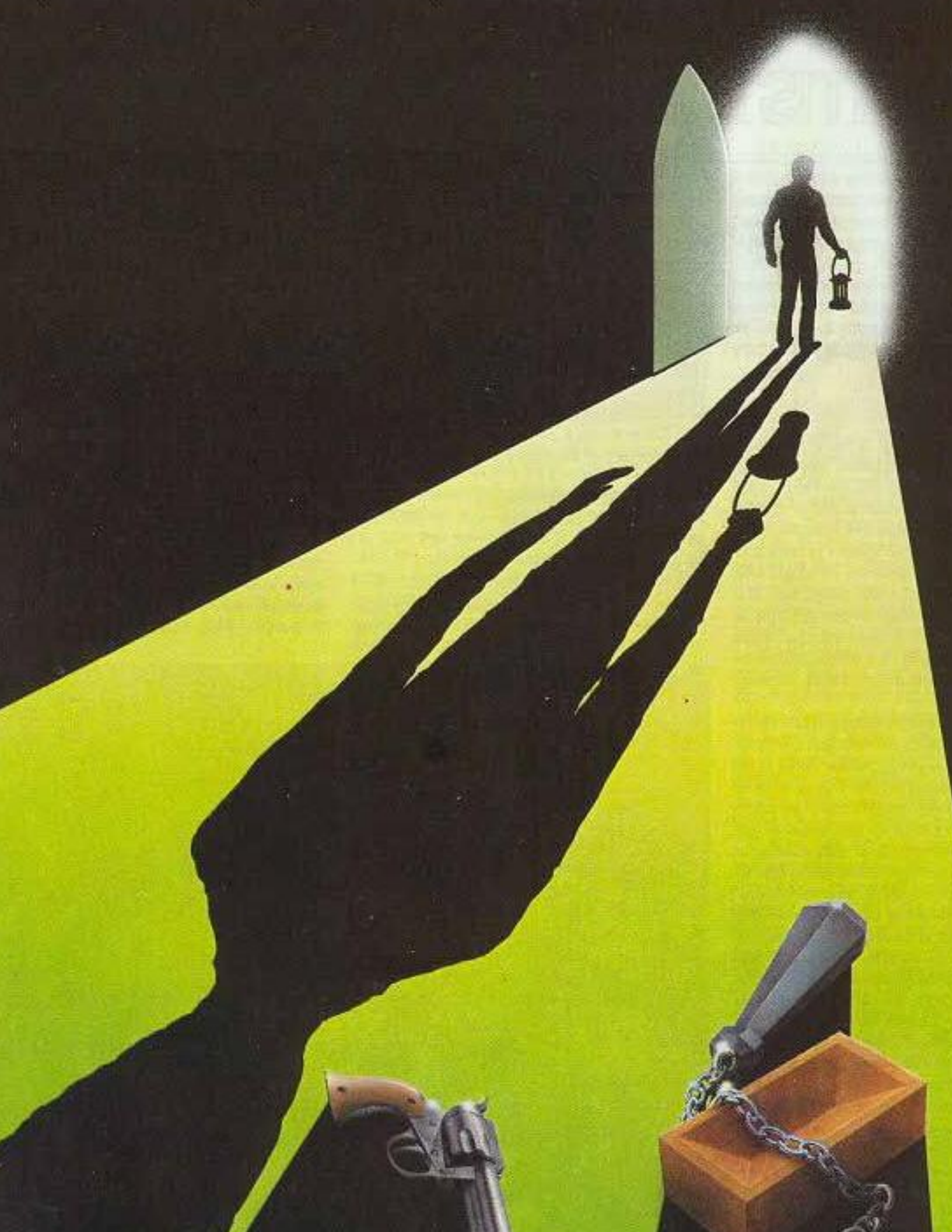


Benvenuti!

Pandor+ è un sistema di sviluppo freeware per creare adventure in lingua italiana o inglese da giocare su lettori a inchiostro elettronico (in particolare il reader Kindle) 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 de "Il covo dei trafficanti" o "Il Mistero di Villa Revoltella". Si consiglia di visitare il sito www.epaperadventures.qlmagic.com per tutte le informazioni e provare a giocare a una delle avventure disponibili! Sul nostro sito troverete anche quanto necessario a scaricare l'ultima versione di Pandor+, del progetto descritto nel tutorial e del tutorial stesso.

Prima che continuate a leggere ecco alcune importanti considerazioni:

- 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 ; questo ci permetterà anche di inviarvi eventuali revisioni successive del tutorial
- Gli autori non sono esperti ne di avventure testuali ne di programmazione. Il progetto nasce dalla nostra passione per i giochi in genere e per l'informatica ed è stato sviluppato un po' alla volta nel tempo libero!
- 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 a epaperadventures@gmail.com (ricordiamo comunque che questa iniziativa è senza scopo di lucro).

Il risultato finale dello sviluppo di un adventure in Pandor+ è **un unico file HTML con codice javascript**. In questo tutorial però non si spiegherà come programmare in HTML o javascript ma solamente quanto necessario a sviluppare un'avventura con Pandor+.

Si presuppone una conoscenza di base di linguaggi di programmazione e in particolare di:

- Tag HTML
- Dichiarazione e assegnazione di variabili e array in javascript
- Chiamata di funzione in javascript
- Operazioni con variabili e array in javascript
- Istruzioni *if, for, return* in javascript

Tuttavia anche 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++ (oppure Notepad) e un browser quale Explorer o FireFox. Attenzione: **Chrome non può essere utilizzato**. Il motivo è che i file HTML prodotti da Pandor+ contengono script che utilizzano cookie e Chrome non supporta i cookie nel caso di file HTML risiedenti localmente sul PC (che utilizzeremo nella fase di debug).

Attenzione anche all'utilizzo con Internet Explorer versione 8 o precedenti: per questi browser i testi e le immagini del gioco possono apparire non centrati quando questo è atteso e a volte vengono persi degli "a capo" (ad esempio tra due paragrafi).

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).

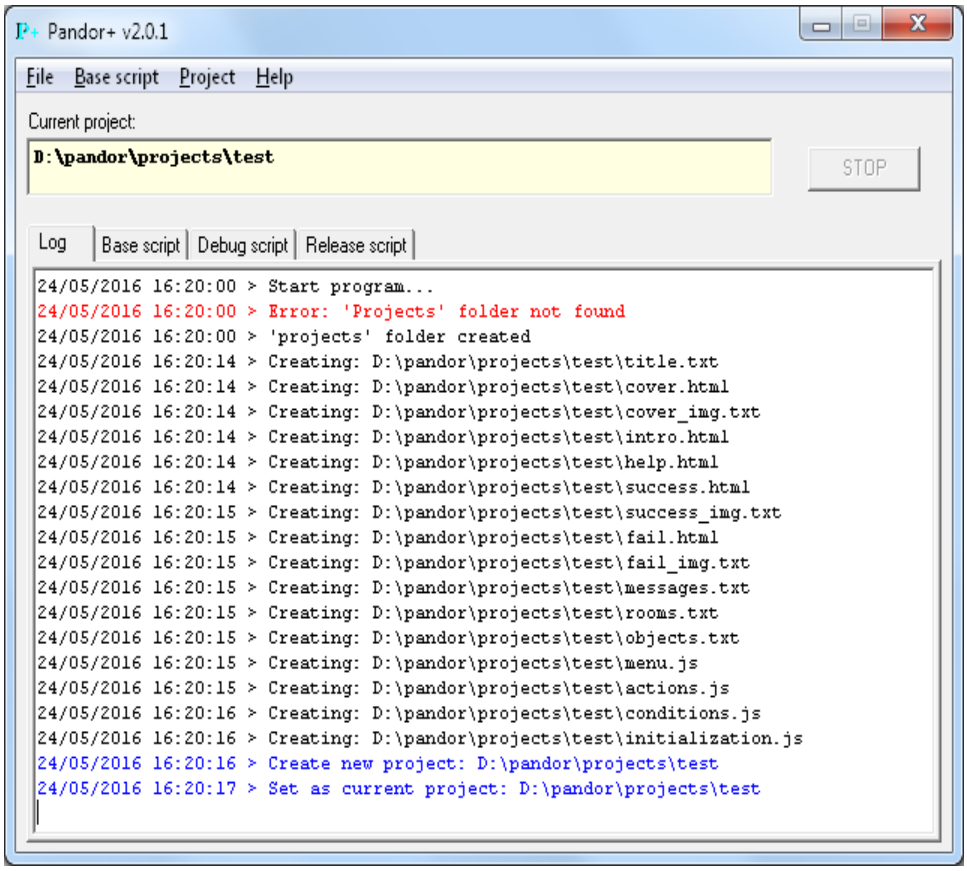
Revisione A.5 – luglio 2016 – Pandor+ v2.0.1 – Script di base v2.0.1

Sommario

Creiamo un nuovo progetto

Per cominciare, copiamo l'applicazione "pandor+.exe" in una nuova cartella (ad esempio: "pandor") e eseguiamola. Pandor+ creerà come prima cosa una cartella "projects".

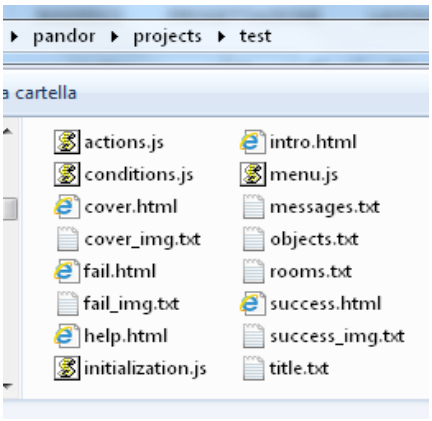
Andiamo nel menu *File* e creiamo un nuovo progetto scegliendo *Create new project*. Digitiamo come nome "test". Pandor+ creerà una sotto-cartella "test" con dentro diversi file.



Ogni progetto viene gestito in una sottocartella della cartella "projects" (attenzione a scegliere quindi nomi di progetto che siano validi anche come nomi di cartella, evitando caratteri quali * e ? ad esempio).

Sotto *File*, possiamo scegliere *Open project folder* che ci aprirà la cartella del progetto e potremo ispezionare i file aprendoli in Notepad++ o notepad.

Qualora le estensioni dei file non fossero visibili (".js", ".html", 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.

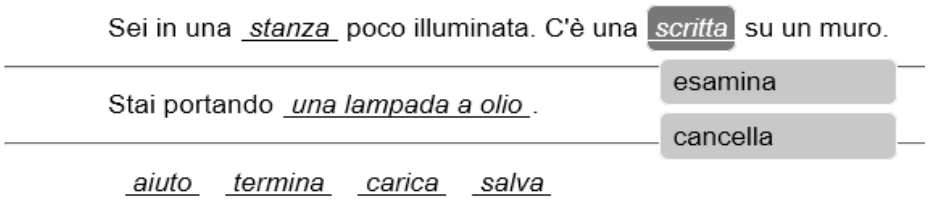


Ogni nuovo progetto creato contiene un'avventura di default con un paio di stanze e un paio di oggetti. Andiamo sul menu *Project* e scegliamo *Generate adventure (only debug)*. Il risultato sarà la generazione di un file "debug.html" (sempre nella cartella test) contenente l'adventure!

```
24/05/2016 16:26:33 > Generated: D:\pandor\projects\test\debug.html
24/05/2016 16:26:33 > Adventure generated with warnings!
```

Per ora ignoriamo i warning (in viola) che ci sono stati dati da Pandor+ in fase di generazione e apriamo "debug.html" con il browser (ad esempio con tasto destro sul file e apri con...). Il browser potrebbe richiedere l'autorizzazione a eseguire script. Autorizziamo (seguendo le istruzioni nel browser) e vedremo quindi apparire la prima schermata dell'adventure di test.

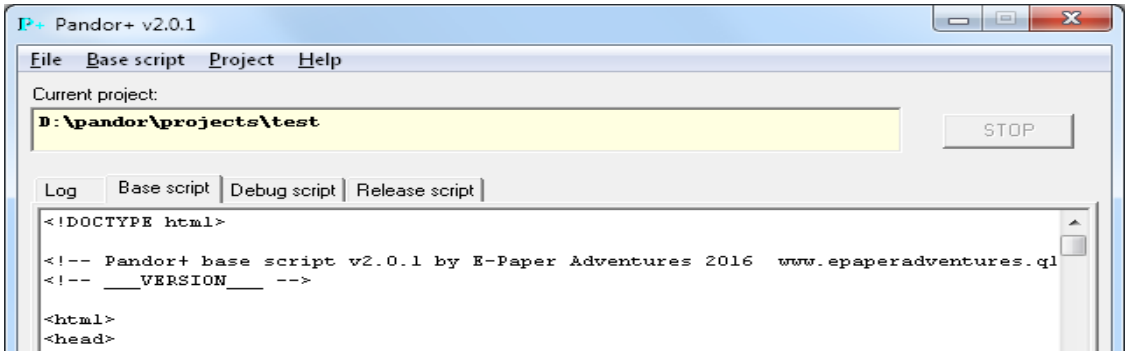
Proviamo a cliccare su "Inizia...": vedremo l'introduzione (vuota – clicchiamo su *Continua*) e quindi l'unica stanza del gioco.



Il gioco si conclude appena usciamo dalla stanza:

L'avventura di default è ben poco entuasiasmante, mancano alcune immagini (segnalate sia da warning in generazione, sia nel gioco con il messaggio *IMAGE FILE IS EMPTY*) e 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 adventure*.

Il punto di partenza della generazione è uno script di base (un mix di HTML e javascript) che si può vedere spostandosi sul tab *Base script*:



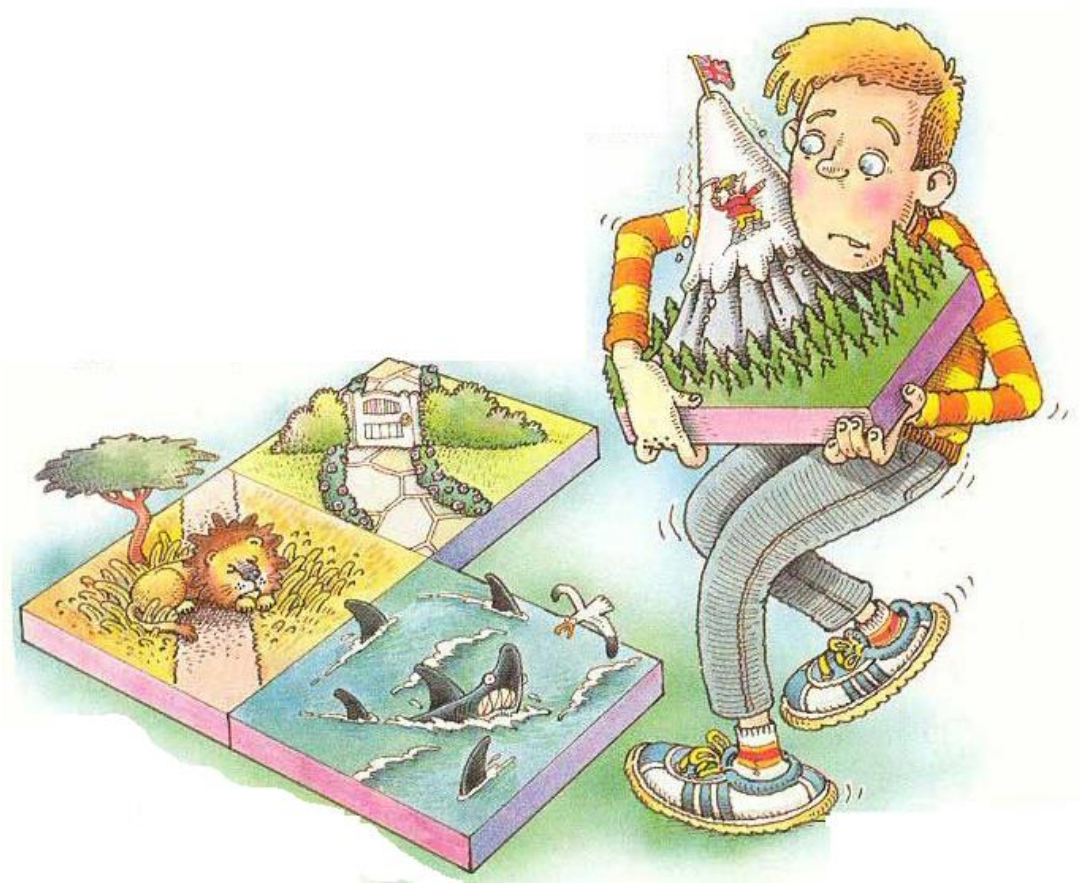
Questo script contiene codice HTML e javascript e delle sezioni incomplete, caratterizzate da "parole" quali ___TITLE___ , ___ACTIONS___ , ___MENUS___ , etc… Al comando di generazione, i file di progetto

vengono elaborati da Pandor+ e inseriti nelle sezioni incomplete per comporre il file finale HTML con l'avventura completa.

Questo processo per noi è trasparente e dovremo solo preoccuparci di scrivere in maniera corretta all'interno dei file di progetto. Ognuno di essi ha un suo preciso scopo e alcune regole da seguire nella sua scrittura.

Possiamo decidere di generare la versione di debug dell'avventura (file "debug.html") oppure quella di debug e anche quella di release (file "release.html"). In quest'ultima risultano modificati i nomi delle funzioni e sono crittati i testi, in modo che non basti una semplice lettura del file del gioco in notepad per poter leggere in chiaro descrizioni, oggetti, messaggi e così via. Il codice di entrambi i file sono visibili nei TAB *Debug script* e *Release script.*

Quando l'avventura è pronta per la pubblicazione, il file release.html va rinominato e poi messo a disposizione su un proprio server web o, ancora meglio, inviato a epaperadventures@gmail.com che lo metterà 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.

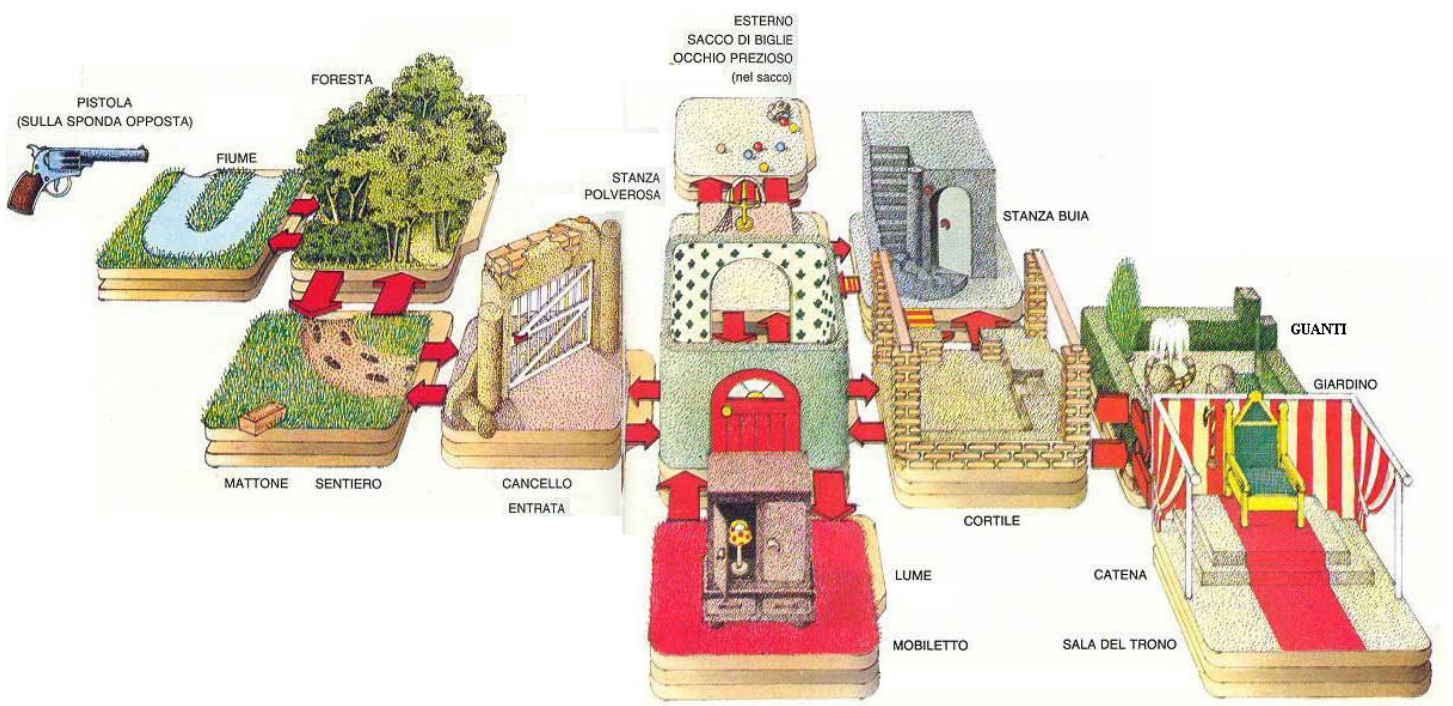
Per sua sfortuna, il protagonista è inseguito da un ispettore delle tasse che gli renderà la vita difficile. Quando l'ispettore compare infatti, possono succedere due cose. Se si ha con se un oggetto, egli se ne appropria, come anticipo del rimborso dell'enorme debito col fisco. Se invece non si ha ancora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

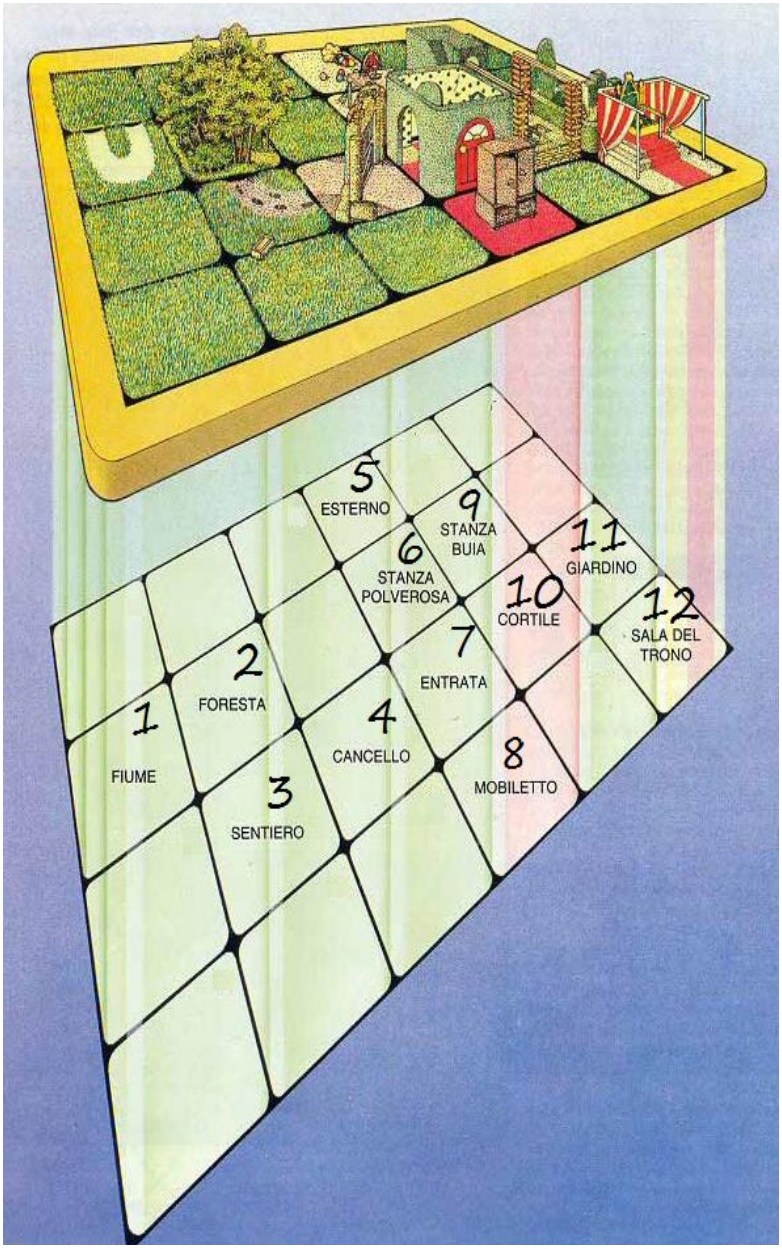
Non tutti gli oggetti che si troveranno nei vari ambienti sono però utili. Ad esempio un mattone molto pesante fa annegare il giocatore se tenterà di attraversare un fiume a nuoto.

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.



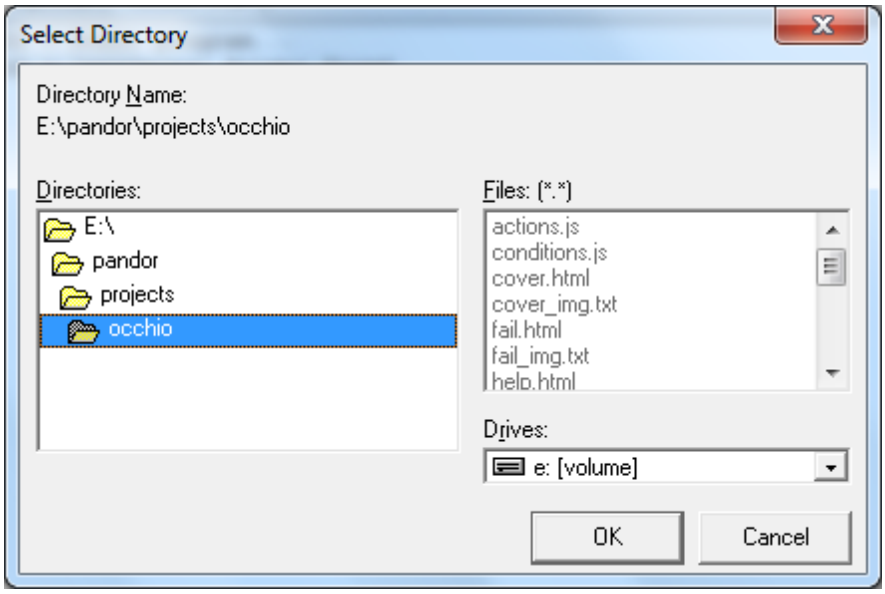
Associamo ad ogni ambiente un numero. Questo ci servirà per scrivere le descrizioni degli ambienti (detti anche "stanze") e i collegamenti tra essi.

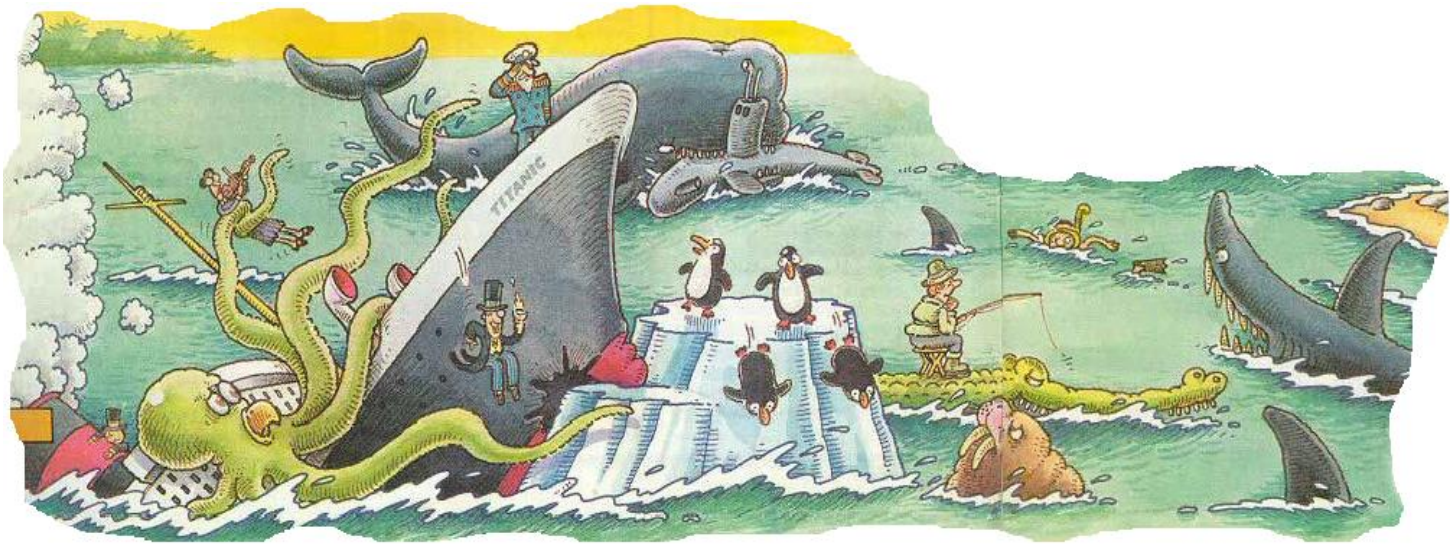


Gli altri dettagli dell'avventura li vedremo via via che analizzeremo e completeremo i file del progetto.

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".

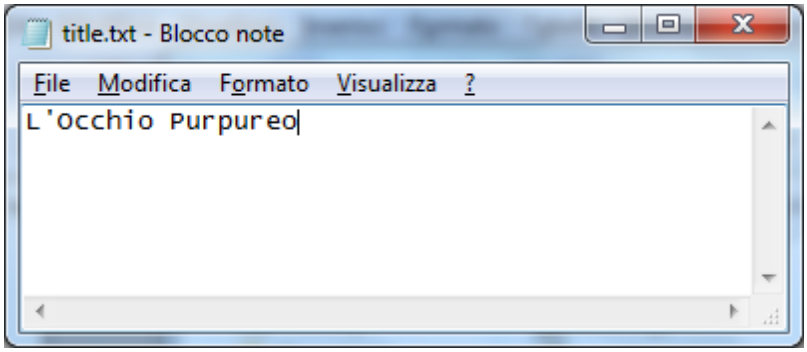




Il file di progetto title.html

La prima cosa da fare è cambiare il titolo della nostra avventura. Apriamo con Notepad++ o notepad il file "title.html" dentro alla sotto-cartella "occhio".

Il contenuto è un'unica linea, di default "Titolo avventura". Modifichiamo in "L'occhio purpureo" e salviamo il file e chiudiamo l'editor.



