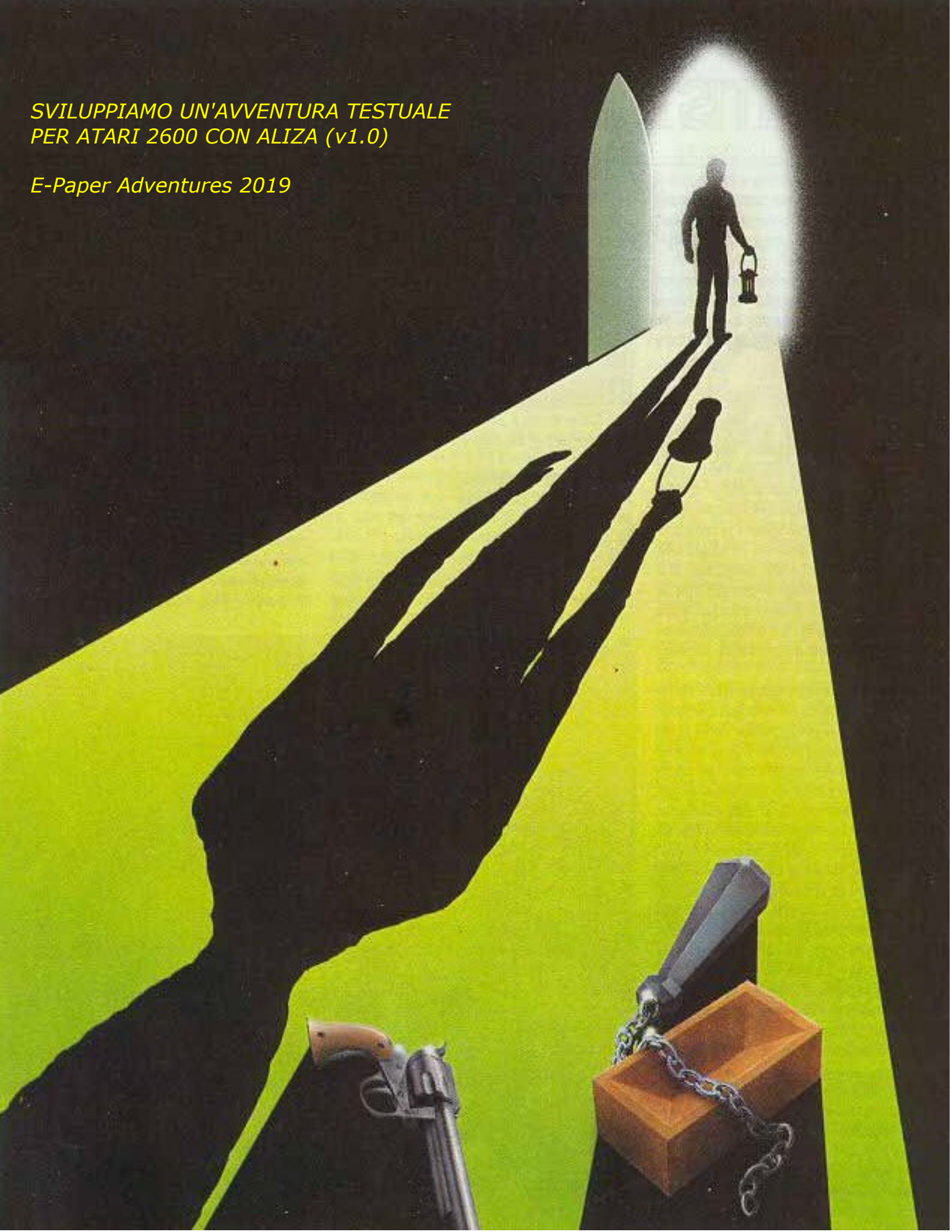
SVILUPPIAMO UN'AVVENTURA TESTUALE PER ATARI 2600 CON ALIZA (v1.0)

E-Paper Adventures 2019



Benvenuti!

Aliza è un sistema di sviluppo freeware per creare avventure testuali su ATARI 2600. L'applicativo per lo sviluppo può essere eseguito su qualsiasi versione di Windows e non richiede installazione.

In questo tutorial verrà realizzata una avventura testuale in inglese "The eyeball", il codice è disponibile sul sito www.epaperadventures.qlmagic.com

Questo progetto è completamente FREE. Se individuate delle possibili modifiche o migliorie (o errori!) potete scriverci all'indirizzo email: epaperadventures@gmail.com ...grazie! :)

Le immagini e alcuni testi di questo tutorial sono presi da alcune riviste degli anni 1984 e 1985 della serie "INPUT: Corso pratico di programmazione per lavorare e divertirsi col computer" dell'Istituto Geografico De Agostini (versione inglese: Marshall-Cavendish). Il materiale è assolutamente obsoleto ma qualora vi fossero dei problemi di copyright contattateci per segnalarcelo (ricordiamo comunque che questa iniziativa è senza scopo di lucro). Inoltre ci teniamo a precisare che questo lavoro è stato ispirato dalla lettura del (bellissimo) libro "Making Games for the Atari 2600" di Steven Hugg.

Il risultato finale dello sviluppo di un progetto in Aliza è **un file con codice assembly**. Tuttavia non è assolutamente necessario conoscere nessun linguaggio di programmazione: i costrutti da utilizzare con Aliza sono molto semplici e nel tutorial troverete numerosi esempi.

Per lo sviluppo di proprie avventure si consiglia l'utilizzo di Aliza in combinazione con Notepad++, il browser Mozilla Firefox (versione 58.0.1 o successiva). In questo tutorial si suppone che tutti questi programmi siano disponibili e installati.

Nel caso che questa iniziativa vi sembri meritevole di una donazione, potete supportare l'associazione:

RICOMINCIO DA CANE ONLUS

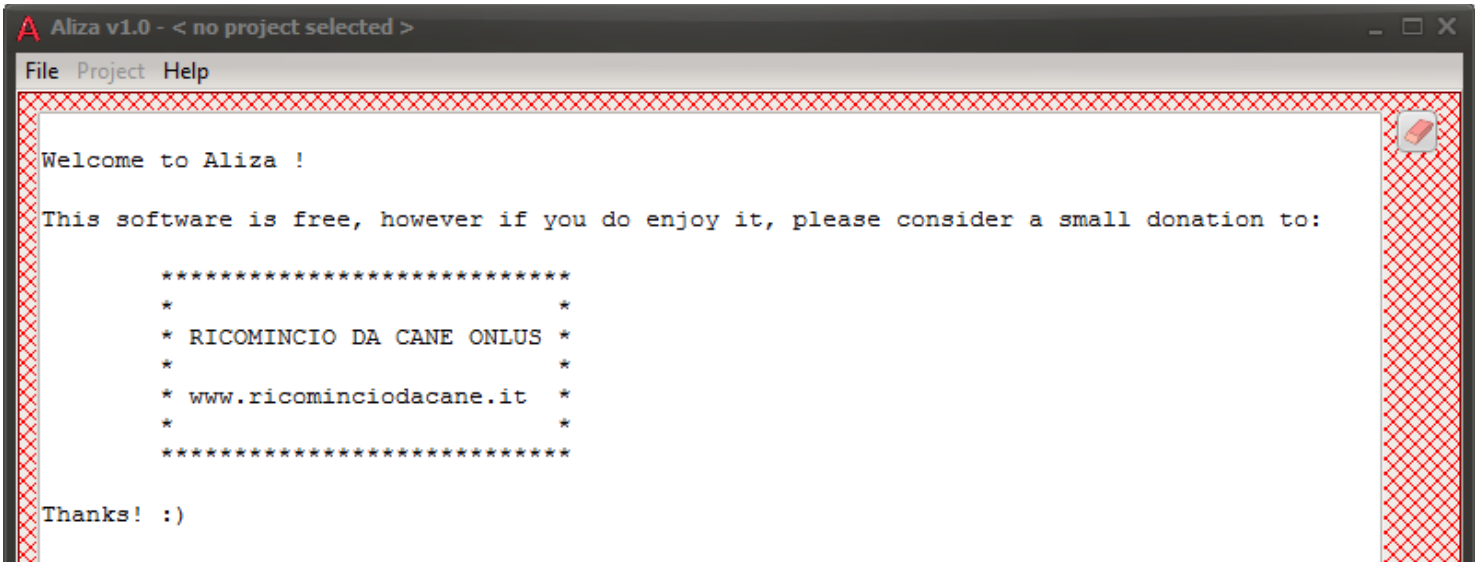
www.ricominciodacane.it

Grazie da tutto lo staff di E-Paper Adventures (Michele, Liviano, Lorenzo, Silvia, Christian e Davide).

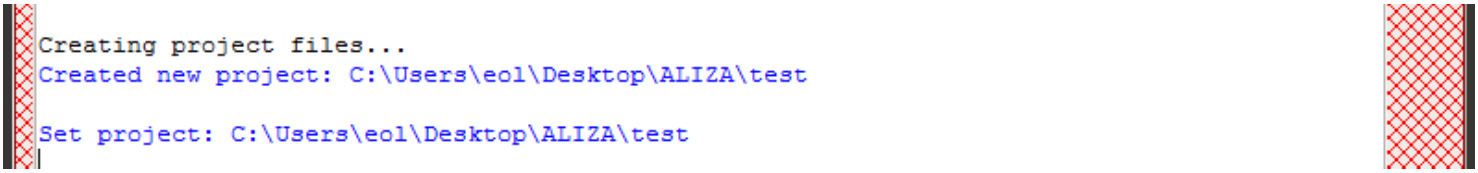
Revisione A.1 – Gennaio 2019

Creiamo un nuovo progetto!

