Inform 6 mette a disposizione la possibilità di stampare il testo in diversi stili

```
Include "Parser";
Include "Verblib";
Include "Replace";

Object Stanza "Una stanza"
with description "Ti trovi in una stanza."
has light;

Object -> scritte "scritte"
with initial "Sulle pareti della stanza sono presenti delle scritte.",
before [;
Examine:
print "Sono diverse:^";
style bold; print "una scritta in grassetto^";
! 1
style roman;  ! eliminiamo lo stile grassetto
! 2
style underline; print "una scritta in corsivo^";
style roman;  ! eliminiamo lo stile corsivo
style reverse; print "una scritta rovesciata^";
style roman;  ! eliminiamo lo stile rovesciato
print "inoltre anche^";
SetColour(5,1); print "una scritta gialla^";
SetColour(1,1);  ! riportiamo i colori alla normalità
style roman;  ! riportiamo lo stile alla normalità
rtrue;  ! 3
],
has static female pluralname;

[ Initialise;
location = stanza;
print "Vuoi supportare i colori (s/n)? ";
if (yesorno()) clr_on = 1;
else clr_on = 0;
];

Include "ItalianG";
```

L'effetto di ciascuna istruzione è facilmente comprensibile una volta visualizzato l'effetto ottenuto esaminando le scritte durante il gioco, quindi sui vari stili c'è poco da dire.

Vediamo invece di analizzare qualche altro particolare che ci tornerà utile in futuro.

Innanzitutto vediamo, nell'oggetto scritte, una nuova proprietà: **initial**. Questa serve per modificare la presentazione dell'oggetto nella stanza. Di default il messaggio sarebbe *Puoi vedere delle scritte qui*, invece con la nostra descrizione diciamo anche che sono visibili sulle pareti. Naturalmente, questa proprietà è utile per dare un tocco personalizzato all'avventura e torna utile per un buon game design. [chi lo sa? magari torna anche utile sapere che ci sono delle pareti... ma, sempre parlando di game design, è bene stare attenti a quello che si nomina: generalmente i giocatori, soprattutto quelli più

scafati, cercano di interagire con qualsiasi cosa nominata nel corso dell'avventura e molto spesso vengono frustrati nel non ottenere una risposta adeguata o, peggio, nel vedere che l'oggetto con cui stanno cercando di interagire non esiste!]

Vediamo anche che abbiamo dato alle scritte un nuovo attributo: **static**. Questo permette al programma di capire che l'oggetto che lo possiede non può essere preso/spostato, rispondendo con una frase di default adeguata ogni volta che si cerchi di prenderlo o spostarlo.
[ad esempio, tutti gli oggetti (non locazioni) definiti degli esempi precedenti possono essere presi e portati con sé!]

Esiste un altro attributo in grado di sortire il medesimo effetto, ovvero **scenery**, ma ne parleremo in un prossimo articolo.

Notiamo poi l'istruzione **print**, che altro non significa che "stampa", cioè "fai comparire sullo schermo". Fino ad ora non l'avevamo mai utilizzata, dal momento che non è mai stato obbligatorio ricorrere ad essa. Qui invece è necessaria perché oltre a stampare qualcosa stiamo richiedendo al programma di fare "contemporaneamente" dell'altro (e cioè cambiare gli stili di testo).
[Molto spesso nelle avventure testuali un'azione altera l'ambiente di gioco tramite la modificazione di variabili create dal programmatore: in questi casi si indicherà sempre il nuovo valore assunto dalla variabile seguito dall'istruzione *print* per stampare il messaggio desiderato]

Vediamo che nei testi quotati delle istruzioni *print* compaiono degli apici **^**: questi servono per dire al programma "vai a capo". Anche questi non erano mai stati utilizzati prima, questo perché di norma quando il programma stampa qualcosa a schermo senza l'istruzione *print* aggiunge automaticamente un a capo. La stessa cosa non avviene se si specifica l'istruzione *print*.