In generale, i caratteri ammessi in questo file sono **esclusivamente** lettere dell'alfabeto italiano (anche accentate) minuscole e maiuscole, numeri, spazio, apostrofo e questi caratteri:

```
! " £ % & ? , . + - : ; * / = [ ] ( )
```

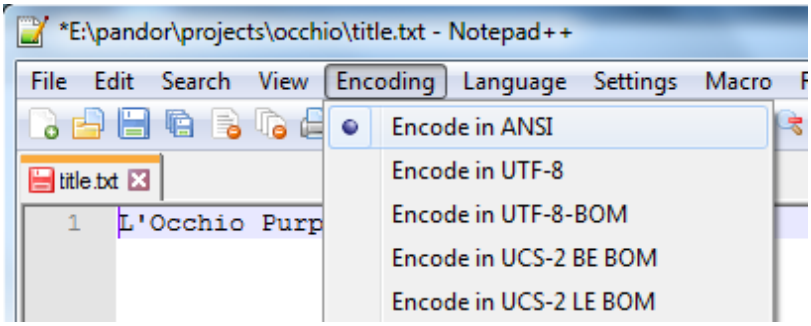
Il carattere & ha un utilizzo speciale: può essere usato per inserire caratteri speciali HTML (per maggiori dettagli visitate: http://www.caratterispecialihtml.com/). Ad esempio per inserire i caratteri < e > , basta aggiungere nel testo < e >

Le lettere accentate come à, è, é, … vengono tradotte automaticamente da Pandor+ nella codifica HTML corretta, quindi potremo scrivere i nostri testi senza bisogno di pensarci troppo.

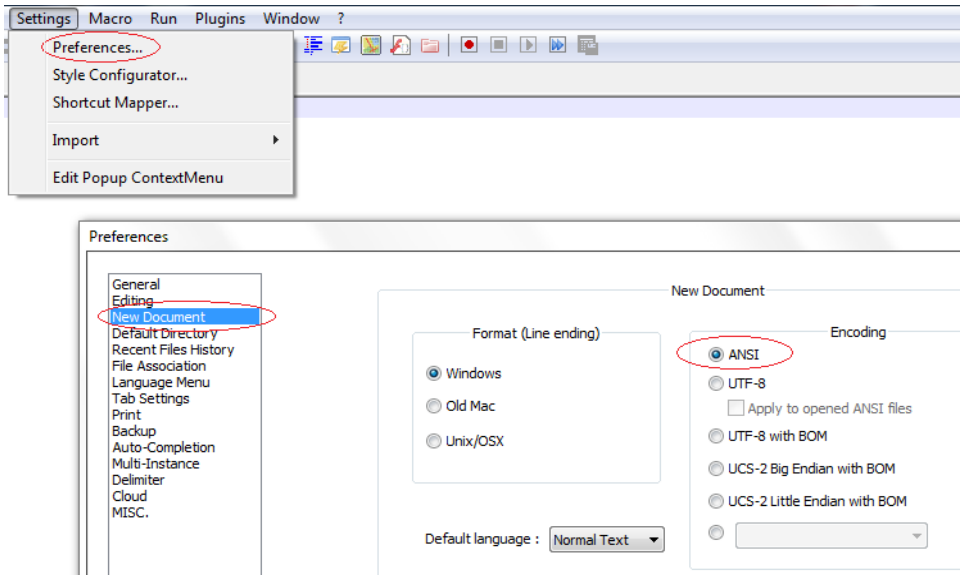
In "title.html" non utilizzare altri caratteri (come ad esempio _ $ \ @ { } ° ^) con l'idea di stamparli su schermo, perché ciò causerebbe errati comportamenti da parte di Pandor+. Alcuni di questi caratteri avranno degli usi speciali che vedremo più avanti. Quanto appena detto è valido per tutti i file di progetto con estensione ".html" !

Allo stesso modo, **attenzione a creare e salvare sempre tutti file di progetto (".html", ".js", ".txt", ...) in codifica ANSI e a non introdurre caratteri non ANSI perchè questo causerebbe errori nello script finale.**

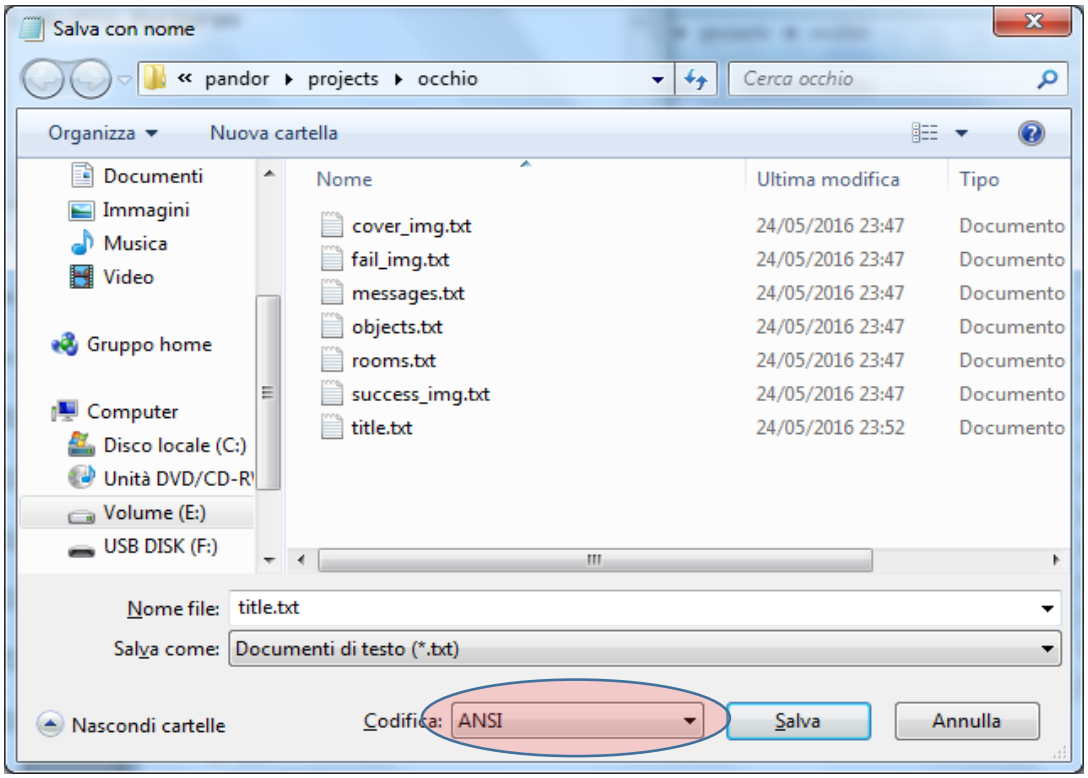
In Notepad++ bisogna selezionare il menu *Encoding* e poi *Encode in ANSI*:



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".

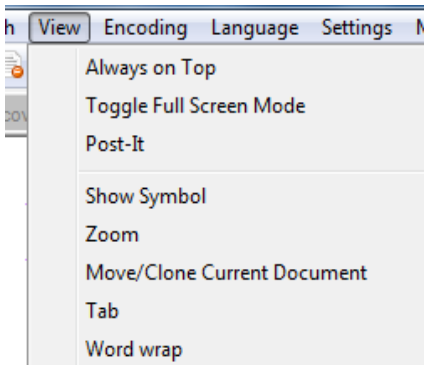


In notepad la codifica è invece visibile quando si "salva con nome":

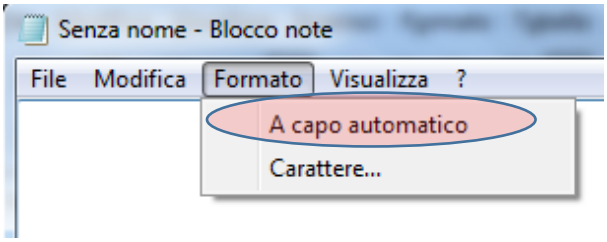


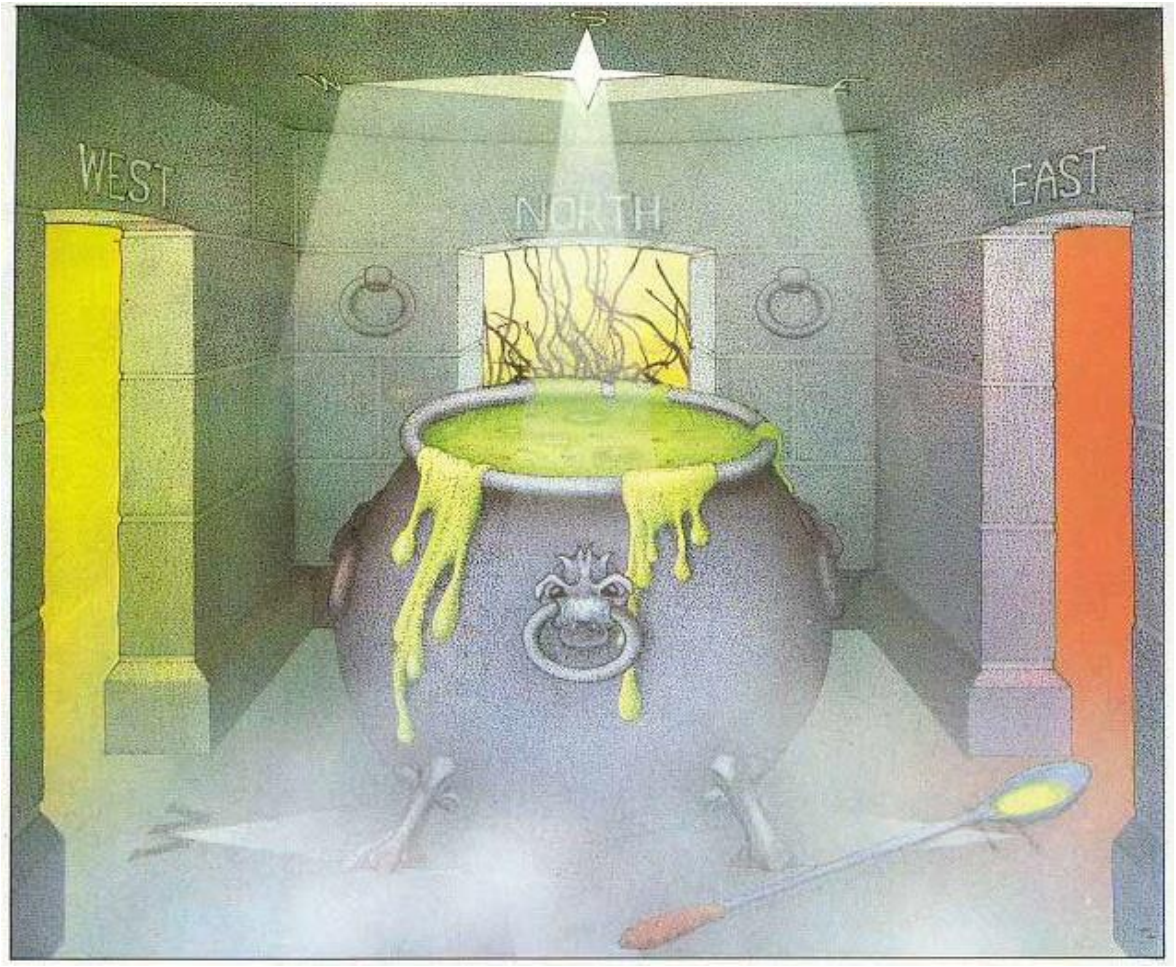
Fate molta attenzione che tutti i file di progetto siano sempre salvati in codifica ANSI, altrimenti Pandor+ potrebbe non essere in grado di utilizzarli correttamente. Attenzione anche a copia-incollare testo proveniente da applicazioni esterne in quanto, se presenti caratteri non ANSI, questo potrebbe 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.



In Notepad la troviamo sotto *Formato*:





Il file di progetto cover.html e cover_img.txt

Il file "cover.html" contiene la prima schermata del gioco. Ecco un esempio da "Il covo dei trafficanti":

Se apriamo il file "cover.html" del progetto di default con l'editor, vedremo il seguente codice HTML:

```
<h1>Titolo avventura</h1>
<pcenter>
___IMAGE___
</pcenter>
<pcenter><b>E-Paper Adventures 2016</b></pcenter>
<br>
```

Il tag HTML *<h1>* serve per il titolo della schermata. Quindi tra *<h1>* e la chiusura del tag *</h1>* scriveremo il titolo della nostra avventura, che dovrebbe corrispondere a quello in "title.html".

<pcenter> è un tag che serve per centrare testo o immagini. La frase o immagine da centrare va poi chiusa con *</pcenter>*.

** serve per scrivere frasi in grassetto, poi chiuse da **.

*
* (va usato da solo, senza tag di chiusura) indica di andare a capo. Questo permette di avere una riga di spazio prima delle opzioni che Pandor+ aggiunge automaticamente alla schermata iniziale (*Inizia l'avventura!* e *Vai al sito*…).

Altri tag da utilizzare in "cover.html" sono ... (per aprire e chiudere un nuovo paragrafo), *...* (per scrivere in corsivo), *<u>...</u>* (per scrivere sottolineato), *<hr>* (come *
*, ma aggiunge una linea nera).

Ad eccezione di "title.html", anche negli altri file di progetto con estensione ".html" potete utilizzare solamente i tag appena elencati.

`___IMAGE___` è invece rimpiazzato automaticamente da una immagine se essa è presente nel file "cover_img.txt", altrimenti viene sostituito dalla frase "IMAGE FILE IS EMPTY". Se non vogliamo mettere immagini nella schermata iniziale, possiamo rimuovere la parte di HTML:

```
<pecenter> ___IMAGE___ </pcenter>.
```

Modifichiamo quindi il file in questo modo:

```
<h1>L'Occhio Purpureo</h1>
<pcenter>
___IMAGE___
</pcenter>
<pcenter><b>E-Paper Adventures 2016</b></pcenter>
<br>
```

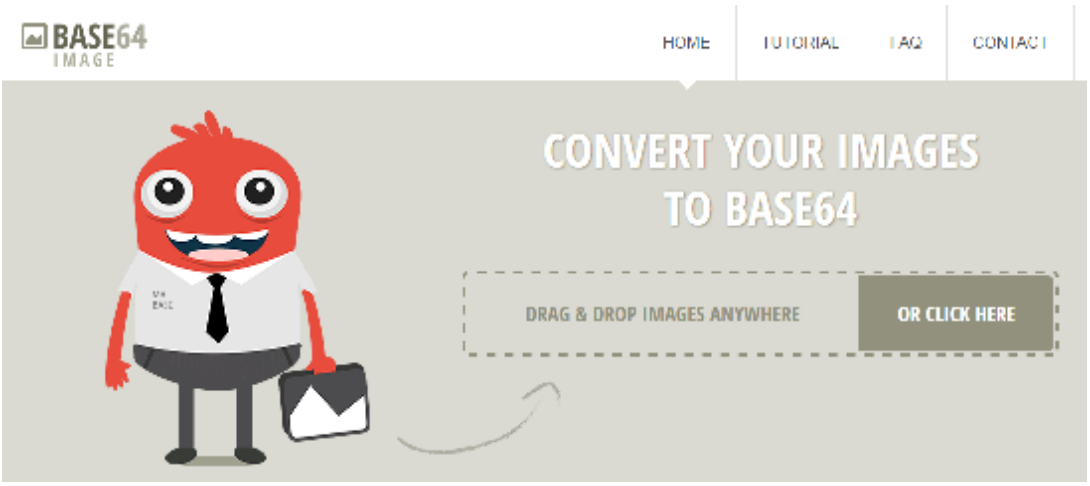
Vediamo ora come aggiungere un'immagine. Poichè Pandor+ genera un unico file HTML che contiene tutto il gioco, le immagini vanno codificate in un modo particolare, chiamata codifica base 64. Oltre alla codifica, dovremo scegliere immagini che abbiano dimensioni opportune anche per schermi di lettori come Kindle.

Come esempio, col browser andiamo all'indirizzo:

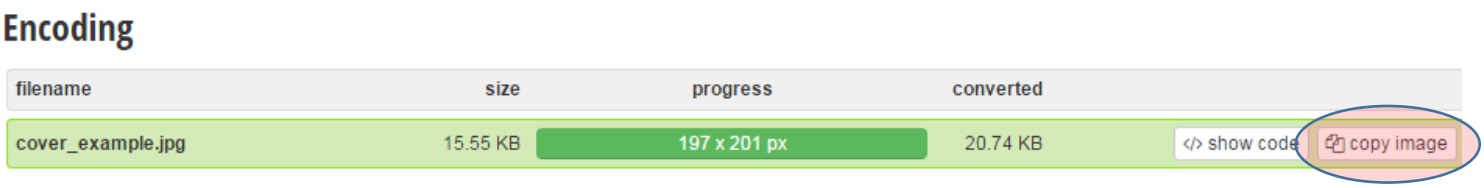
www.epaperadventures.qlmagic.com/cover_example.jpg

e salviamo l'immagine (ad esempio con tasto destro sull'immagine – salva immagine) sul nostro desktop.

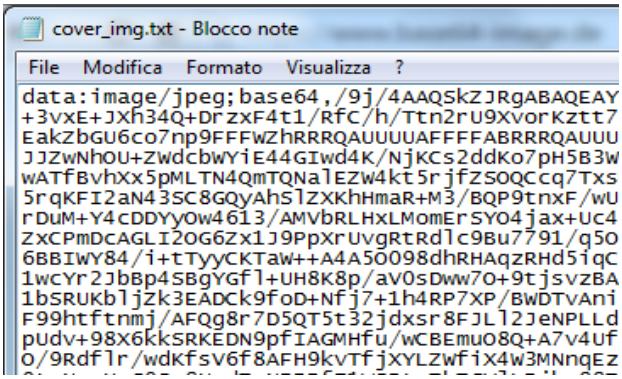
Dobbiamo ora creare la codifica in base 64. Un buon sito per farlo è: https://www.base64-image.de/



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



Apriamo "cover_img.txt" e incolliamo.



Ora salviamo e chiudiamo l'editor.

Andiamo in Pandor+ e torniamo al nostro progetto "occhio". Generiamo il file di debug. A questo punto apriamolo nel browser. Il risultato sarà questo:

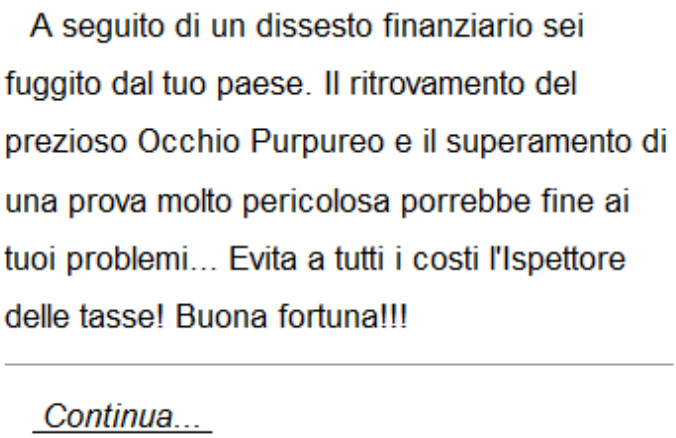
Vedremo più avanti che è anche possibile modificare il testo dei comandi *Inizia l'avventura!* e *Vai al sito...* e l'effettivo indirizzo web collegato.

Il file di progetto intro.html

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

```
<p>
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!
</p>
<hr>
```

Non ci si deve preoccupare degli "a capo" nel mezzo del paragrafo. Infatti in HTML, all'interno di un paragrafo (* ... *), tutti gli "a capo" diventano semplicemente degli spazi.



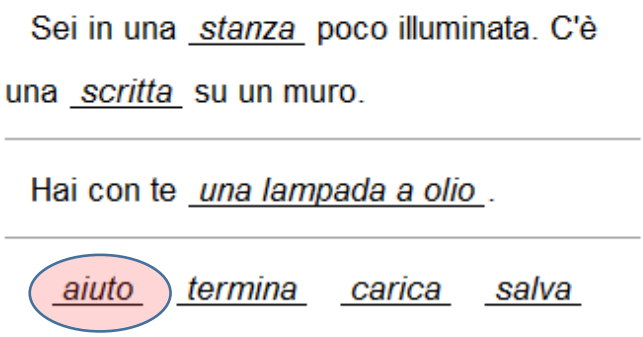
Pandor+ aggiunge in automatico il comando *Continua...* cliccando sul quale si va all'avventura vera e propria.

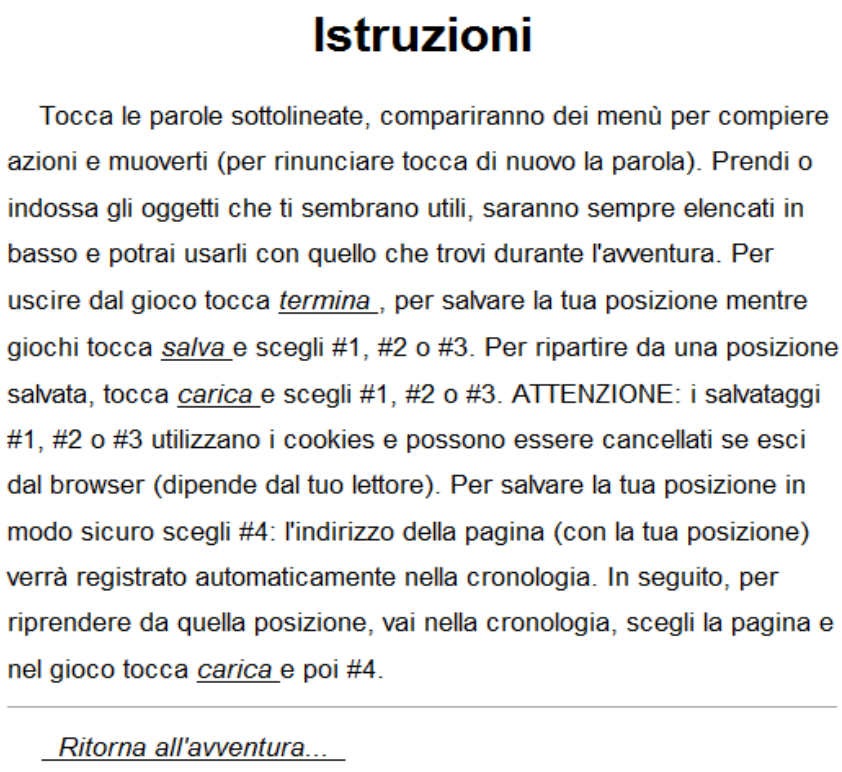
Non è prevista la possibilità di inserire immagini.

Il file di progetto help.html

Quando clicchiamo su *aiuto* nel gioco, la schermata che appare è quella contenuta in "help.html".

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





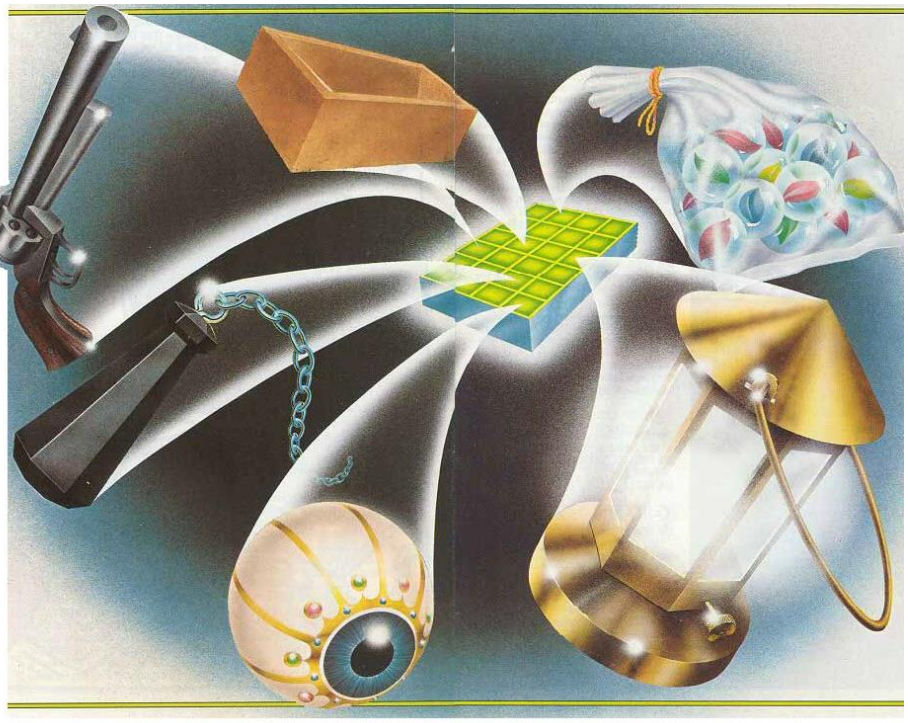
Apriamo comunque il file "help.html" con l'editor e notiamo che dove nella schermata appaiono i comandi come ad esempio *termina*, vengono utilizzate nel codice delle parentesi graffe:

```
in basso e potrai usarli con quello che trovi durante l'avventura.Per uscire dal gioco tocca  <u><i> {f_gmes(11)}
</i></u>, per salvare la tua posizione mentre...
```

basso e potrai usarli con quello che trovi durante l'avventura. Per

uscire dal gioco tocca *termina* , per salvare la tua posizione mentre

In pratica l'utilizzo delle parentesi graffe {f_gmes(11)} fa si che Pandor+ chiami la funzione javascript *f_gmes* (presente nello script di base) che ha come compito quello di recuperare il messaggio di indice 11 dal file "messages.txt". In questo caso il messaggio 11 è proprio la parola "termina".



I file di progetto success.html, fail.html, success_img.txt, fail_img.txt

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

```
<h1>COMPLIMENTI !!!</h1>
<pcenter>
___IMAGE___
</pcenter>
<p>
Hai trovato il prezioso Occhio Purpureo.
Con la sua vendita risanerai i tuoi debiti e vivrai
per sempre di rendita! Ben fatto!!!
</p>
<hr>
```

Come per "cover.html", anche qui è possibile aggiungere una immagine. Possiamo usare quella al link qui indicato e poi incollare la codifica base 64 nel file "success_img.txt":

www.epaperadventures.qlmagic.com/success_example.jpg

Facciamo lo stesso per il file "fail.html", che contiene il contenuto della schermata che compare quando si fallisce l'avventura. L'immagine da usare e da copiare in "fail_img.txt" possiamo trovarla qui:

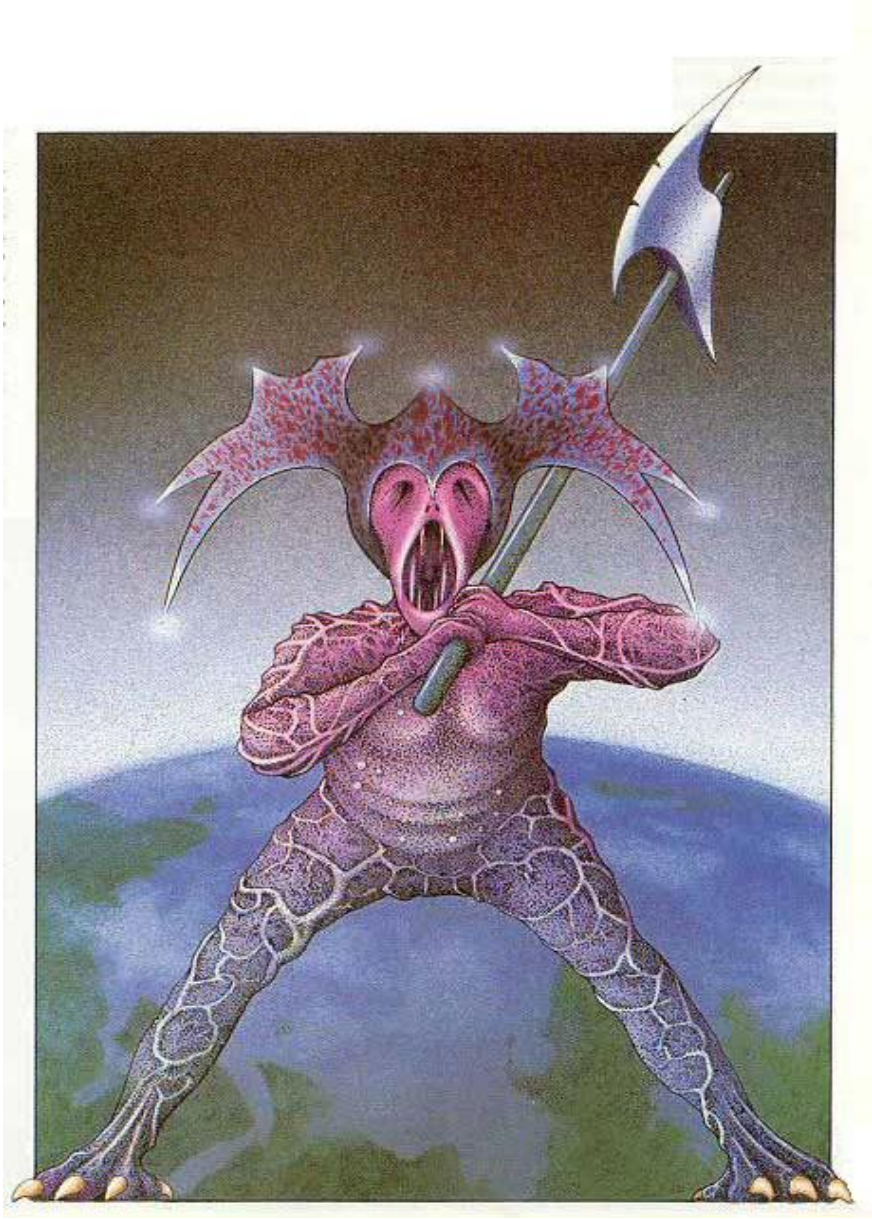
www.epaperadventures.qlmagic.com/fail_example.jpg

Il file "fail.html" lo modifichiamo per adesso invece in questo modo:

```
<h1>HAI FALLITO !</h1>
<pcenter>
___IMAGE___
</pcenter>
<p>
Non sei riuscito a trovare il prezioso Occhio Purpureo.
Ritenta, sarai più fortunato!
</p>
<hr>
```

Una delle domande che ci si può fare e se è possibile modificare le schermate di successo e fallimento dinamicamente, ovvero in base a quello che è successo nell'avventura.

