Note di riferimento a

MINI INFORM

Versione 1.0

15 Gennaio 2007

Sommario:

Introduzione.

Mini Inform è una libreria per creare giochi scritti in Inform 6. E' definita "Mini" perché rispetto alla libreria base di Nelson è assai più piccola e concentrata. Questo non significa che sia incompleta, ma semplicemente che prevede un Word Model assai più semplificato di quello classico.

Normalmente in una avventura testuale nello stile Infocom, un giocatore ha (teoricamente) un campo vastissimo di azioni potenzialmente utili. Il fatto che poi nella pratica difficilmente un autore prevede tutte le possibile azioni non ha importanza, ciò che conta è l'impressione che trae il giocatore.

Mini Inform al contrario offre al giocatore solo poche azioni oltre a quelle di sistema, ovvero: ESAMINA, AGISCI e PARLA. In questo si avvicina molto alle avventure grafiche, l'interazione con un oggetto o coi i personaggi è limitata dal mouse e cambia a seconda del contesto e delle situazioni previste dall'autore.

Perché rinunciare alla libertà che concede la tastiera? La risposta è semplice, se la tastiera concede una flessibilità enorme nell'interazione uomo-macchina, inserire comandi su piattaforme tipo le console, i cellulari o i palmari che non la possiedono (o ne hanno solo dei surrogati) è uno strazio.

Mini Inform viene incontro all'autore che desideri creare una avventura facilmente giocabile su tali piattaforme. Il giocatore, infatti, potrà interagire con l'ambiente virtuale digitando UNA SOLA parola per ogni comando, o in alternativa tramite un CLICK del mouse o della penna ottica.

Per farvi una idea giocate alla "Contessa di karolystria" o provate l'avventura di esempio "Cloak of Darkness".

I file di libreria.

mininf.h

è il cuore della libreria qui sono definiti gli oggetti, le proprietà gli attributi ed il ciclo base del gioco.

routines.h

contiene funzioni utili alla libreria base.

minitalk.h

prevede la gestione dei dialoghi a scelta multipla con i personaggi non giocatori.

mininpc.h

prevede la gestione degli spostamenti dei personaggi non giocatori.

Allegati ai su indicati file ed al presente manuale vi sono anche due esempi di codice (*cloak.inf* e *npc_move_eg.inf*).

Importare le librerie.

Per creare un gioco con Mini Inform è necessario includere nel proprio sorgente il file *mininf.h* più o meno all'inizio di esso ed il file *routines.h* invece verso la fine. Pur non essendo indispensabile si consiglia di inserire anche gli altri due file che facilitano la gestione dei personaggi non giocatori.

Esempio:

```
!% -switch di compilazione
...
include “mininf.h”;
include “minitalk.h”;
      include “mininpc.h”;
      ...
      codice del gioco
      ...
      include “routines.h”;
```

Tutorial: Cloak of Darkness

Questa sezione si occuperà di mostrare come creare un piccolo gioco in Inform utilizzando le Mini Librerie. L'autore dovrebbe conoscere già Inform, le sue istruzioni base e la punteggiatura, ma non c'è bisogno della conoscenza della libreria di Nelson. Per chi non conoscesse le basi di inform è sufficiente leggere i primi paragrafi della Guida a Inform per Principianti.

Cominciamo con il definire la funzione `init()`, indispensabile per qualsiasi gioco con le minilibrerie:

```
[Init ;
    location = foyer;
    move cloak to player;
    print "^^Hurrying through the rainswept November
       night, you're glad to see the bright lights of
       the Opera House. It's surprising that there
       aren't more people about but, hey, what do you
       expect in a cheap demo game...?^^";
    print "CLOAK OF DARKNESS^";
    print "Implementation by Ignazio Di Napoli and Marco
       Falcinelli^";
];
```

Niente di particolarmente difficile, l'unica cosa veramente necessaria è la definizione della locazione iniziale del gioco, assegnata alla variabile `location`.
La seconda istruzione sposta il mantello nell'inventario del giocatore, e quindi vengono stampate alcune righe di introduzione.

Passiamo ora a definire la prima locazione del gioco:

```
Room    foyer "Foyer of the Opera House"
  with  description
              "You are standing in a spacious hall,
              splendidly decorated in red and gold, with
              glittering chandeliers overhead. The
              entrance from the street is to the north,
              and there are doorways south and west.",
        scenic 'gold' 'red' 0 "A splendid decoration!^"
               'chandeliers' 0 "Glittering and shining.",
        cant_go "You only can go to south or west.^",
        directions
             'south' 'bar' 0 bar
             'west' 'cloakroom' 0 cloakroom
             'north' 'out' 0
                "You've only just arrived, and besides,
                the weather outside seems to be getting
                worse.^";
```

essa mostra un esempio dell'uso delle proprietà `description`, `scenic`, `cant_go`, `directions`. Fate presente che i vocaboli definiti i `scenic` sono solo esaminabili e non sono veri e propri oggetti, tale proprietà infatti è pensata per evitare di creare oggetti a non finire per tutti quei particolari di scena per i quali vogliamo definire solo una descrizione. Notate che per le direzioni viene definito oltre che il punto cardinale (obbligatorio) anche il nome della locazione di destinazione, il giocatore potrà quindi spostarsi verso il bar semplicemente digitando "bar" o "south" o anche attraverso l'abbreviazione "s" (prevista dalla funzione `languageToInformese` nella libreria).

