[RElog] – Oel (Pompowacze) port z C64 (część 2) (sheadovas/artykuly/relog/oel-port/oel-pompowacze-port-z-c64-czesc-2/)

Kwi 08, 2017 / Oel Port (sheadovas/category/artykuly/relog/oel-port/)

Analiza statyczna kodu

Hej, w dzisiejszym wpisie chciałbym podzielić się z Wami moimi przeżyciami związanymi z pisaniem portu do gry Oel (więcej info o samym projekcie [tutaj (sheadovas/artykuly/relog/relog-oel-pompowacze-port-z-c64-czesc-1/)]. Zgodnie z nowo przyjętą tradycją omawiane fragmenty kodu dotyczą zmian do commita [81fa7cc (https://github.com/sheadovas/oel-port/commit/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a)].

W poprzednim wpisie zająłem się ogólnymi przygotowaniami, od tamtego momentu odkryłem całkiem sporo. W tym wpisie chciałbym pokazać jakie kroki wykonałem aby móc względnie komfortowo poruszać się po kodzie tej gry.

Pierwszym krokiem jaki postanowiłem zrobić to oczywiście zrozumieć do czego służą poszczególne słowa kluczowe, zmienne, funkcje, itp. Jak łatwo się domyślić dokładne zrozumienie nie jest do końca potrzebne przy zwykłej statycznej analizie (no chyba że przepisuje się kod 1:1 ze wszystkimi bugami ;)), ale nawet do ogólnego zrozumienia trzeba było oryginalny kod nieco pozmieniać.

> Uwaga! W poniższych listingach kodu pokazuję jedynie wybrane **fragmenty**, linki do kompletnych plików w okolicach odpowiednich listingów ;) Oczywiście gorąco zachęcam do przejrzenia ich w całości, bo nie ukrywam że każda linia to sporo pracy ;)

Mając pod ręką [linki z dokumentacją (https://github.com/sheadovas/oel-port/tree/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a#useful-links)] zabrałem się do pracy (samą bazę linków rozbudowywałem w miarę potrzeby), na pierwszy ogień postanowiłem zabrać się za rozbicie spacjami [słów kluczowych (https://www.c64-wiki.com/wiki/Category:BASIC-Command)] (komend), tak aby vim dał radę je pokolorować.

```
# transformacja z postaci:
340 getsz:ifsz>1andsz<7then450

# do postaci:
340 get sz:if sz>1 andsz<7 then 450

# przykładowa komenda vim'a do zamiany
# "<word><command><word>" -> "<word> <command> <word>"
:%s/<command>/ <command> /g
```

Jak widać na powyższym przykładzie nie załatwia nam to np. spójników logicznych, ani operatorów. Zauważamy także prostą strukturę linii kodu:

```
340 get sz
 ^   ^   ^
 |   |   |-- params
 |   |------ command
 |---------- address (offset)
```

Oprócz tego każda linia może składać się z wielu instrukcji, instrukcje można ze sobą „skleić” używając operatora „:”

```C++
 addr   cmd1    :          cmd2
  340   get sz  :  if sz>1andsz<7 then 450
```

Powyższą linię można zapisać w nieco przyjaźniejszy sposób następująco:

```
340
    get sz:
    if sz>1 and sz<7 then
       450
```

Jak widać powyższy zapis jest znacznie bardziej czytelny i różne jego wersje są stosowane w [„przetłumaczonym” listingu (https://github.com/sheadovas/oel-port/blob/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a/notes/listing.bas)]. Samo sklejanie instrukcji jest bardzo ważnym elementem tego języka, a to chociażby z powodu ograniczeń instrukcji *if* która po spełnieniu warunku wykonuje tylko jedną linię po słowie kluczowym *then* (do tego tematu wrócimy jeszcze później)

Kolejnym krokiem użytecznym w analizie było „zapolowanie” na wszystkie funkcje, w tym celu użyłem komendy:

Shell
```
grep -oP listing.bas "gosub \d*" | sort -u
```

W wyniku otrzymałem listę odwołań do konkretnych adresów, które można traktować jako funkcje (jak się okazało pod koniec analizy lista nie była kompletna). Całość zrzuciłem do pliku [functions.txt (https://github.com/sheadovas/oel-port/blob/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a/notes/functions.txt)] i tam stworzyłem „mapę funkcji”, w której zacząłem zmieniać mało zrozumiałe nazwy na bardziej intuicyjne:

```
gosub 1460       -> init_values()
gosub 1900      -> prepare_consts_to_load()
gosub 3500      -> menu_next_player()
```

Gdy już stworzyłem wpis zmiany nazwy w tym pliku to wystarczyło jedynie zamienić wszystkie *gosub* na ich czytelne odpowiedniki w pliku źródłowym (np znaną już komendą vima *„:%s/gosub 1460/init_values()/g„*), nie ukrywam że stworzenie tego pliku znacznie poprawiło szybkość analizy kodu.

Podobnie zacząłem postępować też z innymi wyrażeniami, a także zmiennymi.