In questa versione di Pandor+ è possibile farlo attraverso un trucco che verrà spiegato nella parte finale di questo tutorial, col quale alla scritta "HAI FALLITO!" sostuiremo la descrizione di come il giocatore ha terminato la sua avventura ("Sei morto fulminato!", "Sei finito in gattabuia!", etc).



Il file di progetto messages.txt

Il file "messages.txt" contiene i testi che sono utilizzati per comandi e messaggi in alcuni dei contesti tipici del gioco. Il file è già completo, quindi volendo possiamo mantenerlo così com'è e eventualmente aggiungere dei nuovi messaggi.

Possiamo modificare i messaggi a nostro piacimento, seguendo alcune regole. Tuttavia non deve cambiare il loro scopo ovvero il "contenuto" informativo del messaggio non deve variare.

Ogni messaggio è preceduto da una linea con un indice tra < e >. I messaggi devono stare su un'unica linea, ma vedremo che esiste un trucco per poter scrivere messaggi lunghi su più righe. I caratteri ammessi per i testi in questo file sono **esclusivamente** lettere dell'alfabeto italiano (anche accentate) minuscole e maiuscole, numeri, spazio, apostrofo e questi caratteri:

```
! " £ % & ? , . + - : ; * / = [ ] ( ) < > |
```

In questo caso & verrà visualizzato e non si può utilizzarlo per inserire caratteri in codifica HTML e lo stesso vale per < > che non possono essere usati per tag HTML. Attenzione anche che non si potrà usare < come primo carattere del messaggio.

Vediamo i messaggi nel dettaglio.

```
<01>

Hai con te|Porti con te|Stai portando|Stai portando con te
```

Frase che appare prima del nostro inventario. Utilizzando il separatore | , possiamo indicare che vogliamo che ne venga scelta una a caso automaticamente durante il gioco.

Possiamo utilizzare questa possibilità solo per i messaggi che nel file "messages.txt" di default già utilizzano il separatore |.

```
<02>

Qui vedi|Qui vedi anche|Puoi anche vedere|Qui puoi anche vedere
```

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

```
<03>

Non hai nulla con te.|Non stai portando nulla!|Non stai portando niente...
```

Frase che appare quando non abbiamo nulla.

```
<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 help.

```
<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!
```

Comando per aprire nel browser il sito di riferimento che sceglieremo.

```
<10>
home
```

Utilizzato internamente dallo script di base. Non modificare.

```
<11>
```

```
termina
```

Comando per uscire dal gioco.

```
<12>
Termina avventura
```

Utilizzato internamente dallo script di base. Non modificare.

```
<13>
carica
```

Comando per caricare un salvataggio dell'avventura.

```
<14>
salva
```

Comando per salvare la posizione del giocatore nell'avventura.

```
<15>
Carica posizione...
```

Utilizzato internamente dallo script di base. Non modificare.

```
<16>
Salva posizione...
```

Utilizzato internamente dallo script di base. Non modificare.

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

Parte della frase utilizzata quando viene descritto l'inventario.

```
<18>
usa con...
```

Comando "usa con...". Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<19>
esamina
```

Comando "esamina". Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<20>
prendi
```

Comando "prendi". Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<21>
lascia
```

Comando "lascia". Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<22>
indossa
```

Comando "indossa". Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<23>
togli
```

Comando "togli". Viene aggiunto automaticamente in base alla definizione di un oggetto.

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

Comando per la conferma di uscita dall'avventura.

```
<25>
vai
```

Comando "vai". Viene aggiunto automaticamente in base alla definizione di un oggetto.

```
<26>

Continua...
```

Comando "continua". Compare in diverse schermate, ad esempio per uscire dai messaggi popup.

```
<27>

Stai indossando
```

Parte della frase utilizzata quando viene descritto l'inventario.

```
<28>

Posizione caricata!
```

Messaggio che compare quando si è caricata una posizione di gioco.

```
<29>

Posizione 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 posizione _

prima di uscire dal browser!
```

Messaggio che compare quando si salva una posizione di gioco negli slot 1,2 o 3.

Essendo un messaggio lungo, viene spezzato in più linee (che però saranno considerate come una linea unica) attraverso l'uso dei caratteri SPAZIO + UNDERSCORE come ultimi caratteri della linea da continuare (attenzione a non avere spazi o tab dopo l'underscore!).

Questo vale per qualsiasi altra linea in qualsiasi file di progetto: se una linea termina con SPAZIO + UNDERSCORE, viene concatenata la linea successiva. Pandor+ la considererà una linea unica.

```
<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 per cui non è prevista una azione particolare.

```
<31>

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

Frasi che possono apparire dopo un comando di esamina che rivela un 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 che appaiono quando si tenta di prendere un oggetto ma si hanno già troppe cose.

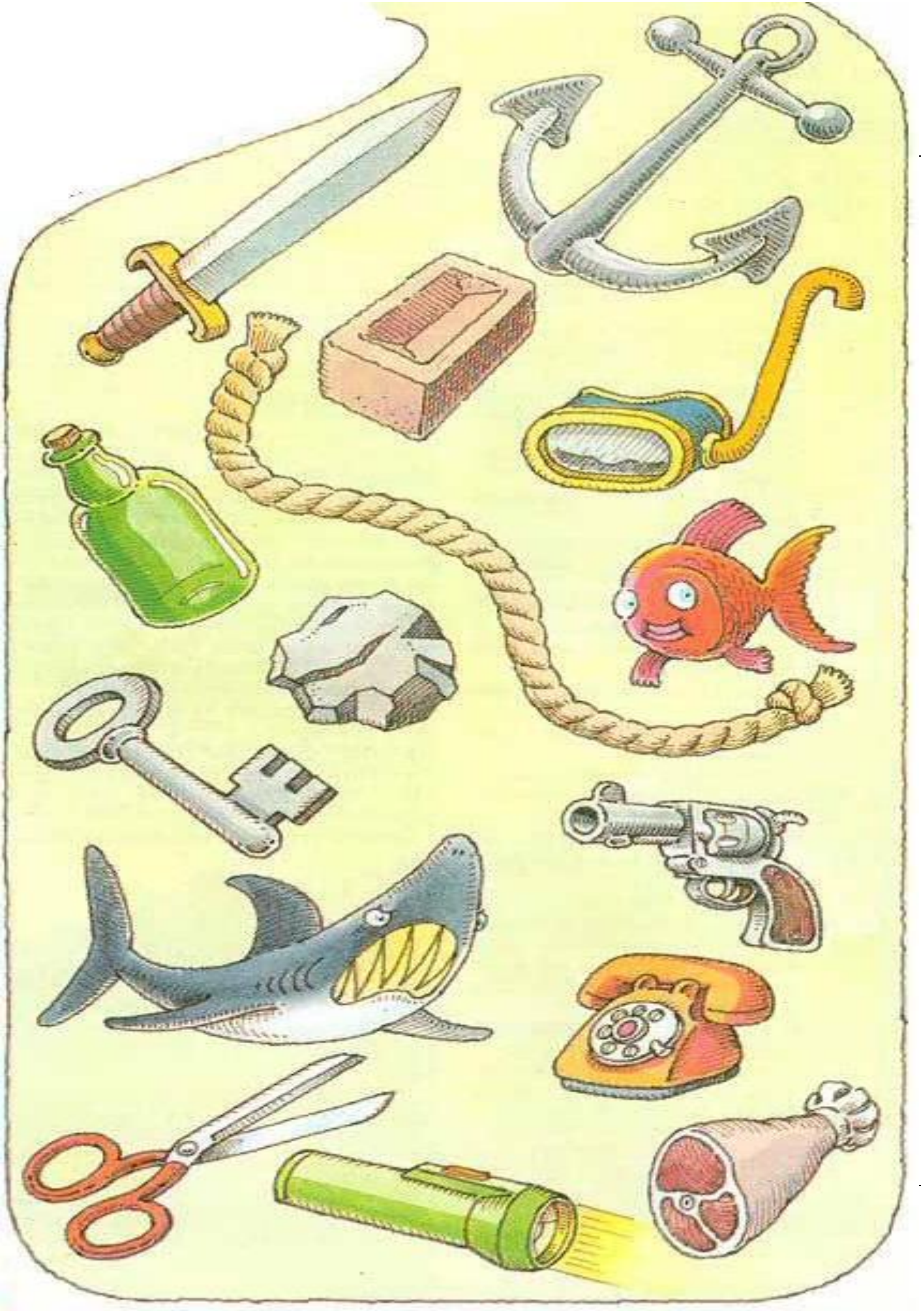
```
<33>

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

Frasi che appaiono a seguito di un comando "usa con" su una coppia di oggetti per i quali non è prevista nessuna azione.

```
<34>

L'indirizzo della pagina verrà automaticamente _

salvato nella cronologia assieme alla tua posizione! _

Quando vorrai riprendere da questo punto _

vai nella cronologia, scegli la pagina e _

quindi nel gioco tocca _

"CARICA" e "#4".
```

Frase che appare quando si salva la propria posizione tramite URL.

Come detto, il file può essere lasciato così com'è. Si possono ovviamente aggiungere messaggi e usarli nell'avventura tramite la funzione javascript *f_gmes.* Vedremo un esempio di questo nel capitolo dedicato alla caratterizzazione dei testi delle schermate di successo e fallimento dell'avventura.



Il file di progetto rooms.txt

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

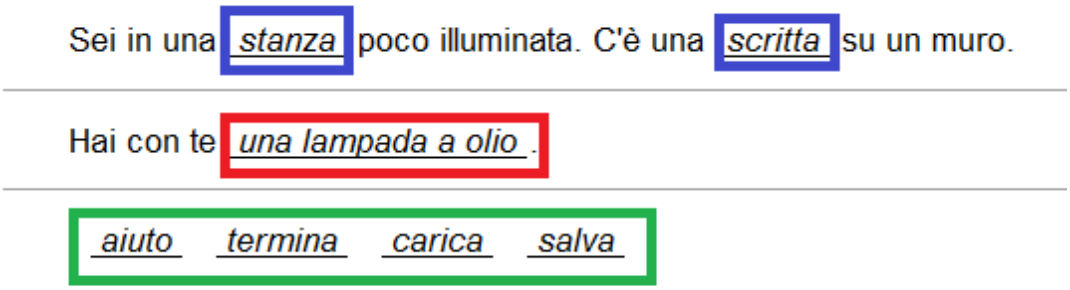
Prima di vedere come va scritto, introduciamo il concetto di **oggetto**. Per Pandor+ un oggetto è tutto quello che può essere cliccato o toccato e al quale è associato un menù di azioni. Li possiamo dividere in 3 tipi.

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.

Gli **oggetti "default"** sono legati ai comandi del motore di gioco di Pandor+.

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



Quindi: "stanza" e "scritta" 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". Per quel che riguarda gli oggetti default, l'unica cosa che possiamo fare è modificare eventualmente il testo associato (ad esempio sostituendo "termina" con "quit") che si trova nel file "messages.txt".

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

```
<01>
Sei in una {stanza|_stanza01_} poco illuminata.
C'è una {scritta|_scritta01_} su un muro.
```

```
<02>

Seconda stanza...
```

In pratica ogni descrizione di un ambiente (o stanza) comincia con una linea di tipo < *numero stanza* >. 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 testato è 99 stanze).

Come si vede, a differenza del file "messages.txt", per la descrizione possiamo utilizzare più linee (gli "a capo" verranno convertiti in spazi automaticamente) senza dover usare SPAZIO + UNDERSCORE.

I caratteri ammessi in "rooms.txt" sono **esclusivamente** lettere dell'alfabeto italiano (anche accentate) minuscole e maiuscole, numeri, spazio, apostrofo e questi caratteri:

```
! " £ % & ? , . + - : ; * / = [ ] ( ) < >

{ } | _
```

Il carattere & anche in questo caso viene visualizzato e lo stesso vale per i caratteri < >. Quindi non si possono introdurre tag o caratteri in codifica HTML.

Le parentesi graffe, il carattere pipe e l'underscore (in rosso) invece non vengono visualizzati e servono a inserire un oggetto statico nella descrizione. Per farlo dobbiamo scrivere:

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

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

```
{stanza|_stanza01_}
```

è un oggetto che viene visualizzato con la parola "stanza" e il cui identificatore è *_stanza01_*

Attenzione: tutti gli identificatori di oggetti devono sempre iniziare e terminare con un unico UNDERSCORE. Tra gli underscore possono essere utilizzate solo lettere minuscole e maiuscole (non accentate) e numeri.

Come buona regola (non obbligatoria), conviene che l'identificatore contenga il numero di ambiente nel quale è presente inizialmente. Nel caso in cui non sia ancora comparso o sia posseduto inizialmente dal giocatore, possiamo usare il numero 00.

Esempi di identificatori validi sono: *_Finestra03_* , *_PortaBlu17_* , *_Nord04_* , *_spada00_* , *_vecchio_* e così via.

Attenzione che Pandor+ non distingue tra maiuscole e minuscole nell'utilizzo degli identificatori. Non si può quindi aggiungere un oggetto *_vecchio_* e poi anche *_Vecchio_* perché per Pandor+ sarebbero lo stesso identificatore. Tuttavia, **bisogna sempre stare attenti a utilizzare gli identificatori sempre con le stesse maiuscole o minuscole** una volta introdotti!

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

Il numero massimo di oggetti (fissi + mobili) testato è 99, non si dovrebbe superare questa quantità oppure la funzione di salvataggio del gioco potrebbe non funzionare.

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

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

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

<02>
Sei in una {foresta pietrificata|_foresta02_}.
A {Sud|_sud02_} scorgi un sentiero mentre
ad {Ovest|_ovest02_} senti il rumore di un fiume.

<03>
Sei su un sentiero fangoso sul quale puoi
procedere a {Nord|_nord03_} ed {Est|_est03_}.

<04>
Sei accanto al {cancello|_cancello04_} che in direzione Est
ti fa entrare nella Città Segreta.
Ad {Ovest|_ovest04_} vedi un sentiero.

<05>
Sei all'esterno di un grande {edificio|_edificio05_}
situato a {Sud|_sud05_}.

<06>
Sei in una stanza polverosa.
```

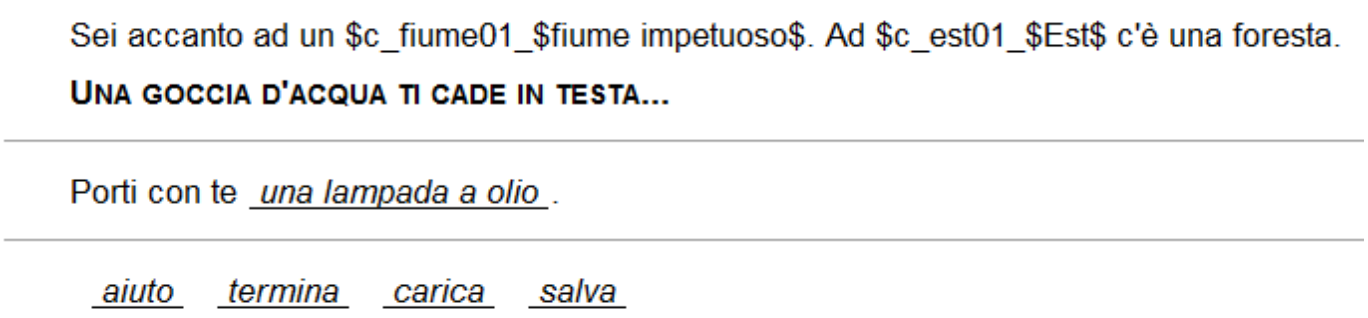
```
Puoi andare a {Nord|_Nord06_}, {Est|_Est06_} e {Sud|_Sud06_}.


<07>

Sei nell'ingresso principale.

Puoi andare a {Nord|_nord07_}, {Est|_est07_}, {Ovest|_ovest07_} e {Sud|_sud07_}.


<08>

Sei in una stanza con un {mobiletto|_mobiletto08_}.

A {Nord|_nord08_} torni verso l'ingresso.


<09>

Sei in una stanza buia.

Non riesci a vedere un'uscita!


<10>

Sei nel cortile. Ti puoi muovere a {Nord|_nord10_}, {Est|_est10_} ed {Ovest|_ovest10_}.


<11>

Sei nel giardino. Da qui puoi proseguire ad {Ovest|_ovest11_} e {Sud|_sud11_}.


<12>

Sei nella sala del {trono|_trono12_}.

Dal soffitto pende una {catena|_catena12_}.

A {Nord|_nord12_} esci verso un grande giardino.
```

Come si può notare la stanza 09 non ha nessun oggetto che permetta di fare alchunchè e quindi nemmeno muoversi, condannando il giocatore a dover terminare il gioco. La descrizione verrà cambiata dinamicamente nel momento in cui il giocatore possiede la lampada e la accende.

Modificato e salvato il file, proviamo a generare l'avventura con Pandor+. Verranno visualizzati diversi warning, relativi al fatto che gli oggetti introdotti nelle descrizioni non sono definiti nel file "objects.txt":

```
05/06/2016 23:18:16 > Warning: missing objects specification
_fiume01_  _est01_  _foresta02_  _sud02_  _ovest02_  _nord03_  _est03_
_cancello04_  _ovest04_  _edificio05_  _sud05_  _Nord10  _Est10_  _Sud10  _nord07_
_est07_  _ovest07_  _sud07_  _mobiletto08_  _nord08_  _nord10_  _est10_  _ovest10_
_ovest11_  _sud11_  _trono12_  _catena12_
```

Vedremo più avanti quali sono tutti i messaggi di warning e di errore che possono aiutarci nel debug della nostra avventura. Proviamo comunque a visualizzare nel browser il file "debug.html".



Come si può vedere la descrizione è quella dell'ambiente 1, ma gli oggetti fissi non si sono trasformati in niente di cliccabile. Rimane anche la lampada e il messaggio aggiuntivo del progetto di default. Provvederemo a breve a sistemare le cose!

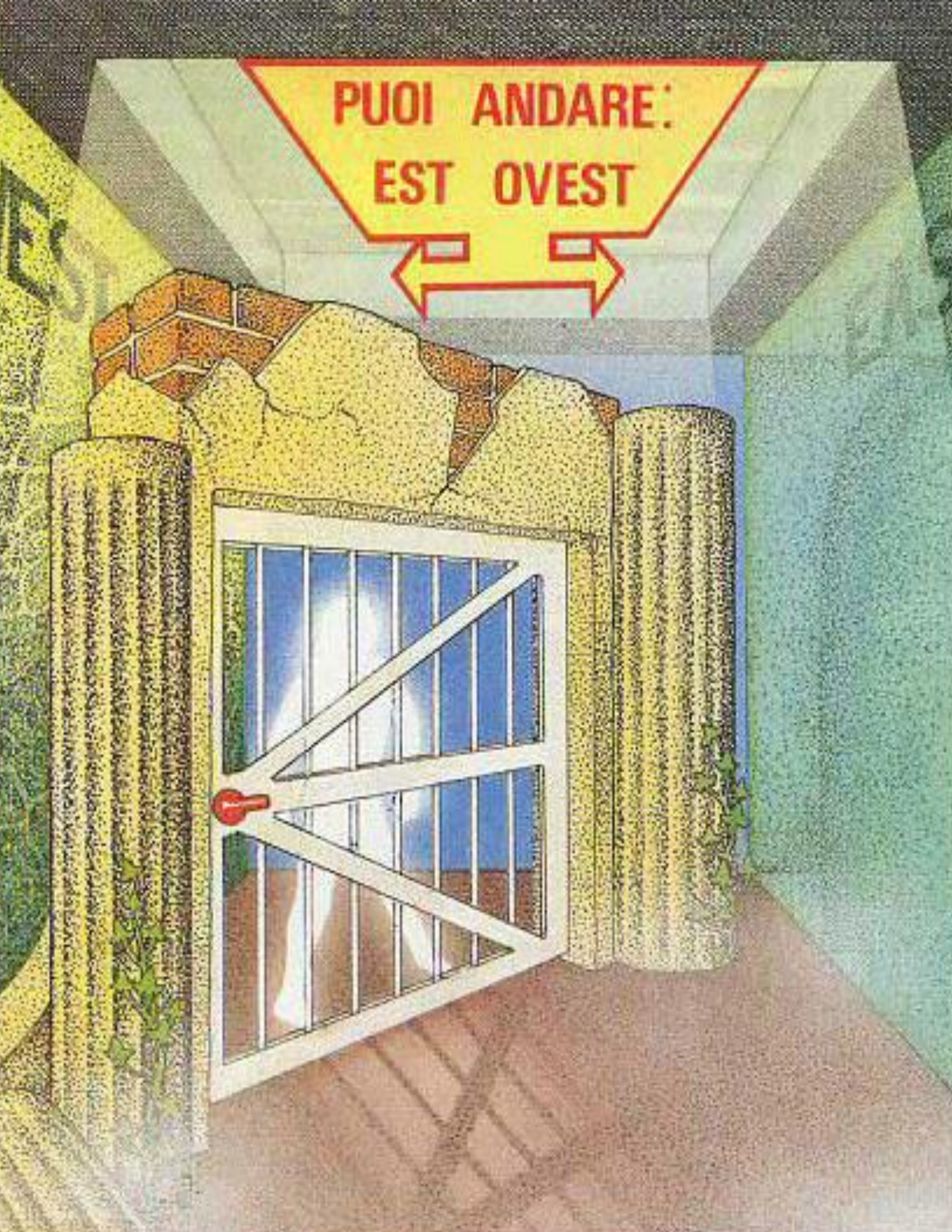
Nota: se stavamo già testando in precedenza il file "debug.html", dobbiamo sempre utilizzare l'opzione *termina* (e conferma) per riportarci alla schermata iniziale e poi effettuare il refresh della pagina (in Explorer premendo F5, ma in generale basta posizionarsi sull'indirizzo e premere INVIO su qualsiasi browser).

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_}.
```

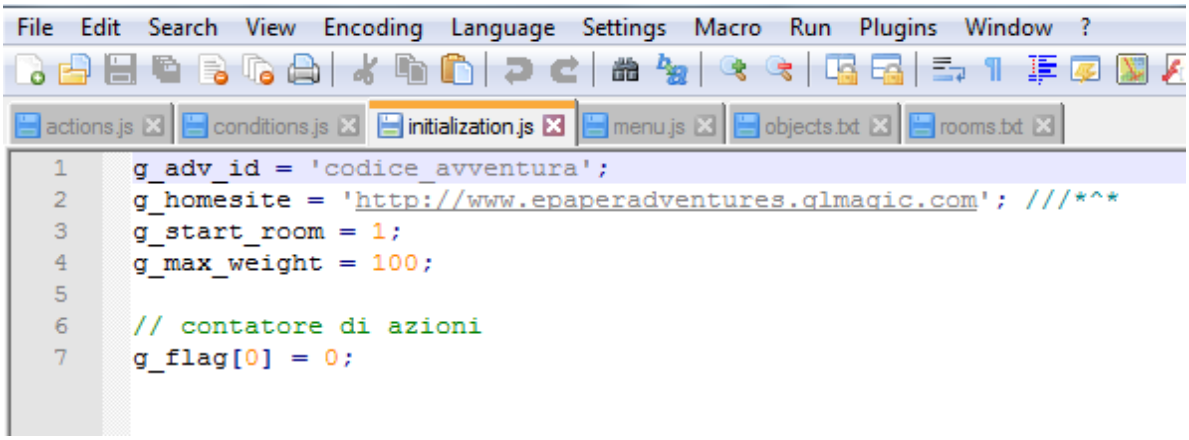
Pandor+ salterà tutte le righe di solo commento. Questo è vero per qualsiasi altro file di progetto.

Attenzione: non utilizzare mai per nei file di progetto tre slash consecutivi (ovvero "///"), neanche nei commenti o Pandor+ non genererà correttamente l'avventura!


PUOI ANDARE:
EST OVEST

Il file di progetto initialization.js

I file con cui ci troveremo di più a lavorare sono sicuramente quelli presenti nella figura seguente.



Il file "initialization.js", come tutti i file con estensione js, contiene codice javascript che Pandor+ trasforma per integrarlo correttamente con lo script di base.

Questo file in particolare ci permette di definire alcune importanti variabili che contengono informazioni fondamentali del gioco.

g_adv_id contiene un codice testuale che deve essere unico per ogni avventura. Usate lettere (non accentate) maiuscole o minuscole, numeri e underscore.

g_homesite contiene il sito a cui si andrà cliccando sul link della schermata iniziale "Vai al sito …". Attenzione: lasciate sempre come ultimi caratteri di questa riga il commento `///*^*` o Pandor+ non utilizzerà in modo corretto il link. Questo è l'unico punto (in tutti i file di progetto) in cui è ammesso che esistano tre slash consecutivi (usatene sempre al massimo due per i vostri commenti!).

g_start_room contiene il numero di stanza da cui partirà l'avventura

g_max_weight contiene il peso (ingombro) massimo che possiamo trasportare (ogni oggetto avrà un suo numero di unità di peso)

Dopo queste, dobbiamo 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.

Questa variabili fanno parte dell'array di numeri interi di nome *g_flag*. Questo array è l'unico che viene salvato nel corso del gioco, quindi possiamo utilizzare solo e solamente i suoi elementi (variabili) per tener conto di cosa accade nell'avventura!

Supponiamo ad esempio che vogliamo tener traccia se il giocatore è passato per la stanza 8 oppure no. Nel codice del gioco ci sarà quindi una condizione del tipo:

SE il giocatore è nella stanza 8

ALLORA g_flag[1] = 1

Quindi, la variabile *g_flag[1]* (elemento 1 dell'array *g_flag*) diventa uguale a 1 quando il giocatore si trova almeno una volta nella stanza 8.

Ora immaginiamo che esista un quadro raffigurante proprio la stanza 8. Se lo si esamina e se si è passati per la stanza 8 vogliamo che compaia un messaggio. La condizione potrebbe allora essere:

SE il giocatore esamina il quadro *E* g_flag[1] è uguale a 1

ALLORA *visualizza il messaggio "Questo quadro rappresenta una stanza che hai già visto..."*

Allo stesso modo potremmo utilizzare altre variabili di *g_flag* di indice 2,3,4... per tener conto di situazioni o eventi analoghi.

Ogni volta che ne introduciamo una, va inizializzata nel file "initialization.js". Ed è una buona idea scrivere un commento che ci aiuti a ricordare il suo scopo.

Le variabili *g_flag[...]* possono assumere valori da 0 a 65533. **Non è quindi possibile usare valori negativi.**

Apriamo ora "initialization.js" del nostro progetto "Occhio" e modifichiamolo come segue.

```
g_adv_id = 'epa_occhio_purpureo';
g_homesite = 'http://www.epaperadventures.qlmagic.com'; ///*^*
g_start_room = 04;
g_max_weight = 100;

// contatore di azioni, incrementato ad ogni azione non vuota
g_flag[0] = 0;

// numero di volte che bisogna tirare la catena per vincere (da 1 a 3)
g_flag[1] = 0;
```

```
// l'ispettore è già comparso una volta (1 = è comparso, 2 = è stato ucciso)
g_flag[2] = 0;

// numero di azioni fatte nella foresta. Se si rimane più di due azioni, si muore pietrificati
g_flag[3] = 0;

// tipo di morte (valori possibili: 35,36,37,38 -> vedi messaggi in messages.txt
// questa variabile viene usata in "fail.html"
g_flag[4] = 0;

// abbiamo già nuotato o no nel fiume (1 = si)
g_flag[5] = 0;

// contatore di azioni dal momento in cui l'ispettore compare
g_flag[6] = 0;
```

Ogni variabile ha uno scopo descritto nel commento e un certo valore iniziale. Il numero massimo di variabili (dimensione array *g_flag*) testato è 99. Oltre si potrebbero avere problemi con la procedura di salvataggio della posizione nel gioco.

"Initialization.js" è in pratica un pezzo di codice javascript che verrà inserito nello script di base. Di conseguenza, i caratteri ammessi all'interno di esso sono tutti quelli ammessi dall'interprete javascript. Lo stesso vale per tutti i file di estensione ".js" del progetto.

Un'ultima nota importante riguarda il controllo del codice scritto nel file "initialization.js". Pandor+ non controlla che il codice javascript sia corretto o che si usino le variabili corrette. Gli eventuali errori di sintassi o usi errati di variabili potranno essere rilevati solamente attraverso il debug col browser. Questo è vero per tutti i file con estensione ".js" del progetto.

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:

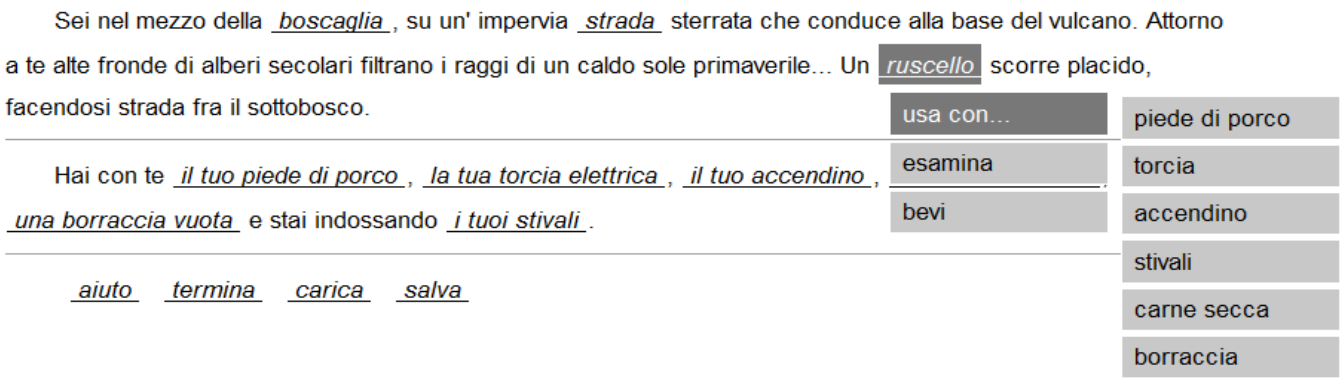
- non prendibili (casa, nave, albero...)
- prendibili (lampada, moneta, mattone...)
- prendibili e indossabili (armatura, zaino, guanti...)

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 *g_max_weight*.

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.



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 potremmo voler aggiungere degli oggetti mobili, specificandone le caratteristiche. Ogni oggetto va definito con una sequenza di linee, in questo modo:

```
<identificatore_oggetto>
desc_short: descrizione breve
desc_long: descrizione lunga
desc_examine: messaggio da mostrare se esaminato
found: identificatore oggetto che compare a seguito di esamina
desc_found: messaggio di scoperta di un nuovo oggetto
room: posizione oggetto
weight: peso se trasportato
weight_worn: peso se indossato
direction: stanza raggiunta in caso di azione di movimento
desc_direction: descrizione azione di movimento
examinable usable takeable wearable disabled
```

Fondamentalmente specificheremo solo le proprietà che ci interessano. Alcune permettono anche 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 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

Vediamo le proprietà una alla volta.

