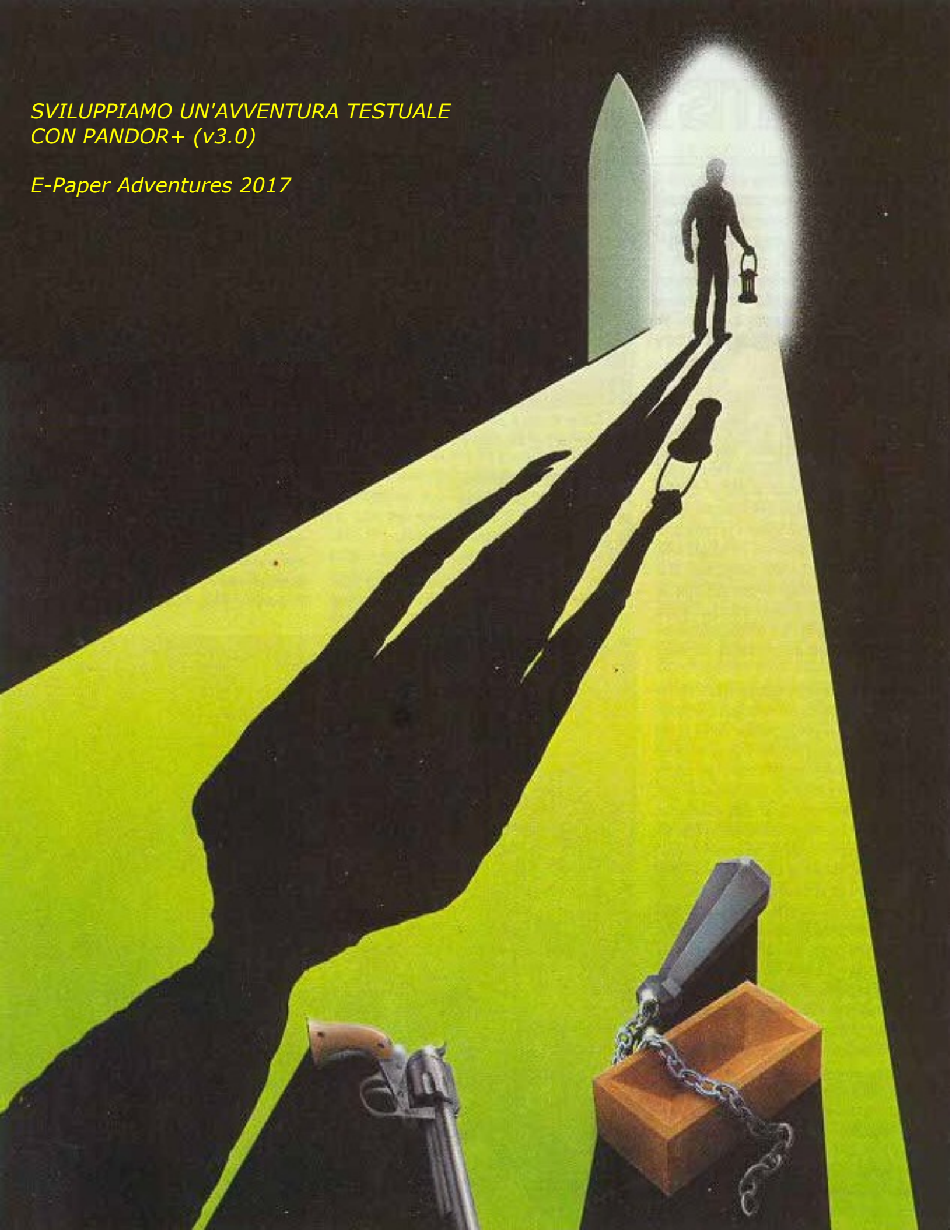
SVILUPPIAMO UN'AVVENTURA TESTUALE CON PANDOR+ (v3.0)

E-Paper Adventures 2017



Benvenuti!

Pandor+ è un sistema di sviluppo freeware per creare avventure testuali in lingua italiana o inglese da giocare su lettori a inchiostro elettronico (in particolare il reader Kindle di Amazon) ma anche su qualsiasi altro dispositivo con un browser Internet che supporti javascript. L'applicativo per lo sviluppo può essere eseguito su qualsiasi versione di Windows e non richiede installazione.

In questo tutorial verrà creata da zero una semplice avventura sullo stile di quelle disponibili sul sito www.epaperadventures.qlmagic.com

Questo progetto è completamente FREE. Potete copiarlo e distribuirlo senza limitazioni. 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. 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).

Il risultato finale dello sviluppo di un avventura testuale in Pandor+ è **un unico file HTML**. Tuttavia non è assolutamente necessario conoscere HTML ed è sufficiente una conoscenza base di javascript. Se non avete mai programmato non preoccupatevi: i costrutti da utilizzare con Pandor+ sono molto semplici e nel tutorial troverete numerosi esempi.

Per lo sviluppo di proprie avventure si consiglia l'utilizzo di Pandor+ in combinazione con Notepad++ e il browser Mozilla Firefox (versione 55 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 farlo alla pagina:

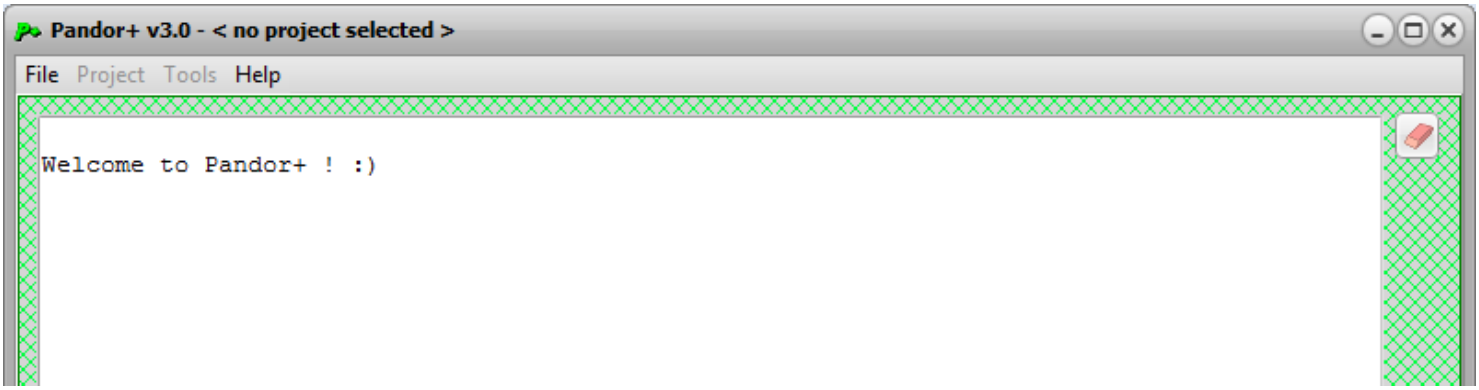
http://www.epaperadventures.qlmagic.com/don.html

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

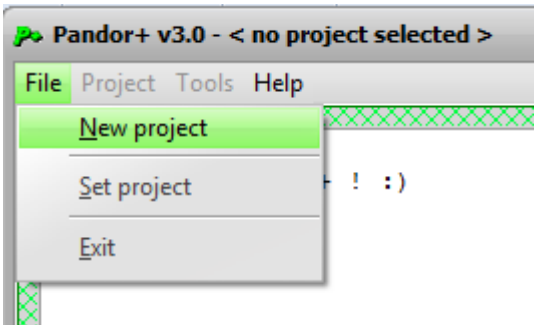
Revisione B.1 – Ottobre 2017

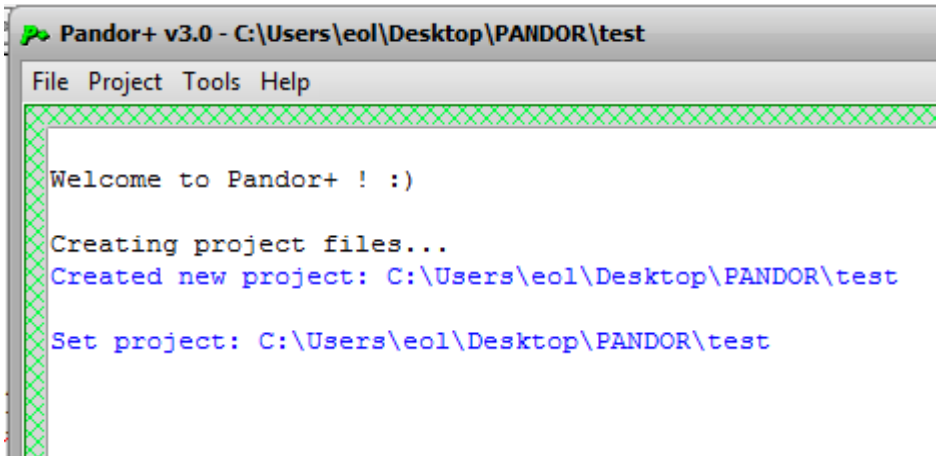
Creiamo un nuovo progetto

Per cominciare, copiamo l'applicazione *pandor+.exe* in una nuova cartella sul desktop (ad esempio: *PANDOR*) ed eseguiamola.



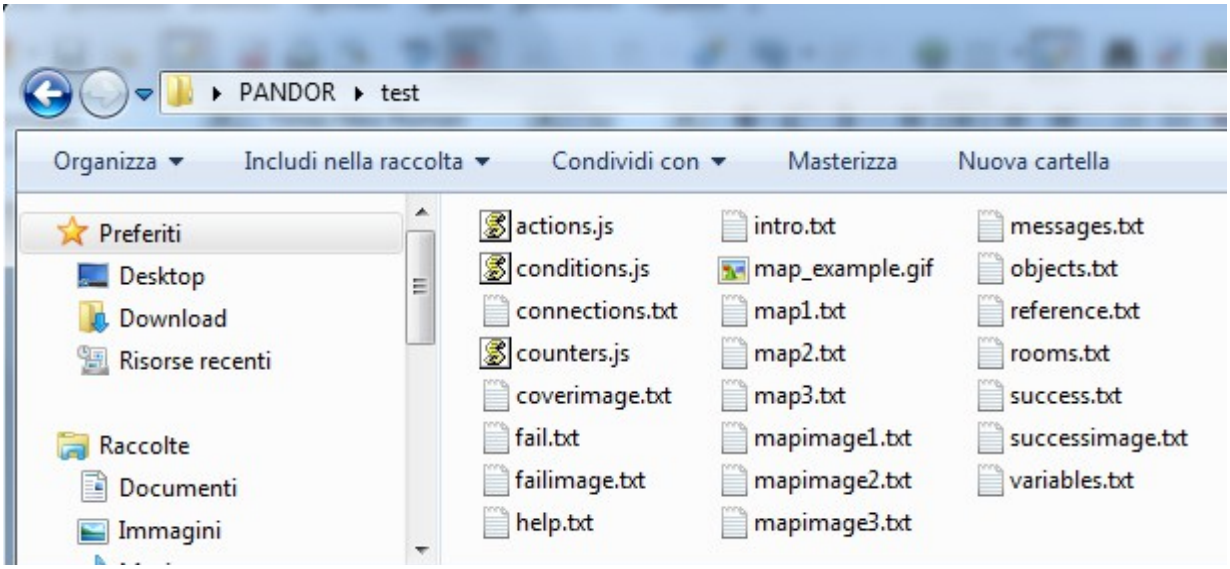
Andiamo nel menu *File* e creiamo un nuovo progetto scegliendo *New project*. Digitiamo come nome "test". Pandor+ creerà una sotto-cartella *test* con dentro diversi file.





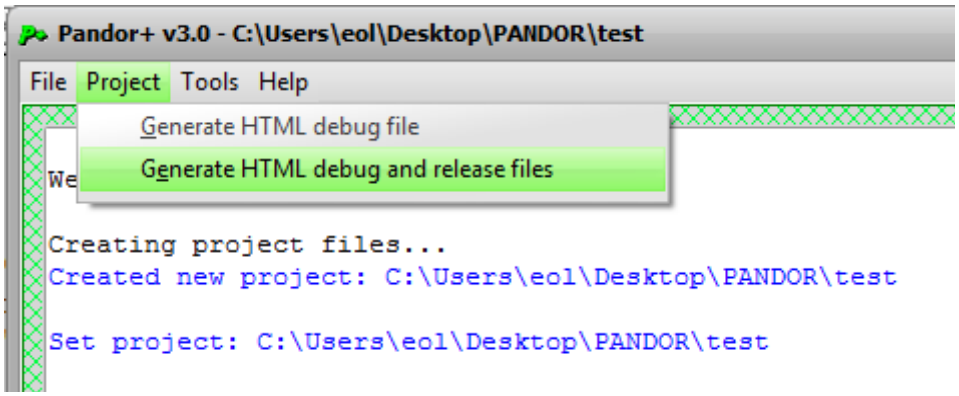
Pandor+ crea nella sotto-cartella *test* tutti i file che definiranno la nostra avventura. Oltre a questo, verrà selezionato come progetto corrente proprio "test".

Esaminiamo i file contenuti nella cartella *test*:

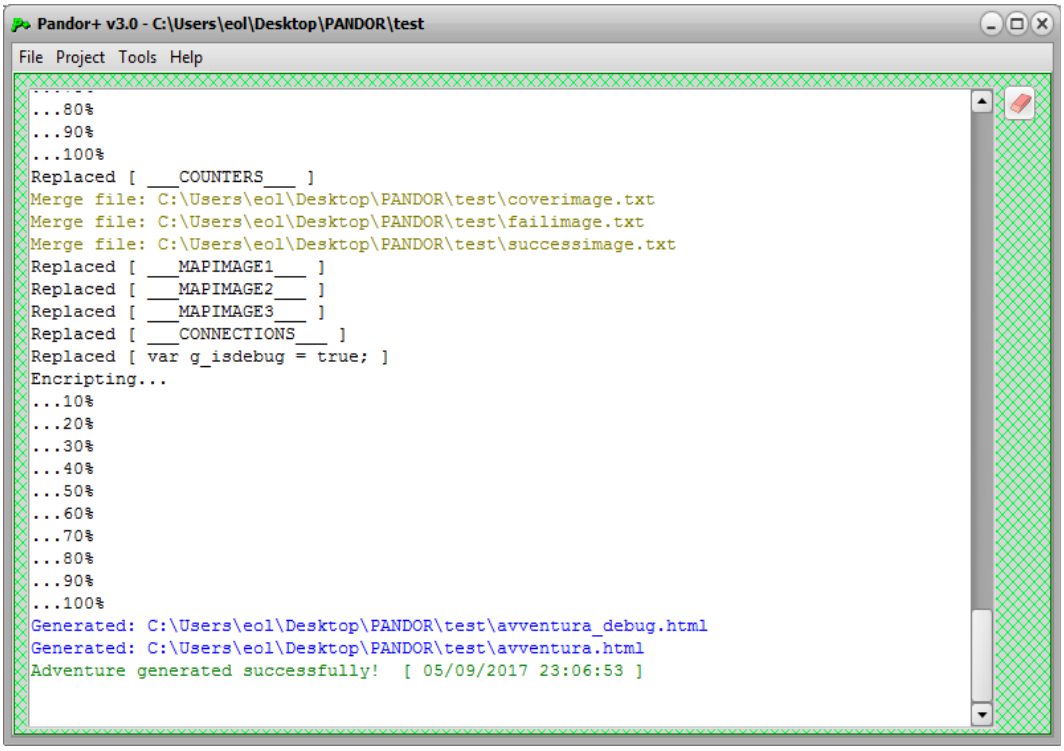


Qualora le estensioni dei file non fossero visibili (".js", ".txt", etc) conviene modificare le opzioni di Windows per renderle tali. Per farlo cercate nelle impostazioni la voce "opzioni cartella" o "opzioni esplora risorse": nel tab "visualizzazione" cercate la voce "nascondi le estensioni per i tipi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

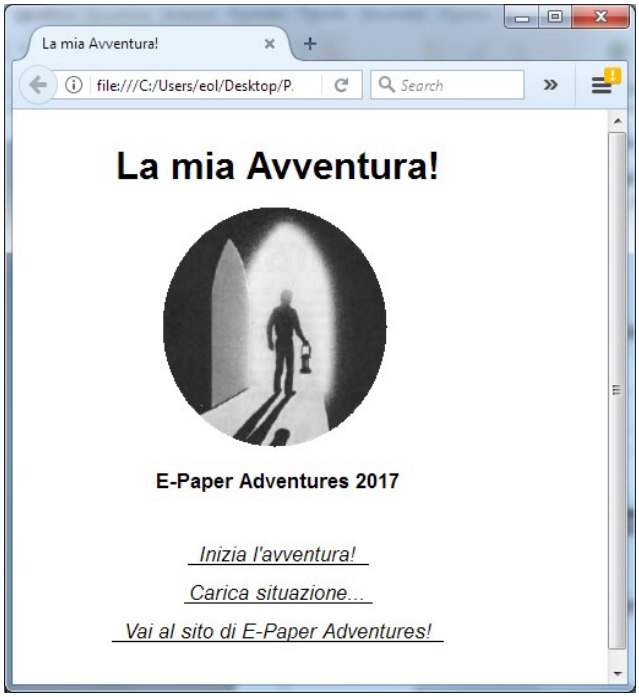
Ogni nuovo progetto creato contiene un'avventura di default con un paio di stanze e un paio di oggetti. Andiamo sul menu *Project* e scegliamo *Generate HTML debug and release files*.



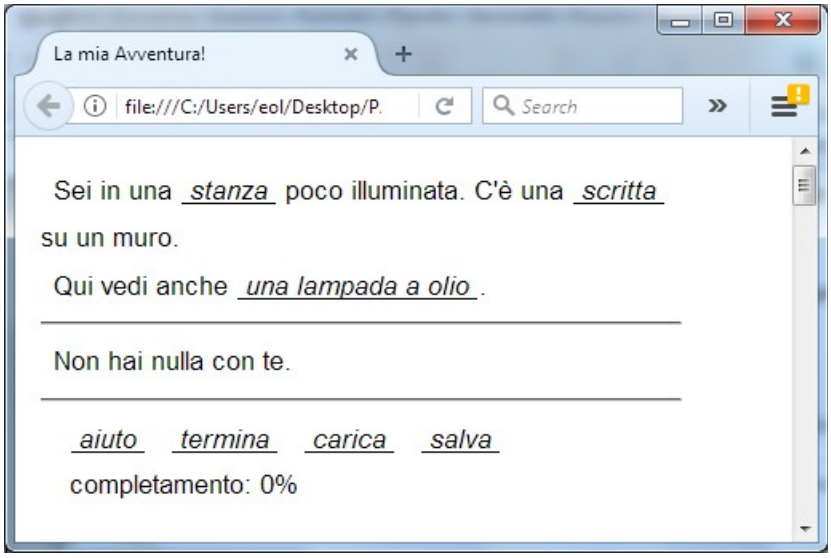
Pandor+ assemblerà i file contenuti nella cartella *test* e il risultato sarà la generazione dei file *avventura_debug.html* e *avventura.html* .



Apriamo *avventura.html* con il browser (ad esempio con tasto destro sul file e apri con...). Il browser potrebbe richiedere l'autorizzazione a eseguire script. Vedremo quindi apparire la prima schermata dell'avventura:



Proviamo a cliccare su *Inizia l'avventura!*: vedremo due pagine di introduzione (clicchiamo su *Continua*) e quindi la prima stanza del gioco.

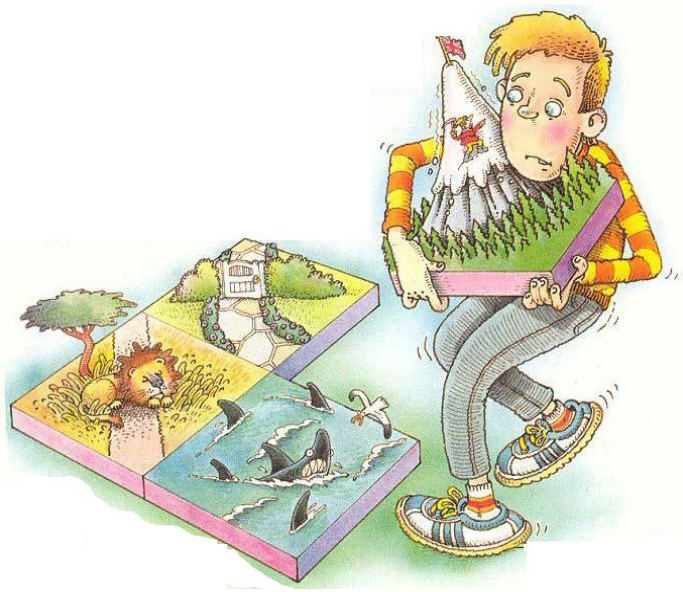


L’avventura di test è ben poco entusiasmante, si può fare poco o niente! Vedremo tra poco la creazione di un’avventura con diverse stanze, oggetti e interazioni di vario tipo. Intanto però vediamo cosa è successo con il comando *Generate HTML debug and release files.*

Pandor+ utilizza uno script di base (un mix di HTML e javascript) contenente il "motore" del gioco in cui sono lasciate incomplete le parti relative all'introduzione, alla descrizione delle stanze, ai messaggi, alle azioni possibili, etc. Al comando di generazione, i file di progetto vengono elaborati da Pandor+ e inseriti nelle sezioni incomplete per comporre il file finale HTML con l’avventura completa.

Possiamo decidere di generare solamente la versione di debug dell’avventura (file *avventura_debug.html*) oppure anche quella di release finale (file *avventura.html*). In quest’ultima risultano crittati i testi, in modo che non basti aprire il file del gioco per leggere descrizioni, oggetti, messaggi e così via. In generale la versione di release viene generata solo quando l'avventura è completa ed è stata testata adeguatamente.

Quando l'avventura è pronta per la pubblicazione, il file HTML va messo a disposizione su un proprio server web o, ancora meglio, inviato a epaperadventures@gmail.com : verrà inserita tra le avventure disponibili del sito www.epaperadventures.qlmagic.com !



L'occhio purpureo

L'avventura che ci permetterà di imparare a sviluppare con Pandor+ si intitola "L'occhio purpureo". In questa avventura il giocatore, che si trova in una situazione finanziaria "nera", è partito alla ricerca di un favoloso e inestimabile gioiello chiamato "Occhio purpureo", nascosto in qualche parte del mondo.

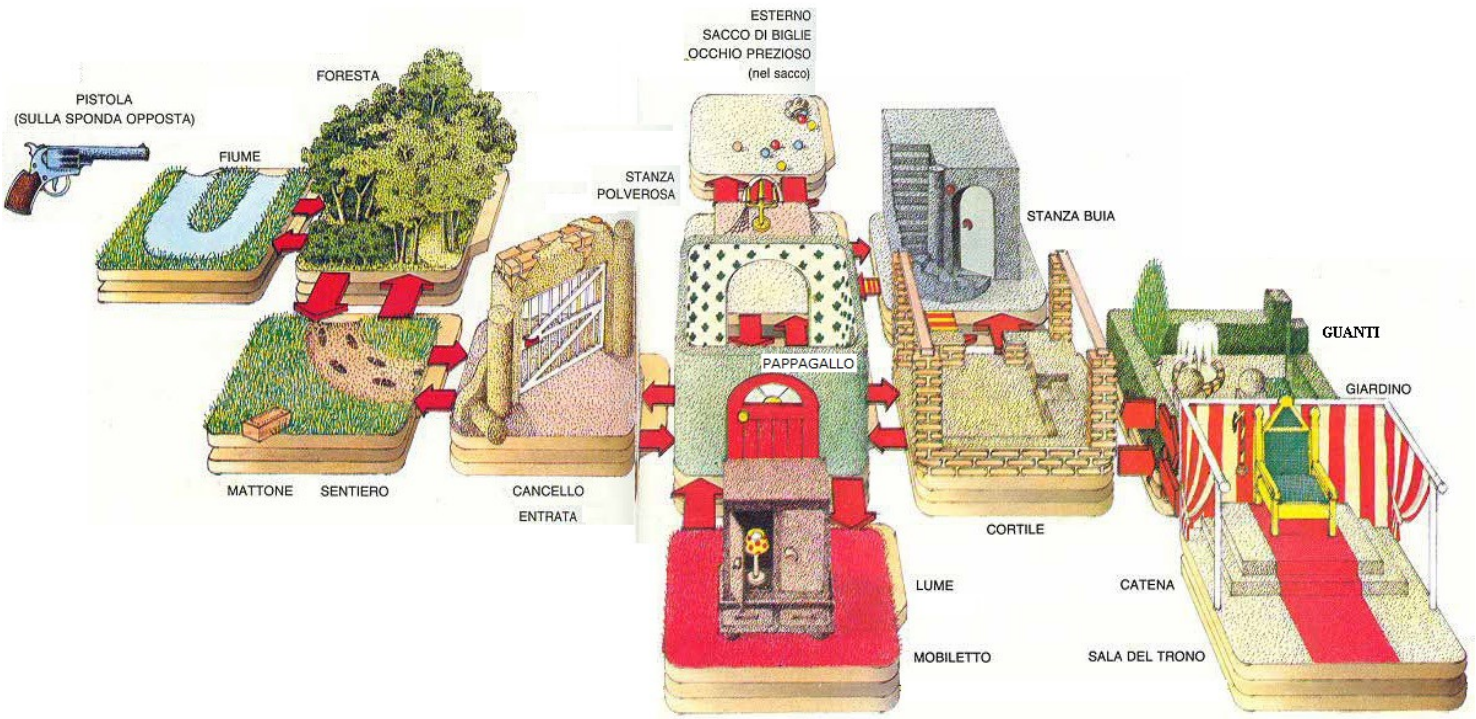
Per sua sfortuna, il protagonista è inseguito da un ispettore delle tasse che gli renderà la vita difficile. Quando l'ispettore compare infatti, possono succedere due cose. Se si ha con se un oggetto, egli se ne appropria, come anticipo del rimborso dell'enorme debito col fisco. Se invece non si ha ancora niente (o non si può pagare) 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.

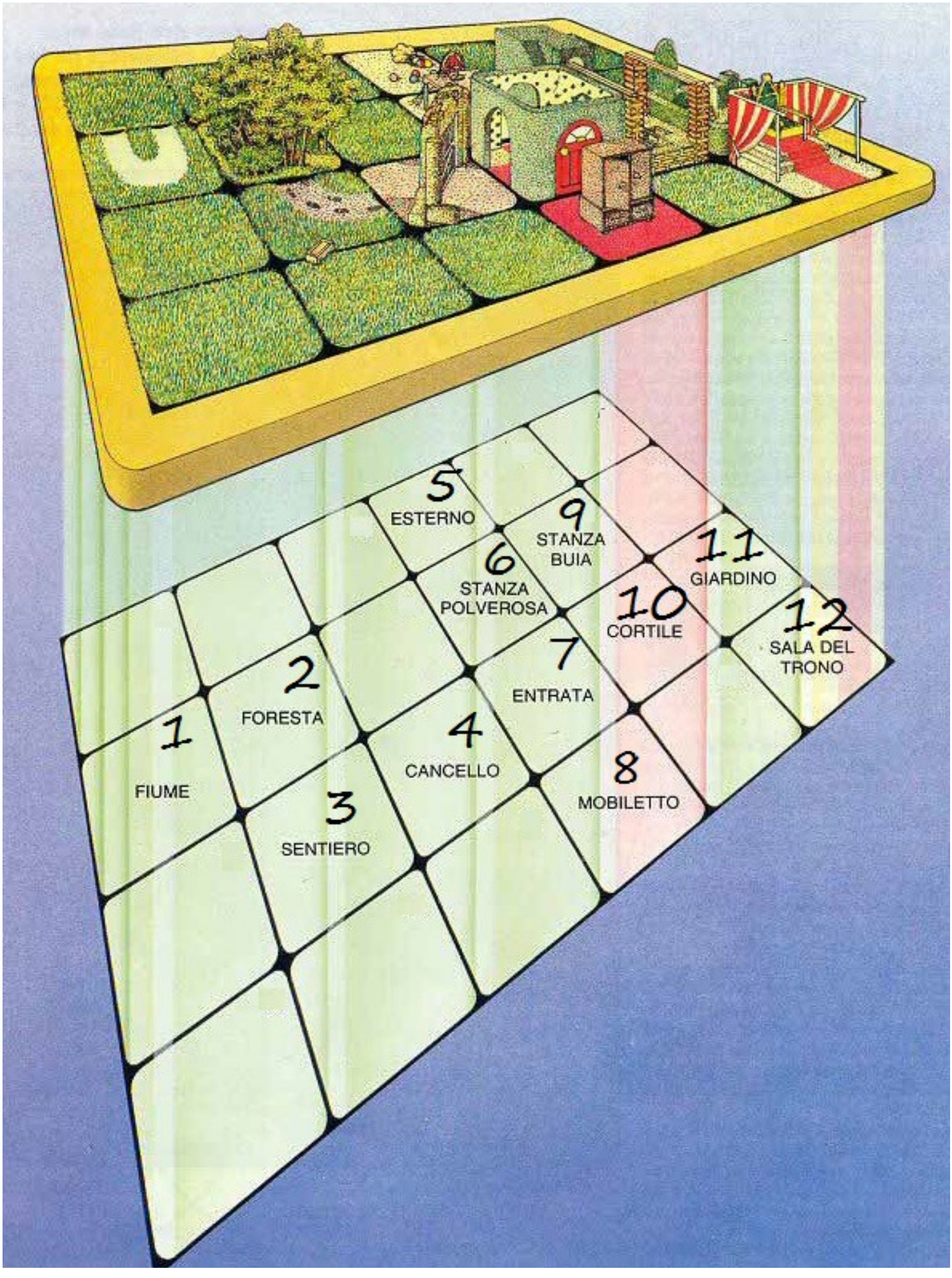
Non tutti gli oggetti che si troveranno nei vari ambienti sono però utili. Ad esempio un mattone molto pesante fa annegare il giocatore se tenterà di attraversare un fiume a nuoto.

E infine l'occhio purpureo è nascosto dentro ad un sacchetto di biglie che il giocatore troverà e dovrà capire di dover rovesciare. Ma non basta: per vincere l'avventura si dovrà essere nella sala del trono e tirare la catena che pende dal soffitto un numero corretto di volte 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:

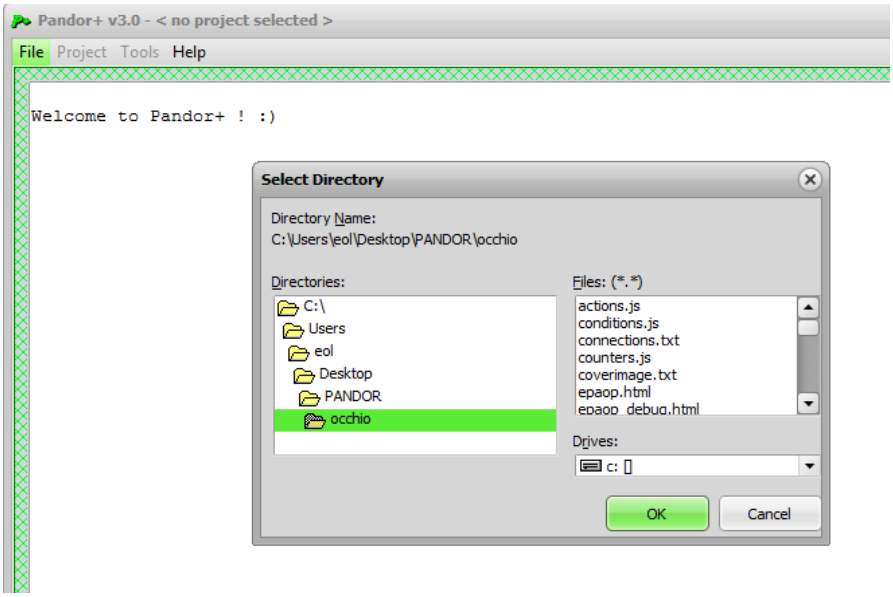


Associamo ad ogni ambiente un numero. Questo ci servirà per scrivere le descrizioni degli ambienti (detti anche "stanze") e gli oggetti in essi.


1
FIUME
2
FORESTA
3
SENTIERO
4
CANCELLO
5
ESTERNO
6
STANZA POLVEROSA
7
ENTRATA
8
MOBILETTO
9
STANZA BUIA
10
CORTILE
11
GIARDINO
12
SALA DEL TRONO

Cominciamo!

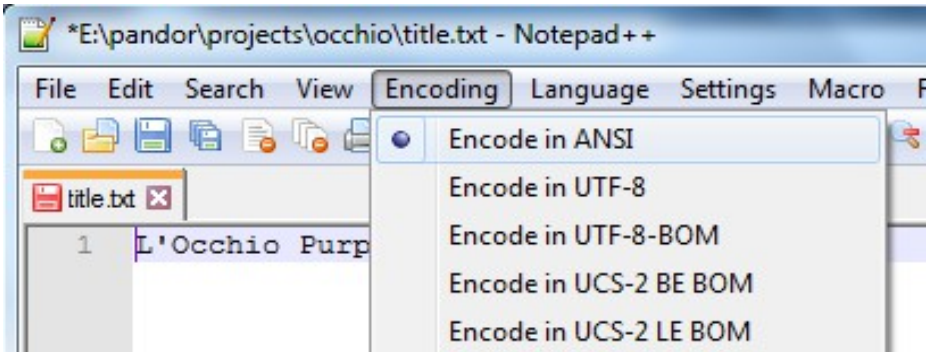
Per iniziare creiamo un nuovo progetto e chiamiamolo "occhio". Automaticamente Pandor+ crea la sotto-cartella *occhio* e imposta automaticamente come progetto corrente proprio *occhio*. Se ora chiudiamo Pandor+ e lo riapriamo, vediamo che nessun progetto è selezionato. Andiamo sotto *File* e scegliamo *Set project*: sull'albero delle cartelle che compare facciamo doppio click sulla cartella *occhio* (a destra vedremo i file in essa contenuti) e quindi scegliamo OK. Il progetto corrente diventerà nuovamente *occhio*.



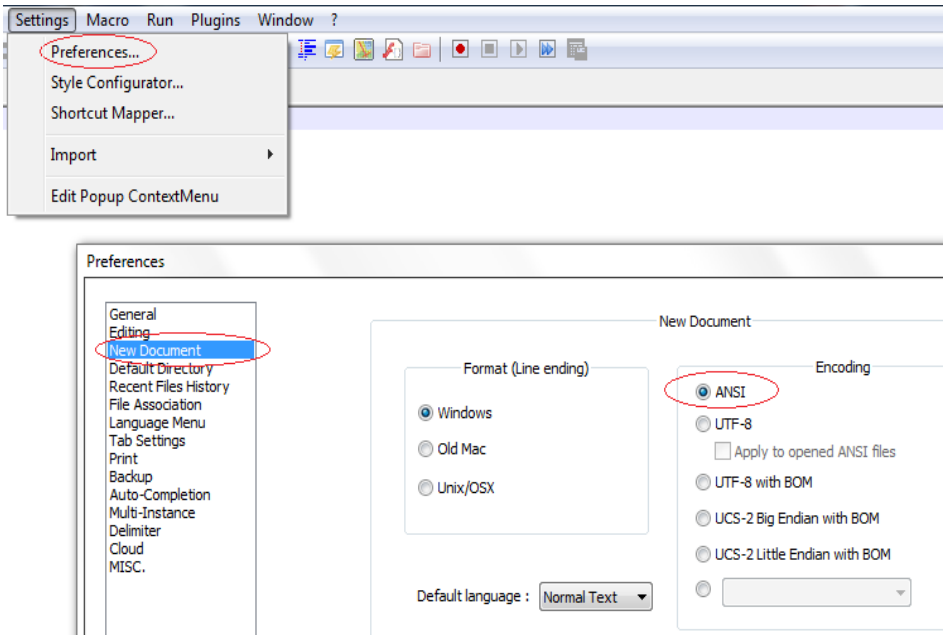
A questo punto possiamo aprire i file di progetto con Notepad++ per modificarli, salvarli e poi rigenerare l'avventura.