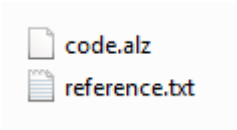
Per cominciare, estraiamo dallo zip e copiamo l'applicazione *aliza.exe* in una nuova cartella sul desktop (ad esempio: *ALIZA*) ed eseguiamola.



Andiamo nel menu *File* e creiamo un nuovo progetto scegliendo *New project*. Digitiamo come nome "test". Aliza creerà una sotto-cartella *test* con dentro diversi file. Oltre a questo, verrà selezionato come progetto corrente proprio "test".

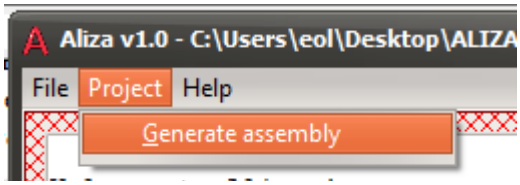


Esaminiamo i file contenuti nella cartella *test*:



Qualora le estensioni dei file non fossero visibili (".alz", ".txt") conviene modificare le opzioni di Windows per renderle tali. Per farlo cercate nelle impostazioni la voce "opzioni cartella" o "opzioni esplora risorse": nel tab "visualizzazione" cercate la voce "nascondi le estensioni per i tipi di file conosciuti" e disattivatela.

Ogni nuovo progetto creato contiene un'avventura di default. Andiamo sul menu *Project* e scegliamo *Generate assembly:*



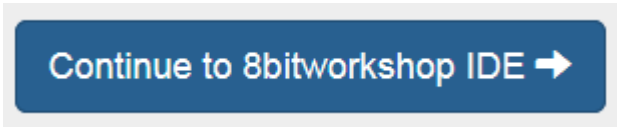
Aliza analizzerà il codice contenuto in *code.alz* nella cartella *test* e il risultato sarà la generazione del file *code.asm*

```
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 ]
```

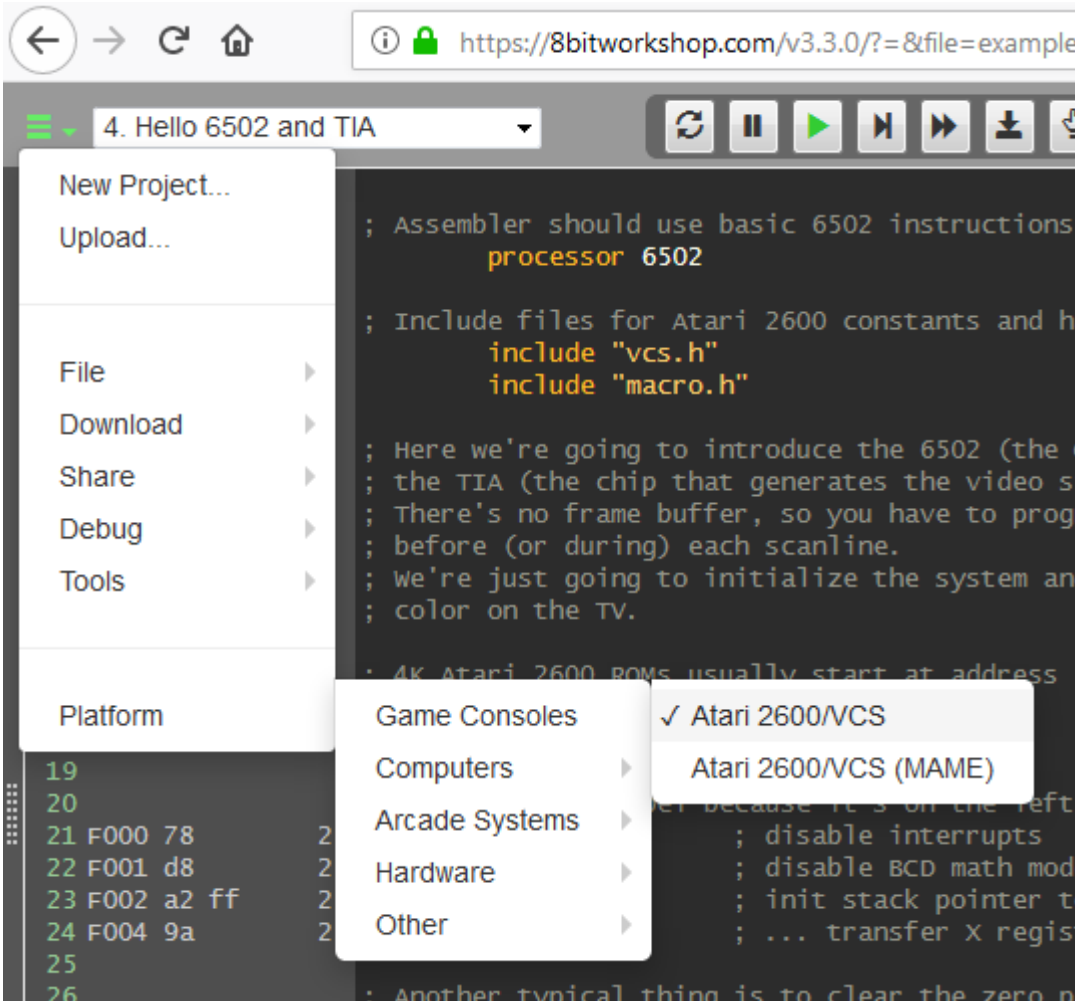
Apriamo *code.asm* con *Notepad++*. Vedremo un codice assembly. Copiamolo e andiamo sul sito web:

https://8bitworkshop.com/

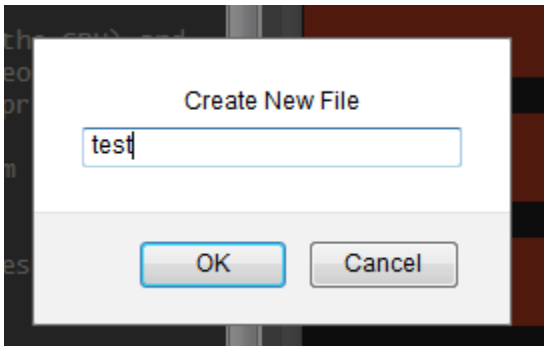
Clicchiamo su



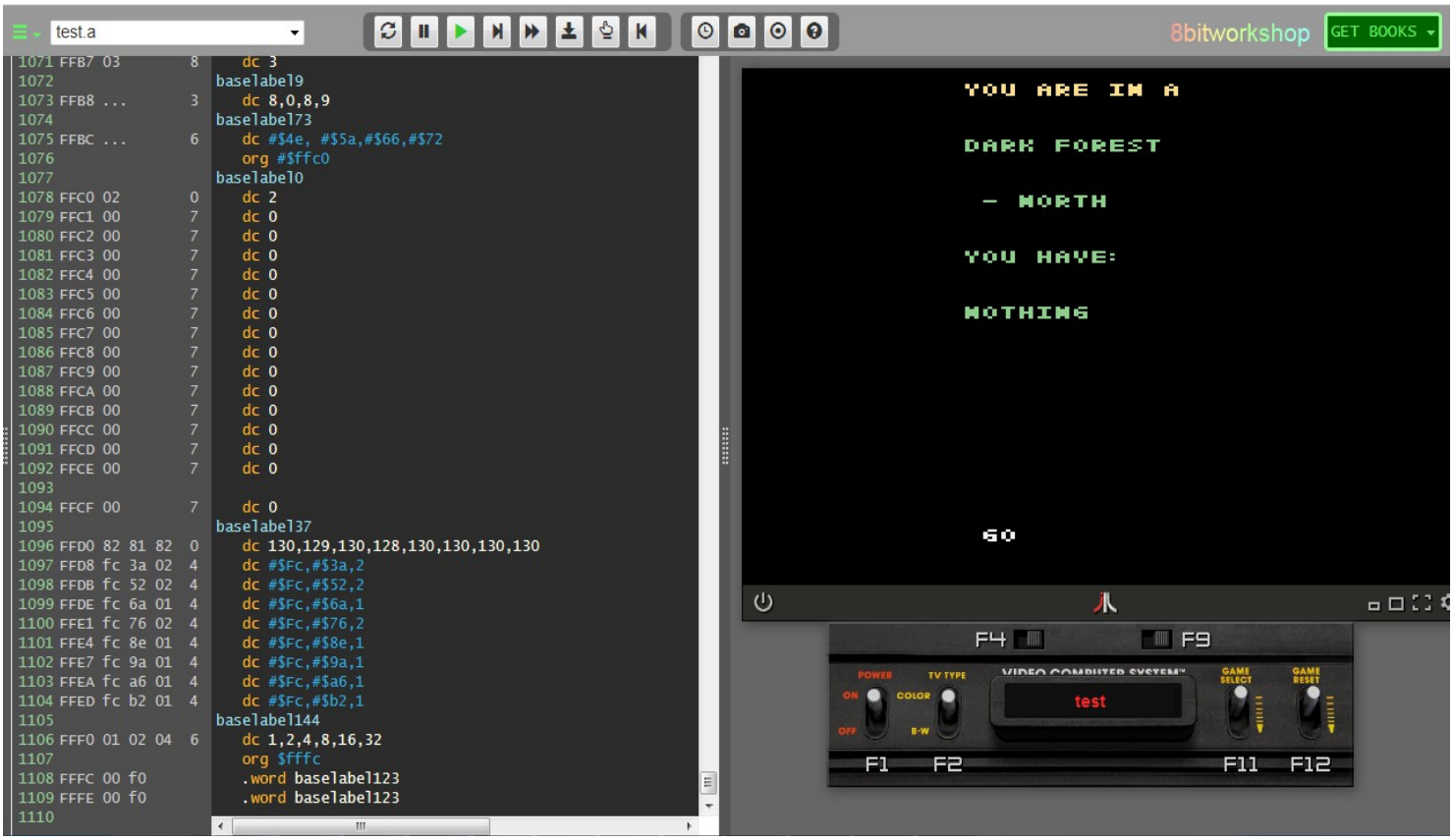
Una volta che la pagina web è caricata, clicchiamo in alto a sinistra e scegliamo come piattaforma ATARI 2600



Poi scegliamo *"New project"* e nella finestra che compare digitiamo *"test"* e invio.



