Merlin's Walls

3-D FOR VCS

—

WHY AND HOW

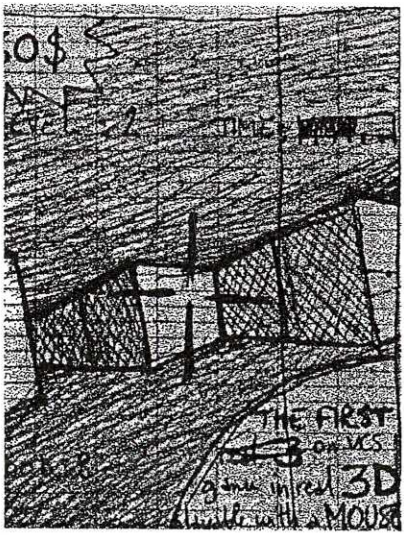
by Igor Barzilai

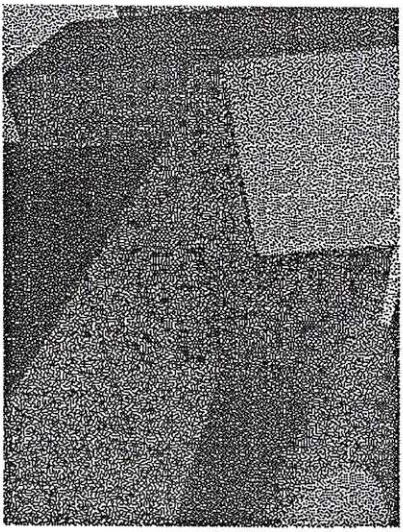
It had been an adventure with lot of hopes, doubts and great moments. Maybe I'll find a way to have the picture right, to put sprites in the maze, to have a better graphic quality and even (why not), to put textures on the walls. But programming is always a question of human and system time and I have other projects for the 2600. So we'll maybe see what the Atari Video Computer System can still do next time.

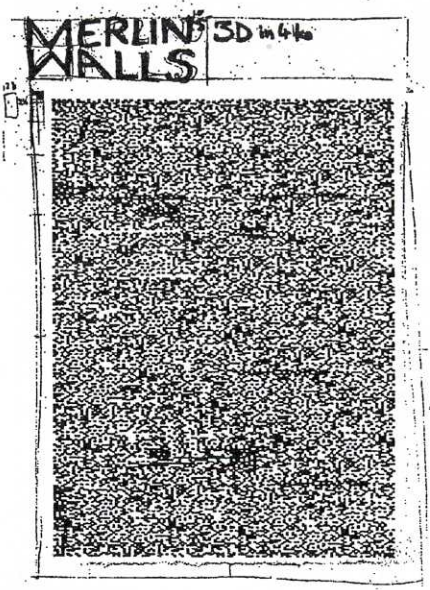
Hope you'll like Merlin's walls,
thanks to you,

Igor.


the First 3D game for your VCS 2600 Atari.

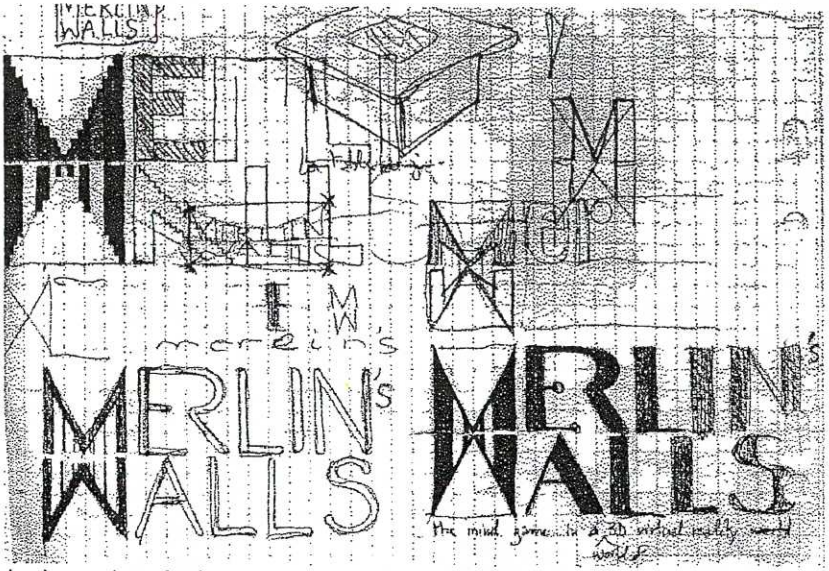

THE FIRST
on VCS
game in real 3D




MERLIN'S WALLS
3D in 4k

Why...

