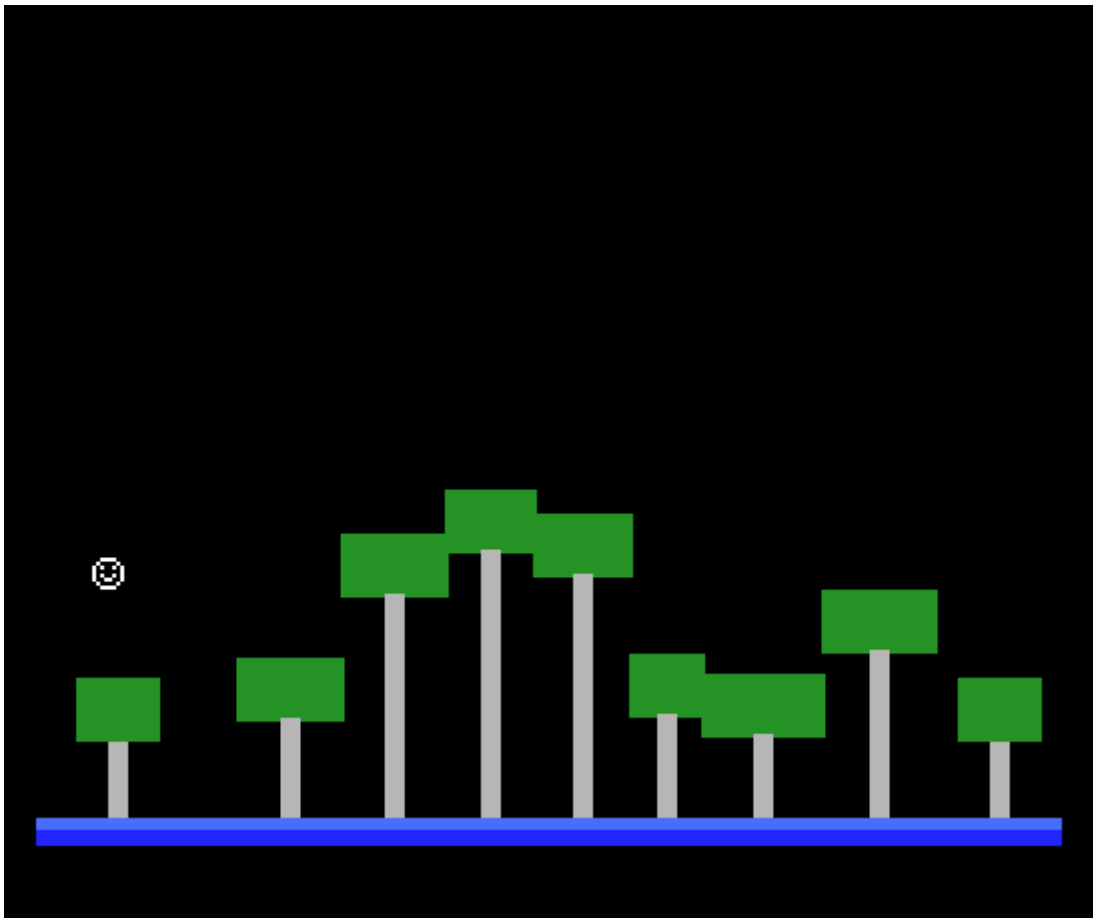
FLOH



This is a game for the BASIC 10 Liner Contest 2020 based on MSX 1 platform. More about the contest can be found here: https://gkanold.wixsite.com/homeputerium/kopie-von-2020

Game info

- Title: Floh (flea)
- Platform: MSX
- Author: Alexandre Lehmann Holzhey
- Language: MSX-BASIC 1.0
- Category: PUR-120

Files description

- README.md: This file, with general information about the game.
- FLOH.BAS: Source code of the game, in BASIC langauge.
- floh.png: Screenshot of the game.
- floh.dsk: Disk image with the game saved inside, for using with emulators.

The game

You are a flea (floh, in Deutsch) and need to keep yourself alive. Unfortunately, there is a water

flooding ongoing and to stay safe it is necessary to keep jumping above the water level. If you stay much time jumping without moving, you start to get tired and will also die. There is a food for the flea at the end of each level, in order to keep going!

How to play

The flea keeps jumping all the time, you have just to move it to the right or to the left, using th arrows keyboard keys (left or right). The moving is inertial, you control how much factor is added to the movement. You can also reduce the movement, going to the oposite direction.

But be aware that the flea will jump less high at each time. If you stay too much time at the same level, you got killed. Each level is a full moving from the left to the right or from the right to the left. A new level will be rendered and the flea will be kept without movement, to make things easier.

There is no end, but each level will have smaller plataforms to jump. If you get killed, the game restarts from first level.

