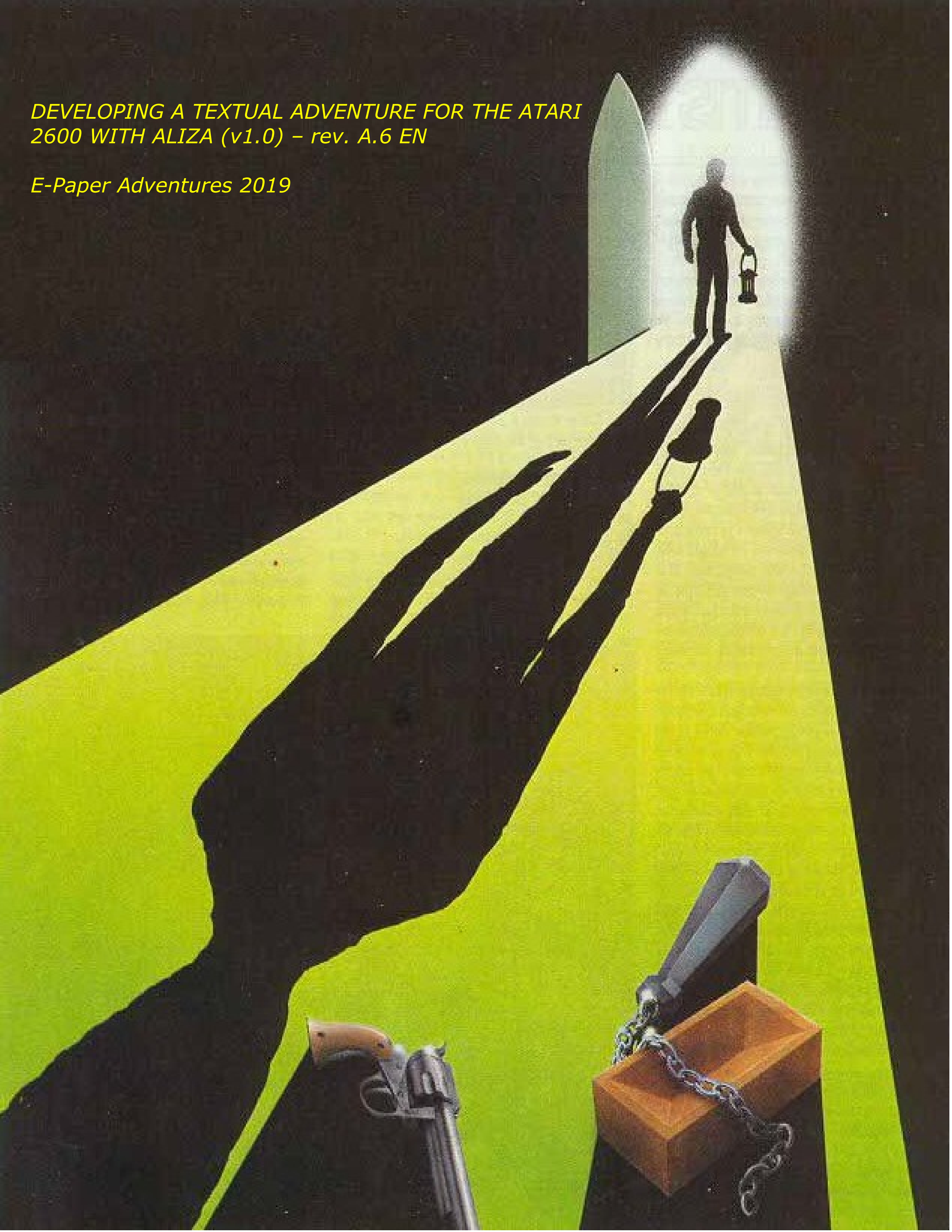
DEVELOPING A TEXTUAL ADVENTURE FOR THE ATARI 2600 WITH ALIZA (v1.0) – rev. A.6 EN

E-Paper Adventures 2019



Welcome!

Aliza is a freeware portable Windows software to create textual adventure on the ATARI 2600.

In this tutorial document an example of adventure is developed step by step: “The eyeball”. All the needed resources are at the link: www.epaperadventures.qlmagic.com

The project is 100% free. In case of suggestions or comments, you can write to us at epaperadventures@gmail.com ...thanks! :)

Images and texts of this tutorial are based on the magazine “INPUT” (Marshall-Cavendish) from 1984/1985. Also, this work is based on the code present in the book “Making Games for the Atari 2600” by Steven Hugg.

The result of the development with Aliza is an assembly file to use in the 8bitworkshop IDE. But to develop the adventure a very simple pseudo-basic language is used.

The reccomended tools are Notepad++ and Mozilla Firefox.

In case you would make a small donation, please consider to do it to:

RICOMINCIO DA CANE ONLUS

www.ricominciodacane.it

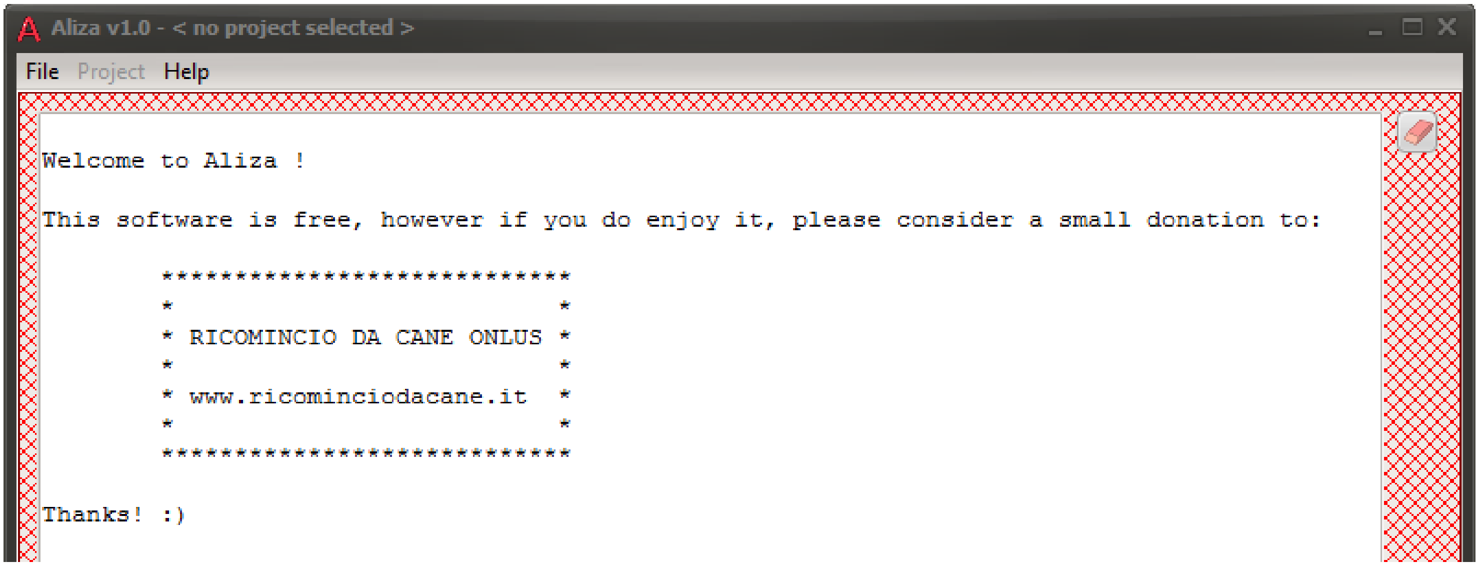
Thanks from the E-Paper Adventures staff!

Revision A.6 – en - Dec 2022

This project is dedicated to Annalisa <3

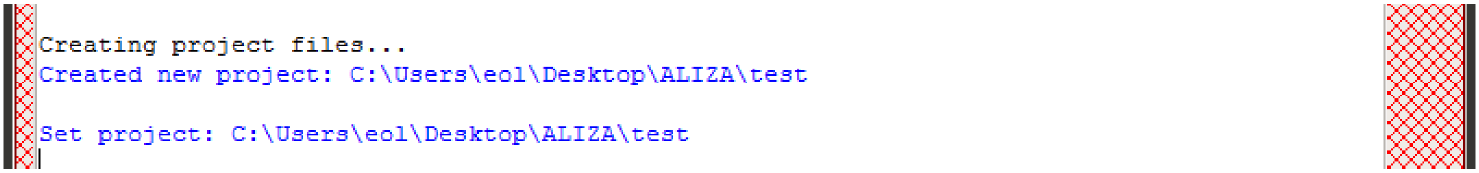
Starting a new project!

To start, extract from the downloaded ZIP file the *aliza.exe* application, for example in a folder on the desktop named “ALIZA” and execute it.



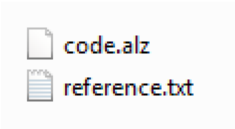
```
Welcome to Aliza !

This software is free, however if you do enjoy it, please consider a small donation to:

         *****************************
         *                           *
         * RICOMINCIO DA CANE ONLUS *
         *                           *
         * www.ricominciodacane.it   *
         *                           *
         *****************************

Thanks! :)
```

Go to the *File* menu and create a new project by *New project.* Select for example "test" as name. Aliza creates a sub-folder *test* with some files. Also, that folder is set as project folder.



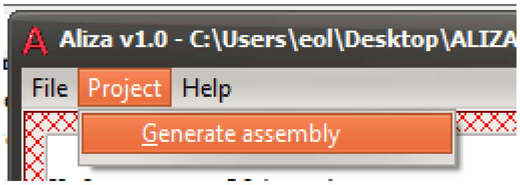
```
Creating project files...
Created new project: C:\Users\eol\Desktop\ALIZA\test

Set project: C:\Users\eol\Desktop\ALIZA\test
```

These are the files in *test:*

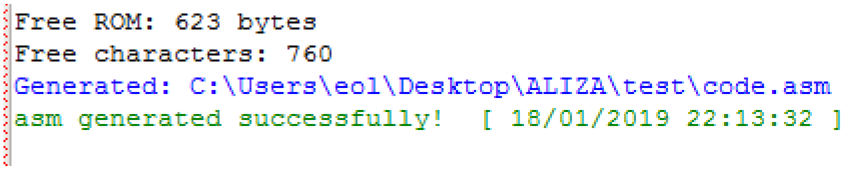


```
code.alz
reference.txt
```

Every projets contains a default adventure pseudo-basic code (code.alz) ready to be compiled. Just to test, go in *Project* menu and choose *Generate assembly:*



Aliza generates the *code.asm* file based on the content of the *code.alz* file.