Definiamo ora le altre due locazioni presenti in questo semplice esempio:

```
Room    cloakroom "Cloakroom"
  with  description
              "The walls of this small room were clearly
```

```
                once lined with hooks, though now only one
                remains. The exit is a door to the east.",
        directions
             'east' 'foyer' 0 foyer;

Room    bar "Foyer bar"
  with  description [;
                "The bar, much rougher than you'd have
                guessed after the opulence of the foyer to
                the north, is completely empty. There
                seems to be some sort of message scrawled
                in the sawdust on the floor.";
        ],
        enter [;
            if (cloak notin player && message notin bar)
                move message to bar;
        ],
        exit [;
            if (cloak notin hook) message.tag++;
        ],
        directions
            'n//' 'north' 'foyer' 0 foyer,
  has ~light;
```

Le uniche novità sono la definizione delle proprietà `enter` ed `exit` della locazione `bar`, e l'associazione ad essa dell'attributo `~light`. Le prime due vengono chiamate nel momento in cui il giocatore entra nel bar o ne esce. L'attributo invece fa del bar una locazione buia, ciò significa che se il giocatore entra nel bar senza possedere una sorgente di luce (o se tale attributo nella locazione non venga cambiato) egli si ritroverà non nella locazione bar ma nella locazione `darkness` (oscurità). La variabile `location` sarà `darkness`, la variabile `real_location` sarà `bar`. A differenza delle librerie di Nelson, nella locazione buia è permesso spostarsi e le direzioni valide sono quelle previste nella locazione reale, tutte le altre proprietà della locazione reale invece non vengono richiamate fintanto che ci si trova nell'oscurità.

Vi ricordate che all'inizio abbiamo dato un mantello al giocatore? Ecco l'oggetto che lo rappresenta:

```
Object  cloak "velvet cloak"
   with
        name 'handsome' 'dark' 'black' 'velvet' 'satin'
                                                'cloak',
        description
            "A handsome cloak, of velvet trimmed with
            satin, and slightly spattered with raindrops.
            Its blackness is so deep that it almost seems
            to suck light from the room.",
         useWith [other;
             if (other == hook) {
                    move cloak to other;
                    give bar light;
                    print "You put the cloak on the hook.";
                    rtrue;
             }
            if (other == nothing) move self to player;
         ];
```

L'oggetto permette di dimostrare l'utilizzo della proprietà `useWith`, essa viene richiamata sia quando si tenta di agire sull'oggetto, sia quando si tenta do combinarlo con un altro oggetto. L'azione su un oggetto è determinata da un comando del tipo "usa oggetto" o più semplicemente dal comando "oggetto" se quest'ultimo possiede l'attributo `seen`, attributo che viene dato ad ogni oggetto una volta che sia stato esaminato dal giocatore. La variabile locale `other` dice alla proprietà se è presente il tentativo di combinare l'oggetto con un altro attraverso un comando del tipo "oggetto1 oggetto2". Nel nostro caso appendere il mantello al gancio accende la luce nel bar (!). Andiamo a vedere l'oggetto gancio:

```
Object  hook "small brass hook" cloakroom
  with  name 'small' 'brass' 'hook' 'peg',
        description [;
            print "It's just a small brass hook, ";
```

```
            if (hook == parent(cloak))
                "with a cloak hanging on it.";
            "screwed to the wall.";
        ],
  has   scenery;
```

Di nuovo esso possiede unicamente l'attributo `scenery`, che dice al programma di non stampare la sua descrizione nella locazione, l'oggetto infatti è già citato nella descrizione della locazione stessa.