Using an emulator

The game was developed on a Sharp Hotbit HB-8000 MSX 1 computer (manufactured at Brazil in 1985). You can use an emulator, i recommend the WebMSX one: https://webmsx.org/?MACHINE=MSX1&DISK=https://github.com/holzhey/floh/raw/master/floh.dsk Just use the provided URL and when ready type `load "a:floh.bas"`. Could be necessary to enter a date in order to boot the system, in this case just press the ENTER key.

Game logic

```
10 DEFINT A-Z:SCREEN 2,0,0:COLOR 5,0,0:W=0:FOR N=0 TO
7:READ V:VPOKE BASE(14)+N,V:NEXT N:V=16:X=14
```

- `DEFINT A-Z` declare all variables as integer, to make sure we run as fast as possible, without decimals calculations.
- `SCREEN 2,0,0` initialize the high resolution graphical mode (256 x 192 pixels, 16 colors), adjust sprites to be small sizes with 8 x 8 pixels and disable key clicks.
- `COLOR 5,0,0` sets the foreground color to 5 (dark blue), background and border colors to 0 (transparent).
- `W=0` initialize our flow state control to indicate a left-to-right level flow (0=left-to-right, 1=right-to-left). This flow is used to check if the flea achieves the end of the level, so we need to know which side it is.
- `FOR N=0 TO 7:READ V` we use a FOR loop to read all the 8 values that will compound the sprite image bitmap.
- `VPOKE BASE(14)+N,V:NEXT N` this is a small optimization for less code: we poke the sprite bitmap directly into VDP memory, using built in BASIC VPOKE command and retrieving the address of the start of sprites memory using BASE(14). The BASE command get information from VDP registers and the 14 one is the sprite memory base address.
- `V=16` is the variable that holds the difficult level. Lower the value, more hard it is. We start with i higher value here.
- `X=14` holds the flea X position at the screen, we need to initialize here due to the fact that each level starts from a different side of the screen, so a new game always start from the left side. Also, it have a small displacement to make some space between the platform at the edges of the screeen, so the player have to be really precise when landing at them. My initial

idea was to limit the movement at both sides of the screen, but this makes the game play slower and i wanted to achieve a very fast and smooth one!

```
30 V=V-1:Y=0:IY=1:IX=0:CLS:C=50:DATA
60,66,165,129,165,153,66,60:D=12:E=14:G=25-V
```

- `V=V-1` decrements the difficult level control, actually increasing it.
- `Y=0:IY=1:IX=0` intialize Y position and both X and Y increment values. We already initialized X before and we don't do it again here, since the flow behaviour of each level... we keep last X position for each new level.
- `CLS` just clear the screen.
- `C=50` prepare the X position that is used inside level rendering loop at line 35. We draw the first and last platform statically, since they are relevant for the game play (they must be there and must be easy to achieve during the game play). So, we skip the first one here and start from 50.
- `DATA 60,66,165,129,165,153,66,60` is just a smile icon bitmap in decimal notation.
- `D=12:E=14:G=25-V` we are counting chars per line here, so we store some values for usage at line below. We need two colors and the platform spacing value. Without this, we go beyond 120 chars at the next line!

```
35 S=V+RND(1)*20:L=RND(1)*50:LINE(C,150-L)-(C+S,165-
L),D,BF:LINE(C+(S/2)-2,165-L)-(C+
(S/2)+2,191),E,BF:C=C+S+G
```

Here we loop rendering the platforms using V to define the size of them. As we go to next levels, the size get smaller.

- `S=V+RND(1)*20` define the size of the platform to be rendered, using V to control the level/size of them.
- `L=RND(1)*50` define the height of the platform, relative to a fixed position (minimum for a good game play).
- `LINE(C,150-L)-(C+S,165-L),D,BF` draw the box for the top part of the platform. It should have color 12 (dark green) because we check for collisions with this color to detect jumps. We use the stored color value, on order to save one char.
- `LINE(C+(S/2)-2,165-L)-(C+(S/2)+2,191),E,BF` draw the base of the platform. This is pure graphical appeal, we do not check for collisions with this, so we use a different color here: 14 (grey). We draw it in the middle part of the size (width) of the platform determined by S. We use the stored value, in order to save another char (after i did it, i realize that other optimization already saved what i need... :-(
- `C=C+S+G` increment the position for the next platform, using the size plus the level spacing pre calculated at line 30.

```
38 IF C<200 THEN 35 ELSE FOR F=10 TO 240 STEP
220:LINE(F,150)-(F+20,165),12,BF:LINE(F+8,166)-
(F+12,191),14,BF:NEXT F
```

- `IF C<200 then 35` repeat the line 35 until we got column 200. Remember, we will draw the last platform in a few bytes, so we don't need to draw it. Since the screen have 256 columns, we stop the loop just before having enough space to draw the last one.
- `ELSE FOR F=10 TO 240 STEP 220:LINE(F,150)-`

`(F+20,165),12,BF:LINE(F+8,166)-(F+12,191),14,BF:NEXT F` is a small tricky loop to render the first and the last platforms. Instead of have lots of LINE commands (that take lots of chars, we are limited here!) we use this loop that begin at column 10 (first platform) and finishes at column 240, but the step value will loop at values 10 then 230 (last platform column). So, is just to reduce the amount of chars in the line.