```
<identificatore_oggetto>
```

Specifica quale oggetto viene definito, ad esempio: <*_lampada01_*>.

```
desc_short: descrizione breve
```

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: descrizione lunga
```

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.

```
desc_examine: messaggio da mostrare se esaminato
```

Se è selezionata la proprietà "examinable" nell'ultima linea, questo è il messaggio che verrà mostrato a seguito dell'azione "esamina". Ad esempio: *E' una vecchia lampada rotta.* Se lungo, il messaggio può essere spezzato in più linee utilizzando SPAZIO+UNDERSCORE. Se non viene specificata questa proprietà, a seguito dell'azione "esamina", verrà visualizzato un messaggio di default tipo "Non trovi nulla di interessante" (vedere "messages.txt").

```
found: identificatore oggetto che compare a seguito di esamina
```

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: messaggio di scoperta di un nuovo oggetto
```

Se è stato specificata la proprietà found, qui scriveremo il messaggio aggiuntivo che compare (un'unica volta) all'azione "esamina". Se lungo, il messaggio può essere spezzato in più linee utilizzando SPAZIO+UNDERSCORE. Se non viene specificata questa proprietà, a seguito dell'azione "esamina", verrà visualizzato un messaggio di default tipo "Hai trovato qualcosa" (vedere "messages.txt").

```
room: posizione oggetto
```

Qui va specificata la stanza in cui si trova l'oggetto nel caso sia un oggetto mobile. Ad esempio: 05 . Nel caso in cui l'oggetto non sia in nessuna stanza, va messo il valore 0. Nel caso in cui sia nell'inventario va messo -1, se è indossato -2 (in questo caso l'oggetto deve avere la proprietà *wearable*).

```
weight: peso se trasportato
```

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

```
weight_worn: peso se indossato
```

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.*

```
direction: stanza raggiunta in caso di azione di movimento
```

Se vogliamo aggiungere al menu dell'oggetto un azione di movimento che porta in un'altra stanza possiamo valorizzare questa proprietà con il numero della stanza. Ad esempio: 13 . Non va di norma utilizzata per oggetti mobili. Inoltre, non dovrebbe mai essere valorizzata a -1 o 999 (valori di fine gioco, come vedremo).

```
desc_direction: descrizione azione di movimento
```

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" (vedi "messages.txt").

```
examinable usable takeable wearable disabled
```

Nell'ultima linea va specificato se l'oggetto ha alcune caratteristiche o no:

- 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 (se non ne abbiamo, l'azione non sarà presente). Le varie azioni "usa con…" tra gli oggetti vanno poi gestite nel file "actions.js".
- 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 si specifica la proprietà *disabled* l'oggetto apparirà come un testo normale e non sarà attivabile nessun menu. Si può successivamente attivarlo in "conditions.js" o "actions.js"

L'idea di base è quindi che le proprietà `direction examinable takeable wearable` 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 dallo script di base il menu "usa con..." con la lista degli oggetti dell'inventario. Le possibili conseguenze delle interazioni tra oggetti andranno specificate in "actions.js".

Per oggetti che si comportano in modo differente ai comandi "esamina", "prendi" , "indossa" bisogna invece rinunciare all'utilizzo delle proprietà viste a favore di una gestione manuale dei menu e delle azioni attraverso i file di progetto "menu.js" e "actions.js". Un altro esempio sono oggetti con l'azione "usa con..." che però vogliamo abbia nel sotto-menu oggetti diversi da quelli dell'inventario. Anche in questo caso non possiamo usare la proprietà "usable", ma dovremo aggiungere a mano l'azione e i sotto-menu (vedremo come!).

In generale, i caratteri ammessi per valorizzare le proprietà sono lettere (maiuscole e minuscole), numeri, spazio e i caratteri:

```
! " £ % & ? , . + - : ; * / = [ ] ( ) < > { } |
```

Il carattere & anche in questo caso viene visualizzato e lo stesso vale per i caratteri < >. Quindi non si possono introdurre tag o caratteri in codifica HTML. **I caratteri $ e _ (UNDERSCORE) non vanno mai usati.**

Apriamo "objects.txt" e modifichiamolo come segue.

```
// oggetti fissi - solo direzioni
// ------------------------------

<_est01_>
direction: 02
desc_direction: vai

<_sud02_>
direction: 03
desc_direction: vai
```

```
<_ovest02_>
direction: 01
desc_direction: vai

<_nord03_>
direction: 02
desc_direction: vai

<_est03_>
direction: 04
desc_direction: vai

<_ovest04_>
direction: 03
desc_direction: vai

<_sud05_>
direction: 06
desc_direction: vai

<_Nord06_>
direction: 05
desc_direction: vai

<_Est06_>
direction: 09
desc_direction: vai

<_Sud06_>
direction: 07
desc_direction: vai
```

```
<_nord07_>
direction: 06
desc_direction: vai

<_est07_>
direction: 10
desc_direction: vai

<_ovest07_>
direction: 04
desc_direction: vai

<_sud07_>
direction: 08
desc_direction: vai

<_nord08_>
direction: 07
desc_direction: vai

<_sud09_>
direction: 10
desc_direction: vai

<_ovest09_>
direction: 06
desc_direction: vai

<_nord10_>
direction: 09
desc_direction: vai

<_est10_>
```

```
direction: 11
desc_direction: vai

<_ovest10_>
direction: 07
desc_direction: vai

<_ovest11_>
direction: 10
desc_direction: vai

<_sud11_>
direction: 12
desc_direction: vai

<_nord12_>
direction: 11
desc_direction: vai




// oggetti fissi - più azioni
// -------------------------

<_fiume01_>
desc_examine: La corrente è molto forte... Stai attento!
examinable
// da aggiungere: nuota -> trovi la pistola

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

```
<_cancello04_>
direction: 07
desc_direction: entra
desc_examine: _
Da qui entri nella Città Segreta, dove si dice sia _
nascosto l'Occhio Purpureo.
examinable

<_edificio05_>
// da aggiungere: esamina -> diversi messaggi

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

<_trono12_>
desc_examine: Il trono è in ferro battuto, circondato da drappi di seta.
examinable usable
// da aggiungere: siediti -> messaggio

<_catena12_>
desc_examine: Oscilla cigolando.
examinable
// da aggiungere: tira 1,2,3 volte -> risultato dipende se ho occhio o no!



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

<_sacchettoPieno05_>
```

```
desc_short: sacchetto
desc_long: un sacchetto pieno
desc_examine: Contiene delle biglie colorate.
room: 05
weight: 20
examinable takeable usable
// da aggiungere: vuota -> compaiono le biglie

<_sacchettoVuoto00_>
desc_short: sacchetto
desc_long: un sacchetto vuoto
desc_examine: Non contiene nulla.
room: 00
weight: 5
examinable takeable usable

<_biglie00_>
desc_short: biglie
desc_long: delle biglie colorate
room: 00
weight: 15
desc_examine: Sono lucenti e coloratissime.
desc_found: Tra di esse scorgi con stupore il prezioso Occhio Purpureo!
found: _occhio00_
examinable usable takeable

<_occhio00_>
desc_short: occhio
desc_long: il prezioso Occhio Purpureo
room: 00
weight: 5
desc_examine: _
Deve valere un mucchio di soldi! I tuoi problemi economici sono finiti!
```

```
Examinable takeable


<_mattone03_>
desc_short: mattone
desc_long: un mattone
room: 03
weight: 40
examinable takeable usable
// da aggiungere: tira -> se c'è ispettore, lo schiva
// da aggiungere: usa con lampada -> si rompe la lampada

<_pistolaCarica00_>
desc_short: pistola
desc_long: una pistola carica
room: 00
weight: 30
takeable usable
// da aggiungere: spara -> se c'è ispettore lo uccido, altrimenti colpo a vuoto

<_pistolaScarica00_>
desc_short: pistola
desc_long: una pistola scarica
room: 00
weight: 20
takeable usable
// da aggiungere: spara -> non succede niente

<_lampadaSpenta00_>
desc_short: lampada
desc_long: una lampada spenta
room: 00
weight: 20
takeable
```

```
// da aggiungere: accendi

<_lampadaAccesa00_>
desc_short: lampada
desc_long: una lampada accesa
room: 00
weight: 20
takeable
// da aggiungere: spegni

<_lampadaRotta00_>
desc_short: lampada
desc_long: una lampada rotta
room: 00
weight: 20
takeable
// da aggiungere: accendi -> non funziona

<_guanti11_>
desc_short: guanti
desc_long: dei guanti da lavoro
desc_examine: Sono dei guanti di gomma.
room: 11
weight: 20
weight_worn: 5
examinable takeable wearable

<_ispettore00_>
desc_short: ispettore
desc_long: Il famigerato ispettore delle tasse!
desc_examine: E' uno spilungone magro e dal sorriso diabolico!
room: 00
examinable usable
```

```
<_cadavere00_>
desc_short: cadavere
desc_long: Il cadavere dell'ispettore delle tasse
desc_examine: R.I.P. !
room: 00
examinable usable
```

Gli oggetti dal primo fino a *_nord12_* sono fissi e avranno un'unica azione nel menu: "vai", collegata ad una direzione che muoverà il giocatore in una certa stanza.

Vediamo gli altri oggetti. Noteremo che alcuni di essi avranno le 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 "acrions.js" non sarà specificata una certa interazione tra gli oggetti, il gioco risponderà automaticamente con un messaggio tipo "Non funziona!" (si veda "messages.txt").

Da notare anche che sono state utilizzate linee di commento per ricordarci cosa dobbiamo fare con i vari oggetti. I commenti vanno sempre in linee separate dalle proprietà.

```
<_fiume01_>
```

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

Aggiungeremo invece a mano (in "menu.js") l'azione "nuota", a seguito della quale comparirà una pistola.

```
<_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 l'uso di SPAZIO+UNDERSCORE nella proprietà *desc_examine* per poter scrivere il messaggio su più righe.

```
<_edificio05_>
```

Questo oggetto non ha nessuna proprietà definita. Verrà aggiunta l'azione "esamina" manualmente, e il risultato dell'azione (mostreremo dei messaggi) sarà diverso di volta in volta.

```
<_mobiletto08_>
```

Viene aggiunta l'azione "esamina" a seguito della quale comparirà la prima volta una lampada. Viene anche aggiunta "usa con..." con l'elenco degli oggetti dell'inventario.

```
<_trono12_>
```

Per il trono vengono aggiunti "esamina" e "usa con...". Verrà aggiunta manualmente "siediti".

```
<_catena12_>
```

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

L'ultimo gruppo di oggetti sono quelli mobili. Per oggetti che possono avere più "stati" (lampada accesa – lampada spenta, ad esempio) è necessario definire un oggetto "gemello" per ogni combinazione di caratteristiche 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 0. Il trucco starà 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.

```
<_sacchettoPieno05_>
<_sacchettoVuoto00_>
```

Il sacchetto può o no contenere delle biglie. Inizialmente viene piazzato "sacchetto pieno" nella stanza 5. Viene preparata la versione vuota (non presente però da nessuna parte, "room" è infatti 0). Per il "sacchetto pieno", aggiungeremo manualmente l'azione "vuota" (sarà proprio questa 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*.

```
<_mattone03_>
```

Causerà la morte del giocatore se tenteremo di nuotare con esso nel nostro inventario. Verrà aggiunta l'azione "lancia" 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. Andrà aggiunta l'azione "spara" manualmente.

```
<_lampadaSpenta00_>
<_lampadaAccesa00_>
<_lampadaRotta00_>
```

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

```
<_guanti11_>
```

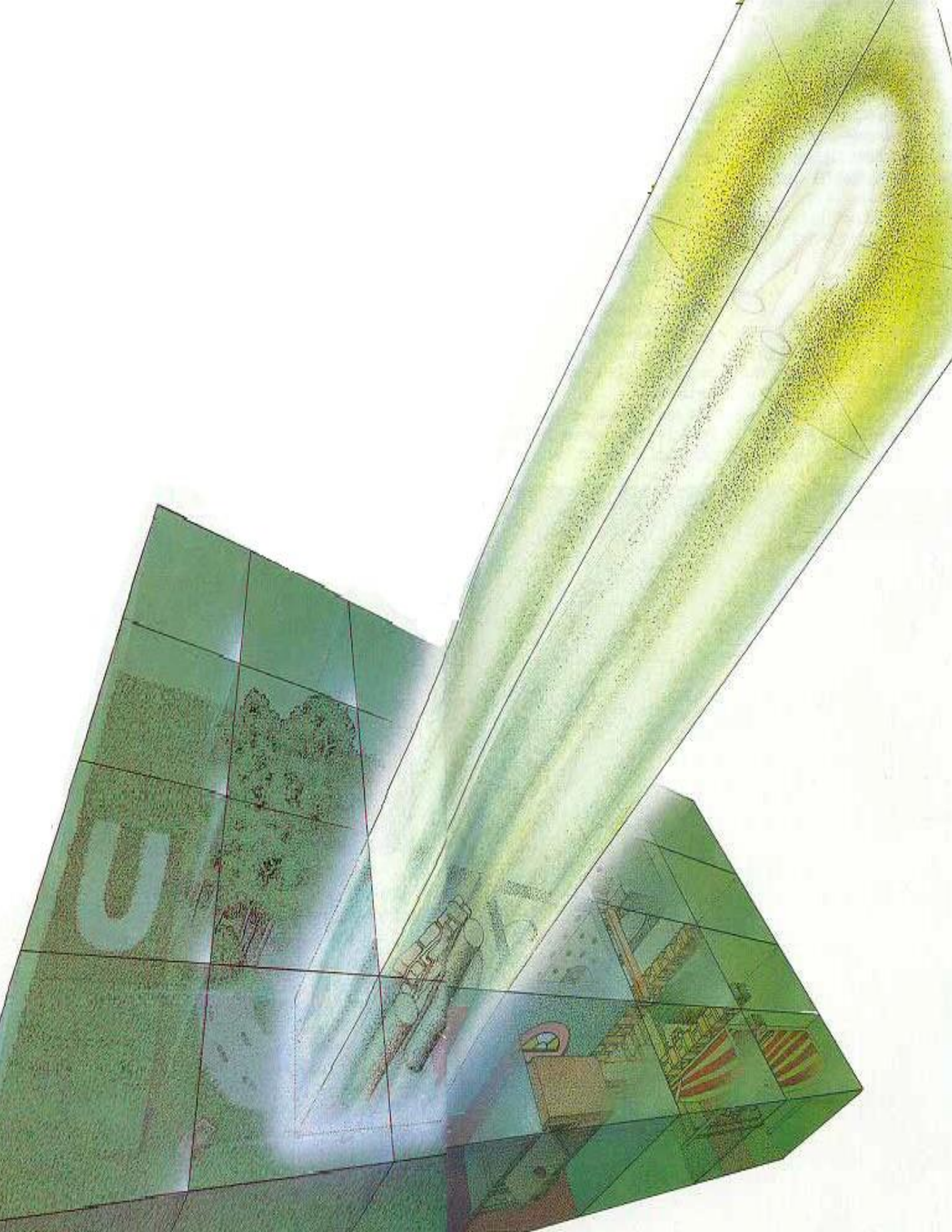
I guanti sono l'unico oggetto indossabile (proprietà "wearable") dell'avventura. Il peso se indossati è 5, contro 20 se trasportati. Serviranno a tirare la catena senza restare fulminati!

```
<_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 avverà nella stessa stanza del giocatore (a meno che non siano la 02 o la 09) dopo 20 azioni dall'inizio dell'avventura. Aggiungeremo a mano le azioni "spara" se il giocatore possiede una pistola o "colpisci" se possiede il mattone.

```
<_cadavere00_>
```

Se uccidiamo l'ispettore, al suo posto comparirà il suo cadavere.



Il file di progetto menu.js

In questo file aggiungeremo le azioni che non sono automaticamente "appese" agli oggetti. Come detto, le proprietà *examinable takeable usable direction wearable* fanno si che cliccando sull'oggetto nel menu siano già presenti le azioni "esamina", "prendi", "lascia", "indossa", "togli", "usa con…", "vai". Ma è possibile aggiungere tutte le azioni che vogliamo a ogni oggetto.

Ogni menu viene definito in modo simile a questo:

```
<identificatore_oggetto>
{
  [if (condizione1)]
    ++( azione1 , sotto_menu_azione1 );
  [if (condizione2)]
    ++( azione2 , sotto_menu_azione2 );
  ...
}
```

Ecco subito un esempio. Immaginiamo di voler aggiungere ad un possibile oggetto *_porta04_* le azioni "esamina", "apri" e "chiudi" in base allo stato della porta. Immaginiamo che sia la variabile *g_flag[10]* a contenere questo stato: 0 = la porta è chiusa, 1 = la porta è aperta. Infine, "esamina", "apri" e "chiudi" non hanno sotto-menù (come accade invece per l'azione "usa con…"). Ecco il codice per questo oggetto:

```
<_porta04_>
{
       ++( @"esamina"@ , null);
       if ( g_flag[10]==0 )
              ++( @"apri"@ , null);
       if ( g_flag[10]==1 )
              ++( @"chiudi"@ , null);
}
```

Il codice potrebbe essere ottimizzato usando il costrutto "if… else…", ma il risultato è lo stesso.

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".

Ad esempio:

```
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 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 c e d sono uguali
        // (una o entrambe le sotto-condizioni si verificano)
}
```

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

Naturalmente dentro ad un blocco *if* o *else* possono esistere altri costrutti *if*:

```
if  (a==b)
{
        // questo blocco viene eseguito se a e b sono uguali!
        if (c==d)
        {
                // questo blocco viene eseguito se a e b sono uguali e c e d sono uguali
        }
        else
        {
                // questo blocco viene eseguito se a e b sono uguali e c e d non sono uguali
        }
}
else
{
        // questo blocco viene eseguito se a e b non sono uguali
}
```

Ritorniamo all'esempio di *_porta04_*. Se il valore dentro *g_flag[10]* è 0 viene chiamata la funzione ++ che ha due parametri, il testo dell'azione da aggiungere al menu e una eventuale lista di sotto-menù. Se non c'è nessun sotto-menù, va messo *null*.

Il testo dell'azione è racchiuso da @" ... "@ . Pandor+ provvederà a "tradurre" in codice HTML tutto quello che si trova in mezzo (lettere con accento, caratteri speciali, etc.). **Non usate mai per i testi delle azioni e per i messaggi le sole virgolette singole (ad esempio: 'apri' va sostituito con @"apri"@).**

Lo script di base mette anche a disposizione alcuni testi di azioni "pronti". Sono quelli relativi a *esamina, prendi, lascia, indossa, togli, vai, usa con…* e sono disponibili nelle seguenti variabili che pescano i testi da messages.txt:

g_use_ = messaggio 18

g_examine_ = messaggio 19

g_take_ = messagio 20

g_leave_ = messaggio 21

g_wear_ = messaggio 22

g_takeoff_ = messaggio 23

g_confirm_ = messaggio 24

g_go_ = messaggio 25

Ad esempio, ecco come aggiungere l'azione "esamina":

```
  ++( g_examine , null);
```

In realtà *++(...)* è sostituito poi da Pandor+ con *f_additem(...)*, una funzione dello script di base, ma l'uso di ++ rende tutto più comodo.

Troveremo diverse chiamate di funzioni nella nostra avventura. Una funzione è semplicemente un blocco di codice che viene richiamato una o più volte al quale passiamo dei parametri e che esegue un compito e può restituirci un risultato. Ad esempio l'istruzione *c = somma(a,b)* usa una ipotetica funzione *somma* che riceve due numeri, li somma e mette il risultato nella variabile *c*. In questo caso ++ (ovvero *f_additem*) non restituisce risultati ma "carica" le azioni nei menù degli oggetti.

Ecco un altro esempio di aggiunta di azioni al menu di un oggetto:

```
<_cassa03_>
{
        ++( @"sposta"@ , null );
        if ( o_room[$(@"_piedediporco_"@)] == -1 )
        {
                ++( @"spacca"@ , null );
        }
}
```

In questo caso viene aggiunta sempre l'azione "sposta" mentre una seconda azione, "spacca", viene aggiunta solo se è verificata una certa condizione: l'oggetto con identificatore *_piedediporco_* deve essere presente nel nostro inventario. Per capirlo, viene usato l'array o*_room* che contiene le posizioni di tutti gli oggetti dell'avventura. L'indice corretto per l'array viene trovato tramite la funzione *$* ("$" è un abbreviazione che viene sostituita in fase di generazione da Pandor+ dalla funzione *f_findObjectIndexById*) che permette di ottenere questa informazione tramite l'identificatore dell'oggetto (racchiuso da @" … "@).

Ecco l'elenco **di tutte e sole** le variabili che possiamo utilizzare per le nostre condizioni:

g_room attuale posizione del giocatore (numero di stanza)

o_room[...] posizione di un oggetto; -2 = indossato; -1 = trasportato; 0 = non presente nel gioco; >0 stanza

g_flag[...] variabili di stato dell'avventura

Eventuali condizioni più complesse possono essere create come visto con gli operatori logici && (AND) e || (OR) e un uso appropriato delle parentesi. Ad esempio:

```
if (( o_room[$(@"_piedediporco_"@)] == -1 ) &&
    ( g_flag[28] == 1 ) && ( g_flag[0] > 15 ))
{
        ++( @"spacca"@ , null );
}
```

In questo caso devono essere verificate tre condizioni allo stesso tempo per avere l'azione "spacca" nel menu dell'oggetto.

Ecco infine un paio di esempi di un'azione con un sotto-menu, sullo stile di "usa con…". Per crearli, serve definire un array locale con gli elementi del sotto-menu.

```
<_leva08_>
{
        var l_list = [];
        l_list[0] = @"avanti"@;
        l_list[1] = @"indietro"@;
        l_list[2] = @"destra"@;
        l_list[3] = @"sinistra"@;
        ++( @"muovi a..."@ , l_list );
}
```

Con questo codice comparirà un azione "muovi a…" con sotto-menu "avanti", "indietro","destra" e "sinistra".

var permette di definire delle variabili locali al blocco di codice relativo all'oggetto.

Una importante regola è quella che per le variabili locali usate nel nostro codice (in tutti i file ".js") vanno usate o singole lettere (i,j,k…) oppure conviene chiamarle con un nome che inizia con "l_" (elle + UNDERSCORE). Questo eviterà errori di generazione dell'avventura da parte di Pandor+.

Immaginiamo ora di voler aggiungere manualmente (cioè non tramite l'opzione *usable* nella definizione dell'oggetto) un'azione "usa con…" con sotto-menu contenente:

- tutti gli oggetti dell'inventario (descrizione corta)
- un oggetto fisso (descrizione corta)
- la frase "me stesso"

Il codice per farlo sfrutta la funzione dello script di base *f_addinv*:

```
<_cordaPenzolante08_>
{
	var l_list = [];
	l_list = f_addinv(l_list); // aggiunge tutti gli oggetti dell'inventario
	var i = l_list.length; // prossimo indice
	l_list[i++] = @"_gabbia08_"@;
	l_list[i++] = @"me stesso"@;
	++( g_use_ , l_list );
}
```

La funzione ++ è fa si che mettendo nella lista l'identificatore dell'oggetto (*_gabbia08_)*, in seguito troveremo come sotto-menu la descrizione corta dell'oggetto stesso (e non l'identificatore).

Per i più arditi, ecco infine il codice per creare per un ipotetico oggetto cassaforte un menu con azioni:

- *sinistra 1*
- *sinistra 2*
- *…*
- *sinistra 9*

Per ognuna delle quali ci sono 9 sotto-menu:

- *destra 1*

- *destra 2*
- *...*
- *destra 9*

Creando così la possibilità di avere più combinazioni "cassaforte - sinistra X – destra Y".

```
<_cassaforte12_>
{
        var l_list = [];
        for (var i=0; i<9; i++)
                l_list[i] = @"destra "@ + (i+1);
        for (var i=0; i<9; i++)
                 ++(@"sinistra "@ + (i+1), l_list);
}
```

Il costrutto *for* permette di ripetere un blocco di codice (o un'unica istruzione, se il blocco {…} è assente) un certo numero di volte. Ad esempio con:

```
for (var i=0; i<9; i++)
```

stiamo chiedendo che la variabile *i* (creata localmente) assuma i valori da 0 a 8 incrementandola ad ogni "giro" di 1 (*i++* è come dire *i=i+1*).

Non rimane che aprire il file "menu.js" e modificarlo come segue, per inserire tutti i menu degli oggetti per l’avventura dell’Occhio Purpureo.

```
<_fiume01_>
{
        ++(@"nuota"@, null);
}

<_edificio05_>
{
        ++(g_examine_, null);
}
```

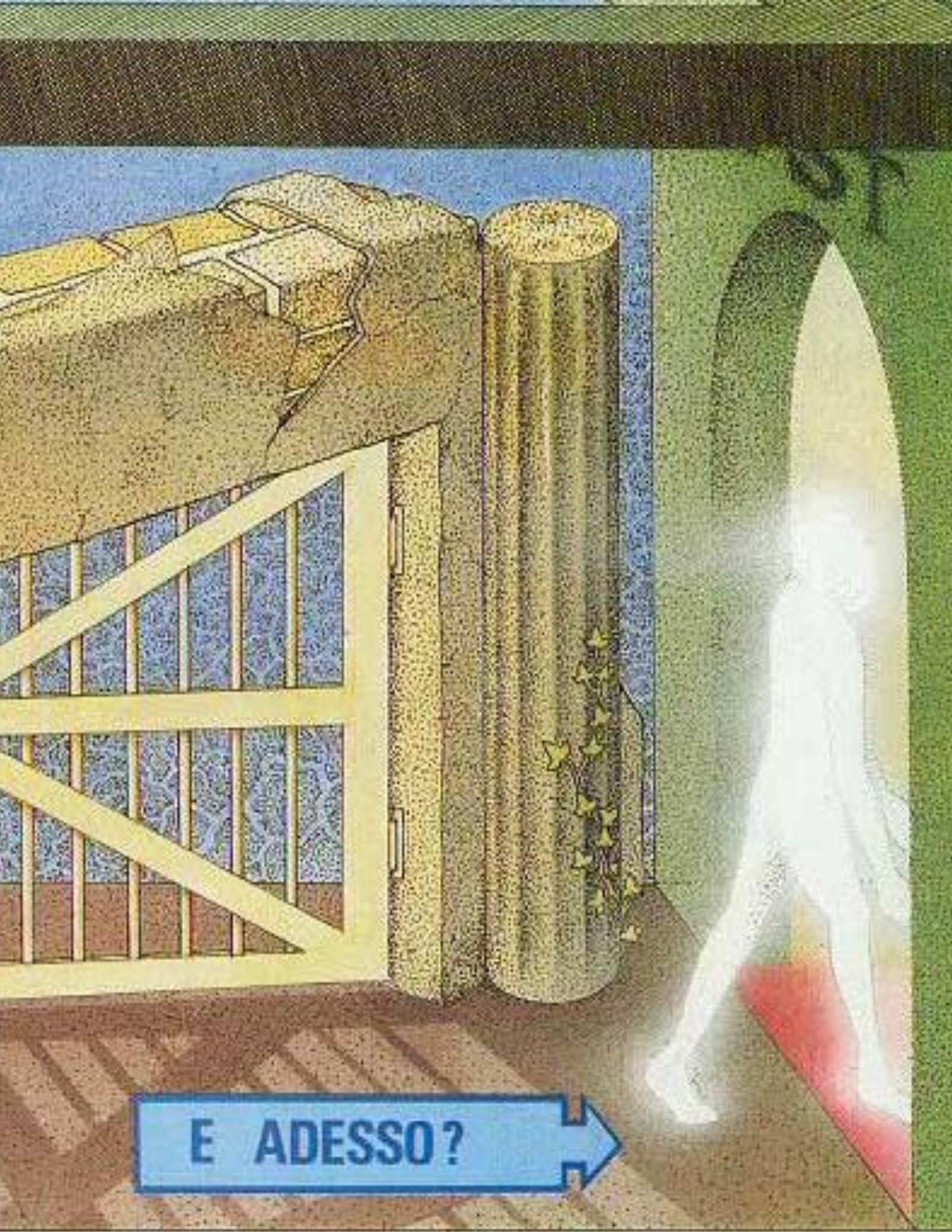
```
<_trono12_>
{
	++(@"siedi"@, null);
}

<_catena12_>
{
	var l_list = [];
	l_list[0] = @"una volta"@;
	l_list[1] = @"due volte"@;
	l_list[2] = @"tre volte"@;
	++(@"tira"@, l_list);
}

<_sacchettoPieno05_>
{
	++(@"vuota"@, null);
}

<_mattone03_>
{
	if (o_room[$(@"_mattone00_"@)] == -1)
	{
		++(@"lancia"@, null);
	}
}

<_pistolaCarica00_>
{
	++(@"spara"@, null);
}
```

```
<_lampadaSpenta00_>
{
        ++(@"accendi"@, null);
}

<_lampadaAccesa00_>
{
        ++(@"spegni"@, null);
}

<_ispettore00_>
{
        if (o_room[$(@"_pistolaCarica00_"@)] == -1)
        {
                ++(@"spara"@, null);
        }
        if (o_room[$(@"_mattone03_"@)] == -1)
        {
                ++(@"colpisci"@, null);
        }
}
```

L’oggetto catena avrà una azione "tira" con un sotto-menu di tre voci: una volta, due volte, tre volte.

L’oggetto mattone avrà un’azione "lancia" solo se è nell’inventario del giocatore.

