Pac-Man for the DEC VT420

François Laagel *
Institute of Electrical and Electronics Engineers

Abstract

Pac-Man is a graphical game designed in 1979 by a team of five people and implemented in Z80 assembly language over the course of seventeen months. This article is an evolutionary account of my own Forth implementation in ANS94 Forth for the Digital VT420 text terminal over a three month period. The C port of the resulting application to Linux and OpenVMS 9.2 will also be briefly covered.

Stress will be laid upon the value of standards throughout this paper. Various development/prototyping tools were required for this implementation to be successful. In essence, this paper is a first person account of an experience in retrocomputing.

1 Background and Motivation

Z79Forth is a single board computer I started designing by the end of 2018. It is based on the Hitachi HD6309 microprocessor, a much improved implementation of the Motorola MC6809. It runs three to five times faster, uses less current and has an extended instruction set.

The firmware I developed for the board is a Forth operating system originally written for the Zilog Z80 back in 1983. Two Git branches are available, allowing an end user to select either a 79-STANDARD subset or an ANS94 Core implementation. The latter was used as a target for this artful endeavour (see also [1]). GNU Forth 0.7.3 under Linux was used as a development platform. To that end, a very high degree of compatibility was essential.

During firmware development, flow control over the serial asynchronous communication line via USB (FTDI-232RL based) was found to be problematic at times. UART management was initially strictly *programmed IO* based. Yet even after switching to an interrupt based scheme, characters were occasionally lost during "cut & paste" of large text chunks. So, I ended up going for an alternate connectivity option over RS232. In 2020, I acquired a DEC VT420 terminal so as to be able to communicate with the board without having to resort to a dedicated frontend system. Worth noticing is the fact that DEC terminals do not support hardware flow control (RTS/CTS based) but implement software flow control (XON/XOFF) instead.

The DEC VT420 [2] is a technological wonder of the nineties. It is a monochrome Intel 8031 based text terminal equipped with an 800 by 400 pixels display. Most importantly, it supports user defined fonts. Which is the reason why I came across the idea of implementing Pac-Man specifically for it.

2 Research

Pac-Man is the brainchild of Toru Iwatani. Iwatani [3] was self-taught in computers without any formal training in programming or graphic design. The game aimed to appeal primarily to women and was originally released in Japan in July 1980.

Pac-Man is driven by the player (via a joystick) and is free to move within the confines of a maze. Every reachable spot of the maze has to be visited at least once. Four ghosts, initially located inside of a pen at the center of the maze, compete against the player for his/her life. Throughout the maze *collectible* items are interspersed. These are either *dots* or *power pellets*, which carry a greater point value and grant Pac-Man some temporary immunity against the ghosts.

Every substantial project begins with reference material collection. Fortunately, Pac-Man has been reverse engineered down to the level of implementation bugs analysis. The ghosts moving strategy was an early concern of mine. The **Gameinternals** web site [4] supplies a thorough documentation with respect to this. The ultimate reference document remains Jamey Pittman's "**Pac-Man dossier**" [5].

In 1995, Roar Thornaes [6] released a C++ based Pac-Man implementation targeting X11/POSIX

*`f.laagel@ieee.org`

under Linux or CYGWIN under Microsoft Windows. I selected his maze's topology–which differs from that of the original game and decided that Thornaes' object oriented approach was the way to go. To that end, I elected to go for GNU Forth 0.7.3's API to object orientation–which was later on made an integral part of the Forth2012 standard [7]. I ended up adopting the specification literally, using a cross-platform development model from GNU Forth 0.7.3 and targeting, ultimately, **Z79Forth/A**. The convenience of adherence to standards for this purpose cannot be emphazised strongly enough.

3 Methodology

The development approach I followed was originally a *shot in the dark*. In retrospect, it still looks to me as a very rational way to conduct such an ambitious project.

Presentation Layer This entailed figuring out how to draw a workable playing screen on a text based terminal. An essential part of this was to come up with a nice looking user defined font, so as to be reasonably faithful to the original implementation.

Basic Business Layer This covers object identification and definition. It also includes the implementation of a very basic gameplay. Initially:

- A shared ghost moving/display policy–mostly based on random direction selection. That policy also addressed the delicate subject of preserving erasable characters items on screen.
- A specific Pac-Man moving/display policy, which had at some point to address gathering collectibles and its gobbling/non-gobbling state. It also handles score management, collision detection and remaining lives count maintenance.