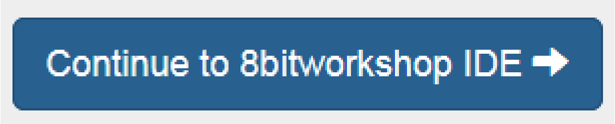


```
Free ROM: 623 bytes
Free characters: 760
Generated: C:\Users\eol\Desktop\ALIZA\test\code.asm
asm generated successfully!  [ 18/01/2019 22:13:32 ]
```

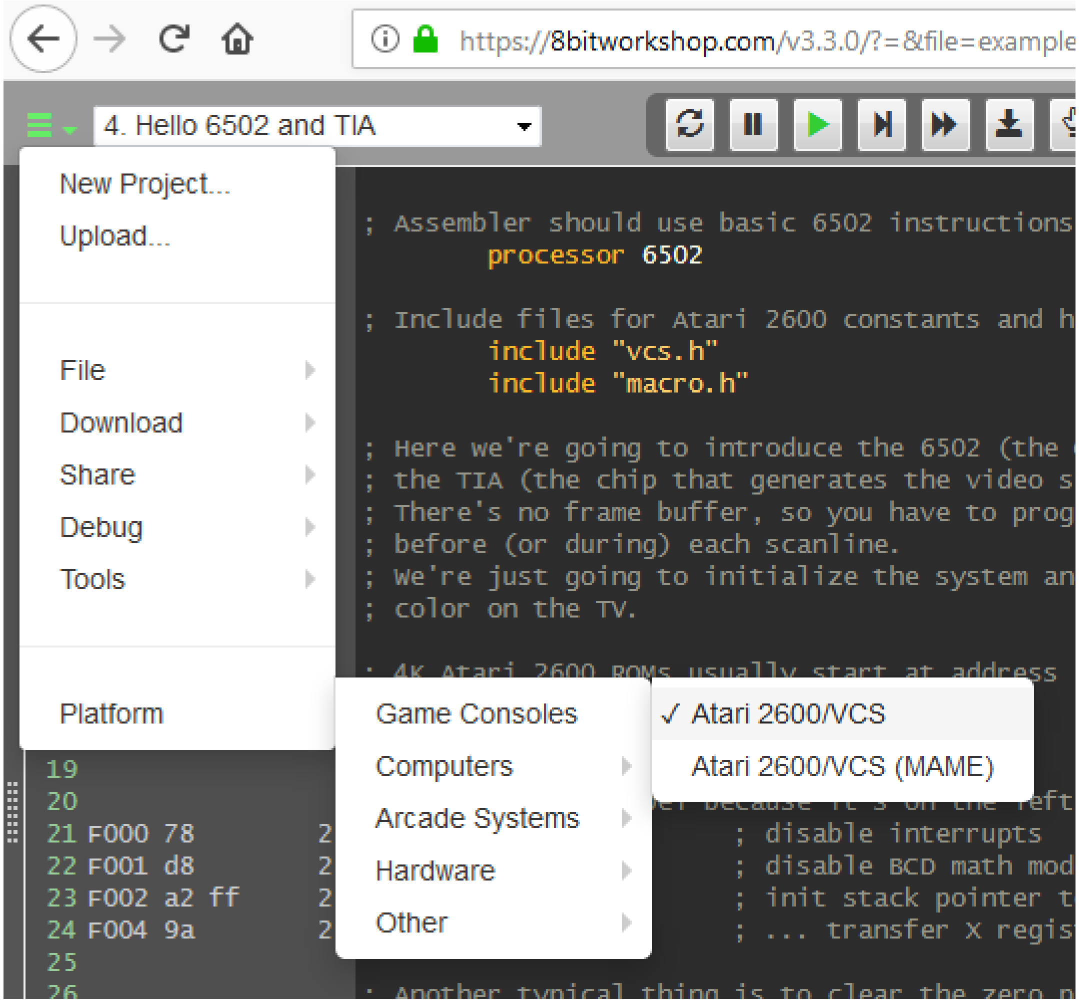
Open *code.asm* with *Notepad++*. It contains assembly code. Copy it, go on:

https://8bitworkshop.com/

Click on:



One the web page is loaded, click on top-left icon and select ATARI 2600 as platform:

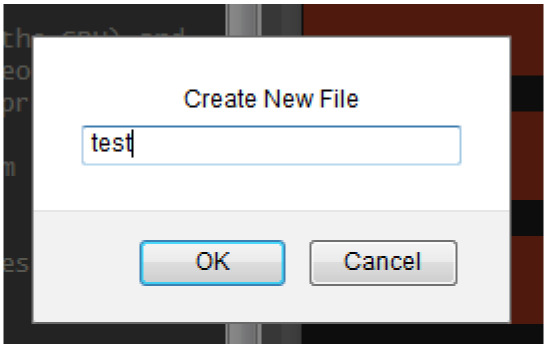


4. Hello 6502 and TIA

```
; Assembler should use basic 6502 instructions
        processor 6502

; Include files for Atari 2600 constants and h
        include "vcs.h"
        include "macro.h"

; Here we're going to introduce the 6502 (the 
; the TIA (the chip that generates the video s
; There's no frame buffer, so you have to prog
; before (or during) each scanline.
; We're just going to initialize the system an
; color on the TV.

; 4K Atari 2600 ROMs usually start at address
```

```
F000 78            2                ; disable interrupts
F001 d8            2                ; disable BCD math mod
F002 a2 ff         2                ; init stack pointer t
F004 9a            2                ; ... transfer X regis

                                ; Another typical thing is to clear the zero p
```

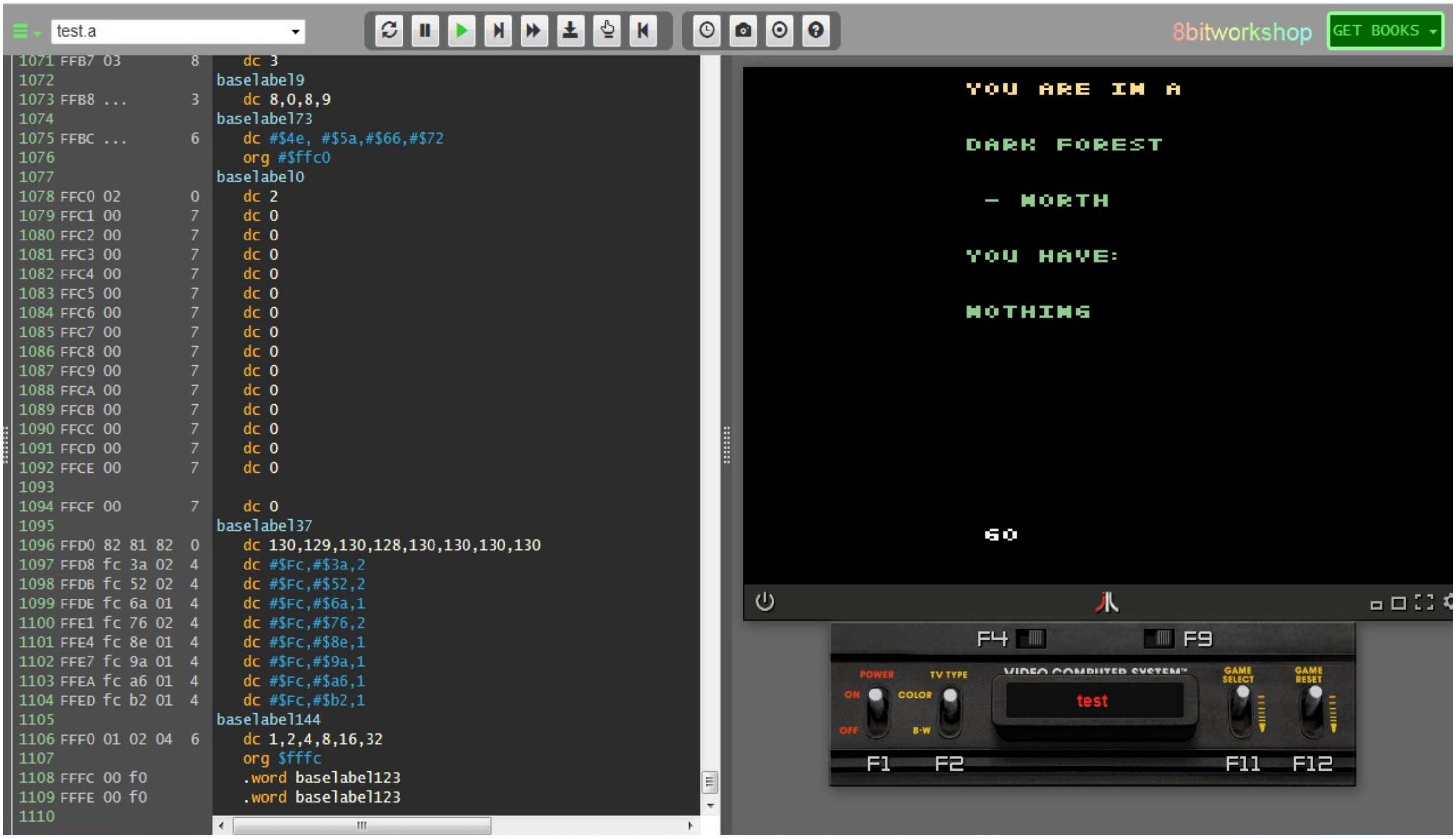
Then, select *"New project",* digit *"test"* and enter.



Create New File

test

Cancel all the code automatically generated and paste the code copied from *code.asm:*



```
FFB7 03          8      dc 3
                     baselabel9
FFB8 ...         3      dc 8,0,8,9
                     baselabel73
FFBC ...         6      dc #$4e, #$5a,#$66,#$72
                        org #$ffc0
                     baselabel0
FFC0 02          0      dc 2
FFC1 00          7      dc 0
FFC2 00          7      dc 0
FFC3 00          7      dc 0
FFC4 00          7      dc 0
FFC5 00          7      dc 0
FFC6 00          7      dc 0
FFC7 00          7      dc 0
FFC8 00          7      dc 0
FFC9 00          7      dc 0
FFCA 00          7      dc 0
FFCB 00          7      dc 0
FFCC 00          7      dc 0
FFCD 00          7      dc 0
FFCE 00          7      dc 0

FFCF 00          7      dc 0
                     baselabel37
FFD0 82 81 82    0      dc 130,129,130,128,130,130,130,130
FFD8 fc 3a 02    4      dc #$Fc,#$3a,2
FFDB fc 52 02    4      dc #$Fc,#$52,2
FFDE fc 6a 01    4      dc #$Fc,#$6a,1
FFE1 fc 76 02    4      dc #$Fc,#$76,2
FFE4 fc 8e 01    4      dc #$Fc,#$8e,1
FFE7 fc 9a 01    4      dc #$Fc,#$9a,1
FFEA fc a6 01    4      dc #$Fc,#$a6,1
FFED fc b2 01    4      dc #$Fc,#$b2,1
                     baselabel144
FFF0 01 02 04    6      dc 1,2,4,8,16,32
                        org $fffc
FFFC 00 f0              .word baselabel123
FFFE 00 f0              .word baselabel123
```

```
YOU ARE IN A

DARK FOREST

 - NORTH

YOU HAVE:

NOTHING


 GO
```

The assembly will be executed in the right window (atari 2600 emulator).