39 LINE(0,188)-(255,191),4,BF:LINE(0,185)-(255,187),5,BF:PLAY "O3S8M1T120L16CR16CE.","O3S8M1T120L16FR16GO5C.":H=0

- `LINE(0,188)-(255,191),4,BF:LINE(0,185)-(255,187),5,BF` just draw the water level, using 2 degress of blue: 4 (dark blue) and 5 (bright blue).
- `PLAY "O3S8M1T120L16CR16CE.","O3S8M1T120L16FR16GO5C."` plays the initial music, just 3 polyphonic notes. We need to reset the envelope and timing since we change them for other sounds later.
- `H=0` reset the control of the jump height. We start from 0 here and count the number of interactions at each jump, in order to know the jump height. Smaller jump height means the flea is weak and we need to kill it... :-(

40 H=H+1:J=0:IY=IY+1:Y=Y+IY:X=X+IX:PUTSPRITE 0,(X,Y),15,0:IF POINT(X+4,Y+IY)=12 THEN J=1 ELSE IF Y>192 THEN 80

- `H=H+1` increment the jump height (iteration counter, actually).
- `J=0` is our flag to determine if there is a jump or not, see handling in this line later.
- `IY=IY+1` adds 1 to Y increment value. This behaves like gravity, increasing the falling speed each iteration.
- `Y=Y+IY` adds the Y increment to the Y position.
- `X=X+IX` adds the X increment to the X position. We handle the X increment value later, when we read keyboard keys.
- `PUTSPRITE 0,(X,Y),15,0` draw the sprite (our smiley icon) at depth 0, position X,Y, color 15 (white) and sprite bitmap number 0 (remember? we used the begin of the memory address for sprites, so numer 0 here, the first one).
- `IF POINT(X+4,Y+IY)=12 THEN J=1 ELSE IF Y>192 THEN 80` here we check for the color in the screen at position X+4 (almost the center of the sprite, since it have 8 bits/pixels width). If the color matches a platform hit, we set our jump flag (J) to 1. Otherwise, we check if the flea hits the water, going to line 80 to kill it in that scenario.

42 C=STICK(0):IF C=3 THEN IX=IX+1 ELSE IF C=7 THEN IX=IX-1

- `C=STICK(0):IF C=3 THEN IX=IX+1 ELSE IF C=7 THEN IX=IX-1` read the keyboard arrows and as right or left was pressed, increase the X increment value or decrease it. This give us that inertial movement behavior.

44 IF J=0 THEN 40 ELSE IY=-IY:PLAY "M3300S8T250O4A":IF H<8 THEN 80 ELSE H=0

- `IF J=0 THEN 40` just repeat the iteration, since we don't have anything else to deal (no jump).
- `ELSE IY=-IY:PLAY "M3300S8T250O4A"` since we have a jump here, let's invert the

Y increment value (like a jump does) and play some bouncing sound (we setup the volume envelope here to a volume down ramp, and use a long note to do it).

- `IF H<8 THEN 80 ELSE H=0` but, if the last height value was too small, means the flea is weak... dead... so we goto line 80 to kill it, otherwise we just reset the jump height in order to count it again.

`55 IF W=0 AND X>230 THEN W=1:GOTO 30 ELSE IF W=1 AND X<30 THEN W=0:GOTO 30 ELSE GOTO 40`

- `IF W=0 AND X>230 THEN W=1:GOTO 30` in case we are going from left-to-right flow and we hit the right side in this jump, let's invert the flow direction and go to next level. We do that from line 30, that recreate everything but keeping the level and x position.
- `ELSE IF W=1 AND X<30 THEN W=0:GOTO 30` is the same approach, but from right-to-left flow. Please note that we only evaluate if the flea achieve the other extremity of the screen when it jumps (in our case, when it touches the platform).
- `ELSE GOTO 40` is a normal jump, no extremity was achieved, just go to main loop.

`80 PLAY "O3S8M100C":FOR F=15 TO 0 STEP -1:COLOR F,F,F:NEXT F:COLOR 5,0,0:X=14:W=0:V=16:GOTO 30`

- `PLAY "O3S8M100C"` since we are dead, play some dark sound... we use an envelope to help on that.
- `FOR F=15 TO 0 STEP -1:COLOR F,F,F:NEXT F:COLOR 5,0,0` just flashes the screen and reset the game colours after that. This happens while we are playing the sound.
- `X=14:W=0:V=16:GOTO 30` we need to setup the initial values for X, W and V in order to restart drawing level after death. Of course, we can also jump to line 10, but i decided to avoid that sprite processing at begining (also the screen command, are a bit slow).