczas kręcenia *IMPURITY* dowiedziałem się bardzo dużo o kamerach i obiektywach, głębi ostrości, ujęciach, śledzeniu obiektów itp., dzięki czemu zdobyłem ogromne doświadczenie, które pozwoliło mi przejść na wyższy poziom wtajemniczenia i w ten sposób zacząłem edytować. Montaż dał o wiele więcej frajdy niż stresujące sesje filmowe.

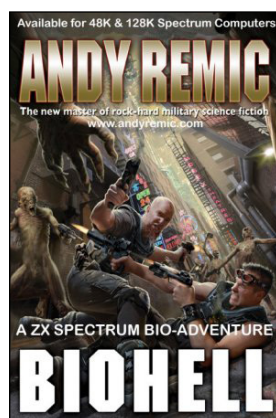
Kiedy ukończyłem prace nad horrorem, zaklinałem się, że nigdy więcej nie zrobię filmu z „aktorami” (Miałem w pamięci co powiedział Alfred Hitchcock: „Nigdy nie powiedziałem, że wszyscy aktorzy to bydlę. Powiedziałem, że aktorzy powinni być traktowani jak bydlę.”). Cóż, moje „bydlę” nigdy nie chciało robić tego, co mu mówiłem, nie znało swoich kwestii, a czasami na planie, wprost odmawiało grania tego, co było w scenariuszu, który wcześniej czytało i na co się akceptując go zgodziło. To była bardzo ciężka praca dla niezależnego filmowca – amatora. Po tych wszystkich przejściach, w moim następnym filmie chciałem czegoś mniej stresującego i tak zaczął kielkować pomysł na film dokumentalny, przy którym NIE będę musiał organizować planu i rekwizytów, a „aktorzy” będą naturszczykami, którzy nie będą musieli uczyć się kwestii i będą spontanicznie reagować na zadawane pytania. Doskonale! Czułem, że problemy znikają za horyzontem jak stado dzikich koni.

Ale jaki temat mógłbym poruszyć?

Gdy ukazała się moja piąta powieść, *BIOHELL*, uruchomiłem moje stare ZX Spectrum i napisałem grę przygodową wykorzystując *GAC (Graphic Adventure Creator)*, w celu promowania książki. To było bardzo przyjemne doświadczenie, przypomnienie sobie pseudojęzyka GAC i praca z prawdziwą grafiką ZX Spectrum.

W tym czasie zaangażowałem się również w *World of Spectrum (WoS)* i tam spotkałem Joffę – Jonathana Smitha, genialnego programistę takich gier jak *Mikie*, *Cobra*, *Batman* i *Firefly*. Joffa wywarł na mnie olbrzymie wrażenie, był tak przeziśłym facetem, że zaczęliśmy korespondować, wymienialiśmy historie i zgodziliśmy się wspólnie nakręcić kolejny horror. Joffa w pewnym momencie zamilkł, a ja zająłem się pisaniem książek i pracowałem nad scenariuszem do *IMPURITY*. Wtedy jeden z forumowiczów z *WoS* skontaktował się ze mną, by przekazać smutną wiadomość o śmierci Joffy. Było to w 2010 roku, zapamiętałem to jako przykry czas w moim życiu.

Po nakręceniu pierwszego horroru naszała mnie inspiracja – Joffa był jednym z bohaterów mojego dzieciństwa, więc OCZYWISTYM tematem filmu dokumentalnego, który chcia-



łem nakręcić miał być ZX Spectrum – komputer, który zawładnął mną w dzieciństwie i wczesnej młodości. Wtedy zdałem sobie też sprawę, że będę mógł SPOTYKAĆ bohaterów z mojego dzieciństwa! Poza Adamem Antem, wszyscy bohaterowie mojego dzieciństwa byli programistami ZX Spectrum, artystami i muzykami, przedstawianymi na fotografiach i w wywiadach w *CRASH* (biblii Spectrum), a także *Your Sinclair* i *Sinclair User* (kupiłem wszystkie). Cudownie!

Miałem przed sobą naprawdę dużo pracy m.in. kilka kontraktów na napisanie książek, a do tego uczyłem angielskiego w niepełnym wymiarze godzin, ale zacząłem równocześnie nawiązywać więcej kontaktów w świecie Spectrum, przygotowywać się do przeprowadzania wywiadów i jakoś, nie pamiętam jak, zdałem sobie sprawę, że istnieje coś takiego jak „Crowdfunding”, dzięki któremu można sfinansować (lub częściowo sfinansować) projekt taki, jak ten film.