Un ultimo oggetto mostra l'utilizzo della proprietà generica `tag`, ed incidentalmente prevede anche l'uscita dal gioco.

```
Object  message "scrawled message"
  with  name 'message' 'sawdust' 'floor',
        description [;
            if (message.tag < 2) {
                print "The message, neatly marked in the
                      sawdust, reads...";
                print "^YOU WON!";
                @quit;
            } else {
                print "The message has been carelessly
                      trampled, making it difficult to
                      read. You can just distinguish the
                      words...";
                print "^YOU LOST!";
                @quit;
            }
        ],
        tag  0
  has scenery;
```

A questo punto il gioco è già giocabile. Nell'esempio è stato inserito però un personaggio (robot) per mostrare l'utilizzo di alcuni altre proprietà della mini libreria.
Andiamo ad analizzarne il codice:

notiamo innanzi tutto che è stato definito come appartenente alla classe Character. Questo per approfittare dell'estensione per i dialoghi a scelta multipla.

```
Character robot "a robot" foyer,
    with
        name 'robot' 'button',
        description "A robot. It's still and silent. It
                             has a button on it.",
        describe "In a corner stands a robot.",
```

notiamo la definizione della proprietà `describe`, essa fa in modo che il robot sia citato nella descrizione attraverso una frase appesa alla descrizione della stanza e non nell'elenco degli oggetti che si possono vedere in essa come normalmente accade nelle avventure testuali.

```
        useWith [other;
             if (other == nothing) {
                 if (self.switchedon == 0) {
                     self.switchedon = 1;
                       "You push the button. The robot is
                       now on. Maybe you can talk to it
                       now.";
                 } else {
                     self.startTalk();
                     rtrue;
                 }
             }
         ],
        switchedon 0,
```

la particolarità di questa `useWith` è che alla prima chiamata si agisce sul personaggio schiacciando il bottone, alla seconda di attiva un dialogo a scelta multipla. A tal proposito andiamo a vedere la definizione delle opzioni:

```
        SayQ [line; switch (line) {
             0: "Good evening.";
             1: "Yes, thank you.";
             2: "No, thank you.";
```

```
    3: "Goodbye.";
}],
```

e quella delle risposte:

```
Respond [line; switch (line) {
    0: self.QuipsOn(2, 1, 2); self.QuipOff(0);
       "Good evening to you. May I take your
       cloak?";
    1: self.QuipsOff(2, 1, 2);
       print "I'll take it in the cloakroom.";
       move cloak to robot;
       give bar light;
       self.nextThink = now + 1;
       rfalse;
    2: self.QuipsOff(2, 1, 2);
       "You can leave it in the cloakroom if you
       want to.";
    3: print "Goodbye.";
       rfalse;
}],
```

notate come le risposte comprendano istruzioni per attivare e disattivare le opzioni di dialogo e come l'istruzione `rfalse` faccia cessare il dialogo mentre il restituire `true` lo faccia continuare.
La seguente proprietà definisce quali opzioni sono attive inizialmente.

```
InitQuips [; self.QuipsOn(2, 0, 3); ],
nextThink 0,
step 0,
think [;
    switch (self.step) {
      0:
        if (parent(player) == parent(self))
           print "You see the robot going
                      towards the cloakroom.^";
        move self to cloakroom;
        self.nextThink = now + 1;
        self.step++;
      1:
        if (parent(player) == parent(self))
           print "The robot hangs your cloak to
                                    the hook.^";
        move cloak to hook;
        self.nextThink = now + 1;
```

```
            self.step++;
         2:
            if (parent(player) == parent(self))
               print "The robot is going back to the
                                          foyer.^";
            move self to foyer;
            if (parent(player) == foyer)
               print "The robot is coming back.^";
      }
   ];
```

non ci rimane che analizzare la proprietà `think`. Essa viene richiamata ad ogni turno se non è prevista la proprietà `nextThink` o se questa è diversa da zero. Nel nostro caso `nextThink` viene posta diversa da zero all'attivazione dell'opzione 1 del dialogo a scelta multipla. Sarà solo allora che `robot.think` verrà eseguita. Alla fine di ogni esecuzione di `think`, `nextThink` verrà posta pari a zero se non diversamente specificato. Nel caso del robot una volta attivata la proprietà richiede di essere attivata altre due volte nel turno successivo e quindi si ferma. Notate come `nextThink` venga posta pari a `now` (turno corrente) più uno, ovvero l'attivazione è prevista per il turno successivo. `Now + 4` avrebbe attivato `think` solo quattro turni dopo.

Qui termina l'analisi di questo breve gioco, al lettore si lascia come esercizio, l'analisi dell'altro esempio allegato al manuale che mostra l'uso degli spostamenti dei personaggi. All'autore sarà utile poi come riferimento i successivi paragrafi che elencano le varie caratteristiche della libreria.

Le azioni.

Ogni azione disponibile nel gioco è rappresentata da un oggetto. Le azioni previste dalla libreria sono:
SAVE, LOAD, RESTORE, RESTART, UNDO, INVENTARIO, ASPETTA, ESAMINA, USA, PARLA.
Sono inoltre previste altre azioni utili al beta test del gioco: RECON, RECOFF, REPLAY. Essi servono rispettivamente a registrare i comandi inseriti, a fermare tale registrazione e a reinserire automaticamente tutti i comandi registrati.

Le direzioni e gli spostamenti.