Przykładowa funkcja w czytelniejszej postaci
```
REM: func init_values()
1460    si=54272:
    fl=si:
    fh=si+1:
    tl=si+2

1480    th=si+3:
    w=si+4:
    a=si+5:
    h=si+6

1500    l=si+24:
    current_year=1983:
    return
```

Interesującym odkryciem było, to że gra operuje na pamięci rzeczywistej, co oznacza że rozmowa ze sprzętem też może odbywać się na tym samym poziomie. Ten fakt jest szczególnie przydatny przy instrukcji *poke*, która może zapisać wartość pod dany adres, np.

```
poke 53280, 1
```

Oznacza: „Zamień kolor ramki na biały”, aby do tego dość należało skorzystać z dwóch rzeczy: 1) mapy pamięci, 2) tabeli kolorów (linki do nich macie w moim repo). Ostatecznie ich czytelniejsza wersja wygląda następująco:

```
poke 53280       -> set_border_color(color)
poke 53281      -> set_inner_color(color)
poke 53272 <val>  -> set_mem_registers(val)
```

Kończąc temat funkcji, tak BASIC i Commodore udostępniają także komendy dostępne przy użyciu print, znalezienie wszystkich przysporzyło mi sporo problemów ostatecznie udało mi się je odnaleźć i przypisać im odpowiednie funkcje:

```
print "{home}"        -> mov_cursor_upp_left()
print "{<num> up}"    -> mov_cursor_up(lines_num=1)
print "{<num> down}"  -> mov_cursor_down(lines_num=1)
print "{<num> rght}"  -> mov_cursor_rght(offset=1)
print "{<num> left}"  -> mov_cursor_left(offset=1)


print "{rvon}<str>"   -> print_black_on_white(string)
print "{rvof}<str>"   -> print_white_on_black(string) (default)
print "{clr}        -> clear_screen(color)
```

Oprócz nich były także inne wariacje, których już tutaj nie zapisałem, ale ich mnemoniki są na tyle proste że nie tworzyłem im oddzielnych funkcji (np zmiana koloru tekstu: „{blk}Print black text”).

Jak wspomniałem już wcześniej, zacząłem także tworzyć mapę zmiennych, która jest dostępna w pliku [variables (https://github.com/sheadovas/oel-port/blob/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a/notes/variables.txt)], jeżeli rzucicie na nią okiem to nie jest ona zupełna, a wynika to z faktu że zmiennych jest masa oraz części funkcjonalności nie byłem w stanie odgadnąć (lub też jeszcze nie zdążyłem), tutaj nie obyło się bez solidnego wgryzienia w kod.

```
sz       -> int players_count         (at addr 450?)
ra$     -> str concern_name
           = "bohr & pump & sohn"
k       -> int start_money
l1      int ???
rz      -> int current_year (init = 1983)
s$      -> char menu_choice
```

Pomocne okazały się stringi, które należało przetłumaczyć z niemieckiego na np angielski, w tym celu grepnąłem kod źródłowy po zawartości cudzysłowów a następnie zacząłem umieszczać w plikach: [strings_translated (https://github.com/sheadovas/oel-port/blob/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a/notes/strings_translated.txt)] oraz [strings_notranslation (https://github.com/sheadovas/oel-port/blob/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a/notes/strings_notranslation.txt)].

strings_translated.txt (fragment)

```
"a = bohrgesellschaft"             -> "a = Drilling company"
"benzinacker "                  -> "Petrol station"
"bitte druecken sie eine taste"         -> "Please press a button"
" das grosse spiel ums grosse geld."        -> "The big game for big money"
" das spiel endet im jahre 2017"        -> "The game ends in 2017"
" die raffinerieoelpreisentwicklung "       -> "The refinery price development"
"einen moment bitte"                -> "one moment please"
"einkauf von "                  -> "Shopping of"
"entscheidungsmoeglichkeit fuer"        -> "Decision - making"
"% erwischt "                   -> "% get "
```

Dojście do obecnej wersji pliku [listing.bas (https://github.com/sheadovas/oel-port/blob/81fa7cc8d3be44e32677d76c72e997d49f9bbf5a/notes/listing.bas)] pochłonęło naprawdę sporo godzin (myślę, że około 15-20), duża część z tego czasu to research oraz analiza kodu, bardzo pomocne okazywały się być skrypty robiące za mnie sporo roboty.

Nie ukrywam, że sam projekt jest bardziej wymagający niż na początku sądziłem, przy porcie 1:1 wiele rzeczy wymaga dokładnego zrozumienia (nie pomaga język niemiecki w którym pisana jest gra), jednakże jest to bardzo relaksujące. Jeżeli ktoś chce mi pomóc w tym porcie to gorąco zachęcam do poszukania „TODO” w kodzie i ich wykonania ;)/

Ty tyle w tym wpisie, do przeczytania w kolejnym, w którym mam nadzieję przedstawię Wam prototyp portu gry ;)

Code ON!