Business Layer Successive Refinements At this point, differentiated moving strategies for each of the four ghost instances were implemented. This also implied *ghost mode* management. At any given time, the ghosts can be in any of the following three states: *scatter*, *chase* or *frightened*. A finite, partially time based, state automaton drives the changes between those states.

Testing This is the fun part, of course. Bug fixing also comes with the territory!

Specific Support for the VT340 The VT340 has a different matrix size for user defined font specification. The gameplay code is entirely shared with the VT420.

Publish the Forth Code as public domain software on Github.

Linux/C Port I am not aware of any working Forth implementation for OpenVMS. So I went for a straight port to C.

OpenVMS/C Adaptations OpenVMS is mostly POSIX compliant but there are some differences.

To do: Validate the C code on OpenVMS and publish it.

4 VT420 Font Development

At some point Pablo Hugo Reda sympathized with the utterly futile nature of my project and suggested *The spriters resource* [8] as a valuable starting point.

When I consulted him for guidance about sprite design, Pablo mentioned *Piskel* [9] as the tool he uses in the context of his side teaching activity. He also said he designed his own sprites manually. Obviously, I took his input and set out to develop my own custom font for the VT420. This turned out to be a tedious and very time consuming operation, which ended up taking me some twelve full-time working days.

The original Z80 based implementation resorted to a 224 by 288 graphical display [10]. Because I targetted a monochrome text terminal, I first had to let the colors go and, instead, focus on distinctive looking aspects for each of the four ghosts. Pac-Man himself is supplied with a visible eye which was not among the features of the original game.

Typical text display terminals of the 1990's offered an 80 x 25 display capability. The VT420 is a renowned member of that class of devices. Because user-defined fonts are specified, in sixel terms, as a bitmap matrix 10 pixels wide and 16 pixels high, I chose to operate in a similar mode, which amounts to 33 columns double width characters by 23 physical rows.

This resulted in a font of 30 double width characters. Figure 1 illustrates the bitmap design

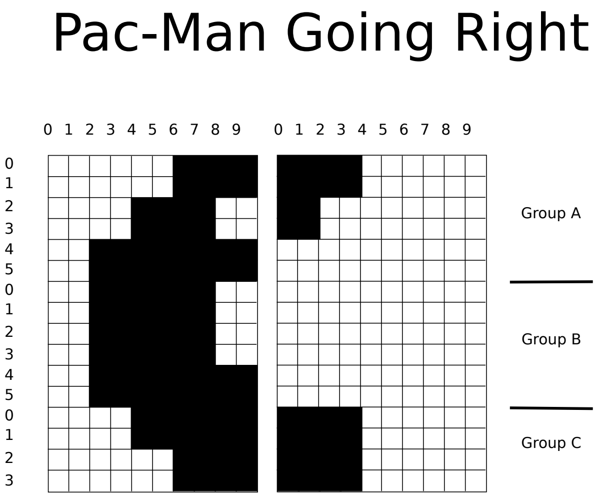

Figure 1: Pac-Man Going Right Sprite

for the use case *Pac-Man Going Right.* Figure 2 shows the corresponding Forth code, an outline of the VT420 font definition protocol (DECDLD) and what it amounts to in ASCII terms.

The concept of virtual rows originated from the notion that, somehow, the aspect ratio of the original game could be preserved, from a timing perspective. Conceptually, a virtual row is twice the physical line number that the terminal can directly address. The game's main engine works by assuming a predictable run time environment and updating the game's current state based on the notion that the game actually works in a grid space that is four times larger than it actually is (as visualized on the terminal): double-width characters combined with a divided by two scheduling on the verticals. Basically, the columns are handled physically and the rows virtually.