Le direzioni e gli spostamenti vengono gestiti dall'autore attraverso la definizione della proprietà `directions`, secondo il seguente schema:

```
Room CENTRO "il centro"
   with
     directions
      'south'     0 "Non puoi andare a sud."
      'west'      0 locazioneOvest,
      ;
```

ovvero la direzione tra apici, uno zero e il nome identificativo della stanza di destinazione od un messaggio alternativo (ma anche una routine) se non si vuole permettere tale spostamento.
Le direzioni base previste dalla libreria sono definite nell'oggetto `obj_directions`.
E' necessario e sufficiente per le direzioni usare i termini in inglese definiti in tale oggetto, i sinonimi e le abbreviazioni saranno gestiti dalla routine `LanguageToInformese`.

Gli eventi.

Gli eventi vengono gestiti attraverso la proprietà `Think`, essa può essere richiamata ad ogni turno o come un timer fatta partire ad un certo punto e poi fermata in un altro. Si veda l'esempio "Cloak of Darkness" per un esempio.

Il ciclo base.

Ecco come si comporta la libreria durante un qualsiasi gioco:

1) All'avvio inizializza la variabile dei turni, i dialoghi con i personaggi non giocatori (se previsti), e quindi chiama la routine `Init()` che deve essere definita dall'autore e deve quantomeno definire la locazione di partenza nella variabile location. Terminata questa fase comincia il ciclo di operazioni che si ripete ad ogni turno.
2) Viene eseguita la funzione entrypoint `GamePreRoutine();`
3) Si controlla se nella locazione esiste una sorgente di luce e vengono definite le variabili `location` e `real_location`
4) Si controlla e perfeziona la posizione di tutti gli oggetti che possono trovarsi in locazioni diverse (ovvero quelli che possiedono la proprietà `found_in`)
5) Viene salvata la posizione per l'undo
6) Viene stampata la statusline
7) Viene stampata la descrizione della locazione
8) Vengono richiamati gli eventi se previsti
9) Viene chiamata la routine entrypoint `BeforePrompt()`

10) Viene stampato il prompt
11) il giocatore inserisce il comando
12) viene fatto il parsing dell'input del giocatore
13) viene eseguito il comando o viene stampato un messaggio di errore
14) viene incrementato il numero dei turni
15) si torna al punto due.

Costanti di libreria.

Costanti per il parsing:

```
BUFFER_LEN = 120;  ! lunghezza dell'array buffer
WORDS_LEN  = 10;   ! lunghezza dell'array words
WORD_SIZE  = 2;    ! lunghezza della parola in byte
```

ITALIANO;
se definita imposta la lingua italiana per i messaggi di libreria

TALK_SINGLE_PRESS;
se definita nelle conversazioni basta premere il numero dell'opzione prescelta senza premere l'invio. In questo caso le possibili opzioni non possono eccedere i numeri da 1 a 9.

Le seguenti costanti sono utilizzate come argomento nella funzione `ClearScreen(arg);`

```
WIN_ALL    0;
WIN_STATUS 1;
WIN_MAIN   2;
```

le seguenti costanti sono utilizzate dalla estensione `miniNpc.h`

path_search 10;
profondità di ricerca per il percorso verso una destinazione

RANDOM_MV 0;
spostamento random del personaggio

PRESET_MV 1;
spostamento su un percorso stabilito del personaggio

MOVETO_MV 2;
spostamento verso una destinazione del personaggio

Variabili di libreria.

`player`
individua l'oggetto giocatore

`location`
contiene la locazione dove si trova il giocatore

`real_location`
contiene la locazione reale del giocatore se questo si trova nella locazione di default `Darkness` (oscurità), altrimenti è uguale a `location`. Per un esempio si veda Cloak of Darkness.

`now`
conteggio di tutte le azioni

`action_inv`
flag, si attiva quando il giocatore chiede l'inventario, evita la stampa della descrizione del giocatore prima dell'elenco degli oggetti posseduti.

`usefullTurns`
variabile dei turni "produttivi", conteggia i turni escludendo le azioni di sistema e quelle che producono un errore. Viene visualizzata nella statusline.

`no_turn`
flag per non contare nella variabile `usefullTurns` le azioni di sistema e i comandi sbagliati. Si attiva dopo i comandi di sistema o i messaggi di errore.

`score`
variabile che conta i punti conquistati dal giocatore.

`invisible_status`
flag per eliminare la status line, viene attivata automaticamente se lo schermo è troppo piccolo.

`v_rec;`
flag per sapere se si è in modalità registrazione o meno.

Le Classi di Libreria

La libreria base prevede unicamente la classe `Room`, di questa classe devono far parte tutti gli oggetti che fungono da locazione per il personaggio giocatore.
Altre classi sono quelle definite dalle estensioni per la gestione dei personaggi non giocatori. In particolare della classe `Character` deve far parte qualsiasi oggetto personaggio per il quale si voglia creare un dialogo a scelta multipla, mentre della classe `NPC` qualsiasi

personaggio per il quale si voglia sfruttare la gestione degli spostamenti.