Note: please read appendix A (bugs found December 2022)

Here is how to play:

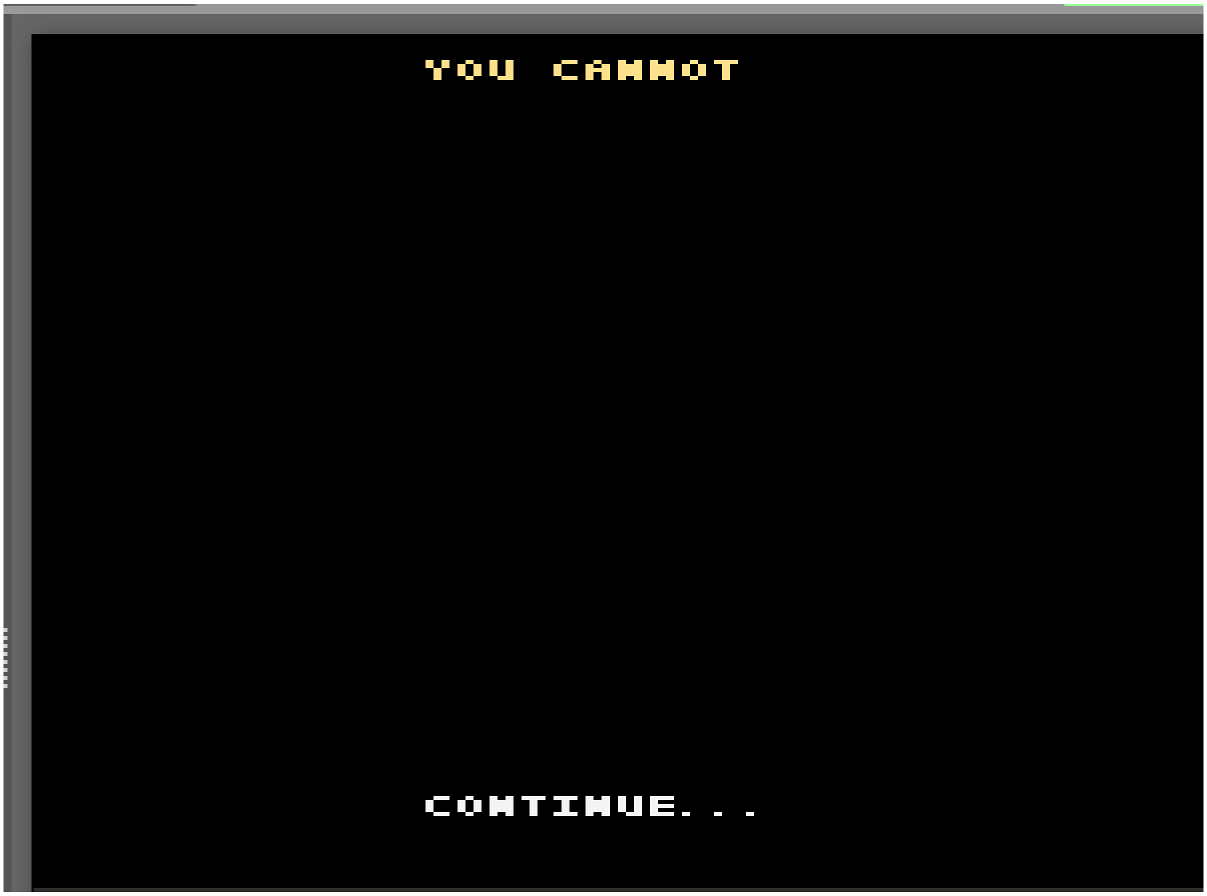
- left or right cursors (player 1 joy left or right) to change the VERB: GO, EXAMINE,TAKE, USE, PUSH, PULL, DROP
- up or down cursors (joy up or down) to select a text line (yellow) containing something to act on
- space (joy fire) to execute the action (for example: EXAMINE DARK FOREST)
- use the option “GAME RESET” to restart the game

Basically, the game screen presents:

- the current room description
- possible directions (north, south, east, west)
- object present in the room (“floating” objects)
- inventory
- the verb selected (in white)

If the text exceed the first screen, going down a second screen is presented.

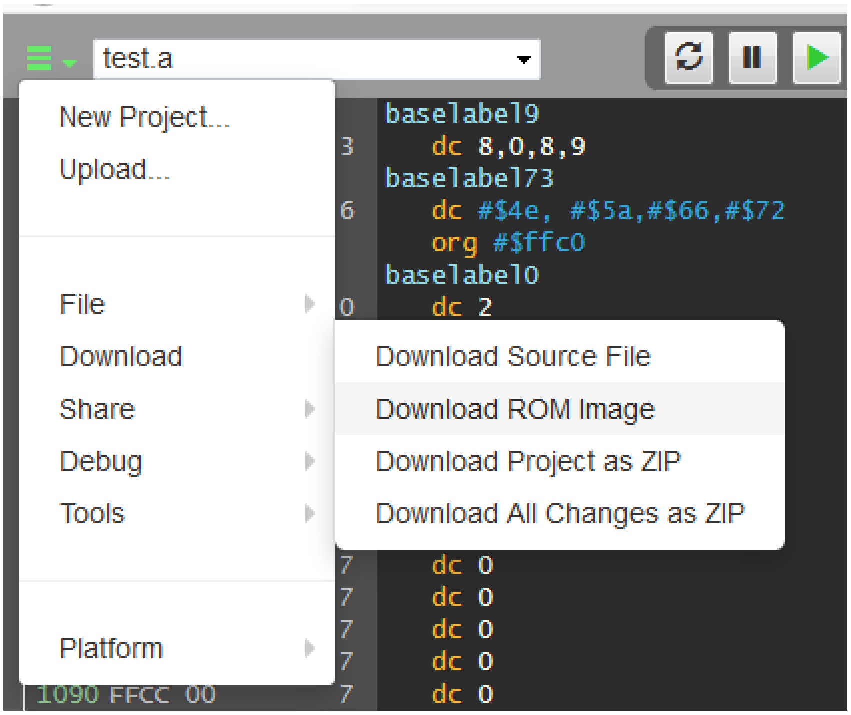
Actions can generate as result a message, for example “YOU CANNOT”. In this case you can only hit space (fire) to continue (if possible).



YOU CANNOT

CONTINUE...

To create a binary file to use in other emulators, there is a “download ROM image” option in the site:



```
baselabel9
   dc 8,0,8,9
baselabel73
   dc #$4e, #$5a,#$66,#$72
   org #$ffc0
baselabel0
   dc 2
```

```
   dc 0
   dc 0
   dc 0
   dc 0
   dc 0
```

The file *reference.txt* created in the folder contains a short reference to write the code of your adventure in *code.alz*

Note: an updated version of the reference file is available in appendix B

The eyeball

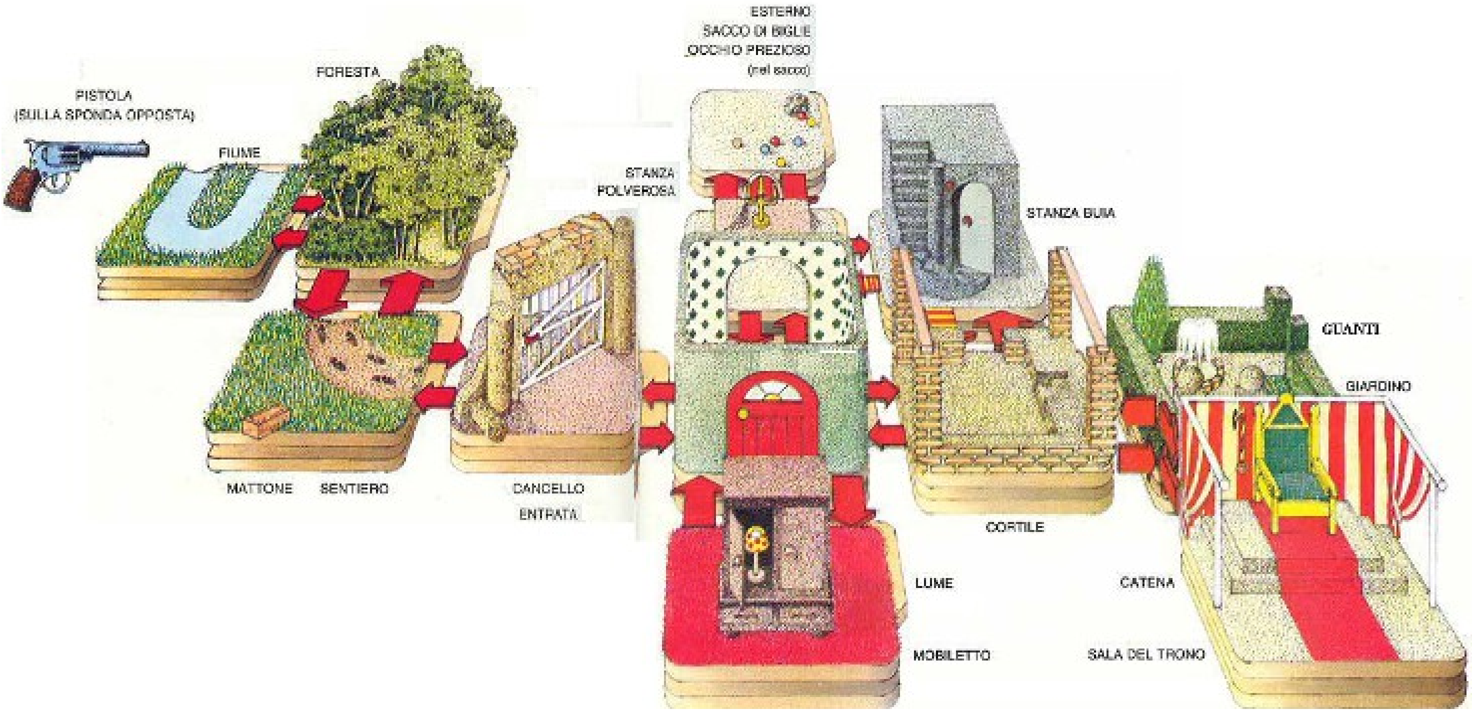
This adventure starts with the player in trouble due to a lot of debts with the revenue authorities. The only solution is to find the “eyeball” gem in the far hidden city.

Unfortunately, the tax-man is chasing the player. If the tax-man captures the player, he goes in prison and the adventure ends.

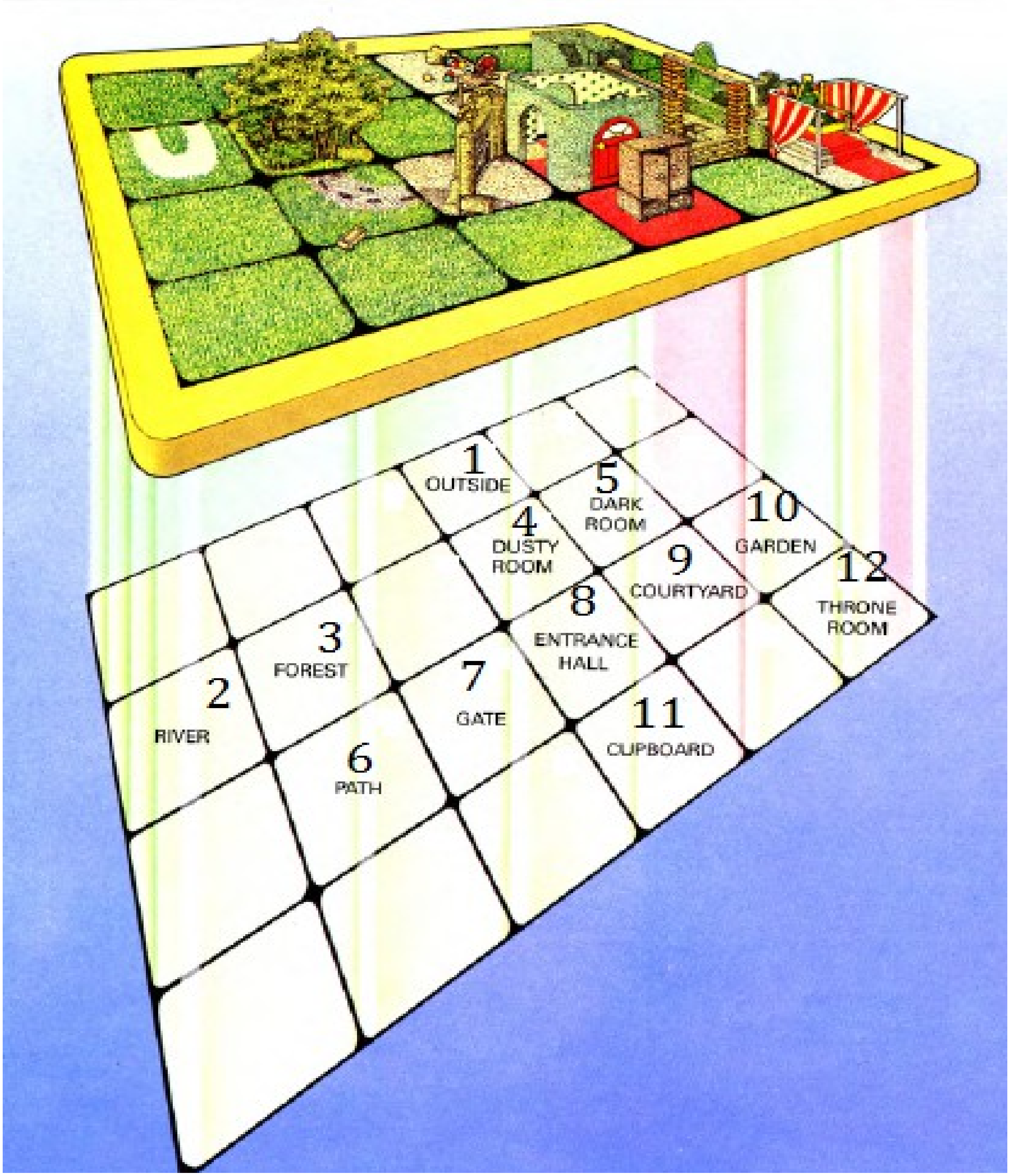
The player needs various objects to win. For example, a lamp is needed to exit from the dark room. Other object are traps. For example, the brick makes the player sink.