Da notare anche per l’oggetto "ispettore" che le azioni "spara" e "colpisci" sono presenti solo se il giocatore possiede in quel momento la pistola carica o il mattone.


E ADESSO?

Il file di progetto conditions.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.

Apriamo il file "conditions.js" e vediamo come è composto nell'avventura di default:

```
if (is_action)
{
	// qui dentro vanno aggiornati i g_flag
	// quando vengono compiute delle azioni

	// g_flag[0] incrementato ad ogni azione
	g_flag[0] = g_flag[0] + 1;

	return;
}

// da qui vanno tutte le condizioni basate sui g_flag,
// sulla posizione del giocatore e degli oggetti

if (f_getrandomint(1,3) == 1)
{
	g_addtext = @"Una goccia d'acqua ti cade in testa..."@;
}
```

Il codice in questo file ci permette di definire dei mutamenti del gioco o dei messaggi da mostrare in base a determinate condizioni. Alcuni esempio 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 le condizioni presenti in "conditions.js". Questo avviene appena prima che lo script di base scriva su schermo la descrizione della stanza, gli oggetti nella stanza e l'inventario.

In questo modo, è possibile effettuare delle modifiche alla situazione del gioco non solo in risposta all'ultima azione compiuta dal giocatore (di questo ci si occupa nel file "actions.js"), ma anche in base a più fattori contemporaneamente.

In generale con "conditions.js" potremo fare delle modifiche alla descrizione della stanze, alla posizione degli oggetti, alla posizione del giocatore e scriveremo un messaggio "popup" o del testo aggiuntivo nella descrizione della stanza.

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 HTML del gioco. Le azioni vuote insomma non sono "mosse" e non modificano mai lo "stato" del gioco e del giocatore.

Anche se alcuni dettagli saranno più chiari dopo aver visto anche "actions.js", la sequenza (semplificata) del ciclo di operazioni eseguite dallo script di base ad ogni azione compiuta dal giocatore è questa:

1. il giocatore compie un'azione, vuota o no
2. se è un azione vuota allora viene fatto quanto necessario (esempio: viene mostrata la schermata di aiuto) e poi si vai al punto 6, altrimenti si continua al punto 3
3. viene valutato "actions.js": se qui l'azione è gestita (*return true*) allora vai al punto 5, altrimenti continua al punto 4. In questa fase può essere modificata la variabile *g_mex* (per mostrare un messaggio "popup")
4. viene gestita l'azione in modo standard, in base alla proprietà degli oggetti coinvolti nell'azione (esamina, prendi, etc) e preparato l'eventuale messaggio di risposta ("Niente di interessante...", "Hai troppe cose!", etc) nella variabile *g_mex*
5. viene valutato "conditions.js" con la variabile *is_action* = true. Di conseguenza viene eseguito solo quello che si trova dentro al corpo di *if (is_action)*
6. viene valutato "conditions.js" con la variabile *is_action* = false. Di conseguenza viene valutato tutto quello che si trova dopo il corpo di *if (is_action)*. In questa fase possono essere anche modificate *g_mex* e *g_addtext*

7. se previsto, viene mostrato un messaggio "popup" col contenuto di *g_mex* dal quale si esce toccando "Continua"
8. se la stanza del giocatore è -1, il gioco termina (schermata di fallimento)
9. se la stanza del giocatore è 999, il gioco termina (schermata di successo)
10. viene mostrata la descrizione in base alla stanza corrente ed agli oggetti presenti. Viene eventualmente aggiunto un messaggio addizionale in grassetto maiuscoletto dopo la descrizione della stanza, quello contenuto nella variabile *g_addtext*

Uno schema meno approssimato del funzionamento del ciclo di gioco è riportato anche nella pagina seguente. La sua comprensione non è comunque strettamente necessaria per scrivere le proprie avventure.

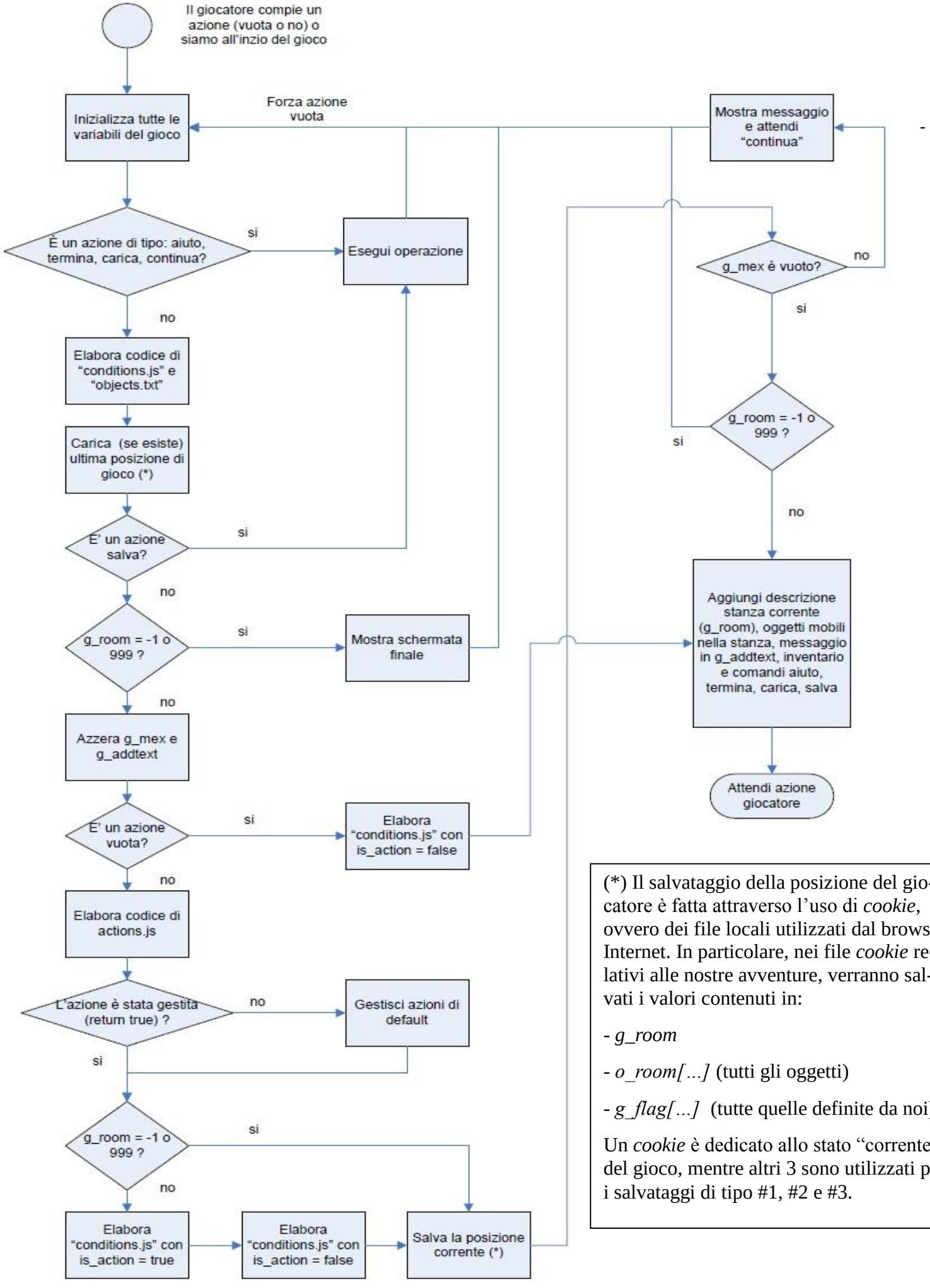
Riprendiamo il nostro file "conditions.js". Si compone essenzialmente di due parti. Una prima dentro ad una istruzione *if* e una seconda parte fuori dalla stessa.

Nel blocco di codice interno di *if (is_action)* vanno modificate le variabili *g_flag* per tener conto di azioni (non vuote) fatte dal giocatore, ovvero del fatto che il giocatore si è realmente "mosso" in termini temporali o spaziali nel gioco.

Ad esempio:

```
if (is_action)
{
        // qui dentro vanno aggiornati i g_flag
        // quando vengono compiute delle azioni

        // g_flag[0] incrementato ad ogni azione
        g_flag[0] = g_flag[0] + 1;

        return; // lasciare sempre questo return !!!
}
```

Questo pezzo di codice fa si che *g_flag[0]* sia incrementato ad ogni azione non vuota, senza porre nessuna condizione. In questo modo *g_flag[0]* 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" che il giocatore impiega in una stanza o per reagire a qualcosa. Se compare un mostro orribile, per esempio, potrebbe essere disponibile una sola azione (mossa) per salvarci!



(*) Il salvataggio della posizione del giocatore è fatta attraverso l'uso di *cookie*, ovvero dei file locali utilizzati dal browser Internet. In particolare, nei file *cookie* relativi alle nostre avventure, verranno salvati i valori contenuti in:

- *g_room*
- *o_room[…]* (tutti gli oggetti)
- *g_flag[…]* (tutte quelle definite da noi)

Un *cookie* è dedicato allo stato "corrente" del gioco, mentre altri 3 sono utilizzati per i salvataggi di tipo #1, #2 e #3.

Da notare il comando *return* : **va sempre lasciato come ultima operazione del corpo di *if (is_action)*.** *return* è il comando che fa "terminare" l'elaborazione del file di progetto nel quale ci troviamo.

Vediamo altri esempi. Se volessimo incrementare la variabile *g_flag[1]* solo se il giocatore si trova nella stanza 05 (ad esempio) e compie una mossa, il codice diventerebbe:

```
if (is_action)
{
        ...
        if (g_room == 05)
        {
                g_flag[1] = g_flag[1] + 1;
        }
        return;
}
```

Quindi, *g_flag[1]* si incrementa solo se siamo nella stanza 5 e non sempre, come era prima per *g_flag[0]*.

Per come è strutturato lo script di base, bisogna tener conto che anche in "conditions.js" le sole variabili su cui possiamo creare le nostre condizioni sono: *g_room*, *o_room[…]* e *g_flag[…]*.

Veniamo alla parte successiva a *if (is_action) { ... }*. Qui vanno invece inserite le condizioni che portano a modifiche allo stato dell'avventura. Le condizioni anche in questo caso si basano sulle variabili appena citate.

Vediamo di fare un po' di esempi per chiarire il tutto!

Ecco un primo esempio di condizione → modifica del gioco. Supponiamo che esista una stanza 05 la cui descrizione (in "rooms.txt") sia la seguente:

```
<05>
Sei dentro una grande stanza con una zampillante {fontana|_fontana05_}.
Puoi andare a {Sud|_sud05_}.
```

Vogliamo che dopo 50 azioni non vuote dall'inizio dell'avventura, la fontana presente nella descrizione della stanza 05 smetta di funzionare: vogliamo quindi cambiare la descrizione della stanza 05. Ecco come fare tramite una condizione in "conditions.js":

```
if (is_action)
{
```

```
        g_flag[0] = g_flag[0] + 1;


        return;
}


if ( g_flag[0] >= 50 ) // dopo 50 azioni non vuote
{
        g_desc[05] = g_desc[30]; // la descrizione della stanza 05 diventa quella della stanza 30
}
```

Il trucco qui è preparare una descrizione di una stanza (in "rooms.txt") che non è usata realmente nel gioco (la numero 30 ad esempio) e modificare la variabile dello script di base *g_desc[...]* che contiene le descrizioni di tutte le stanze. Quindi in "rooms.txt" avremo:

```
<05>
Sei dentro una grande stanza con una zampillante {fontana|_fontana05_}.
Puoi andare a {Sud|_sud05_}.
...
<30>
Sei dentro una grande stanza con una {fontana|_fontana05_} che non sgorga nulla.
Puoi andare a {Sud|_sud05_}.
```

C'è un modo ancora più dinamico per ridefinire la descrizione di un ambiente, assegnando direttamente il nuovo testo di *g_desc*. In questo caso però è necessario specificare eventuali oggetti fissi in un modo un po' particolare:

```
...
if ( g_flag[0] >= 50 )
{
        g_desc[05] =
                @"Sei dentro una grande stanza con una $c_fontana05_$fontana$ che non sgorga nulla. "@ +
                @"Puoi andare a $c_sud05_$sud$."@;


}
```

In pratica al posto di *{testo|identificatore oggetto}* visti in "rooms.txt" dovremo usare:

$c <identicatore oggetto> $ testo $

Questi tipi di modifica delle descrizioni degli ambiente può essere evitata utilizzando oggetti mobili (ad esempio potremmo rendere la fontana un oggetto mobile).

Si noti che la modifica di *g_desc[05]* non è permanente in quanto le uniche variabili che lo script di base "salva" internamente sono unicamente *g_room, g_flag[…] e o_room[…]*. Tutte le altre (comprese *g_desc*) vengono riportate ai valori di inizio avventura ad ogni azione compiuta dal giocatore (quindi *g_desc[05]* riassume la descrizione definita in "rooms.txt" per la stanza 05) . Tuttavia la condizione vista sopra è scritta in modo tale che da una certa mossa in poi, *g_desc[05]* sarà ogni volta (ovvero ad ogni azione vuota o no) "riscritta" con la descrizione voluta.

Altro esempio di condizione. Vogliamo che se il giocatore passa almeno una volta per la stanza 05, la variabile *g_flag[17]* venga messa a 1:

```
if (is_action)
{
        g_flag[0] = g_flag[0] + 1;

        return;
}

if ( g_flag[0] >= 50 )
{
        g_desc[05] = g_desc[30];
}

if ( g_room == 05 )
{
        g_flag[17] = 1;
}
```

In questo modo avremo "registrato" il passaggio del giocatore nella stanza 05 attraverso l'uso di una variabile *g_flag*. Da questo momento, in qualsiasi altra condizione, per sapere se siamo passati per la stanza 5 basterà chiedersi se *g_flag[17]* è uguale a 1.

Aggiungiamo ora che se il giocatore si trova nella stanza 10 (*g_room == 10*) ed è passato per la stanza 05 (*g_flag[17]==1*), verrà mostrato un messaggio (una volta sola):

```
…


if (( g_room == 10 ) && (g_flag[17] == 1) && (g_flag[18] == 0)) // sono nella stanza 10, sono passato per la
                                                                // stanza 05 ed è la prima volta che si
                                                                // verifica questa condizione
{
        g_flag[18] = 1;
        g_mex = g_mex + ' ' + @"Per un attimo vedi qualcosa muoversi nell'ombra..."@;
}
```

Il messaggio verrà mostrato un'unica volta in quanto *g_flag[18]* diviene 1 e la condizione non può più verificarsi.

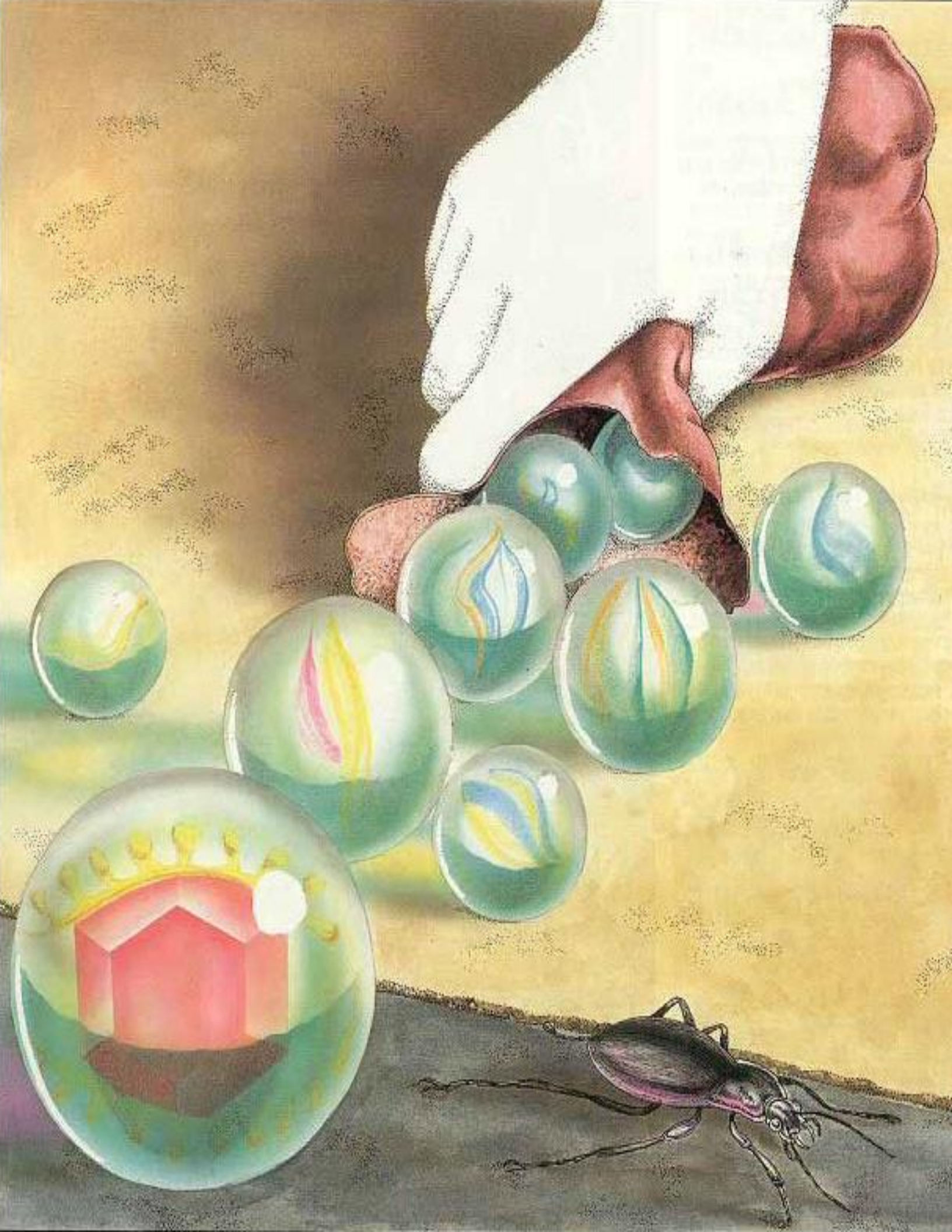
g_mex è una variabile speciale che permette di mostrare messaggi popup che "bloccano" il gioco finchè non si tocca "Continua". Per ogni azione compiuta può apparire un solo e unico messaggio di questo tipo prima che il gioco continui mostrando la descrizione della stanza in cui ci si trova.

In generale, i caratteri ammessi per *g_mex* (e in qualsiasi altra stringa utilizzata nei file ".js") sono lettere (maiuscole e minuscole), numeri, spazio e i caratteri:

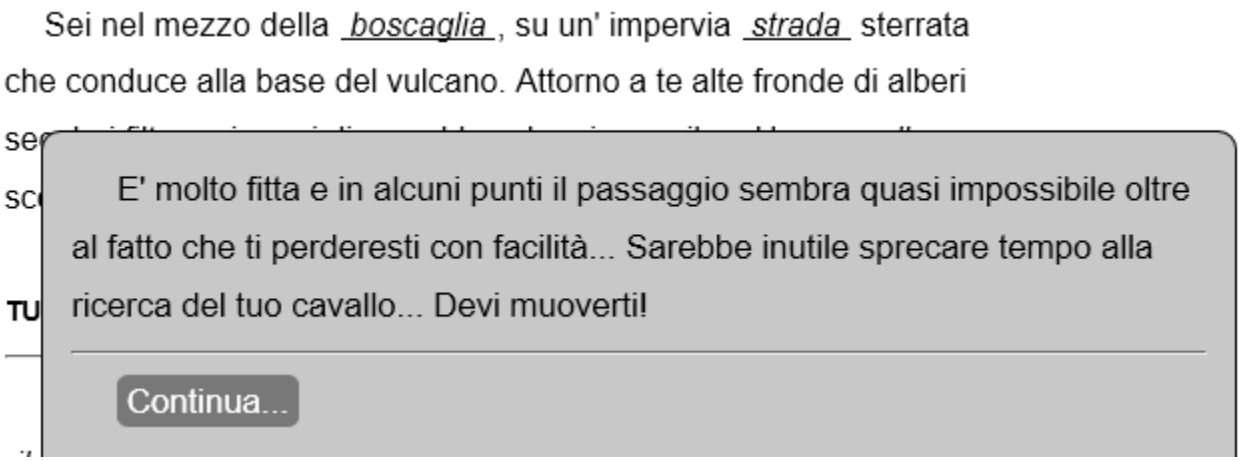
```
! £ % & ? , . + - : ; * / = [ ] ( ) < > { } |
```

Il carattere & anche in questo caso viene visualizzato e lo stesso vale per i caratteri < >. Quindi non si possono introdurre tag o caratteri in codifica HTML. Se necessitiamo delle doppie virgolette, bisogna utilizzare il carattere di escape \ per introdurle nel testo:

```
g_mex = g_mex + ' ' + @"Qualcuno grida \"AAAAAAHHHHH!!!!\" "@;
```



Ecco un esempio di messaggio popup ottenuto con *g_mex* da "Il Covo dei Trafficanti":



Poiché come detto dopo un azione può essere visualizzato un unico messaggio, e *g_mex* può essere usato da più condizioni e per le risposte alle azioni del giocatore, i nostri messaggi vanno sempre "concatenati" (sommati…) ai vecchi, mettendo uno spazio in mezzo:

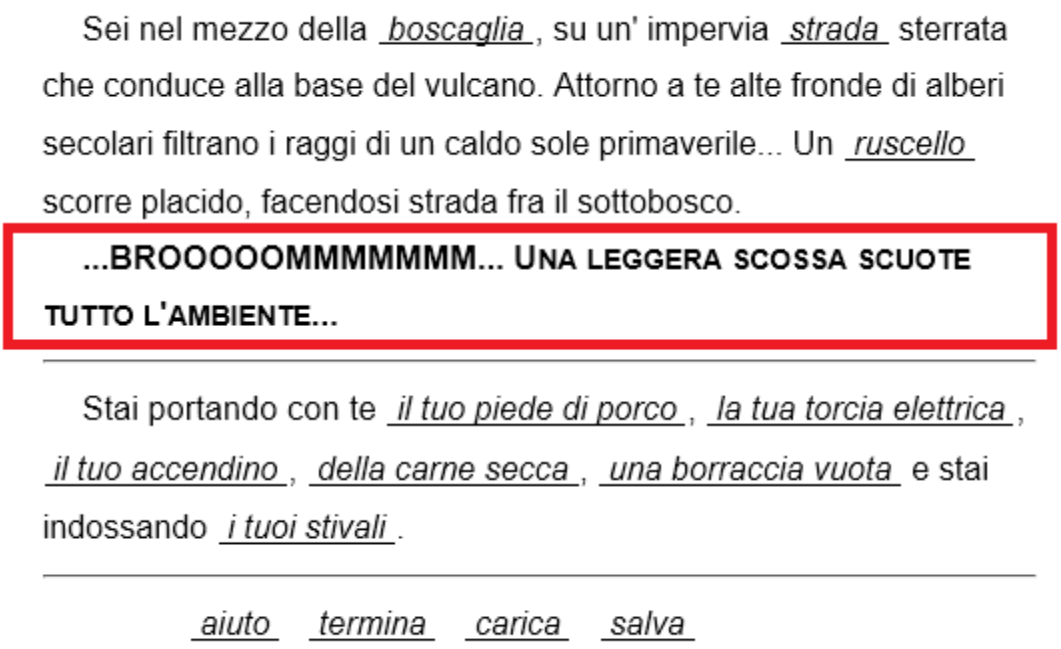
```
g_mex = g_mex + ' ' + <nuovo messaggio>
```

Si può anche aggiungere un "a capo" tra i messaggi tramite uso dei tag HTML tra virgolette singole. Per farlo, basta scrivere:

```
g_mex = g_mex  + '<br><br><p>'+ @"Per un attimo vedi qualcosa muoversi nell'ombra..."@ +'</p>';
```

Tuttavia, dato che l'avventura può essere giocata anche su reader con dimensioni di schermo limitate, gli "a capo" andrebbero limitati il più possibile.

Un'altra possibilità per comunicare al giocatore qualcosa è agire sulla variabile *g_addtext* per aggiungere un messaggio in coda alla descrizione della stanza. Questo messaggio sarà sempre in maiuscoletto grassetto, ed anche in questo caso potrebbe essere che più condizioni vanno ad aggiungere testi in esso. Ecco un esempio preso da "Il Covo dei trafficanti":



Facciamo un esempio di messaggi che si "sommano" in *g_addtext*. Nel codice seguente, le due condizioni potrebbero verificarsi contemporaneamente e il messaggio unico risultante sarebbe "Per un attimo vedi qualcosa muoversi nell'ombra... Senti un fischio in lontananza.".

```
…
if (( g_room == 10 ) && (g_flag[17] == 1) && (g_flag[18] == 0)) // sono nella stanza 10, sono passato per la
                                                                // stanza 05 ed è la prima volta che si
                                                                // verifica questa condizione
{
        g_flag[18] = 1;
        g_addtext = g_addtext + ' ' + @"Per un attimo vedi qualcosa muoversi nell'ombra..."@;
}

if ((g_flag[0] > 20) && (g_flag[0] < 30)) // per un po' di azioni sento un fischio ovunque il giocatore sia
{
        g_addtext = g_addtext + ' ' + @"Senti un fischio in lontananza."@;
}
```

Come si può notare nelle condizioni non viene mai usato il comando *return* in quanto interromperebbe la valutazione di tutte le condizioni in "conditions.js" e abbiamo visto che potrebbero esserci più condizioni che si verificano contemporaneamente. Tuttavia ci sono dei casi in cui possiamo utilizzarlo, e sono quelli nei quali il giocatore fallisce o vince e l'avventura termina. A quel punto, non ha senso valutare altre condizioni o "accumulare" altri messaggi.

Ad esempio, se non abbiamo il salvagente (preso o indossato) e siamo nella stanza 38 (mare aperto?):

```
...
if ((g_room == 38) && (o_room[$(@"_salvagente00_"@)] > -1))
{
        g_mex = g_mex + ' ' + @"Non riesci più a stare a galla… Ben presto muori affogato..."@;
        g_room = -1; // vai a schermata di fine gioco!
        return; // le condizioni dopo questa non mi interessano!!!
}
```

Assegnare come stanza attuale del giocatore il valore -1 significa andare alla schermata di "fallimento" del gioco (vedi "fail.html"). Al contrario, assegnare il valore 999 significa andare a quella di successo ("success.html") e terminare l'avventura.

Vediamo un esempio più complicato. Utilizzando *g_flag[0]*, incrementata ad ogni azione, e *g_flag[21]* (inizializzata a 0), vogliamo far scrivere una frase scelta a caso tra 3 (tramite la funzione javascript *f_getrandomint*) se siamo nella stanza 07. Inoltre la frase appare a intervalli casuali di 5-15 azioni e solo se *g_addtext* non contiene nessun altro messaggio. Ecco il codice:

```
if (is_action)
{
        g_flag[0] = g_flag[0] + 1;
        return:
}

...

if ((g_flag[0]>g_flag[21]) && (g_flag[21]>0) && (g_room==07))
{
  if (g_addtext=='')
```

```
  {
    g_flag[21] = g_flag[0] + f_getrandomint(5,10); // prossima azione da cui far apparire il messaggio
                                                   // = numero azione corrente + numero a caso da 5 a 10
    var x = f_getrandomint(1,3); // numero a caso da 1 a 3
    if (x==1)
      g_addtext =
        @"...BROOOOOMMMMMMM... Una leggera scossa scuote tutto l'ambiente..."@;
    else if (x==2)
      g_addtext =
        @"...BROOOOOMMMMMMM... Improvvisamente tutto trema per qualche secondo..."@;
    else
      g_addtext =
        @"...BROOOOOMMMMMMM... Per qualche lunghissimo istante tutta la grotta trema!!!"@;
  }
}
else if (g_flag[21]==0)  // la prima volta valorizzo g_flag[21] e basta
{
  g_flag[21] = g_flag[0] + f_getrandomint(5,15);
}
```

Ancora un altro esempio. Avevamo parlato della proprietà *disabled* per gli oggetti. La condizione di seguito rende l'oggetto *_pulsante03_* abilitato (che supponiamo definito in "objects.txt" con la proprietà *disabled*) dopo 50 azioni dall'inizio del gioco:

```
if (g_flag[0]>50)
{
       o_enabled[$(@"_pulsante03_"@)] = 1;
       g_flag[35] = 1;  // tengo conto che oggetto ora è enabled
}
```

Ovviamente con *o_enabled[...]* = 0 si renderebbe l'oggetto *disabled*.

Attenzione: non usare mai *o_enabled* nelle condizioni degli *if*. Come detto, solamente *g_room, o_room[…]* e *g_flag[…]* sono salvate nel corso del gioco e mantenute con i valori correnti. Questo è il motivo per il quale si immagina di utilizzare *g_flag[35]* per tener conto dello stato abilitato o no dell'oggetto.