Attenzione però a creare, modificare e salvare tutti file di progetto in codifica ANSI e a non introdurre caratteri non ANSI perché questo causerebbe errori nella generazione del gioco.

In Notepad++ bisogna selezionare il menu *Encoding* e poi *Encode in ANSI*:

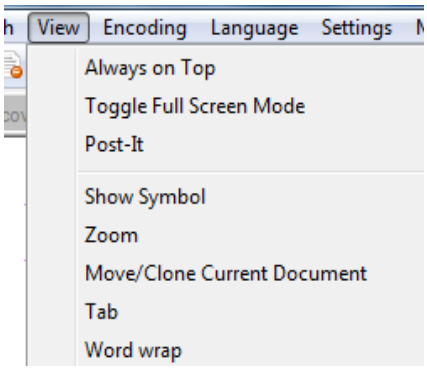


Oppure, prima di iniziare a lavorare, possiamo modificare le opzioni di Notepad++: sotto *Settings*, scegliamo *Preferences* e quindi selezioniamo *New document* e nel box Encoding *ANSI*.



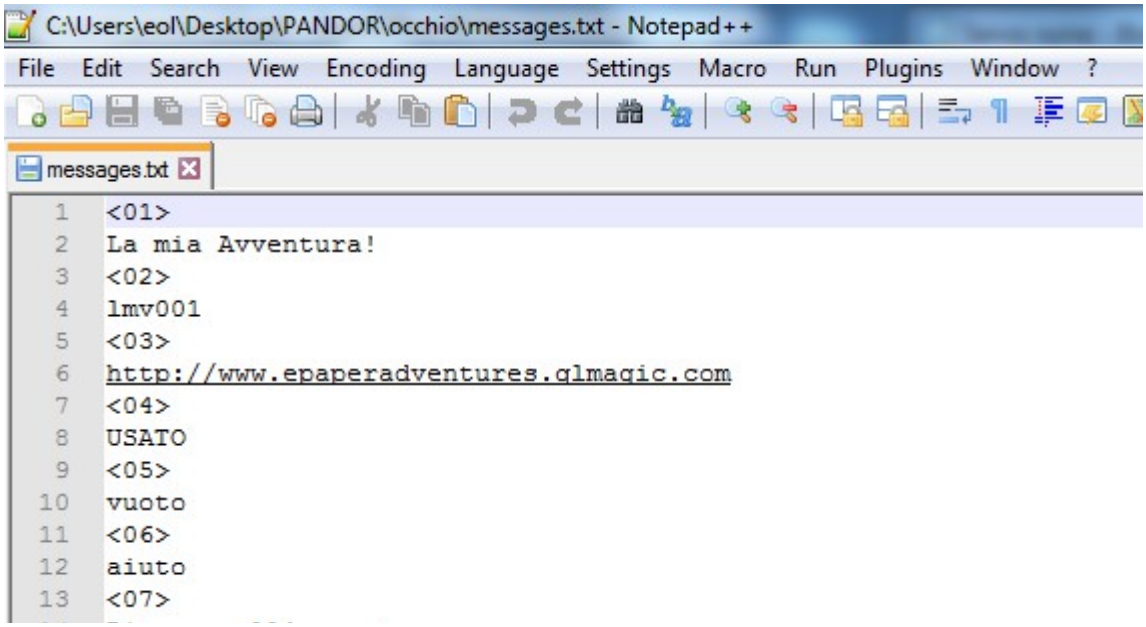
Attenzione anche a copia-incollare testo proveniente da applicazioni esterne in quanto, se presenti caratteri non ANSI, questo potrebbe modificare la codifica dell'intero file.

Un'altra opzione da disabilitare sempre è il *word wrap*, ovvero l'opzione di "A capo automatico". In Notepad++ troviamo l'opzione sotto il menu *View*. Non ci deve essere la spunta.



I file di progetto *messages.txt*

Il file *messages.txt* contiene i testi che sono utilizzati per comandi e messaggi nel gioco. Apriamo il file con Notepad++.



Le linee <*NUMERO*> vanno lasciate inalterate, mentre possiamo modificare le linee di testo sottostanti.

In *messages.txt* sono ammessi solamente i seguenti caratteri:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890 àÀáÁèÈéÉìÌíÍòÒóÓùÙúÚ

!£$%&/()=?'|*+-,;.:<>^#_
```

Se vi sono altri caratteri, Pandor+ bloccherà la generazione del gioco riportando un errore e i caratteri non ammessi:

```
Check chars in file: C:\Users\eol\Desktop\PANDOR\occhio\messages.txt
Error: Invalid character < @ >
```

Alcuni caratteri hanno un uso speciale. Vanno utilizzati solo se sono già previsti nei testi di default.

|	Questo carattere indica a Pandor+ di scegliere a caso uno dei testi separati da esso
^	Questo carattere indica "a capo"
''	Due caratteri ' consecutivi vengono trasformati in virgolette: ″

Ecco di seguito l'elenco dei testi di *messages.txt* modificabili.

```
<01>

La mia Avventura!
```

Titolo dell'avventura, che apparirà nella schermata iniziale.

```
<02>

lmv001
```

Identificatore dell'avventura, utilizzato per distinguere i salvataggi di avventure diverse. Non utilizzare spazi, ma unicamente lettere e numeri.

```
<03>

http://www.epaperadventures.qlmagic.com
```

URL del sito che si può raggiungere dal link nella schermata iniziale.

```
<04>

USATO
```

Appare negli slot di salvataggio se lo slot è stato utilizzato.

```
<05>

vuoto
```

Appare negli slot di salvataggio se lo slot non è stato utilizzato.

```
<06>

aiuto
```

Comando per visualizzare la schermata di aiuto.

```
<07>

Ritorna all'avventura...
```

Comando che da una schermata ci riporta al gioco.

```
<08>

Inizia l'avventura!
```

Comando che da una schermata ci porta all'inzio del gioco.

```
<09>

Vai al sito di E-Paper Adventures!
```

Testo del link nella schermata iniziale.

```
<11>

termina
```

Comando per uscire dal gioco.

```
<13>
carica
```

Comando per caricare un salvataggio dell'avventura.

```
<14>
salva
```

Comando per salvare la propria situazione nell'avventura.

```
<15>
Carica situazione...
```

Comando per caricare un salvataggio dell'avventura.

```
<17>
e stai indossando
```

Parte della frase utilizzata quando viene descritto l'inventario.

```
<18>
usa con
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<19>
esamina
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<20>
prendi
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<21>
lascia
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<22>
indossa
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<23>
togli
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<24>
conferma...
```

Comando nel menu del comando "termina".

```
<25>
vai
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<26>
Continua...
```

Comando nei messaggi e nelle schermate del gioco.

```
<27>
Stai indossando
```

Parte della frase utilizzata quando viene descritto l'inventario.

```
<28>
Situazione caricata!
```

Frase nel messaggio che compare quando si carica una situazione salvata in precedenza.

```
<29>
Situazione salvata!
^
ATTENZIONE: Se esci dal browser
potresti perdere questo salvataggio (dipende dal tuo lettore).
Usa l'opzione #4 per salvare in modo sicuro la tua situazione
prima di uscire dal browser!
```

Frase nel messaggio che compare quando si salva una situazione di gioco (opzioni #1,#2,#3).

```
<30>
... Non trovi niente di interessante.|... Non trovi nulla d'importante.|... Nulla di interessante.|Non hai trovato
nulla.
```

Frasi di default che possono apparire dopo un comando "esamina" su un oggetto.

```
<31>

... Hai trovato qualcosa!|... Hai scoperto qualcosa!|È comparso qualcosa!|La tua ricerca ha rivelato qualcosa...
```

Frasi di default che possono apparire dopo un comando "esamina" su un oggetto che porta a rivelare un altro oggetto.

```
<32>

Non puoi farlo, stai portando troppe cose!|Hai con te troppe cose e non riesci a farlo!|Hai troppe cose per le
mani...|Il tuo equipaggiamento è troppo ingombrante...
```

Frasi di default che possono apparire dopo un comando "prendi" su un oggetto.

```
<33>

... Sembra che non funzioni.|... Non sembra funzionare.|Proprio non sembra funzionare.|Sembra proprio che non
funzioni.
```

Frasi di default che possono apparire dopo un comando "usa con" su un oggetto.

```
<34>

L'indirizzo della pagina verrà automaticamente

salvato nella cronologia assieme alla tua situazione!

Quando vorrai riprendere da questo punto

vai nella cronologia, scegli la pagina e

quindi nel gioco tocca CARICA e #4.
```

Frase nel messaggio che compare quando si salva una situazione di gioco (opzione #4).

```
<35>

te stesso
```

Compare nel sotto-menu del comando "usa con".

```
<36>

completamento:
```

Frase che indica il completamento percentuale del gioco. Se non viene gestito il completamento, va cancellata la frase e lasciata una riga vuota e in questo modo la frase non apparirà nel gioco.

```
<38>

Stai portando con te
```

Frase che appare prima del nostro inventario.

```
<39>

Qui vedi anche
```

Frase che appare per indicare la presenza di oggetti "mobili" nell'ambiente corrente.

```
<40>
Non hai nulla con te.
```

Frase che appare quando il giocatore non ha con se nulla.

```
<43>
mappa
```

Comando nel gioco per accedere alla mappa (se disponibile).

```
<44>
parla con
```

Comando nel menu degli oggetti. Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<45>
Parlare non serve a molto ora...|Parli, ma non succede niente.|Non hai nulla da dire ora...
```

Frase di default che appare quando il giocatore parla con un oggetto. Utilizzando il carattere separatore | , possiamo indicare che vogliamo che ne venga scelta una a caso automaticamente durante il gioco.

```
<46>
E-Paper Adventures 2017
```

Informazione sull'autore del gioco nella schermata iniziale.

```
<47>
COMPLIMENTI!!!
```

Titolo della schermata di vittoria del gioco.

```
<48>
HAI FALLITO...
```

Titolo della schermata di fallimento del gioco.

```
<49>
Torna all'ultima azione...
```

Comando nella schermata di fallimento del gioco per ritornare a prima dell'ultima azione. Se non si vuole rendere disponibile questa opzione, basta cancellare la frase e lasciare una riga vuota.

```
<50>
avventura
```

Titolo del file del gioco. Utilizzare solo lettere, numeri o spazio.

Vediamo ora quali messaggi cambiare per l'avventura "L'occhio purpureo". In generale, questi saranno i testi da modificare per le vostre avventure, mentre gli altri possono essere lasciati inalterati.

```
<01>

L'Occhio Purpureo
```

Nuovo titolo dell'avventura.

```
<02>

lop001
```

Identificatore dell'avventura.

```
<36>

```

Cancelliamo "completamento" e lasciamo una riga vuota in quanto in questa semplice avventura non gestiremo il valore di completamento.

```
<49>

```

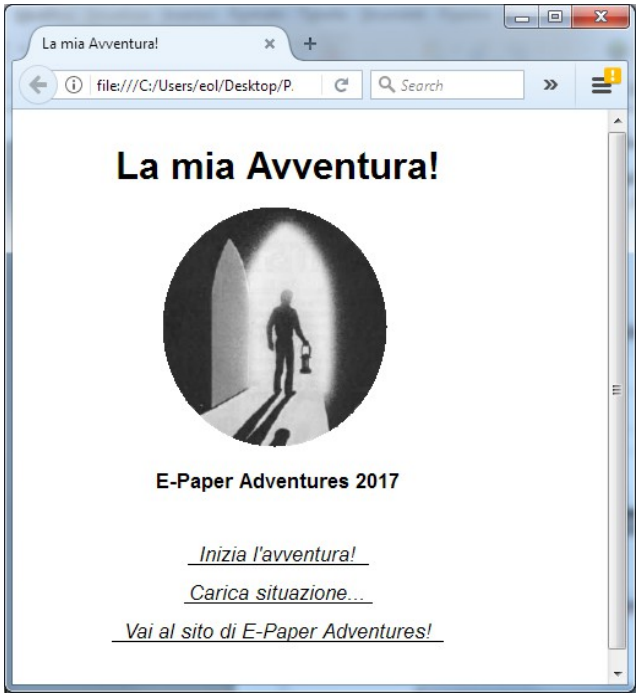
Cancelliamo "Torna all'ultima azione..." e lasciamo una riga vuota in quanto in questa semplice avventura non attiveremo questa opzione.

```
<50>

occhiopurpureo
```

Nome del file che verrà generato.

La schermata iniziale e il file di progetto *coverimage.txt*

La prima schermata del gioco è composta da alcuni testi presenti nel file *messages.txt* e da un'immagine codificata nel file *coverimage.txt*




```
<01>

La mia Avventura!
```

Titolo dell'avventura

```
<03>

http://www.epaperadventures.qlmagic.com
```

URL del sito che si può raggiungere cliccando sul link (in basso)

```
<46>

E-Paper Adventures 2017
```

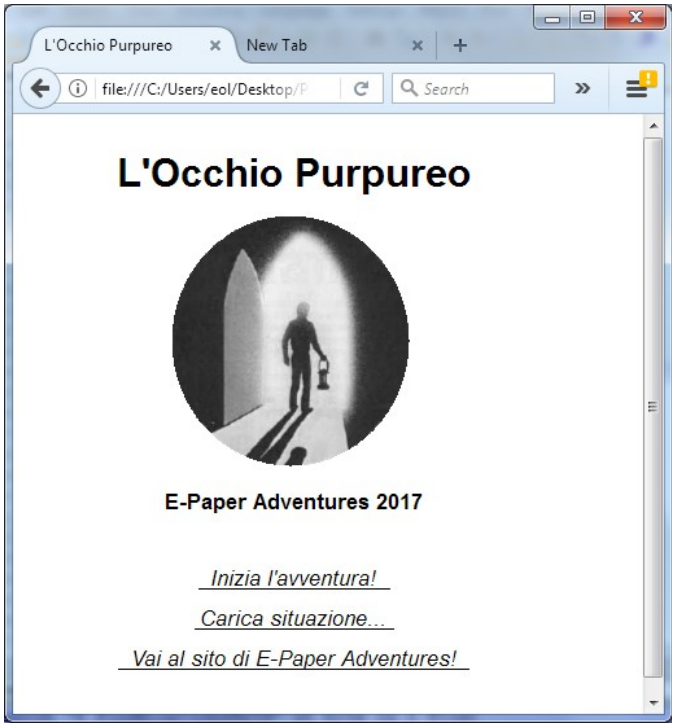
Autore del gioco

```
<09>

Vai al sito di E-Paper Adventures!
```

Testo del link in basso

La nostra unica modifica per l'avventura "L'occhio purpureo" è nel titolo:

```
<01>

L'Occhio Purpureo
```



Anche se per questa avventura utilizzeremo l'immagine di default, vediamo ora come eventualmente scegliere un'altra immagine e sostituirla.

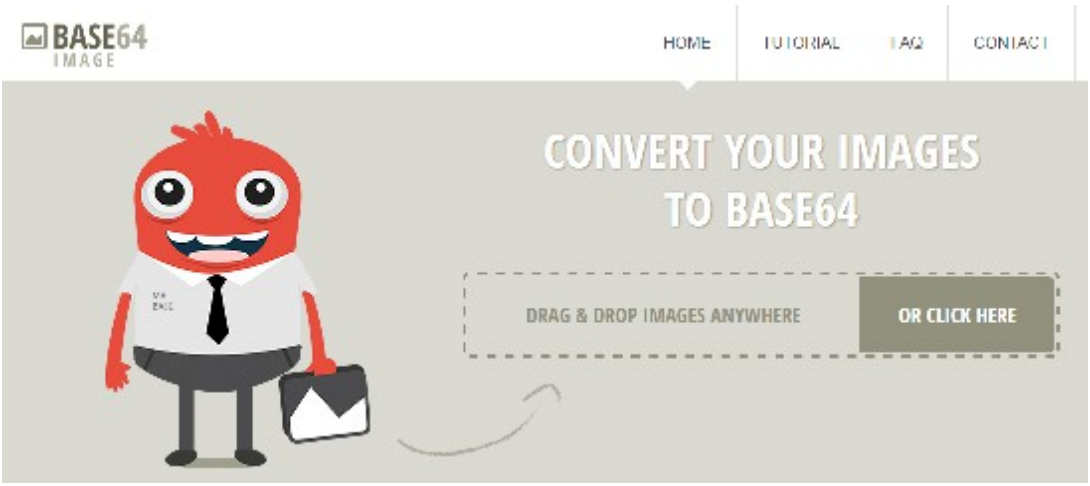
Poichè Pandor+ genera un unico file HTML che contiene tutto il gioco, le immagini vanno codificate in un modo particolare, chiamata codifica base 64. Oltre alla codifica, dovremo scegliere immagini che abbiano dimensioni opportune anche per schermi di lettori come Kindle (200x200 pixels ad esempio).

Come esempio, col browser andiamo all'indirizzo:

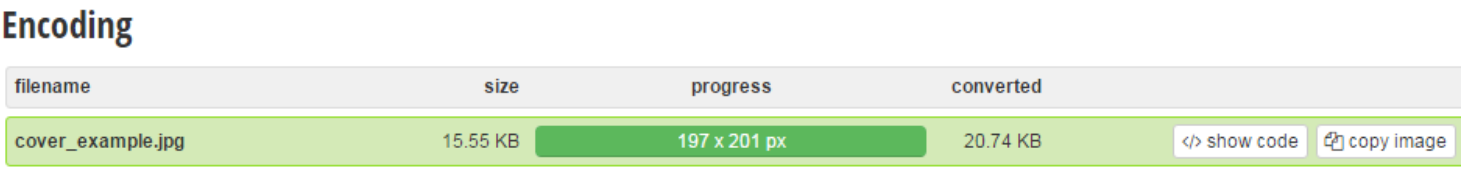
www.epaperadventures.qlmagic.com/cover_example.jpg

e salviamo l'immagine (ad esempio con tasto destro sull'immagine – salva immagine) sul nostro desktop. Questa è esattamente l'immagine che è stata usata come default, ma potrebbe essere una qualsiasi creata da voi.

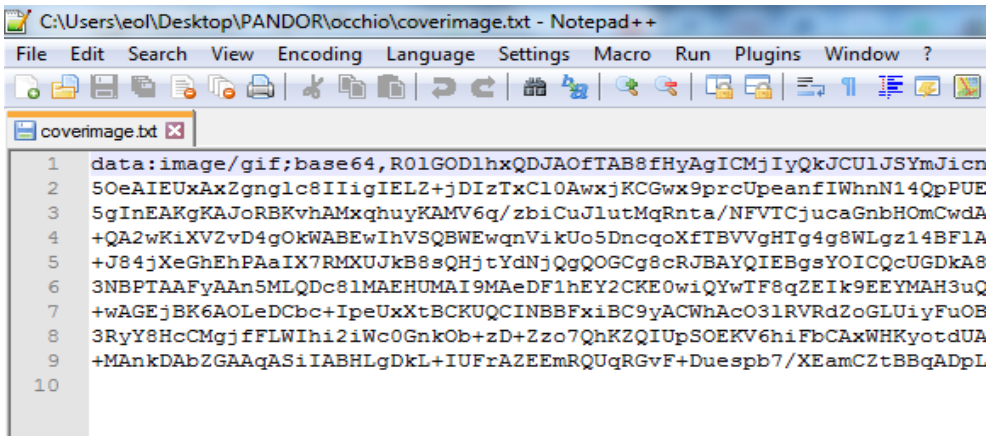
Dobbiamo ora creare la codifica in base 64. Un buon sito per farlo è: https://www.base64-image.de/



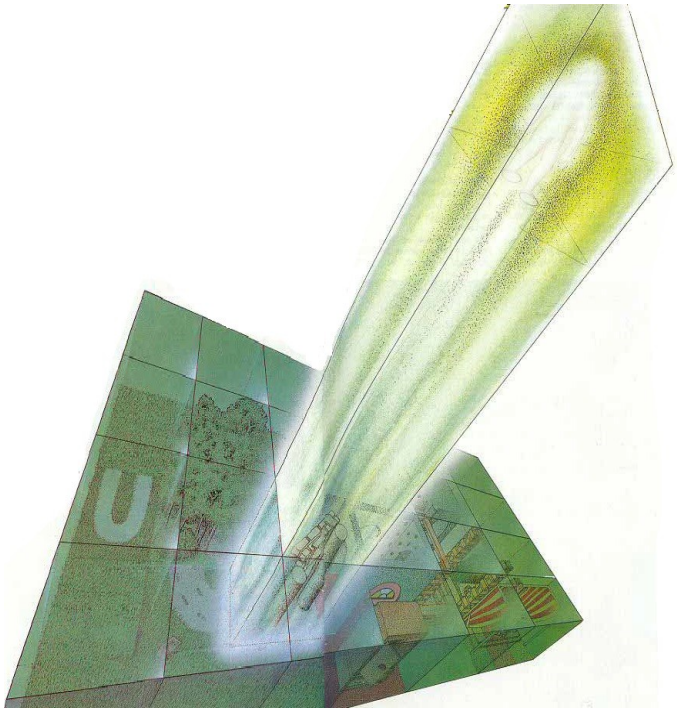
Clicchiamo su *OR CLICK HERE* e scegliamo l'immagine che abbiamo salvato sul desktop. Una volta terminata la codifica, clicchiamo su *COPY IMAGE*.

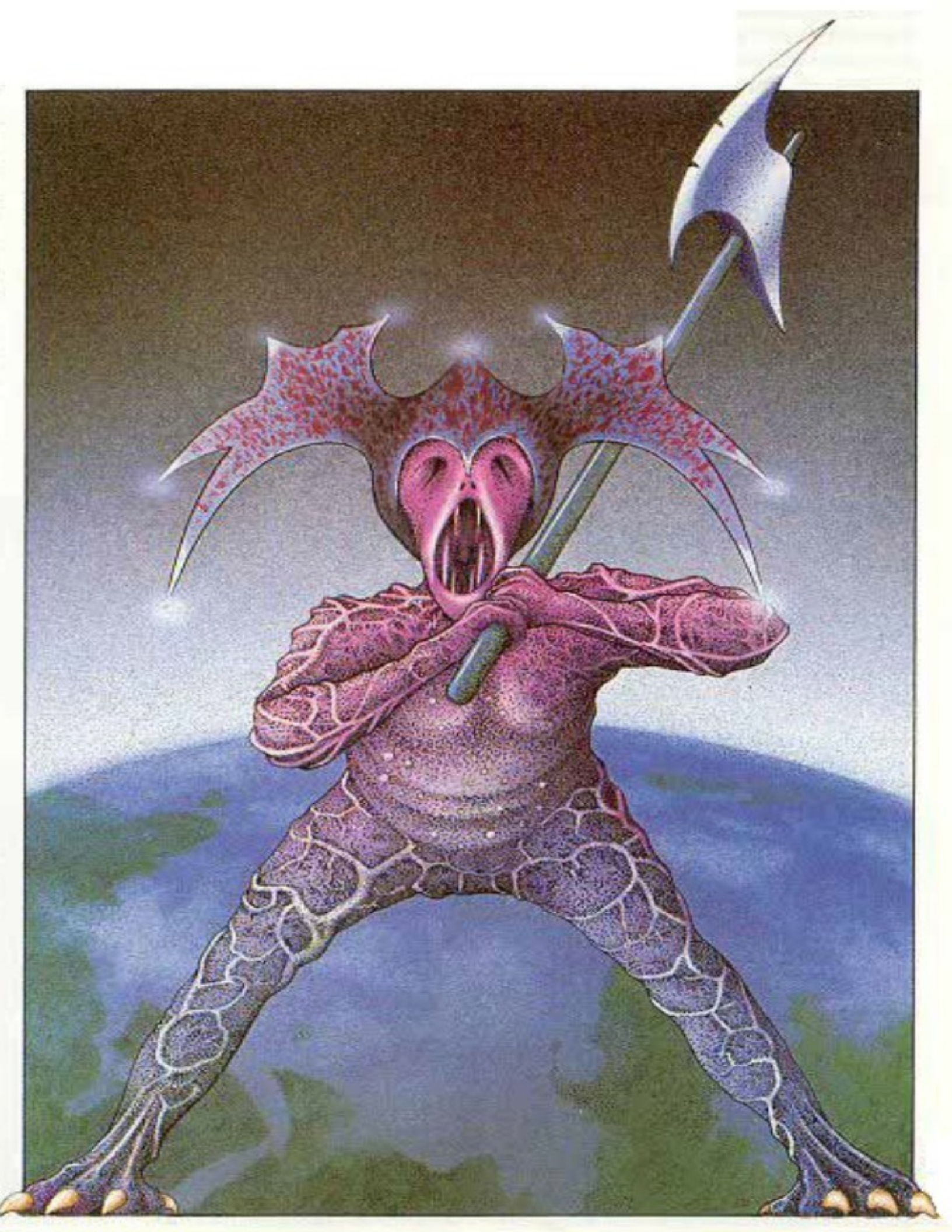


Apriamo *coverimage.txt,* cancelliamone il contenuto e incolliamo.



Ora salviamo. In questo modo, quando genereremo l'avventura la nuova immagine per la schermata iniziale sarà quella scelta e non quella di default.

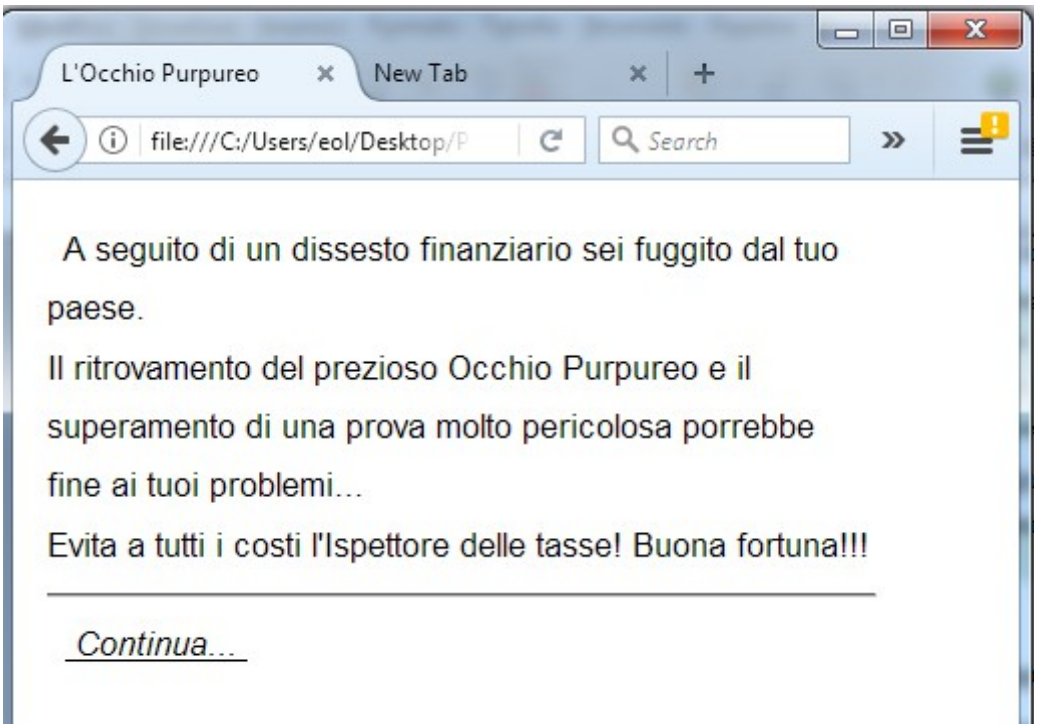




Il file di progetto *intro.txt*

Questo file contiene il testo che descrive la situazione iniziale dell'avventura, che compare dopo la schermata iniziale del gioco. Apriamolo con l'editor e modifichiamolo come segue:

```
A seguito di un dissesto finanziario sei fuggito dal tuo paese.^
Il ritrovamento del prezioso Occhio Purpureo e il superamento di una
prova molto pericolosa porrebbe fine ai tuoi problemi...^
Evita a tutti i costi l'Ispettore delle tasse! Buona fortuna!!!
```



In questo file sono ammessi solamente i seguenti caratteri:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890 àÀáÁèÈéÉìÌíÍòÒóÓùÙúÚ
!£$%&()=?'*+#-,;.:<>^§
```

Alcuni caratteri hanno un uso speciale:

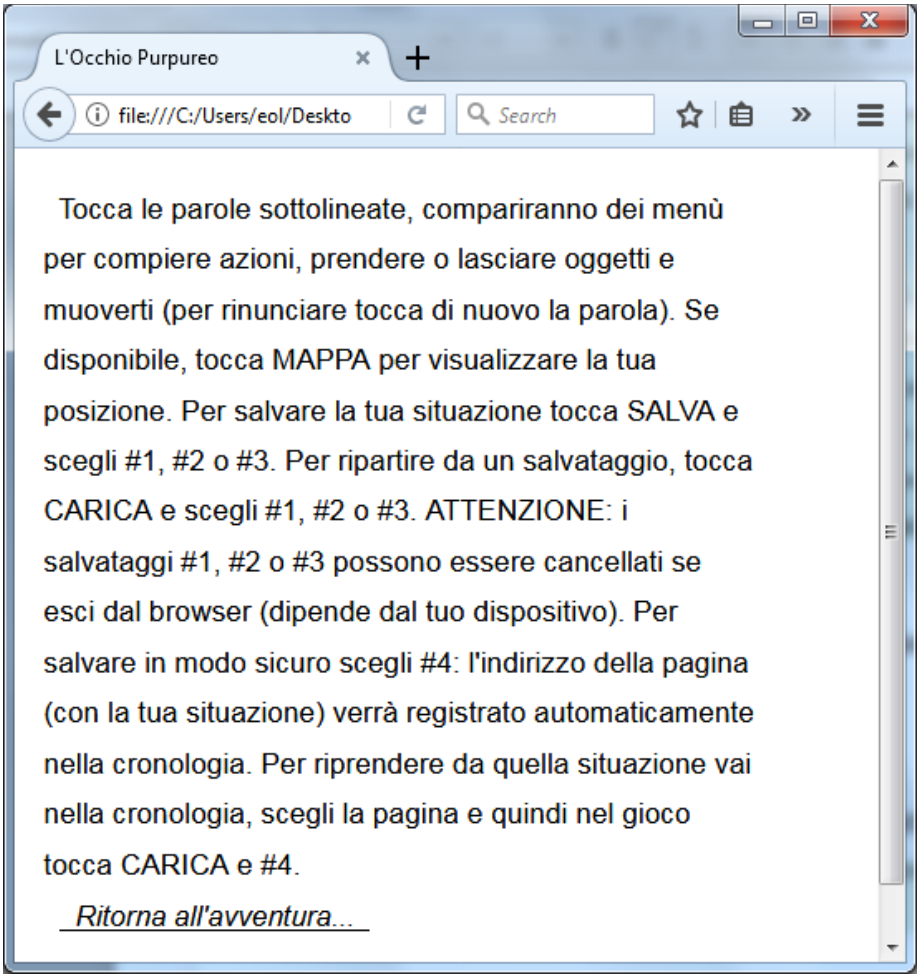
- ^ Questo carattere indica "a capo"
- '' Due caratteri ' consecutivi vengono trasformati in virgolette: ″
- § Questi carattere può essere usato per dividere il testo in più schermate. Pandor+ aggiunge in automatico il comando *Continua...* ad ogni schermata di introduzione.

Il file di progetto *help.txt*

Quando clicchiamo su ***aiuto*** nel gioco, il testo che appare è quello contenuto in *help.txt*.

```
Tocca le parole sottolineate, compariranno dei menu
per compiere azioni, prendere o lasciare oggetti e
muoverti (per rinunciare tocca di nuovo la parola).
Se disponibile, tocca MAPPA per visualizzare la tua posizione.
Per salvare la tua situazione tocca SALVA e scegli #1, #2 o #3.
Per ripartire da un salvataggio, tocca
CARICA e scegli #1, #2 o #3. ATTENZIONE: i salvataggi #1, #2 o #3
possono essere cancellati se esci dal browser (dipende dal tuo dispositivo).
Per salvare in modo sicuro scegli #4: l'indirizzo della pagina
(con la tua situazione) verrà registrato automaticamente nella
cronologia. Per riprendere da quella situazione vai nella
cronologia, scegli la pagina e quindi nel gioco tocca CARICA e #4.
```

Il contenuto del file può essere lasciato così com'è in quanto contiene già tutte le informazioni importanti.



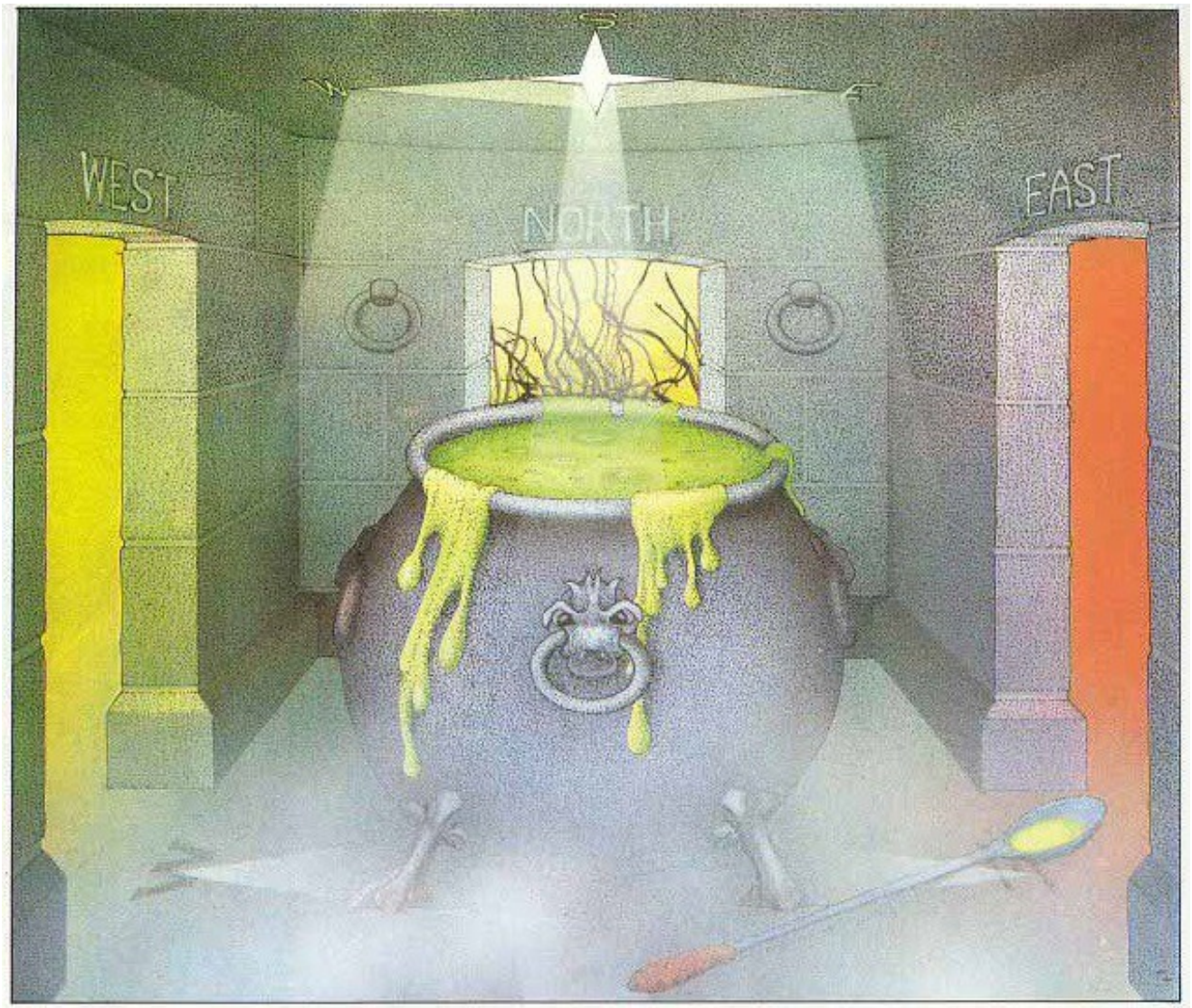
Nel file *help.txt* sono ammessi solamente i seguenti caratteri:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890 àÀáÁèÈéÉìÌíÍòÒóÓùÙúÚ
!£$%&()=?'*+#-,;.:<>
```

Alcuni caratteri hanno un uso speciale:

- ^ Questo carattere indica "a capo"
- '' Due caratteri ' consecutivi vengono trasformati in virgolette: ″

Non è prevista la possibilità di avere più schermate di aiuto.