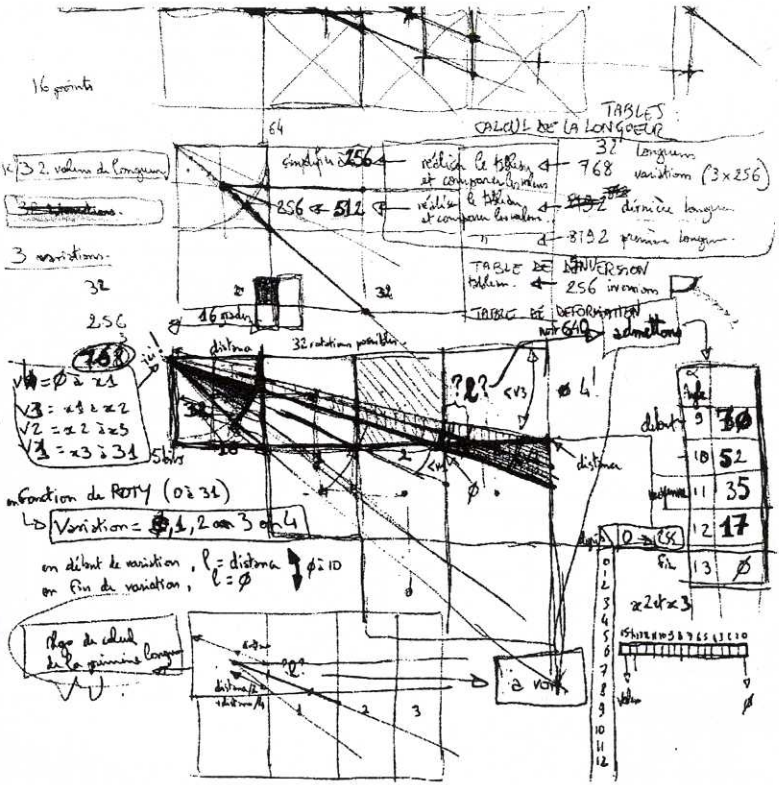

3D MAZE
BONUS PACK
ARCADE FORMAT
ONLY ATARI MAKES THE MOST POPULAR GAMES ON YOUR TV!
INCREDIBLE BUT TRUE!
WE PUT YOU IN THE GAME!
FOR THE FIRST TIME ON YOUR ATARI 2600!
REAL 3D
50 images / second.

