

1. The project

This game was projected to be fit in 10 lines and also show how to move all the 32 sprites of MSX in a fast way although the low speed of MSX-BASIC. To achieve this, the sprites trajectory need to be pre-renderized and the game takes about 3:20 minutes before to be ready to play. Please be patient!

2. How to play

Insert the disk on MSX and after the MSX-DOS be started you need to type freeway.

Then it is necessary to wait data decompressing and pre-renderization of sprites trajectory. During the process some vertical lines will be drawing. At the end of process, the street and the cars will appear.

On that screen you can press:

[SPACE] or [0] Game start.

[1-7] Level select: you need to wait renderization again. Higher numbers means that fast cars has more probability to be present. On high levels, you can try to choose the same level and see very different car speeds setup due the randomization. In level 1, the unique difference is the positioning of each car.

And during the game:

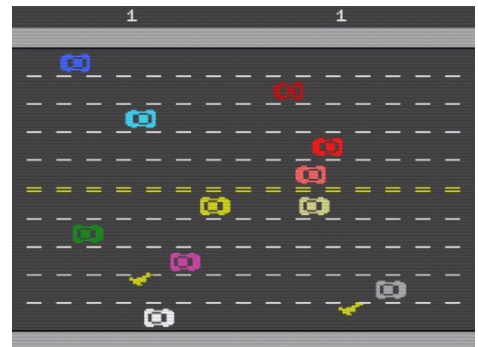
[A/Z] Moves the left chicken.

[Up/Down] Moves the right chicken.

[ESC] Stop the game and back to Ready screen.

You need to hold ESC for up to 8 seconds.

[CTRL+STOP] Quit to BASIC and you can list the code typing LIST.



Whoever passes more chickens across the street, wins the game!

3. Playing on emulators

Note: The car horn sound effect will never or rarely be played on emulators. It happens because the game uses an VDP registry to fast randomize which sound effects to play.

3.1 WebMSX

It's very easy to run, but you need to choose the version without the raindrops because until this moment, it doesn't emulate well the sprite collision disable-bit.

Enter the link <https://webmsx.org/> and drag & drop the file freeway.dsk (or click on Drive-A icon and browse to freeway.dsk). Resets the emulator after the insertion of the disk.

Pressing F12 you can run faster when the game are in the pre-renderization.

The NetPlay of WebMSX works well with this game. It was very nice to play with a friend!

3.2 OpenMSX

It is the current best emulator to play, but can be a little hard to explain here how to setup it for work with floppy disk. OpenMSX only comes with the minimum BIOS-ROM files necessary for ROM type games. But if you have an MSX2 with floppy setup ready, you probably knows what to do.

3.3 BlueMSX

Sprite collision is not well emulated when the players are in the same horizontal line.

3.4 fMSX

It has similar issue like BlueMSX.

3.5 ParaMSX

It has another kind of sprite collision problem. It is impossible to play because the collision never stops.

The **line 0** is a commentary line being used to store data. The data is under a Run Length compression made by Compress.bas program (*included on disk*). The **first 72 caracteres** is a kind of ID/token (I called it "alias" on compress.bas source code): each of them represents a byte used one or more times in sequence by the data. The compressed data informs the program to "calls" for this tokens and also informs how many repetitions of that byte need to be used.

The first character of first line is always stored on RAM at address 8008H and we can easily read this bytes through PEEK function. I only use bytes above 20H (SPACE charcode), since bellow that are the control characters and BASIC use two bytes to store it. I also avoided byte 7FH because is DEL control character. The compressor searches the raw data for all the different bytes used and create IDs to represent them. To represent the bytes 7FH and those bellow 20H, I use the byte 20H before: 20H followed by his code+64.

Each ID invoked by compressed has different lengths options. First ID can reach 16, the next 8 IDs reach 8 and the last 68 IDs have length 1 or 2. The compressor makes the most used bytes come first in the ID area.

The compressed data are used for three objectives: sprite format (goes directly to VRAM), provides coordinates for LINE command (goes to first part of array A) and provides the car's trajectory increments to create the different car speeds (goes to last part of array A). The decompressing process loads VRAM and array, it is done by the **line 2**.

