In questo articolo vedremo come creare oggetti di scenario e oggetti "invisibili". Vediamo il seguente codice sorgente (attenzione: volutamente "scorretto" in alcuni punti):

```
Constant Story "Esempio di scenari";

Constant Headline "un breve tutorial";

Constant ScenicError "Pensa a qualcos'altro.";

Serial "123456";

Release "1";

Include "Parser";

Include "Verblib";

Include "Replace";

Object foresta "foresta lussurreggiante"

with scenic

'alberi' 0 "Sono alti e molto fitti."

'cespugli' 0 "Chiss@`a se ci crescono bacche o se ci si nasconde qualche animale.",

description "Ti trovi in una rigogliosa foresta, i numerosi alberi e cespugli ti impediscono di avere una visuale completa.^Puoi vedere anche un arco di roccia qui.",

has light;


Object -> arco "arco di pietra"

with scenic

'paletto' 0 "Un paletto a forma di croce",

name 'arco' 'di' 'pietra',

before [;

Examine: give scritta ~concealed; print "Un imponente arco di pietra sormontato da un paletto.^Guardandolo meglio noti anche una scritta."; rtrue;

],

has scenery;


Object -> scritta "scritta"
```

with name 'scritta',

before [;

Examine: print "La scritta dice^^"; style underline; print "Chi legge @`e scemo^^"; style roman; "(spiritosi... - pensi).^";

],

has static concealed;

[Initialise;

location = foresta;

];

Include "Scenic_it";

Include "ItalianG";

Innanzitutto vediamo che abbiamo aggiunto cinque nuove istruzioni all'inizio del codice sorgente:

Constant Story permette di definire il titolo dell'avventura che compare nella prima schermata

Constant Headline definisce il sottotitolo

Serial il numero di serie, che deve sempre essere di sei cifre

Release la versione (questo numero è progressivo e sta al programmatore aumentarlo ogni volta che rilascia una nuova versione per far capire ai giocatori a quale stanno giocando, questo soprattutto se le versioni aggiornate contengono alcuni bug corretti: ad una nuova versione generalmente si allega un file in formato .txt in cui si dice cosa è stato cambiato rispetto alle versioni precedenti in ordine cronologico)

Vediamo anche che abbiamo aggiunto, al termine del listato, un'altra libreria: la Scenic_it. Questa libreria si scarica con un pacchetto di librerie aggiuntive sempre dal sito di Vincenzo Scarpa, per utilizzarla bisogna copiare sia il file Italian.h che il file scenic_it.h nella cartella libraries precedentemente creata (importante: il file Italian.h dovrà sostituire quello già presente nella cartella!), infine copiare il file scenic_it_test.inf nella cartella Inform.

[Nota: non è stato detto finora, ma è usuale (moralmente obbligatorio) scrivere da qualche parte - in un altro articolo vedremo dove - i ringraziamenti a tutte le persone che hanno sviluppato i software coi quali si è creata l'avventura testuale: oltre a ringraziare (in automatico) Graham Nelson per lo sviluppo di Inform 6 e Riccardi per WIDE, vanno ringraziate anche le persone che hanno creato le librerie aggiuntive, in questo caso Barnett, Mason, Firth e Gaburri per la scenic_it, quindi informatevi sempre su chi ha creato ciò che state utilizzando!]

Questa libreria ci permette di fare cose molto interessanti, vediamo quali.

Notiamo che nella descrizione della foresta nominiamo alberi e cespugli, giusto per darle un senso di concretezza. Come specificato in un articolo precedente, è buona cosa, per ogni oggetto nominato, aggiungere almeno una descrizione. Sarebbe però tedioso programmare apposta gli oggetti alberi e cespugli a parte: con la libreria scenic_it è possibile aggiungerli come "scenario". Siccome gli alberi e i cespugli si trovano nella foresta, aggiungiamo dopo il *with* la proprietà *scenic*. Vediamo che gli oggetti vanno dichiarati tra singoli apici: questi sono i nomi ai quali risponderà il programma se digitati dal giocatore (esempio: con questo codice *>esamina piante* non funzionerà), dopodiché, per aggiungere la descrizione, bisogna aggiungere uno 0 e infine il testo che vogliamo compaia a schermo.

Notiamo due cose. La prima: la virgola che termina la proprietà va messa dopo l'ultimo oggetto scenario dichiarato, la seconda: ciascuna "descrizione" può essere resa una routine, come abbiamo fatto negli esempi precedenti per la *initialise*, ad esempio:

Object foresta "foresta lussurreggiante"

with scenic

'alberi' 0 [;

if (condizione_1) "Stampa questo testo"; else "Oppure stampa questo";],

permetterà, una volta digitato *>esamina alberi* di stampare uno dei due testi a seconda che valga la condizione_1 o meno.