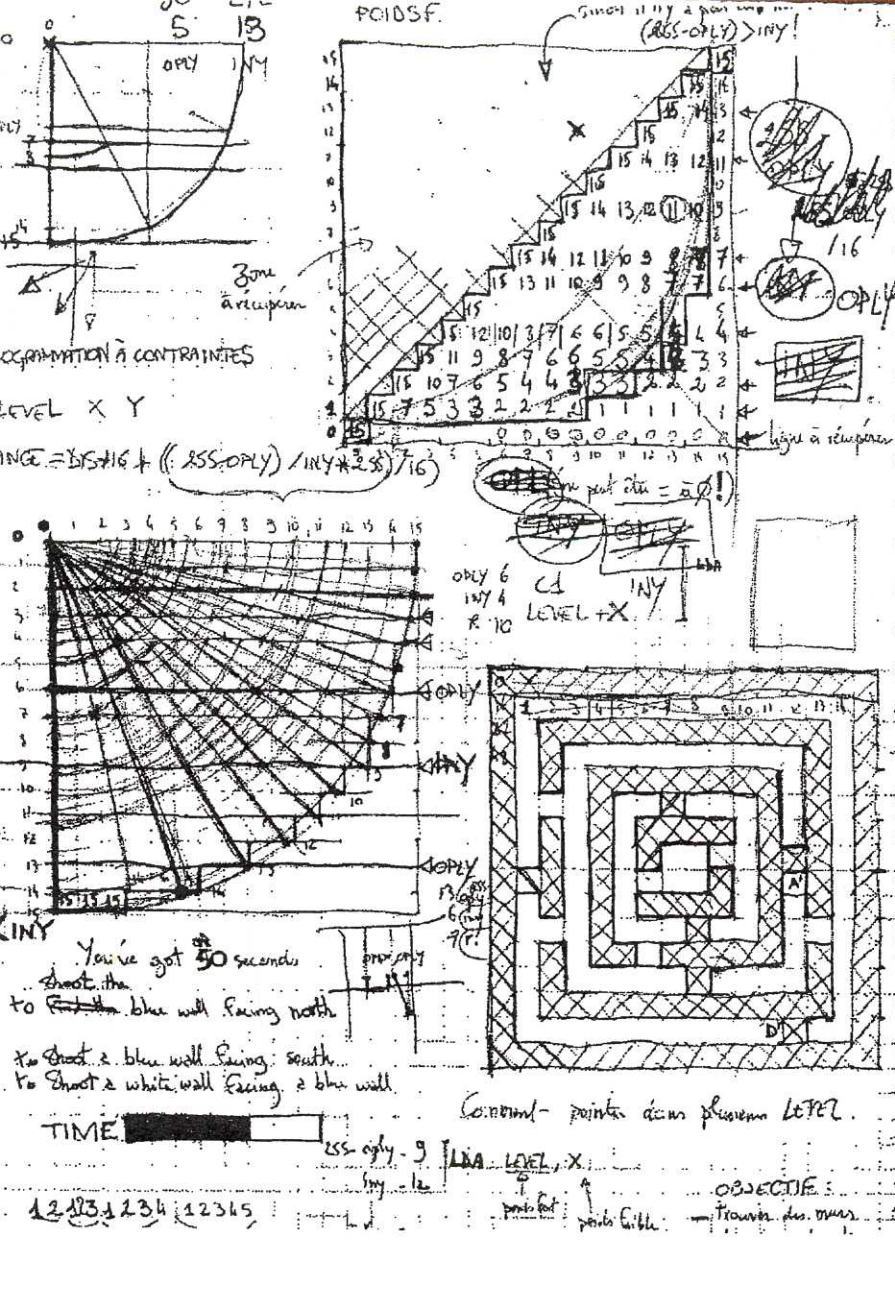


In the early 80's, I was 16, and in the city I lived, there was a supermarket where computers were switched on with a curious screen always saying «READY». It was the time when all the industrial productors were making their home computers. I used to go there after school. One day, a guy showed me how to do things with some keywords. That's how I discovered the Basic on Atari 800 XL. A year later, my parents bought it to me. I never had a week without programming since this day.

I always thought that one of the most interesting thing to do with computers is picture. Remembering a videogame magazine showing the screens made by an air flight simulator for the army, I started to think about how to do «real pictures» with my basic. (I used to say about them «real pictures» but there's now a word to call them: Virtual Reality). But two things were missing: I wasn't good enough in algebra, and even if I was, the Atari basic was too slow to do it in real time. The machine language should be the anti-slow solution, cause guys at Lucasfilm made it on 800 XL with excellent games like «Rescue on Fractalus» or «The Eidolon». Some other made «Capture the flag» and «Way out». It was ten years before Doom and on a chip more than 50 times slower than a Intel 386. So I made a lot of other kind of programs, keeping 3D in my mind and drawing a lot of shems in the expective to do 3D picture in real-time someday...

... Two year ago, I met Eric Bacher who told me about the Stella VCS emulator and the possibilities to make our own cartridges for the Atari Video Computer System. He gave me the bases and the tools for PC to work on it in M.L. My research in 3D were mature and I estimated that the 6507 could give me the strenght needed for a 3D real-time algorithm. In fact, I didn't know if it was possible, but the challenge do do it was so exciting that I started to work on paper.