0 'ÿbøðóε€ x'VUT@?83*^ \ Y X W P L H F D Εµ",qo^M<+Z N C *B A ào J óñĩ!@
#Ši³Ž.³|.³|.³".³~³š.³œ.³ž.³.³-#;çĩ#...ĩĩ#çĩ#n[n[†#ŰwŌ{āwĒ▲ā▲w▲x▲xšāšwšB«†šāfĒš'x|#f
(Ó#ā1āB
ioiBā'ČikwkiC'YieGéiY'İiB'Biİ+ ,İŋk▲k+Űevuā.İŋl▲*»Cývu#ā-İŋk▲+Űývvāu.İŋk.»Cývu/İŋk.Ű

Line 1 is the general initialization. Here we set all variable to integer to improve speed. I only put RND(-TIME) inner DIM to reduce space. It is almost the same as: "DIMA(300):A=RND(-TIME)". RND(-TIME) is equivalent to RANDOMIZE statement of another kinds of BASIC language. It makes the randomic table to be updated according the TIME System Counter that is usually imprevisible.

I setup a DEFFN (define function) to reduce space when necessary to call the function RND(1)*64 many times. OPEN"grp:" allows to use PRINT#1 that prints on graphic mode. All the pages of Screen is clear. I fill from 3 to 0 so that it continues to work on page 0. The street will be drawn on page 0 using color 1 (black) but I will change this palette during the game to make gray and another tones. I fill page 3 with 14 because it will be a coordinate to hide the rain-drops sprites in not defined trajectories.

I intended to put the sprite attribute at page 1 to allow do a COPY from page 2 and 3 to page 1, updating the attributes of all sprites at the same command.

Pages 2 and 3 will receive the pre rendered trajectories to be repeated copied at page 1 during the game. Each line of page 2 or 3 copied will be like one frame of game.

W\$ and W\$() receives sfx to be played with PLAY statement.
The VPOKE puts the X coordinate of the right chicken.
VDP(19) re-adjust the screen do center, because in follow I setup VDP to hide the 8 pixels on the left side. It makes more softly when the car sprites appears on the left.
Unfortunately only MSX2+ can hide this 8 pixels. So, I check peek(45) to know if is a MSX2+, if yes, it changes VDP(26) to do that.

```
PRINT:FOR I=&H8000 TO I+250:PRINT RIGHT$("00"+HEX$(PEEK(I)),2);:NEXT
```

```
FFFFF8FD034988BF20736856555440F3833245E205205920582057205020420482046204420620420402038203620342032203020282026202420222020201F40201E201D201C201B201A20192018201720162015201420132012201120100F0E0D0C0B0A09080706050403020100
```

The HEX display of the *commentary* on line 0. There are 251 bytes. First 72 are the IDs and the compressed data are the last.

[illegible]

Here is the HEX display of the raw data before compression (354 bytes). Table bellow shows the IDs for each byte, were:

ps = position on ID range

dat = the byte represented (!=control char region)

l_n = length in sequence

tt = how many times used in total

```
1 DEFINT A-Z:DIMA(300-RND(-TIME)):DEFFNR=RND(1)*64:OPEN"GRP:"AS#1:SCREEN5,2,0:FORI=0TO3:
  SETPAGE,3-I:COLOR,-(I=3)-14*(I=0),0:CLS:NEXT:W$="v14132o7deb":W$(3)="o7m256m17s8c4":VPOKE-
  32135,181:W$(2)="o7m256m28s8f3":VDP(19)=4:IFPEEK(45)>1THENVDP(26)=VDP(26)OR2
```