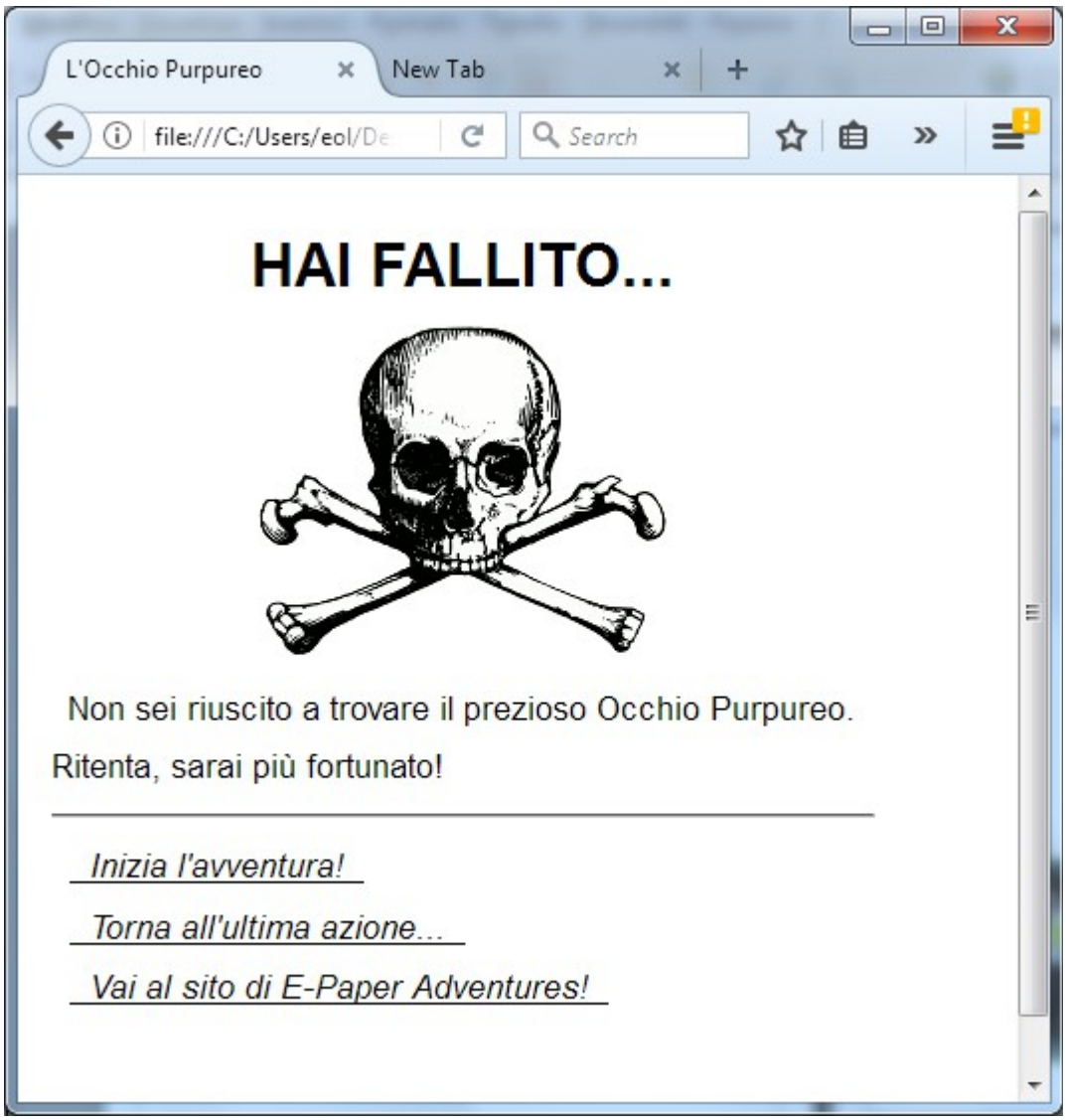
I file di progetto *success.txt*, *fail.txt*, *successimage.txt*, *failimage.txt*

Quando si conclude con successo l'avventura appare una schermata il cui contenuto si trova nel file *success.txt*. Apriamolo e modifichiamolo in questo modo:

```
Hai trovato il prezioso Occhio Purpureo!!!^
Con la sua vendita risanerai i tuoi debiti e
vivrai per sempre di rendita! Ben fatto!!!^
Ora sei finalmente ricco e ti godrai
la bella vita... FANTASTICO!!!
```

Allo stesso modo, modifichiamo *fail.txt*, il cui testo compare nella schermata di "fallimento" del gioco:

```
Non sei riuscito a trovare il prezioso Occhio Purpureo.^
Ritenta, sarai più fortunato!
```



In questi file sono ammessi solamente i seguenti caratteri:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890  àÀáÁèÈéÉìÌíÍòÒóÓùÙúÚ

!£$%&()=?'*+#-,;.:<>^

{}_
```

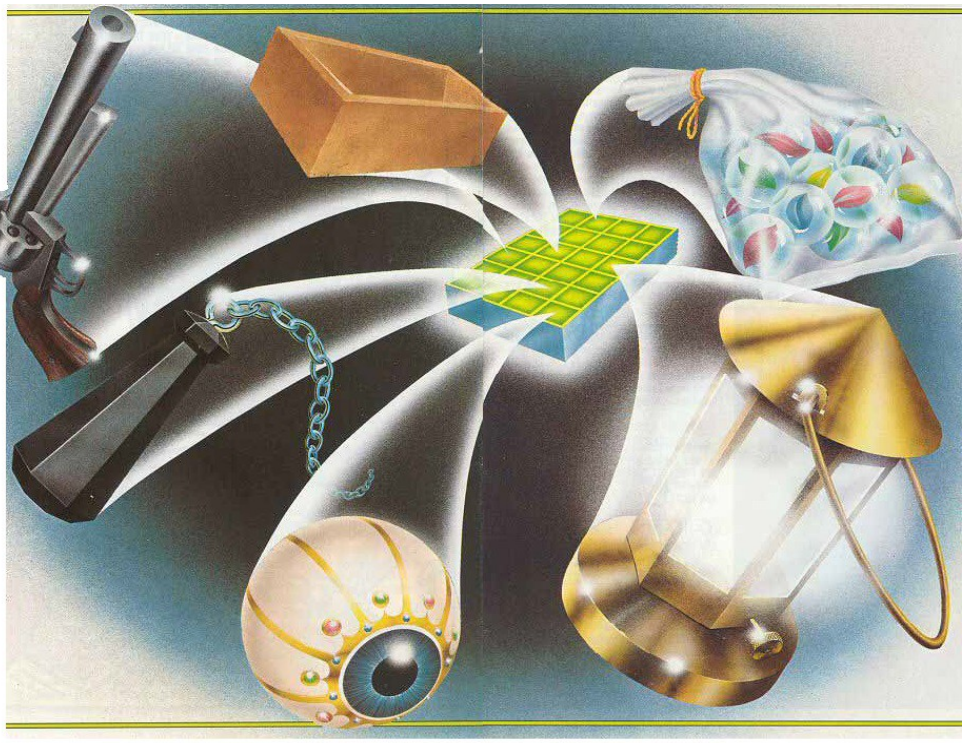
Alcuni caratteri hanno un uso speciale:

^	Questo carattere indica "a capo"
''	Due caratteri ' consecutivi vengono trasformati in virgolette: ″
_ { }	Questi tre caratteri possono essere usati per mostrare il valore della variabile relativa al completamento dell'avventura o ad un eventuale punteggio aggiuntivo. Tuttavia in questo tutorial questo argomento non verrà trattato.

Oltre ai testi sopra, nella schermata di vittoria e fallimento vengono inseriti come titolo i testi <47> e <48> presenti nel file *messages.txt*:

```
<47>

COMPLIMENTI!!!

<48>

HAI FALLITO...
```

Inoltre, tra titolo e testo, sono inserite le immagini codificate in base 64 presenti nei file *successimage.txt* e *failimage.txt;* Per questa avventura lasceremo quelle di default ma potete sostituirle come già descritto per il file *coverimage.txt*



Il file di progetto *variables.txt*

Questo file ci permette di definire le variabili che useremo per stabilire le varie condizioni e situazioni nella nostra avventura. Ad esempio se è stata fatta una certa azione oppure no, oppure quante azioni sono state compiute da un certo momento in poi. Apriamo il file e modifichiamolo in questo modo:

```
// percentuale completamento avventura
0:_vCOMPLETION_=0


// mosse effettuate
1:_vMOVES_=0


// numero di volte che bisogna tirare la catena per vincere (da 1 a 3)
2:_vTirareCatena_=0


// l'ispettore è già comparso una volta (1 = è comparso, 2 = è stato ucciso o è andato via)
3:_vStatoIspettore_=0


// numero di azioni fatte nella foresta. Se si rimane più di due azioni, si muore pietrificati
4:_vAzioniInForesta_=0


// abbiamo già nuotato o no nel fiume (1 = si)
5:_vNuotatoInFiume_=0


// contatore di azioni dal momento in cui l'ispettore compare
6:_vAzioniConIspettore_=0


// descrizione stanza buia
7:_vTestoStanzaBuia_=1
```

Le variabili sono definite da un numero e da un identificatore che deve iniziare e terminare con _ .

Gli identificatori (esempio: *_vAzioniInForesta_*) possono essere composti solo con lettere (non accentate) e numeri. Una buona idea è iniziare gli identificatori delle variabili con la lettera *v* (che sta per variable), per non confondere le variabili con gli identificatori di oggetti e stanze.

Dopo l'identificatore va definito il valore iniziale, con = NUMERO. Ad esempio, *_vNuotatoInFiume_* è inizialmente pari a 0.

Le variabili possono assumere solo valori interi che vanno da 0 a 65533. Il numero massimo di variabili è 100.

Le linee che iniziano con // sono dei commenti per ricordarci lo scopo delle variabili stesse.

Due variabili particolari sono *_vCOMPLETION_* e *_vMOVES_* e vanno sempre lasciate così come sono definite. La prima viene utilizzata per la percentuale di completamento nel gioco, che però non utilizzeremo nell'avventura.

*_vMOVES_* è invece una variabile che va utilizzata per tener conto del numero di azioni compiute dal giocatore.

Facciamo un esempio di uso delle variabili.

Supponiamo che vogliamo tener traccia se il giocatore è passato per la stanza "sala da pranzo" oppure no. Nel codice del gioco scriveremo una condizione del tipo:

SE il giocatore è nella sala da pranzo

ALLORA *_vGiocatoreInSalaDaPranzo_* = 1

In questo modo la variabile *_vGiocatoreInSalaDaPranzo_* diventa uguale a 1 quando il giocatore si trova almeno una volta nella sala da pranzo.

Ora immaginiamo che esista (in un'altra stanza) una fotografia raffigurante proprio la sala da pranzo. Se la si esamina vogliamo che compaia un messaggio diverso in base all'essere stati o no in sala da pranzo:

SE il giocatore esamina la fotografia:

SE *_vGiocatoreInSalaDaPranzo_* è uguale a 1

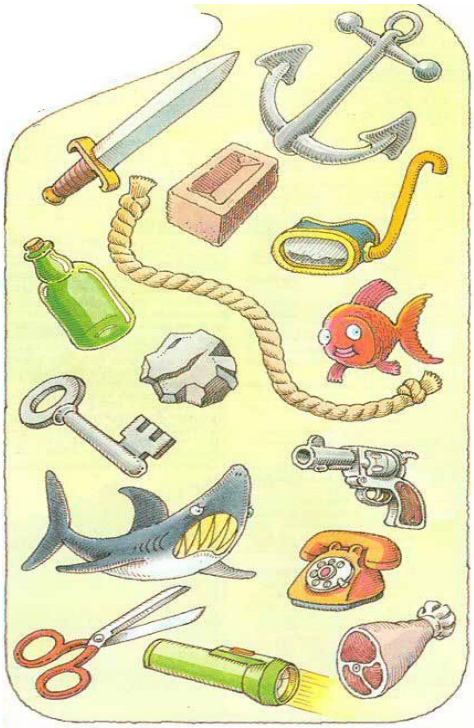
ALLORA *visualizza il messaggio "Questa fotografia mostra la sala da pranzo!"*

ALTRIMENTI *visualizza il messaggio "Questa fotografia mostra un luogo che non hai mai visto."*

In pratica il messaggio che appare nel gioco dipende dal valore di una variabile che tiene conto del fatto di essere stati (=1) o no (=0) nella sala da pranzo.

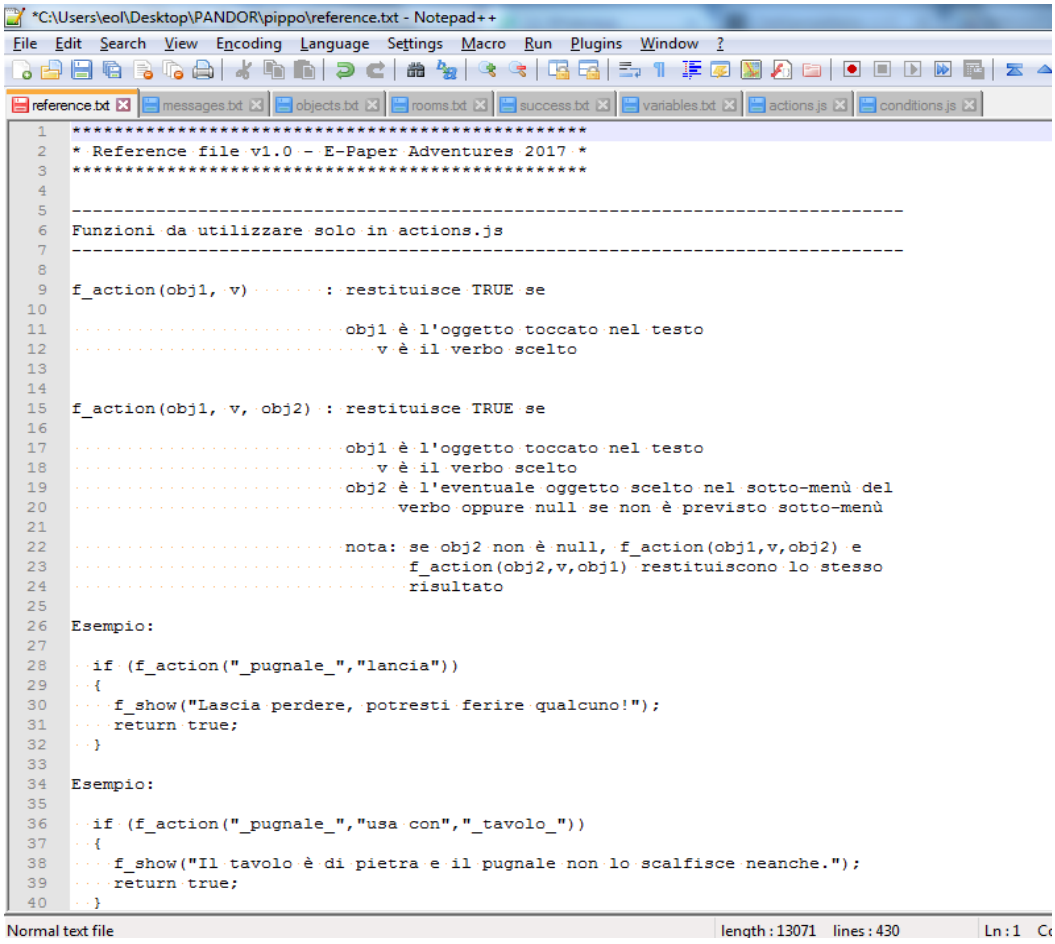
Attenzione infine che gli unici caratteri utilizzabili nel file *variables.txt* sono:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890 /=:_
```

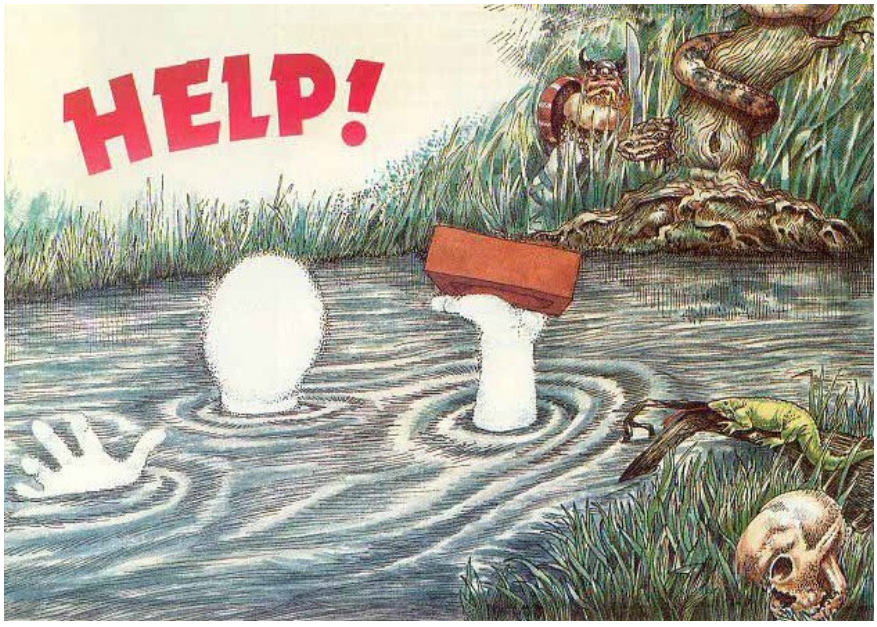


Il file *reference.txt*

Il file *reference.txt* viene generato tutte le volte che si crea un progetto. Tuttavia non è un file utilizzato da Pandor+ per generare il gioco, ma solamente una guida sintetica utile per elaborare gli altri file di progetto.



Val la pena tenerlo aperto in Notepad++ mentre lavoriamo sugli altri file.



Il file di progetto *rooms.txt*

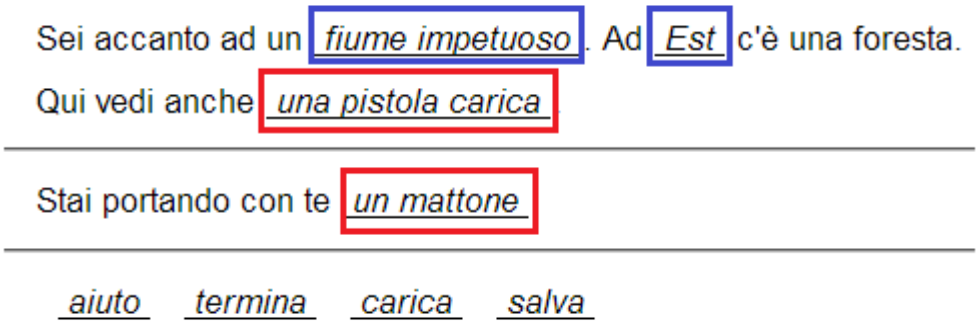
Il file *rooms.txt* contiene le descrizioni degli ambienti che compongono la nostra avventura.

Prima di vedere come va scritto, introduciamo il concetto di **oggetto**. Per Pandor+ un oggetto è tutto quello che può essere cliccato o toccato e al quale è associato un menu di azioni. Li possiamo dividere in due gruppi.

Gli **oggetti "mobili"** sono quelli che sono slegati dalle descrizioni degli ambienti. Sono oggetti che compaiono dopo la frase "Qui puoi anche vedere …" che segue la descrizione dell'ambiente corrente oppure nell'inventario. Possono essere oggetti che si possono prendere o no, persone, creature, etc.

Gli **oggetti "fissi"** sono invece oggetti che compaiono all'interno delle descrizioni degli ambienti. Di norma non sono prendibili. Possono essere invece per esempio parole legate a delle direzioni.

Vediamo come esempio la prima stanza del progetto di default di Pandor+, in cui in rosso sono qui marcati gli oggetti mobili e in blu quelli fissi.



Quindi: "fiume impetuoso" e "Est" sono oggetti fissi a cui sono collegati delle azioni che appaiono quando li tocchiamo o ci clicchiamo sopra.

Degli oggetti mobili parleremo quando vedremo il file *objects.txt*.

Vedremo ora come introdurre oggetti fissi nelle nostre descrizioni. Apriamo *rooms.txt* e vediamo il contenuto.

```
<*01>
_rStanzaBuia_
Sei in una {stanza|_uscitaStanzaBuia_} poco illuminata.
C'è una {scritta|_scrittaStanzaBuia_} su un muro.


<02>
_rGiardino_
Ti trovi in un bel giardino.
^
Da qui non te ne andrai mai più...
```

In pratica ogni descrizione di un ambiente (o stanza) comincia con una linea di tipo < *NUMERO* >. Quindi se decidiamo che nella nostra avventura ci sono 12 stanze, avremo 12 descrizioni precedute da una riga del tipo <*XX*> con XX che va da 1 a 12 (il numero massimo è 79).

Se è presente un * prima del numero, significa che quella è la stanza di partenza del gioco.

Dopo ogni riga <*NUMERO*> va inserita una riga con l'identificatore della stanza. L'identificatore deve iniziare e terminare con _ , in mezzo ai quali possiamo usare solo lettere e numeri. E' una buona regola usare come primo carattere una *r* (che sta per *room*): questo aiuterà a non confonderci con gli identificatori delle variabili e degli oggetti.

Successivamente possiamo inserire il testo della stanza.

I caratteri che possiamo utilizzare in questo file sono:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890 àÀáÁèÈéÉìÌíÍòÒóÓùÙúÚ

!£$%&()=?'*+#-,;.:<>^{}_|§
```

Alcuni caratteri hanno un uso speciale:

^	Questo carattere indica "a capo"
''	Due caratteri ' consecutivi vengono trasformati in virgolette: ″
_	Usato unicamente per gli identificatori degli oggetti
{ | }	Usato per introdurre degli oggetti nel testo della stanza
§	Usato per creare più testi per una stanza

Gli oggetti vengono inseriti nel testo in questo modo:

{ parola o frase associata all'oggetto statico | identificatore oggetto }

Quindi ad esempio nel testo della descrizione dell'ambiente 1:

```
{scritta|_scrittaStanzaBuia_}
```

è un oggetto che viene visualizzato con la parola "scritta" e il cui identificatore è *_scrittaStanzaBuia_*

Le caratteristiche degli oggetti verranno definite nel file di progetto *objects.txt* nei quali si decidono anche quali menu di azioni sono associati all'oggetto.

Il numero massimo di oggetti è 300, tuttavia i primi 9 oggetti sono riservati al motore del gioco.

Tipicamente nello sviluppo di un'avventura inserirete le descrizioni e gli oggetti un po' alla volta. Tuttavia per semplicità ora aggiungiamo invece tutte le descrizioni per l'avventura "L'Occhio Purpureo" in un colpo solo, con tanto di oggetti fissi.

Apriamo il file *rooms.txt* e modifichiamolo come segue:

<01>

rFiume

Sei accanto ad un {fiume impetuoso|_fiume01_}.

Ad {Est|_est01_|vai|_rForesta_} c'è una foresta.

<02>

rForesta

Sei in una {foresta pietrificata|_foresta02_}.

A {Sud|_sud02_|vai|_rSentiero_} scorgi un sentiero mentre

ad {Ovest|_ovest02_|vai|_rFiume_} senti il rumore di un fiume.

<03>

rSentiero

Sei su un sentiero fangoso sul quale puoi

procedere a {Nord|_nord03_|vai|_rForesta_} ed {Est|_est03_|vai|_rCancello_}.

<*04>

rCancello

Sei accanto al {cancello|_cancello04_} che in direzione Est

ti fa entrare nella Città Segreta.

Ad {Ovest|_ovest04_|vai|_rSentiero_} vedi un sentiero.

<05>

rEsternoEdificio

Sei all'esterno di un grande {edificio|_edificio05_}

situato a {Sud|_sud05_|vai|_rStanzaPolverosa_}.

<06>

rStanzaPolverosa

Sei in una stanza polverosa.

Puoi andare a {Nord|_Nord06_|vai|_rEsternoEdificio_}, {Est|_Est06_|vai|_rStanzaBuia_} e {Sud|_Sud06_|vai|
rIngresso}.

<07>

rIngresso

Sei nell'ingresso principale.

Puoi andare a {Nord|_nord07_|vai|_rStanzaPolverosa_}, {Est|_est07_|vai|_rCortile_},

{Ovest|_ovest07_|vai|_rCancello_} e {Sud|_sud07_|vai|_rStanzaMobiletto_}.

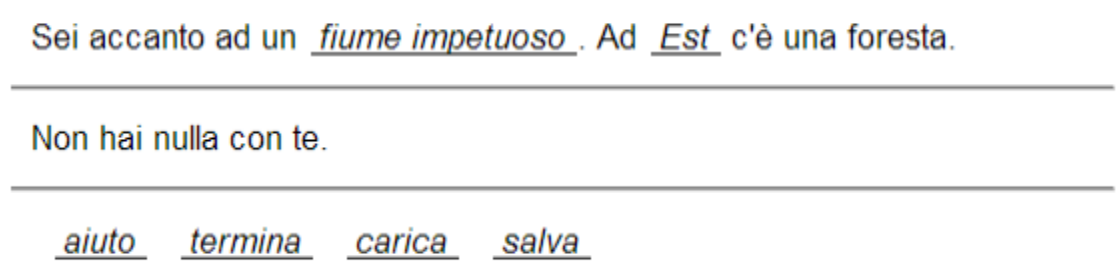
```
<08>
_rStanzaMobiletto_
Sei in una stanza con un {mobiletto|_mobiletto08_}.
A {Nord|_nord08_|vai|_rIngresso_} torni verso l'ingresso.

<09>
_rStanzaBuia_|_vTestoStanzaBuia_
Sei in una stanza buia.
Non riesci a vedere un'uscita!
§
Sei in una stanza buia.
Per tua fortuna la lampada illumina due uscite ad {Ovest|_ovest09_|vai|_rStanzaPolverosa_} e
a {Sud|_sud09_|vai|_rCortile_}.

<10>
_rCortile_
Sei nel cortile. Ti puoi muovere a {Nord|_nord10_|vai|_rStanzaBuia_},
{Est|_est10_|vai|_rGiardino_} ed {Ovest|_ovest10_|vai|_rIngresso_}.

<11>
_rGiardino_
Sei nel giardino. Da qui puoi proseguire ad {Ovest|_ovest11_|vai|_rCortile_} e {Sud|_sud11_|vai|_rSalaTrono_}.

<12>
_rSalaTrono_
Sei nella sala del {trono|_trono12_}.
Dal soffitto pende una {catena|_catena12_}.
A {Nord|_nord12_|vai|_rGiardino_} esci verso un grande giardino.
```

Guardiamo la prima descrizione:

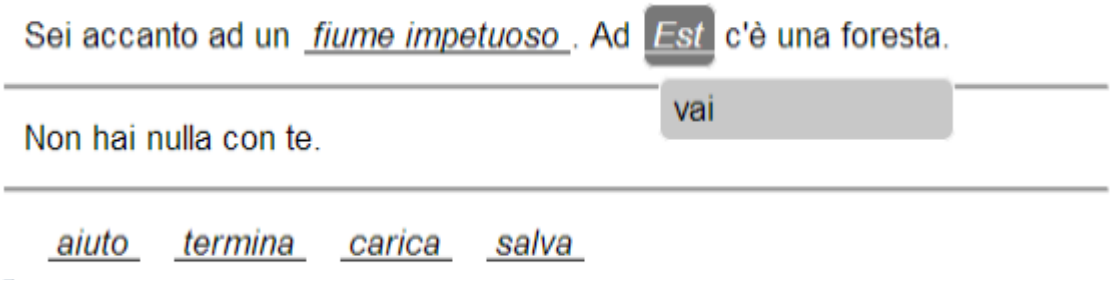
```
<01>
_rFiume_
Sei accanto ad un {fiume impetuoso|_fiume01_}.
Ad {Est|_est01_|vai|_rForesta_} c'è una foresta.
```

Ecco come appare la descrizione nel browser:

L'oggetto *_fiume01_* viene immesso nel testo e la sua definizione completa avverrà nel file *objects.txt*



L'oggetto *Est ,* identificato con *_est01_* , ha una definizione particolare. Infatti in questo caso viene introdotta direttamente una *azione di direzione*. Infatti toccando l'oggetto comparirà *vai* e se toccheremo *vai* il giocatore verrà spostato direttamente nella stanza *_rForesta_*.



Quando utilizziamo questa scorciatoia, non sarà possibile definire altri comportamenti per l'oggetto nel file *objects.txt* .

Un'altra cosa da notare è il fatto che l'identificatore termina con il numero 01. Questa è semplicemente una buona regola per ricordarsi in che stanza è definito l'oggetto, in modo ad esempio da distinguerlo da altri "est" presenti in altre stanze.

Vediamo ora la stanza 9:

```
<09>
_rStanzaBuia_|_vTestoStanzaBuia_
Sei in una stanza buia.
Non riesci a vedere un'uscita!
§
Sei in una stanza buia.
Per tua fortuna la lampada illumina due uscite ad {Ovest|_ovest09_|vai|_rStanzaPolverosa_} e
a {Sud|_sud09_|vai|_rCortile_}.
```

La prima cosa da notare è che la seconda riga presenta l'identificatore *_rStanzaBuia_* seguito da | e poi dall'identificatore della variabile *_vTestoStanzaBuia_*.

Questo serve per dire a Pandor+ che la descrizione della stanza dipende dal valore della variabile *_vTestoStanzaBuia_* .

Se questa variabile è 1, il testo mostrato sarà il primo:

```
Sei in una stanza buia.

Non riesci a vedere un'uscita!
```

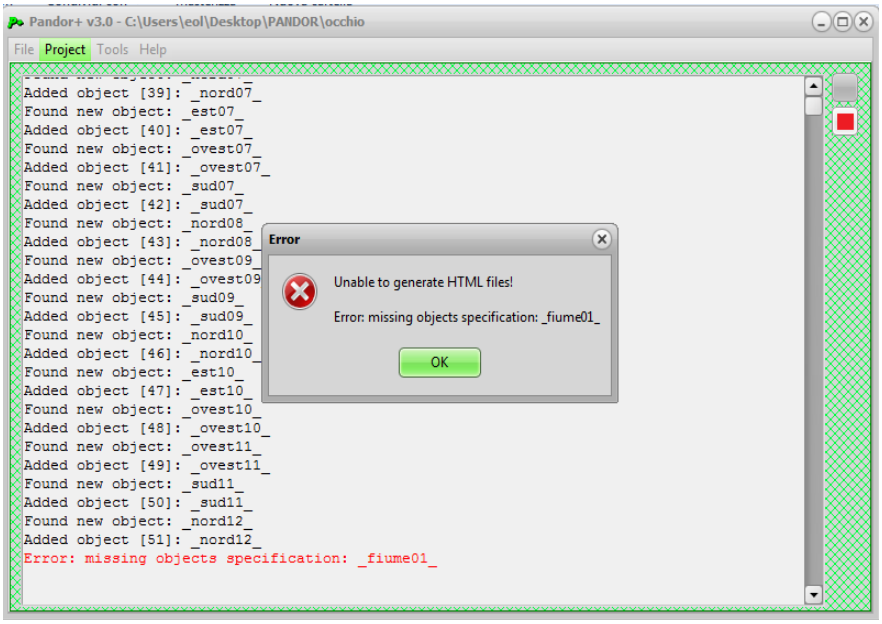
Se è 2, il testo sarà il secondo:

```
Sei in una stanza buia.

Per tua fortuna la lampada illumina due uscite ad {Ovest|_ovest09_|vai|_rStanzaPolverosa_} e

a {Sud|_sud09_|vai|_rCortile_}.
```

I testi sono separati col carattere speciale § . Si possono al massimo definire 5 testi per una stanza.

La variabile *_vTestoStanzaBuia_* verrà modificata nel file *conditions.js* che vedremo più avanti, in base al fatto che il giocatore abbia con se la lampada accesa o no.

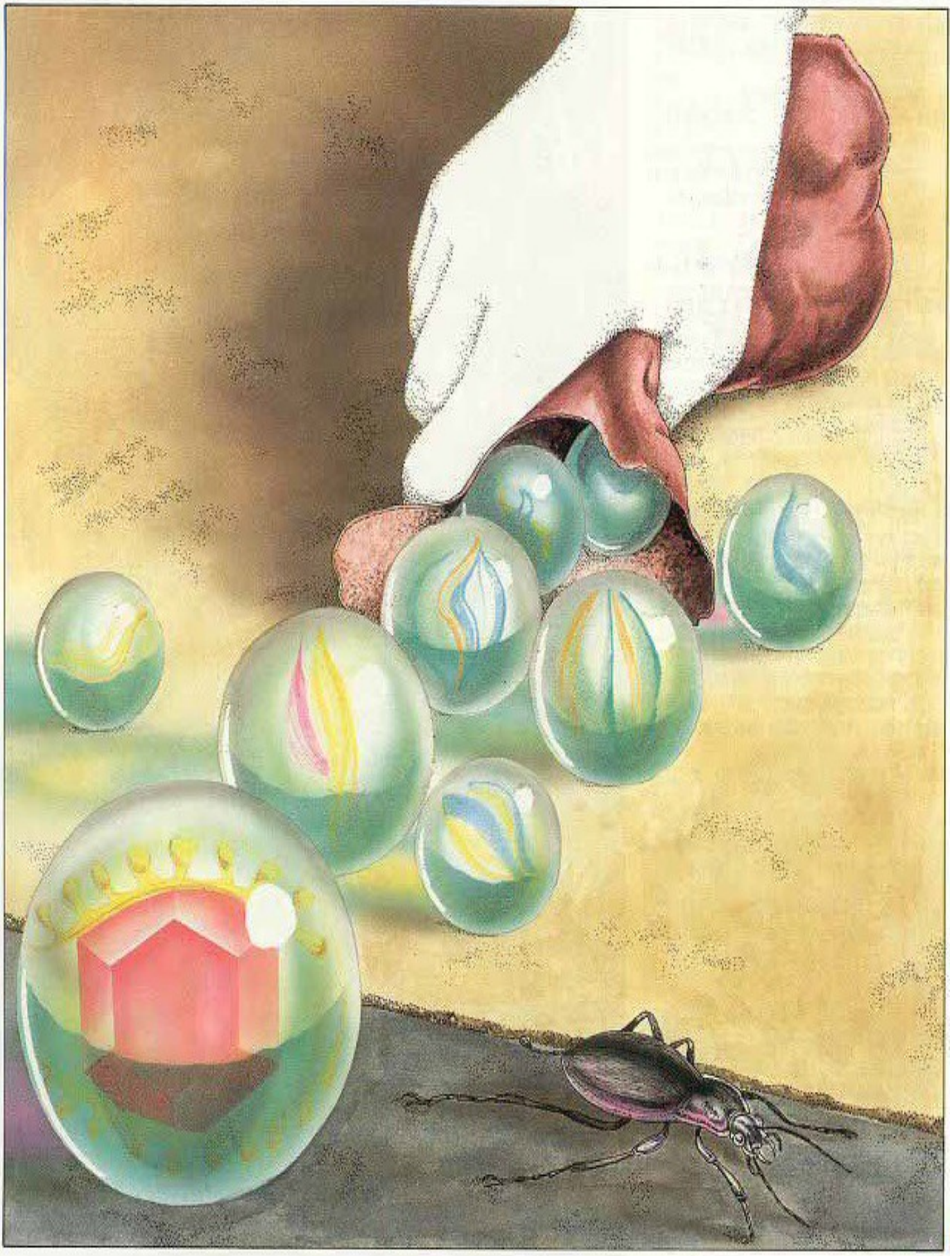
Modificato e salvato il file, proviamo a generare l'avventura con Pandor+. Verrà visualizzato un errore, relativo al fatto che l'oggetto *_fiume01_* non è definito nel file *objects.txt*:



Nota: se stavamo già testando in precedenza il gioco sul nostro browser, dobbiamo sempre utilizzare l'opzione *termina* (e conferma) per riportarci alla schermata iniziale e poi effettuare il refresh della pagina (tasto F5).

Un ultima nota: se ci serve aggiungere dei nostri commenti nel file *rooms.txt*, lo possiamo fare utilizzando delle righe che iniziano con // , ad esempio:

```
<12>

Sei nella sala del {trono|_trono12_}.

