Come promesso, in questo articolo vedremo come attribuire un punteggio negativo e come assegnare un "rango" agli avventurieri. Per fare ciò avremo bisogno di rimpiazzare due routines incluse nella libreria Verblib, come è successo nell'articolo precedente quando abbiamo modificato la funzione *PrintTaskName*. Cominciamo col vedere i **punteggi negativi**.

```
Constant TASKS_PROVIDED;
Constant NUMBER_TASKS = 2;
Constant MAX_SCORE = 1;
Array task_scores --> 1 (-1);

Replace Achieved;

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

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

Object -> cranio "teschio"
with name 'teschio' 'cranio',
initial "Puoi vedere un cranio.^Se lo raccogli guadagni un punto.^Se lo posi perdi un punto.",
description "Un cranio antico.",
after [;
Take: Achieved(0);
Drop: Achieved(1);
];

[ Achieved num;
if (task_done->num==0)
  {  task_done->num=1;
     score = score + task_scores-->num;
```

```
    }
];

[ Initialise;
location = stanza;
];

Include "ItalianG";
```

Vediamo che la libreria è stata modificata tramite l'istruzione **Replace Achieved;** che va dichiarata **prima** di *Include "Verblib";*. La seconda differenza è che, tramite questo rimpiazzo, l'array *task_scores* ha due "trattini" nella "freccia" al posto di uno. I numeri negativi vanno indicati tra parentesi.
Passiamo quindi ad esaminare la routine **Achieved**. [Grazie a Firth!]
task_done è una variabile che indica se una determinata azione è stata eseguita. Questa variabile è la stessa che "capisce" quando un'azione che aumenta il punteggio è già stata compiuta e, nel caso, evita di assegnare nuovamente il punteggio all'azione se ricompiuta. Questa variabile è booleana, ossia o è vera (cioè ha valore 1) o è falsa (cioè ha valore zero). **num** come al solito è una variabile di comodo che abbiamo introdotto apposta, e corrisponde all'indice dell'array che compare nell'istruzione *Achieved();*
Vediamo praticamente cosa succede.
Se si prende il cranio si ha *Achieved(0)*, quindi adesso *num* vale 0. Viene richiamata immediatamente (e automaticamente) la routine *Achieved*. Ora nella routine *Achieved* c'è una condizionale: se la variabile *task_done* relativa alla posizione *num* (= 0) è falsa (e lo è di default) la fa diventare vera, in più aumenta il punteggio del valore contenuto nella posizione 0 dell'array *task_scores*. Se è falsa non succede nulla (ecco perché il punteggio viene incrementato una sola volta!).
Vediamo che abbiamo assegnato un punteggio (negativo) anche quando il cranio viene posato. Si ripete esattamente lo stesso procedimento, ma questa volta *num* ha valore 1, la *task_done* (che questa volta non è più quella relativa a num = 0, ma a num = 1) è falsa ecc...
Quello che andrò a dire adesso dovrebbe risultare ovvio, nel caso non lo fosse provate a giocare l'avventura ricavata da questo codice sorgente: se si prende il cranio, lo si posa e lo si riprende, poiché la variabile *num* (di 0) quando si RIprende il cranio è vera, il punteggio non verrà aumentato un'altra volta. Questa potrebbe essere una situazione non soddisfacente. Prendiamo come esempio l'avventura Zork: scopo dell'avventura è raccogliere diversi manufatti da mettere in una bacheca, tuttavia molto spesso capita di doverli lasciare. Se la routine *Achieved* fosse quella indicata in questo codice sorgente, Zork non potrebbe mai venire completato a punteggio pieno.
Questa sarà una delle poche volte in cui mostrerò una soluzione al problema, dal momento che non è univoca e inoltre sta al genio del programmatore capire quali problemi può dare il proprio codice sorgente e come risolverli.
Proviamo a modificare la routine in questo modo:

```
[ Achieved num;
! prima parte
if (task_done->0==0)
  {  task_done->0=1;
```

```
    task_done->1=0;
    score = score + task_scores-->num;
  }
! seconda parte
if (task_done->1==0)
 {  task_done->1=1;
    task_done->0=0;
    score = score + task_scores-->num;
  }
];
```

Sono il primo ad ammettere che si tratta di una soluzione bovina (non si può nemmeno definire "poco elegante"), ma è giusto per illustrare il concetto. Vediamo che abbiamo "spezzato" la condizionale in modo da farla funzionare diversamente a seconda che *num* valga 0 o 1. Se si prende il cranio *num* vale 0 e quindi viene eseguita la prima parte. L'unica aggiunta fatta è l'istruzione *task_done->1=0;* che è superflua ad inizio partita, essendo già 0 il suo valore di default. Vediamo però cosa succede in questa sequenza (facendo finta che non ci sia la seconda parte):
> prendi cranio
- *task_done->0* diventa 1
- *task_done->1* diventa (in realtà rimane 0)

> lascia cranio
- *task_done->0* vale 1
- *task_done->1* diventa 1

> prendi cranio
- *task_done->0* rimane 1
- *task_done->1* RIdiventa 0 grazie alla nuova istruzione

cioè: alla prossima volta che lasciamo il cranio il punteggio verrà nuovamente decurtato, infatti se non avessimo aggiunto l'istruzione *task_done->1* avrebbe avuto ancora il valore uno.
La stessa cosa vale esattamente per la seconda parte.
Perché questa istruzione è poco funzionale?
Per almeno due motivi:
il primo: se avessimo anche solo un altro valore nell'array *task_scores* avremmo dovuto **necessariamente** esplicitare nella routine *Achieved* il caso in cui *num* sia diverso da 0 o 1, e si capisce bene che per un array *task_scores* con molti elementi la cosa diventa macchinosa
il secondo: questo è un procedimento che vale solo per un oggetto, mentre in Zork questo processo funziona su numerosissimi oggetti.

