Un enigma molto comune nelle avventure testuali è quello di bloccare la strada al giocatore in corrispondenza di una porta che necessita di una chiave per essere aperta. Vediamo quindi come implementare questa situazione tramite il seguente listato:

```
Constant Story "Creazione di porte";
Constant Headline "un esempio";
Include "Parser";
Include "Verblib";
Include "Replace";

Object chiave_magica "chiave"
with name 'chiave',
description "Una normalissima chiave.",
has female;

Object stanza_verde "una stanza colorata di verde"
with description "Vedi solo verde, verdi le pareti, cos@`i come verde @`e una porta incassata.",
n_to porta_magica,
has light;

Object -> porta_magica "porta"
with name 'porta',
description [; print "Una normalissima porta "; if(player in stanza_verde) print "verde"; else print "blu";
print ".^"; rtrue; ],
when_closed "Una porta ti sbarra la strada.",
when_open "La porta che ti sbarrava la strada @`e aperta.",
door_to [;
 if (self in stanza_verde) return stanza_blu; return stanza_verde;
! il self, per quanto visto negli articoli precedenti, si riferisce all'oggetto in cui compare
! in questo caso però si può interpretare self come player (vedi più sotto)
! dal momento che l'oggetto, di fatto, "viene a trovarsi" nella stessa locazione del giocatore
! (laddove questa locazione compaia nella proprietà found_in)
! # l'ultima istruzione appena sopra, per quanto mancante l'else, è un if else * #
```

],

door_dir [; if (self in stanza_blu) return n_to; return s_to;],

with_key chiave_magica,

found_in stanza_verde stanza_blu,

has female static door openable lockable locked;

Object stanza_blu "una stanza colorata di blu"

with description "Vedi solo blu, blu le pareti, cos@`i come blu @`e una porta incassata.",

s_to porta_magica,

has light;

[Initialise;

location = stanza_verde;

move chiave_magica to player;

];

Include "ItalianG";

Cominciamo col dire che molto di ciò che segue dipende dalla proprietà *found_in*, ma che tuttavia non è assolutamente necessaria dal momento che gli effetti di queste istruzioni possono essere ottenuti in altri modi, sebbene più complicati.

Notiamo poi che le due stanze che abbiamo creato non sono collegate l'una all'altra, ma alla porta. Per il momento teniamolo a mente.

Analizziamo la proprietà **found_in**. Vediamo che secondo l'albero degli oggetti la porta dovrebbe comparire solamente nella stanza_verde, con questa proprietà invece è possibile dislocarla anche in altre locazioni, nella fattispecie la stanza_blu. Questo è un meccanismo utile per evitare di dichiarare lo stesso oggetto per ogni locazione in cui compare [per capire l'utilità di questa proprietà: nel manuale originale si fa vedere come è possibile creare una nebbia che compare in più locazioni tramite questo metodo]. Da notare che tuttavia non basta questa proprietà per "inserirlo" in una locazione: bisogna lo stesso specificare una locazione nella dichiarazione dell'oggetto (ovvero *Object -> porta_magica "porta"* oppure *Object porta_magica "porta" stanza_verde*, senza la quale altrimenti la porta non comparirebbe in nessuno dei luoghi voluti). Con la proprietà *found_in* è possibile agire sull'oggetto in qualunque locazione compaia (ad esempio: se per questa porta non avessimo dichiarato questa proprietà, essa sarebbe comparsa solamente nella stanza_verde: non è prettamente un errore, ma teniamo comunque presente che una porta generalmente collega due stanze, quindi a logica dovrebbe comparire in entrambe). Vediamo anche che nella proprietà *found_in* i nomi delle locazioni non sono separati da nulla, se non da uno spazio.

Ad una porta, tramite l'attributo **door** possono essere associati efficacemente alcuni verbi preimpostati, quali **lock**, **unlock** (blocca, sblocca), che possono essere liberamente modificati, ma vedremo in un altro momento come, nonché le proprietà che vedremo in questo articolo. Notiamo comunque che questi verbi compaiono anche nella forma di attributi: **lockable** indica che un oggetto può essere bloccato/sbloccato, **locked** che effettivamente il suo stato iniziale è bloccato. Attenzione: tra i vari attributi ricordarsi sempre di associare ad una porta *openable* (che fino ad ora avevamo visto solo per i contenitori), altrimenti si crea chiaramente un disastro!!