5 An Object Oriented Approach

Although my approach to object orientation is rather primitive–there is no need for inheritance support and an object is little more than a way to store state information in a centralized way–it has proven to be effective to solve the problem at hand. The central concept is that of an *entity.* An entity is an object subjected to various *methods.* There are five entity instances: Pac-Man is described in an entity vector which resides–by convention–at offset 0. The other entities are each of the four ghosts: **Blinky**, **Inky**, **Pinky** and **Clyde**. Pointers to these reside in the same vector. Each entity has an instance number attribute that is used to

```
CREATE softfont
[...]
\ Character "Pacman going right", left half at $25 (%).
\ Left: colunm #0 and #1 unused.
\ Group A  Group B    Group C (top to bottom)
%000000 C, %000000 C, %0000 C, \ Column #0 unused
%000000 C, %000000 C, %0000 C, \ Column #1 unused
%110000 C, %111111 C, %0000 C, \ Column #2 (eff. 0)
%110000 C, %111111 C, %0000 C, \ Column #3 (eff. 1)
%111100 C, %111111 C, %0011 C, \ Column #4 (eff. 2)
%111100 C, %111111 C, %0011 C, \ Column #5 (eff. 3)
%111111 C, %111111 C, %1111 C, \ Column #6 (eff. 4)
%111111 C, %111111 C, %1111 C, \ Column #7 (eff. 5)
%110011 C, %110000 C, %1111 C, \ Column #8 (eff. 6)
%110011 C, %110000 C, %1111 C, \ Column #8 (eff. 7)

\ Character "Pacman going right", right half at $26 (&)
\ Right: columns #8 and #9 unused.
\ Group A  Group B    Group C (top to bottom)
%001111 C, %000000 C, %1111 C, \ Column #0 (eff. 8)
%001111 C, %000000 C, %1111 C, \ Column #1 (eff. 9)
%000011 C, %000000 C, %1111 C, \ Column #2 (eff. A)
%000011 C, %000000 C, %1111 C, \ Column #3 (eff. B)
%000000 C, %000000 C, %0000 C, \ Column #4 (eff. C)
%000000 C, %000000 C, %0000 C, \ Column #4 (eff. D)
%000000 C, %000000 C, %0000 C, \ Column #4 (eff. E)
%000000 C, %000000 C, %0000 C, \ Column #4 (eff. F)
%000000 C, %000000 C, %0000 C, \ Column #8 unused
%000000 C, %000000 C, %0000 C, \ Column #9 unused

\ ----------------------------------------------------------
\ DECDLD spec:
\ DCS Pfn; Pcn; Pe; Pcmw; Pss; Pt; Pcmh; Pcss { Dscs
\ Sxbp1 ; Sxbp2 ;...; Sxbpn ST

1  CONSTANT pfn   \ Font number
1  CONSTANT pcn   \ First soft char at $21--do not override BL!
1  CONSTANT pe    \ Erase only new definitions
10 CONSTANT pcmw  \ Character width is 10 pixels in 80 col mode
0  CONSTANT pss   \ 80 columns, 24 lines
2  CONSTANT pt    \ Full cell
16 CONSTANT pcmh  \ Character height is 16: 6/6/4 in sixels
0  CONSTANT pcss  \ 94-charset, $20 (BL) and $7E (~) are RO
85 CONSTANT ufn   \ User font name is 'U' (argument to DSCS)

\ See: https://vt100.net/docs/vt420-uu/chapter9.html
\ for full parameter description.
??oo{{~~rr/??~~~~~~oo/????BBNNNN; \ Char $25 (%) definition
NNBB??????/??????????/NNNN??????  \ Char %26 (&) definition
```

Figure 2: Pac-Man Going Right Encoding

implement differentiated behaviour (figure 3).

The methods implemented are:

- `strategy`: the word stored in that field is the address of a Forth execution token that determines how the object moves from one clock cycle to the next one.
- `display`: a pointer to a Forth word that displays the object, taking into account its current state attributes. This is invoked on every clock cycle, after the `strategy` method has been called.