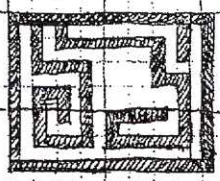

POIDS F.
(255-OPLY) > INY !
OPLY
INY
Zone à récupérer
OGRAMMATION À CONTRAINTES
EVEL X Y
INGE = DIS*16 + ((255-OPLY) / INY * 255) / 16)
OPLY 6
INY 4
R 10
LEVEL + X
Zone à récupérer
You've got 50 seconds
to Shoot the blue wall facing north
to Shoot a blue wall facing south
to Shoot a white wall facing a blue wall
TIME
255 - oply - 9
Inny - 12
LDA LEVEL, X
OBJECTIF :
Trouver des murs
1 2 1 3 1 2 3 4 1 2 3 4 5

```
BORDURE=1
DEBUT  P1=PEEK (START1+B)
       PH=PEEK (START2+B)
       LOCATE P2,Z (=PEEK(LOW+P2))
       IF Z=15 THEN GOTO
INCREBORDURE=BORDURE+1
       IF B<16 THEN GOTO DEBUT
       GOTO LABY_FIN
LABY_NEWCOLOR=COLOR+1
       PLOT P1,COLOR
       PLOT PH,COLOR
LABY_REPERE S=Ø : O=Ø
LABY_SUITE HIGH=PEEK(MOVE2+O)+PH
       LOCATE M1,Z
       IF Z=15 THEN GOTO LABYSUITE
       IF O<4 THEN GOTO REPERE
       IF S=Ø THEN GOTO INCREBORDURE
       O=O+1 : GOTO LABY_SOLUCE
LABYSUITE POKE SOLUCE1Ø, PEEK(MOVE1+O)+P2
       POKE SOLUCEHØ, HIGH
       S=S+1
       O=O+1
       IF S=Ø THEN GOTO INCREBORDURE
       IF S<>1 THEN GOTO CHOIX_2_3
CHOIXØ POKE LOW, PEEK(SOLUCE1Ø)
       HIGH=HIGH+PEEK(SOLUCEHØ)
       POKE HIGH, COLOR
       GOTO LABY_REPERE
CHOIX_2_3 IF S=3 THEN GOTO DERNIERE
CCC1   A=PEEK(OCTET+COMPTEUR)
       INC COMPTEUR
       IF A=Ø THEN GOTO CHOIXØ
CHOIX1 POKE LOW+PEEK(SOLUCE11), COLOR
       HIGH=HIGH+PEEK(SOLUCEH1)
       POKE HIGH, COLOR
       GOTO LABY_REPERE
DERNIERE A=PEEK(OCTET+COMPTEUR)
       INC COMPTEUR
       B=PEEK(OCTET+COMPTEUR)
       INC COMPTEUR
       IF A=Ø THEN GOTO TEST2
       IF B=Ø THEN GOTO INCRE
       GOTO CHOIX1

TEST2  IF B=Ø THEN GOTO CHOIXØ
CCCHOIX2 POKE LOW+PEEK(SOLUCEL2), COLOR
       POKE HIGH+PEEK(SOLUCEH2)
       HIGH=HIGH+PEEK(SOLUCEH2)
```

START ① □ Ecran d'info, start, mise à Ø doivent
LABY. ② □ Construction laby → ⑦
TRIDSDE ③ □ Routine 3D
On trouve
Li3solution ① □ déplacement joystick
□ code spécial → ⑧
TIR ⑤ □ fonctions du tir → ⑦
MUSIC □ musique
⑥ □ Reset-soluce → ①
TEXTE ⑦ □ Affichage texte/image → ③
SHOWLABY ⑧ □ Affichage du laby.

Passage soute

SHOWLABY.



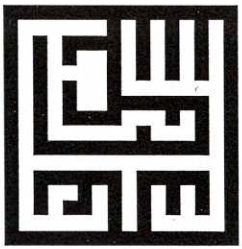
Level Ø01

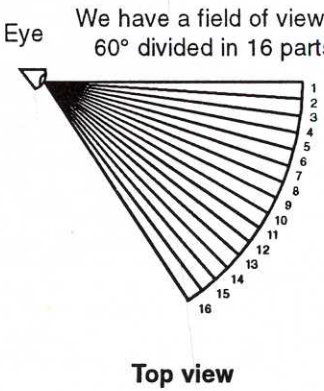
How...

The algorithm

Here is a technical presentation of the 3D algorithm on VCS. It wont explain everything in detail of course. I suppose that most of you are not really interested about how to program 3D, but here are the main lines.