// ricordarsi di aggiungere la frase che il pavimento scricchiola!

Dal soffitto pende una {catena|_catena12_}.
```



Inventario e azioni "usa con"

Prima di continuare con gli altri file di progetto è utile rivedere il funzionamento dell'inventario.

Nel corso dell'avventura si possono trovare oggetti:

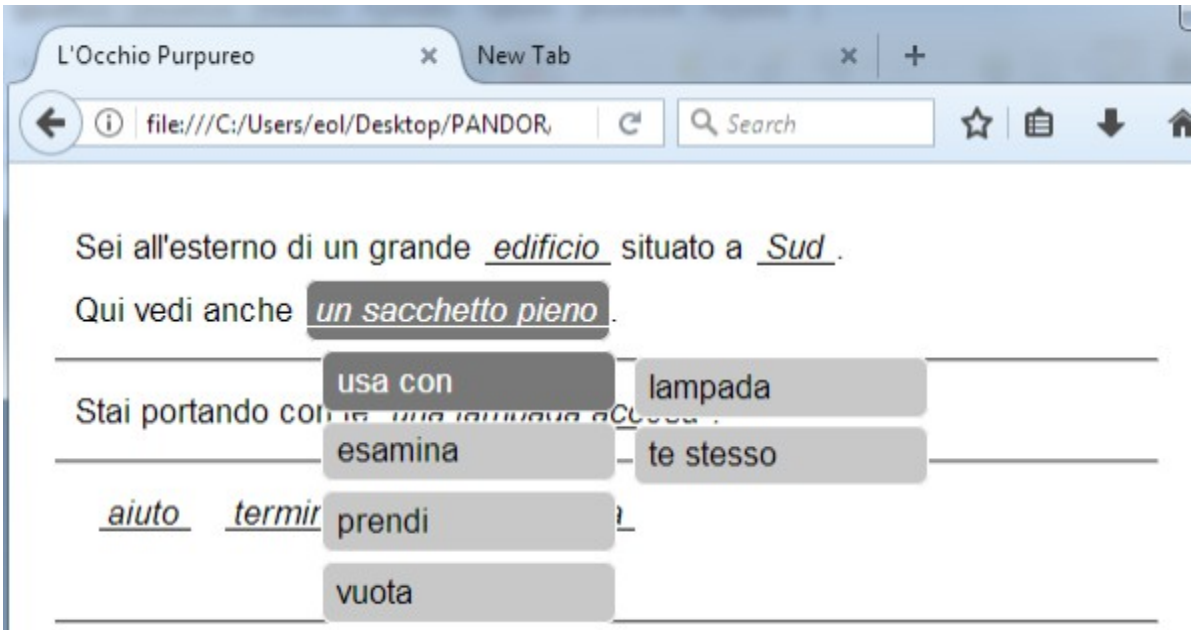
- non prendibili (casa, nave, albero...)
- prendibili (lampada, moneta, mattone...)
- prendibili e indossabili (armatura, zaino, guanti...)

Gli oggetti hanno un peso (ingombro) diverso se portati o indossati. Di norma conviene indossare un oggetto se possibile, in quanto il suo ingombro in questo caso diminuirà.

Nel momento in cui tentiamo di prendere/indossare un oggetto verrà calcolato l'ingombro totale trasportato al momento e verrà sommato l'ingombro dell'oggetto: l'azione ha successo se la somma è inferiore o uguale a *100*.

Un controllo simile viene fatto quando tentiamo di togliere un oggetto indossato, che quindi modificherebbe l'ingombro totale trasportato. Al caso, dobbiamo prima lasciare altri oggetti prima di poterlo togliere.

Gli oggetti possono essere utilizzati in combinazione tramite l'azione "usa con". Gli oggetti che compaiono nel sotto-menu dell'azione "usa con" sono di default solamente quelli presenti nell'inventario. Inoltre, come ultima voce, è sempre disponibile "te stesso", in modo da poter compiere azioni come "usa pozione con te stesso" e simili.



Vedremo comunque un esempio di come creare sotto-menu con le voci che desideriamo.

Il file di progetto *objects.txt*

Dopo averli introdotti nelle descrizioni, gli oggetti fissi vanno definiti in *objects.txt*. Allo stesso modo aggiungeremo gli oggetti mobili, specificandone le caratteristiche. Ogni oggetto va definito con una sequenza di linee, in questo modo:

```
<identificatore oggetto>
:takeable wearable examinable usable talkable
:direction: stanza raggiunta in caso di azione di movimento
:desc_direction: descrizione azione di movimento
:desc_short: descrizione breve
:desc_long: descrizione lunga
:room: posizione oggetto
:weight: peso se trasportato
:weight_worn: peso se indossato
:tag: valore intero libero
:desc_examine: messaggio da mostrare se esaminato
:found: identificatore oggetto che compare automaticamente a seguito di esamina
:desc_found: messaggio di scoperta del nuovo oggetto specificato in found
:desc_talk: messaggio da mostrare la prima volta se si parla con questo oggetto
:desc_next_talk: messaggio da mostrare dalla seconda volta in poi se si parla con questo oggetto
:menu: voci del menu dell'oggetto aggiuntive (oltre a quelle automatiche)
```

Fondamentalmente specificheremo per ogni oggetto solo le proprietà che ci interessano. Alcune permettono di avere dei comportamenti "automatici":

- aggiungere al menu di azioni dell'oggetto l'azione "esamina" e definire il messaggio che appare a seguito dell'azione. Si può anche specificare che a seguito dell'azione compaia un nuovo oggetto (o più oggetti), indicando quale altro messaggio deve apparire
- aggiungere al menu di azioni dell'oggetto l'azione "usa con" con sotto-menu la lista di oggetti dell'inventario
- aggiungere al menu di azioni dell'oggetto un comando di movimento, indicando in quale ambiente porta il giocatore
- aggiungere al menu di azioni dell'oggetto l'azione "parla con" e definire il messaggio che appare a seguito dell'azione la prima volta e le volte successive alla prima

Vediamo ora le righe e proprietà una alla volta.

```
<identificatore_oggetto>
```

Specifica quale oggetto viene definito, ad esempio: <*_lampada01_*>. Per l'identificatore vanno usati solo lettere e numeri e il primo e ultimo carattere deve essere: _

Conviene introdurre nell'identificatore l'indice della stanza nella quale si trova l'oggetto, per evitare doppioni di oggetti simili in stanze diverse.

```
:takeable wearable examinable usable talkable
```

Nella seconda linea va specificato se l'oggetto ha alcune caratteristiche o no:

- Se è presente la parola *takeable* l'oggetto è prendibile (va usata quindi solo per oggetti mobili) e il suo peso è specificato in *weight*. Viene aggiunta in automatico l'azione "prendi" (o "lascia") nel menu dell'oggetto.
- Se è presente la parola *wearable* l'oggetto è indossabile (va usata quindi solo per oggetti mobili con la proprietà "takeable") e il suo peso se indossato è specificato in *weight_worn*. Viene aggiunta in automatico l'azione "indossa" (o "togli") nel menu dell'oggetto.
- se è presente la parola *examinable*, verrà automaticamente aggiunta al menu dell'oggetto l'azione "esamina". A seguito dell'azione, verrà mostrato il messaggio definito nella proprietà *desc_examine* ed eventualmente comparirà l'oggetto specificato in *found*, con relativo messaggio in *desc_found*.
- se è presente la parola *usable*, verrà automaticamente aggiunta al menu dell'oggetto l'azione "usa con" con sotto-menu tutti gli oggetti che abbiamo nell'inventario e "te stesso".
- se è presente la parola *talkable*, verrà automaticamente aggiunta al menu dell'oggetto l'azione "parla con". Il messaggio che apparirà la prima volta che l'azione viene eseguita è specificato in *desc_talk*; se vogliamo che le volte successive il messaggio sia diverso, va valorizzata la proprietà *desc_next_talk.*

L'idea di base è quindi che le proprietà `direction examinable takeable wearable talkable` permettono di creare oggetti di cui non dovremo aggiungere "manualmente" menu o gestire azioni e comportamento.

La proprietà `usable` invece fa si che venga aggiunto automaticamente al menu la voce "usa con" con la lista degli oggetti dell'inventario. Le possibili conseguenze delle interazioni tra oggetti andranno invece specificate "manualmente" in *actions.js*.

```
:direction:
```

Se vogliamo aggiungere al menu dell'oggetto un azione di movimento che porta in un'altra stanza possiamo valorizzare questa proprietà con l'identificatore della stanza. Ad esempio: *_rForesta_* . Non va di norma utilizzata per oggetti mobili.

```
:desc_direction:
```

Se è stata specificata *direction*, qui va specificata l'azione che vogliamo compaia nel menu. Ad esempio: *entra*. Se non viene specificato nulla, di default verrà associata l'azione "vai" (testo <25> di *messages.txt*).

```
:desc_short:
```

E' la descrizione che appare nei menu, ad esempio: *lampada*. Questa proprietà va valorizzata per gli oggetti mobili mentre non va specificata nel caso di oggetti fissi.

```
:desc_long:
```

E' la descrizione che appare nell'inventario e dopo l'indicazione *"Puoi anche vedere ..."*, ad esempio: *una vecchia lampada ad olio*. Questa proprietà va valorizzata per gli oggetti mobili mentre non va specificata nel caso di oggetti fissi.

```
:room:
```

Qui va specificata la stanza in cui si trova l'oggetto nel caso sia un oggetto mobile. Ad esempio: *_rCantina_* . Nel caso in cui l'oggetto non sia in nessuna stanza, va specificato *_rOUT_*. Nel caso in cui sia nell'inventario va messo *_rTAKEN_*, se è indossato *_rWORN_* (in questo caso l'oggetto deve avere la proprietà *wearable*).

```
:weight:
```

E' il peso dell'oggetto. Ad esempio: 10 . Va specificato solo se l'oggetto ha la proprietà *takeable.*

```
:weight_worn:
```

E' il peso dell'oggetto se indossato. Di solito sarà inferiore al peso se non indossato. Ad esempio: 5 . Va specificato solo se l'oggetto ha la proprietà *wearable.*

```
:tag:
```

E' un valore intero, da 0 a 65533. Rimane fisso nel corso di tutta l'avventura. Può essere usato per "marcare" gli oggetti secondo le nostre esigenze.

```
:desc_examine:
```

Se è selezionata la proprietà *examinable* nella seconda linea, questo è il messaggio che verrà mostrato a seguito dell'azione "esamina". Ad esempio: *E' una vecchia lampada rotta.* Se non viene specificata questa proprietà verrà visualizzato un messaggio di default tipo "Non trovi nulla di interessante" (testo <30> di *messages.txt*).

Una nota importante, che riguarda anche *desc_found, desc_talk, desc_next_talk* è che si può fare in modo di separare il testo in più messaggi (intervallati dal comando Continua) tramite il carattere §. Ad esempio:

```
:desc_examine:

E' una foresta molto sinistra...

§

Non resterei molto qui se fossi in te!
```

Un'altra possibilità è far si che il testo venga selezionato a caso da Pandor+ tra più testi separati da |

Ad esempio:

```
:desc_examine:

E' una foresta molto sinistra...|E' una foresta minacciosa...|Tutto sembra orribile qui...
```

Nel caso di utilizzo contemporaneo di | e §, la priorità va a §.

```
:found:
```

Se decidiamo che a seguito dell'azione "esamina" deve comparire un nuovo oggetto mobile, qui va specificato l'identificatore del nuovo oggetto. Ad esempio: *_moneta00_* . L'oggetto ovviamente comparirà un'unica volta nella stanza in cui si trova il giocatore. Nel caso volessimo far apparire più oggetti, vanno elencati separati da "|", ad esempio: *_moneta00_|_spada00_*

```
:desc_found:
```

Se è stato specificata la proprietà *found*, qui scriveremo il messaggio aggiuntivo che compare (un'unica volta) all'azione "esamina". Se non viene specificata questa proprietà, a seguito dell'azione "esamina", verrà visualizzato un messaggio di default tipo "Hai trovato qualcosa" (testo <31> di *messages.txt*).

```
:desc_talk:
```

Se è selezionata la proprietà *talkable* nella seconda linea, questo è il messaggio che verrà mostrato a seguito dell'azione "parla con". Ad esempio: *Il vecchio ti dice <<Buona fortuna !>>* . Se non viene specificata questa proprietà, a seguito dell'azione "parla con", verrà visualizzato un messaggio di default tipo "Parli, ma non succede niente" (testo <45> di *messages.txt*).

```
:desc_next_talk:
```

Se è selezionata la proprietà *talkable* nella seconda linea, questo è il messaggio che verrà mostrato a seguito dell'azione "parla con", dalla seconda volta in poi. Ad esempio: *Il vecchio non ha più nulla da dirti.*

```
:menu:
```

Qui possiamo aggiungere le voci del menu che compaiono quando si tocca l'oggetto. Vanno aggiunte solo le voci che non sono già automaticamente presenti avendo inserito le proprietà *`direction examinable takeable wearable talkable usable`*.

Per aggiungere altre voci inseriamo delle righe del tipo:

```
++"azione";
```

Ad esempio, immaginando di considerare un oggetto "porta", per aggiungere due azioni "apri" e "chiudi":

```
:menu:
++"apri";
++"chiudi";
```

Per aggiungere invece delle voci di menu con delle sotto-voci (come succede per "usa con"), vanno inserite delle righe in questo modo:

```
-"sotto-voce-1";
-"sotto-voce-2";
...
++"azione";
```

Ad esempio, per aggiungere anche l'azione "bussa" e come sotto-voci "una volta", "due volte","tre volte":

```
:menu:
++"apri";
++"chiudi";

-"una volta";
-"due volte";
-"tre volte";
++"bussa";
```

Le sotto-voci vanno quindi specificate prima dell'azione.

La proprietà *menu* permette anche di scrivere del codice per determinare le condizioni in cui far apparire certe azioni o no.

Immaginiamo ad esempio di voler far apparire "apri" e "chiudi" in base allo stato della porta. Immaginiamo che sia la variabile *_vStatoPorta_* a contenere questo stato: 0 = la porta è chiusa, 1 = la porta è aperta. Ecco il codice per questo oggetto:

```
:menu:
if ( _vStatoPorta_ == 0 )
{
  ++"apri";
}
else
{
  ++"chiudi";
}

-"una volta";
-"due volte";
-"tre volte";
++"bussa";
```

Otterremo che "apri" venga mostrato solo se la porta è chiusa.

Il costrutto "if … else…" è quello che useremo di più in assoluto nel codice delle nostre avventure. In pratica ci chiediamo se una certa condizione è verificata o no: se si verifica, viene eseguito il blocco di istruzioni immediatamente dopo la condizione, altrimenti vieni eseguito il blocco presente nell'eventuale clausola "else".

```
if  (a==b)
{
       // questo blocco viene eseguito se a e b sono uguali!
}
else
{
       // questo blocco viene eseguito in caso contrario, cioè a e b non sono uguali
}
```

Naturalmente si possono creare condizioni più complesse, combinando più sotto-condizioni e usando i connettivi logici AND (&&) e OR (||):

```
if ((a==b) && (c==d))
{
	// eseguo questo blocco se a e b sono uguali E (AND) c e d sono uguali
	// (entrambe le sotto-condizioni si verificano)
}

if ((a==b) || (c==d))
{
	// eseguo questo blocco se a e b sono uguali OPPURE (OR) c e d sono uguali
	// (una o entrambe le sotto-condizioni si verificano)
}
```

Come si può notare il blocco "else" non è obbligatorio.

Aggiungiamo alla nostra porta ancora una voce "sfascia", ma solo se il giocatore ha con se un oggetto "scure", che immaginiamo avere identificatore *_scure_*

```
:menu:
if ( _vStatoPorta_ == 0 )
{
  ++"apri";
}
else
{
  ++"chiudi";
}

-"una volta";
-"due volte";
-"tre volte";
++"bussa";

if ( f_istaken("_scure_") == true )
{
  ++"sfascia";
}
```

f_istaken è una funzione di Pandor+ che permette di sapere se un certo oggetto è stato preso dal giocatore. La funzione restituisce *true* o *false*, cioè vero o falso.

In generale per funzioni che restituiscono *true* o *false*, possiamo anche scrivere condizioni in modo semplicato.

```
if ( f_istaken("_scure_") == true )
```

è equivalente a:

```
if ( f_istaken("_scure_") )
```

```
if ( f_istaken("_scure_") == false )
```

è equivalente a:

```
if ( !f_istaken("_scure_") )
```

Oltre a *f_istaken* ci sono molte altre funzioni disponibili per le nostre condizioni: le vedremo meglio quando tratteremo il file *conditions.js*

Un'ultima nota sulla proprietà *menu* degli oggetti. Qualora avessimo bisogno di aggiungere tutti gli oggetti dell'inventario, possiamo farlo tramite la funzione *f_inventory( ).* Ad esempio, per replicare manualmente le sotto-voci fornite dalla proprietà *usable,* basta scrivere:

```
:menu:
-f_inventory();
-"te stesso";
++"usa con";
```

I caratteri utlilizzabili nel file *objects.txt* sono:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890 àÀáÁèÈéÉìÌíÍòÒóÓùÙúÚ
!£$%&/()=?^'|[]{}*+#-_,;.:<>§"
```

Alcuni caratteri hanno un uso speciale:

^	Questo carattere indica "a capo", ma può essere utilizzato solo nelle proprietà *desc_examine, desc_found, desc_talk, desc_next_talk*
''	Due caratteri ' consecutivi vengono trasformati in virgolette: ″
|	Usato nella proprietà *found* per specificare più oggetti oppure usato nelle proprietà *desc_examine, desc_found, desc_talk, desc_next_talk* per specificare a Pandor+ di scegliere a caso uno dei testi
§	Usato per separare in più messaggi i testi contenuti in *desc_examine, desc_found, desc_talk, desc_next_talk*

Apriamo *objects.txt* e modifichiamolo come segue, per aggiungere tutti gli oggetti de "L'Occhio Purpureo".

```
// -------------
// oggetti fissi
// -------------

<_fiume01_>
:examinable
:desc_examine: La corrente è molto forte... Stai attento!
:menu:
++"nuota";

<_foresta02_>
:examinable
:desc_examine: E' una foresta molto sinistra... Non resterei molto qui se fossi in te!

<_cancello04_>
:examinable
:direction: _rIngresso_
:desc_direction: entra
:desc_examine:
Da qui entri nella Città Segreta, dove si dice sia
nascosto l'Occhio Purpureo.

<_edificio05_>
:menu:
++"esamina";

<_mobiletto08_>
:examinable usable
:desc_examine: E' fatto di legno pregiato ma è pieno di tarli.
:desc_found: Sul suo fondo vedi una lampada.
:found: _lampadaSpenta00_

<_trono12_>
:examinable usable
:desc_examine: Il trono è in ferro battuto, circondato da drappi di seta.
:menu:
++"siedi";

<_catena12_>
:examinable
```

```
:desc_examine: Oscilla cigolando.
:menu:
-"una volta";
-"due volte";
-"tre volte";
 ++"tira";

// --------------
// oggetti mobili
// --------------

<_sacchettoPieno05_>
:examinable takeable usable
:desc_short: sacchetto
:desc_long: un sacchetto pieno
:desc_examine: Contiene delle biglie colorate.
:room: _rEsternoEdificio_
:weight: 20
:menu:
++"vuota";

<_sacchettoVuoto00_>
:examinable takeable usable
:desc_short: sacchetto
:desc_long: un sacchetto vuoto
:desc_examine: Non contiene nulla.
:room: _rOUT_
:weight: 5

<_biglie00_>
:examinable usable takeable
:desc_short: biglie
:desc_long: delle biglie colorate
:room: _rOUT_
:weight: 15
:desc_examine: Sono lucenti e coloratissime.
:desc_found: Tra di esse scorgi con stupore il prezioso Occhio Purpureo!
:found: _occhio00_

<_occhio00_>
:examinable takeable
:desc_short: occhio
```

```
:desc_long: il prezioso Occhio Purpureo
:room: _rOUT_
:weight: 5
:desc_examine:
Deve valere un mucchio di soldi! I tuoi problemi economici sono finiti!

<_mattone03_>
:examinable takeable usable
:desc_short: mattone
:desc_long: un mattone
:room: _rSentiero_
:weight: 40
:menu:
if (f_istaken("_mattone03_"))
{
  ++"lancia";
}

<_pistolaCarica00_>
:takeable usable
:desc_short: pistola
:desc_long: una pistola carica
:room: _rOUT_
:weight: 30
:menu:
if (f_istaken("_pistolaCarica00_"))
{
  ++"spara";
}

<_pistolaScarica00_>
:takeable usable
:desc_short: pistola
:desc_long: una pistola scarica
:room: _rOUT_
:weight: 20

<_lampadaSpenta00_>
:takeable
:desc_short: lampada
:desc_long: una lampada spenta
:room: _rOUT_
```

```
:weight: 20
:menu:
++"accendi";

<_lampadaAccesa00_>
:takeable
:desc_short: lampada
:desc_long: una lampada accesa
:room: _rOUT_
:weight: 20
:menu:
++"spegni";

<_lampadaRotta00_>
:takeable
:desc_short: lampada
:desc_long: una lampada rotta
:room: _rOUT_
:weight: 20

<_guanti11_>
:examinable takeable wearable
:desc_short: guanti
:desc_long: dei guanti da lavoro
:desc_examine: Sono dei guanti di gomma.
:room: _rGiardino_
:weight: 20
:weight_worn: 5

<_ispettore00_>
:examinable usable
:desc_short: ispettore
:desc_long: Il famigerato ispettore delle tasse!
:desc_examine: E' uno spilungone magro e dal sorriso diabolico!
:room: _rOUT_
:menu:
if (f_istaken("_pistolaCarica00_"))
{
  ++"spara";
}
if (f_istaken("_mattone03_"))
{
```

```
  ++"colpisci";
}


<_cadavere00_>
:examinable usable
:desc_short: cadavere
:desc_long: Il cadavere dell'ispettore delle tasse
:desc_examine: R.I.P. !
:room: _rOUT_


<_pappagallo07_>
:examinable talkable
:desc_short: pappagallo
:desc_long: Un pappagallo verde e giallo
:desc_examine: Sta appollaiato su un muretto. Ti osserva.
:room: _rIngresso_
:desc_talk: << Buona forrrrrtunaaaa!! >>
:desc_next_talk: Non sembra abbia altro da dirti.
```

Diversi oggetti hanno la proprietà *examinable*. Nel caso in cui non sia specificato un messaggio per "esamina" tramite proprietà *desc_examine* il gioco risponderà automaticamente con messaggi del tipo "Niente di interessante…". Lo stesso accade per *usable*: se in *actions.js* non sarà specificata una certa interazione tra gli oggetti, il gioco risponderà automaticamente con un messaggio tipo "Non sembra funzionare" (si veda <33> in *messages.txt*).

Da notare anche che sono state utilizzate linee di commento. I commenti vanno sempre in linee separate che iniziano con //.

Analizziamo gli aspetti più interessanti degli oggetti definiti.

```
<_fiume01_>
```

Avendo la proprietà *examinable*, verrà aggiunta in automatico l'azione "esamina" al menu. Viene definito anche il messaggio associato. Viene aggiunta in *menu* l'azione "nuota".

```
<_foresta02_>
```

Avendo la proprietà *examinable*, verrà aggiunta in automatico l'azione "esamina" al menu. Viene definito anche il messaggio associato.

```
<_cancello04_>
```

Questo oggetto avrà le azioni "esamina" e "entra". Da notare che nella proprietà *desc_examine* il messaggio è su più righe.

```
:desc_examine:

Da qui entri nella Città Segreta, dove si dice sia

nascosto l'Occhio Purpureo.
```

In generale nei messaggi Pandor+ trasforma sempre più spazi in uno solo e non considera i salti di linea.

Se volessimo forzare un "a capo" dopo la virgola, dobbiamo usare il carattere speciale ^:

```
:desc_examine:

Da qui entri nella Città Segreta,^dove si dice sia

nascosto l'Occhio Purpureo.
```

```
<_edificio05_>
```

Questo oggetto non ha nessuna proprietà "automatica" definita. Viene invece aggiunta a mano l'azione "esamina" in *menu*, e il risultato dell'azione (mostreremo dei messaggi) sarà diverso di volta in volta e lo gestiremo in *actions.js*

```
<_mobiletto08_>
```

Viene aggiunta l'azione "esamina" a seguito della quale comparirà la prima volta una lampada (oggetto *_lampadaSpenta00_*). Viene anche aggiunta "usa con" tramite *usable:* come sotto-voci appariranno automaticamente gli oggetti dell'inventario e "te stesso".

```
<_trono12_>
```

Per il trono vengono aggiunti "esamina" e "usa con". Viene aggiunta manualmente l'azione "siedi".

```
<_catena12_>
```

Anche per la catena viene aggiunto automaticamente"esamina" mediante la proprietà *examinable*. Verrà invece aggiunta manualmente un'azione "tira" con sotto menu "una volta" "due volte" "tre volte". Il risultato dell'azione, gestito in *action.js*, dipenderà dall'avere o no l'occhio purpureo, i guanti e se tiriamo la catena il numero di volte giusto.

Per oggetti che possono avere più "stati" (lampada accesa – lampada spenta, ad esempio) è necessario definire un oggetto "gemello" per ogni stato diverso dell'oggetto che ci occorre. Di solito, variano al massimo la descrizione, il messaggio se esaminato e il peso. Ovviamente un solo oggetto "gemello" alla volta può avere la proprietà *room* diversa da *_rOUT_*. Ad esempio inizialmente l'oggetto lampada spenta sarà nella stanza *_rStanzaMobiletto_* e l'oggetto lampada accesa sarà fuori dal gioco, ovvero la sua *room* sarà *_rOUT_*

Quando accenderemo la lampada spenta, la sua *room* diventerà *_rOUT_* mentre la *room* della lampada accesa diventerà quella della lampada spenta.

Il trucco sta insomma nel far sparire dal gioco un oggetto a favore del suo gemello. Per il giocatore tutto questo sarà ovviamente trasparente, e penserà di aver a che fare con lo stesso oggetto che cambia stato.

```
<_sacchettoPieno05_>

<_sacchettoVuoto00_>
```

Il sacchetto può o no contenere delle biglie. Inizialmente viene piazzato "sacchetto pieno" nella stanza *_rEsternoEdificio_*, mentre il “sacchetto vuoto” è fuori dal gioco, *room* è infatti *_rOUT_*. Per il "sacchetto pieno", aggiungiamo manualmente l’azione "vuota" (sarà proprio questa azione a far sparire il sacchetto pieno e a far comparire quello vuoto).

```
<_biglie00_>
```

Quando vuotiamo il sacchetto, compariranno delle biglie. Se le esaminiamo, comparirà l’Occhio Purpureo. A tal scopo, sono utilizzate le proprietà *found* e *desc_found*.

```
<_occhio00_>
```

Salta fuori automaticamente esaminando le biglie. Inizialmente la sua *room* è quindi *_rOUT_*

```
<_mattone03_>
```

Causerà la morte del giocatore se tenteremo di nuotare con esso nel nostro inventario. Viene aggiunta l’azione "lancia" (se lo abbiamo con noi) che può far credere di poterlo utilizzare contro l’Ispettore delle tasse...

```
<_pistolaCarica00_>

<_pistolaScarica00_>
```

La pistola potrà essere carica (1 colpo) o scarica. Potrà essere usata per uccidere l’ispettore delle tasse. Viene aggiunta l’azione "spara" in *menu*, ma solo se la pistola è nell'inventario.

```
<_lampadaSpenta00_>

<_lampadaAccesa00_>

<_lampadaRotta00_>
```

La lampada sarà indispensabile per uscire dalla stanza buia, o il giocatore non ne uscirà più. Vengono aggiunte le azioni "accendi" e "spegni". Inoltre esisterà la possibilità di romperla se usata col mattone.

```
<_guanti11_>
```

I guanti sono l’unico oggetto indossabile (proprietà *wearable*) dell’avventura. Il peso se indossati è 5, contro 20 se trasportati. Serviranno a tirare la catena senza restare fulminati!

```
<_ispettore00_>
```

Ecco il nostro nemico nell’avventura. Comparirà un’unica volta (come oggetto mobile, ovviamente non prendibile) e ci ruberà un oggetto a meno di non ucciderlo con la pistola. La sua comparsa sarà casuale e avverrà nella stessa stanza del giocatore (a meno che non sia la foresta o la stanza buia) dopo 20 azioni dall’inizio

dell’avventura. Vengono aggiunti le azioni "spara" se il giocatore possiede la pistola e "colpisci" se possiede il mattone.

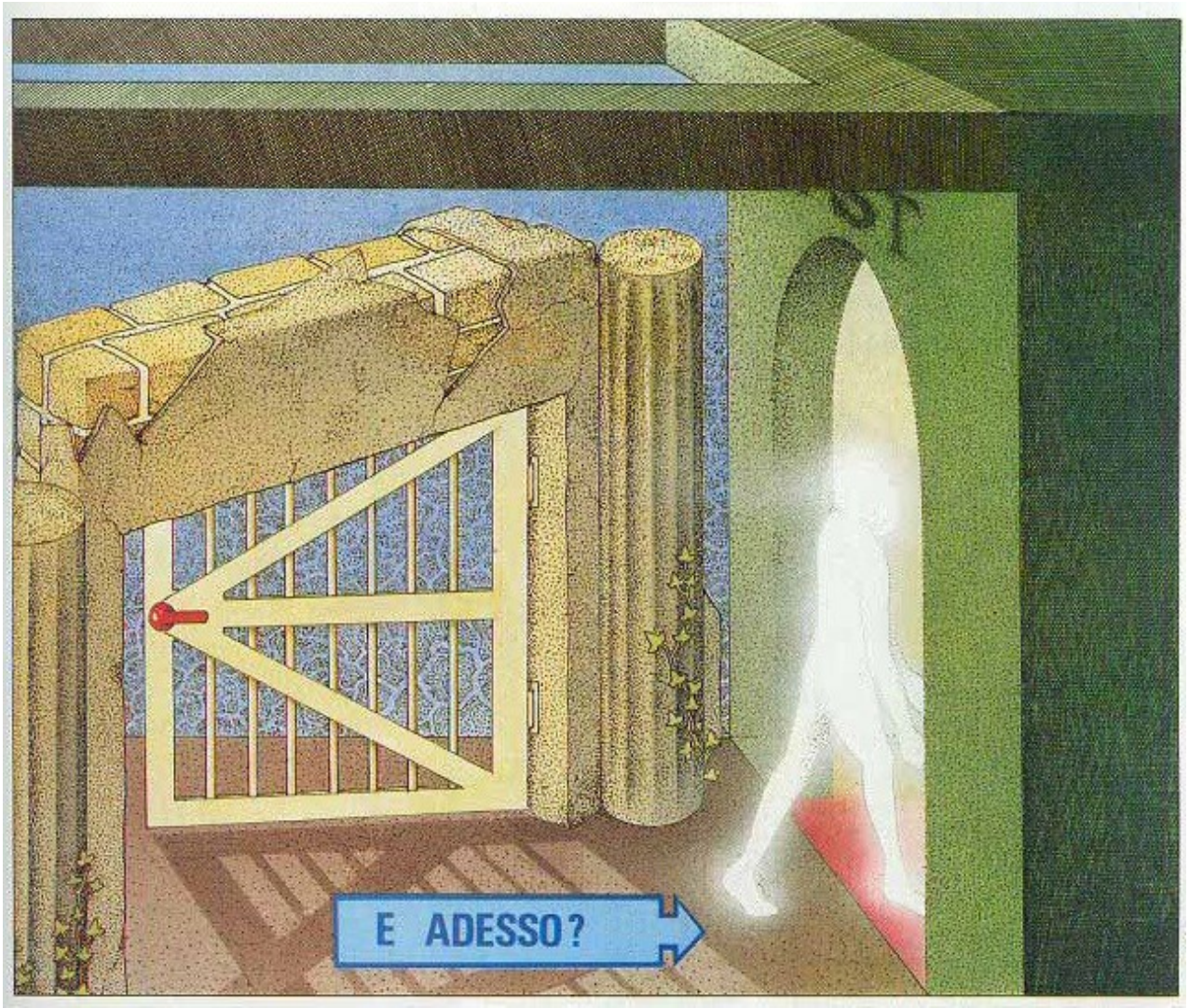
```
<_cadavere00_>
```

Se uccidiamo l’ispettore, al suo posto comparirà il suo cadavere. Inizialmente questo oggetto è fuori dal gioco (*room* è posto a *_rOUT_)*

```
<_pappagallo07_>
```

Il pappagallo ha la proprierà *talkable*: significa che verrà aggiunta in automatico l'azione "parla con".

Il messaggio che compare la prima volta che l'azione viene fatta è specificato in *desc_talk*. Dalla seconda volta in poi, comparirà invece il messaggio specificato da *desc_next_talk* .



I file di progetto *conditions.js* e *counters.js*

Nel corso dell'avventura le posizioni degli oggetti cambiano, il giocatore passa attraverso diverse stanze e compie azioni che mettono in moto degli eventi.

Il codice scritto nei file *counters.js* e *conditions.js* ci permette di definire dei mutamenti del gioco o dei messaggi da mostrare in base a determinate condizioni. Alcuni esempi potrebbero essere:

- se il giocatore passa per una stanza, far apparire un oggetto da qualche parte
- se il giocatore rimane troppo tempo in una stanza, fargli succedere una disgrazia o addirittura far terminare il gioco ("L'aria velenosa ti uccide... Dovevi uscire prima!")
- aggiungere dopo la descrizione della stanza un testo che avverte il giocatore di qualcosa ("Senti uno sparo!" , "Sei allo stremo delle forze, devi mangiare qualcosa!" …)
- modificare la descrizione di una stanza in determinate circostanze (ad esempio presenza di luce o no)

Più in generale, ogni qualvolta il giocatore compie un'azione, il gioco "risolve" l'azione e – prima di mostrare il risultato della stessa - valuta anche il codice presente in *counters.js* e *conditions.js*.

E' importante ora fare una distinzione sul tipo di azione nel gioco. Per azione intendiamo il comando che il giocatore compie cliccando su un oggetto qualsiasi dell'avventura. Alcune azioni corrispondono ad una **mossa** del giocatore (anche se non porta a nessun risultato) come ad esempio "esamina"o "prendi". Nel caso invece di un comando del tipo "aiuto", "carica" o "salva" diremo che si tratta di un azione "vuota". Anche all'inizio del gioco, la prima azione è vuota e lo stesso accade tutte le volte che viene fatto un refresh della pagina web del gioco. Le azioni vuote insomma non sono mosse e non devono modificare mai lo "stato" del gioco e del giocatore.