Quindi - lo diciamo adesso ampliando l'articolo sui contenitori - verrebbe da dire che una "porta" *(door*), secondo Inform 6, è simile a un "contenitore" (*container*). Ebbene sì, dal momento che entrambi possono essere bloccati/sbloccati, aperti/chiusi e ad entrambi quindi può essere associata la proprietà *with_key*, ma, più in generale, **per Inform 6 una porta è tutto ciò che, non essendo locazione, collega un posto ad un altro** (ad esempio una scala o una corda appesa tra due finestre) [vedi esercizio alla fine].

La differenza tra bloccato e chiuso è che quando una porta (contenitore) è bloccata significa che serve qualcosa (generalmente una chiave) per aprirla, mentre quando è chiusa non è detto che sia bloccata, quindi quando si progetta una porta ricordarsi che la sequenza temporale è sblocca->apri (quando si decide che una porta può essere bloccata). A proposito di aperto/chiuso, ricordiamo che il contrario di *open* è **~open**.

Vediamo che la porta ha le proprietà **when_open** e **when_closed**: queste permettono di modificare automaticamente l'*initial* della porta senza creare una routine apposta. Il risultato sarà una descrizione differente della porta quando viene stampato il testo della locazione, a seconda che essa sia aperta o chiusa. Un accorgimento non indispensabile ma che comunque può aumentare la riuscita estetica dell'avventura. Chiaramente queste proprietà si possono aggiungere per gli oggetti dotati dell'attributo *openable*.

La proprietà **door_to** (che, assieme alla *door_dir* è l'unica che, per essere dichiarata, necessita che l'oggetto cui appartiene abbia l'attributo *door*) serve per informare il programma in quale punto cardinale si trova la porta. Anche in questo caso è una proprietà che ci semplifica la vita: senza di essa avremmo dovuto creare una routine nella istruzione *n_to* della stanza_verde come la seguente:

Object stanza_verde "stanza"

…

n_to [; if (porta_magica has open) return stanza_blu; else "Dovresti prima aprire la porta.";],

…

il che è decisamente macchinoso. Notiamo però che per produrre l'effetto della *door_to* è necessario, come specificato all'inizio, collegare le due stanze non tra di loro ma alla porta. Inoltre il *return stanza_blu* è un efficacissimo modo di sostituire il *PlayerTo()* visto qualche articolo fa.

E infine l'ultima proprietà: la **with_key**, che ci permette di evitare di modificare il verbo *unlock* per la porta in un modo simile a quello fatto per la direzione *n_to*. Non mi soffermerò su come modificare questo verbo per mostrare le differenze, dal momento che aprirebbe un capitolo vastissimo che è bene approcciare in altro modo e senza altri concetti già nella testa.

Per il momento basta solo sapere che il verbo *unlock* risponde ai comandi *> sblocca* e *> apri OGGETTO con OGGETTO*. L'effetto sortito sarà efficace se e solo se il giocatore possiederà l'oggetto corrispondente, nella fattispecie la chiave_magica.

Per concludere, vediamo nella routine *Initialise* un'istruzione già introdotta in un esercizio di qualche articolo fa, ovvero

move OGGETTO1 to OGGETTO2;

che permette di spostare l'oggetto1 "all'"/"nell'" oggetto2, con la specifica che in questo caso l'oggetto2 è proprio il giocatore (in Inform 6 ci si riferisce sempre all'oggetto *giocatore* chiamandolo **player**). Questa istruzione, naturalmente, può essere messa praticamente ovunque, il fatto che sia messa nella routine Initialise permette di farla eseguire all'inizio del gioco, ovvero: il giocatore avrà già nell'inventario (richiamabile durante il gioco coi comandi *> i* e *> inventario*) la chiave.