Gli oggetti di libreria.

Come detto tutti i verbi sono oggetti nella libreria. Oltre a questi esistono altri oggetti definiti dalla stessa essi sono:

- `DefaultRoom`, la locazione di partenza di default nel caso l'autore non ne preveda una nella routine `Init()`.
- `Darkness`, la locazione buia dove viene proiettato il giocatore se non è presente una sorgente di luce.
- `obj_directions`, che come detto prevede le direzioni di default
- `Me`, ovvero l'oggetto che rappresenta il giocatore.

Il parser

Il parser è quella parte del codice che è responsabile di interpretare l'input del giocatore e trasformarlo in una azione nel gioco. Il parser di Mini Inform è in grado di interpretare input di una o due parole. Se il comando è composto di una parola, la prima volta che viene scritta viene data la descrizione dell'oggetto, dalla seconda in poi si tenta di agire sull'oggetto se possibile (se non previsto viene ristampata la descrizione), rimane possibile comunque dare comandi usando i verbi come ad es. "esamina oggetto" o "usa oggetto" o ancora "parla oggetto". Se si inseriscono i nome di due oggetti "oggetto1 oggetto2" il parser interpreterà il comando come "usa oggetto1 con oggetto2" o viceversa.

Proprietà della libreria

directions
in una locazione, indica i collegamenti con le altre locazioni nelle direzioni cardinali o attraverso vocaboli appositamente definiti dall'autore (come i nomi delle locazioni). Esempio:

```
...
directions     'north' 'bedroom' 0 bed_room
              'south' 'kitchen' 0 kitchen_room
              'east' 'street' 'out' 0 street_room,
...
```

le direzioni permesse sono quelle cardinali, la funzione `LanguageToInformese` contiene tutti i sinonimi che vengono accettati automaticamente. le direzione permesse sono quelle contenute nell'oggetto di default `obj_directions`.

useWith
indica le interazioni possibili con l'oggetto o tra un oggetto e l'altro, accetta un parametro chiamato usualmente '`other`' che individua un eventuale secondo oggetto. Nel caso la proprietà non sia presente o ritorni `false`, un tentativo di interazione con l'oggetto stamperà la sua descrizione.
Vediamo un esempio:

```
Character robot "a robot" foyer,
  with
    useWith [other;
      if (other == nothing) “agisci sul robot”;
      else if (other == cloak) “dai il mantello al robot”;
      else “Al ROBOT non interessa “, (name) other, “!”;
    ],
    ...
```

enter;
in una locazione, viene richiamata nel momento in cui si entra nella stessa.

exit;
in una locazione, viene richiamata nel momento in cui si esce dalla stessa.

tag;
proprietà generica.

`beforechildrenList;`
viene invocata prima di stampare l'elenco degli oggetti figli dell'oggetto che la possiede. Utile per personalizzare il messaggio di introduzione agli stessi.
In una locazione ad esempio invece del "Puoi vedere" si potrebbe personalizzare il messaggio in "Galleggiano nel vuoto accanto a te:"
Nel personaggio giocatore si potrebbe sostituire il classico "Stai portando:"

`think;`
viene invocata ad ogni turno a prescindere dalla posizione dell'oggetto e se l'oggetto in cui è definita non prevede la proprietà `nextThink` o fintanto che quest'ultima è diversa da zero.

`nextThink;`
flag utile per attivare la proprietà `think` dopo un certo numero di turni o all'avverarsi di un determinato evento. Essa ad esempio può essere posta pari a "`now + 3`" per far eseguire la proprietà `think` tre turni dopo. Una volta che viene eseguita la proprietà `think`, `nextThink` viene resettata se non diversamente specificato all'interno della proprietà `think`. Per un esempio si veda Cloak of Darkness.

`found_in;`
permette di avere lo stesso oggetto in diverse locazioni. Non vengono creati più oggetti, ma è lo stesso che viene spostato nella locazione quando vi è presente il giocatore (il controllo viene fatto ad ogni turno). La proprietà viene disattivata se l'oggetto possiede l'attributo `absent` .

Es. `found_in locazione1 locazione2,`

`describe;`
permette di aggiungere la descrizione di un oggetto in coda a quella della locazione che lo contiene (evitando che compaia nell'elenco degli oggetti dopo "puoi vedere:")

cant_go;
permette di personalizzare il messaggio per i movimenti non disponibili nella locazione. I movimenti intercettati sono quelli non previsti dalla locazione ma presenti nell'oggetto di defualt `obj_directions`.

scenic;
permette di definire in una locazione (quindi non per tutti gli oggetti) una descrizione per i vocaboli di scenografia specificati nella proprietà. Evitando il lavoro di definirli tutti come oggetti a se stanti.
Ad esempio in una ipotetica locazione bosco:

```
        ...
       scenic  'alberi' 'albero' 0 "Faggi alti svettano
                                              verso il cielo"
               'foglie' 0 "Un tappeto di foglie si stende
                                                sul terreno",
         ...
```

la proprietà può contenere fino a 36 elementi.

invent;
permette di modificare il nome dell'oggetto quando compare nella lista dell'inventario del giocatore.

link_points ;
proprietà per la libreria `miniNpc`, necessaria per il calcolo di percorsi per gli spostamenti dei personaggi non giocatori

posDir ;
proprietà usata dalla libreria `miniNpc`, per gli spostamenti dei personaggi non giocatori.