The eyeball is inside a bag full of marbles. To win, the player should be in the trone room, be seated and pull the chain while wearing the gloves. Otherwise the player will be electrocuted or splashed out (!)

Here is the world of the adventure, composed by 12 rooms and several objects.



Every room is indexed by a number.


1
OUTSIDE
5
DARK ROOM
4
DUSTY ROOM
10
GARDEN
9
COURTYARD
12
THRONE ROOM
8
ENTRANCE HALL
3
FOREST
2
RIVER
7
GATE
11
CUPBOARD
6
PATH

The code of Aliza

ATARI standard games uses 4096 bytes as ROM (code) and 128 bytes of RAM memory. Aliza game engine (and graphic kernel) uses 2446 bytes of ROM and 119 RAM bytes, so only 1650 bytes are available for our texts and code (the adventure logic) and only 9 integer variable (byte) from 0 to 255 in RAM.

Any adventure developed have these constraints:

- max 14 rooms
- max 15 floating objects, eventually takeable
- max 16 indexes to attach to text lines of the rooms descriptions
- 8 verbs
- 32 free messages

Texts can contain only these characters:

```
QWERTYUIOPASDFGHJKLZXCVBNM ,.;:!'-
```

Note: see appendix A for bug reported for texts

Text lines (and also floating objects names) are always of 12 characters.

There are no save options, when the adventures ends (the player win or fail), the game has to be reset.

Note: in some cases the Aliza compiler is able to optimize spaces in texts, but basically every character uses 1 byte of ROM. For details on how the spaces optimization works, see appendix D

Now, copy the “eyeball” folder (in the zip downloaded) in the “ALIZA” folder created on the desktop.

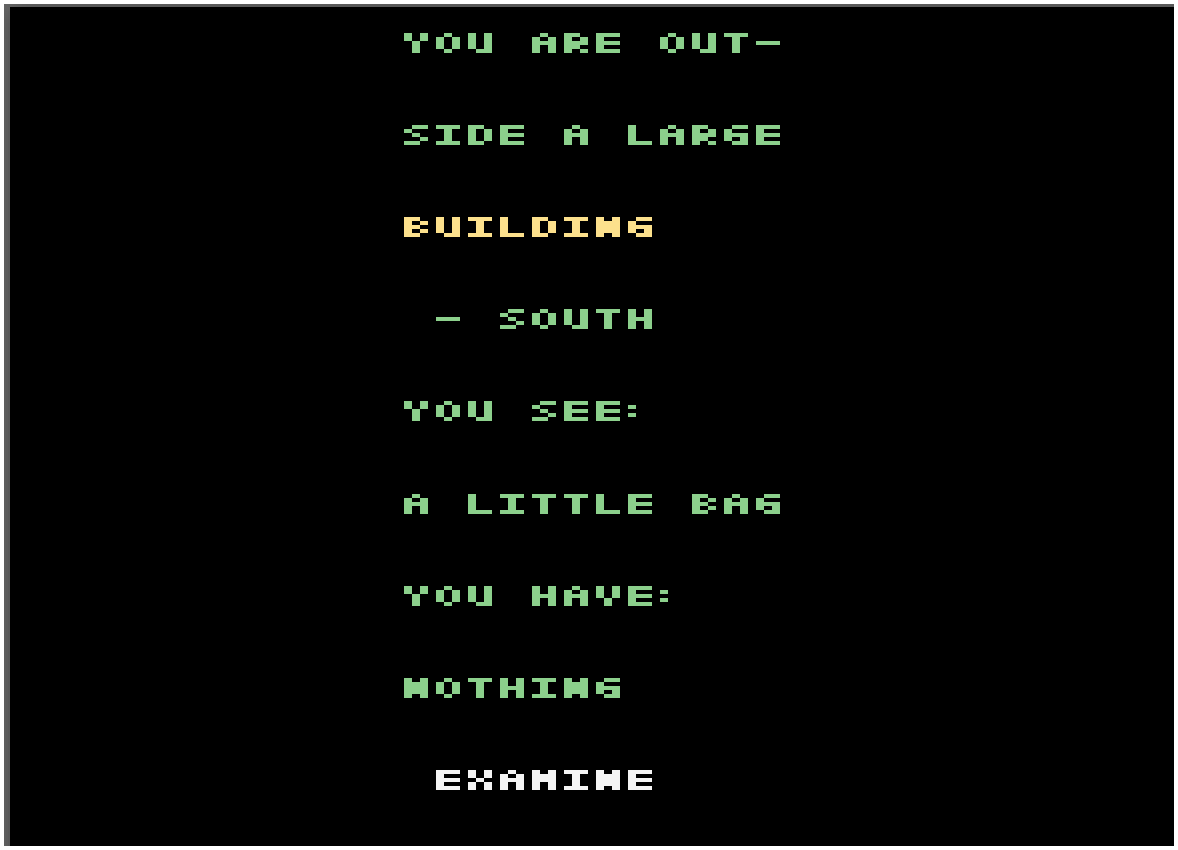
Open *code.alz* with *Notepad++* :

```
; ALIZA
; by E-Paper Adventures 2018  epaperadventures@gmail.com

; the eyeball - story based on marshall-cavendish INPUT computer course

; In the face of financial
; collapse you have fled
; the country. The solution
; to your problems lies
; in finding the fabled
; jewelled eyeball of
; the purple icon and passing
; the final initiative test.
; Avoid the tax inspector
; at all cost.

[COLOR1]
; text color
202

[COLOR2]
; selected row color
255

[COLOR3]
; player command color
15

[CODESIZE]
; code size
650


[START]
; initial room
07

[INVENTORY]
; max number of objects in inventory
3
```

The lines that starts with ; are comments lines, not considered to generate the assembly code.

Below [COLOR...] sections the colors (NTSC, decimal number) to use in the game are defined:

- color 1: text (default: green)
- color 2: text line selected (default: yellow)
- color 3: verb color in last line (default: white)



```
YOU ARE OUT-
SIDE A LARGE
BUILDING
 - SOUTH
YOU SEE:
A LITTLE BAG
YOU HAVE:
NOTHING
 EXAMINE
```

The color values are here:

http://www.randomterrain.com/atari-2600-memories-tia-color-charts.html

Below [CODESIZE] the number of bytes for the adventure code (logic) is written. In the example they are 650, this means that 1650-650 = 1000 bytes are reserved for the texts.

Below [START] there is the start room index (from 01 to 14)

Below [INVENTORY] there is the max number of items that the player can carry. Should be 8 at maximum.

Below [DEFAULTSTRINGS] there are the base messages, used by the game engine. Note: number of lines used for these messages must not be changed.

```
[DEFAULTSTRINGS]
;128
"IT DOES'NT  "
"SEEM TO WORK"
;129
```

```
"NOTHING     "
"INTERESTING "
;130
"YOU CANNOT  "
;131
"YOU HAVE TOO"
"MANY THINGS "
;132
"YOU HAVE:   "
;133
"YOU SEE:    "
;134
"NOTHING     "
;135
"DONE!       "
```

These messages could be used also in the code of the adventure as result of a player action (see *show* command).

Then we have the verbs and the directions names:

```
;-------------
; verbs
;-------------
; 1
" GO         "
; 2
" EXAMINE    "
; 3
" TAKE       "
; 4
" USE        "
; 5
" OPEN       "
; 6
" PUSH       "
; 7
" PULL       "
; 8
" DROP       "

;-------------
" CONTINUE..."
;-------------
;exits
```

```
" - NORTH    "
" - SOUTH    "
" - EAST     "
" - WEST     "
```

Verbs 5,6,7 can be changed. The others should be kept as they are.

“CONTINUE” is the text displayed after any message and require to hit fire button (or space in emulator).

Then there is for each room, the possible exits related to available directions. A number different by 00 is the destination room index.

```
[EXITS]
; room index:north room,south room,east room,west room
01:00,04,00,00
02:00,00,03,00
03:00,06,00,02
04:01,08,05,00
05:00,00,00,00
06:03,00,07,00
07:00,00,08,06
08:04,11,09,07
09:05,00,10,08
10:00,12,00,09
11:08,00,00,00
12:10,00,00,00

; not used
13:00,00,00,00
14:00,00,00,00
```

All the 14 rooms must be specified, even if not used.

In the game GO verb is used in combination with these directions. But GO can be used also with text lines in room description to add other possible directions.

Then we have the floating objects. These are the ones that eventually appear after room description or in the inventory.

```
[OBJECTS]
01:01,Y,"A LITTLE BAG"
02:06,Y,"A RED BRICK "
03:00,Y,"SOME GLOVES "
04:00,Y,"AN OLD GUN  "
05:00,Y,"THE EYEBALL "
06:00,Y,"SOME MARBLES"
07:00,Y,"A LAMP      "
```

```
; not used
08:00,N,"..........."

09:00,N,"THE TAX MAN!"

; not used
10:00,N,"..........."

11:00,N,"A BODY      "
12:00,N,"A DOOR:SOUTH"
13:00,N,"A DOOR:WEST "
14:12,N,"A CHAIN     "

; not used
15:00,N,"..........."
```

Always specify all 15 objects.

After index, there is the room where the object is (00=out of game, 15=in inventory), then if object is takeable (Y or N) and then the name of the object.