6 Scheduling

Any kind of game requires some form of real-time user interaction. This can be accomplished through either an event-triggered approach or a time-based one. Micheal J. Pont [11] strongly advocates the latter, which relies entirely on cooperative task switching and a timer based interrupt handler that is supposed to be able to deal with any unexpected condition, should the need for this ever arise. I

```
BEGIN-STRUCTURE entity
  FIELD:  e.strategy \ Strategy (moving) method
  FIELD:  e.display  \ Display method
  CFIELD: e.resurr   \ # Clock ticks till we're back (ghosts)
  CFIELD: e.reward   \ # points for killing a ghost / 100 (PM)
  CFIELD: e.vrow#    \ Virtual row number
  CFIELD: e.pcol#    \ Physical column number
  CFIELD: e.ivrow#   \ Interfering virtual row number (ghosts)
  CFIELD: e.ipcol#   \ Interfering pcol number (ghosts)
  CFIELD: e.igchr    \ Interfering grid character (ghosts)
  CFIELD: e.pcol0    \ Initial pcol number
  CFIELD: e.vrow0    \ Initial vrow number
  CFIELD: e.dir0     \ Initial direction
  CFIELD: e.hcvr#    \ Home corner vrow# (ghosts)
  CFIELD: e.hcpc#    \ Home corner pcol# (ghosts)
  CFIELD: e.cdir     \ Current direction
  CFIELD: e.pdir     \ Previous direction
  CFIELD: e.idir     \ Intended direction (PM)
  CFIELD: e.revflg   \ Reverse direction directive (ghosts)
  CFIELD: e.inited   \ TRUE if first display has been performed
  CFIELD: e.gobbling \ # Clock ticks till we're fed (PM)
  CFIELD: e.inum     \ Instance serial number
END-STRUCTURE

\ Method invokator.
: :: ( method-xt-addr -- ) @ EXECUTE ;
```

Figure 3: Entity Object Specification

```
: _main ( -- )
  BEGIN
    \ Check if remitems# is 0 here. If so start new level.
    remitems# @ 0= IF
      .initial-grid
      update-level
      level-entry-inits
    ELSE
      entvec @ DUP e.strategy :: \ Pacman's move
      entvec CELL+ #ghosts 0 ?DO
        DUP @ DUP e.strategy :: \ Move the current ghost
        CELL+
      LOOP DROP
      clkperiod MS
    THEN
  AGAIN ;

: main ( -- )
  initialize
  entvec @ TO pacman-addr
  0 remitems# !     \ Force level entry initializations
  PAGE .init-sitrep \ Initial scoreboard

  IFZ7 _main finalize
  IFGF ['] _main CATCH finalize
  IFGF ?DUP IF
  IFGF   0 23 AT-XY ." Caught exception " DUP . CR
  IFGF   ." Depth was " DEPTH . CR
  IFGF   THROW      \ We still want a stack dump!
  IFGF THEN
;
```

Figure 4: Early Engine Code

decided to go for a time-based approach with no interrupt handler support–a reliable time basis remains essential to the game's playability though.

An early incarnation of the game's main engine is shown in figure 4. At that point, involving the `strategy` method still implied an implicit reference to the `display` method.

Z79Forth/A is not a multi-tasking system. Which means the firmware does not have to compromise against any other running task. Properly programmed, it makes the platform ideally suited for a real time type of application. Precisely what was needed under the present circumstances. The Linux kernel in its 5.4.0-150 kernel incarnation (Mint 19.3) is not real time by default and this presents a challenge that cannot be easily overcome. The *chrt(1)* manual page is so poorly written as to be almost incomprehensible by the common man. The default Linux scheduler class will eventually adapt itself to the actual application behaviour and end up doing the right thing–after some time, during which the player may very well be killed by the ghosts!

7 Ghost Mode Management

A scheduler can hide another one. Quoth the **Pac-Man Dossier**:

> Ghosts alternate between *scatter* and *chase* modes during gameplay at predetermined intervals. These mode changes are easy to spot as the ghosts reverse direction when they occur. Scatter modes happen four times per level before the ghosts stay in chase mode indefinitely. [. . .] The scatter/chase timer gets reset whenever a life is lost or a level is completed. At the start of a level or after losing a life, ghosts emerge from the ghost pen already in the first of the four scatter modes.

In *scatter* mode, the ghosts navigate to their home corners. They are:

Pinky	top left
Blinky	top right
Clyde	bottom left
Inky	bottom right

The *gospel* goes so far as to specify the reverse engineered state transition time table (expressed in elapsed seconds).

Mode	Level 1	Levels 2-4	Levels 5+
Scatter	7	7	5
Chase	20	20	20
Scatter	7	7	5
Chase	20	20	20
Scatter	5	5	5
Chase	20	1033	1037
Scatter	5	1/60	1/60
Chase	$+\infty$	$+\infty$	$+\infty$

At this point, I felt the need to develop a ghost mode simulator completely independent from the game itself. Ultimately, I validated the concept and integrated it into the game's engine. I originally implemented the delays in strict millisecond terms; it later proved more useful to express those as multiples of the scheduler's timeslice unit (`clkperiod`).