Usłyszałem o *Revival* (targach gier retro) wyszukując informacji w Google. Tak się złożyło, że byłem też w kontakcie z Markiem R. Jonesem (grafikiem firmy *Ocean*, który pracował przy wielu grach na Spectrum), postanowiłem to wykorzystać i umówiliśmy się na spotkanie na wcześniej wspomnianej imprezie (sierpień 2014). Za główny cel postawiłem sobie przeprowadzenie wywiadu z Markiem, który przy tej okazji zasugerował, aby przeprowadzić wspólny wywiad z nim i z innym spectrumowym grafikiem, jego przyjacielem Simonem Butlerem. Ucieszyłem się, ponieważ jestem wielkim fanem pracy ich obu. Następnie skontaktowałem się z Christem Wilkinsem (*Retro Fusion Books*), który był wtedy współorganizatorem (chyba) i także umówiliśmy się na szybką rozmowę.

Jakby tego było mało, tuż przed samym wyjazdem dowiedziałem się, że będzie tam legenda *ID Software* – John Romero. Pomyślałem, że muszę spróbować przeprowadzić wywiad z Johnem Romero! Wiedziałem, że to nierealne, ale musiałem spróbować.



Tak więc, uzbrojony w kamerę SONY, którą filmowałem *IMPURITY*, ciągnąc ze sobą Roya Younga (mojego przyjaciela i zastępcę reżysera) i Garego Maina (fotografa), pojechalśmy do Wolverhampton w Wielkiej Brytanii.

Byłem bardzo podekscytowany wywiadem z Markiem i Simonem, ale jednocześnie nie dawałem sobie żadnych szans na przeprowadzenie wywiadu z Johnem Romero, więc póki co nie zwracałem sobie nim już głowy, do czasu. Po przybyciu na miejsce, udaliśmy się prosto do baru, by przy pomocy płynów dodać sobie odwagi.

Po obejrzeniu sali, na której zobaczyłem Clive'a Townsenda (znanego ze stworzenia gry *Saboteur*), razem z kolegami podeszliśmy do sceny, na której wywiadu udzielał John. Kiedy przechodził obok mnie ze swoją żoną Brendą Romero (legendą samą w sobie), przeszedł mnie dreszcz *DOOM*, *QUAKE*, *ŁAŁ! ODLÓT!*

Mistrz był na scenie, a ja obmyśliłem sprytny plan. Jeśli przekonam jego żonę, żeby udzielił mi wywiadu, to mam gwarancję, że uda mi się to zrobić. Miałem ze sobą asa w rękawie w postaci mojej drugiej książki, która nosiła tytuł *QUAKE*. Nie miała ona nic wspólnego z grą, ale jej tytuł był formą mojego skromnego hołdu dla dokonań *ID Software*. Zebrałem się w sobie, podeszedłem do Brendy i zaczęliśmy rozmowę. Była urocza. Pokazałem jej książkę, wyjaśniłem, że jestem fanboym i poprosiłem o pomoc. Kiedy John zszedł ze sceny – mimo że czekała na niego kolejka ludzi (i ku mojemu zażenowaniu) Brenda zaciągnęła mnie na przód kolejki i popędziła Johna Romero, mnie, Roya i Garego do sali konferencyjnej.

Łał! Niesamowity, krępujący, wspaniały i nieoczekiwany zwrot wydarzeń!

Przeprowadziłem wywiad z Johnem Romero używając aparatu, wręczyłem mu kopię książki *QUAKE*, a następnie zadałem mu kilka pytań na temat Spectrum. Po krótkiej rozmowie pożegnałem go, a on musiał się zastanawiać, kim do cholery jesteśmy i dlaczego pytaliśmy go o Spectrum, ponieważ

był Amerykaninem i prawdopodobnie wychował się na C64. W każdym razie, to był zaszczyt i przywilej spotkać się i przeprowadzić (krótki) wywiad z legendą gier, nawet jeśli nigdy tak naprawdę nie używał Spectrum (powiedział, że miał z nim krótko styczność) i nie mógł wymienić żadnej gry.