A questo punto clicchiamo col tasto destro in mezzo al codice assembly, premiamo CTRL+A per selezionare tutto e poi CTRL+V per incollare l'assembly generato da Aliza:



Il nostro assembly verrà eseguito e potremo giocare all'avventura di test nel riquadro in alto a destra. Se ci clicchiamo dentro, potremo giocare in questo modo:

- cursore sinistro e destro (joystick giocatore 1 a sinistra e destra) per cambiare l'azione da compiere: GO (vai), EXAMINE (esamina), TAKE (prendi), USE (usa), OPEN (apri), PUSH (premi), PULL (tira), DROP (lascia)
- cursore su o giù (joystick giocatore 1 su o giù) per selezionare una riga della attuale schermata (diventa gialla)
- spazio (sparo) per eseguire l'azione (ad esempio: EXAMINE DARK FOREST)

- la levetta in basso a destra “GAME RESET” simula il reset della console, e fa ripartire l'avventura dall'inizio

Fondamentalmente, ogni schermata è composta da:

- descrizione della stanza
- direzioni possibili (nord, sud, est, ovest)
- eventuali oggetti ulteriori nella stanza
- inventario
- azione che si vuole eseguire (in bianco)

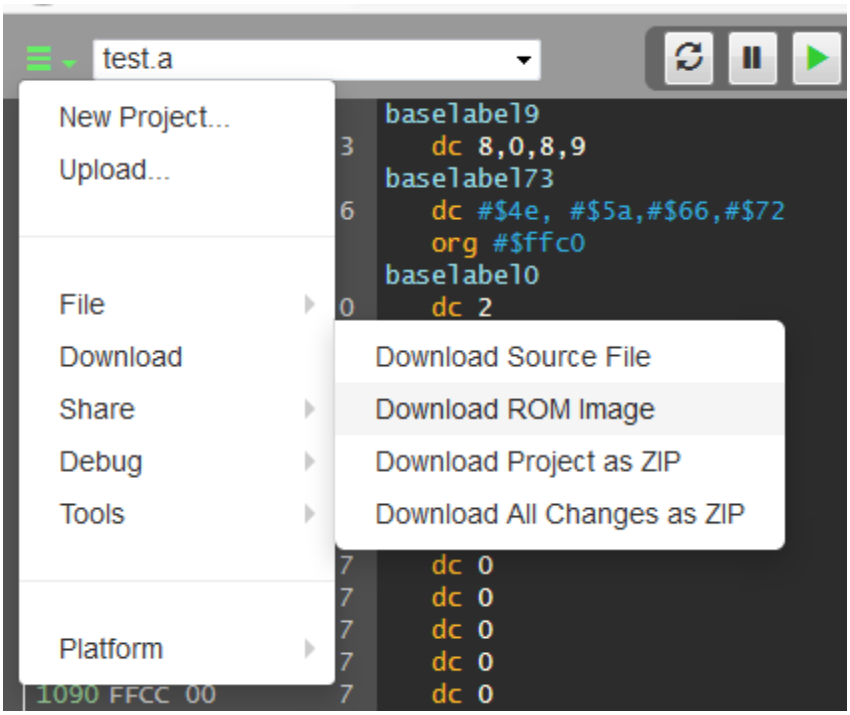
Può capitare che il numero di linee di testo siano più di quelle a disposizione nello schermo: in questo caso andando in giù oltre l'ultima riga si salta alla “pagina” di testo successiva (da questo, andando in su, si ritorna alla prima pagina).

Le varie azioni possono generare come risultato un messaggio, ad esempio “YOU CANNOT” (non puoi). In questo caso, l'unica azione possibile e premere spazio (sparo) per ritornare al punto dove eravamo.

Possiamo anche creare un file binario da utilizzare con l'emulatore STELLA, che trovate completamente free in Internet:

https://stella-emu.github.io/

Sempre in alto a sinistra, selezioniamo “download ROM image”:

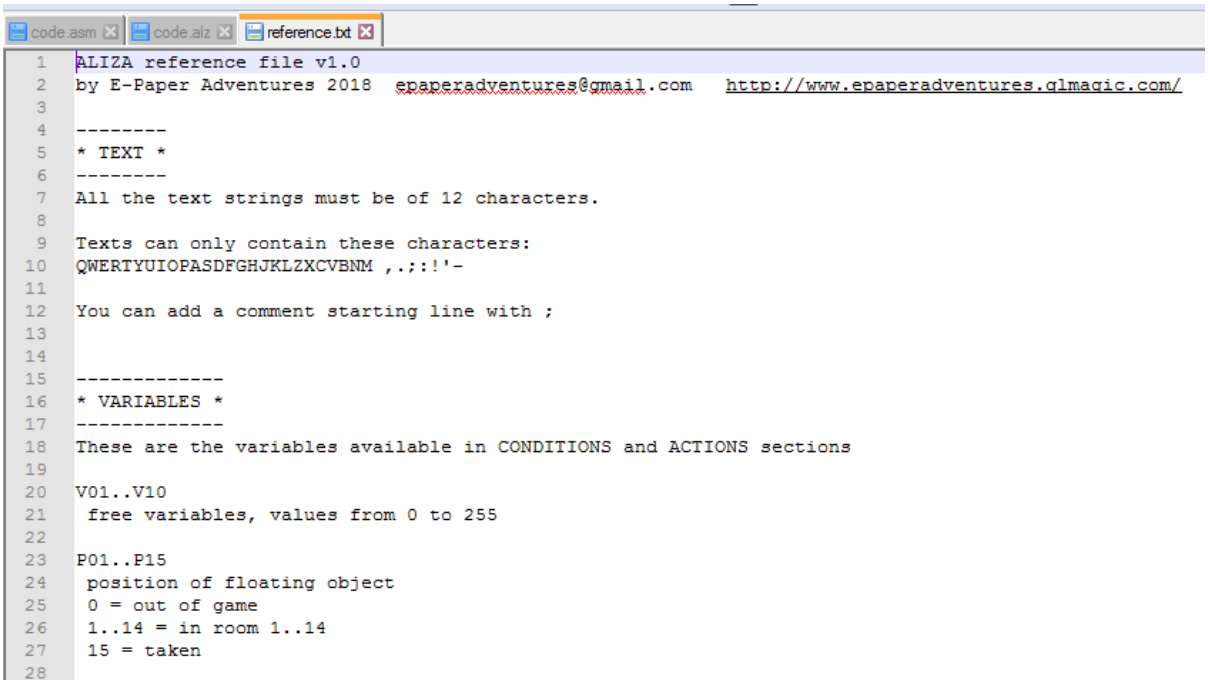


Il file ROM che verrà creato può essere usato con STELLA per giocare la nostra avventura.

Infine, esiste un sito che permette di creare cartucce reali per ATARI 2600, qualora voleste realizzare un videogame vero e proprio (e anche venderlo!) :

https://atariage.com/store/index.php?l=product_detail&p=949

Vedremo adesso come funziona un'avventura testuale creata con Aliza e come si scrive il codice per crearne una. Una nota: tutte le volte che creiamo un progetto, verrà aggiunto un file *reference.txt* che contiene un guida rapida di riferimento per lo sviluppo con Aliza.



The eyeball

L'avventura che ci permetterà di imparare a sviluppare con Aliza si intitola "The eyeball"(la lingua scelta è l'inglese, in quanto tutti i giochi per ATARI 2600 sono esclusivamente in inglese). In questa avventura il giocatore, che si trova in una situazione finanziaria "nera", è partito alla ricerca di un favoloso e inestimabile gioiello chiamato "eyeball", nascosto in qualche parte del mondo.