Then we have [TEXT] section that contains the free messages, from 01 to 32, that can be shown by *show* command in the code.

```
[TEXT]

01
"IT IS FULL  "
"OF MARBLES  "

02
"IT IS EMPTY "

03
"THE CHAIN IS"
"HANGING     "

04
"ONE OF THEM "
"IS THE      "
"JEWELLED    "
"EYEBALL!    "

05
```

```
"THE TAX     "
"INSPECTOR   "
"SUDDENLY    "
"APPEARS!    "

06
"AS YOU DID  "
"NOT HAVE    "
"MONEY, HE   "
"LOCKS YOU   "
"IN A DEEP   "
"DUNGEON...  "

07
"BANG! YOU   "
"KILLED HIM! "

08
"WHAT A SHAME"
"YOU DRAWNED!"

09
"YOU FIND A  "
"GUN ON THE  "
"OTHER SIDE  "
"OF THE RIVER"

10
"YOU GET WET "

11
"THE MARBLES "
"ROLL OVER   "
"THE FLOOR   "

12
"OH NO! HE   "
"DODGED IT!  "

13
"YOU GET     "
"FLUSHED DOWN"
```

```
"THE TOILET  "
"AND GO ROUND"
"THE BEND... "

14
"WELL DONE!  "
"YOU HAVE    "
"COMPLETED   "
"THE GAME!   "

15
"NOW YOU FEEL"
"LIKE A KING "

16
"BZZZZZ! YOU "
"GET SCHOKED."

17
"YOU FIND    "
"SOMETHING..."

18
"THERE IS A  "
"WRITING ON  "
"THE WALL:   "
"ALL YOU NEED"
"IS GLOVES   "

19
"AAARGH!!!   "
"YOU GET     "
"PETRIFIED!  "

20
"THE NIGHT IS"
"COMING...   "
"HURRY UP!   "

21
"IT IS TOO   "
"DARK TO     "
```

```
"CONTINUE... "
"YOU FAILED. "
```

Messages indexes must be progressive. Maximum lines for message is 8. As usual lines are always of 12 characters.

The next section [ROOMSTEXT] has the rooms descriptions (max 8 lines). Room indexes should be progressive, max 14. Some rooms (at the end of the list) can be not present:

```
[ROOMSTEXT]

01
"YOU ARE OUT-"
"SIDE A LARGE"
"BUILDING    ",18

02
"YOU ARE BY A"
"FAST FLOWING"
"RIVER       ",17

03
"YOU ARE IN A"
"PETRIFIED   "
"FOREST      "

04
"YOU ARE IN A"
"DUSTY ROOM  "

05
"YOU ARE IN A"
"DARK ROOM   "

06
"YOU ARE ON A"
"MUDDY PATH  "

07
"YOU ARE BY  "
"THE GATE OF "
"THE HIDDEN  "
"CITY        "
```

```
08
"YOU ARE IN  "
"THE ENTRANCE"
"HALL        "

09
"YOU ARE IN  "
"A COURTYARD "

10
"YOU ARE IN  "
"THE GARDEN  ",19

11
"YOU ARE IN  "
"THE CUPBOARD",20

12
"YOU ARE IN  "
"THE THRONE  ",21
"ROOM        "
```

Any line could “contain” an object to use to perform actions. We just have to put a “,” and an index from 17 to 32.

```
"BUILDING    ",18
```

So we can intercept later in code the action “EXAMINE” + line “BUILDING” for example.

If we use “>” instead of “,” we indicates that if the player select GO verb plus this line, the player will be automatically moved to the room of the index after “>”

```
02
"YOU ARE ON A"
"MUDDY PATH. "
"THERE IS A  "
"LITTLE HUT  ">03
```

If the player select “LITTLE HUT” and GO, it will be moved to room 3.

Some notes about the possible verbs.

GO: is the only verb usable with the directions NORTH, SOUTH, EAST, WEST created by the EXITS section. The GO + direction action cannot be intercepted in ACTIONS section. If we want to manage manually an exit, we have to put it as floating object or as object related to a text line.

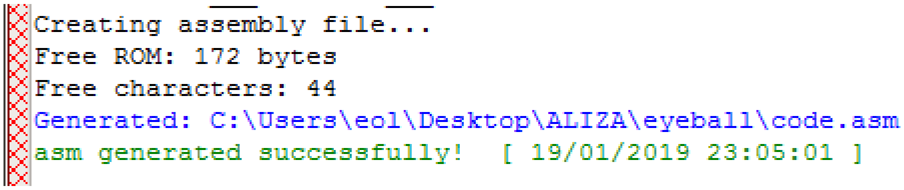
EXAMINE: if not intercepted, the engine will show “NOTHING INTERESTING” message.

TAKE/DROP: give the ability to manage the inventory automatically for the floating objects specified as takeable. Also the engine shows “YOU HAVE TOO MANY THINGS” in case the player has already the max number of items allowed. Or it shows “YOU CANNOT” in case the action is not permitted.

USE: if not intercepted, the engine shows “IT DOESN'T SEEM TO WORK”

PUSH, PULL,OPEN:if not intercepted, the engine shows “YOU CANNOT”

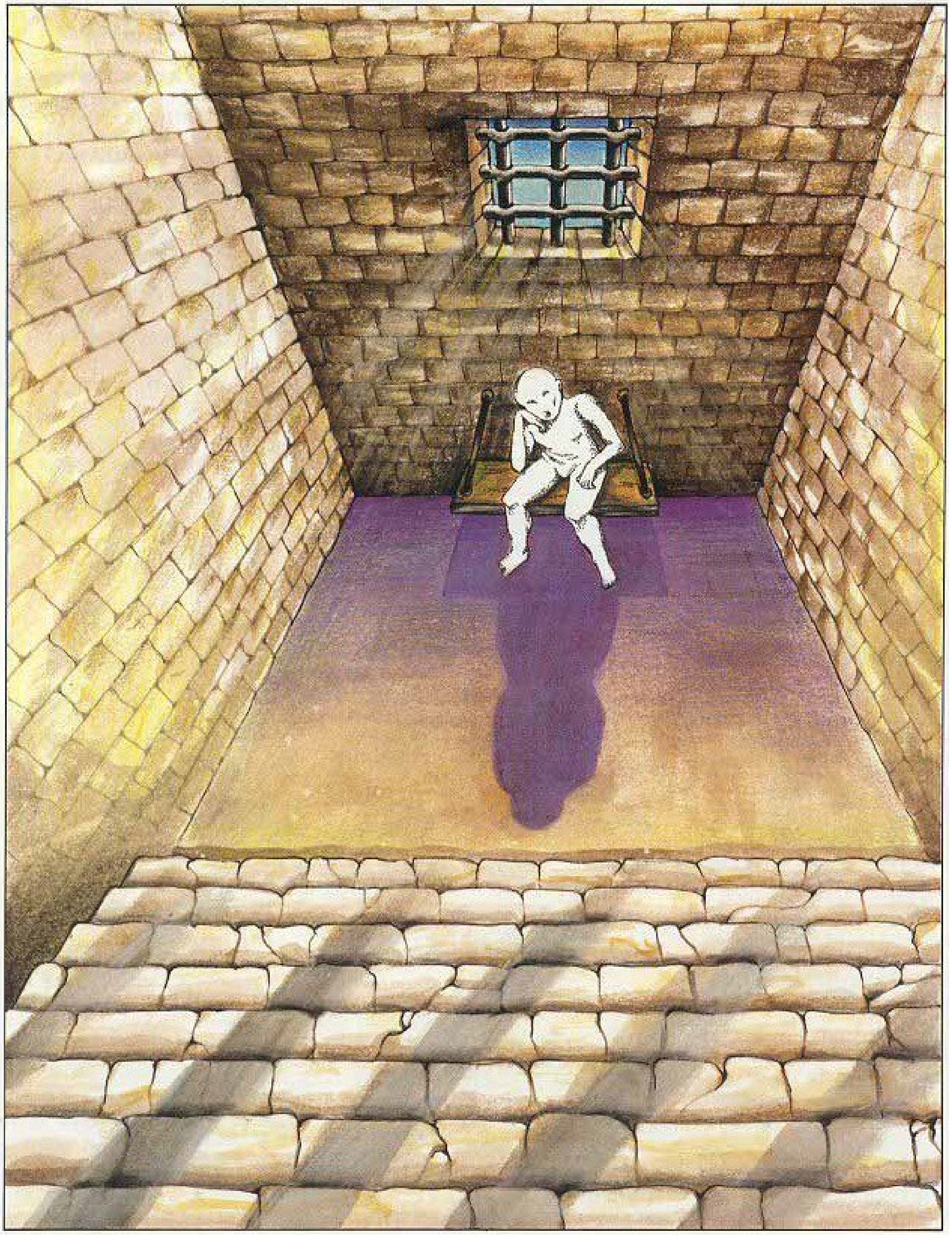
Before to look at [ACTIONS] and [CONDITIONS], where the adventure logic is coded, let's try to generate the assembly of “eyeball”.



```
Creating assembly file...
Free ROM: 172 bytes
Free characters: 44
Generated: C:\Users\eol\Desktop\ALIZA\eyeball\code.asm
asm generated successfully!  [ 19/01/2019 23:05:01 ]
```

As it is displayed, 172 bytes are still available for the code and 44 bytes are still available for texts.

Note: in case you need a bit of more ROM to complete your adventure, you can use the trick described in appendix C to obtain 60 extra bytes.



Variables

These are the variables available for managing the adventure in the code:

POS: current position (room) of player

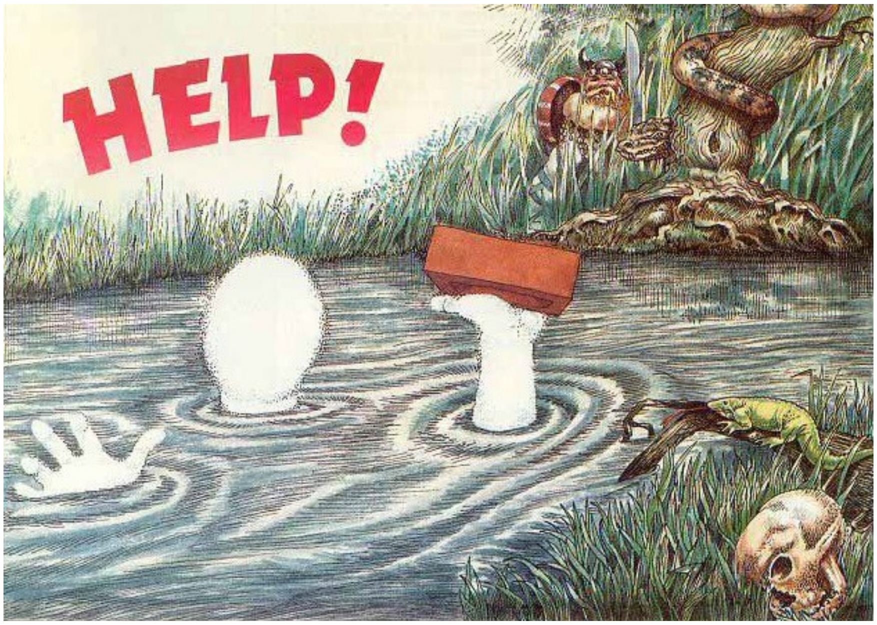
P01..P15: position of object of index 01..15. 00 = out of game, 15 = in inventory

V01..V09: free variable, from 0 to 255, at the game start all are set to 0

In “eyeball”, variables have this meaning:

```
; VARIABLES
; V01 = number of actions done
; V02 = is 1 if bag already examined
; V03 = is 1 if marbles already examined
; V04 = number of actions done in the forest
; V05 = is 1 if inspector already appeared
; V06 = is 1 if garden already examined
; V07 = is 1 if cupboard already moved
; V08 = is 1 if player is sitting on throne
; V09 = is 1 if player has seen the writings
```

V01..V09 can be managed as at bit level (8 bit for variable), but the related commands are very expensive in term of ROM code size and should be avoided.



ACTIONS and CONDITIONS code

These sections contain the code that implement the logic of our adventure.

Every time an action is performed by player, the engine execute the ACTION code. If the action is not intercepted in that code, the game engine manage automatically the action and shows the correct message (DONE, YOU CANNOT, ...)

After any action, and after evaluating the ACTION code, the CONDITIONS code is executed. This part of the code is used to count the actions done to make something happen or to simply display a message.