Następnie zrobiliśmy kilka zdjęć i nakręciliśmy kilka ujęć B-roll, po czym przenieśliśmy się na umówione spotkanie z Markiem R. Jonesem i Simonem Butlerem. Teraz znowu stałem się nerwowy, obaj byli weteranami przemysłu, a Simon nawet pracował dla legendarnego Imagine (a w naszym wywiadzie do drugiego filmu, rzeczywiście mówił o legendarnych megagrach *Psychopase* i *Bandersnatch*).

Znaleźliśmy ustronne miejsce w pobliżu innego baru, dokonaliśmy uprzejmych prezentacji, a gdy Roy ustawił kamerę (jest perfekcjonistą, jeśli chodzi o poprawne wykonywanie takich rzeczy) Mark i Simon wyładowali stos gier na Speccy, a ja przekazałem im zarys pytań do przeczytania przed rozmową. Muszę tutaj zaznaczyć, że byłem nowy w świecie wywiadów i opracowałem „ogólny” zestaw pytań niezwiązanych z żadną indywidualną osobą – po prostu zaznaczyłem w nawiasie sekcję: „Dostosuj do określonego programisty/artysty/muzyka”, nie zdając sobie sprawy z tego, że mówi to wiele o braku przygotowania (co nie było prawdą, ponieważ przeprowadziłem swoje badania, po prostu nie przełożyłem wysnutych na ich podstawie wniosków na zindywidualizowane pytania – co było bardzo głupim posunięciem).

Nigdy nie zapomnę, jak Simon Butler przeglądając pytania, marszczył brwi (Simon nie jest człowiekiem, który chowa się za tarczą przeciw głupcom), patrząc na mnie tymi przeszywającymi niebieskimi oczami, po czym mówi: „Dostosuj do określonego programisty/artysty/muzyka”. Mimo że nie było to pytanie, czułem się zmuszony odpowiedzieć i odpowiedziałem: „Eeee...”. Na szczęście Mark R. Jones wtrącił się, prawdopodobnie widząc mój dyskomfort, był przecież w znacznie bardziej przyjaznych stosunkach z Simonem Butlerem niż ja,

ten początkujący dziennikarz i dano mi spokój, gdy coś bełkotałem na temat braku czasu na spersonalizowanie pytań.

„Nota dla siebie – zawsze się przygotowuję, nigdy nie przybywam nieprzygotowany, okazuję każdemu rozmówcy szacunek, na jaki zasługuje, gdy poświęca ci swój cenny czas, aby pojawić się w twoim filmie dokumentalnym!” To się nigdy więcej nie powtórzyło.

W każdym razie krótko porozmawialiśmy o pytaniach, rozwiązaliśmy problem z kablem mikrofonu i natychmiast rozpoczęliśmy właściwy wywiad. Jako pierwszy na pytania odpowiadał Mark R. Jones, który największy entuzjazm wykazywał przy pytaniach o Spectrum, jego historii i sprzęcie oraz oczywiście o grach i grafice, przywołując przy tym pojedyncze tytuły ulubionych gier, nad którymi pracował. Następnie inicjatywę przejął drugi z twórców – Simon, który jak sam się przyznał (w późniejszych wywiadach) jest osobą, dominującą, lubiącą mieć wszystko pod kontrolą. Na szczęście był tak samo entuzjastycznie nastawiony do idei wywiadu jak Mark, dzięki czemu ja nagrałem momenty przekomarzania się pomiędzy tymi dwiema legendami gier, które momentami przemieniało się z konwersacji w monolog Simona.

Zarówno Mark, jak i Simon byli przyjaźnie nastawieni do wywiadu i na pewno cieszyli się ze spotkania. (W przeszłości pracowali nad takimi tytułami jak: *Cosmic Wartoad*, *Gift from the Gods*, *Gryzor*, *Wizball*, *Tai Pan* spośród stu innych i to był prawdziwy zaszczyt móc ich spotkać. Nie zdawałem sobie sprawy, że będę robić o wiele bardziej szczegółowe wywiady z obydwojema artystami jeszcze raz!)



Wyczerpani, opuściliśmy Wolverhampton i skierowaliśmy się tam, gdzie mogłem przejrzeć moje nagranie (wtedy jeszcze na taśmach HD), zrobić kopie zapasowe i wykorzystać zebrany materiał jako podstawę do uruchomienia akcji na Kickstarterze na nakręcenie filmu *Memoirs of a Spectrum Addict*.