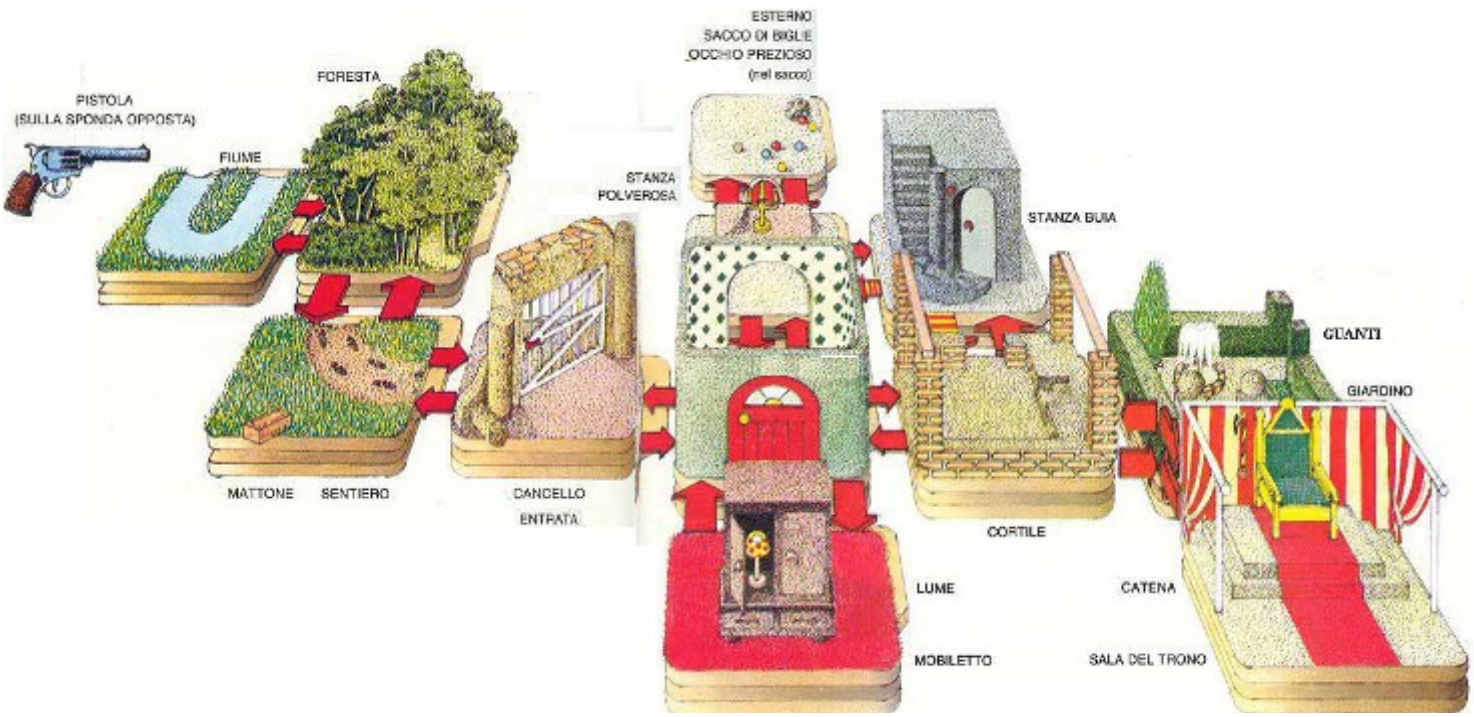
Per sua sfortuna, il protagonista è inseguito da un ispettore delle tasse che gli renderà la vita difficile. Quando l'ispettore compare infatti il giocatore finisce in gattabuia e l'avventura termina.

Come in ogni avventura ci sono parecchi oggetti che servono al giocatore. Una lampada ci permetterà ad esempio di trovare l'uscita in una stanza buia.

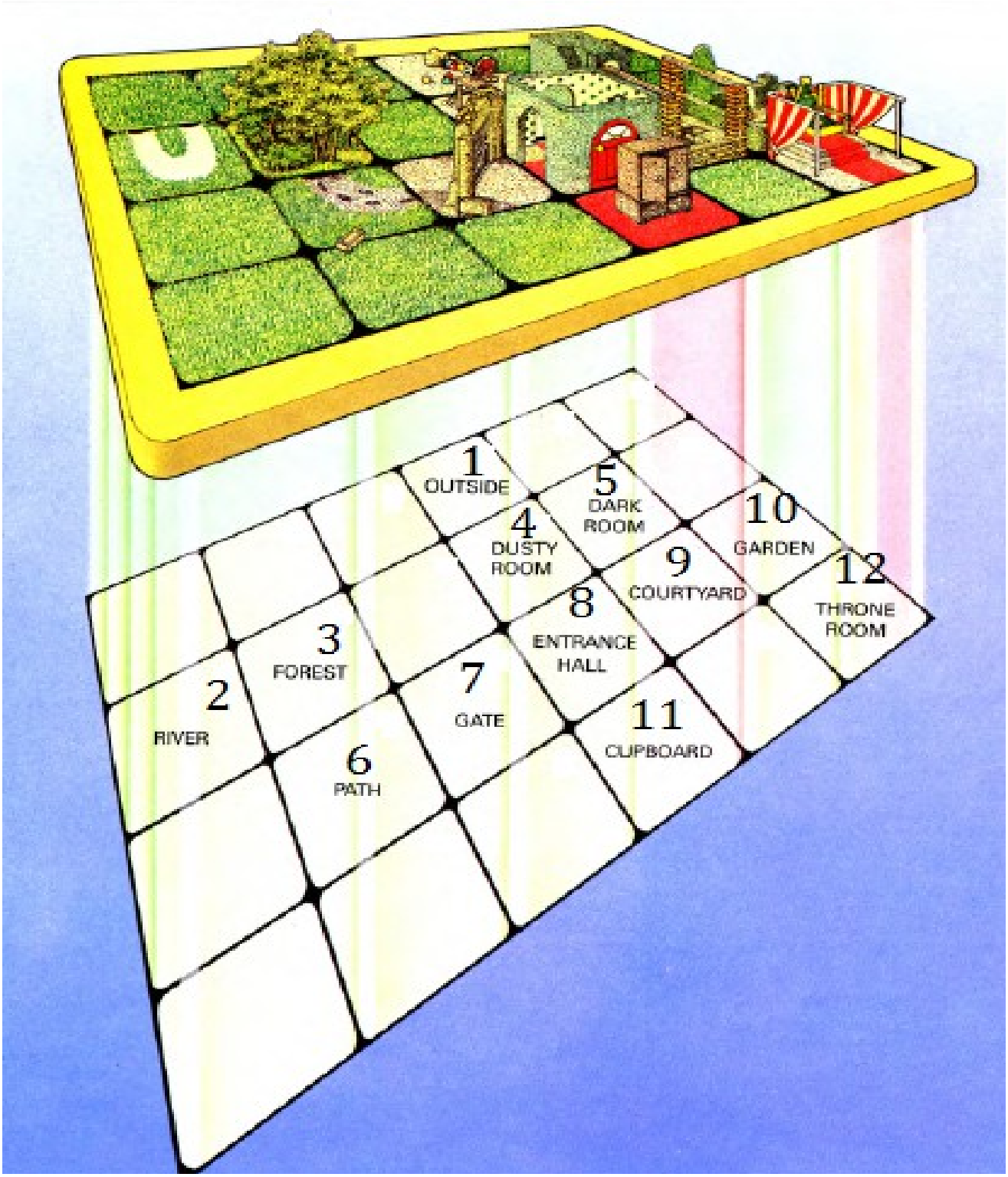
Non tutti gli oggetti che si troveranno nei vari ambienti sono però utili. Ad esempio un mattone molto pesante fa annegare il giocatore se tenterà di attraversare un fiume a nuoto.

E infine l'occhio purpureo è nascosto dentro ad un sacchetto di biglie. Ma non basta: per vincere l'avventura si dovrà essere nella sala del trono, sedersi e tirare la catena che pende dal soffitto indossando dei guanti! In caso contrario moriremo fulminati o la sala del trono diventerà una toilette gigante e si verrà sciacquati fuori dall'avventura!

Ecco la mappa del mondo in cui il giocatore si trova proiettato, composto da 12 ambienti e alcuni oggetti:



Associamo ad ogni ambiente un numero. Questo ci servirà per scrivere le “pagine” degli ambienti.


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

Il codice di Aliza

L'ATARI 2600 è una computer dalle risorse limitate (il costo di lancio doveva essere molto contenuto per essere accessibile al più vasto pubblico possibile) e di conseguenza programmare giochi (e sopratutto la grafica) risulta estremamente difficile e sfidante.

I programmatori dell'epoca erano dei veri e propri maghi nell'usare le esigue risorse del sistema per creare perle come *Pitfall* o *Adventure*.

In sintesi, l'ATARI 2600 mette a disposizione 4096 bytes di ROM (codice) e 128 bytes di memoria RAM. Aliza usa un bel po' di queste risorse per preparare la grafica (testo) e il motore delle nostre avventure testuali. Quello che rimane, sono 1650 bytes da distribuire tra i nostri testi e la “logica” della nostra avventura, e 10 variabili da 1 byte ciascuna che possono contenere un numero da 0 a 255.

Il “mondo” della nostra avventura potrà essere composto:

- da un massimo di 14 ambienti collegati tra loro con le direzioni NORD,SUD,EST,OVEST
- da un massimo di 15 oggetti che possono cambiare locazione e essere in caso presi dal giocatore
- da un massimo di 16 oggetti “dentro” alla descrizione degli ambienti
- da un massimo di 8 azioni possibili (di cui 3 libere)
- da un massimo di 32 messaggi liberi oltre a quelli di default (“non puoi”)

I testi possono contenere solamente i seguenti caratteri:

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

E le linee di testo devono per forza essere di 12 caratteri esatti. Anche i nomi degli oggetti sono limitati a 12 caratteri.