8 AI Integration

There is, of course, no such thing as artificial *intelligence*. The only intelligence that can be perceived by an end user is that of the programmers' and the size of their factual database–this results, at best, in *unpredictable outcome computing*. And yet, the various ghosts moving strategies, as outlined in the **Pac-Man Dossier**, still refer to this as *AI*. The bottom line is that, when the *chase* mode is in effect, each ghost instance has a specific policy that is involved every time a direction change is possible. That policy set is fairly simple yet conducive to a great gameplay experience.

Blinky targets Pac-Man's current location.

Pinky targets a position that is four tiles ahead of Pac-Man's current moving direction.

Inky targets the end of a vector twice as long as the one originating from **Blinky** to Pac-Man's moving direction extrapolated by four half tiles.

Clyde does not know what it's doing. Its direction changes are as unpredictable as the LFSR based random number generator. The latter is, by design, completely deterministic.

From a programming perspective, this all comes down to the primitive listed as figure 5.

```
\ For every bit set in bitmap, we need to evaluate the
\ Euclidian distance between the potential next location and
\ the target tile. Finally we return the direction that
\ minimizes the distance.
\ Note: U> is ANS94 'Core Ext', so I'm using it.
: ghost.dirselect-nav2target ( self bitmap tvr tpc )
  ( -- new-dir )
  3 ROLL 3 ROLL          \ S: tvr\tpc\self\bitmap
  0 65535                \ S: tvr\tpc\self\bitmap\minoff\minval
  dir_blocked dir_up DO
    2 PICK I bitset? IF  \ Direction I is an option
      3 PICK e.pcol# C@
      \ S: self\bitmap\minoff\minval\pc-cur
      4 PICK e.vrow# C@
      \ S: self\bitmap\minoff\minval\pc-cur\vr-cur

      I case!
      dir_left  case? IF SWAP 1- SWAP THEN
      dir_right case? IF SWAP 1+ SWAP THEN
      dir_up    case? IF 1- THEN
      dir_down  case? IF 1+ THEN
      \ S: tvr\tpc\self\bitmap\minoff\minval\pc-next\vr-next

      \ Selected target is at [tvr, tpc].
      7 PICK - DUP *
      \ S: tvr\tpc\self\bitmap\minoff\minval\pc-next\dy^2
      SWAP 6 PICK - DUP *
      \ S: self\bitmap\minoff\minval\dy^2\dx^2
      +                  \ We compare the squared distance
    ELSE
      65535              \ uint16 max
    THEN
    \ S: tvr\tpc\self\bitmap\minoff\minval\minval-new

    2DUP U> IF
      NIP            \ S: tvr\tpc\self\bitmap\minoff\minval-new
      NIP I SWAP         \ S: tvr\tpc\self\bitmap\I\minval-new
    ELSE
      DROP
    THEN
  LOOP                   \ S: tvr\tpc\self\bitmap\minoff\minval

  DROP NIP NIP NIP NIP ;
```

Figure 5: Ghosts' Navigation Code

9 VT340 Port

This presented no real technical difficulty. The 340 has a higher resolution than the 420 and user defined fonts on the 340 are specified via a 10 by 20 bitmap matrix, as opposed to a 10 by 16 one for the 420. To that end I devised a C based font scaler utility which saved me a lot of time.

I also gave *fake text colour* rendition a try using ReGIS (`Remote Graphic Instruction Set`). It turned out that the required overhead on both the terminal processing abilities and the limited serial line throughput did not make such an option viable.

10 OpenVMS C Port

Back in April 2023, *VMS Software Inc.* announced the release of OpenVMS 9.2 for hobbyists (see [12]), targeting the x86-64 platform. This was, of course, music to my ears and I jumped on the bandwagon as soon as I was able to.

I started with a low ball target: a Linux/gcc 7.5 implementation. I resorted to a 32 bit cell representation for compatibility with VMS. Here again, standards come in handy. POSIX.1 (see [13]) naturally comes to mind when it comes to Unix interoperability. I did base my C port on that specification and it works reasonably well under GNU/Linux Mint 19.3 (5.4.0-150 kernel).

I experimented with POSIX based Forth essential primitives support via a dedicated prototyping tool. These central words were only a simple subset: `MS ?KEY AT-XY KEY CR`. The game logic inherited from the original Forth code was entirely preserved.