Pierwsza akcja na Kickstarterze, którą prowadziłem w 2014 roku, zakończyła się niepowodzeniem, ale w trakcie drugiej z 2015 roku udało się zebrać około 6,5 tysiąca funtów (zbiórkę wsparł również niezrównany Chris Wilkins z *Fusion Retro Books*). Te pieniądze, po wypłaceniu natychmiast zostały wydane na kamerę Black Magic Ursa 4K z chłodzeniem wod-



nym, filmującą w standardzie Apple ProRes 4:2:2. Pomyślałem, że jeśli mam to zrobić, to zrobię to dobrze i przyszłościowo poprzez filmowanie w 4K. Co za idiotyczny pomysł! Musiałem także kupić obiektyw, nowy naprawdę mocny statyw mogący unieść tę bestię (1000 funtów), nowe baterie (kolejne 1000 funtów), profesjonalne mikrofony i kable, karty pamięci. Prawdopodobnie wydałem około 12 tysięcy funtów na sprzęt – zanim jeszcze zacząłem kręcić. Ups.

Wtedy nie chodziło o pieniądze, to była praca z miłości i wiedziałem, że dzięki kampanii na Kickstarterze, a co za tym idzie finansowaniu z góry, film będzie miał przynajmniej widownię – nie zrobię czegoś bezsensownego, czego nikt nigdy nie obejrzy. Inaczej jaki to ma sens? To wyniszcza. Przynajmniej w przypadku Spectrum istniała społeczność i byłem bardzo podekscytowany tym faktem, szczególnie gdy dowiedziałem się, że tak naprawdę nie ograniczała się tylko do Wielkiej Brytanii – jak pierwotnie sobie wyobrażałem – ale ma zasięg globalny.

Film *Memoirs of a Spectrum Addict* pojawił się na całym świecie – i w każdym miejscu to samo w sobie stanowiło ekscytujące doświadczenie.

Miałem już sprzęt, zacząłem ciężko pracować, kontaktując się z programistami, grafikami i muzykami. Wielu odmówiło, a niektórzy okazali się nieuchwytni (Tim Stamper ?!), co było przykre, ponieważ było wiele osób, których udział w filmie wydawał mi się nieodzowny. Miałem przyjemność korespondować z Costa Panayi, twórcą *TLL*, *Cyclone* i *Highway Encounter* i przeprowadziliśmy wiele pasjonujących rozmów, niestety nie udało mi się przekonać go do udzielenia wywiadu. Nie zdawałem sobie sprawy, jak onieśmielające może być dla moich rozmówców „usadowienie kogoś przed kamerą i mikrofonem”. Powoli z wywiadu na wywiad zacząłem się uczyć i zmieniać sposób, w jaki podchodziłem do ludzi, podkreślając, że nie będę inwazyjny, będzie zabawnie i nie będzie żadnych nacisków z mojej strony. Kto chciałby być wystawiony na presję włączonej kamery, która może wychwycić dosłownie wszystko? Dlatego z etycznego punktu widzenia, zawsze jasno określałem, że jestem tam po to, by pokazać osoby, z którymi przeprowadzam wywiad z jak najlepszej strony i co mam nadzieję mi się udało. A czego na pewno próbowałem.

Ostatecznie kręcąc te dwa filmy przeprowadziłem wywiady z tak wieloma osobami, że mógłbym napisać książkę o przygodach, które mnie spotkały. Mark R. Jones był pierwszym rozmówcą, z którym przeprowadziłem wywiad nową kamerą 4K i to sprawiło, że ja (i myślę, że Mark również) poczułem się znacznie swobodniej, ponieważ mieliśmy za sobą już jeden nagrany wywiad, więc ten stał się czymś w rodzaju próby. Mark był miły, a jego entuzjazm i zaangażowanie w Spectrum biło od niego podczas całego wywiadu, przeprowadzonego w jego domu w Salford.