Non esiste la possibilità di salvare la propria posizione di gioco e in caso di vittoria o di fallimento, va resettata la console tramite il tasto di reset e si riparte dall'inizio.

La vera sfida però è scrivere meno testo possibile (che divora la poca ROM a disposizione) creando un'avventura avvincente! Infatti (spazi a parte, che Aliza tenta di ottimizzare) ogni lettera nei nostri testi occupa 1 byte.

Copiamo la cartella “eyeball” nella cartella di Aliza, apriamo Aliza e settiamo come progetto “eyeball”.

Apriamo con *Notepad++* il file *code.alz* dentro alla cartella “eyeball” e iniziamo a guardare come è composto.

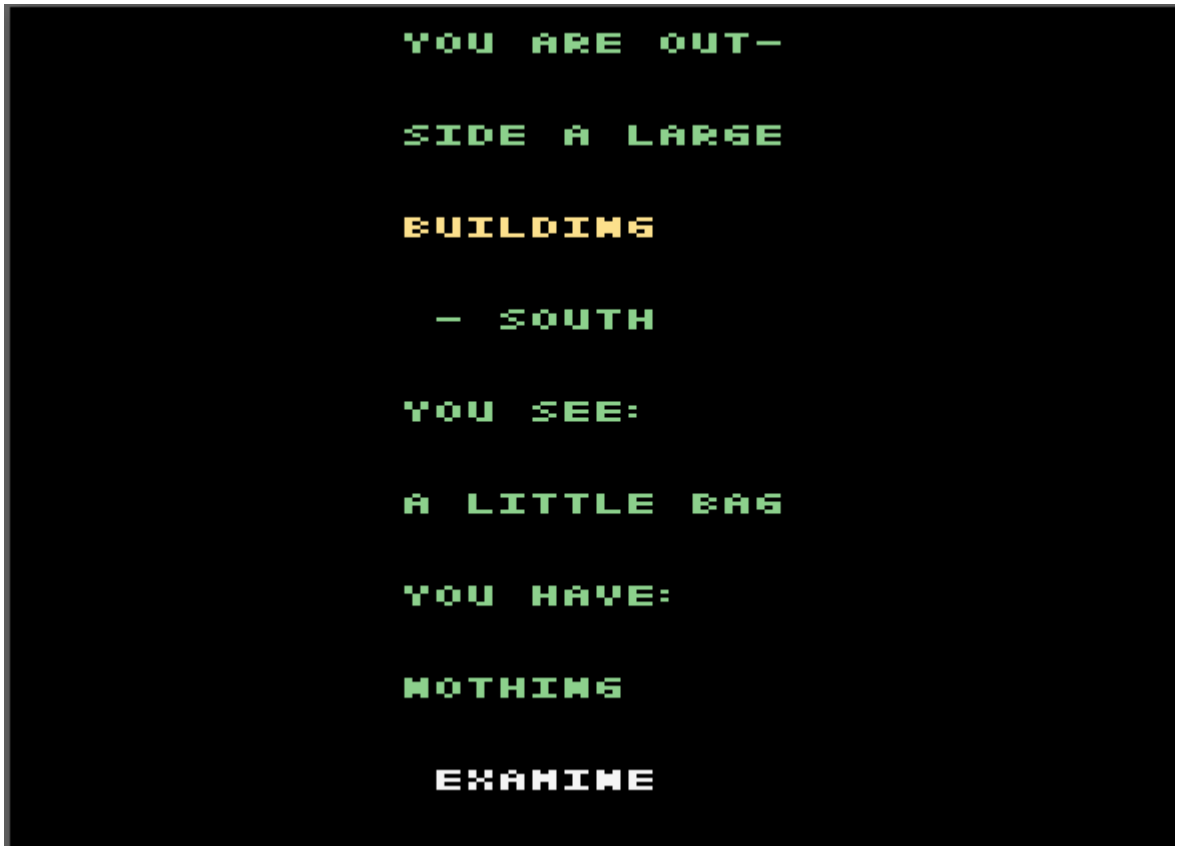
```
; ALIZA
; by E-Paper Adventures 2018  epaperadventures@gmail.com

; 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
```

Tutte le linee che cominciano con ; sono linee di commento e non verranno processate per la creazione del codice assembly.

I valori sotto [COLOR...] definiscono il colore da usare per i testi del gioco:

- color 1: colore di base (default: verde)
- color 2: colore della linea di testo selezionata (default: giallo)
- color 3: colore dell'azione scelta in basso (default: bianco)



La tabella dei valori per i colori la potete trovare qui:

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

Il sito è una risorsa incredibile per chiunque voglia cimentarsi nella programmazione di giochi per ATARI 2600, sopratutto nel linguaggio BATARI BASIC.

Fate attenzione che Aliza genera giochi in “formato” NTSC e che i valori sul sito sono in esadecimale mentre Aliza accetta solo valori decimali.

Sotto [CODESIZE] dobbiamo invece scrivere il numero di bytes riservati per la logica della nostra avventura. Essendo il valore 650, ed essendo la ROM totale 1650, significa che per i nostri testi avremo 1650-650 = 1000 caratteri (qualcosa di più perchè gli spazi potrebbero essere ottimizzati se possibile).

Sotto [START] va il numero della stanza di partenza, da 1 a 14.

Sotto [INVENTORY] il numero massimo di oggetti trasportabili. Questo non dovrebbe superare i 3-4 oggetti, e questo è dovuto più che altro al problema che al massimo abbiamo a disposizione 16 linee di testo su due schermate. Considerando che la descrizione della stanza sarà tipicamente di 3-4 righe, che le direzioni possibili sono 4, e che alcune righe sono “prese” per frasi come “YOU SEE” o “YOU HAVE”, permettere al giocatore di avere più di 4 oggetti porta al rischio che in alcuni casi le 16 linee non bastino (in questo caso semplicemente non si vede tutto quello che si dovrebbe vedere).

Sotto [DEFAULTSTRINGS] troviamo invece i messaggi di base, anche quelli in reazione a comandi che non vengono “catturati” dal codice che andremo a scrivere successivamente.

Attenzione: se modifichiamo questi messaggi (ma il “senso” degli stessi deve rimanere uguale), devono essere mantenute lo stesso numero di linee usate. E ogni riga, deve essere per forza di 12 caratteri.

```
[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!       "
```

L'indice sopra ogni messaggio (commento) potrà essere utilizzato per far apparire qual messaggio di base qualora ne avessimo bisogno.

Seguono le azioni (verbi) possibili, anch'essi indicizzati, e le 4 direzioni di base per muoversi tra gli ambienti.

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


;-------------
" CONTINUE..."
;-------------
;exits
" - NORTH    "
" - SOUTH    "
" - EAST     "
" - WEST     "
```

I verbi 5,6,7 sono quelli che in caso possono essere modificati liberamente. Gli altri, devono essere mantenuti nel loro significato ma si possono usare eventualmente dei sinonimi (GET invece di TAKE, ad esempio).

Dopo i verbi c'è il testo che appare sotto ai messaggi che compaiono a seguito di un azione. In questo caso il giocatore può solamente premere “sparo”.

Il blocco successivo è quello che collega i vari ambienti tra loro.

```
[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
```

Il primo numero (2 cifre) è la stanza di cui vogliamo descrivere le uscite. Seguono 4 numeri a due cifre che indicano quale altra stanza raggiunge il giocatore andando a NORD,SUD,EST ed OVEST. Se il numero e 00, significa che quella direzione non è possibile.

Vanno sempre specificate tutte e 14 le stanze, anche se alcune non sono usate.

E' possibile anche creare uscite ulteriori rispetto a quelle “standard” appena descritte. Vedremo dopo un esempio di questo.

Il pezzo di codice successivo descrive gli oggetti che vengono visualizzati nella parte “YOU SEE:” oppure nell'inventario. Sono quindi oggetti che partono da una certa locazione.

```
[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,"............"
```

Bisogna sempre specificare tutti e 15 gli oggetti.

Per ognuno di essi, abbiamo una linea con l'indice dell oggetto (2 cifre), la locazione di partenza (da 01 a 14 = nella relativa stanza, 00 = fuori dal gioco, 15 = nell'inventario), l'indicazione se l'oggetto è prendibile (Y) o no (N) e infine la descrizione dell'oggetto di massimo 12 caratteri.

La sezione successiva sotto [TEXT] contiene proprio tutti i messaggi (da 01 a 32) che vogliamo far apparire nel gioco a seguito di azioni o condizioni particolari.