In generale:

- in *counters.js* potremo modificare le variabili del gioco tutte le volte che viene compiuta una mossa (azione non vuota) dal giocatore, per tener conto del passaggio "di tempo" o del numero di mosse che il giocatore fa in una determinata circostanza
- in *conditions.js* potremo cambiare posizione degli oggetti, posizione del giocatore e mostrare un messaggio "popup" o del testo aggiuntivo dopo la descrizione della stanza in base a condizioni che dipendono dallo stato delle variabili o degli oggetti o del giocatore

Anche se alcuni dettagli saranno più chiari dopo aver visto anche *actions.js*, la sequenza (semplificata) del ciclo di operazioni eseguite dal motore del gioco ad ogni azione è questa:

1. Se è un azione vuota (ad esempio un refresh della pagina) vai al punto 5, altrimenti prosegui al punto 2
2. viene valutato *actions.js*: se qui l'azione è gestita allora esegui il codice e poi vai al punto 4, altrimenti continua al punto 3. Nel codice può essere chiamata la funzione *f_show* (per mostrare un messaggio "popup")
3. viene gestita l'azione in modo automatico dal motore del gioco, in base alla proprietà degli oggetti coinvolti nell'azione e chiamata *f_show* con l'eventuale messaggio di risposta ("Niente di interessante...", "Hai troppe cose!", etc)
4. viene eseguito *counters.js*
5. viene eseguito *conditions.js*. In questa parte possono essere chiamate *f_show* per far visualizzare un messaggio popup oppure *f_append* per aggiungere del testo dopo la descrizione della stanza
6. se è stata chiamata *f_show*, viene mostrato un messaggio "popup" dal quale si esce toccando "Continua"
7. se è stato chiamato *f_lose*, il gioco termina (schermata di fallimento)

8. se è stato chiamato *f_win*, il gioco termina (schermata di successo)
9. viene mostrata la descrizione in base alla stanza corrente ed agli oggetti presenti. Viene eventualmente aggiunto un messaggio addizionale in grassetto maiuscoletto se è stata chiamata *f_append*

In pratica:

- **in caso di azioni <u>vuote</u> viene eseguito solamente *conditions.js***
- **In caso di mosse (azioni <u>non vuote</u>) vengono invece eseguiti *actions.js*, *counters.js, conditions.js***

L'importanza di questa distinzione sarà chiara tra poco.

Vediamo nel dettaglio il file *counters.js*. Dentro ad esso vanno modificate le variabili per tener conto del "tempo che passa", ovvero del fatto che il giocatore si è realmente "mosso" in termini temporali o spaziali nel gioco.

```
// ogni azione incrementa la variabile vMOVES
_vMOVES_ = _vMOVES_ + 1;
```

Questo pezzo di codice fa si che *_vMOVES_* sia incrementato ad ogni azione. In questo modo *_vMOVES_* può essere usata per tener conto del "passaggio di tempo", ovvero di quante mosse abbia fatto il giocatore dall'inizio del gioco. Questo è molto utile per determinare eventi nel gioco che dipendono dal "tempo". Ad esempio aggiungere un testo con scritto "Il pendolo fa DONG!" ogni tot mosse.

Vediamo altri esempi. Se volessimo incrementare la variabile *_vMosseInCantina_* solo se il giocatore si trova nella stanza *_rCantina_* , il codice di *counters.js* diventerebbe:

```
_vMOVES_ = _vMOVES_ + 1;

if ( f_isplayerin("_rCantina_") == true )
{
 _vMosseInCantina_ = _vMosseInCantina_ + 1;
}
```

Quindi, *_vMosseInCantina_* si incrementa solo se il giocatore si trova nella cantina.

In generale le condizioni che scriviamo possono utilizzare le variabili definite in *variables.txt* e una serie di funzioni messe a disposizione da Pandor+, qui di seguito elencate:

```
f_getcurrentweight()                    restituisce attuale ingombro degli oggetti trasportati

f_istaken( "_oggetto_" )                restituisce true se oggetto "_oggetto_" è portato o indossato
                                        dal giocatore
```

```
f_ishere( "_oggetto_" )                    restituisce true se oggetto "_oggetto_" è portato o indossato
                                           dal giocatore oppure si trova nella stanza del giocatore

f_isworn( "_oggetto_" )                    restituisce true se oggetto "_oggetto_" è indossato dal
                                           giocatore

f_isout( "_oggetto_" )                     restituisce true se oggetto "_oggetto_" è fuori dal gioco


f_isplayerin( "_stanza_" )                 restituisce true se giocatore è nella stanza "_stanza_"

f_isin( "_oggetto_" , "_stanza_" )         restituisce true se oggetto "_oggetto_" è in stanza room

f_whereisplayer()                          restituisce la stanza dove si trova il giocatore,
                                           ad esempio "_rForesta_"

f_whereis( "_oggetto_" )                   restituisce la stanza dove si trova l'oggetto "_oggetto_"
                                           (ad esempio: "_rForesta_") oppure
                                                  "_rOUT_"    se l'oggetto è "fuori dal gioco"
                                                  "_rTAKEN_"  se l'oggetto è preso dal giocatore
                                                  "_rWORN_"   se l'oggetto è indossato dal giocatore

f_descshort( "_oggetto_" )                  restituisce la descrizione corta dell'oggetto

f_desclong( "_oggetto_" )                   restituisce la descrizione lunga dell'oggetto

f_tag( "_oggetto_" )                        restituisce il valore tag dell'oggetto
```

In *counters.js* vanno quindi modificate le variabili per tener conto che il giocatore ha effettivamente eseguito una azione non vuota in una certa circostanza.

Veniamo ora invece lo scopo di *conditions.js*.

Fondamentalmente in questo file vanno invece inserite le condizioni che portano a modifiche allo stato dell'avventura. Tuttavia, a differenza di *counter.js*, questo file viene eseguito dal motore del gioco anche in caso di azioni vuote (come un refresh della pagina web).

Ne consegue che quando scriviamo il codice in questo file, dobbiamo porci la seguente domanda: se il giocatore effettuasse 1000 volte il refresh della pagina, lo stato del gioco rimane coerente?

Facciamo un esempio. Vogliamo che se il giocatore si trova in cantina, venga incrementata la variabile *_vQuanteVolteGiocatoreInCantina_* ad ogni mossa del giocatore. Quando questa variabile diventa uguale a 5 (ovvero se il giocatore ha fatto 5 mosse in cantina) il giocatore muore.

Immaginiamo di scrivere il seguente codice in *conditions.js:*

```
if ( f_isplayerin("_rCantina_") )
{
  _vQuanteVolteGiocatoreInCantina_ = _vQuanteVolteGiocatoreInCantina_ + 1;
  if ( _vQuanteVolteGiocatoreInCantina_ == 5 )
  {
    f_show("OH NO! Un enorme ragno scende dal soffitto e ti inietta il suo veleno mortale...");
    f_lose();
  }
}
```

Questo codice non va bene!

Infatti se immaginiamo di fare molte volte il refresh della pagina (quindi un'azione vuota), *_vQuanteVolteGiocatoreInCantina_* viene incrementato comunque! Quindi, al quinto refresh, il giocatore morirebbe senza aver compiuto delle vere mosse.

L'errore è che se il giocatore è in cantina e si fa refresh della pagina (o in generale un'azione vuota), *_vQuanteVolteGiocatoreInCantina_* viene incrementata comunque, perchè *conditions.js* viene eseguito anche in caso di azioni vuote. La soluzione è separare il tutto in due condizioni, una per *counters.js,* eseguito solo in caso di mosse vere*:*

```
// da mettere in counters.js
if ( f_isplayerin("_rCantina_") )
{
  _vQuanteVolteGiocatoreInCantina_ = _vQuanteVolteGiocatoreInCantina_ + 1;
}
```

e l'altra per *conditions.js:*

```
// da mettere in conditions.js
if ( _vQuanteVolteGiocatoreInCantina_ == 5 )
{
  f_show("OH NO! Un enorme ragno scende dal soffitto e ti inietta il suo veleno mortale...");
  f_lose();
}
```

In questo modo, anche se facciamo molte volte il refresh della pagina, *_ vQuanteVolteGiocatoreInCantina_* non viene incrementato.

Se ora volessimo che, se siamo in cantina, venga anche aggiunto (dopo la descrizione della stanza) un testo "minaccioso", basterebbe aggiungere a *conditions.js* il seguente codice:

```
if ( f_isplayerin("_rCantina_") )
{
  f_append("...Delle ombre si muovono sul soffitto...");
}
```

E' corretto che questa ultima condizione stia in *conditions.js?* Si, perchè anche in caso di refresh della pagina web è corretto che ogni volta appaia la frase minacciosa.

Facciamo qualche altro esempio.

Supponiamo che esista una stanza la cui descrizione (in *rooms.txt*) sia la seguente:

```
<05>
_rStanzaDellaFontana_|_vTestoStanzaDellaFontana_
Sei dentro una grande stanza con una zampillante {fontana|_fontana05_}.
Puoi andare a {Sud|_sud05_}.
§
Sei dentro una grande stanza con una {fontana|_fontana05_} che non sgorga nulla.
Puoi andare a {Sud|_sud05_}.
```

*_vTestoStanzaDellaFontana_* è una variabile inizialmente uguale a 1. Come abbiamo visto, vuol dire che la descrizione della stanza mostrata nel gioco sarà la prima:

```
Sei dentro una grande stanza con una zampillante {fontana|_fontana05_}.
Puoi andare a {Sud|_sud05_}.
```

Vogliamo però che dopo 50 azioni dall'inizio dell'avventura, la descrizione della stanza diventi la seconda:

```
Sei dentro una grande stanza con una {fontana|_fontana05_} che non sgorga nulla.
Puoi andare a {Sud|_sud05_}.
```

Ecco come fare scrivendo una condizione in *conditions.js*:

```
if ( _vMOVES_ >= 50 )
{
  _vTestoStanzaDellaFontana_ = 2;
}
```

Altro esempio. Vogliamo ora che se il giocatore passa almeno una volta per la stanza della fontana, la variabile *_vGiocatorePassatoPerStanzaFontana_,* inizialmente uguale a 0, venga messa a 1:

```
if ( _vMOVES_ >= 50 )
{
  _vTestoStanzaDellaFontana_ = 2;
}

if ( f_isplayerin("_rStanzaDellaFontana_") )
{
  _vGiocatorePassatoPerStanzaFontana_ = 1;
}
```

In questo modo avremo "registrato" il passaggio del giocatore nella stanza della fontana attraverso l'uso di una variabile *_vGiocatorePassatoPerStanzaFontana_* .

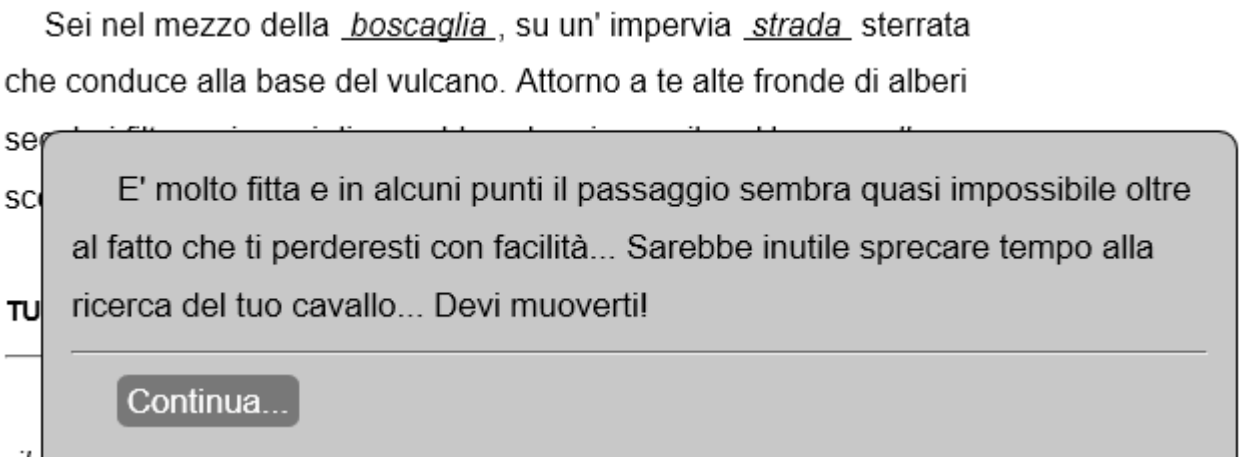
Da questo momento, in qualsiasi altra condizione, per sapere se siamo passati per la stanza della fontana basterà chiedersi se *_vGiocatorePassatoPerStanzaFontana_* è uguale a 1.

Aggiungiamo ad esempio ora che se il giocatore si trova nella stanza *_rSalone_* ed è passato per la stanza della fontana, verrà mostrato un messaggio ma una volta sola. Per farlo abbiamo bisogno di un'altra variabile, *_vMostratoMessaggioOmbra_*, inizialmente uguale a 0.

```
if ( _vMOVES_ >= 50 )
{
  _vTestoStanzaDellaFontana_ = 2;
}

if ( f_isplayerin("_rStanzaDellaFontana_") )
{
  _vGiocatorePassatoPerStanzaFontana_ = 1;
}

if ( (f_isplayerin("_rSalone_"))
      && (_vGiocatorePassatoPerStanzaFontana_ == 1)
      && (_vMostratoMessaggioOmbra_ == 0)
   )
{
  _vMostratoMessaggioOmbra_ = 1;
  f_show ("Per un attimo vedi
           qualcosa muoversi nell'ombra...");
}
```

Il messaggio verrà mostrato un'unica volta in quanto *_vMostratoMessaggioOmbra_* diviene 1 e la condizione non può più verificarsi (perchè per poter essere eseguita, *_vMostratoMessaggioOmbra_* deve essere 0).

f_show è una funzione che permette di mostrare messaggi popup che "bloccano" il gioco finchè non si tocca *Continua*. Ecco un esempio di messaggio popup:



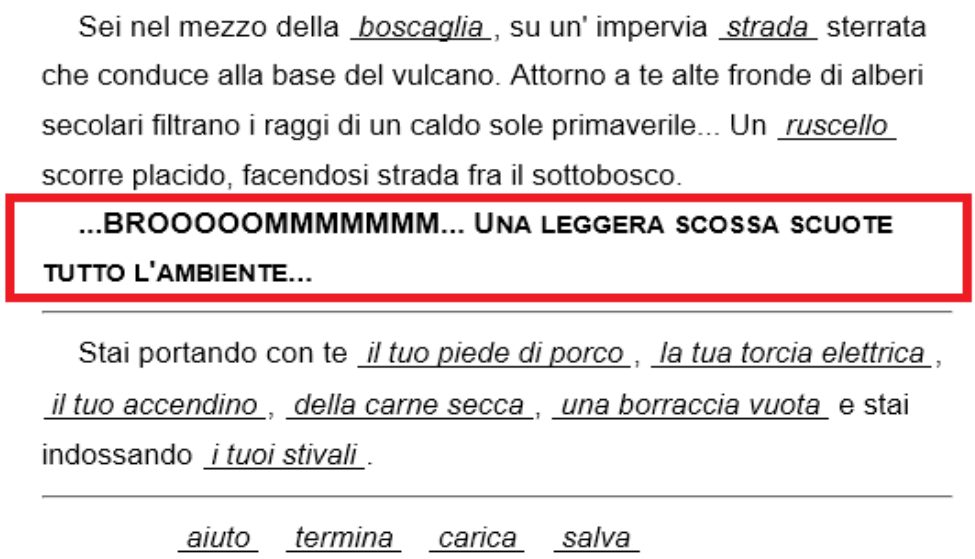
Se viene chiamata più volte *f_show,* Pandor+ accorpa i vari testi in un unico messaggio.

Ecco l'elenco di funzioni che possiamo utilizzare per agire sul gioco:

```
f_show("...")          mostra un messaggio popup con comando "continua"
                       Il testo passato può contenere i seguenti caratteri speciali:
                       '' = inserisce carattere "
                       ^  = causa un "a capo"
                       |  = divide il testo in sotto-testi. Verrà scelto a caso uno di
                            questi sotto-testi
                       §  = divide il messaggio in più schermate (ha priorità rispetto |)

f_append("...")        aggiunge testo dopo la descrizione della stanza
                       Il testo può contenere i seguenti caratteri speciali:
                       '' = inserisce carattere "
                       ^  = causa un "a capo"
                       |  = divide il testo in sotto-testi. Verrà scelto a caso uno
                            dei sotto-testi

                       nota: f_append si può usare SOLO in conditions.js

f_textisempty()        funzione che restituisce true se non è stato ancora
                       aggiunto nessun testo tramite f_append.

                       nota: da usare SOLO in conditions.js
```

```
f_rndtxt("...")      estrae e restituisce (a caso) uno dei testi separati da |.

f_getcurrentweight() restituisce attuale ingombro degli oggetti trasportati

f_getrandomint(a,b)  restituisce un numero intero casuale tra a e b compresi

f_win()              conclude il gioco con successo

f_lose()             conclude il gioco con fallimento

f_move("_oggetto_","_stanza_")  sposta oggetto _oggetto_ nella stanza _stanza_
                                Valori speciali di _stanza_:

                                     "_rOUT_"    per far uscire l'oggetto dal gioco

                                     "_rTAKEN_"  per dare oggetto al giocatore

                                     "_rWORN_"   per far indossare al giocatore l'oggetto

                                In alternativa:

                                     f_moveout("_oggetto_")       sposta oggetto _oggetto_ fuori dal gioco

                                     f_givetoplayer("_oggetto_")  sposta oggetto _oggetto_ nell'inventario

                                     f_putonplayer("_oggetto_")   fa indossare _oggetto_ al giocatore

f_moveplayer("_stanza_")  : sposta giocatore in stanza _stanza_
```

Una delle funzioni più importanti è *f_append*, che permette di aggiungere testo dopo quello della descrizione della stanza e degli eventuali oggetti presenti in essa. Eccone un esempio:



Da notare che il messaggio sarà sempre in maiuscoletto grassetto, e se *f_append* viene chiamata più volte, Pandor+ accoderà i vari testi.

Ecco un esempio di uso di *f_append* in *conditions.js*. Vogliamo far aggiungere al testo della stanza *_rGrotta_* una frase scelta a caso tra 3. La frase appare a intervalli casuali di 5-15 azioni:

```
// se uguale a 0, la variabile vProssimaScossa viene
// posta uguale al numero corrente di mosse fatte dal giocatore
// più un numero casuale da 5 a 15
if ( vProssimaScossa == 0 )
{
  _vProssimaScossa_ = _vMOVES_ + f_getrandomint(5,15);
}

// quando vMOVES (incrementata ad ogni mossa del giocatore) raggiunge il valore
// contenuto in vProssimaMossa, aggiungi un messaggio ma solo se il giocatore è nella stanza
// della grotta
if ((_vMOVES_ == _vProssimaScossa_) && (f_isplayerin("_rGrotta_")))
{
  // rimettiamo a 0 la variabile vProssimaScossa
  _vProssimaScossa_ = 0;

  // aggiungi una delle tre frasi, scegliendola a caso, dopo la descrizione della stanza
  f_append("
        ...BROOOOOMMMMMMM... Una leggera scossa scuote tutto l'ambiente...|
        ...BROOOOOMMMMMMM... Improvvisamente tutto trema per qualche secondo...|
        ...BROOOOOMMMMMMM... Per qualche lunghissimo istante tutta la grotta trema!!!
  ");
}
```

Infine ecco un'altra funzione che ci può essere utile nelle nostre condizioni: *f_getcurrentweight()* che ritorna il peso di tutto quello che trasportiamo. Ad esempio, con troppo peso, il giocatore potrebbe affondare nelle sabbie mobili appena mette piede nella relativa stanza:

```
// se il preso trasportato è maggiore di 50 e il giocatore è nella stanza
// delle sabbie mobili
if ((f_getcurrentweight() > 50) && (f_isplayerin("_rSabbieMobili_"))
{
        f_show("Fai pochi passi e sprofondi nelle orribili sabbie mobili...");
        f_lose();
}
```

Chiamare la funzione *f_lose* significa andare alla schermata di "fallimento" del gioco (vedi *fail.txt*). Al contrario, chiamare *f_win* significa andare a quella di successo (*success.txt*) e terminare l'avventura.

Prima di vedere cosa scrivere in *counters.js* e *conditions.js* per l'Occhio Purpureo, una nota sui caratteri ammessi in questi file. Questi sono:

```
qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890  àÀáÁèÈéÉìÌíÍòÒóÓùÙúÚ
!"£$%&/()=?^'|[]{}*+#-_,;.:<>§
```

Dedichiamoci ora alle condizioni dell'avventura "L'Occhio Purpureo". Apriamo *counters.js* e modifichiamolo come segue:

```
// ogni azione incrementa la variabile MOVES
_vMOVES_ = _vMOVES_ + 1;

// se sono nella foresta, incrementa vAzioniInForesta
if ( f_isplayerin("_rForesta_") == true )
{
  _vAzioniInForesta_ = _vAzioniInForesta_ + 1;
}
// altrimenti mettilo a 0, in modo da ricominciare il conto delle azioni
// nella foresta da zero se ci si ritorna
else
{
  _vAzioniInForesta_ = 0;
}
```

```
// se è comparso l'ispettore, incrementa vAzioniConIspettore
if ( _vStatoIspettore_ == 1 )
{
  _vAzioniConIspettore_ = _vAzioniConIspettore_ + 1;
}
```

La prima condizione incrementa *_vAzioniInForesta_* solo se il giocatore si trova nella foresta, e in caso contrario la azzera. In questo modo ogni volta che il giocatore esce dalla foresta il conteggio delle mosse fatte nella stessa riparte da 0.

La seconda condizione invece incrementa *_vAzioniConIspettore_* se *_vStatoIspettore_* è uguale a 1. Vedremo quando *_vStatoIspettore_* viene messa a 1 e perchè.

Apriamo ora *conditions.js* e modifichiamolo come segue.

```
// se abbiamo (o è presente) la lampada accesa nella stanza buia,
// cambia la descrizione della stanza
if ( (f_isplayerin("_rStanzaBuia_")) && (f_ishere("_lampadaAccesa00_")) )
{
  _vTestoStanzaBuia_ = 2;
}
else
{
  _vTestoStanzaBuia_ = 1;
}

// se sono nella foresta per una o due azioni, scrivi una frase aggiuntiva
if ( f_isplayerin("_rForesta_") )
{
  if ( _vAzioniInForesta_ == 1 )
  {
    f_append("Questa foresta ha qualcosa di inquietante... Ti senti strano...");
  }
  else if ( _vAzioniInForesta_ == 2 )
  {
    f_append("Senti uno strano formicolio lungo il corpo...
              Il tuo istinto ti dice di andartene...");
  }
}
```

```
// se non è mai stata valorizzata, mettiamo in vTirareCatena il numero di volte
// che bisogna tirare la catena del trono per vincere l'avventura
if ( _vTirareCatena_ == 0 )
{
  _vTirareCatena_ = f_getrandomint(1,3);
}

// se sono nella foresta, alla terza azione fai morire il giocatore
if ( ( f_isplayerin("_rForesta_") ) && ( _vAzioniInForesta_ == 3 ) )
{
  f_show("
    Improvvisamente ti rendi conto di non poterti più muovere...^
    Ti stai pietrificando come gli alberi!!! Che fine terribile...
    ");

  f_lose();

  // inutile proseguire in conditions.js
  return;
}



// l'ispettore compare dopo 20 azioni dall'inizio dell'avventura
// se non sono nella foresta o nella stanza buia e se non è mai apparso.
if (
      // ispettore non è ancora apparso
      ( _vStatoIspettore_ == 0 )
      // giocatore ha fatto almeno 20 azioni
      && ( _vMOVES_ >= 20 )
      // non sono nella foresta e nella stanza buia
      && (!f_isplayerin("_rForesta_")) && (!f_isplayerin("_rStanzaBuia_"))
   )
{
  // ispettore compare solo una volta
  _vStatoIspettore_ = 1;

  // ispettore compare nella stessa stanza del giocatore
  f_move( "_ispettore00_" , f_whereisplayer() );

  f_show("E' comparso l'ispettore delle tasse!");
}
```

```
// se l'ispettore è comparso e abbiamo fatto una azione non vuota in sua presenza
if ( ( _vStatoIspettore_ == 1 ) && ( _vAzioniConIspettore_ == 1 ) )
{
  // ispettore esce di scena
  _vStatoIspettore_ = 2;

  // se il giocatore ha almeno un oggetto, l'ispettore ne prende uno
  // a caso e se ne va
  var lConta = 0;

  // oggetto scelto dall'ispettore
  var lOggettoScelto;

  // punteggio random per decidere quale oggetto verrà preso dall'ispettore
  var lPunteggio = 0;

  // ciclo su tutti gli oggetti
  foreachobject{
    // se oggetto è trasportato dal giocatore
    if (f_istaken(id))
    {
      lConta = lConta + 1;

      // per ogni oggetto genero un numero a caso da 1 a 10
      var lTemp = f_getrandomint(1,10);

      // se è maggiore del massimo ottenuto finora, l'oggetto scelto diventa questo
      if (lTemp > lPunteggio)
      {
          lPunteggio = lTemp;
          lOggettoScelto = id;
      }
    }
  }

  // se il giocatore ha almeno un oggetto
  if (lConta > 0)
  {
    // l'ispettore scompare
    f_moveout("_ispettore00_");
```

```
    // l'oggetto scompare
    f_moveout(lOggettoScelto);

    f_show(
     "L'ispettore delle tasse esige il riscatto per i tuoi debiti
      e preleva uno dei tuoi oggetti come anticipo! Per tua fortuna
      poi se ne va via...");
  }
  // altrimenti finisce in gattabuia!
  else
  {
    f_show(
     "L'ispettore delle tasse esige il riscatto per i tuoi debiti ma tu non hai nulla
      da dargli... Purtroppo per te la tua pena è la gattabuia...");

    f_lose();

    return;
  }

}
```

Inizialmente vi sono delle condizioni per modificare la descrizione della stanza buia se possediamo la lampada accesa e per aggiungere alcune frasi di avvertimento tramite *f_append* se siamo nella foresta.

Viene ucciso il giocatore se rimane per più di due azioni nella foresta.

Viene deciso il numero di volte che la catena va tirata. Poiché *_vTirareCatena_* all'inizio è 0, la prima volta la condizione è verificata e *_vTirareCatena_* viene valorizzata con un valore casuale da 1 a 3. Da questo momento in poi quindi la condizione non potrà più essere verificata perché *_vTirareCatena_* non sarà mai più uguale a 0.

Viene fatto comparire l'ispettore dopo 20 azioni dall'inizio dell'avventura. L'ispettore compare nella stanza del giocatore. Quando appare, *_vStatoIspettore_* viene messo a 1.

Se l'ispettore è comparso e abbiamo già fatto un'azione con lui presente (che non ha portato alla sua morte), egli deruberà il giocatore di un oggetto tra quelli trasportati (se ne ha) oppure si finirà in gattabuia e il gioco terminerà.

In questa parte del codice viene fatto uso di variabili *locali* javascript, ad esempio:

```
  var lConta = 0;
```

Queste variabili perdono il loro valore una volta usciti dalla condizione in cui sono definite. Possono contenere interi o testi, a seconda delle necessità. Vengono definite con la sintassi: *var nomevariabile = valore iniziale;*

Per il nome di queste variabili, non usate mai il carattere _ !

Tramite un ciclo *foreachobject* vengono poi passati in rassegna tutti gli oggetti dell'avventura e scelto tramite un punteggio casuale quale di quelli trasportati prelevare.

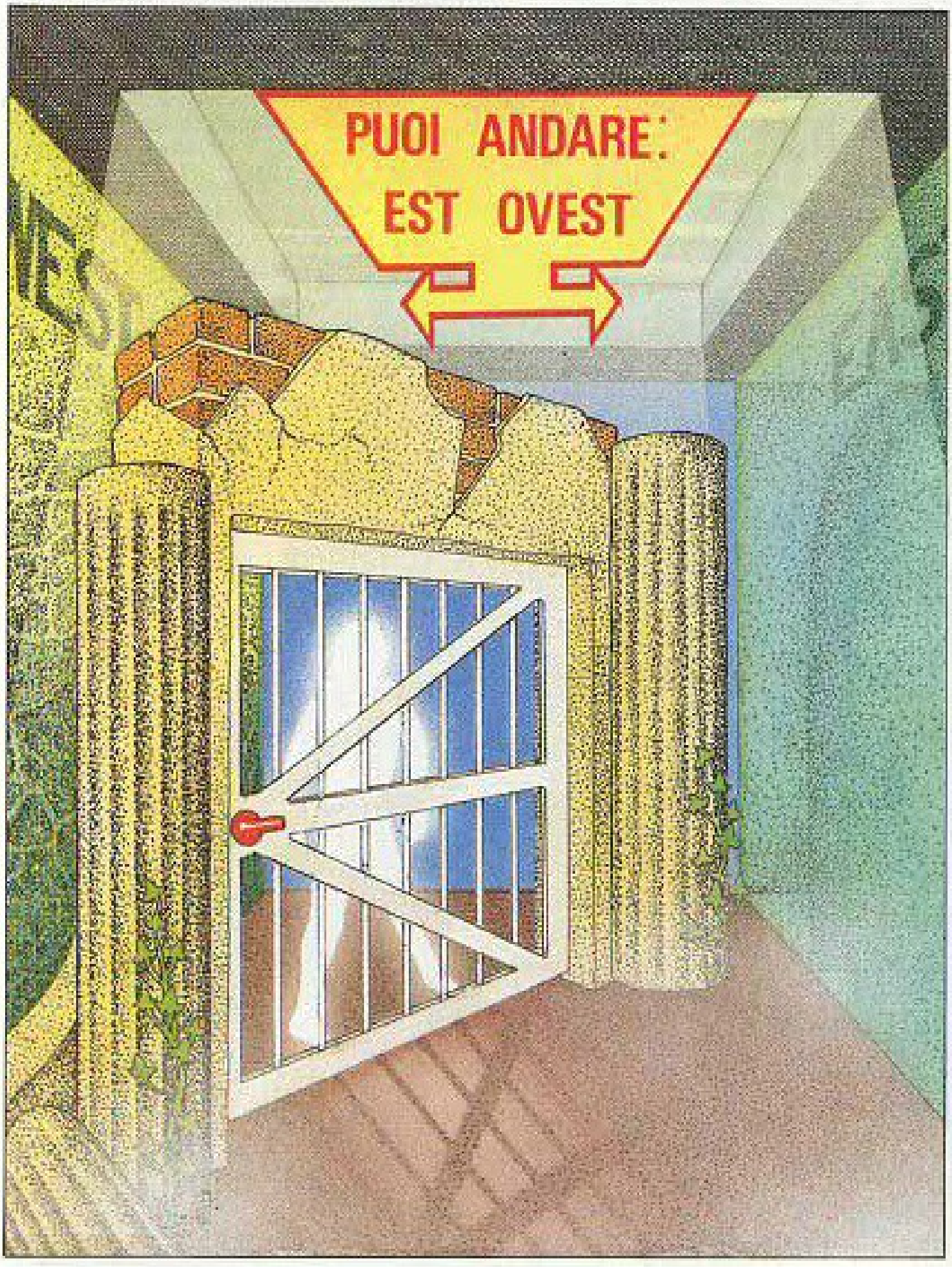
foreachobject{ ... } è un istruzione speciale che "cicla" il codice scritto tra { } per ogni oggetto dell'avventura, il cui identificatore è disponibile nella variabile locale id, definita implicitamente. Ad esempio:

```
var ls = "";


foreachobject{
  // il codice dentro il corpo di foreachobject
  // verrà eseguito per ogni oggetto, il cui identificatore
  // sarà disponibile nella variabile locale id


  if ( f_tag(id) == 1 )
  {
    ls = ls + f_desclong(id) + "^";
  }


}
```

Nell'esempio, verrà eseguito il codice in rosso per ogni oggetto dell'avventura.

In questo modo verrà aggiunta, nella variabile locale *ls*, le descrizioni lunghe di tutti gli oggetti con la proprietà "tag" uguale a 1.

Si noti infine che tutte le volte che il giocatore "muore", viene bloccata l'esecuzione di *conditions.js* tramite l'istruzione *return*, in quanto non ci interessa considerare altre condizioni.


PUOI ANDARE:
EST OVEST

Il file di progetto *actions.js*

Siamo arrivati all'ultimo file di progetto che modificheremo. Nel file *actions.js* vanno intercettate tutte le azioni fatte dal giocatore e che non sono gestite automaticamente dal motore del gioco (come ad esempio "prendi"/"lascia" per oggetti con proprietà *takeable, "*esamina" per la proprietà *examinable*, etc).

Quando il giocatore esegue un'azione non vuota, *action.js* viene passato in rassegna per vedere se abbiamo deciso di gestirla.

Se l'azione non è gestita da noi e non è tra quelle "automatiche" basate sulle proprietà degli oggetti, ci penserà il motore del gioco a rispondere con un messaggio "Non sembra funzionare!" o simili.

Vediamo quindi come catturare un'azione. Dovremo creare delle condizioni (*if* ...) utilizzando:

- la funzione *f_action*
- le funzioni già viste in *conditions.js* (*f_istaken, f_whereis, etc.*)
- le variabili definite in *variables.txt* (ad esempio: *_vApertaCassa_*)

f_action è una funzione che ritorna *true* se è stata compiuta una certa azione. Ad esempio:

```
if (f_action("_ruscello14_", "usa con", "_borraccia00_" ))
{
  ...
}
if (f_action("_ruscello14_", "usa con", "te stesso" ))
{
  ...
}
```

Il codice tra *{ }* verrà eseguito solo se il giocatore ha toccato l'oggetto *_ruscello14_* , selezionato l'azione "usa con" e poi *_borraccia00_* , ovvero *ruscello – usa con – borraccia*

f_action può essere usata anche per azioni senza *sotto-voci*, come *ruscello – bevi*

```
if (f_action("_ruscello14_", "bevi"))
{
  ...
}
```

Dentro il corpo dell' *if,* possiamo poi specificare ulteriori condizioni basate sulle funzioni già viste per *conditions.js*, oppure sulle variabili definite da noi:

```
if (f_action("_ruscello14_", "usa con" , "_borraccia00_" ))
{
  if ( _vBorracciaPiena_ == 0 )
  {
    f_show("Hai riempito la borraccia.");
    _vBorracciaPiena_ = 1;
```

```
  }
  else
  {
    f_show("E' già piena!");
  }

  return true;
}
```

La condizione quindi verrà eseguita solo se gli oggetti coinvolti sono la borraccia e il ruscello e l'azione è "usa con". In realtà la funzione accetta anche l'inversione degli oggetti: andrebbe bene sia "ruscello – usa con – borraccia" che il viceversa "borraccia – usa con – ruscello".

Nel corpo della condizione, che viene eseguito quando la condizione è verificata, viene valorizzata la variabile *_vBorracciaPiena_* e visualizzato un messaggio in base allo stato della borraccia.

L'altra cosa importante è il comando *return true* che chiude il blocco dell'*if*. **Tutte le volte che infatti un'azione viene gestita, dobbiamo ricordarci di scrivere l'istruzione *return true* o l'azione verrà erroneamente valutata e risolta anche dal motore del gioco (che risponderebbe coi messaggi di default)**.

Si noti che la condizione di esempio appena vista non richiede che il giocatore sia in una particolare stanza in quanto l'oggetto *_ruscello14_* è in questo caso un oggetto fisso, che può essere presente solo nella stanza appropriata.