Chiaramente, come mostrato dall'arco di pietra, un oggetto di scenario può anche essere dichiarato non necessariamente solo all'interno della locazione: l'oggetto palo è infatti dichiarato nell'oggetto arco di pietra. Avremmo potuto dichiarare il palo anche nella foresta, ma cosa sarebbe successo se al posto di un arco di pietra avessimo avuto un oggetto X trasportabile?

Se avessimo questo oggetto X e non fossimo nella foresta, una volta digitato *>esamina palo* avremmo ottenuto un messaggio di errore perché il palo è stato dichiarato nella foresta, in cui non ci troviamo.

A proposito di messaggi di errore: cosa succede se proviamo ad interagire con gli alberi (ad esempio toccandoli)?

Dal momento che sono scenici, il programma risponde con una risposta di default (qualcosa come "Non è importante ai fini del gioco"). Questa risposta, però, è personalizzabile tramite l'istruzione che non avevo descritta: la *Constant ScenicError*. Attenzione! Questo è un messaggio di default per tutti gli oggetti scenario, quindi cercate di trovare una risposta adeguata applicabile ad ogni oggetto scenografico!

Quindi, per concludere, la proprietà *scenic* permette di creare solo e soltanto una descrizione di oggetti non dichiarati "autonomamente". La presenza di questi oggetti non è "rivelata" dalle descrizioni del programma (a meno che non siano indicate dal programmatore in qualche altra descrizione).

Vediamo poi che l'arco di pietra ha un nuovo attributo: **scenery**. Questo serve per dire al programma di non stampare la sua *initialise* (né quella di default né quella eventualmente dichiarata da noi) assieme alla descrizione dell'ambiente, inoltre lo informa che l'arco di pietra non può essere né preso né spostato. Nel caso il giocatore volesse provarci, sarà data una risposta di default (del tipo "Non hai forza sufficiente").

Lo stesso effetto è sortito dagli attributi congiunti delle scritte: **static** (=l'oggetto non può essere mosso/preso) e **concealed** (="nascondi l'oggetto nella descrizione della locazione). Vediamo che, però, se esaminiamo l'arco di pietra, viene modificato l'attributo *concealed* delle scritte, il che significa che, da quando avremo esaminato l'arco di pietra, vedremo comparire nella stanza anche la descrizione iniziale di default delle scritte ("Puoi vedere delle scritte qui").

Torneremo comunque a parlare dell'attributo *concealed* quando parleremo di come far muovere gli oggetti tra le stanze.

[Nota: per questi tre attributi, anche *scenery*, non è necessaria la libreria scenic_it]

Che differenza c'è quindi tra un oggetto la cui descrizione è dichiarata con la proprietà *scenic* (ad esempio gli alberi) e un oggetto con l'attributo *scenery* (ad esempio l'arco)? La risposta è contenuta nelle righe precedenti: un oggetto con l'attributo *scenery* è un oggetto "indipendente" dalla locazione, nel senso che va creato a parte, e con esso si può interagire in altri modi che solo esaminandolo. E' buona cosa, però aggiungere molti oggetti "scenic" per dare concretezza all'avventura, senza dover necessariamente creare oggetti apposta.

Che differenza c'è invece tra l'usare semplicemente *scenery* oppure *static e concealed* insieme? Questa è una differenza molto sottile e la individuano i giocatori scafati. Inform 6 per alcuni verbi prevede una possibilità d'azione multipla: ad esempio, se in una stanza ci sono diversi oggetti è data la possibilità al giocatore di digitare *>prendi tutto* e di raccogliere gli oggetti che non hanno né l'attributo *static* né l'attributo *scenery,* ma se uno di questi è concettualmente inamovibile verrà anche data una - inutile - risposta del tipo "Il tale oggetto non può essere preso". Se immaginiamo di aver programmato una stanza con anche solo cinque di questi oggetti, leggere cinque volte che un oggetto non può essere preso può essere fastidioso. Per evitarlo bisogna aggiungere agli oggetti per i quali non si vuole questa risposta l'attributo *concealed*. (Il consiglio è quindi quello di aggiungere comunque l'attributo *concealed* a qualsiasi oggetto abbia *scenery*)

Ad esempio, col codice sorgente proposto, se provassimo a digitare *>prendi tutto* vedremmo comparire "Arco di pietra: non può essere preso" (non così per le scritte, che hanno l'attributo *concealed*). Quest'ultima è una finezza di game design, se non è chiara provate a modificare nel codice sorgente i vari attributi per vedere cosa producono le loro combinazioni.

Anche per oggi è tutto, per stavolta nessun esercizio, dal momento che gli effetti degli attributi visti possono essere ottenuti in mille modi diversi (anche "illeciti").