These are the instructions usable in the code, spaces and number of digits must be respected. Also, Aliza is case sensitive.

```
done
```

exit from section ACTIONS or CONDITIONS (the next instruction of sections are skipped)

```
show XXX
```

XXX = 001-032 or 128-135 (message index)

show the message. In section ACTIONS also force exiting from the section

```
end XXX
```

XXX = 001-032 or 128-135 (message index)

Show message and end the game (reset required). Note: if used in CONDITIONS, it has priority respect any message of a *show* instruction in ACTIONS

```
XXX=YYY
```

XXX is any variable

YYY is any variable or a number from 000 to 255

Assign YYY to XXX

For example:

POS=002

move player to room 2.

```
XXX++
```

Increment variable XXX by 1.

```
XXX--
```

Decrement variable XXX by 1.

```
XXX&=YYY
```

Assign (XXX and YYY) to XXX.

For example, if V01 is 3 (00000011)

V01&=002

makes V01 = (00000011) & (00000010) that is (00000010) = 2

```
XXX|=YYY
```

Assign (XXX or YYY) to XXX.

For example, if V01 is 3 (00000011)

V01|=004

makes V01 = (00000011) | (00000100) that is (00000111) = 7

```
if [test1]
&& [test2]    (optional)
...
&& [testN]    (optional)
 [...]
end if
```

[test1]...[testN] are comparison expressions that must be all true.

[…] is the code executed if all expressions are true (can contain others IF blocks)

[test?] can be:

XXX=YYY → XXX equal to YYY
XXX<YYY → XXX less than YYY
XXX>=YYY → XXX major than YYY
XXX<>YYY → XXX not equal to YYY
XXX&YYY=ZZZ → XXX and YYY equal to ZZZ (binary and)

with XXX variable and YYY,ZZZ variables or numbers from 000 to 255

For example:

```
          if POS=002
           V01++
          end if
```

if player is in room 2, increment V01 by 1.

```
          if POS=002
          && V02<005
           V01++
          end if
```

if player is in room 2 and V02 is less than 005, increment V01 by 1

The last command can be used only in ACTION to intercept the player actions.

```
action X,YY
 [...]
end action
```

X = verb index (1 = GO, 2 = EXAMINE, ...)

YY = object index 01..15 o 17..32

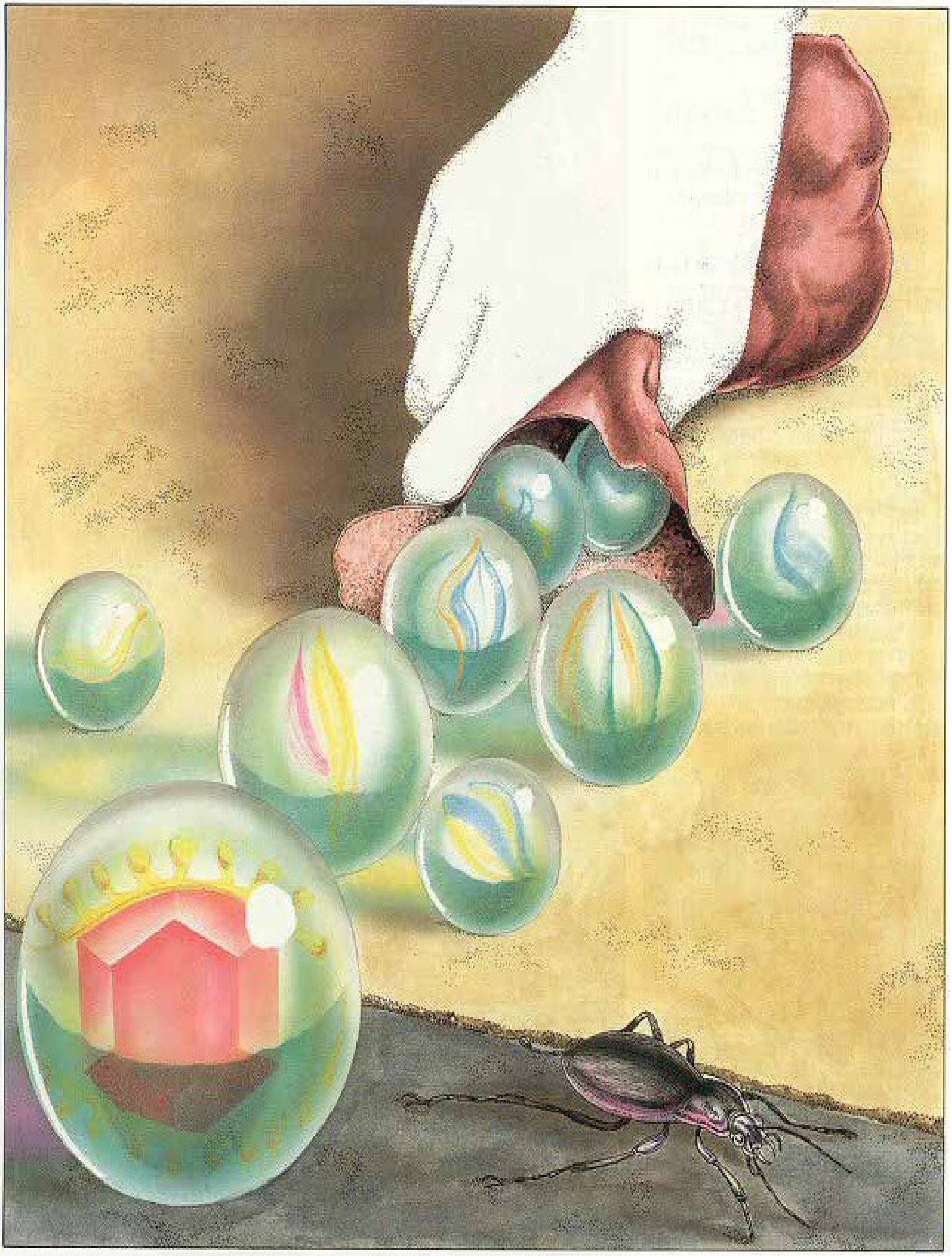
In case action is intercepted, code in […] is executed.

No action command can be nested in action command.

[…] can contain IF blocks.

To intercept any action made with a certain verb, use YY = 00

To intercept any action made with a certain object, use X = 0



ACTIONS and CONDITIONS

Now lets analyze CONDITIONS code of “eyeball”.

```
[CONDITIONS]


; after any action, increment V01 by 1 till 50
if V01<050
 V01++
end if


; at 25th action, warn player that night is coming (not in dark room)
if V01=025
&& POS<>005
 show 020
end if


; at 50th action, end game (not in dark room)
if V01=050
&& POS<>005
 end 021
end if
```

The first IF block checks if V01, used to count number of actions made by player, is less than 50. If so, V01 is incremented by 1.

The second IF block checks if V01 is equal to 25 and also if player is not in room 5: in that case message 20 is shown.

The third IF, in case V01 is 50 and the position is not room 5, ends the game with message 21.

```
; the dark room:
; if player has the lamp or the lamp is in the room
; show two object (12,13) as room exits
; south (go to 9), east (go to 4)
P12=000
P13=000
if P07=015
 P12=005
 P13=005
end if
if P07=005
 P12=005
 P13=005
end if
```

This block set to 0 (out of game) the floating objects 12 e 13 that are addtional exits that the player can see only if he has the lamp or the lamp is in the room 5. In that case the position of the exits become room 5.

```
; the forest
; if you stay for 3 moves, you get petrified
if POS=003
 V04++
 if V04=003
  end 019
 end if
end if
if POS<>003
 V04=000
end if
```

If player is in the forest (room 3) and stay for 3 actions, it will be petrified. The count is made with variable V04. When the player is in another room, the counter is reset to 0.

```
; inspector: if is appeared and still alive, after 1 move game over
if V05=001
&& P09=010
 end 006
end if

; inspector: appears in the garden
if V05=000
&& POS=010
 P09=010
 V05=001
 show 005
end if
```

If the tax-man (object 9) is already apperead (V05=1) and he is in room 10, the games is ended with message 6.

This block is related to the enemy of the player, the tax-man. The first IF checks if the taxman is already appeared (V05=1) and if its position is room 10(garden) and in that case it means that the action made by the player was not the “correct one” (USE GUN). So the game ends with message 6.

The second IF checks if the tax-man is not appeared (V05=0) and if the position of the player is the garden and in that case the tax-man is located in the same room, V05 is set to 1 and message 5 is shown.

```
; if not in the throne room, clear V08
if POS<>012
 V08=000
end if
```

This IF verifies if the player is in a room different from the throne (12) and in that case set V08 to 0. V08 is set to 1 when the player seats on the throne (see below)

Now let's look at the ACTIONS section, where we intercept all the player actions we need to manage manually.

```
[ACTIONS]

; exits of dark room
action 1,12
 POS=009
 done
end action
action 1,13
 POS=004
 done
end action
```

First action is GO + object 12 (“A DOOR:SOUTH") that is an additional exit in the dark room (5) if the player has the lamp. In case the player is moved to room 9 and the engine is forced to exit from the section ACTIONS (it has no sense to evaluate the other actions).

Then there is GO + object 13 (“A DOOR:WEST") that is the other additional exit, that moves the player to room 4.

```
; swimming in the river
action 1,17
 ; if player has the brick, he dies
 if P02=015
  end 008
 end if
 ; if first time (gun out of game), gun appears
 if P04=000
  P04=002
  show 009
 end if
 ; otherwise just a message
 show 010
end action
```

First if GO + RIVER and if player has the brick (P02=15) then the game ends with the message 8. Otherwise if the gun (object 4) is out of the game, then it appears in the room and message 9 is displayed. If gun was already appeared, just show message 10.

```
; writings on the building
action 2,18
 V09=001
 show 018
end action
```

If building (object 18) is examined (in room 1) then V09 is set t 1 and message 18 is shown.

```
; examine bag
action 2,01
 if V02=000
  show 001
 end if
 show 002
end action
```

If marbles bag is examined, and bag is not empty (V02=0) then message 1 is shown otherwise 2.

```
; examine garden
action 2,19
 if V06=000
 && V09=001
  V06=001
  P03=POS
  show 017
 end if
end action
```

If EXAMINE GARDEN (object 19) and it is the first time player examine it (V06=0) and player has read the writing on the building (room 1) then V06 is set to 1 and object 3(gloves) are positioned is the current player room. Then message 17 is shown.

Note that show is always the last command of the inner if code block, because always causes exiting from the ACTIONS section!

```
; open bag
action 5,01
 if V02=000
```

```
  V02=001
  P06=P01
  show 011
 end if
end action
```

If player opens the bag, and the bag is not already open (V02=0), V02 is set to 1, and the marbles are positioned in same room of the bag and then message 11 is shown.

```
; examine marbles
action 2,06
 if V03=000
  V03=001
  P05=POS
  show 004
 end if
end action
```

If marbles are examinated, for the first time (V03=0) then V03 is set to 1 and the gem appears and the message 4 is shown.

```
; examine chain
action 2,14
 show 003
end action
```

EXAMINE CHAIN → show message 3

```
; push cupboard
action 6,20
 if V07=000
  V07=001
  P07=POS
  show 017
 end if
end action
```

If PUSH THE CUPBOARD for the first time (V07=0) then V07 is set to 1 and the lamp appears in the same room of the player. Then message 17 is shown.

```
; use throne
action 4,21
 if V08=000
  V08=001
  show 015
 end if
end action
```

USE THRONE, if player is not seat (V08=0), makes the player seat (V08=1) and message 15 is shown.