```
[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. "
```

Ogni messaggio è quindi creato da una linea con l'indice (attenzione: gli indici devono essere progressivi) e da 1 a 8 righe di testo da 12 caratteri.

Il blocco successivo [ROOMSTEXT] è quello della descrizione delle stanze. Anche in questo caso, la prima riga è l'indice della stanza (attenzione che devono essere progressivi) e poi possiamo avere da 1 a 8 righe di testo. E' possibile non utilizzare tutte e 14 le stanze, come in questo caso:

```
[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        "
```

Qui, oltre al testo, è possibile indicare ad Aliza che una particolare linea “contiene” un oggetto di cui è specificato l'indice (da 17 a 32). Ad esempio:

```
"BUILDING    ",18
```

Potremo così intercettare azioni sull'oggetto BUILDING nel momento in cui il giocatore seleziona la riga “BUILDING” e preme sparo (ad esempio: EXAMINE BUILDING).

C'è anche un'altra possibilità. Possiamo indicare che se il giocatore ha scelto l'azione GO (vai) e preme sparo su una certa linea, allora il giocatore deve essere automaticamente spostato ad una stanza diversa. Per farlo, basta far seguire al testo della linea > e il numero della stanza. Nell'avventura di default ad esempio:

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

Se il giocatore seleziona la riga “LITTLE HUT” (piccola capanna) e l'azione GO (vai), verrà spostato alla stanza 3.

GO è anche la sola azione che il giocatore può selezionare per le direzioni NORD,SUD,EST,OVEST.

Prima di vedere le sezioni in cui stabiliremo la logica della nostra avventura, vediamo ancora il comportamento delle azioni possibili.

GO: abbiamo già visto che è l'unica azione che si può usare per le direzioni standard e quelle inserite come righe nella descrizione delle stanze. Attenzione: GO unito alla direzioni standard (NORD,SUD,EST,OVEST) della stanza non può essere intercettato per creare una gestione particolare (ad esempio impedire quella direzione finchè non accade prima qualcosa). Invece questo è possibile per le direzioni inserite come righe della stanza..

EXAMINE: se non specificata, questa azione porta alla risposta “NOTHING INTERESTING” (niente di interessante)

TAKE: assieme a DROP, permette di prendere oggetti o di lasciarli. Nel caso ne abbiamo già troppi, apparirà il messaggio “YOU HAVE TOO MANY THINGS”. Se non si può prendere l'oggetto, “YOU CANNOT”

USE: se non specificata, questa azione porta alla risposta “IT DOESN'T SEEM TO WORK” (non sembra funzionare)

PUSH, PULL,OPEN:se non specificate, queste azioni portano alla risposta “YOU CANNOT” (non puoi)

Come detto, PUSH,PULL,OPEN possono essere liberamente cambiate. Anche EXAMINE e USE in realtà, ma tenete conto del fatto che la risposta standard sarà “niente di interessante” e “non sembra funzionare” invece di “non puoi”. Al caso, nel file assembly, per modificare i messaggi standard vanno cambiati gli indici dei messaggi (da 128 a 135) nell'ultima parte del codice:

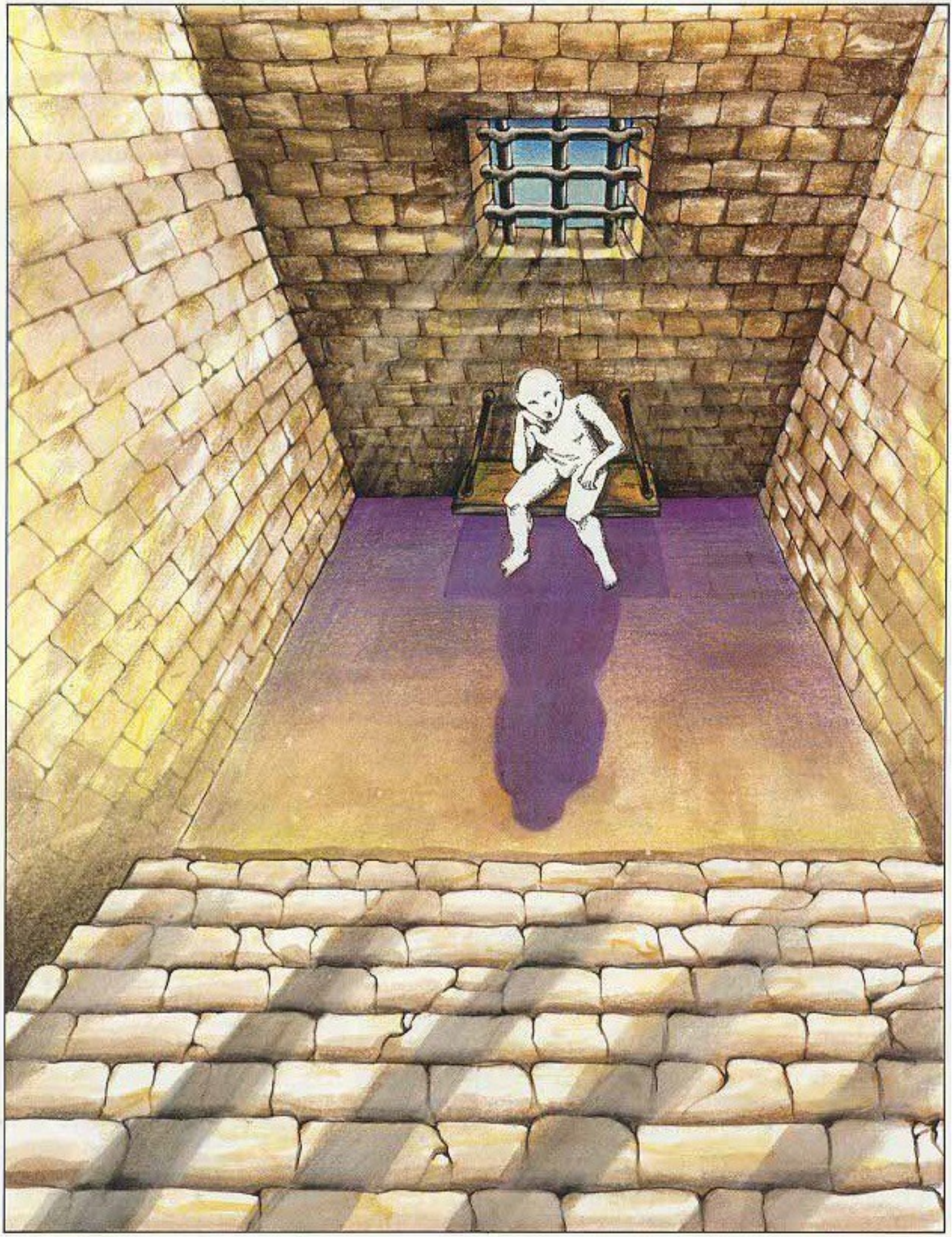
```
    dc 0
baselabel37
    dc 130,129,130,128,130,130,130,130
    dc #$Fc,#$3a,2
    dc #$Fc,#$52,2
    dc #$Fc,#$6a,1
    dc #$Fc,#$76,2
    dc #$Fc,#$8e,1
    dc #$Fc,#$9a,1
    dc #$Fc,#$a6,1
    dc #$Fc,#$b2,1
baselabel178
    dc 1,2,4,8,16,32
    org $fffc
    .word baselabel123
    .word baselabel123
```

La linea evidenziata è proprio quella che associa un messaggio standard a ogni azione non intercettata dalla logica della nostra avventura: messaggio 130 per “GO”, 129 per “EXAMINE, e così via.

Prima di vedere le sezioni [ACTIONS] e [CONDITIONS], dove appunto viene definita la logica dell'avventura, proviamo a generare l'assembly di “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 ]
```

Come si può vedere, rimangono solo 172 + 44 byte tra codice e testo. Questo per ribadire che è fondamentale che i testi siano il più possibile sintetici ma efficaci, in modo da avere più spazio possibile per creare le interazioni che rendono interessante la nostra avventura. Vedremo ora come farle.



le variabili dell'avventura

Aliza mette a disposizione le seguenti variabili per creare la logica della nostra avventura:

POS: è la posizione (stanza) in cui si trova il giocatore

P01..P15: è la posizione (stanza) in cui si trova l'oggetto 01..15. Se il valore è 0, l'oggetto è fuori dal gioco, se 15 è nell'inventario del giocatore.

V01..V10: sono variabili che possiamo usare liberamente, e possono contenere un valore da 0 a 255. Inizialmente hanno tutte valore 0.

In “eyeball”, le variabili libere sono così utilizzate:

```
; 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
```

V10 non è quindi utilizzata.

V01..V10 possono anche essere usate a livello dei bit che le compongono, nel momento in cui avessimo bisogno solo di flag del tipo 0/1. Infatti un byte (e quindi una variabile) può essere vista come l'insieme di 8 bit (8 valori 0 o 1) e quindi con una variabile possiamo tener traccia di 8 eventi accaduti nell'avventura.

Tuttavia questo tipo di utilizzo delle variabili è molto più dispendioso a livello di ROM e quindi va evitato se possibile.

Istruzioni possibili

Queste sono le sezioni in cui decidiamo cosa far accadere nella nostra avventura a seguito delle azioni del giocatore.

Ogni volta che eseguiamo una azione che non sia GO + NORD,SUD,EST,OVEST (automatiche), il motore del gioco esamina cosa c'è in ACTIONS e se abbiamo intercettato la particolare azione appena compiuta. In caso negativo, risponde col messaggio standard associato al verbo oppure nel caso di TAKE e DROP, gestisce lo spostamento dell'oggetto automaticamente (rispondendo “DONE” se l'azione è andata a buon fine oppure “YOU CANNOT”).

Successivamente a ogni azione, viene eseguito il codice di CONDITIONS, dove ad esempio possiamo stabilire che se abbiamo eseguito più di un certo numero di azioni accada qualcosa, come ad esempio la comparsa di un oggetto o la morte del giocatore.