Ecco un altro esempio. Catturiamo l'azione "accendino – usa con – torcia" tenendo conto se il giocatore è nelle stanze *_rCantina_* o *_rGrotta_* oppure no*:*

```
if (f_action("_accendino10_", "usa con", "_torcia00_"))
{
  if ((f_isplayerin("_rCantina_")) || (f_isplayerin("_rGrotta_")))
  {
    // qui arrivo se il giocatore è in cantina o nella grotta
    ...
  }
  else
  {
    // qui arrivo se il giocatore è in qualsiasi altra stanza
    ...
  }
}
```

Per la maggior parte dei casi useremo quindi *f_action,* ma se vogliamo costruire delle condizioni particolari per catturare una certa gamma di azioni, possiamo usare le seguenti funzioni:

- *f_obj1( ) : restituisce identificatore oggetto toccato nel testo*
- *f_verb( ) : restituisce l'azione*
- *f_obj2( ) : restituisce identificatore oggetto scelto nell'eventuale sotto-menu dell'azione oppure è uguale a "" se non esiste sotto-menu*

Ad esempio, volendo catturare tutte le azioni sull'oggetto *_pistola00_* basta scrivere:

```
if ( f_obj1() == "_pistola00_" )
{
  ...
}
```

Poi si può pensare di gestire i verbi differenti all'interno del corpo della condizione:

```
if ( f_obj1() == "_pistola00_" )
{
        if (f_verb == "spara")
        {
                ...
                return true;
        }
        else if (f_verb == "scarrella")
        {
                ...
                return true;
        }
}
```

Anche azioni di direzione possono essere gestite in modo manuale se non è stato fatto in modo automatico tramite le proprietà *direction* e *desc_direction*. Ad esempio, immaginiamo che *_porta08_* abbia nel proprio menu (creato manualmente) l'azione "entra". Vogliamo che se il giocatore esegue questa azione l'esito dipenda dall'avere con se o no un certo oggetto:

```
if (f_action("_porta08_","entra"))
{
       //  se sto indossando il medaglione...
        if (f_istaken("_medaglione11_"))
        {
                f_show("Un fascio di luce colpisce il medaglione... Riesci a passare incolume.");
                // sposta il giocatore nell'ambiente _rStanzaFinale_
                f_moveplayer("_rStanzaFinale_");
                return true;
        }
        else
        {
                f_show("Un fascio di luce ti colpisce in pieno petto pietrificandoti...");
                // fine gioco
```

```
                f_lose();
                return true;
        }
}
```

L'aspetto più interessante è quello della gestione di oggetti "gemelli" che rappresentato il diverso stato di uno stesso oggetto. Ad esempio, immaginiamo di aver definito due oggetti gemelli in *objects.txt*:

```
<_torciaaccesa00_>
:examinable takeable usable
:desc_short: torcia
:desc_long: una torcia accesa
:desc_examine: E' accesa.
:room: _rOUT_
:weight: 5
:menu:
++”spegni”;

<_torciaspenta05_>
:examinable takeable usable
:desc_short: torcia
:desc_long: una torcia spenta
:desc_examine: E' spenta.
:room: _rSala_
:weight: 5
:menu:
++”accendi”;
```

All'inizio il giocatore troverà la torcia spenta nella stanza *_rSala_* .

In *action.js* gestiremo le azioni "accendi" e "spegni":

```
if (f_action("_torciaspenta05_","accendi"))
{
   // il giocatore ha l'accendino!
   if (f_istaken("_accendino00_"))
   {
     f_show("Hai acceso la torcia!");

     // facciamo "comparire" la torcia accesa
     // nella stessa locazione della torcia spenta
```

```
     f_move("_torciaaccesa00_", f_whereis("_torciaspenta05_"));


     // facciamo "scomparire" la torcia spenta...
     f_moveout("_torciaspenta05_");


     return true;
   }
}


if (f_action("_torciaaccesa00_","spegni"))
{
   f_show("Hai spento la torcia!");


   // facciamo "comparire" la torcia spenta
   // nella stessa locazione della torcia accesa
   f_move("_torciaspenta05_", f_whereis("_torciaaccesa00_"));


   // facciamo "scomparire" la torcia accesa...
   f_moveout("_torciaaccesa00_");


   return true;
}
```

Il giocatore non si accorgerà dell'esistenza di due oggetti distinti ma sembrerà che ci sia un oggetto unico che cambia stato (ovvero descrizioni e proprietà) in base alle azioni che facciamo su di esso. Un modo più veloce per fare la stessa cosa è usare la funzione *f_swap(obj1, obj2)* che scambia di posizione i due oggetti indicati. Lo stesso codice diventa quindi:

```
if (f_action("_torciaspenta05_","accendi"))
{
   if (f_istaken("_accendino00_"))
   {
     f_show("Hai acceso la torcia!");
     f_swap("_torciaaccesa00_", "_torciaspenta05_");
     return true;
   }
}


if (f_action("_torciaaccesa00_","spegni"))
{
   f_show("Hai spento la torcia!");
   f_swap("_torciaaccesa00_", "_torciaspenta05_");
   return true;
}
```

E' arrivato il momento di modificare il file *action.js* per l'avventura “L'Occhio Purpureo”. Apriamo il file e modifichiamolo in questo modo:

```
// se nuotiamo nel fiume
if (f_action("_fiume01_","nuota"))
{
  // non abbiamo mai nuotato
  if ( _vNuotatoInFiume_ == 0 )
  {
    // per sapere che abbiamo già nuotato
    _vNuotatoInFiume_ = 1;
    // abbiamo il mattone con noi
    if (f_istaken("_mattone03_"))
    {
      f_show("Oh no!!! Sei troppo pesante e vieni risucchiato in un gorgo...");
      // vai alla schermata di fallimento...
      f_lose();
    }
    else
    {
      f_show("Raggiungi la riva opposta dove trovi una pistola...
              A fatica la riporti dall'altra parte asciutta. Ora è ai tuoi piedi.");
      // compare la pistola nella stanza
      f_move("_pistolaCarica00_","_rFiume_");
    }
  }
  else
  {
    f_show("Una nuotata ti è bastata...");
  }
  return true;
}

// se si esamina edificio, si legge una scritta con il numero
// di volte che bisogna tirare la catena del trono per vincere
if (f_action("_edificio05_","esamina"))
{
  var lScritta =
    "Sulla parete c'è una scritta scolorita: << ZUCAR " + _vTirareCatena_ + " ";
                  // zucar in triestino significa tirare
```

```
  if ( _vTirareCatena_ == 1 )
    // zucar 1 volt-A
    lScritta = lScritta + "VOLTA ! >> ...";
  else
    // zucar 2 o 3 volt-E
    lScritta = lScritta + "VOLTE ! >> ...";

  f_show(lScritta);

  return true;
}

// se ci si siede sul trono appare un laconico messaggio
if (f_action("_trono12_","siedi"))
{
  f_show("Ti senti un re! Però è scomodo.");
  return true;
}

// accendi lampada
if (f_action("_lampadaSpenta00_","accendi"))
{
  f_show("Hai acceso la lampada!");
  // scambiamo di posizione la lampada accesa e quella spenta
  f_swap("_lampadaAccesa00_", "_lampadaSpenta00_");
  return true;
}
// spegni lampada
if (f_action("_lampadaAccesa00_","spegni"))
{
  f_show("Hai spento la lampada!");
  // scambiamo di posizione la lampada accesa e quella spenta
  f_swap("_lampadaAccesa00_", "_lampadaSpenta00_");
  return true;
}

// usa lampada con mattone -> la lampada si rompe
if (f_action("_lampadaAccesa00_","usa con","_mattone03_"))
{
  f_show("Hai rotto la lampada!");
  // scambiamo di posizione la lampada accesa e quella rotta
```

```
  f_swap("_lampadaAccesa00_", "_lampadaRotta00_");
  return true;
}
if (f_action("_lampadaSpenta00_","usa con","_mattone03_"))
{
  f_show("Hai rotto la lampada!");
  // scambiamo di posizione la lampada spenta e quella rotta
  f_swap("_lampadaSpenta00_", "_lampadaRotta00_");
  return true;
}

// se si vuota il sacchetto compaiono le biglie
if (f_action("_sacchettoPieno05_","vuota"))
{
  // sostituiamo il sacchetto vuoto a quello pieno
  f_swap("_sacchettoVuoto00_", "_sacchettoPieno05_");

  // facciamo comparire le biglie nella stanza in cui siamo
  f_move("_biglie00_", f_whereisplayer());

  f_show("Vuoti il sacchetto: delle biglie si spargono a terra.");

  return true;
}

// tiriamo catena del trono:
// se si sbaglia numero di volte o non si ha l'occhio, si viene sciaquati via
// se il numero di volte è giusto ma non abbiamo indossato guanti, si muore fulminati
if ((f_obj1() == "_catena12_") && (f_verb() == "tira"))
{
  var lx = 0;
  if (f_obj2() == "una volta")
    lx = 1;
  else if (f_obj2() == "due volte")
    lx = 2;
  else if (f_obj2() == "tre volte")
    lx = 3;
  if (
     // il numero di volte che tiriamo la catena è giusta
       ( lx == _vTirareCatena_ )
         &&
       // e abbiamo l'occhio
```

```
      f_istaken("_occhio00_")
    )
  {
    // se indossiamo i guanti
    if (f_isworn("_guanti11_"))
    {
      f_show("...Un passaggio segreto si apre verso un sentiero fiorito...");
      // vai alla schermata di successo!
      f_win();
    }
    else
    {
      f_show("BZZZZZZZZZZZZ!!! Una scarica ti fulmina all'istante...");
      // vai alla schermata di fallimento!
      f_lose();
    }
  }
  else
  {
    f_show("AAAARGH!!! Il pavimento si apre e dall'alto una cascata d'acqua ti trascina via!...");
    // vai alla schermata di fallimento!
    f_lose();
  }
  return true;
}

// spariamo: l'esito dipende se l'ispettore è nella stanza o no
// oppure usiamo la pistola carica con l'ispettore
if (
    // pistola carica - spara
    f_action("_pistolaCarica00_","spara")
    // pistola carica - usa con - ispettore
    || f_action("_pistolaCarica00_","usa con","_ispettore00_")
    // ispettore - spara
    || f_action("_ispettore00_","spara")
    )
{
  // se l'ispettore è nella nostra stanza
  if (f_ishere("_ispettore00_"))
  {
    f_show("Hai ucciso l'ispettore delle tasse!!!");
    // facciamolo scomparire dal gioco
```

```
    f_moveout("_ispettore00_");
    // compare il suo cadavere
    f_move("_cadavere00_",f_whereisplayer());
    // ispettore è morto
    _vStatoIspettore_ = 2;
  }
  else
  {
    f_show("Hai sprecato l'unico colpo...");
  }
  // in entrambi i casi sostituisco la pistola scarica a quella carica
  f_swap("_pistolaScarica00_", "_pistolaCarica00_");
  return true;
}

// lanciamo il mattone: l'esito dipende se l'ispettore è nella stanza o no
// oppure usiamo il mattone con l'ispettore oppure lo colpiamo col mattone
if (
    // mattone - lancia
    f_action("_mattone03_","lancia")
    // mattone - usa con - ispettore
    || f_action("_mattone03_","usa con","_ispettore00_")
    // ispettore - colpisci
    || f_action("_ispettore00_","colpisci")
    )
{
  // se l'ispettore è nella nostra stanza
  if (f_ishere("_ispettore00_"))
  {
    // schiva il mattone
    f_show("L'ispettore delle tasse ha schivato il mattone!!!");
    // il mattone resta nella stanza
    f_move("_mattone03_",f_whereisplayer());
  }
  else
  // altrimenti non facciamo nulla
  {
    f_show("E dove lo vorresti lanciare?...");
  }
  return true;
}
```

Il file è abbastanza commentato quindi il codice dovrebbe essere chiaro. Vediamo comunque di analizzare le parti più significative.

Interessante è l'azione "edificio – esamina" che mostrerà un messaggio che rivelerà al giocatore il numero di volte che è stato stabilito come corretto per vincere il gioco tirando la catena (in *_vTirareCatena_*). Il messaggio da mostare con *f_show* viene composto da testo e dal contenuto della variabile stessa.

Le azioni "lampada spenta – accendi" e "lampada accesa – spegni" mostrano come far comparire un oggetto gemello e far sparire l'altro tramite *f_swap.* Allo stesso modo viene anche gestita l'azione di rottura della lampada.

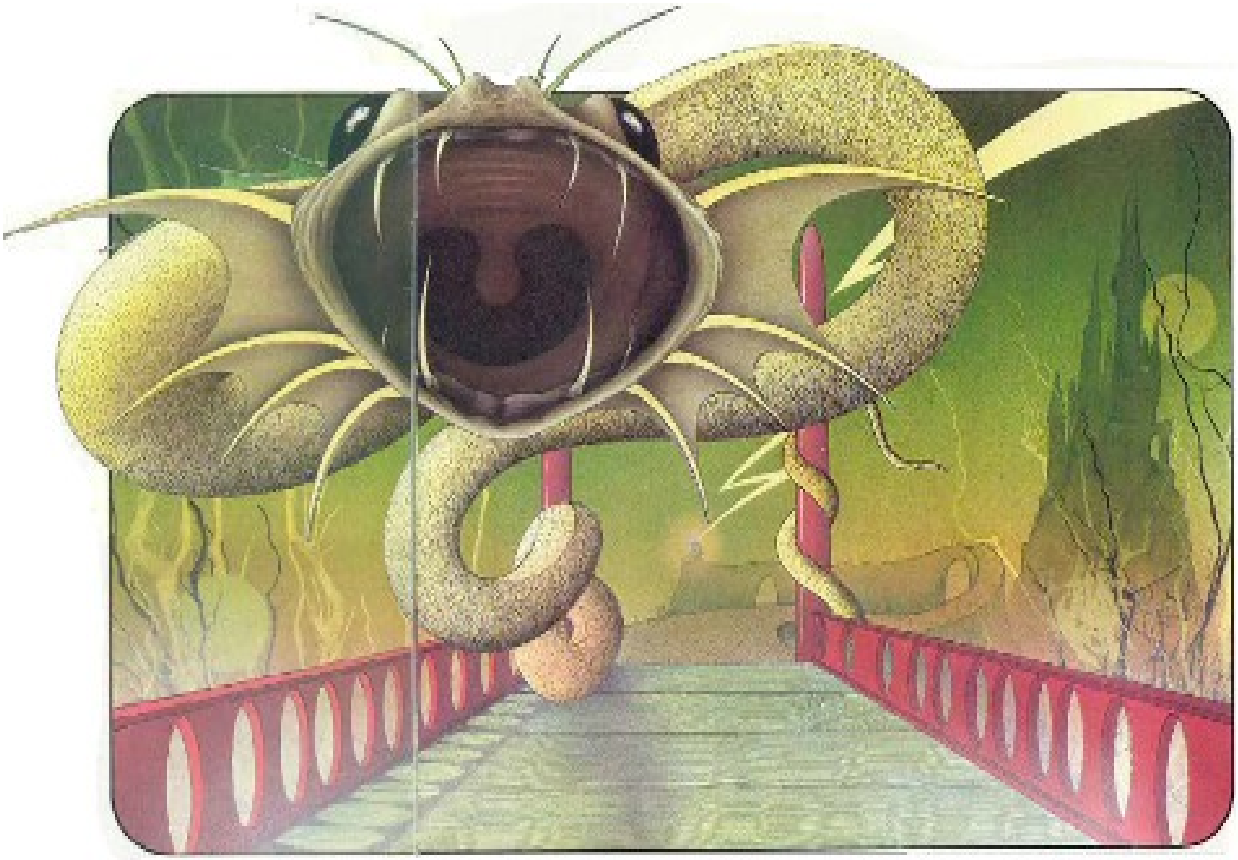
La condizione più complessa è quella dell'azione "catena – tira" che non usa *f_action* ma valuta *f_obj1, f_verb, f_obj2* per poter distinguere tramite *f_obj2* quante volte è stata tirata. In base a questo, al fatto che si abbia l'Occhio e indossato i guanti, si hanno diversi esiti.

Le penultima condizione è relativa allo sparare all'ispettore (possibile in vari modi) che lo fa uscire dal gioco (al suo posto compare il suo cadavere). Se se si spara quando non c'è, si spreca il colpo. In entrambi i casi la pistola carica sparisce e al suo posto compare quella scarica.

L'ultima condizione è relativa al lancio del mattone, che l'ispettore schiverà (se presente) senza conseguenze o che porterà ad un messaggio che ci invita a pensare meglio a quello che facciamo.

Con questa modifica l'avventura dell'Occhio Purpureo è pronta! Tutti i file di progetto sono stati modificati e non ci resta che generare l'avventura e provarla.

I prossimi capitoli saranno dedicati alla risoluzione di errori e problemi nella scrittura delle nostre avventure.

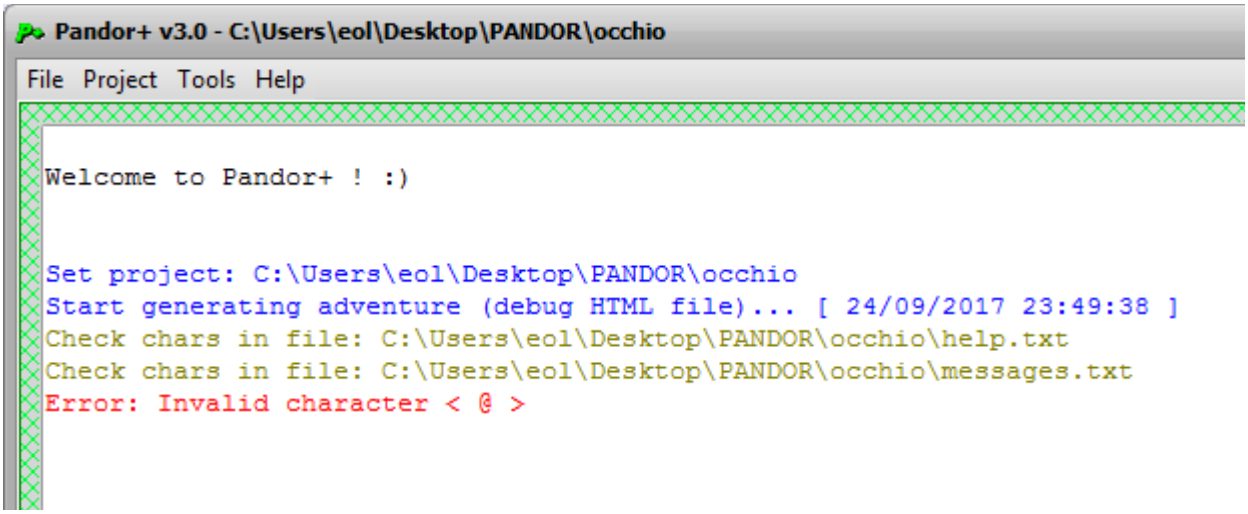


Warning ed Error durante la generazione

Man mano che sviluppate la vostra avventura conviene generare spesso il file di debug e correggere eventuali errori.

Qui di seguito una lista dei possibili messaggi che bloccano la generazione dell'avventura. Quando si presenta un errore, va risalito il log per individuare il file nel quale si è verificato.

Ad esempio, se introduciamo un carattere @ in *messages.txt* :



l'errore è segnalato sotto la riga indicante l'operazione di verifica dei caratteri usati in *messages.txt*

Di seguito l'elenco completo di tutti gli errori possibili e del loro significato.

*duplicated definition/object/room/variable: _xxx_*

E' stato usato lo stesso identificatore per definire oggetti, stanze o variabili

Files invalid or corrupted!

Uno dei file di progetto è stato modificato in modo errato

process stopped by user

E' stato premuto "stop" durante la generazione dei file HTML

unable to find project folder!

La cartella di progetto non è disponibile

failed object merging: xxx

La descrizione di una stanza comprende oggetti non definiti correttamente

Invalid character: < X >

E' stato usato un carattere (X) non valido

invalid definition: xxx

E' stato usato un identificatore non valido per definire oggetti, stanze o variabili

Invalid file name: xxx

Nome di file invalido

maximum number of objects reached

Sono definiti più di 300 oggetti nell'avventura

*missing objects specification: _xxx_*

L'oggetto _xxx_ non è stato definito in *objects.txt*

Project name is already used!

Esiste già una cartella con questo nome di progetto

start room is not unique

Solo una stanza può essere scelta come stanza iniziale del gioco

syntax error: xxx

Errore di sintassi nella definizione di un oggetto, stanza o variabile

Unable to create new project folder or files!

Impossibile creare la cartella di progetto o i file di progetto

Unable to generate HTML files!

Impossibile generare i file HTML dell'avventura

Unable to merge: xxx

Impossibile unire il file di progetto xxx al file HTML dell'avventura

*unknown :found: object specification: _xxx_*

Oggetto _xxx_ non definito

*unknown definition: _xxx_*

Identificatore sconosciuto

unknown direction in rooms.txt: xxx

Identificatore stanza sconosciuto nella specifica di una direzione

unknown object property: xxx

Proprietà invalida in *objects.txt*

*unknown room: _xxx_*

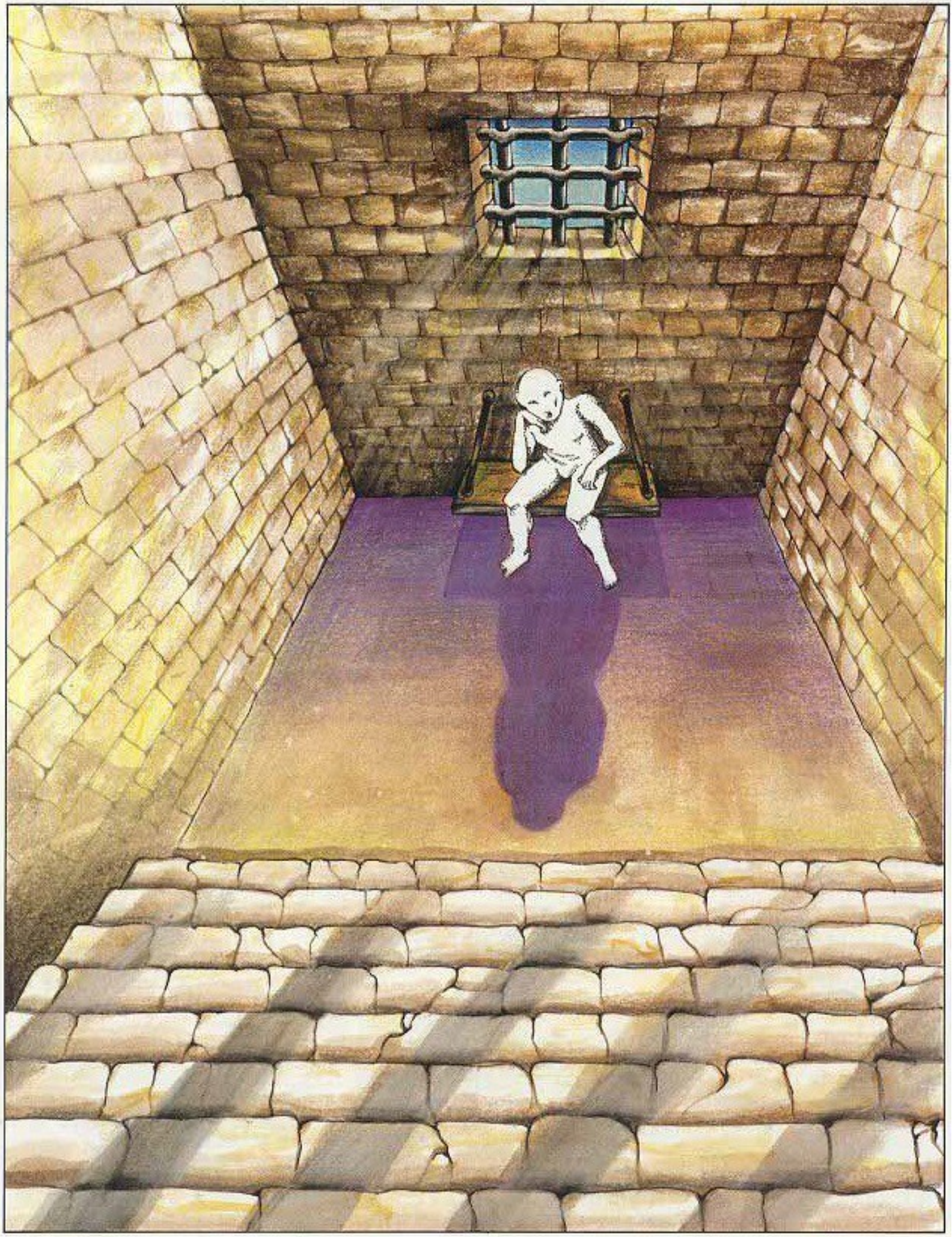
Identificatore di stanza sconosciuto

*unknown variable: _xxx_*

Variabile sconosciuta

wrong propriety for fixed object: xxx

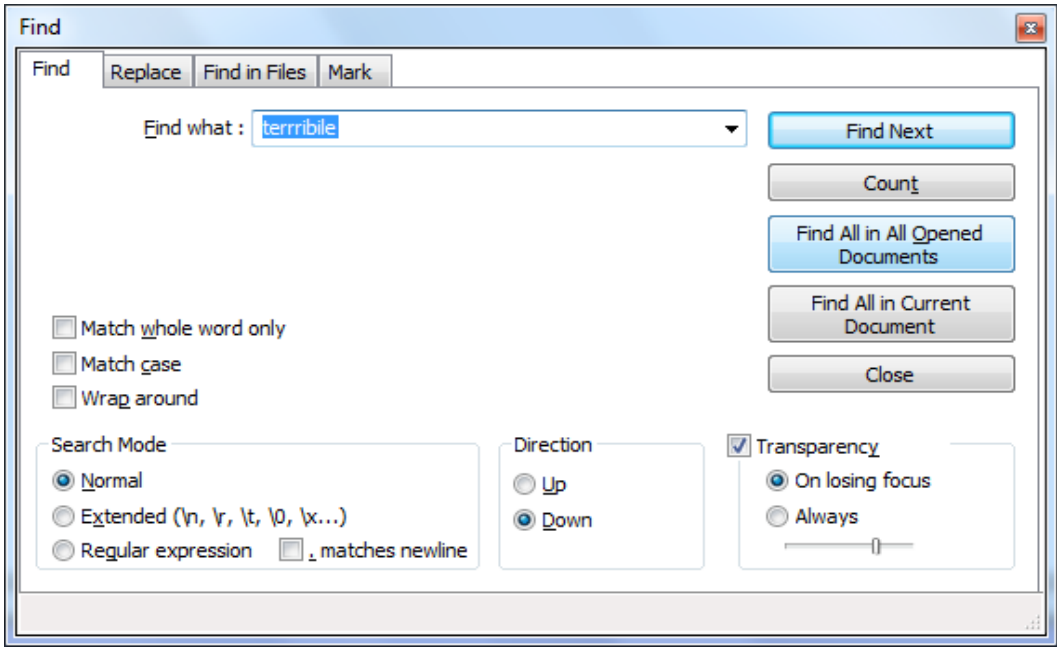
Uso di una proprietà non valida per un oggetto di tipo fisso



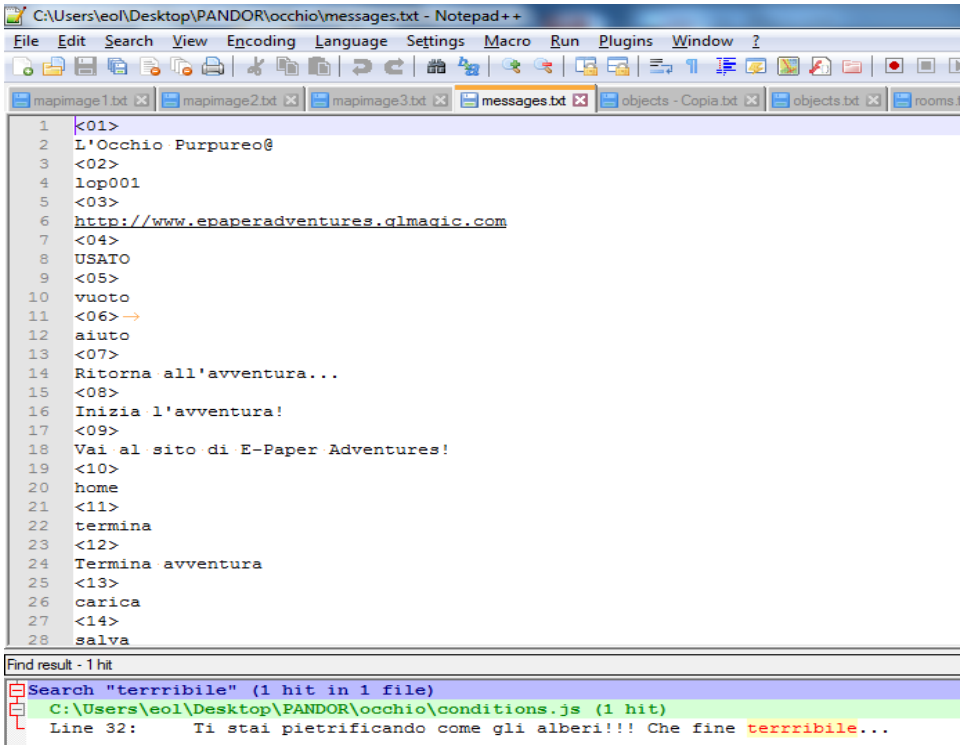
Il debug dell'avventura

Anche se i file dell'avventura vengono generati, non è detto che non vi siano refusi, errori "logici" o errori nel codice scritto.

I refusi sono semplici da correggere: basta modificare i file di progetto che li contengono e rigenerare l'avventura. Conviene aprire tutti i file (tranne quelli generati) in Notepad++ e cercare il testo del refuso tramite l'opzione *Search → Find in files* :



Tramite "Find All in All Opened Documents" troveremo poi in basso tutte le posizioni in tutti i file in cui si trova il refuso. Basta fare doppio-click sulla linea e andremo direttamente al file e alla posizione dove effettuare la correzione.



Per gli errori logici, conviene utilizzare le informazioni aggiuntive di debug che compaiono aprendo in Firefox il file generato *avventura_debug.html.*

Infatti, sotto il normale testo del gioco, vi è una parte composta dalla posizione del giocatore e 3 elenchi:

La posizione del giocatore è specificata con *Player is in room #N [_xxx_]*. Il giocatore si trova nella stanza di identificatore *_xxx_*, numero N.

Il gruppo (1) contiene le variabili definite da noi e il loro valore attuale. Questo è molto utile per capire se le variabili stanno "funzionando" come ci aspettiamo in base a quello che il giocatore ha fatto.

Il gruppo (2) elenca gli oggetti mobili dell'avventura. Accanto ad essi possiamo trovare scritto:

- *is out*: significa che l'oggetto è fuori dal gioco
- *is taken*: significa che l'oggetto è trasportato dal giocatore
- *is worn*: significa che l'oggetto è indossato dal giocatore
- *is in room #N [_xxx_]*: significa che l'oggetto si trova nella stanza *_xxx_* che è la numero *N*
- *[found]* : significa che l'oggetto è stato esaminato ed è comparso o sono comparsi gli oggetti specificati nella sua proprietà *found*
- *[talked]* : significa che il giocatore ha parlato almeno una volta con l'oggetto (definito con la proprietà *talkeable)*

Il gruppo (3) è relativo agli oggetti fissi dell'avventura, quindi normalmente vedremo semplicemente scritto *is fixed.* Tuttavia, se abbiamo modificato la posizione di questi oggetti nel codice (cosa non corretta per gli oggetti fissi), potremo avere:

- *is fixed but in room #N [_xxx_] → ????* : l'oggetto si trova in una stanza
- *is fixed but taken/worn → ERROR* : l'oggetto è trasportato o indossato

Altre informazioni invece possono essere:

- *[found]* : significa che l'oggetto è stato esaminato ed è comparso o sono comparsi gli oggetti specificati nella sua proprietà *found*
- *[talked]* : significa che il giocatore ha parlato almeno una volta con l'oggetto (definito con la proprietà *talkeable)*

Esiste anche un quarto gruppo di informazioni di debug, relativo alle mappe analizzeremo alla fine del tutorial.

Durante i nostri debug col browser, tutte le volte che modificheremo uno o più dei nostri file di progetto sarà necessario resettare la nostra avventura ovvero:

- salvare tutti i file di progetto con le modifiche
- rigenerare il file *xxx_debug.html* con Pandor+
- riaprire il file *xxx_debug.html* nel browser (se non aperto)
- eventualmente terminare l'avventura per riportarla all'inizio (comando *termina → conferma*)
- fare il refresh della pagina (con F5)

Nel caso di modifiche limitate ai testi o alle immagini, gli evenuali salvataggi effettuati (1,2,3 o 4) saranno ancora utilizzabili. Questo non sarà invece possibile nel caso in cui:

- modifichiamo codice in *counters.js , conditions.js, actions.js* (oltre ai testi descrittivi)
- modifichiamo/aggiungiamo/cancelliamo variabili
- modifichiamo/aggiungiamo/cancelliamo oggetti (oltre ai testi descrittivi)
- modifichiamo/aggiungiamo/cancelliamo stanze (oltre ai testi descrittivi)

Veniamo ora al debug in caso di errori nel codice scritto nei file *counters.js, conditions.js* e *actions.js* .

Il codice in questi file è fondamentalmente codice pseudo-javascript ,in quanto in esso sono introdotti alcuni elementi non javascript come ad esempio le nostre variabili definite in *variables.txt* .

Tuttavia il file HTML generato è completamente javascript, in quanto tutti i nostri file di progetto vengono "tradotti" in codice javascript o HTML e assemblati nel file generato.

Se vi sono errori nel codice della nostra avventura vedremo comportamenti errati nel gioco (testi sbagliati, oggetti fuori posto, azioni che non fanno quello che devono…) oppure la pagina si bloccherà e rimarrà bianca o verremo avvertiti da messaggi di errore del browser stesso.

Possiamo però utilizzare il debugger di FireFox per individuare dove si trova l'errore o scoprire il motivo di un comportamento errato dell'avventura. Il file da utilizzare è sempre e solamente quello di debug, in quanto quello di release ha i testi criptati e sono rimossi i commenti.