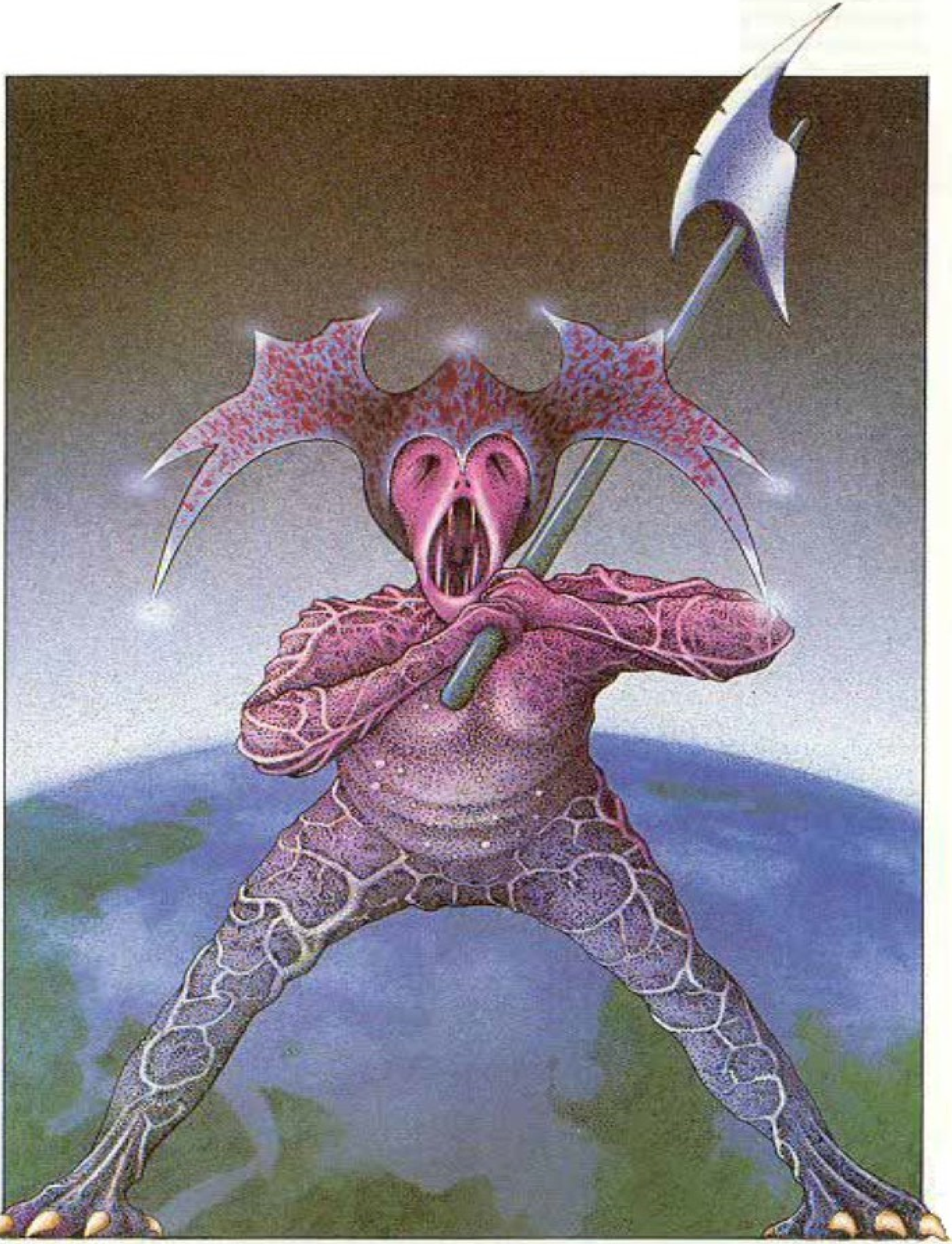
```
; use brick (when inspector)
action 4,02
 if P09=POS
  show 012
 end if
end action

; use gun (when inspector)
action 4,04
 if P09=POS
  P09=000
  P11=POS
  show 007
 end if
end action
```

If player is in the same room of tax-man, USE BRICK shows the message 12. USE GUN kills the tax-man, that will be put out of play (P09=000) and its body(11) appears in the same room of the player. Message 7 is shown.

```
; pull chain
action 7,14
 if P03=015
 && P05=015
 && V08=001
  end 014
 end if

 if P03<>015
  end 016
 end if
 end 013
end action
```

When player pulls the chain:

- if player has the gem, the gloves and is on the trone, the game ends and the player win
- if player hasn't the gloves it is electrocuted, in the other cases is splashed out



Debug

To debug the adventure, starting the game in a certain condition, the only way is to write in CONDITIONS something like this:

```
if V09=000
 POS=???
 P01=???
 ...
 V01=...
 ...
 V09=001
 done
end if
```

So doing, after any first action, the player will be moved in a certain location with certain variables set as we need to test certain parts of the game, at the cost of variable V09.

Here are some errors thar can be signalled during assembly generation.

Invalid character < ... >

Invalid character used in code.alz

Invalid or missing [...] section

Missing section in code.alz

File code.alz not found!

File code.alz not found

Invalid code size

Invalid code size (ACTIONS + CONDITIONS)

out of ROM memory

Text or code exceed ROM limits

invalid string: ...

Invalid text line (not 12 chars?)

Syntax error: ...

Invalid command used in ACTIONS or CONDITIONS

Invalid text sequence: ...

A message index is probably wrong or skipped

Too many characters.

Out of the ROM dedicated to texts

Too many strings.

Too lines for a message

Syntax error: unclosed action ...

“action” command not closed with “end action”

Syntax error: action not open ...

“end action” without “action”

Syntax error: if not open ...

“end if” without “if”

Syntax error: unclosed if

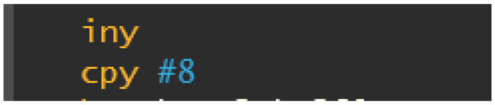
“if” without “end if”

Unknown variable: …

invalid variabile

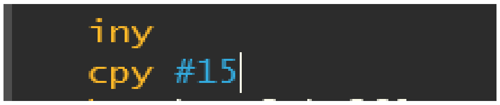
Appendix A: Bugs of version 1.0

(1) There is a bugged line in the assembly code generated by Aliza in the *code.asm* file. It is needed to replace a *cpy* instruction as below. To find it, just search the *cpy #8* after an *iny* instruction. Change:



```
    iny
    cpy #8
```

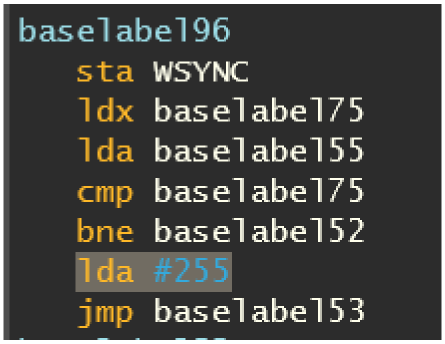
to



```
    iny
    cpy #15
```

(2) The color of the selected line is always yellow (value after section COLOR2 has no effect). To solve this:

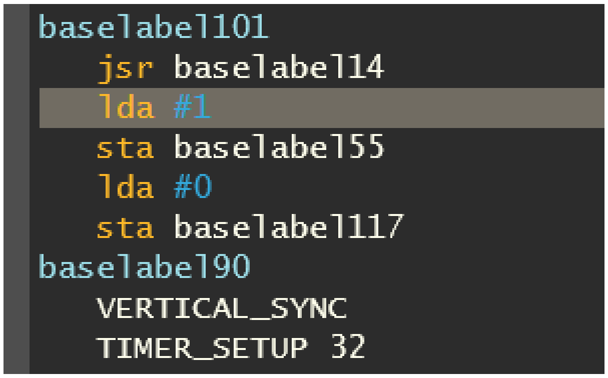
- Search an instruction *lda #255* that is in a code block starting with *sta WSYNC*



```
baselabel96
    sta WSYNC
    ldx baselabel75
    lda baselabel55
    cmp baselabel75
    bne baselabel52
    lda #255
    jmp baselabel53
```

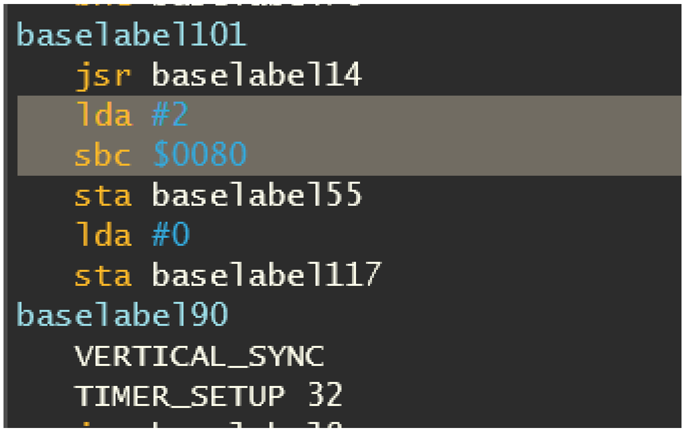
- modify the 255 with the color value you want for the text selected color

(3) The first line of a message displayed with *show* or *end* is in COLOR2 color instead of COLOR1. In case you want to have all the message lines in COLOR1, you need to change that *lda #1* line in the asm code (look for *TIMER_SETUP 32*, the line is just before). Change:



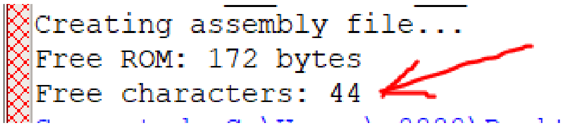
```
baselabel101
    jsr baselabel14
    lda #1
    sta baselabel55
    lda #0
    sta baselabel117
baselabel90
    VERTICAL_SYNC
    TIMER_SETUP 32
```

to:



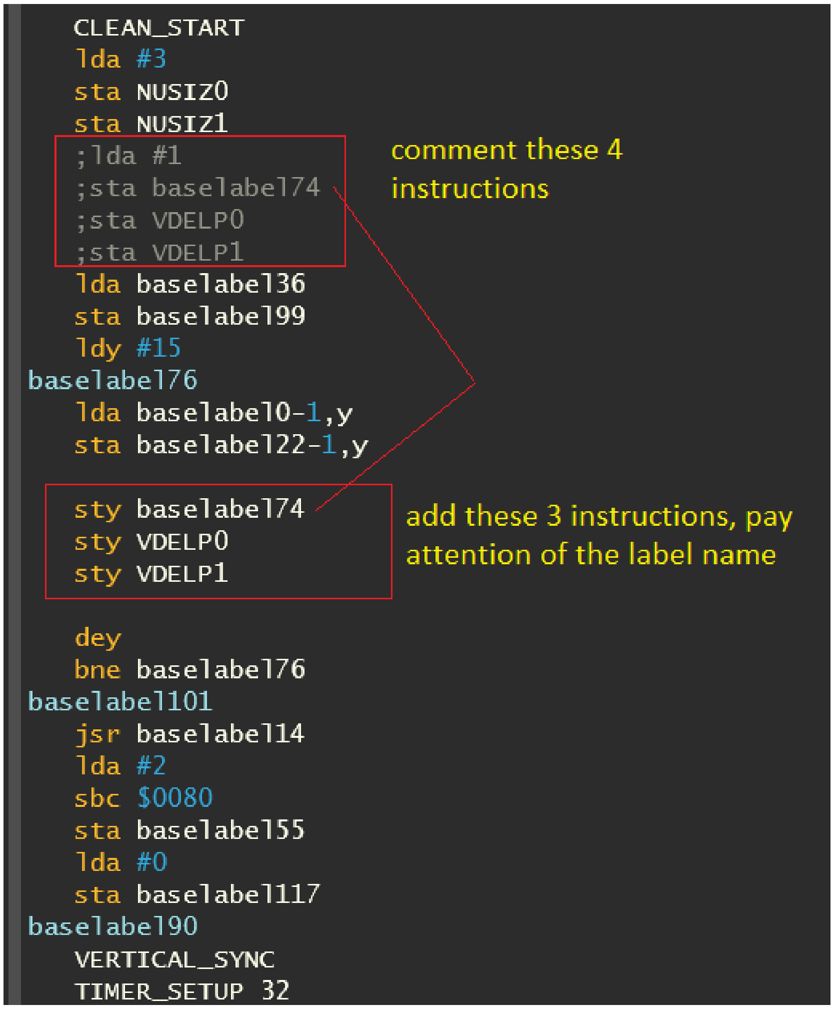
```
baselabel101
    jsr baselabel14
    lda #2
    sbc $0080
    sta baselabel55
    lda #0
    sta baselabel117
baselabel90
    VERTICAL_SYNC
    TIMER_SETUP 32
```