Per finire riguardo al punteggio, si può anche scegliere di **non** mostrare al giocatore quando guadagna un punto nelle descrizioni visualizzate durante il gioco: questo si ottiene inserendo nella routine *Initialise* l'istruzione **notify_mode = false;** (i cambiamenti di punteggio verranno però, ovviamente, mostrati nella barra in alto - vedi prossimo codice sorgente).

Veniamo ora all'indicazione del rango:

```
Constant TASKS_PROVIDED;
Constant NUMBER_TASKS = 3;
Constant MAX_SCORE = 3;
Array task_scores -> 1 1 1;


Replace PrintRank;


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

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

Object -> cranio "teschio"
with name 'teschio' 'cranio',
initial "Puoi vedere un cranio. Se lo raccogli guadagni un punto.",
description "Un cranio antico.",
after [;
Take: Achieved(0);
];


Object -> doblone "doblone"
with name 'doblone',
initial "Puoi vedere un doblone. Se lo raccogli guadagni un punto.",
description "Un doblone antico.",
after [;
Take: Achieved(0);
];
```

```
Object -> pugnale "pugnale"
with name 'pugnale',
initial "Puoi vedere un pugnale. Se lo raccogli guadagni un punto.",
description "Un pugnale antico.",
after [;
Take: Achieved(0);
];

[ PrintRank;
print "^";
if(score==0) print "Sei ancora alle prime armi";
if(score==1) print "Sei sulla buona strada per diventare un avventuriero";
if(score==2) print "Continua cos@`i!";
if(score==3) print "Sei il pi@`u grande avventuriero testuale!";
print "^";
rtrue;
];

[ Initialise;
location = stanza;
notify_mode=false;  ! non viene detto quando si conquistano punti
];

Include "ItalianG";
```

Vediamo che in questo caso la parte di libreria da modificare è la PrintRank, e la corrispettiva istruzione va messa, al solito, prima di Include "Verblib";
La routine non è nulla di trascendentale: si tratta di scegliere il rango da attribuire ad ogni punteggio e mettere le istruzioni condizionali in modo opportuno. In questo esempio il punteggio non varia di molto, essendo presenti solo tre valori nell'array *task_scores* ed essendo il punteggio massimo di 3, quindi possiamo permetterci di inserire tre istruzioni condizionali. Nel caso ci fosse molta più variabilità possiamo, anziché utilizzare l'operatore **==,** utilizzare < e > (rispettivamente "minore" e "maggiore") oppure <= e >= (rispettivamente "minore o uguale" e "maggiore o uguale"), ricordandoci

però sempre di includere tutte le possibilità di punteggio. Questo, ad esempio, è il codice relativo della mia ultima avventura testuale:

```
[PrintRank;
 print "^^Il tuo rango @`e:^";
 if (explorer < 5) print "Iniziato.^";
 if (explorer > 4 && explorer < 15) print "Novizio.^";
 if (explorer > 14 && explorer < 20) print "Apprendista.^";
 if (explorer > 19 && explorer < 26) print "Esploratore Abile.^";
 if (explorer > 25 && explorer < 33) print "Gran Cercatore.^";
 if (explorer > 32 && explorer < 40) print "Allievo Maestro.^";
 if (explorer > 39 && explorer < 47) print "Maestro dell'Esplorazione.^";
 if (explorer == 47) print "Gran Maestro dell'Esplorazione.";
print "^";
rtrue;
];
```

Come potete vedere, inoltre, in questo codice non compare la variabile *score* ma la variabile *explorer* (che va da un minimo di 0 a un massimo di 47), una variabile che ho personalmente definito in modo da stampare il rango non relativo al punteggio ottenuto nel corso dell'avventura ma relativo ad altre azioni che possono essere compiute e che, al posto di incrementare il punteggio, incrementano (senza dichiararlo) questa stessa variabile [nota di game design: questo aumenta la longevità dell'avventura!].
[Altra nota che potevo anche inserire prima: tutti gli esempi che faccio sono per far conoscere le istruzioni e capire come utilizzarle, sta poi al programmatore capire in quale modo creativo usarle, svincolandosi dagli esempi fatti, per far risultare la propria avventura originale!]
L'unica cosa degna di nota è che in questo caso è necessario specificare l'istruzione *print* per stampare i rispettivi testi.

Per gli esercizi relativi a questa parte ho pensato di non includere screenshots né proporre particolari problemi perché sarebbero risultati entrambi pesanti: produrrò invece un'avventura testuale giocabile nel mio sito e voi dovrete provare a creare un codice sorgente che dia gli stessi risultati, quindi se volete, quando vi capita, venite a fare una visita su http://www.mcbones.altervista.org/imparare-inform-6.html