Line 2 starting defining a very low sound for the motor sfx. Now we will read the compressed data. "O" receives the RAM address that points the commentary line but in a position just between the IDs and the data, so that, to the left of this point we have each ID and to the right we have the compressed data. On the compressed data side each value in a specific formula, indicates which ID must be selected (which byte) and this same value also indicates how many repetitions it must be done using this byte. So the decompressing process separate the value in different ranges to identify what ID must be selected and how many repetitions must be provided to the following FOR loop that will fills the VRAM or the A() array. When it send the sprites format to the VRAM, it also copy to an another VRAM region. It is because during the game, this two regions will be swaped using one VDP registry in the way to change boths chicken animations in one BASIC command improving speed.

```
2 SOUND3,15:O=-32688:FORI=0TOI+178:A=PEEK(I)-35:G=(A>15AND A<81):H=(A>80):P=-(1+(A-16)\8)*G-
  (9+(A-80)\2)*H:R=-(A<16)*AMOD16-G*AMOD8-H*AMOD2:B=PEEK(O-P-1):B=B-(PEEK(O-P-2)-96)*(B=32):
  FORD=DTOD+R:IFD<98THENA(D)=BELSEVPOKED+30622,B:VPOKED+28574-32*(D>193),B
```

Line 3 finish the past loop and goes to draw the street on Page 0 using LINE command with values decompressed from A(). Also sends the Y coordinate of each street stripe to the respective car Y coordinate and selects the sprite format ID of each car according his respective direction. At the end fills Sprite Color Table according each layer. The raindrops receives 39 that means color 7 (cian) and also sets bit 5 that disables the sprite collision, the chicken receives 10 (yellow) and cars receive a different color according his index.

```
3 NEXTD,I:FORI=0TO13:FORJ=0TO224STEP16:LINE(A(C+3)+J,A(C+2))-(A(C)+J,A(C+1)),A(56+I),BF:
  NEXT:VPOKEC-32180,A(C+1+4*(I>5)):VPOKEC-32178,4-4*(I>4)-12*(I>10):C=C+4:NEXT:SETPAGE,1:
  FORI=0TOI+511:VPOKEI,-39*(I<304)-10*(I>479)-(I\16-14)*(I>303AND I<480):NEXT:L=4
```

Line 4 starting configuring the frequency of the scoring sound and goes to start the pre-renderization of rain. Each first part of the line of page 2 receive a frame with all rain sprites attributes filled, we need only to created numbers for his coordinates X and Y. The second part of the line will receive the cars attributes in following. This lines will update attribtues of the 30 sprites. The remaing 2 sprites are the chickens and will be updated in the conventional way. The renderization used 170 lines in each page that is a good number to make a car cross the screen using a constant speed of 3 increments. I think would better to use 256 lines, but the MSX-BASIC don't allow to use the COPY command beyond the line 212.

After the rain, starts the renderization of the cars. There are one renderization for the page 2 (without rain) and other for the page 3 (together with the rain). We copy all the static atributes (Y coordinate and sprite format ID) first to "speed up" the renderization, since the we only need to change X coordinate to move the cars.

```
4 SOUND4,95:SETPAGE2,2:FORW=0TO72STEP4:FORI=0TO169:A=128*I:VPOKEA+W,16+(W*14+I*24)MOD200:
  VPOKEA+W+1,(W*8-I*10)AND255:NEXTI,W:FORP=2TO3:SETPAGEP,P:FORI=0TO169:COPY(152,4)-(240,4),1
  TO(152,I):NEXT:FORI=0TO10:A=77+I*4:IFI<>6THENT=70+4*INT(FNR*P/(192/L))
```

About **line 5**, the IF I<>6 on line 4 and X(5) appears because I want to put 2 cars in this line.

The lasts values of array A() contains the table with the number of times and the increments to be applied to the coordinate X for each car. There are 7 types of car speed comportment and 4 bytes for each type. The first byte of the table is the number of times to use the increment and the second is the increment value. The next 2 bytes has the same mean, to create a second variation for that speed.

W\$(0) and W\$(1) receives the car passing sound in different speeds.

```
320 DATA 086,1,084,2 : ' Very-Slow to Slow
330 DATA 084,2,086,1 : ' Slow to Very-Slow
340 DATA 128,1,042,3 : ' Extreme-slow to Normal
350 DATA 169,3,001,3 : ' Normal
360 DATA 085,2,085,4 : ' Slow to Fast
370 DATA 128,4,042,6 : ' Fast to Very-Fast
380 DATA 127,5,043,3 : ' Very-Fast to Normal
```

```
R1=086:I1=1 : R2= 084:I2=2
Ok
? "Total: ";R1+R2,"Traject: ";R1*I1+R2*I2
Total: 170 Traject: 254
Ok
```

Creation of the car speed increments on the program Compress.bas. The total of frames must be 170 and the traject must be a multiple of 256 (width of screen) for that car restart the traject in the same position.