We've got a maze in a 16 x 16 grid. Why 16 ? Because 16 x 16 is 256, and 256 is the number of combination that a byte can do. A byte is 8 bits, and the 6507 is an 8 bits chipset. This is just an example to show that everything had been done with numbers from 0 to 255 for one reason: talk to the chip in his native language as fast as possible.



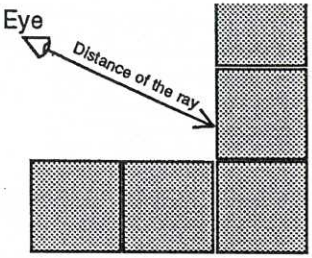


Top view

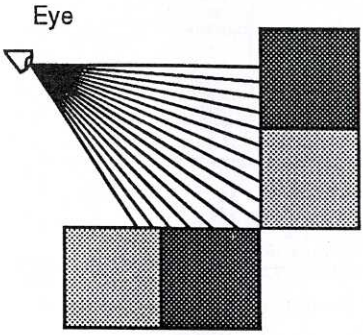
We are in this maze, and have a position, rotation, and a field of view. In our case, the field of view is made by 16°. A complete circle in our system is made by 96° (not 360 degree nor 200 grades !). Compared to our classic system, we have the sensation to see with a 60° angular optic.

For different reasons, I finally use 12 degrees instead of 16. So, let's continue with ou 16 degrees.

So we ray-trace this 16 rays from the eye through the space of the maze. It means that we throw a ray from the eye of the player in the direction of the first degree (of the 16) and when we see a wall, we stop and look the distance we've done.

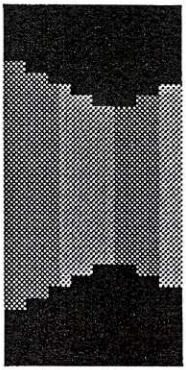


Top view

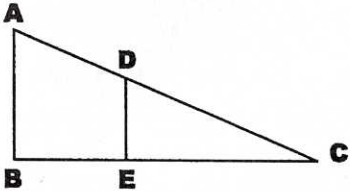


Top view

More the distance is long, more the wall will be little. We do it with the 2nd ray, with the 3rd until the 16th. When it's done, we've got the complete picture of the maze from the point of view of the eye's player. The VCS does this computing 60 times per second.



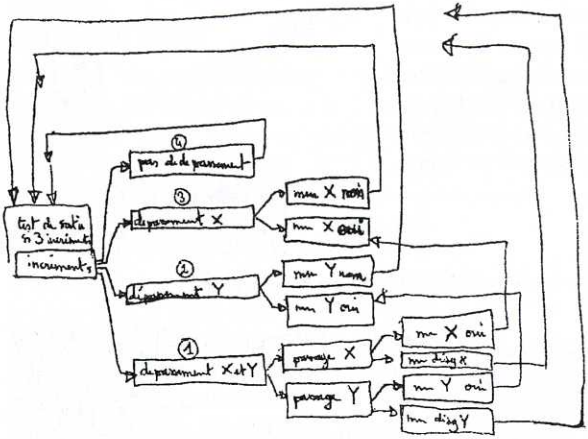
Eye view



If you want to really understand how we calculate it, you can refer to a 3D programming book. In résumé, the algorithm is based on the Pythagore theory about the triangle wich says :

$$\frac{AB}{AC} = \frac{DE}{DC}$$

... wich is the only mathematic part of the algorithm, the rest is just logic.