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:

```
...
if ((f_getcurrentweight() > 50) && (g_room == 07))
{
	g_mex = g_mex + ' ' + @"Fai pochi passi e sprofondi nelle orribili sabbie mobili..."@;
	g_room = -1;
	return;
}
```

Dedichiamoci ora alle condizioni dell'avventura dell'Occhio Purpureo. Apriamo "conditions.js" e modifichiamolo come segue.

```
if (is_action)
{

	// CONDIZIONI CHE TENGONO CONTO DELLE AZIONI FATTE
	// ----------------------------------------------

	// g_flag[0] incrementato ad ogni azione
	g_flag[0] = g_flag[0] + 1;

	// se sono nella foresta, incrementa g_flag[3]
	if (g_room == 02)
	{
		g_flag[3] = g_flag[3] + 1;
	}
	else // altrimenti mettilo a 0, in modo da ricominciare il conto delle azioni
```

```
            // nella foresta da zero se ci si ritorna
       {
              g_flag[3] = 0;
       }

       // se è comparso l'ispettore, incrementa g_flag[6]
       if (g_flag[2] == 1)
       {
              g_flag[6] = g_flag[6] + 1;
       }

       return;
}

// CONDIZIONI CHE NON MODIFICANO LO STATO DEL GIOCO
// -----------------------------------------------

// se abbiamo (o è presente) la lampada accesa nella stanza buia, cambia la descrizione della stanza
if ( ( (o_room[$(@"_lampadaAccesa00_"@)] == -1) ||  // lampada in inventario
       ((o_room[$(@"_lampadaAccesa00_"@)] == g_room))  // lampada nella stanza in cui siamo
     )
        && (g_room == 09)
   )
{
       g_desc[09] =
              @"Sei in una stanza buia.
              Per tua fortuna la lampada illumina due uscite ad $c_ovest09_$Ovest$ e
              a $c_sud09_$Sud$."@;
}

// se sono nella foresta per una o due azioni, scrivi una frase aggiuntiva
if (g_room == 02)
{
```

```
        if (g_flag[3]==1)
        {
                g_addtext = g_addtext + ' ' +
                        @"Questa foresta ha qualcosa di inquietante... Ti senti strano..."@;
        }
        else if (g_flag[3]==2)
        {
                g_addtext = g_addtext + ' ' +
                        @"Senti uno strano formicolio lungo il corpo... Il tuo istinto ti dice di andartene..."@;
        }
}



// CONDIZIONI CHE MODIFICANO LO STATO DEL GIOCO
// ------------------------------------------

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

// se sono nella foresta, alla terza azione fai morire il giocatore
if ((g_room == 02) && (g_flag[3]==3))
{
        g_mex = g_mex + ' ' +
                @"Improvvisamente ti rendi conto di non poterti più muovere...
                Ti stai pietrificando come gli alberi!!! Che fine terribile..."@;
        g_flag[4] = 37; // si muore pietrificati (vedi messages.txt)
        g_room = -1;
        return; // inutile proseguire in "conditions.js", il giocatore è morto...
}
```

```
// l'ispettore è comparso e abbiamo fatto una azione non vuota (g_flag[6] è 1)
// e non è stato ucciso (altrimenti g_flag[2] sarebbe 2)
if ((o_room[$(@"_ispettore00_"@)] != 0) && (g_flag[2] == 1) && (g_flag[6] == 1))
{
	// se il giocatore ha almeno un oggetto, l'ispettore ne prende uno
	// a caso e se ne va
	var l_contaOggettiTrasportati = 0;
	var l_totaleOggettiNelGioco = o_room.length; // o_room ha tanti elementi quanti sono gli oggetti
	var l_oggettoScelto = 0; // oggetto scelto dall'ispettore
	var l_punteggioRandom = 0; // punteggio random per decidere quale oggetto

	// ciclo con i che va da g_defaultobjnum (indice del primo oggetto non default)
	// al numero totale di oggetti nel gioco
	for (var i = g_defaultobjnum; i < l_totaleOggettiNelGioco; i++)
		if (o_room[i]<0) // portato o indossato
		{
			l_contaOggettiTrasportati = l_contaOggettiTrasportati + 1;
			var x = f_getrandomint(1,10); // per ogni oggetto genero un numero a caso da 1 a 10
			if (x > l punteggioRandom) // se è maggiore del massimo ottenuto finora, l'oggetto scelto
			{                          // diventa questo
				l_punteggioRandom = x;
				l_oggettoScelto = i;
			}
		}

	// se il giocatore ha almeno un oggetto
	if (l_contaOggettiTrasportati > 0)
	{
		o_room[$(@"_ispettore00_"@)] = 0; // l'ispettore scompare
		o_room[l_oggettoScelto] = 0; // l'oggetto scompare
		g_mex = g_mex + ' ' +
			@"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..."@;
        }
        else // altrimenti finisce in gattabuia!
        {
                g_room = -1;
                g_flag[4] = 38; // si finisce in gattabuia (vedi messages.txt)
                g_mex = g_mex + ' ' +
                        @"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..."@;
                return; // inutile proseguire in "conditions.js", il giocatore è morto...
        }

}

// l'ispettore compare dopo 20 azioni dall'inizio dell'avventura
// se non sono nella stanza 02 o 09 e se non è mai apparso.
if (
        (g_flag[2] == 0)  // ispettore non ancora apparso
        && (g_flag[0] >= 20) // giocatore ha fatto almeno 20 azioni
        && (g_room != 02) && (g_room != 09) // non sono nelle stanze 02 e 09
   )
{
        g_flag[2] = 1; // non comparirà più
        o_room[$(@"_ispettore00_"@)] = g_room; // compare nella stanza
        g_mex = g_mex + ' ' + @"E' comparso l'ispettore delle tasse!"@;
}
```

La prima parte del file è quella relativa a *if (is_action)*: vengono incrementate alcune delle variabili *g_flag* in base a determinate condizioni, come descritto nei commenti. Ad esempio tutte le volte che siamo nella foresta, "parte" un contatore di mosse (*g_flag[3]*) che viene rimesso a 0 non appena andiamo in un'altra stanza.

Nella seconda parte, vi sono delle condizioni per modificare la descrizione della stanza buia se possediamo la lampada accesa e per aggiungere alcune frasi di avvertimento tramite *g_addtext* se siamo nella foresta.

La terza parte è quella relativa a condizioni che modificano davvero lo stato dell'avventura:

- viene deciso il numero di volte che la catena va tirata (una volta per ogni nuova partita). Poiché *g_flag[1]* all'inizio è 0, la prima volta la condizione è verificata e *g_flag[1]* viene valorizzata con un valore casuale da 1 a 3. Da questo momento in poi quindi la condizione non potrà più essere verificata perché *g_flag[1]* non è più uguale a 0
- viene ucciso il giocatore se rimane per più di un'azione nella foresta. Da notare l'uso di *g_flag[4]* per "forzare" il messaggio della schermata di fallimento (vedremo come funziona)
- se siamo in presenza dell'ispettore e abbiamo fatto un'azione (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. Tramite un ciclo *for* vengono passati in rassegna tutti gli oggetti dell'avventura e scelto tramite un punteggio random quale di quelli trasportati prelevare (mettendo il suo indice – cioè la sua posizione nell'array *o_room* – in *l_oggettoScelto*). Questo codice è un buon esempio per eventualmente implementare dinamiche simili nelle proprie avventure, scegliendo oggetti in base ai criteri voluti. La variabile *g_defaultobjnum* dello script di base contiene l'indice del primo oggetto definito dall'utente
- Viene fatto comparire l'ispettore dopo 20 azioni dall'inizio dell'avventura. L'ispettore rimane nella stanza del giocatore per solamente un'azione (il numero di azioni in presenza dell'ispettore viene contato da *g_flag[6]*)

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

Uno degli aspetti più importanti da capire nello scrivere le condizioni è che mentre "conditions.js" con *is_action = true* viene valutato una ed un'unica volta a seguito di un'azione (non vuota) , "conditions.js" con *is_action = false* può essere richiamato più volte, ad esempio perché il giocatore salva la propria posizione o viene fatto un refresh della pagina web.

In altri termini, non dobbiamo mai ragionare come se a seguito di una azione (non vuota) la sequenza sia:

- viene valutato *actions.js*
- viene valutato *conditions.js* con *is_action = true*
- viene valutato *conditions.js* con *is_action = false*
- viene mostrato il messaggio e poi la schermata della stanza

ma che potrebbe essere del tipo:

- viene valutato *actions.js*
- viene valutato *conditions.js* con *is_action = true*

- viene valutato *conditions.js* con *is_action = false*
- viene valutato *conditions.js* con *is_action = false*
- viene valutato *conditions.js* con *is_action = false*
- …
- viene valutato *conditions.js* con *is_action = false*
- viene mostrato il messaggio e poi la schermata della stanza

Di conseguenza, nel caso di condizioni che "mandano avanti" l'avventura (cioè ne modificano lo stato, ovvero le variabili *g_flag[], g_room, o_room[]*) bisogna fare in modo che siano verificate solo per azioni "nuove" ed un'unica volta. Per farlo, bisogna utilizzare la variabili *g_flag* che vengono incrementate in *if (is_action) { ... }*

Quindi in definitiva potremo scrivere quattro tipi di condizioni in "conditions.js":

1) quelle interne a *if (is_action) { ... }*: aggiornano le variabili *g_flag* per tener conto delle azioni (non vuote) fatte in determinate circostanze
2) quelle esterne a *if (is_action) { ... }* che saranno di tre tipi:
   a. Quelle che scrivono solamente del testo aggiuntivo (*g_addtext*) e **non** modificano lo stato del gioco (quindi non modificano variabili *g_flag[], g_room, o_room[]*, etc).
   b. Quelle che modificano solamente la descrizione di stanze (*g_desc[]*) e **non** modificano lo stato del gioco (quindi non modificano variabili *g_flag[], g_room, o_room[]*, etc)
   c. Quelle che modificano lo stato del gioco (quindi modificano variabili *g_flag[], g_room, o_room[]*, etc)

Come detto, le condizioni che dovremo scrivere in modo che siano verificate una ed un'unica volta sono le ultime (2c). Il modo per farlo è sempre attraverso le variabili *g_flag* che devono aiutarci a capire se la condizione è stata già eseguita o no. Inoltre, non dovremo mai usare funzioni "random" come *f_getrandomint* per determinare se la condizione è avvenuta o no.

Ad esempio, la condizione vista sopra:

```
...


// l'ispettore compare dopo 20 azioni dall'inizio dell'avventura
// se non sono nella stanza 02 o 09 e se non è mai apparso.
if (
        (g_flag[2] == 0)  // ispettore non ancora apparso
        && (g_flag[0] >= 20) // giocatore ha fatto almeno 20 azioni
        && (g_room != 02) && (g_room != 09) // non sono nelle stanze 02 e 09
   )
{
        g_flag[2] = 1; // non comparirà più
        o_room[$(@"_ispettore00_"@)] = g_room; // compare nella stanza
}
```

è resa "eseguibile una volta sola" dalla condizione *(g_flag[2] == 0)* e dal successivo assegnamento *g_flag[2] = 1*;

Se volessimo però che il numero di azioni non sia 20 ma un numero casuale, potremmo utlizzare una variabile *g_flag* (esattamente come per il numero di volte che bisogna tirare la catena) per generare all'inizio dell'avventura un numero di azioni casuali (ad esempio da 20 a 30) e usarla per decidere quando far apparire l'ispettore:

```
// valorizziamo una volta sola g_flag[7] con un numero casuale da 20 a 30
if (g_flag[7]==0)
{
        g_flag[7] = f_getrandomint(20,30);
}


if (
        (g_flag[2] == 0)  // ispettore non ancora apparso
        && (g_flag[0] >= g_flag[7]) // giocatore ha fatto almeno g_flag[7] azioni
        && (g_room != 02) && (g_room != 09) // non sono nelle stanze 02 e 09
   )
{
        g_flag[2] = 1; // non comparirà più
        o_room[$(@"_ispettore00_"@)] = g_room; // compare nella stanza
        g_mex = g_mex + ' ' + @"E' comparso l'ispettore delle tasse!"@;
}
```

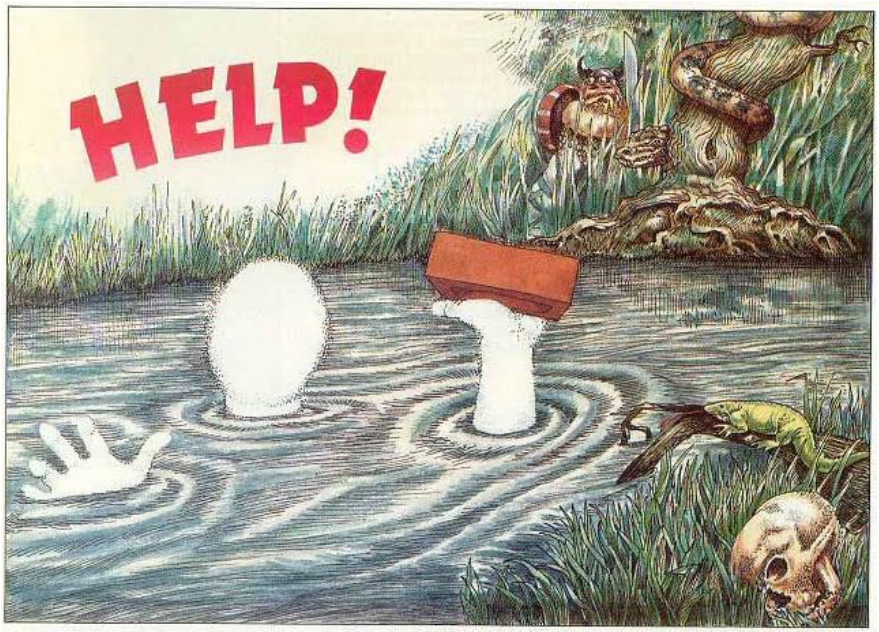
Nel caso in cui avessimo bisogno invece di scrivere una condizione che avvenga per più azioni successive, possiamo utilizzare il contatore di azioni *g_flag[0]* (che si incrementa ad ogni azione) per capire se stiamo rieseguendo la condizione per la stessa azione oppure no. Basta salvare il suo valore in un'altra variabile *g_flag* che "registra" l'ultima azione eseguita.

Ad esempio immaginiamo di inizializzare *g_flag[30]* a zero e di usarlo per tener traccia delle azioni già valutate o no. Ecco come fare:

```
if (g_flag[30] != g_flag[0])

{

  g_flag[30] = g_flag[0]; // potrò rientrare qui solo se il contatore di azioni

                          // viene incrementato

  g_mex = g_mex + ' ' + @"Nuova azione!"@;

}
```

All'inizio *g_flag[0]* e *g_flag[30]* sono entrambe = 0, quindi non si entra nella condizione. Alla prima azione non vuota del giocatore, *g_flag[0]* diventa uguale a 1, quindi la condizione si verifica (*0 != 1*) e *g_flag[30]* viene messo anch'esso a 1. Verrebbe scritto il messaggio "Nuova azione!".

Se ora immaginiamo che avvenga un refresh della pagina web e venga valutato nuovamente "conditions.js", la condizione non è più valida perché le due variabili hanno lo stesso valore (1). Solo ad una nuova azione non vuota *g_flag[0]* diventerà uguale a 2 e la condizione potrà essere vera.



Il file di progetto actions.js

Siamo arrivati all'ultimo file di progetto da gestire. In questo file vanno intercettate tutte le azioni fatte dal giocatore e che non sono gestite automaticamente dallo script di base (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 "manualmente" e non lasciar fare allo script di base. Ad esempio, se un'oggetto ha la proprietà *usable*, è molto probabile che verrà usato in combinazione con oggetti con i quali vogliamo non accada nulla. In questo caso ci penserà lo script di base a rispondere con un messaggio "Non sembra funzionare!" o simili.

Vediamo quindi come catturare un'azione. Dovremo creare delle condizioni (*if* ...) basate sulle seguenti variabili dello script di base:

- *g_aid1* : contiene l'identificatore dell'oggetto sul quale si è attivato il menu di azioni (esempio: *_candela03_*)
- *g_verb* : contiene il testo dell'azione (esempio: *vai*)
- *g_aid2* : contiene l'identificatore dell'oggetto scelto nel sotto-menu dell'azione oppure il testo del sotto-menu scelto (nel caso in cui si sia aggiunto un testo generico). Questa variabile può essere *null*. (esempio: *_fuoco04_* oppure *armadio*)
- *g_room* : stanza attuale nel quale si trova il giocatore
- *g_flag[..]* : variabili di stato del gioco
- *o_room[..]* : posizione degli oggetti nel gioco (-2 = indossato, -1 = preso, 0 = non presente, >0 nella relativa stanza)

E' fondamentale che per gli identificatori di oggetti e i testi dei sotto-menu si utilizzino minuscole e maiuscole esattamente uguali a come sono stati introdotti in "menu.js" o l'azione potrebbe non essere riconosciuta.

Nessun altra variabile dovrebbe essere utilizzata per le nostre condizioni in quanto, come già detto, le altre variabili vengono riportate al loro valore originario ad ogni azione.

Un'eccezione è rappresentata dalla funzione *f_getcurrentweight()* che ci dà il peso attuale trasportato e può essere usata per creare condizioni basate sullo stato attuale del nostro ingombro.

C'è un'altra funzione molto comoda per catturare delle azioni, ed è la funzione dello script di base *f_ais*. Essa permette in un colpo solo di gestire *g_aid1*, *g_verb* e *g_aid2*. Ecco subito un esempio in cui catturiamo l'azione "ruscello – usa con – borraccia" in "actions.js":

```
…
if (f_ais(@"_ruscello14_"@, g_use_ , @"_borraccia00_"@ ))
{
  if (g_flag[8]==0)
  {
    g_mex =
     @"Hai riempito la borraccia."@;
    g_flag[8] = 1;
  }
  else
  {
    g_mex =
     @"E' già piena!"@;
  }

  return true;
}
```

Analizziamo il codice. La funzione *f_ais* accetta tre parametri che sono l'oggetto "cliccato", l'azione e l'eventuale oggetto (o testo) del sotto-menu dell'azione. Per l'azione è stata usata una delle variabili "pronte" con le azioni base già visti presenti in "messages.txt", che riportiamo qui per comodità:

g_use_ = messaggio 18

g_examine_ = messaggio 19

g_take_ = messaggio 20

g_leave_ = messaggio 21

g_wear_ = messaggio 22

g_takeoff_ = messaggio 23

g_confirm_ = messaggio 24

g_go_ = messaggio 25

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 preparata la variabile *g_mex* per visualizzare un messaggio in base allo stato della borraccia, mantenuto nella variabile *g_flag[18]*.

E' importante notare che in questo caso il messaggio non viene aggiunto a *g_mex* (*g_mex= g_mex+...*) ma direttamente assegnato (*g_mex = ...)*. Il motivo è che "actions.js" è il primo dei file ad essere valutato e quindi, prima della sua esecuzione, *g_mex* è sicuramente vuota.

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 dallo script di base (che risponderebbe coi messaggi standard)**.

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

Ecco invece un esempio più complesso. Catturiamo un'azione "accendino – usa con – torcia" possibile però solo nelle stanza 05 o 06, se abbiamo indossato un elmetto e se la variabile *g_flag[11]* è diversa da 1 (qualunque cosa questo voglia dire…) :

```
if (    f_ais(@"_accendino10_"@, g_use_ , @"_torcia00_"@ )
        && ( (g_room == 05) || (g_room == 06) )
        && ( o_room[$(@"_elemetto00_"@)] == -2 )
        && ( g_flag[11] != 1 )
   )
{
        ... // qui faremo qualcosa...
}
```

In generale la funzione *f_ais* può essere usata anche per catturare azioni senza il secondo oggetto. Ad esempio:

```
if (    f_ais(@"_borraccia00_"@, @"bevi"@, null )
{
        ...
}
```

Questa condizione cattura l'azione "borraccia – bevi".

Per la maggior parte dei casi useremo *f_ais*, ma se vogliamo effettuare delle condizioni particolari per catturare tutta una certa gamma di azioni, possiamo usare direttamente in modo appropriato *g_aid1*, *g_verb* e *g_aid2*.

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

```
if (g_aid1 == @"_pistola00_"@)
{
        ...
}
```

Poi si può pensare di gestire le varie azioni su di essa all'interno del corpo della condizione:

```
if (g_aid1 == @"_pistola00_"@)
{
        if (verb == @"spara"@)
        {
                ...
                return true;
        }
        else if (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_ais(@"_porta08_"@,@"entra"@,null))
{
        if (o_room[$(@"_medaglione11_"@)]==-2) //  se sto indossando il medaglione...
        {
                g_mex = @"Un fascio di luce colpisce il medaglione... Riesci a passare incolume."@;
                g_room = 09; // sposta il giocatore nell'ambiente 09
                return true;
        }
        else
        {
                g_mex = @"Un fascio di luce ti colpisce in pieno petto pietrificandoti..."@
                g_room = -1; // fine gioco
                return true;
        }
}
```

Il corpo dell'*if* appena visto è interessante perchè introduce una delle possibili cose che possiamo far capitare a seguito di un'azione, cioè far visualizzare un messaggio popup tramite *g_mex*. Ecco un elenco di altre operazioni che tipicamente scriveremo quando catturiamo un’azione.

Valorizzare una variabile di stato g_flag: questo viene fatto per tener conto che l'azione è stata fatta e poter ad esempio creare esiti diversi la seconda volta che il giocatore la ripete. Ad esempio:

```
if (f_ais('_ruscello14_',g_use_,'_borraccia00_'))
{
  if (g_flag[8]==0)
  {
    g_mex =
     @"Hai riempito la borraccia."@;
    g_flag[8] = 1;
  }
  else
    g_mex =
```

```
    @"E' già piena!"@;
  return true;
}

if (f_ais('_borraccia00_',g_examine_,null))
{
  if (g_flag[8]==1)
    g_mex =
     @"E' piena d'acqua!"@;
  else
    g_mex =
     @"E' vuota!"@;
  return true;
}
```

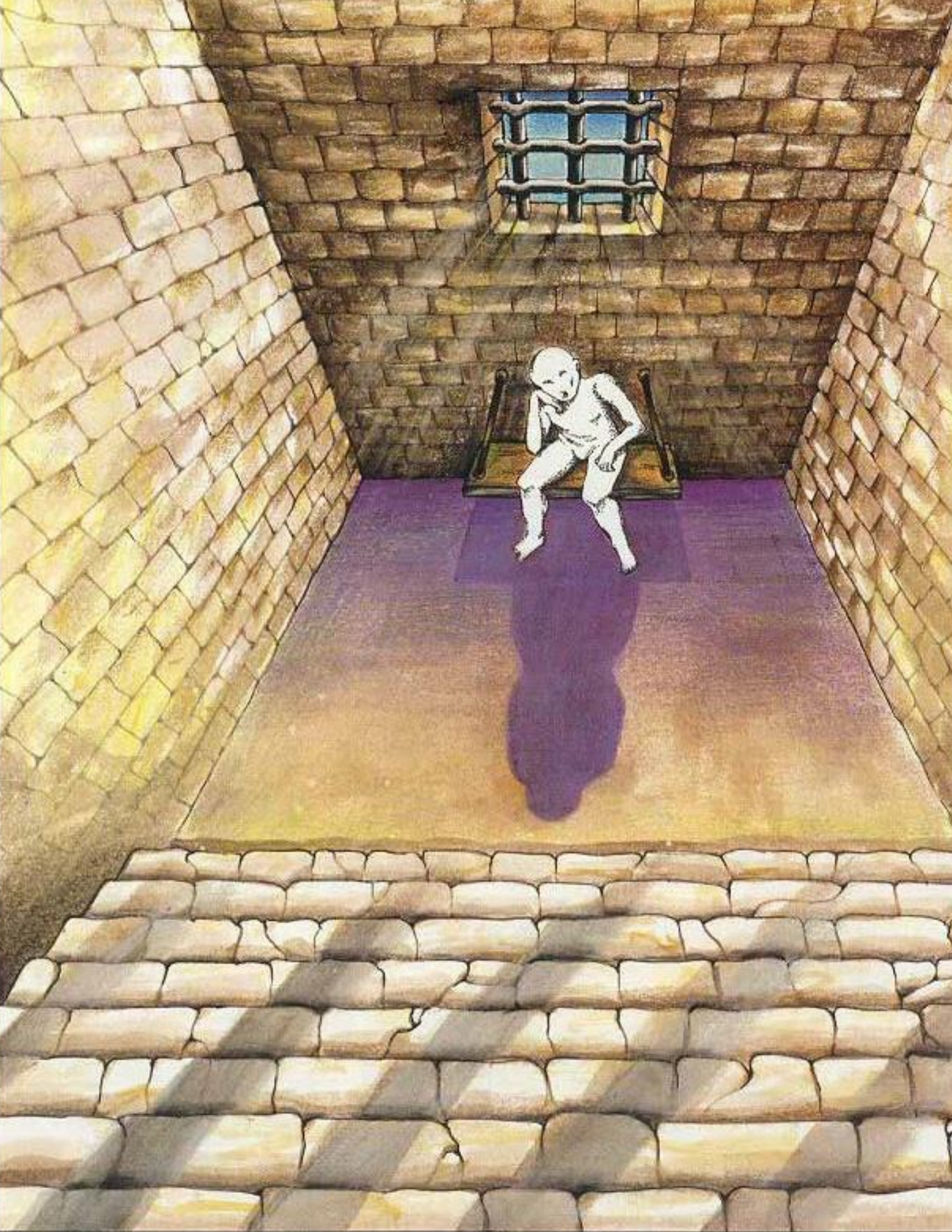
g_flag[8] tiene conto se la borraccia è piena o vuota e determina cosa succede esaminandola o ad ogni azione "ruscello – usa con – borraccia". Poi vi sarà un azione “borraccia – bevi” che farà tornare *g_flag[8]* a 0.

Spostare il giocatore in una nuova stanza: questo si ottiene semplicemente assegnando il numero di stanza alla variabile *g_room*, come nell'esempio precedente. Con questa variabile possiamo anche far andare il gioco alla schermata di vittoria (assegnando il valore 999) o di fallimento (assegnando il valore -1) e quindi far terminare il gioco.

```
if (f_ais('_strada14_',@"sali"@,null))
{
  g_room = 17; // il giocatore viene spostato nella stanza 17
  return true;
}
```

Come si vede non è obbligatorio mostrare un messaggio, a seguito dell'azione il gioco mostrerà direttamente una nuova schermata con la descrizione della stanza 17.

Spostare oggetti: questo significa sia far cambiare di locazione un oggetto che farlo proprio sparire dal gioco. Il modo per farlo è assegnare la proprietà *o_room* dell'oggetto.



Ad esempio se a seguito di una certa azione vogliamo far uscire l'oggetto *_spada00_* dal gioco (in modo che non sia presente in nessuna stanza):

```
        o_room[$(@"_spada00_"@)] = 0;
```

Per farlo apparire nella stanza corrente dove si trova il giocatore:

```
        o_room[$(@"_spada00_"@)] = g_room;
```

Per farlo apparire nella stanza 18 (ad esempio):

```
        o_room[$(@"_spada00_"@)] = 18;
```

L'aspetto più interessante è quello della gestione di oggetti "gemelli" (visti nel capitolo dedicato a "objects.txt") che rappresentato il diverso stato di uno stesso oggetto. Ad esempio, immaginiamo di aver definito due oggetti mobili gemelli in "objects.txt":

```
<_torciaaccesa00_>
desc_short: torcia
desc_long: una torcia accesa
desc_examine: E' accesa.
room: 00
weight: 5
examinable takeable usable

<_torciaspenta05_>
desc_short: torcia
desc_long: una torcia spenta
desc_examine: E' spenta.
room: 05
weight: 5
examinable takeable usable
```

All'inizio il giocatore troverà la torcia spenta nella stanza 05.