At the end of line 5 I setup the left chicken X coordinate. The I use VDP(5)/VDP(12) to setup the Sprite Format Table to an address in Page 1 of Screen 5. The formula is: Sprite Format Address = 128 * (VDP(5)-3) and VDP(12) is the page.

```
5 X(I)=-X(I)*(P=3)-(X(5)+24+FNR*3)*(P=2):FORJ=TTOT+3STEP2:FORR=1TOA(J):
X(I)=(X(I)+A(J+1)*(I<5)-A(J+1)*(I>4))AND255:VPOKEA,X(I):A=A+128:NEXTR,J,I,P:SETPAGE0,0:
A$="00v11cv13cv14c01v12b":W$(1)="11"+A$:W$(0)="12"+A$:VPOKE-32131,69:VDP(5)=7:VDP(12)=1
```

Line 6 stops the motor/rain sound for when it returns from another match. Then PEEK(-1043) reads the space key and at the same time PEEK(-1051) receives the numbers 0 to 7. I use LOG function * 1.45 to easily transform bits 1 to 7 (1,2,4,8,16,32,128) in the numbers 1 to 7. If the user presses the numbers, the program goes to **line 4** to restart renderization in the another selected level L. I could use INKEY\$, but is very boring when you press two times SPACE and it goes to keybuffer: when the match finishes, it restarts involuntarily.

If SPACE is pressed it initializes the game variables. Scores are put in -1 and the chickens are put in the top, so that soon in the start of the game the Score is updated from -1 to 0, clearing any past score of screen from another match.

In this line the external loop "FOR I" controls how many 170 frames cycles must be run until it changes the weather and finally finish the game returning to the start of line 6. You can change the duration of the match modifying this line. According "I" counter, streert color palette and noise registry are updated to change the weather.

```
6 SOUND9,0:A=254ORPEEK(-1043)XORPEEK(-1051):IFA>1THENL=LOG(A)*1.45:GOTO4ELSEIFA=0GOTO6ELSE
E=0:F=0:Y=0:Z=0:S=-1:T=S:FORI=0TO80:N=(I>8ANDI<16):D=3+N:SOUND7,56+14*N:O=20-8*N:
COLOR=(1,3+N+(I<20),2+N,2):VPOKE-32134,O:VPOKE-32130,O:IFI=23ORPEEK(-1044)=251GOTO6
```

Line 7 has the main loop that counts each frame. The frame is updated two times in relation to the chickens. It speeds up the cars movimentation and turns the game more interesting.

It is necessary to always fix the motor/rain volume channel (SOUND 9,10)because the PLAY statament that plays the other sfx resets the volumes of all channels when it finishes to play any sound.

A=VDP(Q-5) is a way to refresh the VDP registry that receives the collision coordinates. It was necessary because I could not use ON SPRITE GOSUB only using 10 lines.

All sprites are updated on the COPY statement: VDP Hardware transfer gets almost a entire line of Page 2 or 3 and copy to Page 1 that it is the place where a I put the sprites attribute table. In this line each 2 pixels is one byte (Screen 5 mode). Each byte will be used to fills attribute of one sprite. Each 4 bytes controls one sprite. The meaning of the bytes are: Y coord., X coord., Format ID, nothing.

PEEK(-1046)reads the keys A/Z and then I update the coordinates using only expressions(no "IFs"). Then I check if one of the chickens has arrived to the top, if yes, plays the sound and update the score.

```
7 FORJ=0TO169STEP2:SOUND9,10:A=VDP(Q-5):COPY(0,J)-STEP(239,1),DTO(0,4),1:Y=-(Y-(3*(PEEK(-
1049)=191)-3*(PEEK(-1046)=127))*(E=0)-(D=2))*(Y<195)-9*(E>2)-194*(Y>194):IFY>8GOTO8ELSE
S=S+1:SOUND12,40:SOUND4,85:SOUND10,16:SOUND13,0:PRESET(62,3):PRINT#1,S:Y=197
```

Line 8 starts swapping the sprite format table using VDP(6), it animates the both chickens. Peek(-3099) is a little fast way to get VDP(6) value. It is equivalent to VDP(6)=VDP(6)XOR1.