NoTalk;
l'oggetto `LipsCmd` corrispondente al verbo PARLA controlla la presenza di questa proprietà per vedere se l'autore ha predisposto un messaggio diverso da quello di default per gli oggetti non parlanti, se il giocatore tenta un dialogo con loro.

`startTalk`
questa proprietà va chiamata ogni qual volta si desidera far iniziare una conversazione a scelte multiple con un personaggio. Es. `personaggio.startTalk();`

`SetQuip`
Imposta le possibili opzioni che ha a disposizione il giocatore dialogando a scelta multipla con il personaggio.

`QuipOn`
Attiva una opzione in un dialogo a scelta multipla.

`QuipOff`
Disattiva una opzione in un dialogo a scelta multipla.

`QuipsOn`
Attiva più opzioni contemporaneamente in un dialogo a scelta multipla.

`QuipsOff`
Disattiva più opzioni contemporaneamente in un dialogo a scelta multipla.

`TestQuip`
Controlla lo stato di una opzione in un dialogo a scelta multipla.

`movestarget`
proprietà della libreria `miniNpc`, indica locazione da raggiungere se il tipo di movimento è quello verso un obiettivo.

`moves_step`
controlla a che punto del percorsosi trova il personaggio non giocatore

`moves_stop`
Se pari a `true` ferma qualsiasi movimento del personaggio non giocatore

nextMovesThink
molto simile a `nextThink` ma dedicata ai movimenti dei personaggi non giocatori

path_size
lunghezza del percorso calcolato verso un obiettivo

path_stage
a che punto ci si trova nel calcolo del percorso

path 0 0 0 0 0 0 0 0 0 0 0 0,
un array per contenere il percorso calcolato

casualties
è pari a un numero intero e misura la probabilità che ha un personaggio non giocatore di spostarsi a prescindere dal tipo di movimento a cui è soggetto 1=100% 4=25% (default)

destination,
la locazione verso cui si muove l'npc ad ogni turno

prepath 0 0 0 0 0 0 0 0 0 0 0 0,
un array per contenere il percorso precalcolato ed impostato dall'autore

when_exit
viene chiamata ogni qual volta il personaggio non giocatore esce dalla locazione in cui si trova il giocatore, di default stampa un messaggio del tipo "Il personaggio si allontana"

when_enter
analoga alla precedente, ma quando il personaggio non giocatore entra nella locazione dove si trova il giocatore.

moves_type,
tipo di movimento del personaggio non giocatore può essere pari a 0 (random), 1(precalcolato) o 2 (verso un obiettivo).

doAtTarget
questa proprietà viene eseguita quando il personaggio non giocatore raggiunge la propria destinazione.

moves_think
analoga a `think` ma dedicato ai movimenti.

Attributi di libreria.

scenery
l'oggetto è considerato di scena e non compare nelle descrizioni delle locazioni o nell'inventario. E' posseduto anche da tutti gli oggetti VERBI che sono definiti come figli dell'oggetto giocatore.

hidden;
oggetto nascosto, non compare nelle descrizioni, al momento tale attributo è posseduto solo dall'oggetto `ME` (giocatore)

showchildren;
la descrizione dell'oggetto mostra un elenco degli oggetti che contiene, è posseduto dalle locazioni e dall'oggetto `ME` (giocatore)

showName;
prima della descrizione dell'oggetto viene visualizzato il nome dello stesso in grassetto. E' posseduto dalle locazioni.

seen;
viene attribuito agli oggetti una volta che sono stati esaminati

absent;
impedisce lo spostarsi degli oggetti con proprietà `found_in`, dare l'attributo `absent` non fa sparire l'oggetto, ma gli impedisce solo di spostarsi in altre locazioni attraverso la proprietà `found_in`. Se si vuole eliminare del tutto un oggetto è necessario dopo avergli dato l'attributo `absent` operare una istruzione `remove`.

light;
rende l'oggetto una sorgente di luce. Di default ogni locazione ha questo attributo.

Entrypoints

GamePreRoutine();
Viene chiamata all'inizio di ogni turno prima di qualsiasi operazione.

BeforeParsing();
Viene chiamata prima del parsing dell'input del giocatore.