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

```
if (f_ais(@"_torciaspenta05_"@,@"accendi"@,null) &&
     (o_room[$(@"_accendino00_"@]==-1)) // il giocatore ha l'accendino!
{
       g_mes = @"Hai acceso la torcia!"@;
                                          // facciamo "comparire" la torcia accesa
                                          // nella stessa locazione della torcia spenta
       o_room[$(@"_torciaaccesa00_"@] = o_room[$(@"_torciaspenta05_"@] ;

       o_room[$(@"_torciaspenta05_"@] = 0; // facciamo "scomparire" la torcia spenta...

       return true;
}

if (f_ais(@"_torciaaccesa00_"@,@"spegni"@,null))
{
       g_mex = @"Hai spento la torcia!"@;
                                          // facciamo "comparire" la torcia spenta
                                          // nella stessa locazione della torcia accesa
       o_room[$(@"_torciaspenta05_"@]  = o_room[$(@"_torciaaccesa00_"@];

       o_room[$(@"_torciaaccesa00_"@] = 0; // facciamo "scomparire" la torcia accesa...

       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.

Se alcune azioni sono comuni a tutti gli oggetti gemelli, vanno gestite con delle condizioni ripetute, ad esempio immaginiamo di essere in una stanza (35) davanti ad un baratro e il giocatore sceglie di lanciare la torcia. Potremmo avere la torcia accesa o la torcia spenta e quindi vanno scritte due condizioni:

```
if (f_ais(@"_torciaspenta05_"@,@"lancia"@,null) && (g_room==35)) // siamo dinanzi ad un baratro oscuro
{
	g_mes = @"Hai gettato via la torcia nel baratro!"@;
	o_room[$(@"_torciaspenta05_"@)] = 0; // facciamo "scomparire" la torcia
	return true;
}
if (f_ais(@"_torciaaccesa00_"@,@"lancia"@,null) && (g_room==35)) // siamo dinanzi ad un baratro oscuro
{
	g_mes = @"Hai gettato via la torcia nel baratro!"@;
	o_room[$(@"_torciaaccesa00_"@)] = 0; // facciamo "scomparire" la torcia
	return true;
}
```

E' sempre possibile spostare un oggetto qualsiasi nell'inventario del giocatore, ponendo *o_room[...]* uguale a -1. Tuttavia tipicamente è meglio farlo apparire solamente nella stanza, avvertendo il giocatore con un messaggio, in modo che poi possa essere gestito automaticamente tramite proprietà *takeable*.

Qualora avessimo bisogno di richiamare un messaggio di "messages.txt" possiamo utilizzare la funzone *f_gmes(...)* che recupera per noi il testo del messaggio di un certo indice. Ad esempio:

```
	g_mex = f_gmes(33);
```

mostrerà un messaggio di tipo "Non funziona!" (scelto a caso tra quelli disponibili).

E' arrivato il momento di modificare il file "action.js" per l'avventura dell'Occhio Purpureo. Apriamo il file e modifichiamolo in questo modo:

```
// se nuotiamo nel fiume
if (f_ais(@"_fiume01_"@,@"nuota"@,null))
{
	if (g_flag[5]==0) // non abbiamo mai nuotato
	{
		g_flag[5] = 1; // per sapere che abbiamo già nuotato
		if (o_room[$(@"_mattone03_"@)]==-1) // abbiamo il mattone con noi
```

```
		{
			g_mex = @"Oh no!!! Sei troppo pesante e vieni risucchiato in un gorgo..."@;
			g_room = -1; // vai alla schermata di fallimento...
			g_flag[4] = 35; // si muore affogati (vedi messages.txt)
		}
		else
		{
			g_mex = @"Raggiungi la riva opposta dove trovi una pistola...
				A fatica la riporti dall'altra parte asciutta. Ora è ai tuoi piedi."@;
			o_room[$(@"_pistolaCarica00_"@)] = g_room; // compare la pistola nella stanza
		}
	}
	else
	{
		g_mex = @"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_ais(@"_edificio05_"@,g_examine_,null))
{
	g_mex = @"Sulla parete c'è una scritta scolorita: << ZUCAR "@ + // zucar in triestino significa tirare
		' ' + g_flag[1] + ' ';
	if (g_flag[1] == 1)
		g_mex = g_mex + @"VOLTA ! >> ..."@; // zucar 1 volta
	else
		g_mex = g_mex + @"VOLTE ! >> ..."@; // zucar 2 o 3 volte
	return true;
}

// se ci si siede sul trono appare un laconico messaggio
```

```
if (f_ais(@"_trono12_"@,@"siedi"@,null))
{
        g_mex = @"Ti senti un re! Però è scomodo."@;
        return true;
}

// accendi lampada
if (f_ais(@"_lampadaSpenta00_"@,@"accendi"@,null))
{
        g_mex = @"Hai acceso la lampada!"@;
        // sostituiamo la lampada accesa a quella spenta (che scompare)
        o_room[$(@"_lampadaAccesa00_"@)] = o_room[$(@"_lampadaSpenta00_"@)];
        o_room[$(@"_lampadaSpenta00_"@)] = 0;
        return true;
}
// spegni lampada
if (f_ais(@"_lampadaAccesa00_"@,@"spegni"@,null))
{
        g_mex = @"Hai spento la lampada!"@;
        // sostituiamo la lampada accesa a quella spenta (che scompare)
        o_room[$(@"_lampadaSpenta00_"@)] = o_room[$(@"_lampadaAccesa00_"@)];
        o_room[$(@"_lampadaAccesa00_"@)] = 0;
        return true;
}
// usa lampada con mattone -> la lampada si rompe
if (f_ais(@"_lampadaAccesa00_"@,g_use_,@"_mattone03_"@)
          || f_ais(@"_lampadaSpenta00_"@,g_use_,@"_mattone03_"@)
   )
{
        g_mex = @"Hai rotto la lampada!"@;

        // sostituiamo la lampada rotta a quella accesa o spenta (che scompaiono)
        var x = o_room[$(@"_lampadaSpenta00_"@)]; // locazione lampada spenta
```

```
	// se la lampada spenta non è presente, allora metto in x la locazione della lampada accesa
	if (x==0)
		x = o_room[$(@"_lampadaAccesa00_"@)];
	o_room[$(@"_lampadaRotta00_"@)] = x;
	o_room[$(@"_lampadaSpenta00_"@)] = 0;
	o_room[$(@"_lampadaAccesa00_"@)] = 0;
	return true;
}

// se si vuota il sacchetto compaiono le biglie
if (f_ais(@"_sacchettoPieno05_"@,@"vuota"@,null))
{
	// sostituiamo il sacchetto vuoto a quello pieno
	o_room[$(@"_sacchettoVuoto00_"@)] = o_room[$(@"_sacchettoPieno05_"@)];
	o_room[$(@"_sacchettoPieno05_"@)] = 0;
	// facciamo comparire le biglie nella stanza in cui siamo
	o_room[$(@"_biglie00_"@)] = g_room;

	g_mex = @"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 ((g_aid1 == @"_catena12_"@) && (g_verb == @"tira"@))
{
	var x = 0;
	if (g_aid2 == @"una volta"@)
		x = 1;
	else if (g_aid2 == @"due volte"@)
		x = 2;
	else if (g_aid2 == @"tre volte"@)
```

```
                x = 3;
        if ((x == g_flag[1]) // il numero di volte che tiriamo la catena è giusta
                        && (o_room[$(@"_occhio00_"@)]==-1)) // e abbiamo l'occhio
        {
                        if (o_room[$(@"_guanti11_"@)]==-2) // se indossiamo i guanti
                        {
                                g_mex = @"...Un passaggio segreto si apre verso un sentiero fiorito..."@;
                                g_room = 999; // vai alla schermata di successo!
                        }
                        else
                        {
                                g_mex = @"BZZZZZZZZZZZZ!!! Una scarica ti fulmina all'istante..."@;
                                g_room = -1; // vai alla schermata di fallimento!
                                g_flag[4] = 36; // si muore fulminati (vedi messages.txt)
                        }
        }
        else
        {
                g_mex = @"AAAARGH!!! Il pavimento si apre e dall'alto una cascata d'acqua ti trascina via!..."@;
                g_room = -1; // vai alla schermata di fallimento!
                g_flag[4] = 35; // si muore affogati (vedi messages.txt)

        }
        return true;
}

// spariamo: l'esito dipende se l'ispettore è nella stanza o no
// oppure usiamo la pistola carica con l'ispettore
if (f_ais(@"_pistolaCarica00_"@,@"spara"@,null) // pistola carica - spara
                || f_ais(@"_pistolaCarica00_"@,g_use_,@"_ispettore00_"@) // pistola carica - usa con... - ispettore
                || f_ais(@"_ispettore00_"@,@"spara"@,null)) // ispettore - spara
{
        // se l'ispettore è nella nostra stanza
```

```
        if (o_room[$(@"_ispettore00_"@)] == g_room)
        {
                g_mex = @"Hai ucciso l'ispettore delle tasse!!!"@;
                o_room[$(@"_ispettore00_"@)] = 0; // facciamolo scomparire dal gioco
                o_room[$(@"_cadavere00_"@)] = g_room; // compare il suo cadavere
                g_flag[2] = 2; // ispettore è morto
        }
        else
        {
                g_mex = @"Hai sprecato l'unico colpo..."@;
        }
        // in entrambi i casi sostituisco la pistola scarica a quella carica
        o_room[$(@"_pistolaScarica00_"@)] = o_room[$(@"_pistolaCarica00_"@)];
        o_room[$(@"_pistolaCarica00_"@)] = 0;
        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 (f_ais(@"_mattone03_"@,@"lancia"@,null)  // mattone - lancia
        || f_ais(@"_mattone03_"@,g_use_,@"_ispettore00_"@) // mattone - usa con... - ispettore
        || f_ais(@"_ispettore00_"@,@"colpisci"@,null) ) // ispettore - colpisci
{
        // se l'ispettore è nella nostra stanza
        if (o_room[$(@"_ispettore00_"@)] == g_room)
        {
                g_mex = @"L'ispettore delle tasse ha schivato il mattone!!!"@; // schiva il mattone
                o_room[$(@"_mattone03_"@)] = g_room; // il mattone resta nella stanza
        }
        else // altrimenti non facciamo nulla
        {
                g_mex = @"E dove lo vorresti lanciare?..."@;
        }
```

```
        return true;
}


return false;
```

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 *g_flag[1]*). Il messaggio *g_mex* viene composto da testo e dal contenuto della variabile stessa. Si notino gli spazi messi tramite virgolette singole (' ').

Le azioni "lampada spenta – accendi" e "lampada accesa – spegni" mostrano il trucco di far comparire un oggetto gemello e far sparire l'altro.

Viene anche gestita l'azione di rottura della lampada. Poteva anche essere fatta con due funzioni *f_ais* distinte, una per l'oggetto lampada accesa e una per l'oggetto lampada spenta. Nel codice invece si è fatta un'unica condizione nel corpo della quale si cerca quale oggetto è effettivamente in gioco (stanza diversa da 0).

La condizione più complessa è quella dell'azione "catena – tira" che non usa *f_ais* ma valuta *g_aid1, verb* e *g_aid2* per poter distinguere tramite *g_aid2* quante volte l'abbiamo tirata. In base a questo, al fatto che si abbia l'Occhio e indossato i guanti, si hanno diversi esiti. Il numero di volte che bisogna tirare la catena sta in *g_flag[1]*.

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.

Come già detto, l'ultima istruzione, da lasciare sempre, è *return false* che fa si che l'azione, non intercettata dalle nostre condizioni, sarà gestita dallo script di base.

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 il file "debug.html" e correggere eventuali errori.

Qui di seguito una lista dei possibili messaggi che indicano problemi nella generazione dell'adventure. Se ve ne sono e il file viene generato comunque, è probabile che non funzionerà o avrà comportamenti inattesi.

Warning: missing '}'

Warning: missing '{'

Nella descrizione gli oggetti non sono codificati correttamente

Warning: image file is empty - <nome file>

Il file immagine richiesto è vuoto

Warning: duplicated object <identificatore oggetto>

Nelle descrizioni è presente più volte lo stesso oggetto

Warning: missing objects specification <identificatori oggetti>

Uno o più oggetti non sono definiti in "objects.txt"

Warning: menu declared for unknown object - <identificatore oggetto>

In "menu.js" sono descritti menu per oggetti non esistenti

Warning: wrong string delimiter: "@ ? <testo>

Warning: wrong string delimiter: @" ? <testo>

Errato uso di @" o "@ per un testo

Warning: unknown object ? <identificatore oggetto>

Identificatore di oggetto sconosciuto tra @" o "@

Warning: line too long - <numero linea> <testo>

La linea dello script è troppo lunga

Warning: g_room instead of o_room ? - <numero linea> <testo>

Si sta usando g_room al posto di o_room

Warning: _debug_de_ hint not found

Error: Invalid base script, missing #TITLE

Lo script di base non è corretto

Error: 'Projects' folder not found

Error: unable to create folder: 'projects'

Error: unable to find 'projects' folder

Error: unable to create new project folder or files!

Error: unable to create <nome file>

Error: fail filtering file <nome file>

Error: failed object merging into description <numero descrizione>

Error: unable to merge

Errore di lettura o scrittura di file o cartelle

Error: process stopped by user

L'utente ha interrotto la generazione

Error: wrong syntax at line <numero linea> <linea>

Nel file "menu.js", "rooms.txt", "messages.txt" i caratteri < > sono usati in maniera non corretta

Error: unknown object propriety at line <numero linea> <linea>

Nel file "objects.txt" si sta specificando una proprietà non esistente

Error: unable to generate adventure

Non è possibile generare il file del gioco

Error: Adventure files generated but errors!

Il file del gioco è generato con errori

Dopo la linea di Warning o Error, può anche essere presente nel log il pezzo di codice antecedente al punto in cui si verifica il problema. Il codice potrebbe risultare in parte diverso dal nostro, in quanto Pandor+ lo adatta allo script di base.

Alcuni warning ed error sono relativi al singolo file di progetto che viene processato in quel momento. Ad esempio, se per un oggetto scriviamo "esaminable" al posto di "examinable":

```
01/07/2016 14:23:11 > Added message: 37
01/07/2016 14:23:11 > Added message: 38
01/07/2016 14:23:11 > Merging: E:\pandor\projects\occhio\objects.txt
01/07/2016 14:23:11 > Filtering file: E:\pandor\projects\occhio\objects.txt
01/07/2016 14:23:11 > Added object: _est01_
01/07/2016 14:23:11 > Added object: _sud02_
01/07/2016 14:23:11 > Added object: _ovest02_
...
01/07/2016 14:23:11 > Added object: _biglie00_
01/07/2016 14:23:11 > Error: unknown object propriety at line 183 - esaminable
01/07/2016 14:23:11 > Added object: _occhio00_
01/07/2016 14:23:11 > Added object: _mattone03_
```

Per sapere in che file agire basta risalire dall'errore o warning all'indicazione di che file era in quel momento elaborato:

```
01/07/2016 14:23:11 > Merging: E:\pandor\projects\occhio\objects.txt
```

A partire dalla linea "Converting @ strings…", eventuali altre indicazioni invece sono relative al file "debug.html" generato e quindi dovremo ricercare a mano il pezzo di codice indicato in uno dei nostri file ".js" e cercare di individuare l'errore. Ad esempio se in "actions.js" una delle nostre funzioni *f_ais* fa un uso errato dei delimitatori @" e "@:

```
// se ci si siede sul trono appare un laconico messaggio
if (f_ais(@"_trono12_"@,@"siedi@",null))
{
        g_mex = @"Ti senti un re! Però è scomodo."@;
        return true;
}
```

questo è quello che vedremo in fase di generazione:

```
01/07/2016 14:26:48 > Converting @ strings...
01/07/2016 14:26:48 > Warning: wrong string delimiter: @" ?
/ se ci si siede sul trono appare un laconico messaggio
if (f_ais('_trono12_','
01/07/2016 14:26:48 > Warning: wrong string delimiter: @" ?
/ se ci si siede sul trono appare un laconico messaggio
```

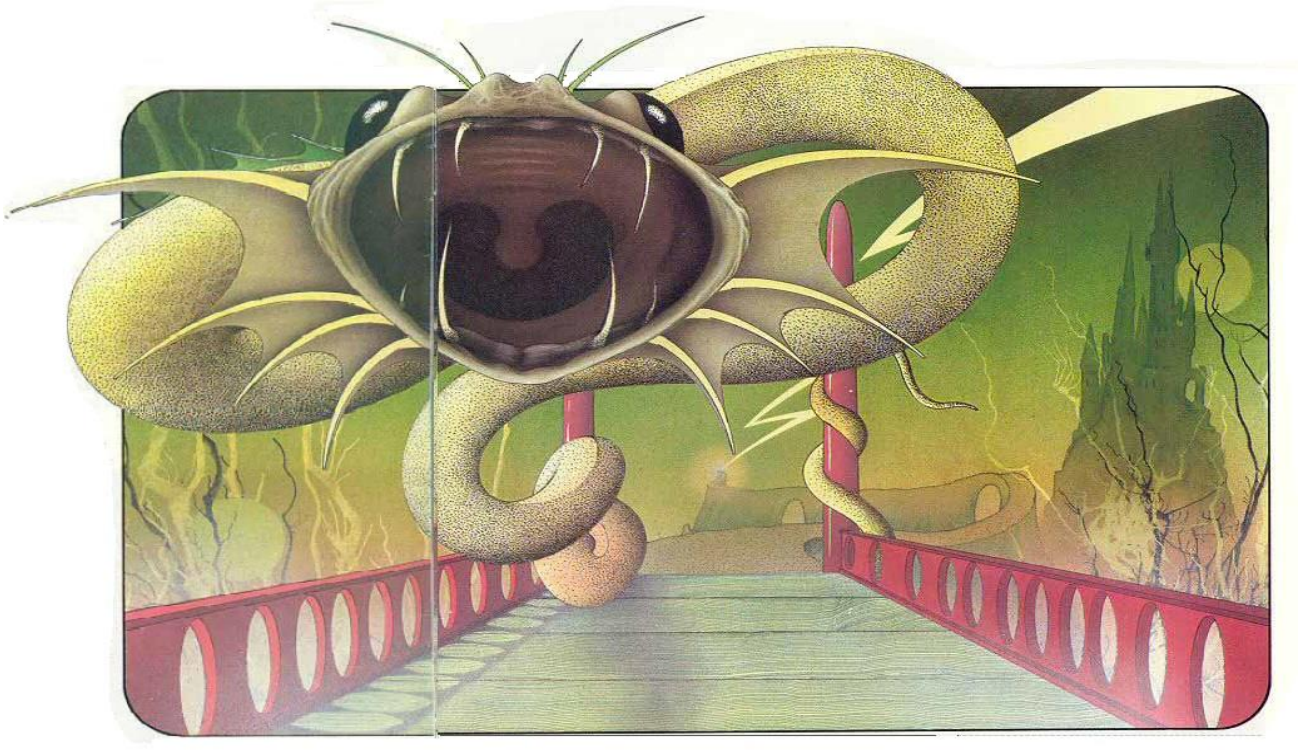
```
if (f_ais('_trono12_','
01/07/2016 14:26:48 > ...10%
01/07/2016 14:26:49 > ...20%
```

I warning ci avvertono in questo caso che stiamo utilizzando male i delimitatori nel pezzo di codice riportato. Come si vede, il codice in "debug.html" è diverso per alcuni caratteri ma è comunque riconoscibile.

Un'altra importante indicazione è quella relativa alla presenza tra @" … "@ di un identificatore di un oggetto che non è stato definito. Supponiamo ad esempio che in qualche nostro file di progetto abbiamo usato "_nattone03_" al posto di "_mattone03_". Ecco il warning che ci sarà dato:

```
07/07/2016 10:32:17 > Converting @ strings...
07/07/2016 10:32:17 > ...10%
07/07/2016 10:32:17 > Warning: unknown object ? _nattone03_
```

Sarà compito nostro cercare con l'editor nei nostri file di progetto "_nattone03_" e correggere l'errore.



Il debug col browser Internet

La versione attuale di Pandor+ fornisce un supporto molto limitato al debug e alla correzione degli eventuali errori nei file di progetto.

Se vi sono errori nel codice javascript della nostra avventura è possibile che Pandor+ non dia nessun warning o errore e che quindi 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 del browser stesso.

Possiamo però utilizzare il debugger di Internet Explorer (o del browser che utilizziamo) 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 "debug.html", e deve essere stato generato senza warning o errori da Pandor+.

Bisogna tener conto che "debug.html" è una fusione dello script di base e dei nostri file di progetto che vengono poi modificati.

Per fare un esempio, le espressioni @" … "@ vengono trasformate in un codice HTML equivalente racchiuso tra virgolette singole. Di conseguenza non sempre è semplice (ispezionando il codice della pagina) capire da quale file di progetto è derivato il pezzo di codice che stiamo osservando. Ad esempio:

```
        @"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..."@
```

diventa

```
        'L\'ispettore delle tasse esige il riscatto per i tuoi debiti ma tu non hai nulla  da dargli... Purtroppo
per te la tua pena &egrave; la gattabuia...'
```

Un’altra importante differenza è che tutte le nostre variabili saranno anticipate da "window." che indica che la variabile è globale per tutta la pagina web del gioco. Ad esempio:

```
        if (g_flag[5]==0) // non abbiamo mai nuotato
```

diventa

```
        if (window.g_flag[5]==0) // non abbiamo mai nuotato
```

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

Un’importante premessa da fare è che, durante i nostri debug col browser, **tutte le volte che modificheremo uno o più dei nostri file di progetto** sarà necessario:

- Salvare tutti i file di progetto con le modifiche

- rigenerare il file "debug.html" con Pandor+
- riaprire il file nel browser o se era già aperto, fare il refresh della pagina (in Explorer con F5)
- terminare l'avventura per riportarla all'inizio
- fare il refresh della pagina (in Explorer con F5)

Chiameremo la procedura appena vista "reset dell'avventura".

Oltre a questo, molto probabilmente gli evenuali salvataggi (1,2,3 o 4) non saranno utilizzabili perché non "sincronizzati" con l'ultima versione del gioco.

Se invece non modifichiamo i nostri file di progetto, possiamo tranquillamente utilizzare i salvataggi per provare tutte le nostre stanze, azioni, condizioni, etc.

Come fare quindi a partire direttamente in una certa posizione nel gioco, con certi oggetti e con certi *g_flag* settati come vogliamo per poter debuggare una certa situazione?

Il trucco è aggiungere un pezzo di codice in "conditions.js" che fa si' che il gioco parta in una condizione diversa da quella "normale". Ad esempio, se nell'avventura dell'Occhio volessimo partire con la pistola carica e nella stanza del trono, simulando di aver già fatto almeno 18 azioni, ecco come modificare il file "conditions.js":

```
if (is_action)
{

        // CONDIZIONI CHE TENGONO CONTO DELLE AZIONI FATTE
        // ----------------------------------------------

        // g_flag[0] incrementato ad ogni azione
        g_flag[0] = g_flag[0] + 1;

        // se sono nella foresta, incrementa g_flag[3]
        if (g_room == 02)
        {
                g_flag[3] = g_flag[3] + 1;
        }
        else // altrimenti mettilo a 0, in modo da ricominciare il conto delle azioni
             // nella foresta da zero se ci si ritorna
```

```
		{
			g_flag[3] = 0;
		}

		// se è comparso l'ispettore, incrementa g_flag[6]
		if (g_flag[2] == 1)
		{
			g_flag[6] = g_flag[6] + 1;
		}

		return;
	}

	// DEBUG
	if (g_flag[0]==0)
	{
		g_flag[0] = 18;
		o_room[$(@"_pistolaCarica00_"@)] = -1;
		o_room[$(@"_pistolaScarica00_"@)] = 0;
		g_room = 12;
	}
	//

	// CONDIZIONI CHE NON MODIFICANO LO STATO DEL GIOCO
	// -----------------------------------------------

	// se abbiamo (o è presente) la lampada accesa nella stanza buia, cambia la descrizione della stanza
	if ( ( (o_room[$(@"_lampadaAccesa00_"@)] == -1) ||  // lampada in inventario
		((o_room[$(@"_lampadaAccesa00_"@)] == g_room))  // lampada nella stanza in cui siamo
	    )
	     && (g_room == 09)
	   )
	{
```

```
...
```

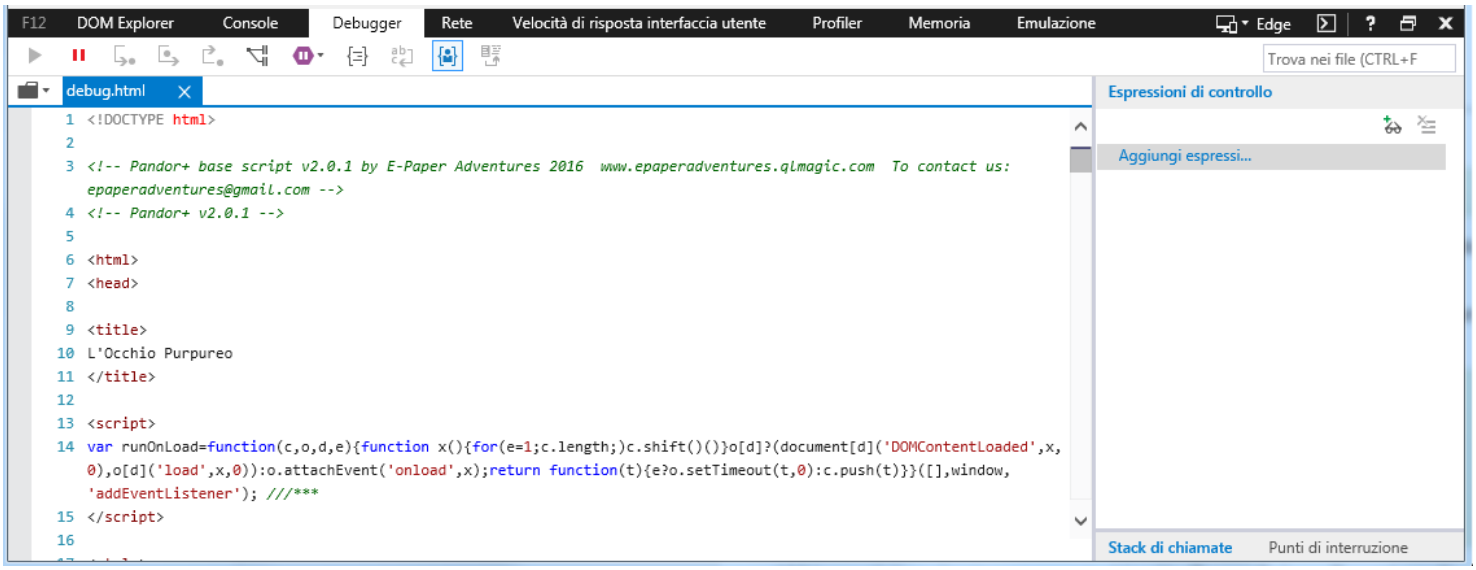
Questo sistema può essere utilizzato solo se *g_flag[0]* è utilizzato come visto finora, cioè per tener conto delle azioni non vuote fatte (ogni azione incrementa di 1 questa variabile) e il pezzo di codice di debug va inserito immediatamente dopo *if (is_action) { ... }*

Facciamo ora un primo semplice esempio di individuazione di un errore che non è stato rilevato da Pandor+. Apriamo "actions.js" e modifichiamolo in questo modo:

```
// se nuotiamo nel fiume
if (f_ais(@"_fiume01_"@,@"nuota"@,null))
{
        if (g_flag[5]==0)) // non abbiamo mai nuotato
```

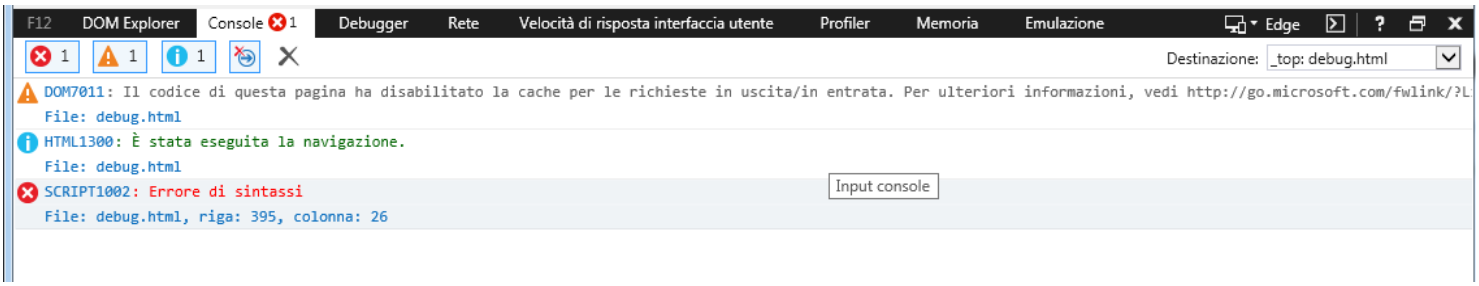
Abbiamo cioè aggiunto una parentesi in più alla condizione. Resettiamo l'avventura (Pandor+ non ci avverte di nulla…) e vedremo che la pagina del gioco sarà completamente bianca!

Iniziamo dunque il nostro debug. In Explorer, premiamo F12. Attendiamo e nella parte bassa del browser apparirà un menu a TAB:

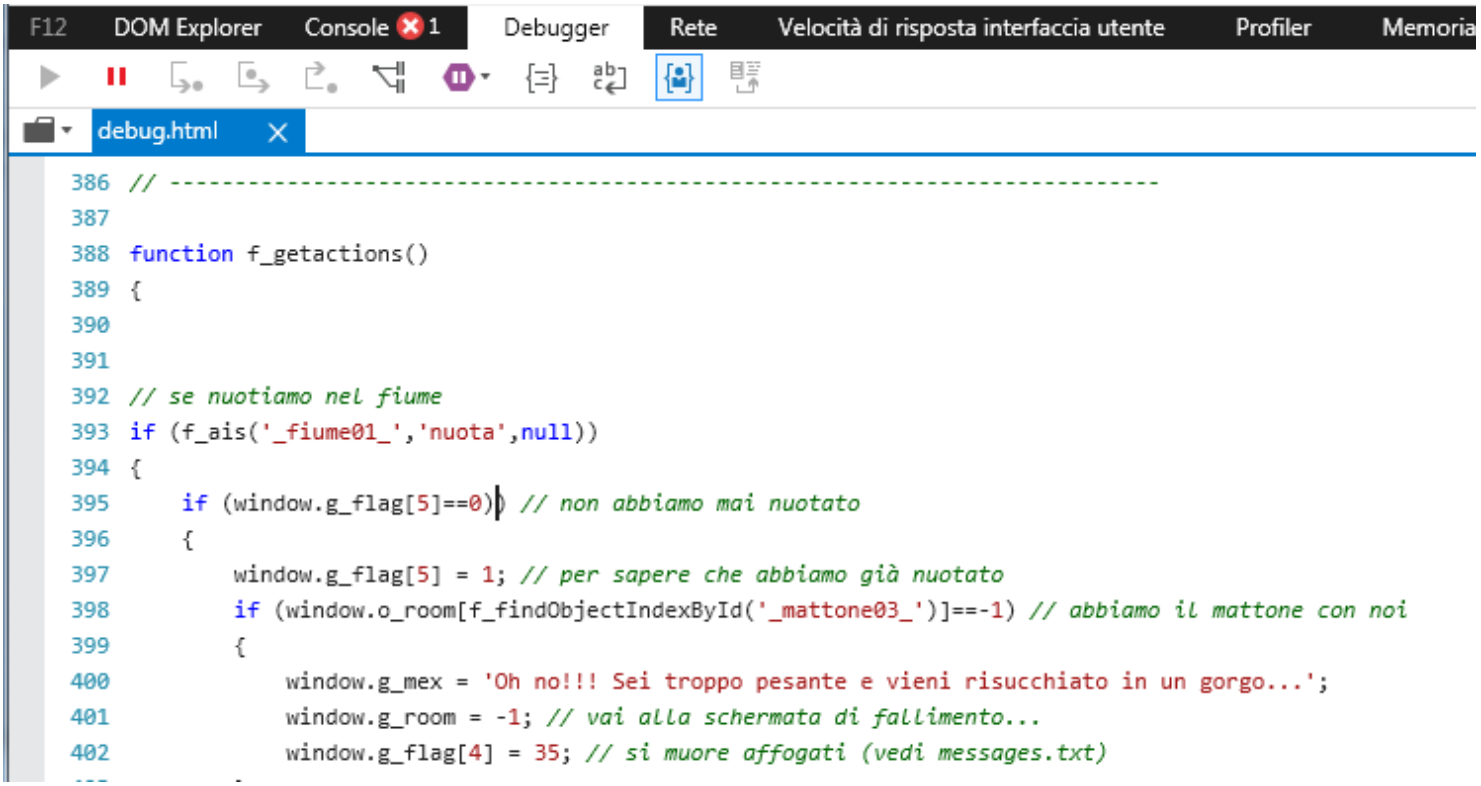


Clicchiamo su "Console", clicchiamo nel mezzo della pagina vuota (dove dovrebbe esserci il testo del nostro gioco…) e premiamo F5 per fare il refresh della pagina:

Ecco che ci viene indicato che c'è un errore di sintassi ad una certa linea di "debug.html":



Clicchiamo proprio sulla linea sotto "Errore di sintassi" (File: …) e vedremo che verrà mostrato il codice con il cursore posizionato esattamente dove si trova l'errore:

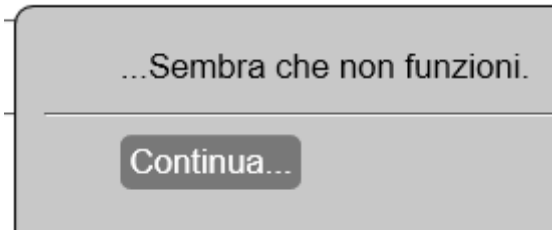


Come si vede nel codice è stato aggiunto "window." prima della variabile *g_flag[5]* ma è comunque riconoscibile che ci troviamo in "actions.js" e quindi possiamo andare a modificare proprio questo file di progetto facilmente. Poi sarà necessario resettare l'avventura e riprovare.

Ci sono però errori che non sono altrettanto evidenti, ad esempio modifichiamo sempre questa parte di "actions.js" in questo modo:

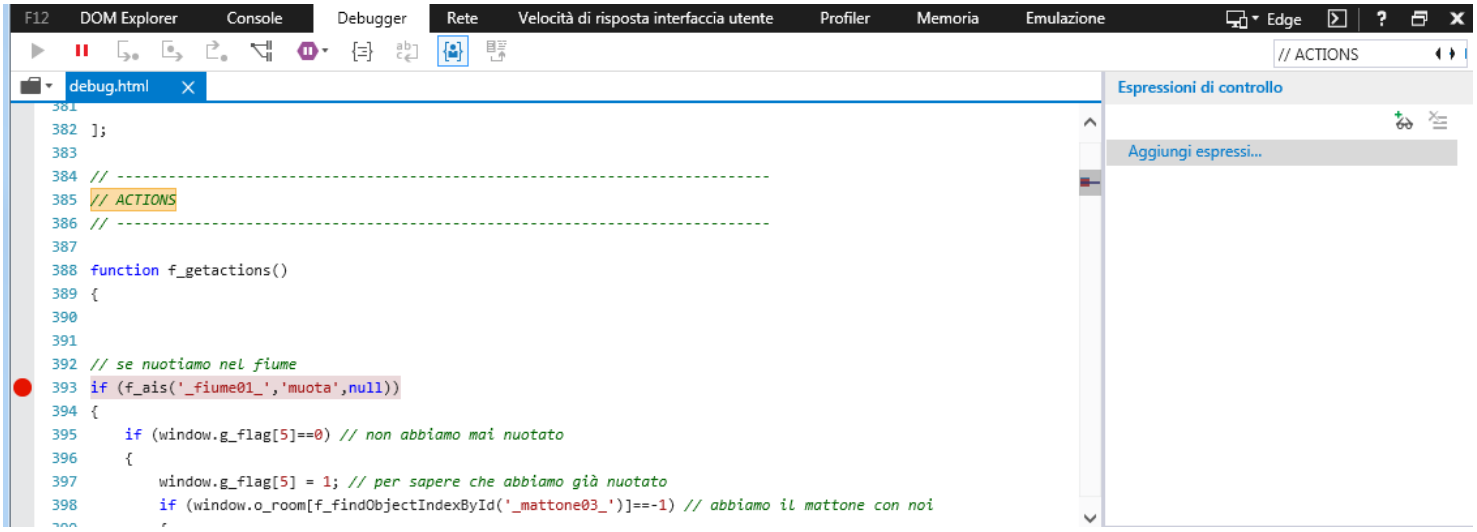
```
// se nuotiamo nel fiume
if (f_ais(@"_fiume01_"@,@"muota"@,null))
{
        if (g_flag[5]==0) // non abbiamo mai nuotato
```

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 (e ad esempio comparire la pistola), il gioco mostrerà un messaggio:



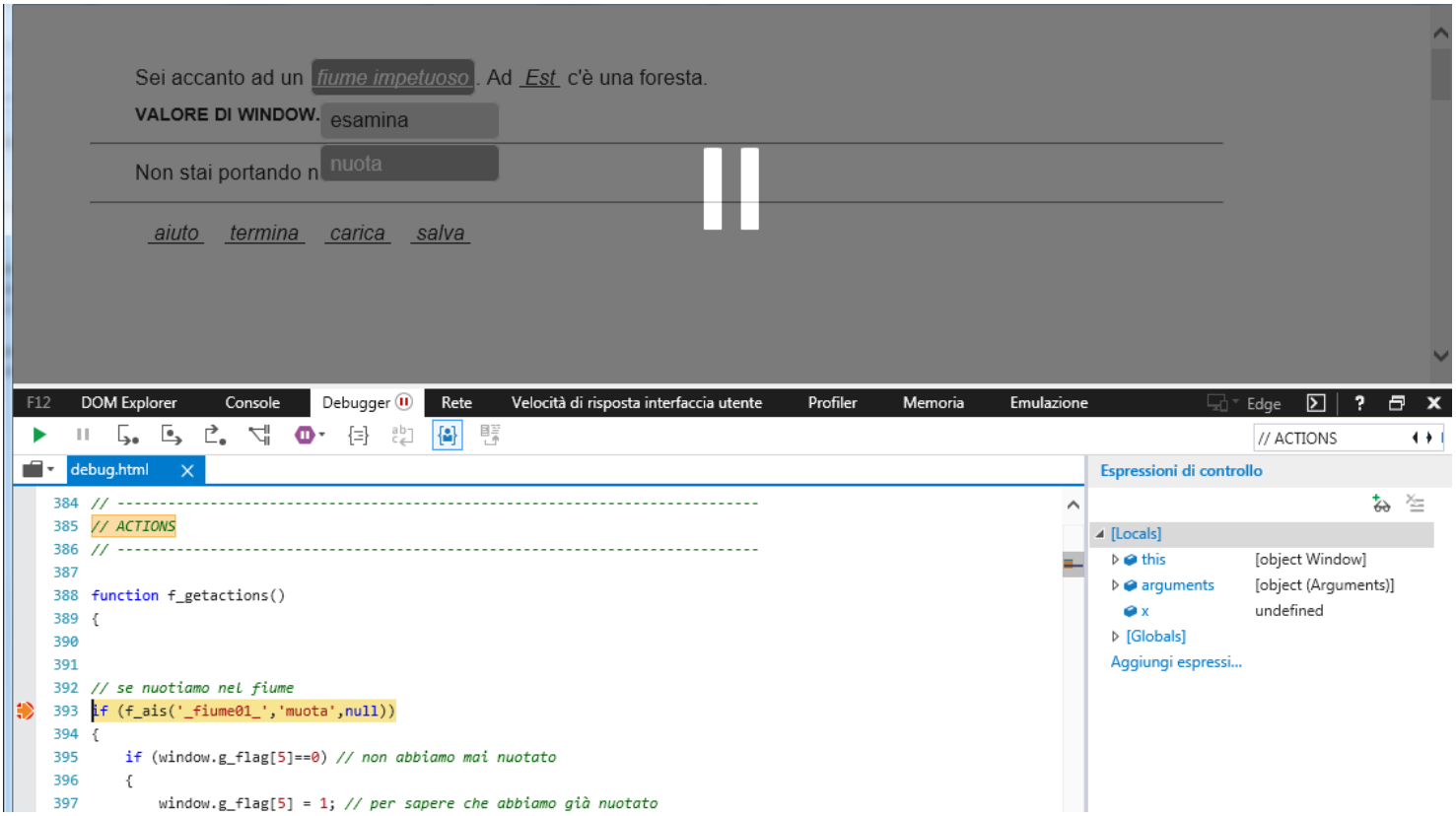
Questo perché l'azione "fiume – nuota" non è stata intereccettata dalle nostre condizioni in "actions.js".

Per capire cosa succede, premiamo nuovamente F12, e nel TAB "Debugger", mettiamo nel campo di ricerca (in alto a destra) la frase "// ACTIONS" per cercare il pezzo di codice in cui si trovano le nostre condizioni di "actions.js".



A questo punto clicchiamo a sinistra della condizione che vogliamo esaminare: in questo modo aggiungeremo un "breakpoint" per far fermare lì l'esecuzione del codice javascript e poter ispezionare le variabili che ci interessano o capire se stiamo o no entrando nel corpo della condizione.

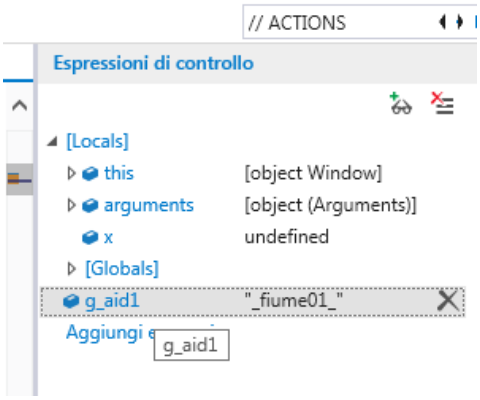
Proviamo ora a ripetere l'azione "fiume – nuota". La pagina si bloccherà e vedremo che verremo posizionati proprio sulla linea del breakpoint.



Attenzione che a volte è necessario abbassare tutte le finestre di tutte le altre applicazioni per poter vedere la finestra di Explorer sotto debug.

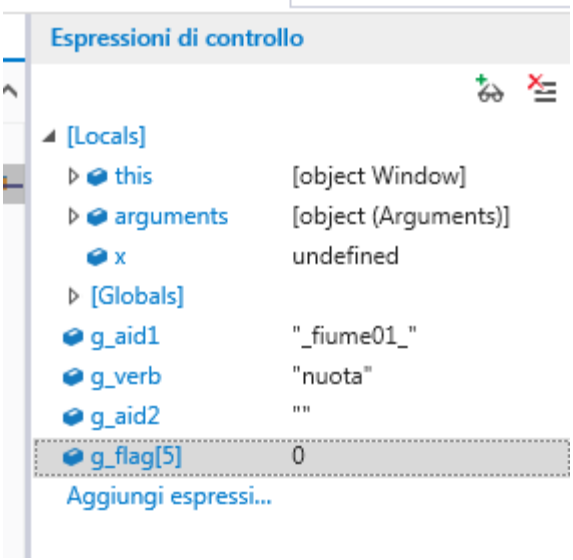
A questo punto possiamo usare i pulsanti in alto per vedere cosa accade nel nostro codice.

Prima di farlo però, clicchiamo a destra dove è scritto "Aggiungi espressione" e scriviamo: "g_aid1" e invio.



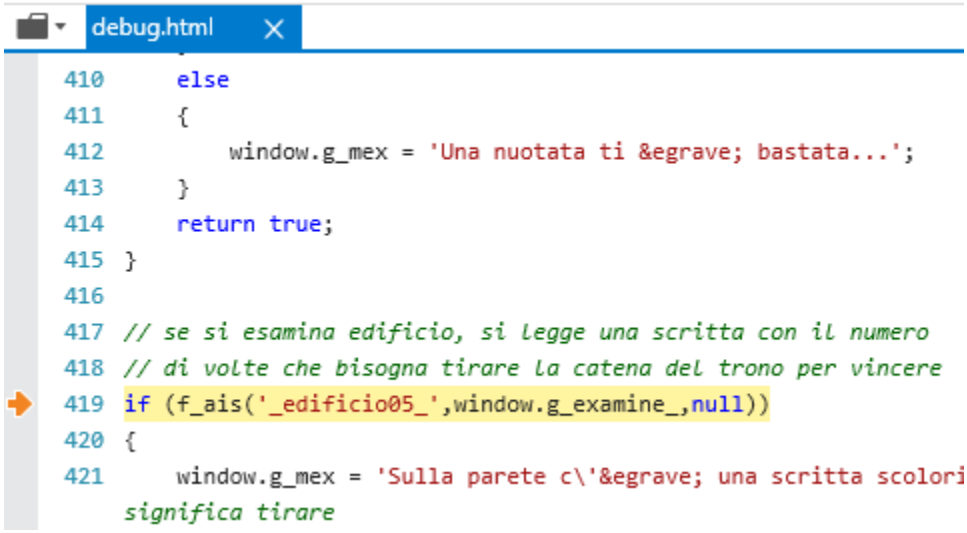
Il debugger ci mostrerà l'attuale valore della variabile *g_aid1* che è quella che contiene l'identificatore dell'oggetto su cui stiamo agendo. In pratica possiamo sempre conoscere in questo modo il valore di tutte le variabili nel gioco e capire cosa sta accadendo!

Aggiungiamo anche "g_verb" e "g_aid2" e "g_flag[5]":



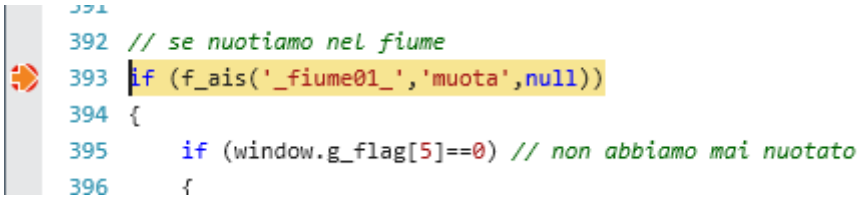
L'azione è propria quella che ci aspettiamo. Non resta che capire il perché non "entriamo" nella nostra condizione. Premiamo il pulsante in alto:

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.

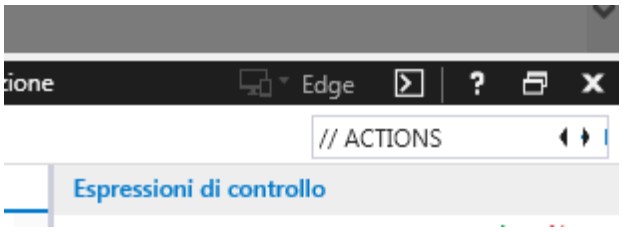


Un'ispezione visiva della linea porterà a individuare l'errore di scrittura e quindi a risolvere il problema.

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").

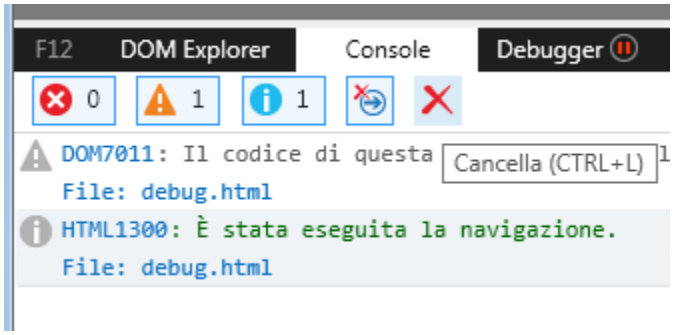
Possiamo anche scorrere il codice manualmente tramite la barra di scorrimento o cercare delle frasi usate nei commenti per arrivare più velocemente al punto che ci interessa.

Per terminare il debug, basta cliccare sulla "X" in alto a destra:



Per togliere invece un breakpoint basta ricliccare sopra il pallino rosso che lo rappresenta.

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 la "X":



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

C'è un altro subdolo errore che non viene rilevato facilmente, ed è quello di sbagliare il nome delle variabili. Poiché in javascript non è necessario dichiarare esplicitamente una variabile, se scrivessimo ad esempio:

```
g_rooom = 12;
```

al posto di

```
g_room = 12;
```

non verrebbe segnalato alcun errore ne da Pandor+ ne dal browser. Il motivo è che l'istruzione *g_rooom = 12* crea una nuova variabile locale di nome *g_rooom* che non è ovviamente quella che vogliamo modificare.

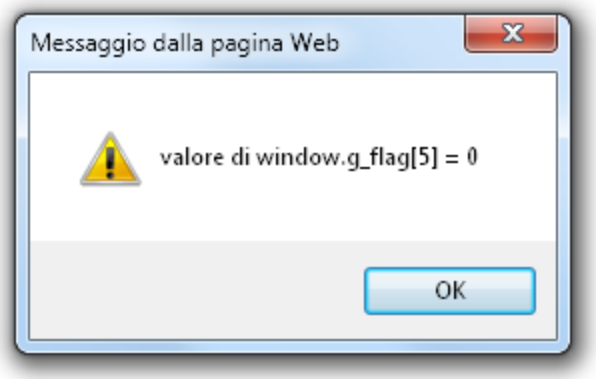
Come visto però si può indagare nel codice per capire perché una nostra condizione non viene eseguita e controllare quella parte di codice per trovare errori di questo tipo.

Altri sistemi per debuggare il nostro progetto è quello di utilizzare la 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) e visualizzare il valore di una variabile.

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

```
// se nuotiamo nel fiume
if (f_ais(@"_fiume01_"@,@"nuota"@,null))
{
        alert('valore di g_flag[5] = '+g_flag[5]);

        if (g_flag[5]==0) // non abbiamo mai nuotato
```

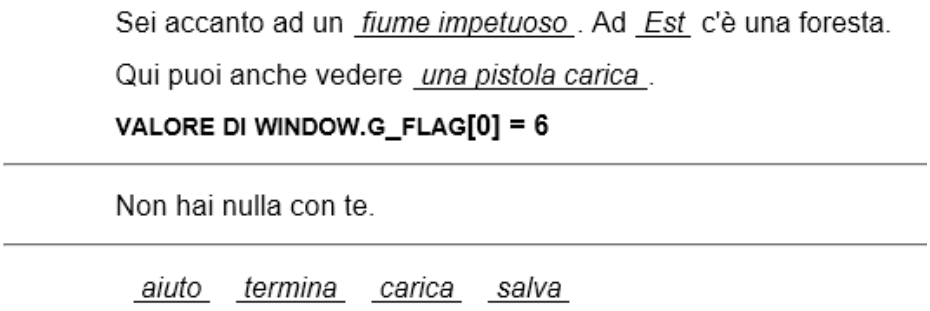
Se ora resettiamo l'avventura e proviamo a fare "fiume – nuota" un paio di volte, vedremo apparire un messaggio che ci indica lo stato della variabile.



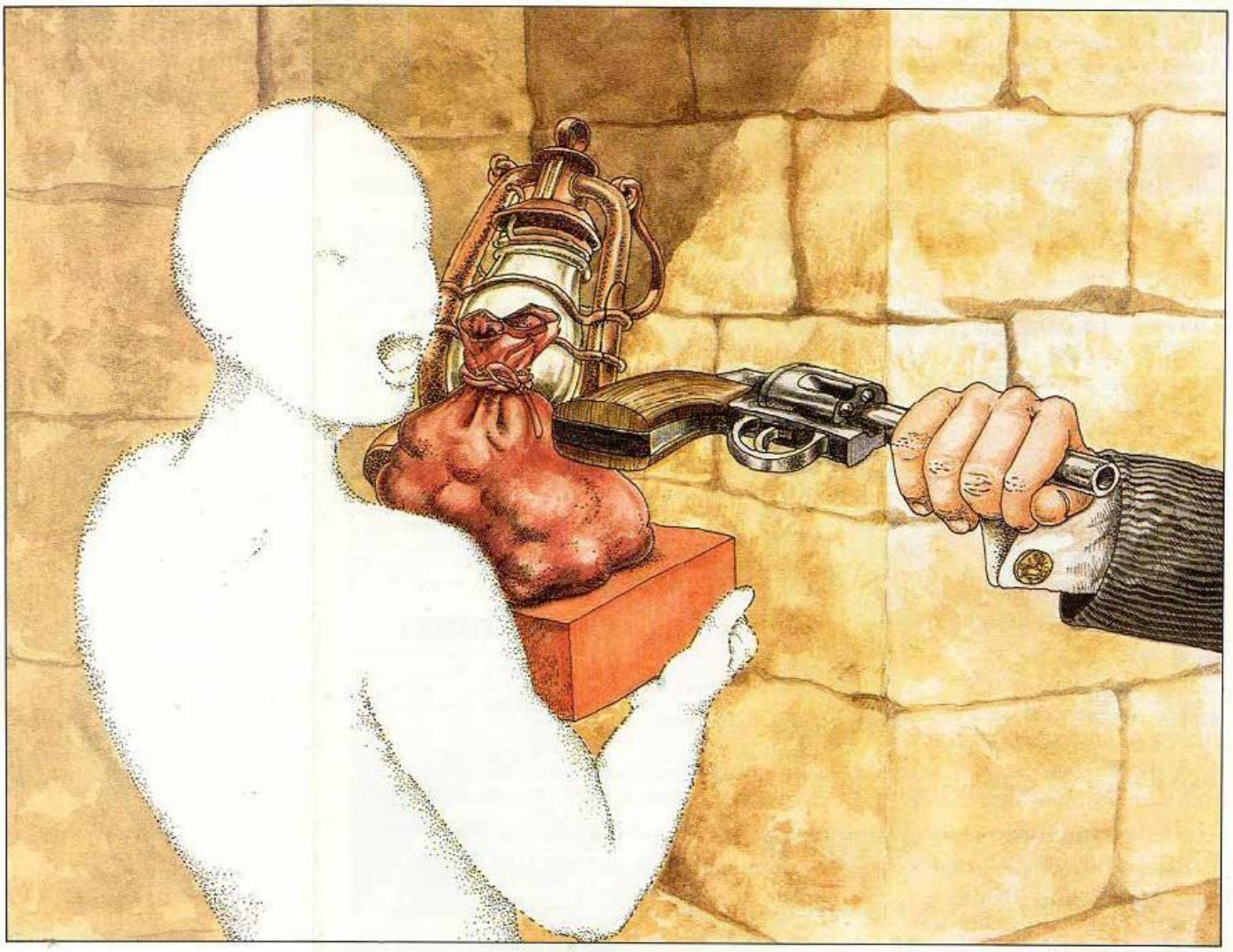
Allo stesso modo possiamo pensare di usare *g_addtext* per scrivere dei messaggi di controllo con il contenuto delle variabili che ci interessano. Per esempio, basta aggiungere alla fine di "conditions.js" (come ultima linea):

```
g_addtext = g_addtext + ' valore di g_flag[5] = '+g_flag[5];
```

Ed ecco che ora, per ogni stanza, avremo il valore di *g_flag[0]*.

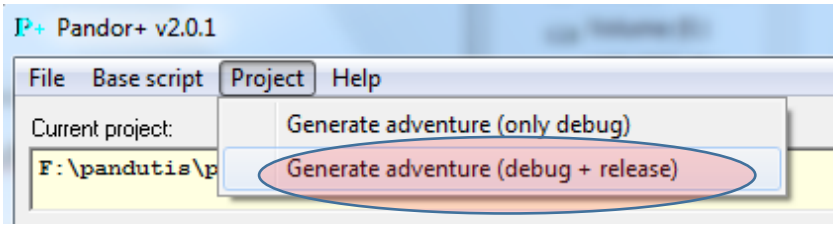


Infine qualora un oggetto non si comportasse come ci attendiamo (mancano azioni, non è cliccabile, etc) può sempre essere dovuto ad un errore in "objects.txt" o in "menu.js".



Finalizzare il gioco

Una volta che la nostra avventura è pronta e il file "debug.html" funziona correttamente, possiamo passare a generare la versione finale nella quale i testi saranno criptati in modo che un'eventuale apertura del file html non permetta di scoprire facilmente come risolvere il gioco.



Generiamo quindi il file "release.html" utilizzando l'opzione evidenziata. Successivamente il file va rinominato col nome finale.

Il modo più semplice per renderlo disponibile sul web è contattarci: saremo ben felici di aggiungere la vostra avventura alle nostre nella pagina web: http://www.epaperadventures.qlmagic.com/adv.html

I prossimi capitoli del tutorial sono dedicati ad un uso avanzato di Pandor+ e non sono strettamente necessari per creare un’avventura.

Aggiungere funzioni javascript

Nei file "actions.js" e "conditions.js" potremmo aver bisogno di una funzione javascript definita da noi. Il modo per farlo è aggiungere nella cartella del progetto un file con nome "functions.js" (che non viene creato automaticamente da Pandor+). Nel file possiamo inserire funzioni che poi possiamo richiamare negli altri file.

Per esempio, ecco una funzione per far sparire tutti gli oggetti "mobili" che si trovano in una stanza:

```
function f_rimuovioggetti(l_stanza)
{
  for (var i = g_defaultobjnum; i < o_room.length; i++)
    if (o_room[i]==l_stanza)
      o_room[i]=0;
}
```

g_defaultobjnum è l'indice del primo oggetto definito dall'utente. Quelli di indice precedente sono utilizzati internamente dallo script di base.

o_room.length contiene il numero totale di oggetti definiti nell'avventura.

Usate sempre per i parametri della funzione dei nomi che iniziano con "l_" (l sta per local). Questo eviterà possibili errori quando Pandor+ aggiungerà la vostra funzione allo script finale. Allo stesso modo, per le variabili interne alla funzione, utilizzate nomi di una sola lettera (i, j, k...) oppure nomi che iniziano con "l_" (l_conta, l_numerostanza, etc).

In "actions.js" potremmo usare la funzione a seguito di una certa azione, per far scomparire tutti gli oggetti "mobili" nella stanza in cui ci troviamo, in questo modo:

```
if ( … )
{
        g_mex = g_mex + ' ' + "Scompare tutto!!!";
        f_rimuovioggetti(g_room);
        return true;
}
```

Modificare dinamicamente le schermate finali del gioco

Riprendiamo il nostro file "success.html". Immaginiamo ora di voler aggiungere il numero di mosse compiute per arrivare alla fine del gioco.

Per farlo, utilizziamo le parentesi graffe, in mezzo alle quali possiamo specificare una delle nostre variabili *g_flag*, in particolare quella di indice 0 che conta il numero di azioni compiute:

```
<h1>COMPLIMENTI !!!</h1>
<pcenter>
___IMAGE___
</pcenter>
<p>
Hai trovato il prezioso Occhio Purpureo.
Con la sua vendita risanerai i tuoi debiti e vivrai
per sempre di rendita! Ben fatto!!!
</p>
<p>Hai compiuto {g_flag[0]} mosse!</p>
<hr>
```

Se invece vogliamo scrivere un messaggio diverso, una possibile strada da seguire è aggiungere dei messaggi in "messages.txt". Ad esempio apriamo il file e aggiungiamo i messaggi di indice 35, 36, 37 e 38:

```
...
<34>
L'indirizzo della pagina verrà automaticamente _
salvato nella cronologia assieme alla tua posizione! _
Quando vorrai riprendere da questo punto _
vai nella cronologia, scegli la pagina e _
quindi nel gioco tocca _
"CARICA" e "#4".
<35>
Sei morto affogato!
<36>
```

```
Sei morto fulminato!
<37>
Sei morto pietrificato!
<38>
Sei finito in gattabuia!
```

E modifichiamo in questo modo il file "fail.html":

```
<h1>{f_gmes(g_flag[4])}</h1>
<pcenter>
___IMAGE___
</pcenter>
<p>
Non sei riuscito a trovare il prezioso Occhio Purpureo.
Ritenta, sarai più fortunato!
</p>
<hr>
```

Il titolo della schermata dipenderà dal contenuto della variabile *g_flag[4]*, che nel codice dell'avventura viene valorizzato a 35,36,37 o 38 in base al triste destino che coglie il giocatore. La funzione *f_gmes* viene quindi chiamata con un indice di messaggio che può essere 35,36,37 o 38 e il messaggio mostrato dipenderà quindi dalla situazione in cui il gioco è terminato.

A differenza dei messaggi, in questa versione di Pandor+ non è previsto di poter visualizzare immagini diverse.

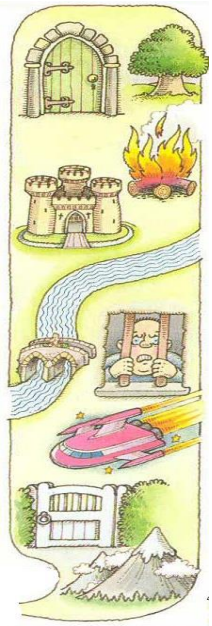
Un'altra strada è quella di definire dei messaggi attraverso nostre funzioni javascript. Ad esempio per mostrare il grado di bravura del giocatore (Genio! - Sagace! - Scarso! …) oppure indicare la percentuale di completamento dell'avventura. Per farlo, possiamo ad esempio definire una funzione (attraverso il file "functions.js", come visto in precedenza) che in base a tutto quello che è successo nell'avventura, restituisce una frase con la percentuale di completamento. Ecco una possibile idea di implementazione:

```
function f_percentuale()
{
	var l_perc = 0; // percentuale completamento avventura
	if (…) l_perc = l_perc + 10;  // aumento del 10% la percentuale
```

```
        if (…) l_perc = l_perc + 5;
     ...
        return @"Hai completato il"@ +' '+ l_perc +@"% dell'avventura!"@;
}
```

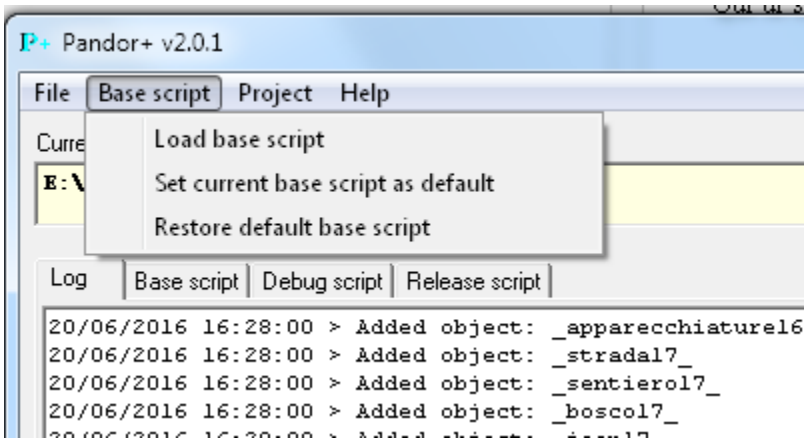
Le condizioni dei vari *if* potrebbero riguardare il possesso di un oggetto, l'aver risolto un enigma, etc. Si noti l'aggiunta di uno spazio prima della variabile *l_perc*. Il file "fail.html" potrebbe essere dunque così modificato:

```
<h1>{f_gmes(g_flag[4])}</h1>
<pcenter>
___IMAGE___
</pcenter>
<p>
Non sei riuscito a trovare il prezioso Occhio Purpureo.
Ritenta, sarai più fortunato!
</p>
<p>{f_percentuale()}</p>
<hr>
```



Il menu "Base Script"

Il menu "Base script" di Pandor+ permette di caricare uno script di base diverso attraverso l'opzione "Load base script".



Una volta caricato il file è possibile fare in modo che diventi lo script di default per tutti i progetti, scegliendo "Set current base script as default". Verrà creata una copia dello script nella cartella "projects" di nome "base.js", che verrà letto ad ogni riavvio di Pandor+.

L'ultima opzione permette di cancellare un eventuale file "base.js" presente nella cartella "projects", in modo da ripristinare l'uso dello script di base di default della versione corrente di Pandor+.

Infine, se volessimo utilizzare degli script di base diversi per i vari progetti, basta copiare manualmente il file "base.js" nella directory del progetto.