Następnie przyszedł czas na rozmowę z Clivem Townsendem. Był to dla mnie bardzo interesujący wywiad (podobnie jak wszystkie inne, które były dla mnie unikalne niczym czerwone diamenty), który zarazem był dla mnie kolejną lekcją na temat ich przeprowadzania. Po raz pierwszy zobaczyłem Clive'a na



imprezie Revival, lecz byłem zbyt zdenerwowany, aby do niego podejść. Przypuszczam, że również John Romero, Mark R. Jones i Simon Butler zaprzętały mój umysł do tego stopnia, że mogłem przyjąć tylko tyle stresu jako fanboy tego dnia, ale pamiętam, że mijałem Clive'a, siedzącego przy stole z uruchomioną grą Saboteur, całkowicie samego, rozpaczliwie chciałem podejść i powiedzieć: „Cześć kolego, jak leci? Jestem fanem twoich gier odkąd skończyłem dwanaście lat”. Byłem zbyt nieśmiały i tego nie zrobiłem. Teraz Clive mieszka niedaleko Bath, powiedział mi, że nie prowadzi samochodów, więc planem, który wymyśliłem (okazał się zresztą dobrym planem i wykorzystałem go jeszcze wiele razy) było zarezerwowanie pobliskiego hotelu i obejrzenie jego zdjęć w Internecie, aby upewnić się, że mają dobry bar, a do tego jeszcze salę konferencyjną, z której moglibyśmy oficjalnie, bądź nie, skorzystać w celu przeprowadzenia wywiadu. Zauważyłem, że niektóre hotele są naprawdę przyjaźnie nastawione i starają się pomóc, a niektóre po prostu odmawiają w ciemno. Był jeszcze inny trik, który wykorzystałem, po prostu pytałem obsługę w recepcji zdaniem w stylu: „Czy mogę się trochę pokręcić po hotelu?” Zwykle odpowiadali „tak” i tym samym udzielali zgody.

Czasem robiliśmy to po partyzancku, co zawsze było ekscytujące np. kiedy pracowaliśmy z Jasem Austinem i Mevem Dincem dopóki nie wyrzucano nas z tego hotelu (na szczęście już po zakończeniu wywiadu – winię za to Meva), mimo że miałem pozwolenie na filmowanie w barze.

Pojechałem do domu Clive'a i zabrałem go do hotelu. Po drodze rozmawialiśmy i miałem wrażenie, że obaj byliśmy bardzo zestresowani. Trochę mnie to zaskoczyło, bo dlaczego miałby się mną stresować? Wróciliśmy do hotelowego baru i rozmawialiśmy jak ustawić sprzęt filmujący. Wciąż poznawałem ten zestaw, więc martwiłem, czy wybrane ustawienie pozwoli uzyskać tę piękną głębię pola, którą widzisz w filmach, a Clive pozował i trzymał różne rzeczy, więc mogłem to doskonale wykorzystać. Przekazałem mu kopię pytań i zacząłem się martwić, czy wystarczy nam czasu. Czas był luksusem w trakcie nagrywania, i nie chodziło bynajmniej o pozwolenie na nagrywanie, które miałem. Ostrzegano mnie, że bar może zacząć wypełniać się w porze lunchu – coś, czego się wte-

dy nauczyłem – tłum może spowodować spustoszenie w tym cholernym dźwięku.

W końcu postanowiliśmy przeprowadzić wywiad i zadałem pierwsze pytanie: „Kiedy zobaczyłeś pierwszego Spectrum?” Odpowiedź Clive'a była bardzo, bardzo krótka, niemal przycięta. Zadałem drugie pytanie: „Jaka jest pierwsza gra, którą naprawdę wyróżniał się Spectrum?” Odpowiedź kolejny raz była zaskakująco krótka: „Pamiętam, że grałem w *Knight Lore* w biurach firmy *Durell* i uważałem, że to była niesamowita gra.” Pomyślałem, szlag, jeśli wszystkie jego wypowiedzi będą takie krótkie, ciężko będzie je umieścić w filmie!

W każdym razie, zacząłem mieć problem z kablem audio i przeszedłem przed kamerę, spoglądając na ogrom sprzętu, który miałem (Black Magic Ursa jest fizycznie wielka, a miałem jeszcze przewody mikrofonowe i słuchawki oraz wiele dodatków ułożone wokół kamery w pudełkach) uświadomiłem sobie, jak to wszystko może onieśmielać ludzi siedzących przed tym. Wywiad zamiast być zabawną i ekscytującą rzeczą stał się dość stresujący, zwłaszcza dla kogoś kto jest mocno zaangażowany w to, o czym opowiada. Clive naprawdę był zaangażowany w Spectrum, będącego trampoliną jego kariery programistycznej. I tak powróciłem do tego odwiecznego stabilizatora, tego tysiącletniego zwiotczającego alkoholu.

– Mam drobne problemy. Zrobimy sobie przerwę i wychylimy po pincie? – zasugerowałem. Oczywiście Clive'a rozbłyły jak gwiazdy neutronowe.