Principle of the ray-traced system

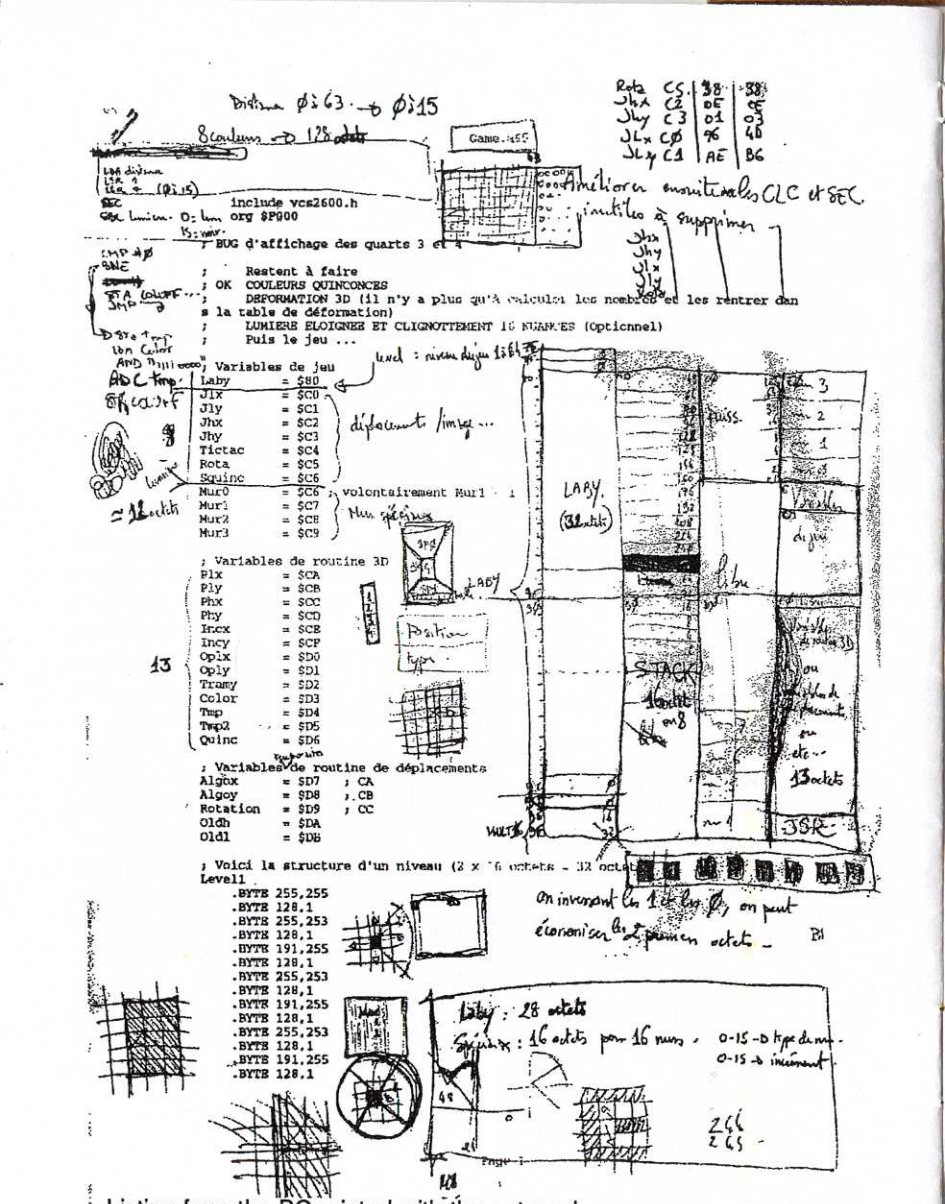
After having developped the theoric method to use, I started to program it in Basic on a 800 XL. Why Basic ? Because it's nearer to our mathematic language than the M.L. and then easier to talk. So I used it to test the integrity of the algorithm. I Wrote it on paper, keeping in back mind the fact that it will be turned in 6507 machine language later. It took maybe 10 months of writing and debugging. When it finally worked, I translated it in M.L. on paper and tried it with PCAE on my PC 75Mhz... Another long time of debugging. At this time, I was about one year of work on Merlin's Wall's... but the funniest part had to come: the game.

```
:PLY=JLY:R=RND(0)+Y:IF R>55 THEN R=R-36
21  INX=PEEK(SIST+R):INY=PEEK(COCO+R):D
IS=-1
22  IF R<80 THEN 25
23  IF R<72 THEN 110
24  GOTO 150
25  IF R>23 THEN 70
30  DIS=DIS+1:IF DIS<4 THEN C=0:GOTO 95
0
31  OPLX=PLX:OPLY=PLY:REM  OPHX=PHX:OP
HY=PHY
33  PLY=PLY+INY
34  IF PLY<256 THEN GOTO 36
35  PLY=PLY-256:PHY=PHY+1:GOTO 39
36  PLX=PLX+INX:IF PLX>255 THEN GOTO 45
37  GOTO 49
39  PLY=PLY+INX:IF PLX>255 THEN GOTO 50
40  C=PEEK(LEVEL+PHX+PEEK(MULT16+PHY))
41  IF C=0 THEN GOTO 30
42  ORI=0:DISTANCE=PEEK(MULT16+DIS)+PEE
K(POTOSL+INT((255-OPLY)/16)*16+(PEEK(DIV
16+INY)))
43  GOTO 190
45  PLX=PLX-256:PHX=PHX+1
46  C=PEEK(LEVEL+PHX+PEEK(MULT16+PHY))
47  IF C=0 THEN GOTO 30
48  ORI=1:DISTANCE=PEEK(MULT16+DIS)+PEE
K(POTOSL+INT((255-OPLX)/16)*16+(PEEK(DIV
16+INX)))
49  GOTO 190
50  PLX=PLX-256:PHX=PHX+1:C1=PEEK(SECT+
PEEK(DIV16+OPLX)+16*INT((OPLY)/16))
51  REM  PLX=PLX-256:PHX=PHX+1:C1=PEEK
(SECT+INT((OPLX)/16)+16*INT((OPLY)/16))
52  IF C1>INX THEN GOTO 60
55  C=PEEK(LEVEL+PHX+PEEK(MUL 16)+PHY))
56  IF C=0 THEN GOTO 40
57  GOTO 48
```