BeforePrompt();
Vene chiamata subito prima della stampa del prompt.

BeforeInventary();
Viene chiamata prima della stampa dell'inventario.

NewRoom();
Viene chiamata ogni qualvolta ci si sposta in un'altra stanza attraverso le direzioni cardinali.

TimePasses();
Viene chiamata al termine di ogni azione utile, ovvero i turni che non restituiscono messaggi di errore o che non

sono generati da VERBI di sistema come SALVA ad esempio.

Funzioni

La libreria base contiene diverse funzioni ad uso interno:

`LibMessages`

Contiene i messaggi di sistema in lingua italiana ed inglese.

`ObjectFromDictName`

Ricava dal vocabolo di dizionario trovato dal parser l'oggetto a cui corrisponde. E' usata dal parser.

`PrintOrCommand`

Esegue routine o stampa i messaggi o esegue i movimenti tra le locazioni. E' usata dal parser.

`CheckForDirections`

Controlla se la parola di dizionario trovata dal parser è una direzione. E' usata dal parser.

`CheckForScenic`

controlla se la parola di dizionario trovata dal parser è un vocabolo di scena, ovvero definito in nella proprietà scenic. E' usata dal parser.

`CheckForLight`

Controlla se esiste una sorgente di luce nella locazione. E' usata dal parser.

`PrintWordAddress (w);`

stampa della "w"-esima parola dell'input del giocatore contenuta nell'array Buffer. E' usata dal parser per i

messaggi di errore sulle parole non comprese nel dizionario.

Il file `routines.h` contiene le seguenti funzioni:

`NextWord(wn);`
restituisce la parola successiva alla wn(esima) nell'array words.

`WordAddress(wordnum);`
restituisce l'indirizzo della parola wordnum.

`LTI_Insert`
LTI_insert permette di aggiungere spazi all'interno dell'array buffer, è utilizzata la `languageToInformese` per permettere la modifica del buffer.

`LanguageToInformese`
Questa funzione permette di leggere l'input del giocatore e di modificarlo in alcuni casi perchè venga compreso dal parser. Di default è utilizzata per definire dei sinonimi e delle abbreviazioni alle direzioni cardinali. Ma può avere altri utili usi, soprattutto per il porting in lingue straniere.

`Tokenise__(b p);`
risistema l'array p tratto da b (se quest'ultimo è stato modificato). E' utilizzata da `languageToInformese`.

`KeyCharPrimitive`
Richiede al giocatore la pressione di un tasto per proseguire, la routine restituisce il carattere premuto.

`CommonAncestor (o1, o2);`

restituisce il genitore più prossimo che contiene direttamente o indirettamente i due oggetti, o restituisce `false` se non esiste.

`IndirectlyContains (o1, o2);`

Restituisce `true` se o1 contiene direttamente o indirettamente o2.

`ClearScreen(arg);`

Pulisce lo schermo nella parte definita dall'argomento, che dovrebbe essere una tra le costanti WIN_ALL, WIN_STATUS, WIN_MAIN.

`YesOrNo();`

Aspetta un input dal giocatore del tipo SI/NO. Restituisce `true` se la risposta è si, false se è no.

`DrawStatusLine();`

Disegna la status line sullo schermo. Di default questa contiene solo il numero di turni ed i punti conquistati.

`StartMoves (npc, type, target);`

Comincia a muovere il personaggio npc, verso una locazione target secondo un modo type=0... Random; type=1... PreSetPath

`StopMoves (npc);`

Ferma qualsiasi movimento del personaggio npc.

Gestire i personaggi non giocatori.

Gestire i dialoghi a scelta multipla
Ogni personaggio non giocatore per il quale si desidera impostare un menu di conversazione a risposta multipla deve appartenere alla classe `Character`, e deve avere le proprietà `SayQ(x)` e `Respond(x)`, che stampano l'x-esimo titolo e la x-esima risposta; inoltre `Respond` normalmente fa altre cose, oltre a stampare la risposta (come ad esempio attivare o disattivare altre opzioni oppure uscire dal menu).
Ogni personaggio non giocatore deve avere anche una funzione `InitQuips()`, che deve inizializzare (vedi sotto) i 'quip' (flag che indicano se una certa coppia frase/risposta è attiva o no) che sono attivi di default (all'inizio dell'avventura).

`NPC.SetQuip(42, 1)` attiva il 42-esimo quip del personaggio NPC, mentre `NPC.SetQuip(42, 0)` lo disattiva. `NPC.QuipOn(42)` e `NPC.QuipOff(42)` sono abbreviazioni per gli stessi comandi.

`NPC.QuipsOff(N, a, b, ...)` disattiva N quips indicati dagli argomenti successivi (a, b, ...); N varia da 0 a 6, visto che inform accetta al massimo 7 argomenti per funzione. `NCP.QuipsOn(N, a, b, ...)` fa esattamente ciò che pensate.