I colori, fortunatamente, sono preimpostati. Esistono numerosi casi in cui, vuoi per il compilatore o l'interprete che si utilizza, bisogna definire uno stile colorato apposta per ogni colore che si vuole utilizzare tramite alcune routine: è un procedimento non molto laborioso ma per il momento troppo complicato che quindi ora come ora non vedremo.

L'istruzione predefinita SetColour(,) è proprio quella che permette di cambiare colore al testo. Ecco di seguito le varie corrispondenze.

DEFAULT 1
NERO 2
ROSSO 3
VERDE 4
GIALLO 5
BLU 6
MAGENTA 7
CIANO 8
BIANCO 9
VIOLA 7
AZZURRO 8

Nell'istruzione SetColour(,) il numero prima della virgola indica il colore del testo, quello dopo il colore dello sfondo. Durante la programmazione non fatevi trarre in inganno dallo sfondo del vostro interprete: potreste averlo blu di default e, andando a reimpostare i colori normali potreste digitare sbadatamente 6 senza contare che magari l'interprete di un altro giocatore ha di default il colore bianco, il risultato sarebbe catastrofico per l'altro giocatore. Lo stesso discorso vale per il colore delle

scritte. Quando dovete reimpostare i colori normali fate riferimento sempre al "colore" default, ossia al numero 1.

Nonostante il nostro impegno per dare un po' di vivacità (si spera funzionale all'avventura) al testo, potrebbe accadere che qualcuno non voglia o *non possa* visualizzare i colori, pena un qualche crash o bug del gioco non desiderato. Ecco perché nella routine *Initialise* (che, pur non avendolo mai specificato, è la prima cosa in assoluto che viene eseguita dall'interprete), abbiamo aggiunto la possibilità di chiedere al giocatore se vuole i colori.

L'istruzione **yesorno()** è una comoda istruzione preimpostata che permette di fare una domanda al giocatore per cui ci si aspetta semplicemente una riposta affermativa o negativa (affinché la risposta sia efficace il giocatore deve digitare solamente *> s* o *>n*, ecco spiegato perché chiediamo al programma di suggerire la risposta stampando a schermo *(s/n)*).
Vedremo l'istruzione **if else** nei prossimi articoli, per ora vi basti sapere che se il giocatore digita *> s* viene cambiato a 1 il valore della variabile (preimpostata) *clr_on*, il che significa che vedrà visualizzati i colori, altrimenti non vedrà alcun tipo di colore.
[Consiglio di game design: per questo motivo conviene limitare al minimo l'utilizzo di colori, nel caso servano ad evidenziare un particolare importante del testo: piuttosto rifatevi agli stili predefiniti]

Infine l'istruzione **rtrue** (**return true**) che dice al programma di non stampare, oltre al testo che abbiamo specificato, anche il testo di default dell'azione *Examine* compiuta sulle scritte. Questa infatti è l'eventualità che si verifica se specifichiamo l'istruzione *print*. Va indicata sempre al termine del "blocco di azioni" che vogliamo vengano eseguite "contemporaneamente": se la mettessimo pri ma, il programma si interromperebbe nel punto in cui è stata inserita (relativamente all'azione compiuta).
Per capire praticamente come funziona, provate nel codice sorgente proposto a cancellare la riga commentata con 3, poi ad inserire *rtrue* nella riga commentata con 1 e infine a cancellarla anche da quella riga e a metterla in quella commentata con 2.

Un altro modo di indicare *rtrue* congiunto all'istruzione *print* è di scrivere **print_ret** al posto di (dell'ultimo) *print*. Con *print_ret* il programma va a capo in automatico.
In poche parole: queste istruzioni sortiscono il medesimo effetto

a)
```
print "vedi ";
print  "una ";   ! nota che l'a capo nel listato non corrisponde all'a capo del testo stampato
print "scritta:^";
print "Per oggi è tutto^";
rtrue;     ! oppure return true;
```

b)
```
print "vedi"; print "una scritta:^"; print_ret "Per oggi è tutto";
```

c)
```
!  (immaginiamo un with description che precede il testo quotato)
"Per oggi è tutto"
```