PREMIER QUART

```
30 DIS=DIS+1:IF DIS=MAX THEN C=0:GOTO 190
31 OPLX=PLX:OPLY=PLY
33 PLY=PLY+INY
34 IF PLY<256 THEN GOTO 36
35 PLY=PLY-256:PHY=PHY+1:GOTO 39
36 PLX=PLX+INX:IF PLX>255 THEN 45
37 GOTO 46
38 PLX=PLX+INX:IF PLX>255 THEN 50
40 XX=PHY:AA=PEEK(MULT16+YY):AA=AA+PHX:YY=AA:AA=PEEK(LEVEL+YY):C=AA
41 IF C=0 THEN GOTO 30
42 ORI=0:DISTANCE=PEEK(MULT16+DIS)+PEEK(PCIDSL+INT((256-OPLY)/16)*16+(PEEK(DIV15
+INY)))
43 GOTO 190
45 PLX=PLX-256:PHX=PHX+1
46 XX=PHX(LEVEL+PHX+PEEK(MULT16+PHY))
47 IF C=0 THEN GOTO 30
48 ORI=1:DISTANCE=PEEK(MULT16+DIS)+PEEK(PCIDSL+INT((256-OPLX)/16)*16+(PEEK(DIV15
+INX)))
49 GOTO 190
50 PLX=PLX-256:PHX=PHX+1:C=PEEK(SECT+PEEK(DIV16+OPLX)+16*INT(OPLY/16))
52 IF C1>INY THEN GOTO 55
55 X=PEEK(LEVEL+PHX+PEEK(MULT16+PHY)):REM TEST CASE A DROITE
56 IF C=0 THEN GOTO 46
58 GOTO 48
60 X=PEEK(LEVEL+(PHX-1)+PEEK(MULT16+PHY)):REM TEST CASE EN BAS
61 IF C=0 THEN GOTO 46
62 GOTO 42
```