Infine `NPC.TestQuip(42)` restituisce il valore (1 o 0) del 42-esimo quip.

Di default si possono usare 56 quip, numerati da 0 a 55.

Se si definisce la costante `TALK_SINGLE_PRESS`; la libreria accetta come scelta la pressione di un singolo tasto (numerico). Ovviamente in questo caso ci possono essere in ogni istante solo 9 opzioni di dialogo possibili (solo nove quip attivi).

Per uscire dal dialogo, la risposta (nella proprietà `respond`) deve ritornare FALSE, se invece restituisce TRUE allora il dialogo continua.

Gestire i movimenti dei personaggi non giocatori

La gestione è assicurata tramite l'inclusione nel proprio sorgente del file `miniNpc.h`. La libreria permette di impostare i movimenti di uno o più personaggi non giocatori, purché questi facciano parte della classe NPC. Per far partire i movimenti basta dare il comando:

```
StartMoves(npc, type, target);
```

dove i tre argomenti indicano il nome dell'oggetto personaggio, il tipo di movimento (rappresentato da una costante) ed una eventuale locazione obiettivo.
I tipi di movimenti al momento previsti sono tre:

1) spostamenti casuali, il comando è

```
StartMoves(npc, RANDOM_MV);
```

Ad ogni turno il personaggio si sposta verso una delle locazioni adiacenti a caso.

2) spostamenti seguendo un percorso prestabilito, il comando è `StartMoves(npc, PRESET_MV);`

Ad ogni turno il personaggio si sposta di una locazione seguendo un elenco che deve essere definito nella proprietà `npc.prepath`. Es:

```
Object Pippo "pippo" class NPC
    with....
         prepath cucina salotto corridoio atrio strada,
        ...
```

ogni locazione indicata nell'array deve essere adiacente a quella precedente.

3) spostamenti verso una locazione definita. il comando è

```
StartMoves(npc, MOVETO_MV, locazione_target);
```

In qualsiasi punto della mappa si trovi il personaggio, viene calcolato un percorso che lo conduca alla locazione indicata al terzo argomento e quindi ad ogni turno si sposta verso di essa.
Questo tipo di movimento necessita che la locazione obiettivo sia EFFETTIVAMENTE raggiungibile da qualsiasi punto si possa trovare il personaggio al momento in cui viene dato il comando, che tutti i collegamenti tra le locazioni interposte tra la locazione di partenza (qualsiasi essa sia) e quella di arrivo (definita dall'autore) siano BIUNIVOCI ovvero se da A si può passare a B allora anche da B si può passare ad A, infine che il percorso non sia superiore ai DIECI spostamenti (questo limite può essere innalzato). ULTERIORI problemi potrebbero presentarsi se si cambiano i collegamenti tra le locazioni in modo dinamico DOPO che il programma ha calcolato il percorso del personaggio (il che avviene solo al momento della partenza e non viene ricalcolato ad ogni turno).

A prescindere dal tipo di movimento, ogni personaggio della classe NPC possiede una proprietà `casualties` essa stabilisce la probabilità che il personaggio si muova ad ogni turno. `casualties == 1` rappresenta il 100% ovvero il personaggio muove ad ogni turno. `casualties == 2` rappresenta il 50%. Di default la probabilità di movimento di un personaggio è del 25% (`casualties==4`).

Per fermare un personaggio basta dare il comando `StopMoves(npc);`

Ogni personaggio della classe NPC possiede anche altre due proprietà `when_enter` e `when_exit` esse vengono chiamate quando il personaggio arriva o esce dalla locazione dove si trova il giocatore, di default stampano un messaggio del tipo "Arriva Tizio" o "Tizio si allontana".

Un'ultima proprietà interessante è `npc.doAtTarget` essa viene richiamata quando un personaggio giunge a destinazione, sia che si stia muovendo su un percorso predefinito che su di uno calcolato (ma non se si muove a caso).

I personaggi hanno bisogno di un daemon per i movimenti, la libreria però non utilizza la proprietà `think` a questo scopo (in tal modo essa rimane libera anche nei personaggi in movimento), ma essa prevede una nuova proprietà `moves_think`. Essa è interna alla libreria e l'autore non deve preoccuparsene, se non per il fatto che deve tenere conto che per richiamare tale proprietà ad ogni turno la libreria utilizza l'entrypoint `BeforePrompt();`

L'autore che avesse bisogno di utilizzare tale entrypoint oltre alla libreria estensiva `miniNpc.h` dovrebbe pertanto comprendere le istruzioni ivi contenute.

Mini Inform i sinonimi e le lingue straniere

Adattare le librerie alle lingue straniere è estremamente facile, basta tradurre le stringhe presenti nella routine `LibraryMsg`, ed aggiungere le direzioni nella lingua prescelta come sinonimi nella funzione `LanguageToInformese`.