In Nihilum Reverteris - Retro Game for ZX Spectrum 128K

Robert 'Hellboj' Straka[1,2], Michał 'Yerzmyey' Bukowski [1]

[1]UBU Lab, Faculty of Management and Social Communication
Jagiellonian University in Kraków

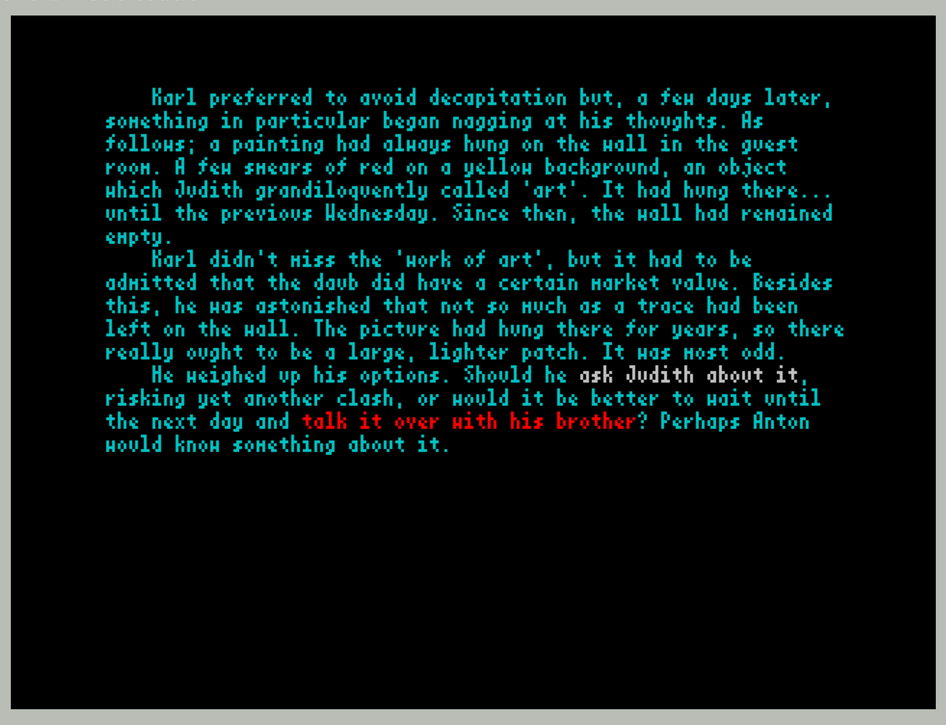
[2]Department of Heat Engineering and Environment Protection,Faculty of Metals Engineering and Industrial Computer Science
AGH University of Science and Technology

Introduction

Developed for an 8bit ZX Spectrum computer, In nihilum reverteris represents an old school sci-fi text adventure game comparable with former text games from the 80' era. The game itself behave as interactive book with different pathways, the player can take during the gameplay. We used the assembly language to code the game engine and to allow 64x24 character screens. The text is accompanied with graphical screens and music. Due to enormous size of the whole project and to minimize tape/disk operations, the game was developed for 128K versions of ZX Spectrum only and divided into two parts. There are two language versions available, Polish and English, each contains 22 locations, through which player advance by selecting action links in every location.

Hardware and software constraints of ZX Spectrum

Due to a limited memory in 48K model, we developed the game only for 128K model, where the memory banks are used to store graphical screens, texts and ingame music played by AY-3-8912 chip. The large size of graphics and texts results division of the game to two parts which have to be loaded from tape/disk. We also select to implement our own text printing routine which allows for 64 characters per line (4x8 pixels for one character), instead of the ROM based routine (32 characters per line). In order to minimize a code size, we used an assembly language and developed everything from scratch, apart from block loading routines from the ZX ROM. There's also the Easter egg hidden in the game as there was still some free RAM after finishing the game engine. Music is played in the background employing the interrupt mode 2 of the ZX Spectrum. No other hardware features were used, due to the fact that ZX Spectrum doesn't have any, also we did not use any high level language apart from the BASIC loader.