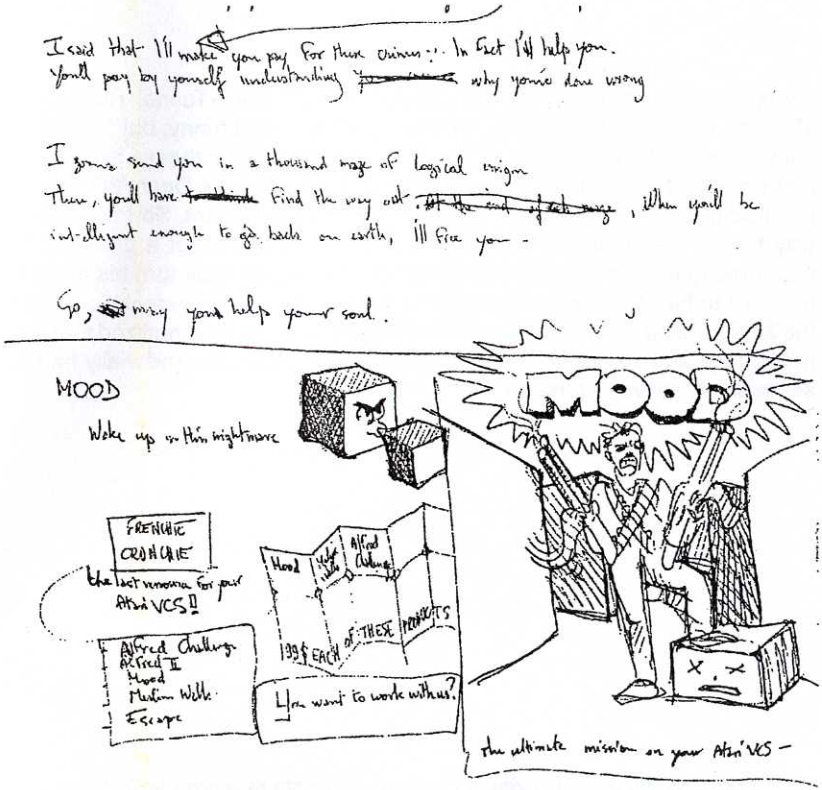
Listing in Basic printed with a 1027 Atari printer



```
; BUG d'affichage des quarts 3 et 4
;
;     Restent à faire
; OK  COULEURS QUINCONCES
;     DEFORMATION 3D (il n'y a plus qu'à calculer les nombres et les rentrer dan
s la table de déformation)
;     LUMIERE ELOIGNEE ET CLIGNOTTEMENT 16 NUANCES (Opticnnel)
;     Puis le jeu ...

; Variables de jeu
Laby    = $80
Jlx     = $C0
Jly     = $C1
Jhx     = $C2
Jhy     = $C3
Tictac  = $C4
Rota    = $C5
Squinc  = $C6
Mur0    = $C6 ; volontairement Mur1 - 1
Mur1    = $C7
Mur2    = $C8
Mur3    = $C9

; Variables de routine 3D
P1x     = $CA
P1y     = $CB
Phx     = $CC
Phy     = $CD
Incx    = $CE
Incy    = $CF
Opix    = $D0
Opiy    = $D1
Tramy   = $D2
Color   = $D3
Tmp     = $D4
Tmp2    = $D5
Quinc   = $D6

; Variables de routine de déplacements
Algox   = $D7   ; CA
Algoy   = $D8   ; CB
Rotation = $D9  ; CC
Oldh    = $DA
Oldl    = $DB

; Voici la structure d'un niveau (2 x 16 octets = 32 octets
Level1
        .BYTE 255,255
        .BYTE 128,1
        .BYTE 255,253
        .BYTE 128,1
        .BYTE 191,255
        .BYTE 128,1
        .BYTE 255,253
        .BYTE 128,1
        .BYTE 191,255
        .BYTE 128,1
        .BYTE 255,253
        .BYTE 128,1
        .BYTE 191,255
        .BYTE 128,1
```

Listing from the PC printed with the notepad

When I started this project, I wanted this program not to be just a 3D demo, but a strategy or adventure game based on an original scenario. Then, what can we do in a maze with different color walls ? Find the way out. I draw a first maze and put some special walls in it (walls that can be pushed, destroyed...). By placing them differently, I obtained differents levels of difficluty. One idea is that when you did one level in a laby, it helps you for the second one in the same laby because you learn to know the space. Then with four different labys and four levels on each laby, we've got the complete 16 levels of Merlin's Walls.



There are few games using 3D visions on VCS like «Tunnel Runner» or «Escape from the Mindscape», they are beautifull and funny, but I wanted a game in real 3D. I mean that, when you turn right or left in theses games, you turn directly of 90°. The 3D I dreamed of had to be like the Doom 3D (without textures of course): totally free to move wherever you want. So I searched a way to ask it as fast as possible to the 6507 chipset. I met a problem that made me having to turn the picture 90°, so the player must turn his head (or the T.V.) to be able to play. The second bad point is the low resolution due to the time missing and to the Atari 2600 playfield system, but I realized that after holding the joystick a minute, I completely forgot the pixels, and really had the sensation to move in a maze.