Il linguaggio utilizzato in ACTIONS e CONDITIONS è molto semplice e le istruzioni possibili sono molto poche. Attenzione che devono essere scritte rispettando esattamente il numero di spazi o di cifre e i nomi delle variabili (Aliza è case sensitive).

```
done
```

termina l'esecuzione della sezione ACTIONS o CONDITIONS

```
show XXX
```

XXX = 001-032 or 128-135 (indice di un messaggio libero o di un messaggio di base)

Mostra il messaggio e se usato nella sezione ACTIONS, esce dalla sezione

```
end XXX
```

XXX = 001-032 or 128-135 (indice di un messaggio libero o di un messaggio di base)

Mostra il messaggio e termina il gioco (attende un comando di reset)

```
XXX=YYY
```

XXX è una variabile

YYY è una variabile oppure un numero da 0 a 255

Assegna il valore YYY a XXX

Ad esempio:

POS=002

sposta il giocatore nella stanza 2.

```
XXX++
```

Incrementa il valore della variabile XXX di 1. Se il valore supera 255, XXX torna a 0

```
XXX--
```

Decrementa il valore della variabile XXX di 1. Se il valore diventa meno di 0, XXX diventa 255

```
XXX&=YYY
```

XXX è una variabile

YYY è una variabile oppure un numero da 0 a 255

Assegna il valore (XXX & YYY) a XXX con & l'operazione di AND binaria

Ad esempio, se V01 è uguale a 3 (00000011)

V01&=002

significa V01 = (00000011) & (00000010) ovvero (00000010) = 2

```
XXX|=YYY
```

XXX è una variabile

YYY è una variabile oppure un numero da 0 a 255

Assegna il valore (XXX | YYY) a XXX con | l'operazione di OR binaria

Ad esempio, se V01 è uguale a 3 (00000011)

V01|=004

significa V01 = (00000011) & (00000100) ovvero (00000111) = 7

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

[test1]...[testN] sono espressioni che devono essere tutte vere perciò che il codice […] venga eseguito.

[…] a sua volta può contenere altri blocchi if... end if

[test?] può essere:

XXX=YYY → XXX uguale a YYY

XXX<YYY → XXX minore di YYY

XXX>=YYY → XXX maggiore o uguale di YYY

XXX<>YYY → XXX diverso da YYY

XXX&YYY=ZZZ → (& è l'operazione AND binaria) XXX & YYY uguale a ZZZ

con XXX una variabile e YYY,ZZZ variaibile o un numero da 0 a 255

Ad esempio:

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

se il giocatore si trova nella stanza 2, incrementa V01 di 1.

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

se il giocatore si trova nella stanza 2 e allo stesso tempo la variabile V02 è minore di 5, incrementa V01.

L'ultimo costrutto si può usare solamente in ACTION e permette di intercettare le azioni compiute dal giocatore.

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

X = indice dell'azione (1 = GO, 2 = EXAMINE, ...)

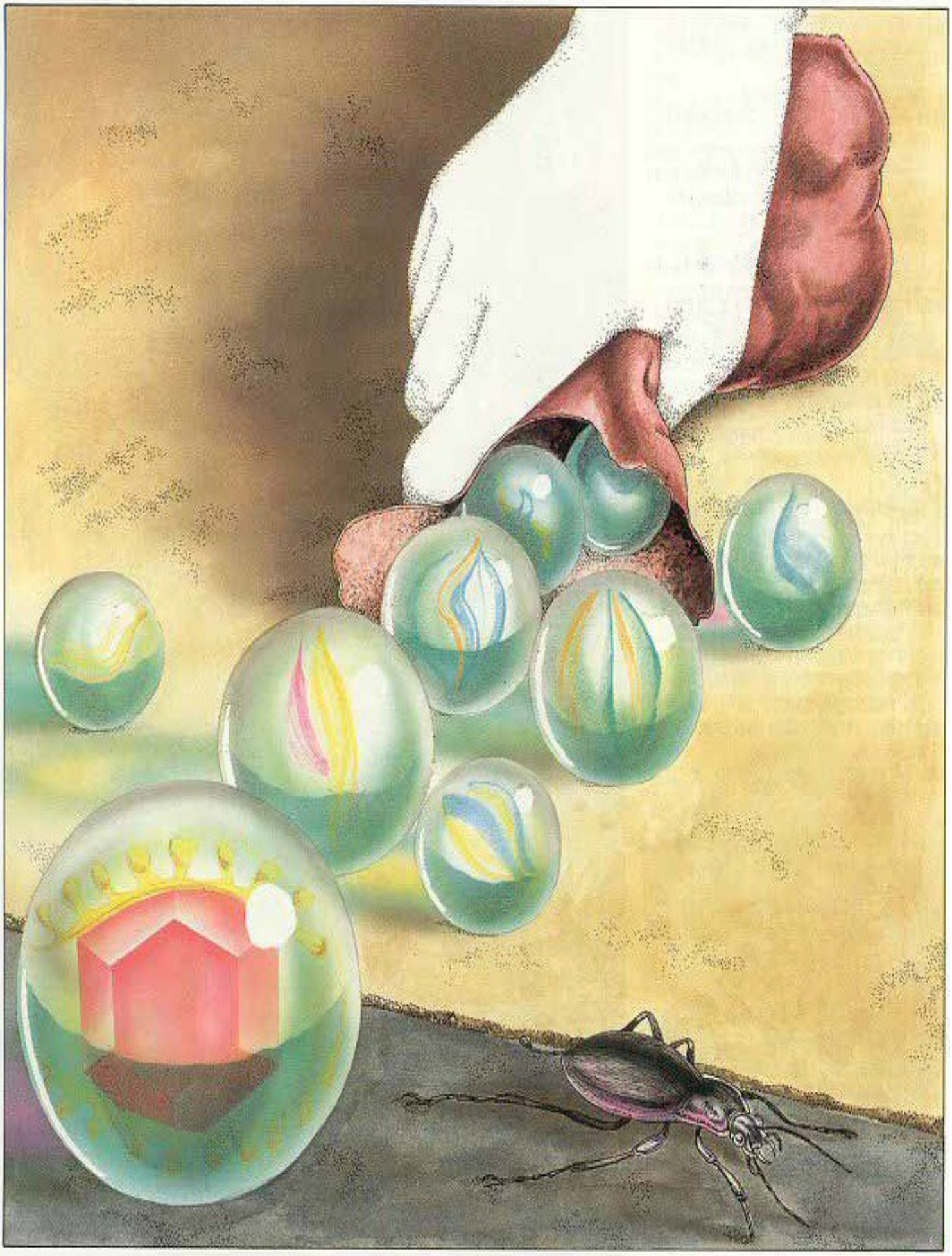
YY = oggetto su cui avviene l'azione, valori validi sono 01..15 o 17..32

In caso l'azione sia quella specificata, viene eseguito il codice in […]

Attenzione: non si possono innestare blocchi “action” dentro ad altri blocchi “action”.

Se si vuole catturare tutte le azioni indipendentemente dall'oggetto, usare 00 come YY

Se si vuole catturare tutte le azioni su un oggetto indipentemente dal verbo, usare 0 come X



ACTIONS e CONDITIONS

Andiamo ora a vedere la sezione CONDITIONS di “eyeball” e analizziamolo.

```
[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
```

Il primo blocco if controlla se la variabile V01, usata per contare il numero di azioni compiute è minore di 50. In tal caso, V01 viene incrementata di 1. Ricordiamoci che CONDITIONS viene chiamata tutte le volte che il giocatore compie una qualsiasi azione.

Il secondo blocco if controlla se la variabile V01 (numero di mosse) è uguale a 25 e contemporaneamente deve essere che la posizione del giocatore non sia la stanza 5: in tal caso viene mostrato il messaggio di indice 20. Attenzione: a differenza di quello che accadrebbe in ACTION, show NON termina l'esecuzione del codice di CONDITIONS.

Quello che accade dunque è che viene mostrato il messaggio 20 e poi verrà mostrato (se previsto) il messaggio innescato dall'azione del giocatore.

Il terzo if è simile al secondo, ma in caso sia verificato, il gioco termina col messaggio di indice 21 (sono state fatte troppe mosse).

```
; 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
```

Questo blocco anzitutto mette a 0 (fuori dal gioco) le posizioni degli oggetti 12 e 13. Questi sono in realtà delle "uscite" che il giocatore vede solo se ha la lampada con se. Infatti, se la posizione della lampada è nell'inventario (P07=015) allora le posizione degli oggetti 12 e 13 diventano la stanza 5 e quindi vedremo le due uscite.

Il secondo if è simile, ma in questo caso basta che la lampada sia presente nella stanza (P07=005).

```
; 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
```

Questa parte riguarda la stanza foresta, nella quale il giocatore viene pietrificato se rimane in essa per più di 2 mosse. Il primo if infatti controlla se il giocatore è nella foresta e in tal caso incrementa V04 (che tiene conto delle mosse fatte nella foresta) e poi controlla se il numero di mosse fatte è 3 ed in tal caso termina il gioco col messaggio 019. Il secondo if rimette invece a 0 il numero di mosse nella foresta se il giocatore non è nella foresta.

```
; 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
```

Questo pezzo di codice riguarda il nemico del giocatore, l'ispettore delle tasse. Il primo if controlla se l'ispettore è già apparso (V05=1) e la sua posizione è la stanza 10 (giardino), in tal caso significa che l'unica azione che aveva a disposizione il giocatore non è stata quella di sparargli (che sposta l'oggetto ispettore fuori dal gioco) e quindi il gioco termina col messaggio di indice 6.