Coding Strategy

The game engine was implemented in Z80 assembly language and compiled using PASMO compiler on the PC. The main loop of the engine tests keyboard input and performs appropriate actions like printing the next page of the text, select next/previous active link and switch to the selected text location.

```
loop    cp 0x0f
        jp nc,load              ;check for loading of next block
        push af
        call ploc               ;printing location stored in register a
        ld b,25
w142    halt
        djnz w142               ;wait for a while
        call keyb               ;check keyboard
        pop bc
        inc a                   ;in case of selection of active link
        jr nz,nextloc           ;go to next location
        ld a,b                  ;or restore location number to register a
        jr loop
```

The ingame music and highlighting of the active and inactive action links are done within the interrupt routine, which is called 50 times per second. Music itself is stored in the bank nr. 6, and have to be paged to the RAM accessible by the CPU every time the interrupt routine is called and repaged in the end.

```
        di
        ...
        ld a,(0x5b5c)   ;page bank nr. 6
        push af         ;where the music is stored
        and 0xf8
        or 6
        ld bc,0x7ffd
        ld (0x5b5c),a
        out (c),a
        call msx_play   ;play the music
        pop af
        ld bc,0x7ffd
        ld (0x5b5c),a
        out (c),a       ;repage the previous bank
        ld a,(location)
        call alink      ;check for action links
        ...
        reti
```

Data storage optimization

The game is divided into two parts and each part is fitted to the available memory on the ZX Spectrum, i.e. again divided into 16 KB blocks, where engine resides in the fast memory area (bank 2 which is paged to 0x8000-0xc000). Texts, screen and music are stored in the rest of the banks and paged as needed from the address 0xc000. The layout of the data for both parts of the English version is detailed below, symbols S & T stands for screen data and text data respectively followed by a number of datablocks (for screen data this is usually 2 as one screen takes up to 13824 bytes and the size bank is 16 KB i.e. 16384 bytes), three locations are also stored in the bank nr. 2 right behind the code.

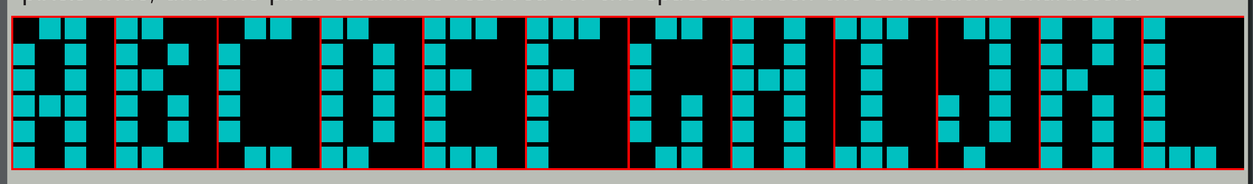
```
              Bank 0   Bank 1   Bank 3   Bank 4   Bank 6   Bank 7
     0xffff +--------+--------+--------+--------+--------+--------+
            |   T4   |   T5   |   S2   |   S2   |AY MUSIC|   T3   | PART I
            |        |        |        |        |        |        |
            +--------+--------+--------+--------+--------+--------+
            |   T3   |   T4   |   S2   |   S2   |AY MUSIC|   S1   | PART II
            |        |        |        |        |        |   T1   |
     0xc000 +--------+--------+--------+--------+--------+--------+
            | Bank 2 | Any of these banks can be paged to the memory
            |4x8 font| area between 0xc000 - 0xffff
            |   T3   | font and tables are stored together with
            | engine | game engine in the fast memory area
     0x8000 +--------+
            | Bank 5 | this area of memory is slow due to the privileged
            |        | access of the ULA chip which is responsible for
            |        | generation of the screen signal, memory banks
            | screen | where ULA slows down the reading/writing and code
     0x4000 +--------+ execution are called "contended" memory banks
            | ROM 0  |
     0x0000 +--------+
```

In total we have almost 140 KB of graphical (55296 bytes) and text (87915 bytes) of data.

Ingame 4x8 pixel font

In order to make texts more compact on screen, we used 4x8 pixel font which allows 64 character per line, full font is on the figure below, two characters are stored in 8 bytes and are rotated in case their position within the screen is in the opposite nibble. Few special characters (Polish, French and formatting symbols) were also added.