Warning: This change causes that the bytes available for your code and text are no longer 1650 but 1647. This means that when you have compiled your code, you should pay attention to see that at least 3 bytes are still free for the text, otherwise your asm will exceed 4K.



```
Creating assembly file...
Free ROM: 172 bytes
Free characters: 44
```

In case you want to avoid this problem, you can do a little opitimazion of some lines of code (again just before *TIMER_SETUP 32)* . You need to comment 4 lines and add 3 instructions as in the following figure. Doing this, you will have again 1650 bytes available.



```
   CLEAN_START
   lda #3
   sta NUSIZ0
   sta NUSIZ1
   ;lda #1
   ;sta baselabel74
   ;sta VDELP0
   ;sta VDELP1
   lda baselabel36
   sta baselabel99
   ldy #15
baselabel76
   lda baselabel0-1,y
   sta baselabel22-1,y

   sty baselabel74
   sty VDELP0
   sty VDELP1

   dey
   bne baselabel76
baselabel101
   jsr baselabel14
   lda #2
   sbc $0080
   sta baselabel55
   lda #0
   sta baselabel117
baselabel90
   VERTICAL_SYNC
   TIMER_SETUP 32
```

comment these 4 instructions

add these 3 instructions, pay attention of the label name

(4) The character semicolon ";" cannot be used in any text string due to a bug in the compiler.

Appendix B: last version of quick reference file

```
ALIZA reference file v1.4
by E-Paper Adventures 2022  epaperadventures@gmail.com   http://www.epaperadventures.qlmagic.com/

--------
* TEXT *
--------
All the text strings must be of 12 characters.

Texts can only contain these characters:
QWERTYUIOPASDFGHJKLZXCVBNM ,.;:!'-

You can add a comment starting line with ;


-------------
* VARIABLES *
-------------
These are the variables available in CONDITIONS and ACTIONS sections

V01..V09
 free variables, values from 0 to 255

P01..P15
 position of floating object
 0 = out of game
 1..14 = in room 1..14
 15 = taken

POS
 position of player (room) 1..14


---------------------
* SECTIONS COMMANDS *
---------------------
This is the list of commands that you can use
in sections ACTIONS and CONDITIONS.

done
 to exit from CONDITIONS or ACTIONS section
```

```
show XXX
 XXX = 001-032 or 128-135
 show message of index XXX and if in ACTIONS exit from section

end XXX
 XXX = 001-032 or 128-135
 show message of index XXX and end the game waiting for reset

XXX=YYY
 XXX is a variable
 YYY is a variable or a number from 000 to 255
 Assign value YYY to XXX

XXX++
 Increment by 1 the variable XXX. Note: if XXX=255, after XXX++ the value in XXX will be 0

XXX--
 Decrement by 1 the variable XXX. Note: if XXX=0, after XXX-- the value in XXX will be 255

XXX&=YYY
 XXX is a variable
 YYY is a variable or a number from 000 to 255
 Assign to XXX the value given by (XXX & YYY) with & the binary AND


XXX|=YYY
 XXX is a variable
 YYY is a variable or a number from 000 to 255
 Assign to XXX the value given by (XXX | YYY) with | the binary OR


if [test1]
&& [test2]    (optional)
...
&& [testN]    (optional)
 [...]
end if

 if [test1], [test2] ... [testN] are all true, [...] is executed.
 Note: [...] can contain nested if ... end if commands

 [test?] could be:
    XXX=YYY       ---> XXX equal to YYY

    XXX<YYY       ---> XXX less than YYY

    XXX>=YYY      ---> XXX equal or greater than YYY

    XXX<>YYY      ---> XXX not equal to YYY

    XXX&YYY=ZZZ   ---> (& is binary AND) XXX AND YYY equal to ZZZ


 with XXX a variable and YYY,ZZZ a variable or a number from 000 to 255



action X,YY

 [...]

end action


 X = index of the selected action 1..8

 YY = object selected 01..15 or 17..32

 Execute [...] if player does the action

 Note: cannot contain nested action ... end action

 Note: use it only in ACTIONS section

 Note: if X = 0, you capture any action on object YY

 Note: if YY = 00, you capture any action of index X
```

Appendix C: adding 60 bytes to the available ROM

In case you are out of memory and you need extra bytes for your adventure, you can do this trick to obtain 60 extra bytes:

1) Individuate two messages of your TEXT section; the first must be made of 2 lines, the second of 3 lines. Let's say for example that their are messages 4 and 21.

2) Change the messages 128-135 of DEFAULTSTRINGS section to these:

```
;128
"XXXXXXXXXXXX"
"XXXXXXXXXXXX"
;129
"XXXXXXXXXXXX"
"XXXXXXXXXXXX"
"XXXXXXXXXXXX"
;130
"YOU CANNOT. "
;131
"TOO THINGS. "
;132
"YOU HAVE:   "
;133
"YOU SEE:    "
;134
"NOTHING.    "
;135
"DONE!       "
```

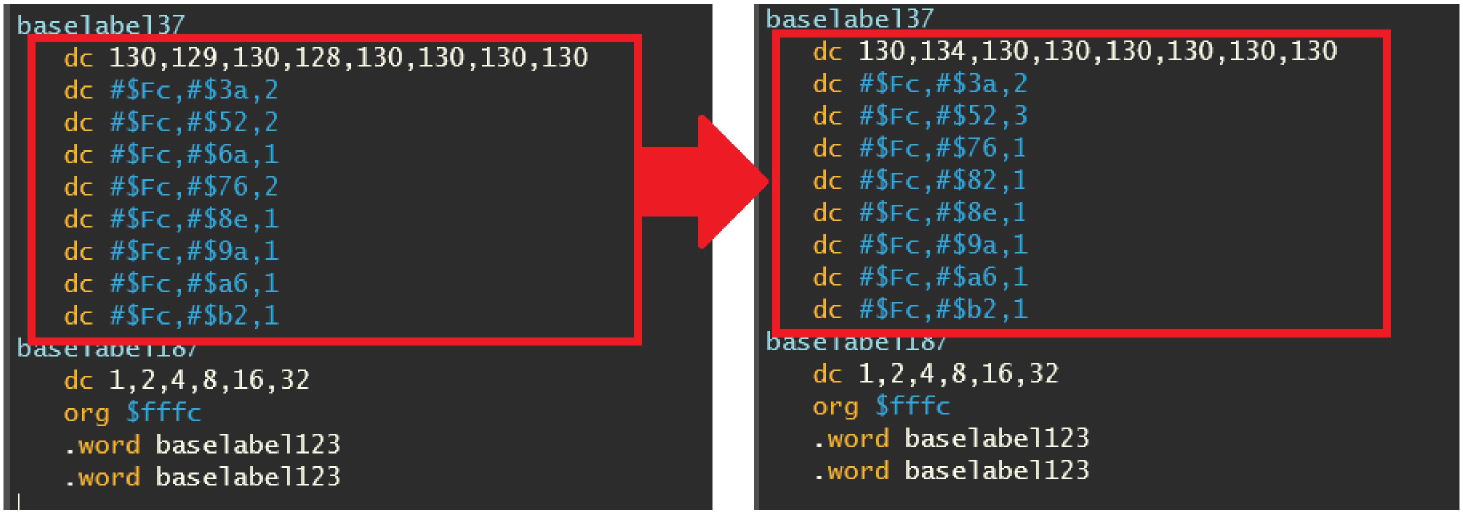
   and replace message 128 lines with the first selected message (4) and message 129 with the second selected message (21)

3) In your CONDITIONS and ACTIONS section, replace all *show* and *end* commands with 128 and 129 to 130 and 134 respectively. For example, *show 128* becomes *show 130*

4) In your CONDITIONS and ACTIONS section, replace all *show* and *end* commands with 004 and 021 to 128 and 129 respectively. For example, *show 021* becomes *show 129*

5) In TEXT section, copy your last two messages, for example 25 and 26, to the position 4 and 21. Then delete messages 25 and 26. In your CONDITIONS and ACTIONS section, replace all *show* and *end* commands with 025 and 026 to 004 and 021 respectively. For example, *show 025* becomes *show 004*

6) Generate the asm file, and in the asm code replace these lines in the red box at the end of the code:



```
baselabel37
   dc 130,129,130,128,130,130,130,130
   dc #$Fc,#$3a,2
   dc #$Fc,#$52,2
   dc #$Fc,#$6a,1
   dc #$Fc,#$76,2
   dc #$Fc,#$8e,1
   dc #$Fc,#$9a,1
   dc #$Fc,#$a6,1
   dc #$Fc,#$b2,1
baselabel187
   dc 1,2,4,8,16,32
   org $fffc
   .word baselabel123
   .word baselabel123
```

```
baselabel37
   dc 130,134,130,130,130,130,130,130
   dc #$Fc,#$3a,2
   dc #$Fc,#$52,3
   dc #$Fc,#$76,1
   dc #$Fc,#$82,1
   dc #$Fc,#$8e,1
   dc #$Fc,#$9a,1
   dc #$Fc,#$a6,1
   dc #$Fc,#$b2,1
baselabel187
   dc 1,2,4,8,16,32
   org $fffc
   .word baselabel123
   .word baselabel123
```

Pay attention to replace all the numbers correctly.

The new code is redirecting the default messages for examine and use verbs to “NOTHING.” and “YOU CANNOT.”, freeing the messages 128 and 129 for us. Being in total 5 lines of 12 characters, this means 60 bytes more for our code or text.

Note that all default messages (130-135) must be made by only 1 line.

Appendix D: text spaces optimization

Aliza makes some optimization on the text lines in order to reduce the bytes used.

A lines normally occupies 12 bytes.

In the following cases, only for lines in TEXT section, the number of used bytes is different:

- if the last 2 characters of the line are spaces, the number of used bytes will be 10
- if the last 4 characters of the line are spaces, the number of used bytes will be 8
- if the last 6 characters of the line are spaces, the number of used bytes will be 6

For example:

```
"xxxxxxxxxxx "
```

(11 *X* + 1 space) uses 12 bytes

```
"xxxxxxxxxx  "
```

(10 *X* + 2 space) uses 10 bytes

```
"xxxxxxx     "
```

(7 *X* + 5 spaces) uses 8 bytes

```
"xxxxxx      "
```

(6 *X* + 6 spaces) uses 6 bytes

```
"xxx         "
```

(3 *X* + 9 spaces) uses 6 bytes