One chicken sprite coordinate is updated. I read Up/Down key by PEEK(-1043) and update the cars using the same COPY again. Then it checks if the other chicken has arrived to the top and updates his score.

```
8 VDP(6)=PEEK(-3099)XOR1:VPOKE-32132,Y:Z=-(Z-(3*(PEEK(-1043)=223)-3*(PEEK(-1043)=191))
*(F=0)-(D=2))*(Z<195)-9*(F>2)-194*(Z>194):COPY(0,J+1)-STEP(239,1),DTO(0,4),1:IFZ>8GOTO9ELSE
T=T+1:SOUND12,40:SOUND4,96:SOUND10,16:SOUND13,0:PRESET(174,3):PRINT#1,T:Z=197
```

Line 9 starts updating the another chicken's sprite coordinate, then I read bit 5 of VDP(8) (status register S#0) to check sprite collision and read VDP(-3) that receives the collision coordinate X. It defines which chicken has collided.

When you read VDP(-5) it clears the collision coordinate X of registry VDP(-3), without this, VDP(-3) stays with the previous stored number. Now VDP(-3) keeps receiving any kind of "strange" numbers until the VDP finds a collision (note: on emulators these "strange" numbers not happen). The "A" variable would have received, sometimes VDP(-5), if sprite collided or, VDP(-3) if not (see **line 7** and **9**). I use "Q" variable to swap between VDP(-3) or VDP(-5). If no collision occurs, on each 76 frames the program invokes a sound effect according "A". How no collision occurred, "A" must have imprevisible values that selects the sounds to be played. I used that because it's faster than RND function.

```
9 VPOKE-32136,Z:Q=2:E=E+(E>0):F=F+(F>0):IFVDP(8)AND32THENPLAYW$:IFVDP(-3)<99THENE=5:VPOKE-32130,16:NEXTJ,ELSEF=5:VPOKE-32134,16:NEXTJ,ELSEQ=0:IFE=1ORF=1THENVPOKE-32130,O:VPOKE-32134,O:NEXTJ,ELSEIFJMOD76<>32ORPLAY(1)THENNEXTJ,ELSEPLAYW$(AAND3):NEXTJ,I
```

5. Limitations

5.1 Pre-renderization is boring.

5.2 While a player is sufrerring collision, the second player collisions can not be detected, the other player can passes trough the car, but is a very rare situation.

5.3 The car positioning is randomic and, sometimes many cars are positioned near each other at the same speed, creating a kind of "big walking wall". It can turn the game hard for one player and easy for the other. Although this, when weather changes to rain the car's trajectory are changed and it can swaps the situation, turning the game harder to the other player.

6. Conclusions

6.1 The pre-renderization technics allowed the game to reaches 10 FPS and it was playable. Using more lines would be possible to use ON STICK GOSUB and ON SPRITE GOSUB, allowing to remove the manually sprite collision checking. I think it could reach around 12-15 FPS only using MSX-BASIC.

6.2 This technics also allows to easily reaches 60fps using XBASIC (aka Kunbasic) compiler, built in on some MSX2+ computers, or the recent Amaury Carvalho's compiler, called MsxBasToRom (<https://launchpad.net/msxbas2rom>). Maybe I will try to do this or write an article about that.

6.3 I realize that CALL MUSIC(0) / PLAY #2 statement of FM-BIOS is faster to be called and allows to play faster notes, even when PLAY#2 is used to play only PSG sounds. The original PLAY of BIOS don't play faster notes like "L64" for example, it ignores and plays it like about "L32".

6.4 The horns sfx volume is a little high and can be irritating because it was necessary to use PSG envelopes and it only works at the maximum volume.

Author: Carlos Olivi

Feb/18th/2021