Magnification of the few characters to reveal their pixel composition. In fact, the character are only 3 pixels wide, and one pixel column is reserved for the space between the consecutive characters.



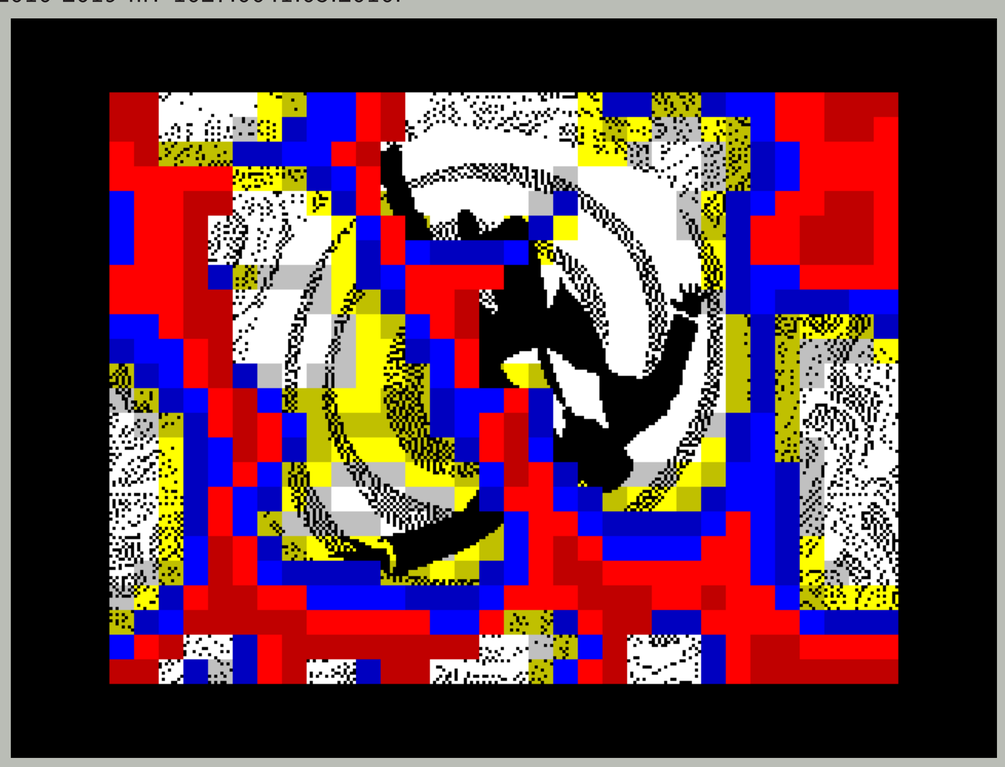
Outro

Small and classical demoscene effect called "plasma" was added to the last screen of the game to underline our relation to the demoscene and also serves as eyecandy for the player after the game completion. The listing of the effect is given below.

```
        di                      ld l,a
        ld (spr+1),sp           ld h,0xa0
        ld ix,0xa9e0            pop de
        ld sp,0xa200            ld a,(hl)
mke     ld a,(ix-0x20)          ld (de),a
        add a,(ix-1)            ld a,ixh
        add a,(ix+0)            cp 0xad
        add a,(ix+0x20)         jr c,mke
        srl a               spr ld sp,0
        dec a                   ei
        srl a                   ld hl,0xa9e0
        ld (ix+0),a             ld b,0x20
        inc ix              ns  ld a,r
        ld a,(ix+0x1f)          and 0x0f
        cp 0x0f                 ld (hl),a
        jr c,lin                inc l
        ld a,0x0f               djnz ns
lin     srl a                   ret
```

Acknowledgment

Project was realized within research project "Twórcze programowanie. Laboratorium" supported by the Ministry of Research and Higher Education program "Narodowy program Rozwoju Humanistyki" in years 2016-2019 nr. 1027.0641.65.2016.



https://www.facebook.com/ubulaboratorium/ strakrob@gmail.com yerzmyey@interia.pl