Il secondo if verifica se l'ispettore non è ancora apparso (V05=0) e se la posizione del giocatore è il giardino e in tal caso l'ispettore viene collocato nella stessa stanza e V05 viene posta a 1 e si fa apparire il messaggio 5.

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

Questo if verifica se siamo in una stanza diversa da quella del trono (12) e in tal caso mette a 0 la variabile V08.

Essa viene messa a 1 quando ci si siede sul trono, come vedremo tra poco.

Passiamo ora alla sezione ACTIONS: qui intercetteremo tutte le azioni che ci interessano per creare le interazioni del giocatore con l'avventura.

```
[ACTIONS]


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

Questo primo pezzo cattura due azioni.

La prima è GO + oggetto 12 (“A DOOR:SOUTH") che è una delle uscite che vediamo nella stanza buia (5) se abbiamo la lampada. In caso il giocatore esegua questa azione, la variabile POS (posizione del giocatore) diventa la stanza 9.

La seconda è GO + oggetto 12 (“A DOOR:WEST") che è l'altra uscita che vediamo nella stanza buia se abbiamo la lampada. In questo caso il giocatore va nella stanza 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
```

Questa parte intercetta GO + RIVER e se abbiamo il mattone (P02=15) allora il gioco termina con un messaggio che dice al giocatore che è affogato. Se invece la pistola è fuori dal gioco (e non siamo affogati...) allora la pistola viene messa nella stanza e il giocatore viene avvertito da un messaggio di averla trovata. Se invece la pistola era già apparsa, viene mostrato il messaggio di indice 10.

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

Se viene esaminato l'edificio (nella stanza 1) ovvero l'oggetto 18, la variabile V09 viene messa a 1 e viene mostrato il messaggio 18.

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

Se si esamina il sacchetto delle biglie che non è stato vuotato (V02=0) allora viene mostrato il messaggio 1m altrimenti il 2.

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

```
 end if
end action
```

Se esaminiamo il giardino (oggetto 19) e non lo abbiamo ancora esaminato (V06=0) e abbiamo letto la scritta sull'edificio (stanza 1), allora V06 viene messa a 1 e l'oggetto 3 (guanti) viene messo nella stanza dove si trova il giocatore (ovvero il giardino). Infine viene mostrato il messaggio 17.

Attenzione: nella sezione ACTIONS, show mostra il messaggio ed esce dalla sezione, quindi deve essere sempre l'ultima istruzione.

```
; open bag
action 5,01
 if V02=000
  V02=001
  P06=P01
  show 011
 end if
end action
```

Se il giocatore apre il sacchetto, e non è ancora aperto (V02=0), V02 diventa 1, compaiono le biglie (vengono spostate nella stessa stanza del sacchetto) e viene mostrato il messaggio 11.

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

Se vengono esaminate le biglie, se è la prima volta (V03=0) allora V03 viene messo a 1 e compare “l'eyeball” e viene mostrato il messaggio 4.

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

Esaminando la catena, viene mostrato il messaggio 3.

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

Spingendo l'armadio (PUSH THE CUPBOARD) per la prima volta (V07=0), V07 viene messo a 1 e compare la lampada (la sua posizione diventa quella del giocatore).

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

USE THRONE ci fa sedere se non lo siamo già (V08=0).

```
; 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
```

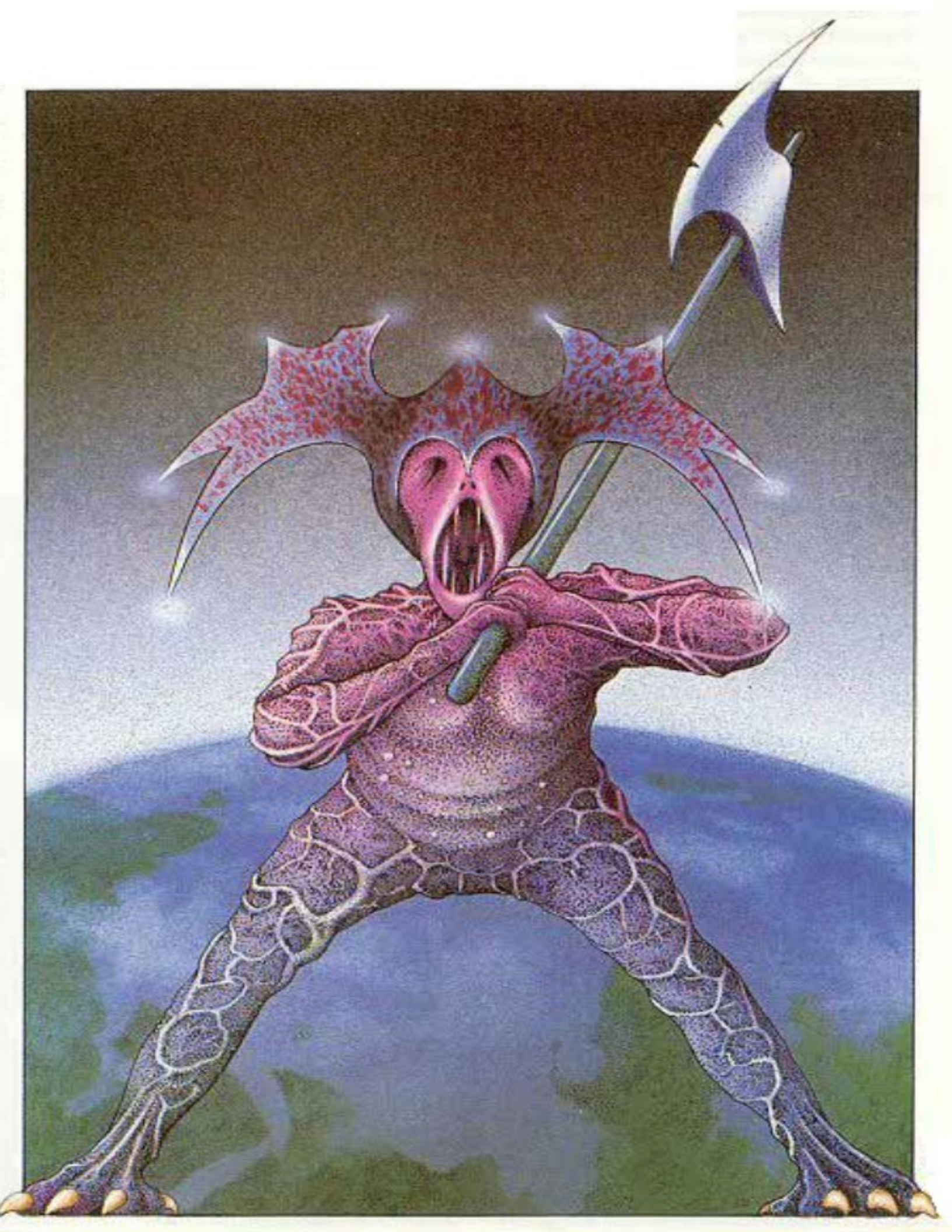
Se siamo in presenza dell'ispettore (P09=POS) ovvero è nella nostra stessa stanza, USE BRICK fa mostrare il messaggio 12. USE GUN invece ucciderà l'ispettore, il relativo oggetto viene messo fuori dal gioco e compare il cadavere (A BODY).

```
; 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
```

Infine se tiriamo la catena sul trono, possono accadere diverse cose.

Se abbiamo l'eyeball, i guanti e siamo seduti, allora il gioco termina con un messaggio di vittoria.

Se non abbiamo i guanti finiamo fulminati, in tutti gli altri casi veniamo “sciacquati via”.



Il debug dell'avventura

Al momento, questa versione di Aliza non permette un debug specifico: bisogna provare l'avventura giocandola e cercando eventuali errori di logica. Nel caso in cui volessimo simulare una certa situazione, il trucco è aggiungere all'inizio della sezione CONDITIONS una parte di codice di questo tipo:

```
if V10=000
 POS=???
 P01=???
 ...
 V01=...
 ...
 V10=001
 done
end if
```

Così facendo, immediatamente dopo una prima azione qualsiasi, e solo per una volta (usando la variabile V10), la posizione del giocatore, degli oggetti e il valore delle altre variabili libero saranno settate come abbiamo bisogno per testare una certa parte dell'avventura.

Durante la generazione del codice assembly Aliza segnalerà eventuali errori nel codice con alcuni messaggi, indicando il contenuto della linea relativa all'errore.

Invalid character < ... >

Si è utilizzato un carattere non valido nel file code.alz

Invalid or missing [...] section

Manca una sezione nel file code.alz

File code.alz not found!

File code.alz non trovato

Invalid code size

Si sta specificando una dimensione per ACTIONS e CONDITIONS non valida

out of ROM memory

Esaurita la memoria ROM

invalid string: ...

Stringa non valida (non del formato corretto)

Syntax error: ...

Errore di sintassi nel codice

Invalid text sequence: ...

Indice del testo o messaggio non corretto

Too many characters.

Troppi caratteri (ROM esaurita)

Too many strings.

Troppe stringhe per un singolo messaggio

Syntax error: unclosed action ...

un blocco action non è chiuso con end action

Syntax error: action not open ...

end action senza action

Syntax error: if not open ...

end if senza if

Syntax error: unclosed if

if senza end if

Unknown variable: …

variabile non esistente