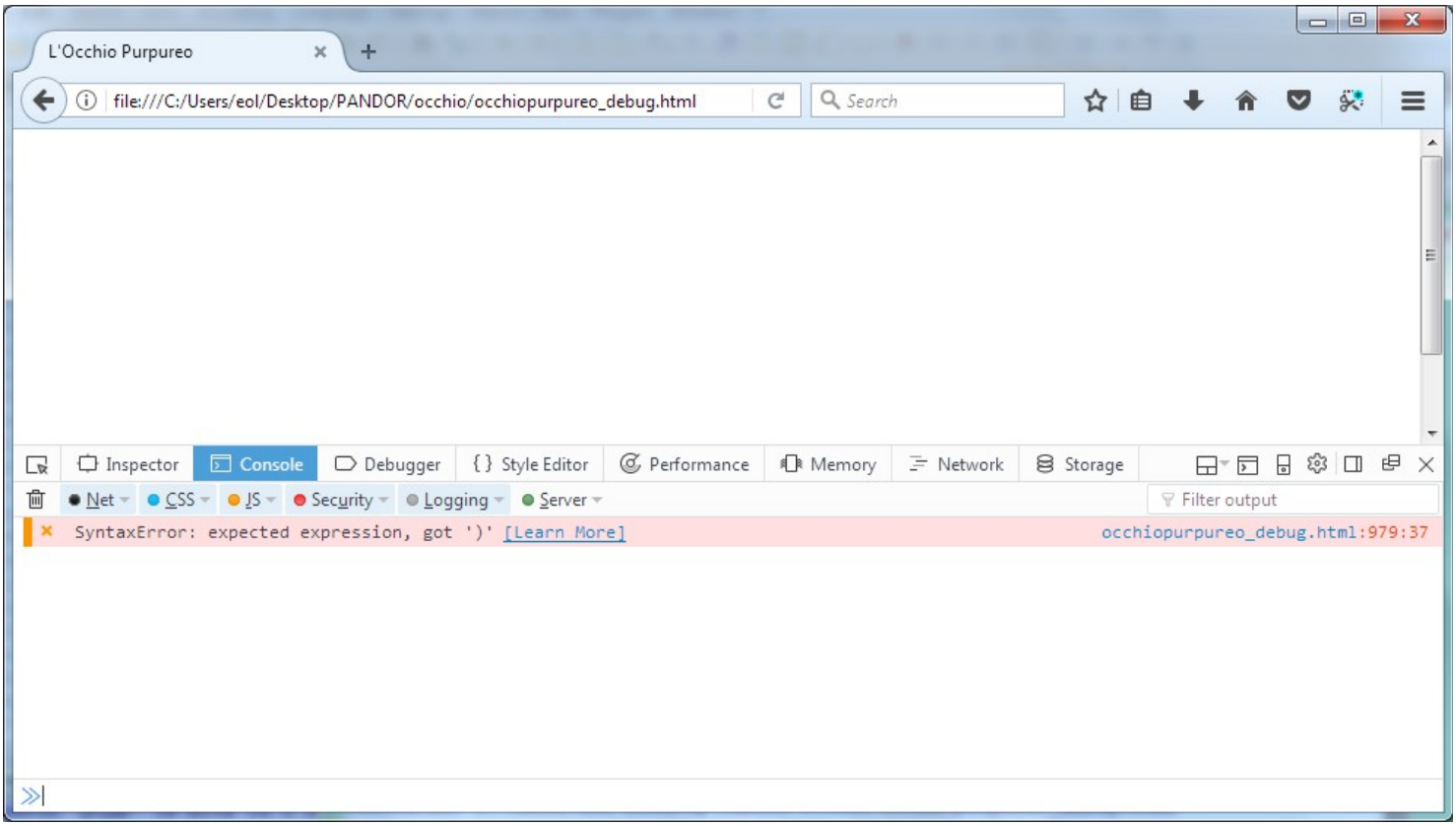
Proveremo ora a creare qualche errore nell'avventura dell'Occhio Purpureo per vedere come individuarli e correggerli.

Apriamo *actions.js* e modifichiamolo in questo modo:

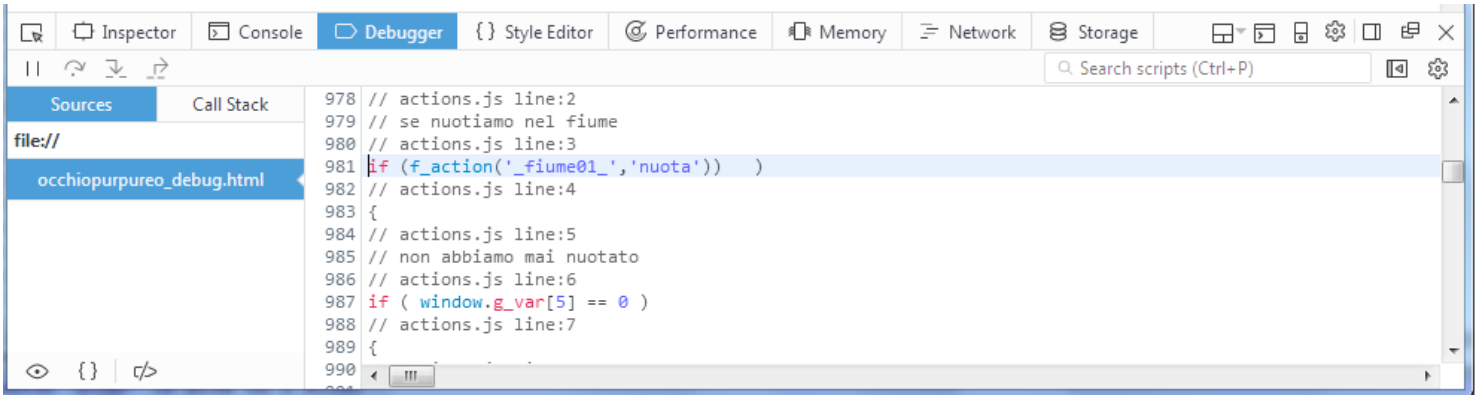
```
if (f_action("_fiume01_","nuota"))     )
{
  // non abbiamo mai nuotato
  if ( _vNuotatoInFiume_ == 0 )
  {
```

Abbiamo cioè aggiunto una parentesi in più alla condizione. Salviamo *actions.js*, rigeneriamo e resettiamo l'avventura e vedremo che la pagina del gioco sarà completamente bianca!

Iniziamo dunque il nostro debug. In FireFox, premiamo F12. Attendiamo e nella parte bassa del browser apparirà un menu orizzontale a TAB, clicchiamo sul TAB "Console":



Ecco che ci viene indicato che c'è un errore di sintassi dovuto ad una parentesi non attesa. Cliccando a destra su "occhiopurpureo_debug.html" andremo alla linea dove si trova l'errore.:



Pandor+ ha aggiunto, subito sopra la linea effettiva di codice, una linea di commento che indica la posizione della linea sottostante, in questo caso:

// actions.js line:3

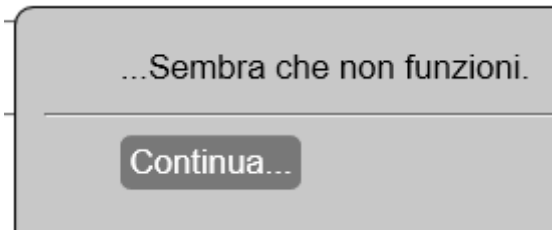
Ovvero, l'errore si trova alla linea 3 del file *actions.js* . Andrà quindi corretto in Notepad++.

Una volta individuato e eliminato l'errore, sarà necessario resettare l'avventura e riprovare.

Ci sono però errori che non sono altrettanto evidenti, ad esempio modifichiamo *actions.js* in questo modo:

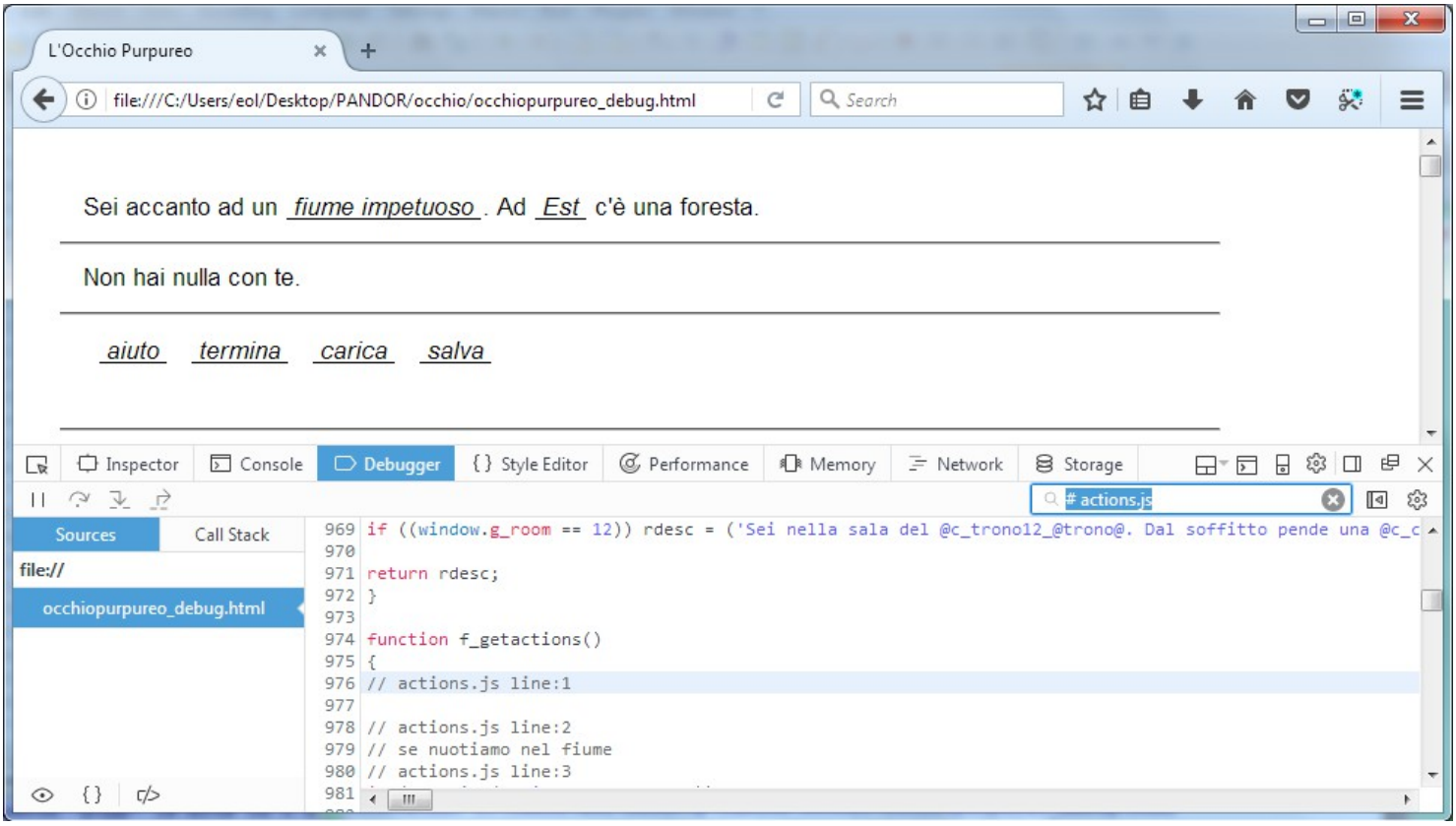
```
if (f_action("_fiume01_","muota"))
{
  // non abbiamo mai nuotato
  if ( _vNuotatoInFiume_ == 0 )
  {
```

Abbiamo cioè scritto "muota" invece di "nuota". Se ora resettiamo l'avventura e proviamo a giocare, e proviamo a nuotare nel fiume, vedremo che invece di vedere i messaggi attesi il gioco mostrerà un messaggio:

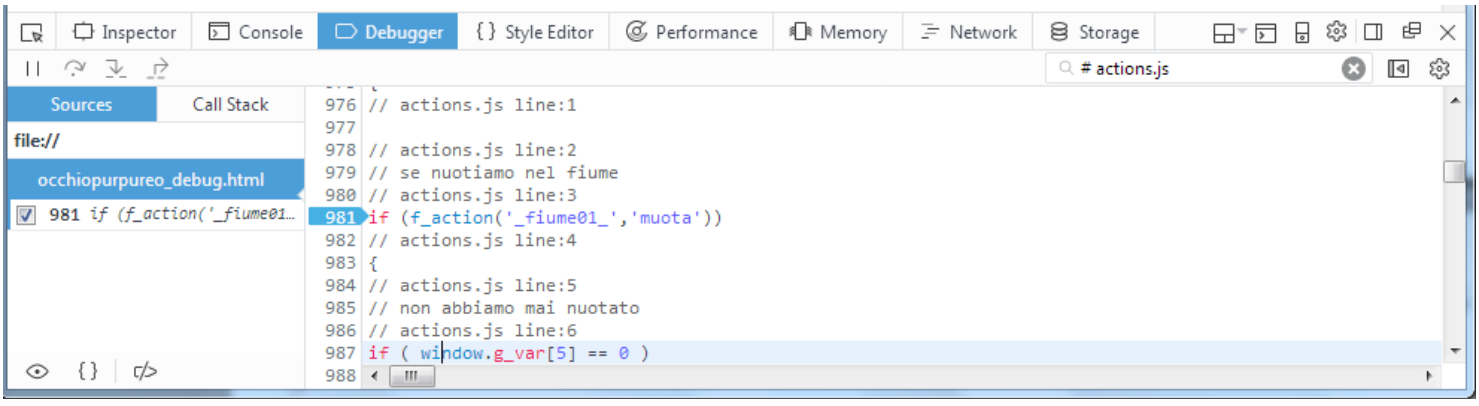


Questo perché l'azione "fiume – nuota" non è stata catturata in *actions.js*.

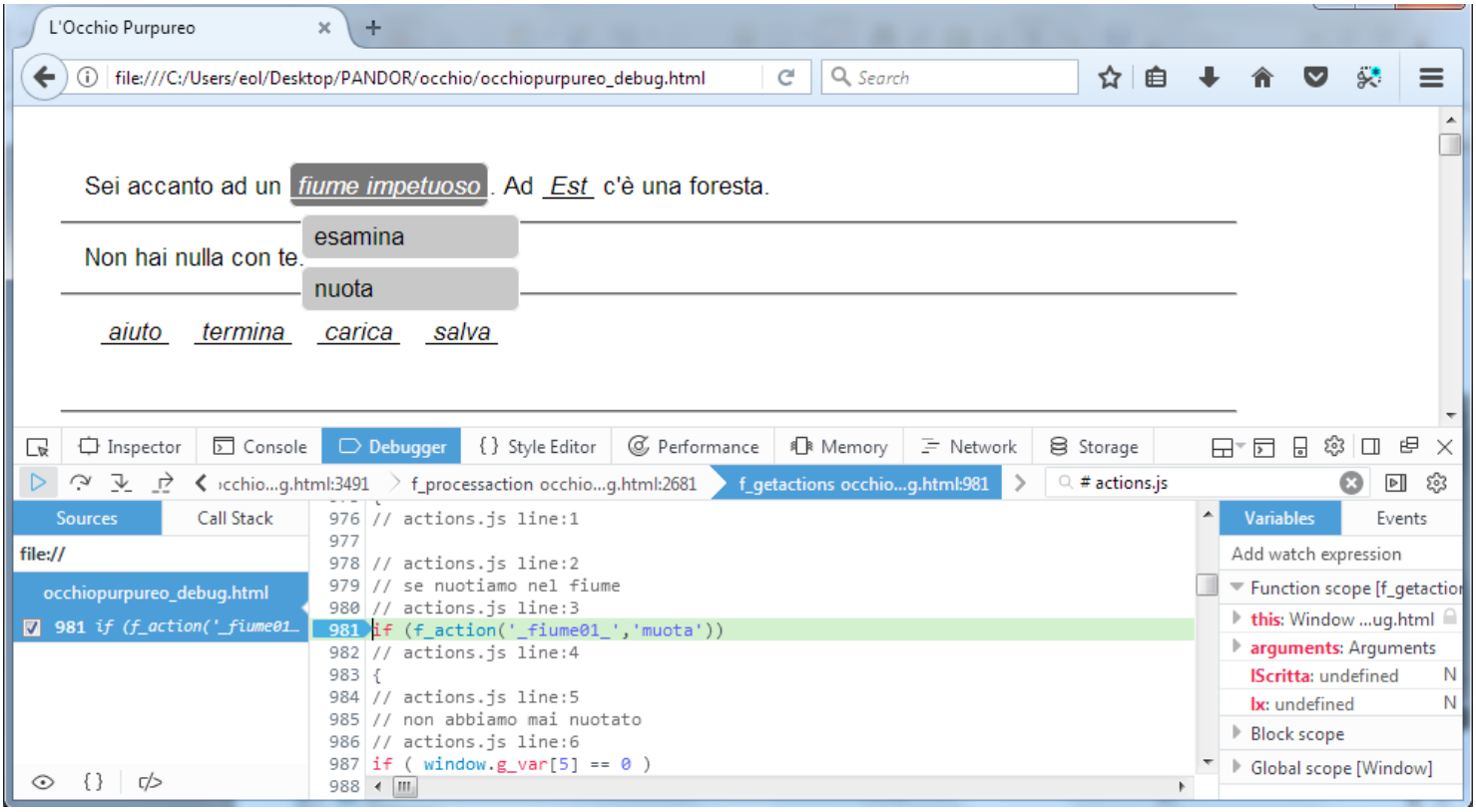
Per capire cosa succede, premiamo nuovamente F12, e nel TAB "Debugger", mettiamo nel campo di ricerca (in alto a destra) la frase di ricerca "# actions.js" per cercare il pezzo di codice in cui si trova la parola *actions.js*.



Scrollando in basso, raggiungiamo la condizione di nostro interesse. Clicchiamo a sinistra della condizione che vogliamo esaminare, sul numero di linea del debugger: in questo modo aggiungeremo un "breakpoint" per far fermare lì l'esecuzione del codice javascript e capire se stiamo o no entrando nel corpo della condizione.



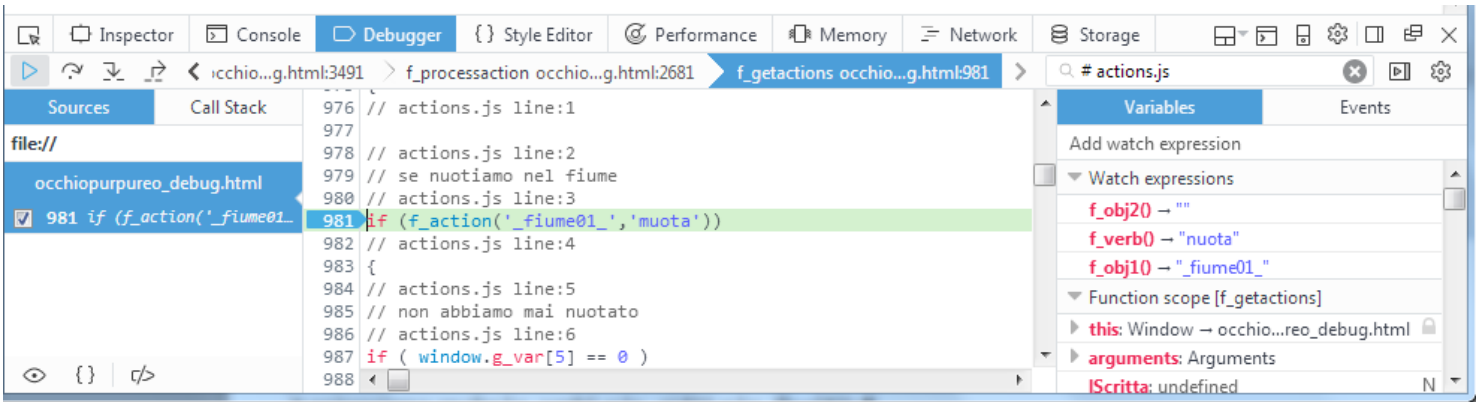
Proviamo ora a ripetere l'azione "fiume – nuota". La pagina si bloccherà e vedremo che verremo posizionati proprio sulla linea del breakpoint.



A questo punto possiamo usare i pulsanti in alto a sinistra per verificare passo-passo il nostro codice.

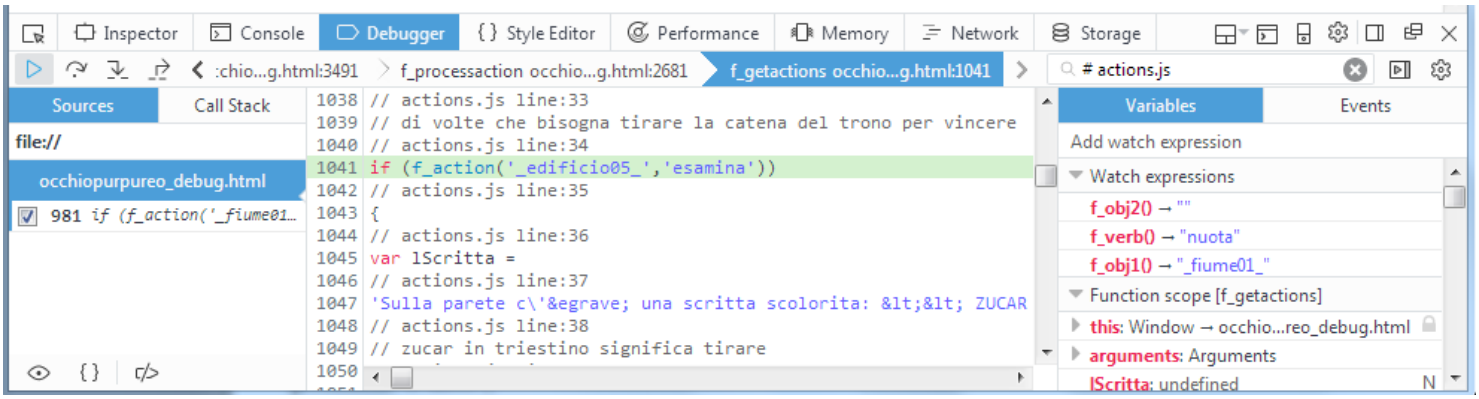
Prima di farlo però, clicchiamo a destra dove è scritto "Add watch expression" e scriviamo:

- "f_obj1( )" e poi invio
- "f_obj2( )" e poi invio
- "f_verb( )" e poi invio



Il debugger ci mostrerà l'identificatore dell'oggetto su cui stiamo agendo (ovvero il risultato della funzione *f_obj1*), l'azione (*f_verb*) e l'eventuale sotto-voce (*f_obj2*). Non resta che capire il perché non "entriamo" nella nostra condizione, visto che tutto sembra corretto. Premiamo il pulsante in alto:

Che esegue una linea di codice alla volta. Vedremo che il codice "salta" molto più sotto, saltando tutta la parte interna della nostra condizione.



Quindi significa che la condizione in cui ci aspettavamo di entrare non è appunto stata eseguita. A questo punto premiamo il tasto:

per far ripartire il gioco e ripetiamo l'azione "fiume – nuota". Il codice si bloccherà nuovamente sul nostro breakpoint. Un'ispezione visiva della linea porterà a individuare l'errore di scrittura e quindi a risolvere il problema.

Tipicamente metteremo i nostri breakpoints all'inizio delle nostre azioni o all'inizio delle nostre condizioni (per trovare il punto nel codice basta cercare "# conditions.js" o "# counters.js").

Possiamo anche scorrere il codice manualmente tramite la barra di scorrimento o cercare delle frasi usate nei commenti per arrivare più velocemente al punto che ci interessa.

Per terminare il debug, basta cliccare sulla "X" in alto a destra: Per togliere invece un breakpoint basta ricliccare sopra di esso.

Di solito dovremo correggere un errore alla volta. Via via che li correggiamo, val la pena cancellare il log degli errori nel TAB "Console", tramite il pulsante "cestino":

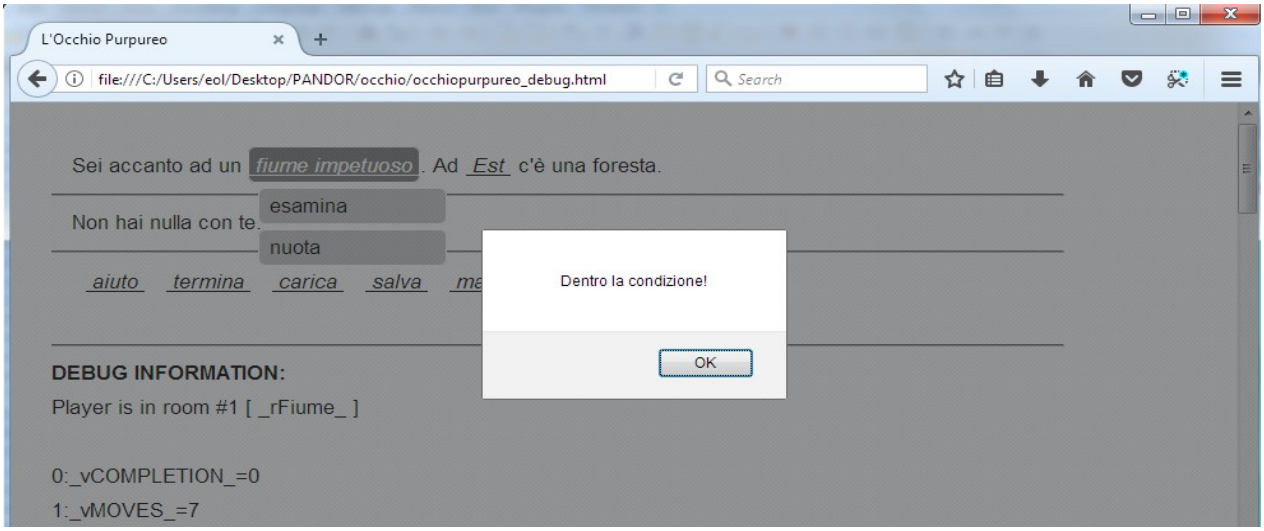
per poi rieffettuare l'azione di interesse o fare il refresh della pagina (F5) per rivedere l'errore corrente.

Altri sistemi per debuggare il nostro progetto è quello di utilizzare aggiungere nel nostro codice una chiamata alla funzione javascript *alert* che permette di visualizzare un messaggio popup con un contenuto voluto. Può essere comodo ad esempio per capire se stiamo "passando" in un certo punto del nostro codice (ad esempio in una condizione).

Ad esempio, modifichiamo l'azione "fiume – nuota" in *actions.js*:

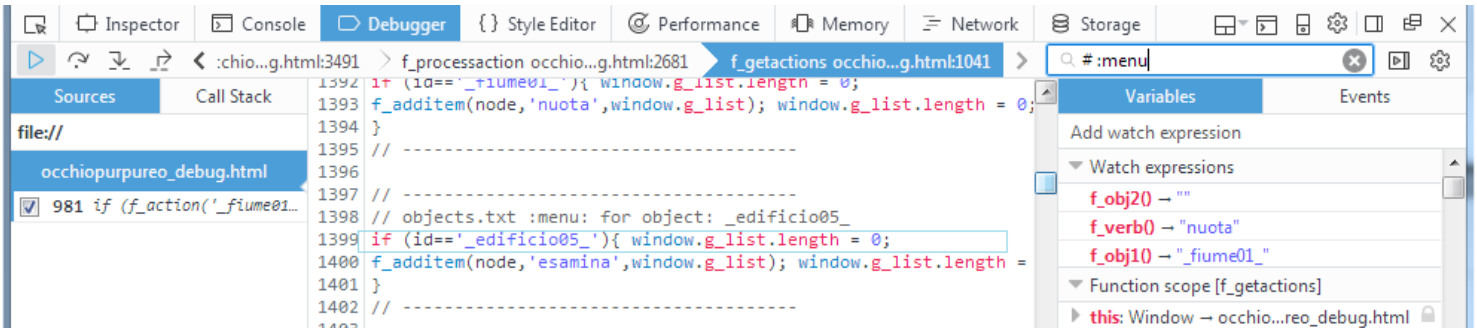
```
// se nuotiamo nel fiume
if (f_action("_fiume01_","nuota"))
{
  alert("Dentro la condizione!");

  // non abbiamo mai nuotato
  if ( _vNuotatoInFiume_ == 0 )
  {
    // per sapere che abbiamo già nuotato
```

Se ora resettiamo l'avventura e proviamo a fare "fiume – nuota" un paio di volte, vedremo apparire il messaggio popup.



Infine qualora un oggetto non si comportasse come ci attendiamo (mancano azioni, non è cliccabile, etc) può anche essere dovuto ad un errore in *objects.txt* nella parte della proprietà *menu* che può ospitare del codice.

In questo caso il browser potrebbe puntarci ad un errore in una zona di codice di questo tipo:



La riga di commento:

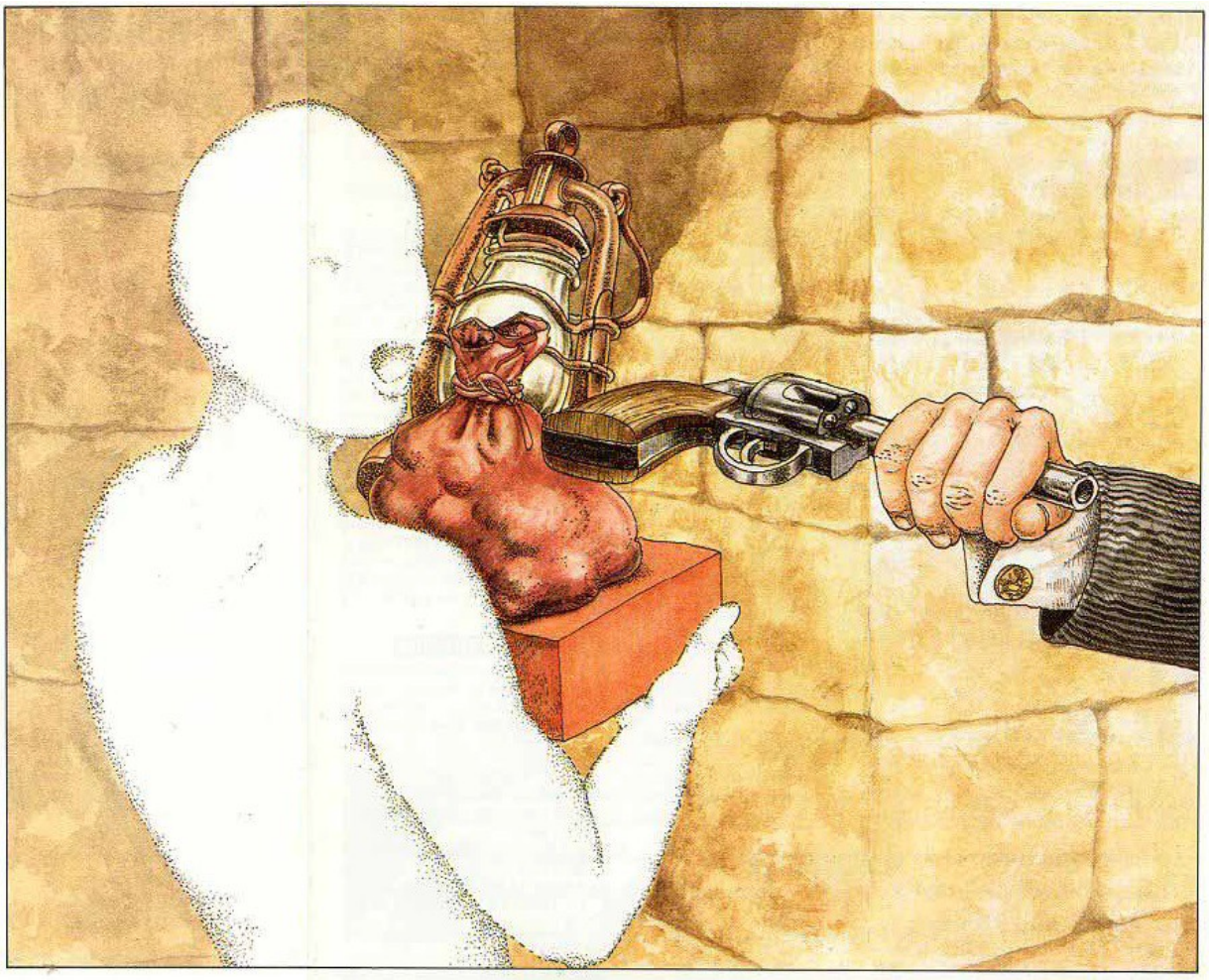
*// objects.txt :menu: for object: _edificio05_*

indica che il codice sottostante si trova in origine nella parte relativa alla proprietà *menu* dell'oggetto *_edificio05_* in *objects.txt*

Un'ultima nota su un errore molto subdolo: se nel codice sbagliamo il nome di una variabile, ad esempio al posto di *_vMOVES_* scriviamo *vMOVES_* (senza il primo _) , sia Pandor+ che il browser non segnaleranno nessun errore. In javascript infatti non è obbligatorio definire le variabili in modo esplicito e quindi, un'istruzione del tipo:

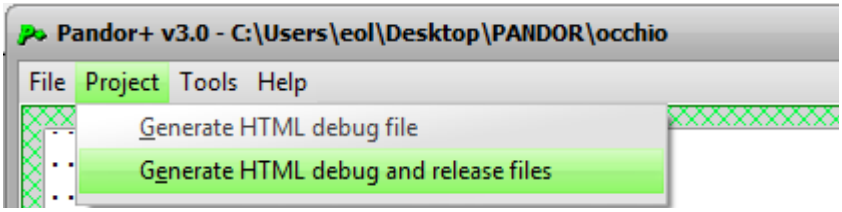
```
vMOVES_ = _vMOVES_ + 1;
```

causerà la creazione “al volo” di una nuova variabile *vMOVES_* , diversa da *_vMOVES_* e quindi l'istruzione non farà quello che ci aspettiamo. In caso di comportamenti anomali, controllate quindi le che tutte le variabili siano scritte correttamente.

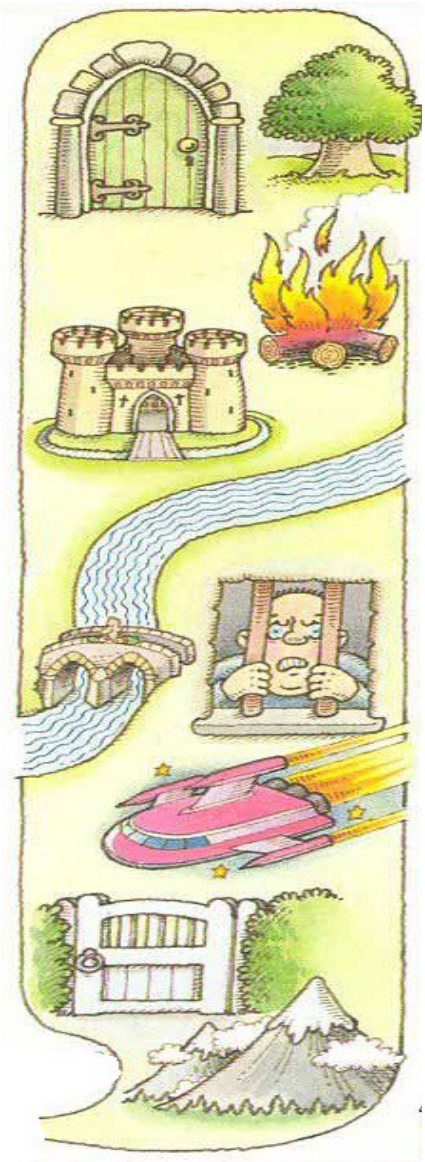


Finalizzare il gioco

Una volta che la nostra avventura è pronta e il file HTML di debug funziona correttamente, possiamo passare a generare la versione di release nella quale i testi del gioco saranno criptati e le informazioni di debug nascoste.



Generiamo quindi il file di release utilizzando l'opzione evidenziata.



Differenze tra le versioni 3.0 e 2.0 di Pandor+

Rispetto alla versione precedente di Pandor+ (2.0), la nuova versione (3.0) introduce caratteristiche che semplificano lo sviluppo delle avventure, tra le quali:

- non è necessario conoscere HTML
- le conoscenze necessarie di javascript sono state ridotte il più possibile
- è stata resa "trasparente" la gestione dello scipt di base del motore del gioco
- l'interfaccia di Pandor+ è stata semplificata con i soli comandi essenziali
- i testi nel codice ora sono introdotti con semplici virgolette " ed è più facile introdurre testi su più linee
- gli oggetti fissi di direzione possono essere introdotti direttamente nel testo delle stanze
- i menu degli oggetti sono ora introdotti direttamente nelle proprietà degli oggetti ed in modo più semplice
- le variabili hanno nomi liberi e sono definite in un file a parte
- la creazioni di condizioni è facilitata da diverse funzioni già pronte
- stanze con descrizioni diverse sono gestite direttamente con variabili
- il file di debug presenta informazioni sullo stato del giocatore, delle variabili e degli oggetti
- il file di debug presenta commenti utili a individuare la posizione di errori nel codice javascript
- è introdotto *counters.js* per isolare le condizioni che devono incrementare variabili solo in caso di mosse (azioni non vuote)
- la generazione del file finale è stata velocizzata

Sono inoltre introdotte le mappe automatiche, che tratteremo come ultimo argomento.

Attenzione infine che progetti sviluppati con la versione precedente non sono compatibili con la versione nuova.