L'istruzione "inversa" del *move to* è la **remove**, che permette di eliminare dall'albero degli oggetti un qualunque oggetto [nota bene: l'oggetto non viene dissolto, viene semplicemente messo in un "limbo" dal quale può eventualmente essere ripescato in futuro, ad esempio tramite una successiva istruzione *move to*: se, ad esempio, avessimo evitato di dichiarare *move chiave_magica to player;* nella *Initialise*, la chiave_magica si troverebbe proprio in questo "limbo" (notare che non è stata indicata nessuna locazione di appartenenza, né esplicitamente né con la freccia, nella dichiarazione dell'oggetto chiave_magica!)].

[Già che ci siamo: un'altra nota di game design preannunciata qualche articolo fa - forse al momento ancora un po' troppo complicata, ma che probabilmente riprenderò anche in seguito. Immaginiamo di aver creato un'avventura testuale in cui qualche oggetto è "invisibile" in una stanza e la sua presenza viene scoperta solo dopo aver risolto qualche enigma: abbiamo visto che il meccanismo da utilizzare è l'attributo *concealed* (che verrà tolto tramite l'istruzione *give OGGETTO ~concealed;* in seguito alla soluzione di quegli enigmi). Perfetto. Cosa succederebbe, però, se lo stesso giocatore rigiocasse una seconda volta l'avventura testuale? Potrebbe ricordarsi che quell'oggetto è già presente, per quanto "invisibile", e magari scegliere di interagire con esso prima di aver risolto gli enigmi da noi pensati. Questo potrebbe non intaccare l'avventura, ma in generale è bene prevenire questa situazione. Esiste un modo per evitare che ciò accada, ovvero creare una variabile apposta che si "attiva" solo dopo che abbiamo risolto gli enigmi, e l'"attivazione" di questa variabile permette anche l'interazione con l'oggetto. Questo metodo funziona, ma lo sconsiglio per due motivi: il primo è perché non abbiamo ancora visto le variabili, il secondo perché dovremmo verificare se l'"attivazione" di quella determinata variabile impostata da noi è vera o falsa per ogni possibile interazione con l'oggetto: chiaramente se le possibilità di interazione sono tante dovremo scrivere nel listato molti *if else*. Un altro modo di risolvere il problema è, anziché dare l'attributo *concealed* all'oggetto, quello di dichiararlo al di fuori dell'albero degli oggetti ed utilizzare l'istruzione *move OGGETTO to LOCAZIONE;* in seguito alla risoluzione degli enigmi da noi pianificati. Questo chiaramente è inutile se si pensa che la nostra avventura non verrà mai rigiocata una seconda volta, ma: chi lo vuole pensare? Perciò si può utilizzare (anche) questo meccanismo per aumentarne la longevità].

Per concludere, un esercizio ripreso direttamente dal manuale originale di Graham Nelson, la cui soluzione è da me opportunamente modificata compatibilmente con le conoscenze che ho introdotto fino ad ora:

Creare un ponte di assi su un precipizio [che si estende da "sponda vicina" a "sponda lontana"] che collassa se il giocatore tenta di camminarci possedendo qualche oggetto.

[aiuto: la condizione "se il giocatore possiede qualche oggetto" si traduce con *if(children(player)>0)...*]

[Notare, come detto nell'articolo, che può tornare utile considerare un oggetto concettualmente come una porta!

Naturalmente, se dovessimo permettere di compiere una qualche azione sul ponte, dovremmo dichiararlo come una locazione, ma non è questo il caso]

*Come visto per il *return true*, ogni volta che si arriva ad eseguire un *return* l'istruzione in cui esso è contenuto viene comunque terminata. Il che significa che se il giocatore è nella stanza_verde (cioè se la condizione dell'*if* è vera) si esegue il *return stanza_blu* e l'istruzione viene terminata, venendo ignorato anche ciò che segue nella stessa proprietà, se invece il giocatore non è nella stanza_verde (cioè se la condizione dell'*if* è falsa) non viene eseguito il *return stanza_blu* (come ci si aspetta) e viene (come di norma) eseguito ciò che segue, ossia *return stanza_verde*: questo è proprio il risultato dell'if else, la differenza è che per la peculiarità delle istruzioni in questo specifico caso non è stato necessario specificare l'*else*.