– Tak!

– Piwo?

– Dla mnie cydr. Jestem spragnionym ninja. (Tak to brzmiało lub coś w ten deseń, z pewnością te słowa stały się żartem między mną a Clivem od pierwszego wywiadu).

Poszedłem do recepcji i zamówiłem dwa kufle, a potem przez chwilę siedzieliśmy rozmawiając przy barze. Następnie zamówiłem dla Clive'a drugi kufel, co go wyraźnie rozluźniło i wróciliśmy do wywiadu. Teraz odpowiedzi Clive'a były znacznie bardziej wymowne i szczegółowe, a my zakończyliśmy pierwszą rundę pytań w dobrej kondycji i mieliśmy drugą przerwę, więc mogłem przetransportować kamerę i sprzęt dźwiękowy na zewnątrz do ładnego małego letniego domu, który wyszukałem w ogrodach hotelu.

Clive uprzejmie pomógł mi przenieść cały zestaw, on miał przerwę na dymka, podczas gdy ja znów się objąłem (stałem się o wiele szybszy dzięki kolejnym wywiadam i nauczyłem się uszczuplać ilość sprzętu, który nosiłem – lub przynajmniej zostawiać większość w bagażniku mojego samochodu, ograniczałem się do tego co było absolutnie konieczne).

Nakręciłem kilka udanych ujęć, Clive spacerujący po terenie hotelu, Clive odchodzący ode mnie, Clive idący w moją stronę, Clive przechodzący obok kamery. W rzeczywistości nie było to w celu ustawienia ujęć, to było dla mnie osobistą rozrywką. Potem Clive wyszedł, by zapalić papierosa, a w hotelowych ogrodach znajdował się żywopłot dość niski i zacząłem kręcić długie ujęcie Clive'a, a kiedy przesuwaliśmy kamerę, zrobił przewrót ninja przeskakując nad żywopłotem i gdy znów się pojawił, jego papieros był nienaruszony. a on kontynuował spacer... „Szlag! Szkoda, że nie miałem go akurat w kadrze!” Częściowo mi się udało, a wycinek tej sceny umieściłem w napisach końcowych filmu *Spectrum Addict: Load FILM2*, po czym Clive przesłał mi wiadomość: „Nie zdawałem sobie sprawy, że to sfilmowałeś!” Piękno kamery. Nagrywa (prawie) wszystko.

Kiedy znów zaczęliśmy kręcić, Spragniony Ninja pił trzeci kufel cydru i naprawdę wpadł w wir wywiadu, opowiadając se-



CLIVE TOWNSEND, ANDY REDMIC

kwencję wydarzeń, które doprowadziły do rozpoczęcia pracy w *Durell Software*, technicznych aspektach programowania gry *Saboteur*, plus kilka innych rzeczy, które nigdy nie trafiły do filmów.

Zakończyliśmy nagrywanie. Normalnie to by było wszystko i rozstalibyśmy się, ale musiałem odwiedzić Clive'a do domu. Zaprosił mnie, abym obejrzał wioskę Wells, gdzie Edgar Wright, Simon Pegg i Nick Frost nakręcili film *Hot Fuzz*. Zrobiliśmy kilka zdjęć zrobionych przed słynną fontanną Sandford oraz na High Street, a następnie przenieśliśmy się do miejscowej gospody, by coś zjeść, uzupełnić braki płynów i odbyć długą, radosną pogawędkę o dniach chwały ZX Spectrum.

Na koniec chciałbym bardzo podziękować wszystkim, którzy poświęcili swój czas, energię i cierpliwość na wywiady do filmów: *Memoirs of Spectrum Addict* i *Spectrum Addict: Load FILM2*. Każdy wywiad był wyjątkowy, zabawny, ekscytujący i nie tylko spotkałem moich bohaterów z dzieciństwa, którzy dla mężczyzny (przepraszam, uzewnętrzniam się tutaj, że nie przeprowadzałem wywiadów z kobietami... może ulegnie to zmianie przy okazji kręcenia FILM3?) byli bardzo mili i łaskawi. Dziękuję! A może powinienem spisać wszystkie te wywiady, jak te wspomnienia i stworzyć z nich książkę? :-)

ANDY REMIC. STYCZEŃ 2019 R.



4 kwietnia 2020

Warszawa, Pub Potok, Potocka 14

`specy.pl/party`