Aggiungere la percentuale di completamento

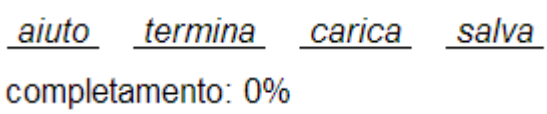
Aggiungere la percentuale di completamento

Una delle variabili che devono sempre essere definite in *variables.txt* è *_vCOMPLETION_*

Nel caso in cui volessimo mostrare durante il gioco la percentuale di completamento della nostra avventura, è necessario non cancellare il messaggio *<36>* in *messages.txt* :

```
<36>

completamento:
```

In questo modo, il valore di *_vCOMPLETION_* sarà sempre visibile dopo i comandi di default.



A questo punto dovremo decidere noi dove incrementare la percentuale di completamento. Tipicamente lo faremo a seguito di azioni del giocatore o quando scoprirà qualcosa di importante. Quindi, nelle nostre condizioni in *actions.js* e *conditions.js* basterà aggiungere nel codice:

```
_vCOMPLETION_ = _vCOMPLETION_ + XX;
```

dove *XX* è la percentuale da incrementare. E' chiaro che dovremo fare in modo che la somma totale non sia superiore a 99% (Pandor+ comunque al caso mostrerà al massimo 99). Inoltre, dovremo stare attenti a incrementare un'unica volta il punteggio se l'azione può essere ripetuta più volte.

Bisogna fare anche attenzione che non è possibile incrementare *_vCOMPLETION_* nel caso di azioni gestite automaticamente con le proprietà degli oggetti (*examinable, takeable, talkeable*, etc). Ad esempio, se abbiamo un oggetto *examinable* (e relativa proprietà *desc_examine*) non possiamo incrementare *_vCOMPLETION_* . Per farlo, dobbiamo rinunciare alla proprietà *examinable*, aggiungere nella proprietà menu l'azione "esamina", e gestire la stessa in *actions.js*:

```
if (f_action("_oggetto_","esamina"))
{
  If (_vEsaminatoOggetto_ == 0)
  {
     _vCOMPLETION_ = _vCOMPLETION_ + XX;
    _vEsaminatoOggetto_ = 1;
  }
  f_show("Hai trovato quello che cercavi! Bel colpo!");
  return true;
}
```

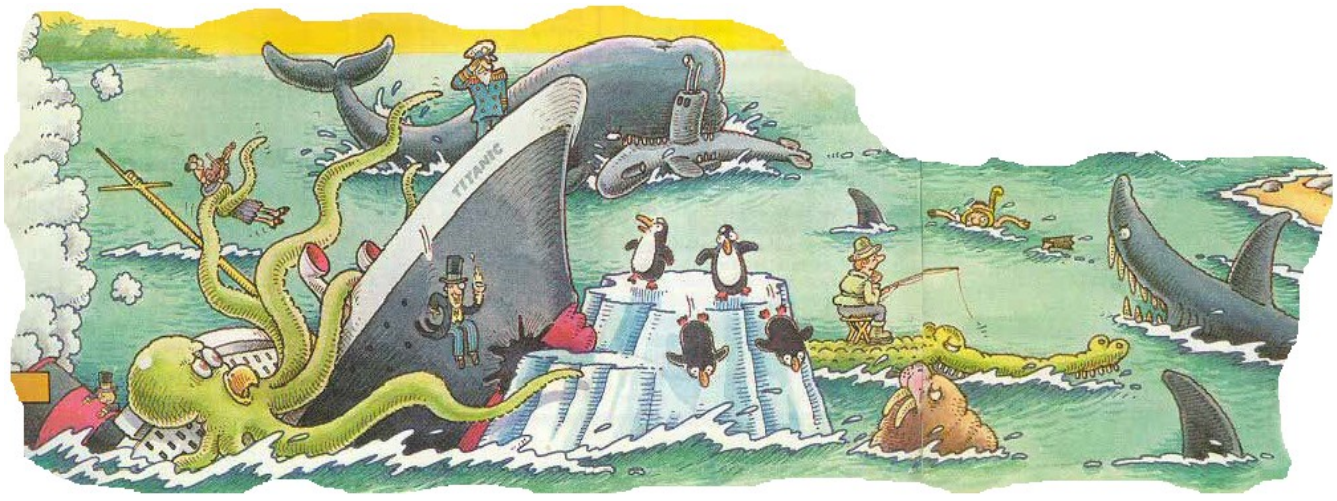
In questo codice è anche mostrato come incrementare un'unica volta *_vCOMPLETION_*: basta definire una variabile inizialmente uguale a 0 e che diviene 1 una volta fatto l'incremento.

Nel caso in cui volessimo mostrare il completamento raggiunto nella schermata di fallimento (*fail.txt*), basta aggiungerlo in questo modo nel testo della stessa:

```
Non sei riuscito a trovare il tesoro!

Ritenta, sarai più fortunato!

Completamento: {_vCOMPLETION_} %.
```

Ovvero tramite le parentesi *{ }*. Con questo sistema è possibile mostrare qualsiasi variabile, e questo può essere utile per creare dei punteggi alternativi da mostrare anche nella schermata di vittoria (*success.txt*):

```
Hai trovato il tesoro!

Ora sei finalmente ricco e ti godrai la bella vita... FANTASTICO!!!

Hai inoltre raccolto {_vNumeroGemme_} gemme!
```

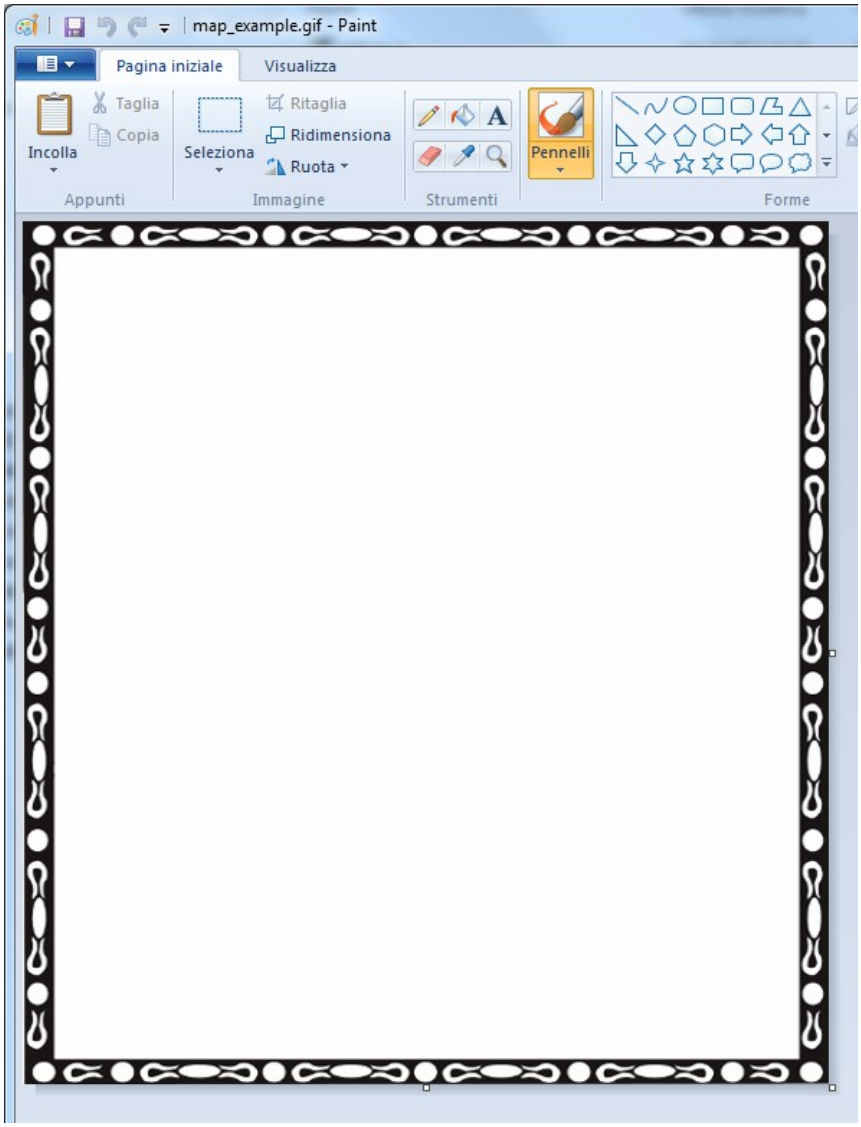


Aggiungere mappe

Pandor+ v3.0 permette di aggiungere mappe che si “svelano” via via che il giocatore visita le diverse stanze dell’avventura. L'opzione *mappa* sarà utilizzabile solo in browser con HTML5.

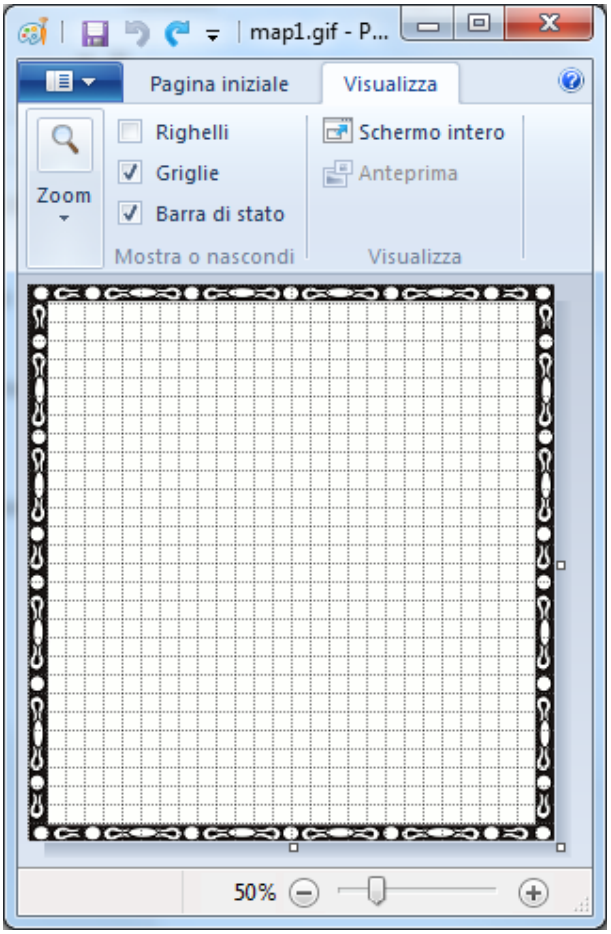
E’ possibile creare fino a 3 mappe e selezionare quale deve vedere il giocatore in base allo stato del gioco. Le mappe devono essere create alla fine di tutto lo sviluppo dell’avventura.

La prima cosa da fare è disegnare le mappe. Poiché devono avere dimensioni 560x600 ed essere in formato GIF, Pandor+ ci viene in aiuto fornendoci, in ogni cartella di progetto, un’immagine di base da modificare, *map_example.gif* . Apriamola col programma Paint di Windows:

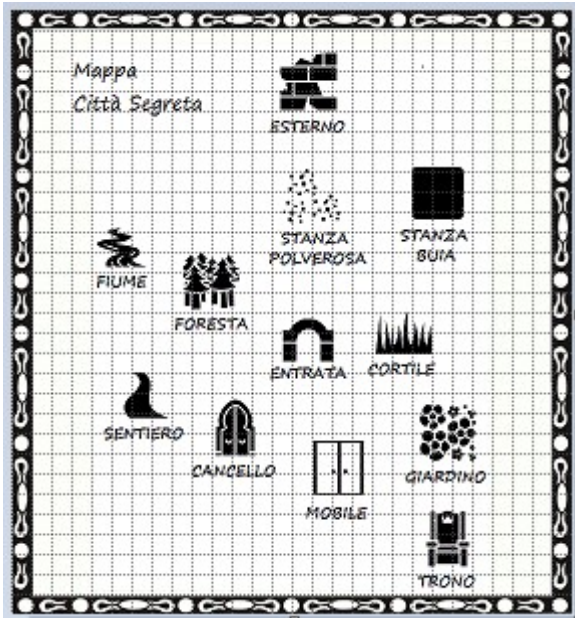


A questo punto possiamo salvare come map1.gif e iniziare a modificarla. Teoricamente è possibile usare colori e trasparenza ma conviene usare solo toni di grigio o bianco/nero (così da rendere migliore la visualizzazione sui reader) e l’idea di base è quella di disegnare le stanze cercando di collocarle utilizzando una griglia di 20 x 20 pixels.

In Paint si può ottenere questo attivando la griglia nel menu Visualizza e zoomando al 50%:

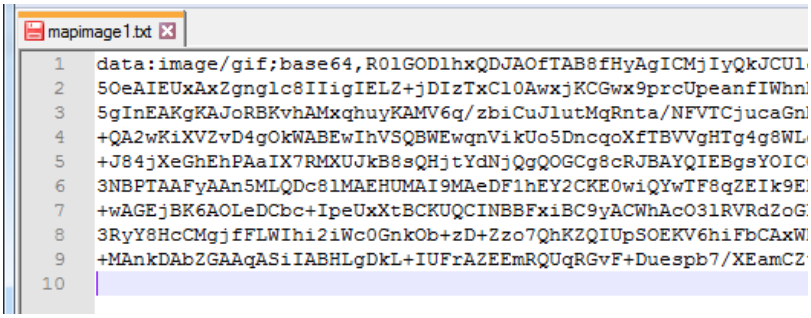


Ecco ad esempio la mappa dell'avventura "L'occhio purpureo":



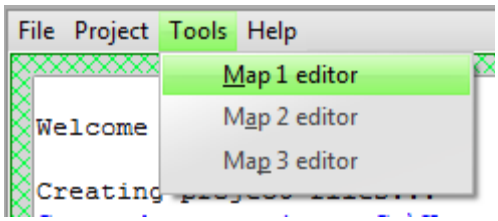
L'idea è quella che le varie stanze non devono "condividere" nessuno dei quadratini 20x20 della griglia.

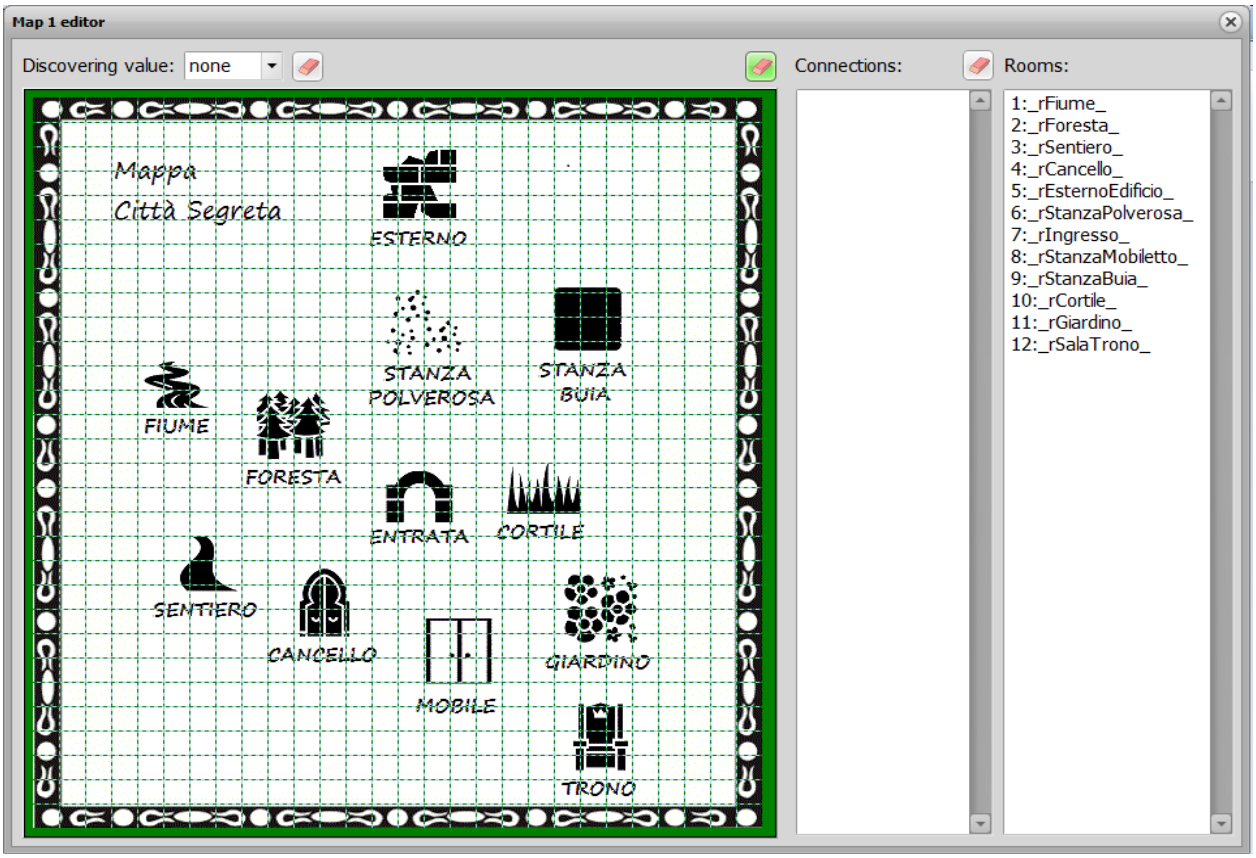
Una volta salvata, l'immagine va trasformata in codifica base 64 (come già visto in precedenza) e il risultato inserito in *mapimage1.txt* :



La procedura va ripetuta in caso per la mappe 2 e 3 se presenti, andando a modificare *mapimage2.txt* e *mapimage3.txt*

A questo punto apriamo Pandor+, selezioniamo il nostro progetto e carichiamo la mappa 1, tramite il menu *Tools*:





La finestra che si apre presenta l'immagine e una griglia di 20x20 pixels.

Sulla destra abbiamo l'elenco di tutte le stanze dell'avventura con il relativo indice numerico.

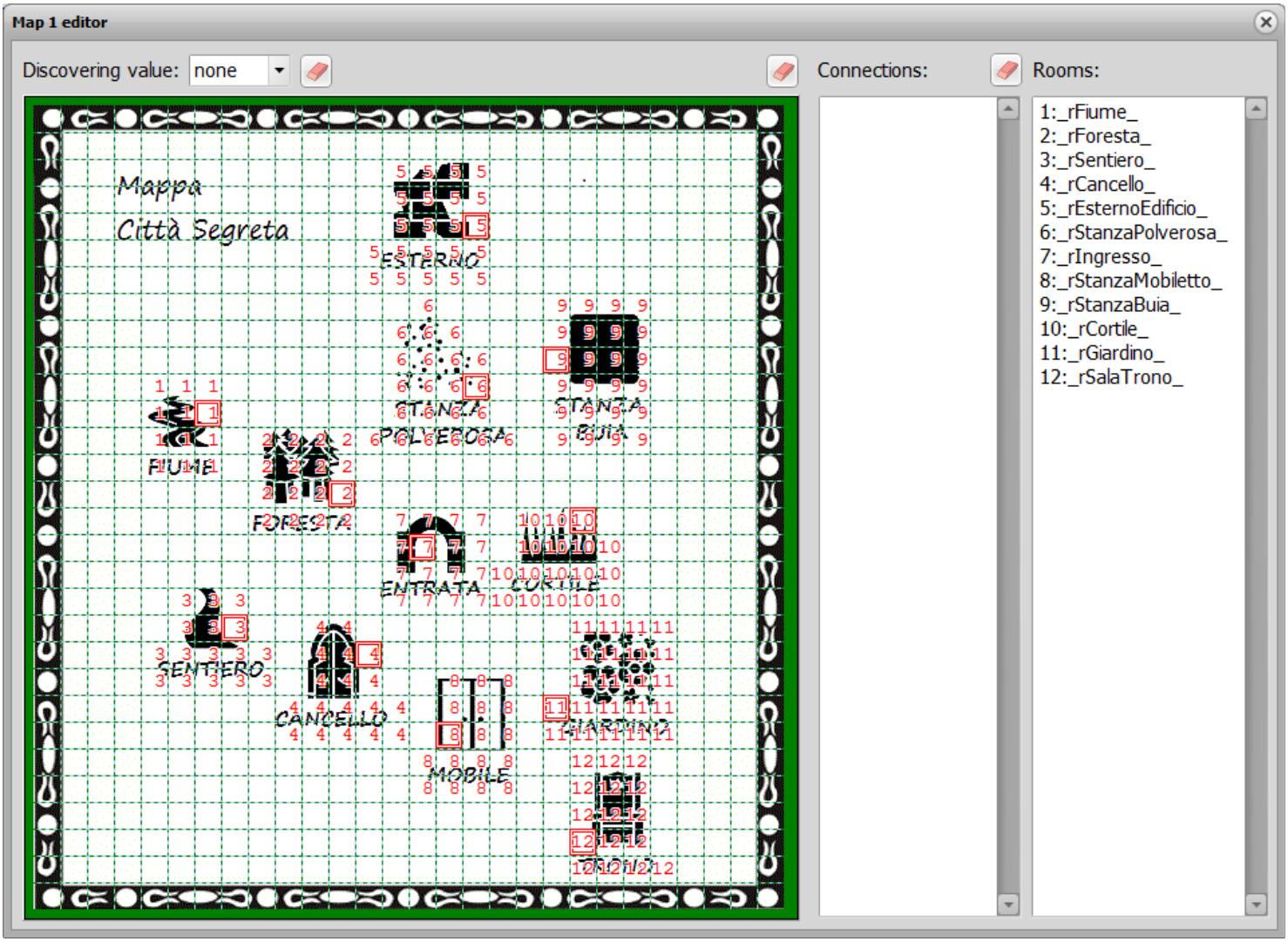
Il primo passo è coprire ogni sezione di immagine relativa ad una stanza con il numero della stanza stessa. In questo modo indicheremo a Pandor+ che nel momento in cui il giocatore si troverò in quella stanza, quella sezione di immagine dovrà essere mostrata.

Selezioniamo dal menu a tendina *Discovering value* il numero 01 e iniziamo a cliccare sui quadratini che "coprono" l'immagine relativa alla stanza "fiume":

In caso di errore, possiamo deselzionare cliccando col tasto destro del mouse o scegliendo dal menu *none* e poi cliccando dove vogliamo togliere i numeri.

Oltre a coprire la stanza, dobbiamo anche indicare dove si troverà il "mirino" indicante la posizione del giocatore se ci troviamo in quella stanza. Per farlo, basta cliccare ancora una volta su uno degli "1":

L'operazione va ripetuta per tutte le stanze (e ogni gruppo di numeri deve avere il numero per il "mirino"). Ecco il risultato finale:

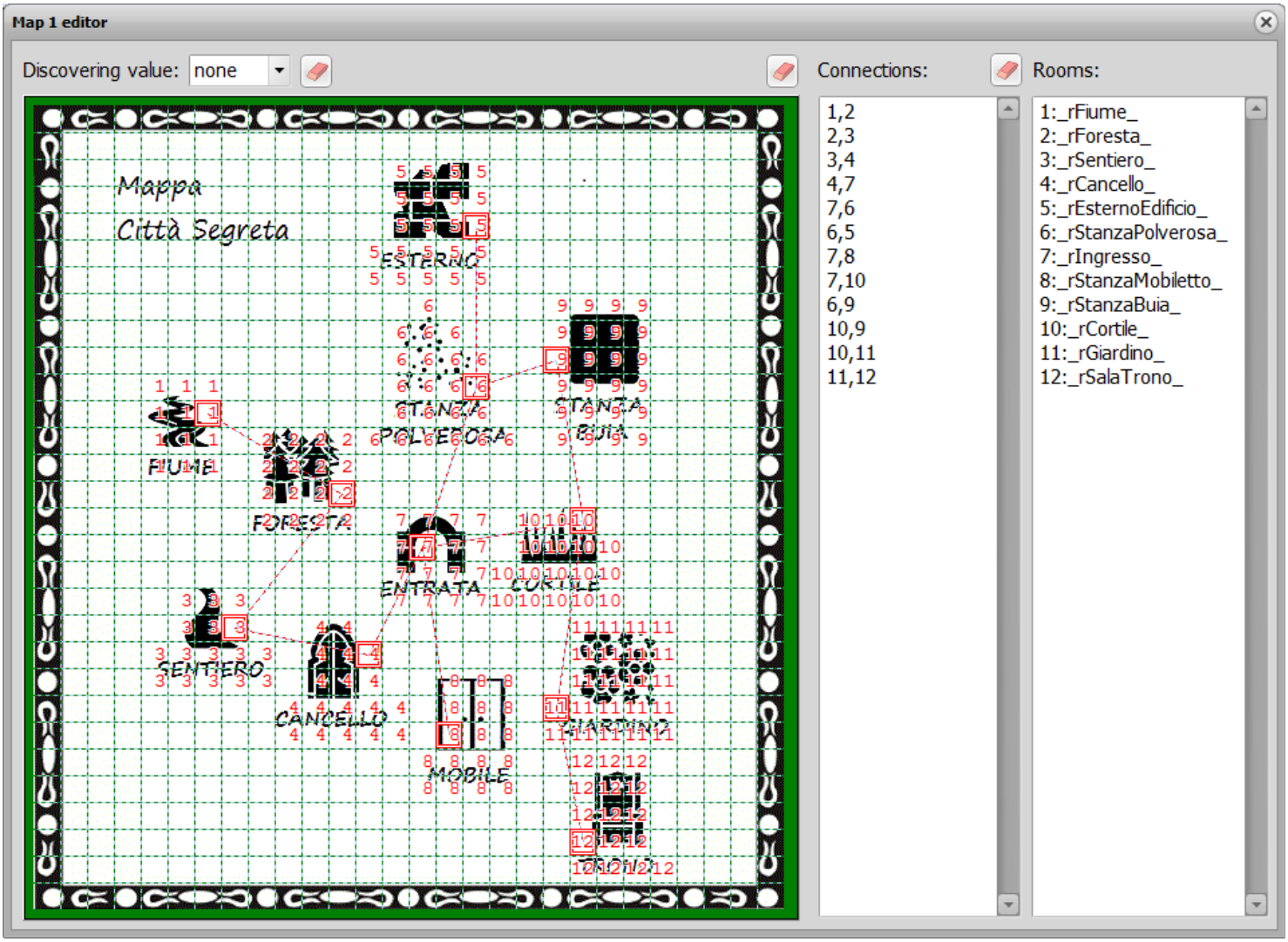


Le eventuali parti non coperte da numeri, come il bordo e la scritta “Mappa Città Segreta”, saranno sempre visualizzate. Invece, come già detto, ogni parte coperta da numeri si “scoprirà” solo quando il giocatore sarà passato per la stanza definita con quel numero.

L'ultima cosa da fare è scrivere l'elenco di connessioni tra le stanze. Ovvero, se il giocatore può passare da una stanza ad un'altra, bisogna scrivere la coppia numerica delle due stanze (separata da virgola) nella sezione *Connections*.

Il risultato è che Pandor+ creerà automaticamente delle linee tra i numeri “mirino” delle varie stanze connesse.

Ecco l'elenco di coppie di stanze completo per la mappa dell'avventura “L'Occhio purpureo”:



A questo punto possiamo chiudere la finestra e salvare le modifiche effettuate.

Una volta generato il gioco, il risultato sarà la presenza del comando “mappa”:

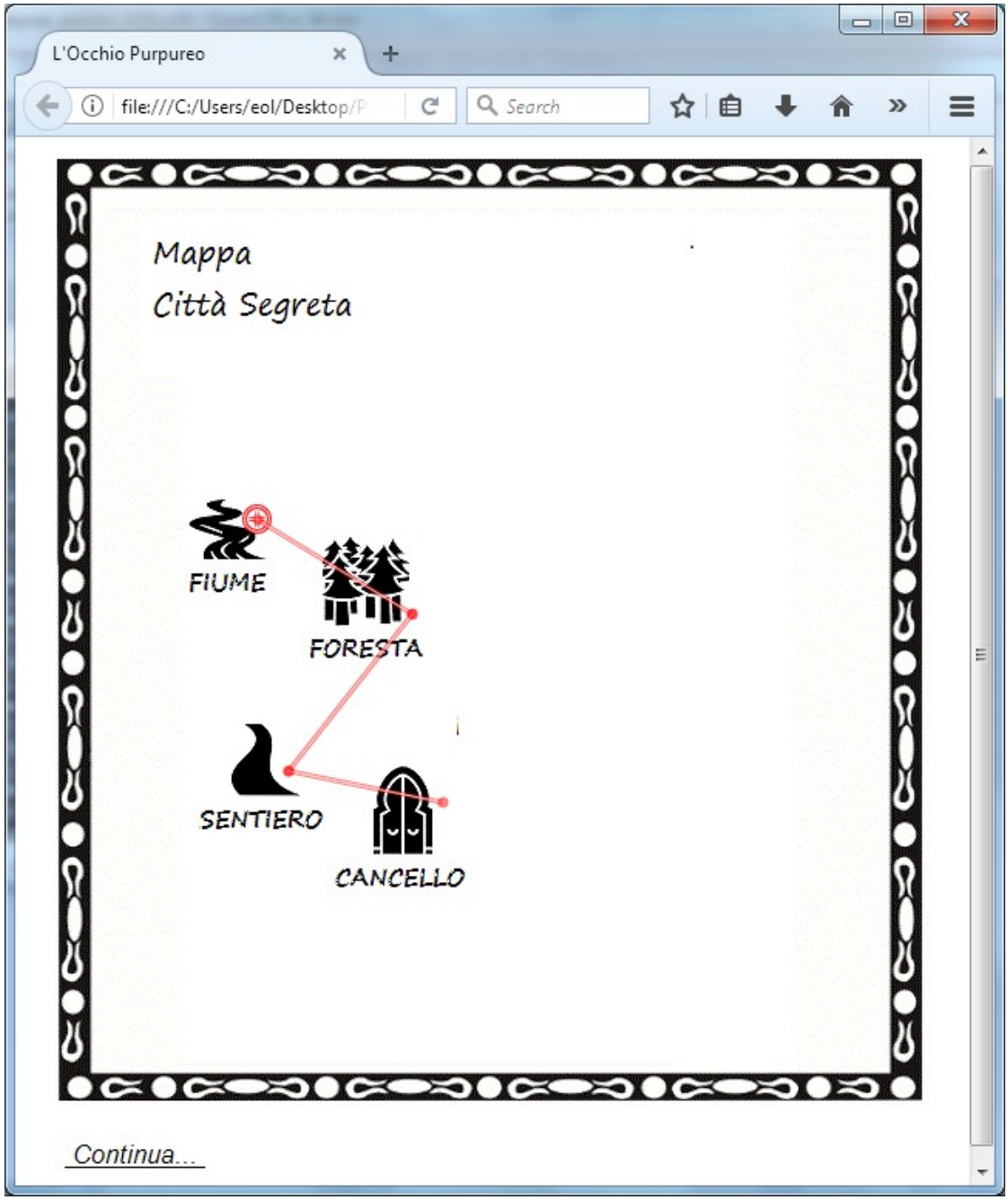
Sei accanto ad un *fiume impetuoso*. Ad *Est* c'è una foresta.

Non hai nulla con te.

---

aiuto *termina* *carica* *salva* *mappa*

e premendo mappa saranno visibili solo le stanze visitate e le eventuali connessioni:



Il “mirino” è posizionato sulla stanza fiume, essendo il giocatore al momento lì.

Vi può essere l'esigenza di non voler far apparire una stanza “automaticamente” al passaggio del giocatore in essa. In questo caso, nella definizione della stanza in *rooms.txt* basta aggiungere # al numero della stanza:

```
<#01>
_rFiume_
Sei accanto ad un {fiume impetuoso|_fiume01_}.
Ad {Est|_est01_|vai|_rForesta_} c'è una foresta.
```

Invece, per far apparire manualmente una sezione di mappa coperta da un certo numero *N*, basta usare nel codice (in *conditions.js* o *actions.js*) la funzione *f_discovermap(N).*

E' possibile in questo modo scoprire parti di mappa quando vogliamo, coprendole con numeri non utilizzati per stanze (ad esempio con numeri alti).

Rimane il caso di gestione di più mappe.

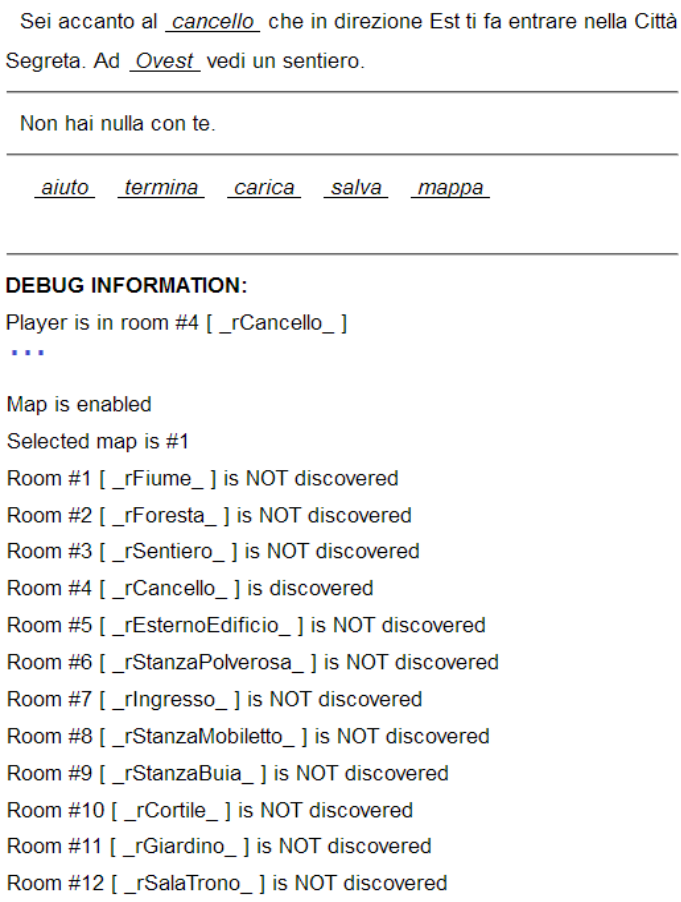
In questo caso, per passare da una mappa all'altra, sarà necessatio chiamare nel codice (in *conditions.js* o *actions.js*) la funzione *f_selectmap(M)* con *M* uguale al numero di mappa (1,2 o 3) che vogliamo mostrare quando il giocatore usa l'opzione *mappa*.

Ad esempio:

```
// in conditions.js


// se il giocatore si trova in una delle stanze del castello
if (     f_isplayerin("_rIngressoCastello_")
      || f_isplayerin("_rSalaCastello_")
      || f_isplayerin("_rCucinaCastello_")
   )
{
  // seleziona la mappa 2
  f_selectmap(2);
}
else
{
  // altrimenti seleziona la mappa 1
  f_selectmap(1);
}
```

In caso di più mappe, la mappa selezionata inizialmente sarà sempre la 1.

Vi sono alcune informazioni relative alle mappe che possiamo vedere durante il debug col browser. Questo sono presenti nella parte più bassa della pagina di gioco:



Anzitutto è evidenziata la mappa selezionata (1,2 o 3). Quindi, per ogni stanza, è indicato se la stessa è "scoperta", ovvero se le zone della mappa coperte dal numero della stanza sono visibili (*is discovered*) o no (*is NOT discovered*).

Ecco infine un elenco di possibili errori in Pandor+ relativi alla gestione delle mappe:

File connections.txt not found! / File mapX.gif not found! / File mapX.txt not found!

Il file non è stato trovato

Invalid connections.txt file! / Invalid mapX.txt file! / Invalid mapX.gif image file!

Il file non è valido

Invalid connections!

Si stanno specificando delle connessioni non valide (numeri di stanza non validi)

Invalid mapX.gif image size (not 560x600)

L'immagine *mapX.gif* non è 560x600 pixels

Missing connection point for area: X

Bisogna specificare il numero "mirino" per il gruppo di numeri X

Unable to save mapX.txt file!