Upon startup, Forth expects the terminal to be in raw mode. In POSIX terms, this is expressed as an ad'hoc **tcgetattr/tcsetattr** system call sequence. Under OpenVMS, things are slightly different in that this has to have a *libcurses* equivalent incantation (**noecho/crmode**).

VMS also departs from the POSIX specification, although in a very marginal way, since the `select` system call is not implemented as a standard system interface but as a part of the network API.

A final note regarding C coding standards: one should definitely not rely on the *0b* prefix for specifying binary literals in C. This is a `gcc` extension and it is in no way standardized!

11 Conclusions

The *final* Forth code is about 2400 lines long. This amounts to about 60 blocks of CompactFlash storage. The user defined font requires some 600 lines for the VT420. The rest of the material is divided about evenly between comments and effective code. This software has been pushed into the public domain and published on Github [14]. The C port is of the same *order of magnitude*.

It works reasonably well under **Z79Forth**, partly because system resources do not have to be shared between competing processes in this environment. The Linux/C port can also, somehow, behave itself in a satisfactorily manner, after a few seconds. The OpenVMS/C port has proven to be a radical failure due to my inability to enable software flow control on the target system.

This being said, under **Z79Forth**, the lack of firmware supported exceptions can lead to disconcerting program termination at times. Currently, the traditional Unix SIGINT (emitted when Control-C is parsed from the controlling serial communication line) ends up being routed to the error handler routine. The latter issues an ASCII *Shift In* control character, causing the terminal to revert to the default character set. And yet, at the time of this writing, it still fails to restore the cursor status as being enabled (a VT200 control sequence). It all comes down to being able to emit `<CSI>25h` when you have only nine bytes of EEPROM available.

User-level exceptions (based on Mitch Bradley's `CATCH/THROW` model) have been successfully prototyped on the platform. The underlying code remains experimental and unpublished. Ideally, this should be an integral part of the firmware as supplied by default. At this point, *it is clear that more research is required!*

Key takeaways:

- *Standards Matter.*
- *Ad'hoc Tools* also are essential and will cut down your development time substantially.

References

[1] Bret Victor
Stop Drawing Dead Fish
`https://www.youtube.com/watch?v=ZfytHvgHybA`

[2] Digital
VT420 Text Terminal, commercial brochure
Digital Equipment Corporation, 1990.
`https://vtda.org/docs/computing/DEC/Terminals/EC-F0682_VT420TextTerminalBrochure_1990.pdf`

[3] *Programmers at Work, Toru Iwatani*
Microsoft Press, 1986. Penguin Books, 1989.
ISBN: 1-55615-211-6.
`https://programmersatwork.wordpress.com/toru-iwatani-1986-pacman-designer/`

[4] Game Internals
Understanding Pac-Man Ghost Behavior
`https://gameinternals.com/understanding-pac-man-ghost-behavior`

[5] Jamey Pittman
The Pac-Man Dossier. August 11, 2015
`https://pacman.holenet.info/`

[6] Roar Thornaes
Source code repository for the C++ Pac-Man Implementation
`https://sources.debian.org/src/pacman/10-6/`

[7] Forth2012 Standard Committee
Standard Specification. FACILITY EXT word set, BEGIN-STRUCTURE
`https://forth-standard.org/standard/facility/BEGIN-STRUCTURE`

[8] The Spriters Resource
The VG Resource, 2024.
`https://www.spriters-resource.com`

[9] Piskel
An interactive online sprite design application
`https://www.piskelapp.com`

[10] Jamey Pittman
NAMCO Physical Platform Specifications, 2015.
`https://pacman.holenet.info/#Specifications`

[11] Michael J. Pont
Patterns for Time-Triggered Embedded Systems
ACM Press/Addison-Wesley, 2001
ISBN: 0-201-33138-1

[12] VMS Software Inc.
OpenVMS 9.2-2 Public Release Announcement
`https://vmssoftware.com/about/v922/`

[13] The Open Group
IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)
`https://pubs.opengroup.org/onlinepubs/9699919799/`

[14] François Laagel
Source Code Repository for "Pac-Man for the DEC VT420"
Github, January 2024.
`https://github.com/forth2020/frenchie68/tree/main/pacman`