Guida a Inform–Glulx

a cura di
Marco Falcinelli

Prima Edizione 2006

GUIDA A INFORM-GLULX, prima edizione

a cura di Marco Falcinelli.

Editing and Design grafico: Marco Falcinelli.

Traduzioni e testi di: Marco Falcinelli e Paolo Vece.

Copertina: *Road to Glulx*, schizzo a matita di Marco Falcinelli, 2006.

SOMMARIO

Prefazione

Saltare la prefazione, è cosa risaputa, sembra essere lo sport preferito del lettore, per questa volta vi chiedo di fare uno sforzo e leggere fino in fondo questa pagina che vi aiuterà ad orientarvi in questa guida. Quello che avete tra le mani, infatti, non è il classico manuale con un capo ed una coda. Ma piuttosto una guida di riferimento da leggere e consultare nel momento in cui avete bisogno di approfondire o riportare alla memoria un argomento.

Questo volume raccoglie infatti le traduzioni di alcuni documenti che sono usciti negli ultimi anni in ordine sparso e li mette assieme in modo che possano essere facilmente consultabili e confrontabili tra loro dall'autore.

Ci siete ancora? Bene, non distraetevi e non cedete alla tentazione di girare pagina. I documenti raccolti in questa guida hanno lo scopo di andare a compensare ciò che non è scritto nell'*Inform Designer's Manual* di Graham Nelson, fermo alla quarta versione del 2001. Nel frattempo Inform e Glulx, non sono cambiati molto, ma si sono integrati ed hanno messo a disposizione dell'autore un sistema più potente e flessibile. Fate attenzione, questa guida non sostituisce né è un surrogato del DM4, anzi, ne da per scontata se non la conoscenza approfondita quanto meno una lettura. Il manuale di Nelson è sempre il punto di riferimento, mentre questa guida vi servirà solo per approfittare dei miglioramenti intercorsi negli ultimi quattro anni su Inform6 ed in particolare si concentrerà sulla sua integrazione con Glulx. Se volete arricchire il vostro gioco di contenuti multimediali siete nel posto giusto.

Non vi distraete proprio ora, mi raccomando. Il volume che avete tra le mani presenta nei primi sei capitoli le *Inform Release Notes* pubblicate su internet assieme alla nuova versione del compilatore `Inform 6.30` e alle `librerie 6/11`. Esse spiegano tutte le novità presenti nella libreria, le nuove funzioni disponibili, l'integrazione con Glulx e anche i bug corretti. Successivamente dal capitolo 7 al 10, troverete *Gull* una serie di articoli scritti da Adam Cadre e tradotti in italiano nella quasi totalità da Paolo Vece. Per questa occasione la traduzione di Gull è stata completata ed aggiornata alle ultime novità. Il capitolo 11 presenta,inoltre, al lettore la *libreria* `SGW` di Alessandro Schillaci, che propone un approccio più soft a Glulx e permette all'autore non esperto di arricchire l'aspetto del proprio gioco con poca fatica. Infine troverete nelle appendici utili approfondimenti sulle funzioni più complesse ed una completa guida di riferimento a Glulx e alla libreria infglk.h.

Riassumendo, la presente guida può essere letta dal principio alla fine senza difficoltà o consultata al bisogno nella ricerca di una soluzione o del funzionamento di una routine. Se vi ci perdete, non disperate, invocate aiuto su `it.comp.giochi.avventure.testuali` ed aiutateci a migliorarla....

Pomezia, 13 Settembre 2006

Marco Falcinelli

§1 Introduzione

Il documento che state per leggere è un aggiornamento alla quarta edizione del 2001 dell'Inform Designer's Manual di Graham Nelson. La sua stesura si è resa necessaria per informare gli autori di avventure testuali delle novità apportate al sistema in questi ultimi anni ed in particolare rispetto al compilatore 6.21 e le librerie 6/10 che furono rilasciate nel lontano 1999 ed a cui fa riferimento il manuale del 2001. L'obiettivo principale è quello di mettere ordine tra i problemi che erano stati identificati nell'elenco di Patch di Inform:

```
http://www.inform-fiction.org/patches/index.html
```

E quindi di spiegare le poche migliorie che sono state inserite, perlopiù basate sulle idee segnalate dagli autori nell'elenco dei suggerimenti per Inform:

```
http://www.firthworks.com/roger/suggest.html
```

Sebbene tutti i bug conosciuti siano stati corretti, l'approccio per le modifiche di Inform è stato molto conservativo. Non tutti i suggerimenti, infatti, sono stati implementati, ma sono stati selezionati ricorrendo a tre criteri:

• evitare i cambiamenti che avrebbero potuto creare problemi ai giochi già realizzati (retrocompatibilità).

• la minimizzazione delle caratteristiche che avrebbero richiesto un aggiornamento all'*Inform Designer's Manual*;

• essenzialità e semplicità.

Con il permesso di Graham Nelson, l'aggiornamento di Inform e la redazione di questo documento sono stati prodotti dallo sforzo di un gruppo di volontari, il cui entusiasmo è stato moderato da una certa mancanza di familiarità con le funzioni interne di Inform. Perciò, ci siamo concentrati su quei cambiamenti che abbiamo ritenuto "sicuri"; l'implementazione di alcune buone idee è stata posposta fino al giorno in cui non ci sentiremo abbastanza sicuri di sapere cosa stiamo per fare. Abbiamo perciò evitato quei cambiamenti che avrebbero potuto, a causa della nostra ignoranza, creare una moltitudine di bug.

Detto questo, il rilascio delle librerie 6/11 e del compilatore 6.30 presenta alcune novità di rilievo. Innanzi tutto ci si è basati sul compilatore e le librerie bipiattaforma – Z-machine e Glulx – di Andrew Plotkin, che erano direttamente

derivate dal compilatore 6.21 e la libreria 6/10. Il risultato è che le due Virtual Machine (VM) sono state accorpate in un unico compilatore ed in una unica libreria che, sebbene continui di default a produrre Z-code, può in alternativa generare codice per la Glulx VM nel caso in cui venga aggiunta l'opzione `-G` alla compilazione, maggiori informazioni su questo argomento possono essere trovate nel paragrafo 5 "*Il supporto a Glulx*". Ma come primo passo sarà bene prima elencare i cambiamenti apportati al compilatore ed ai file della libreria[1].

§2 Ringraziamenti

Troppe persone hanno contribuito al rilascio della nuova versione, riportando e risolvendo bug, dando utili suggerimenti, fornendo aiuto e supporto, testando, e tanto altro, per essere elencati in una lista di ringraziamenti. Pertanto ecco un ringraziamento generale a tutti coloro che hanno contribuito a portare a termine questo lavoro, ed in particolare a Graham per averlo permesso ed ad Andrew per il suo lavoro da pioniere su Glulx.

Troverete, sicuramente, delle imperfezioni in questa versione che potranno essere segnalate, come al solito, nella lista delle Patch o in quella dei Suggerimenti. Ogni argomento serio dovrebbe essere anche postato sul newsgroup rec.arts.int-fiction, con l'intestazione indicante "`[Inform63] …`"[2].

Per quanto riguarda Glulx, questo volume ha un debito morale con tutti coloro che hanno contribuito alla sua diffusione pubblicando articoli e tutorial su di esso. Su tutti il nostro ringraziamento più sincero va a Andrew Plotkin, Adam Cadre e Marnie Parker.

---

[1] Se avete tradotto la libreria 6/10 in un'altra lingua, potrete trovare molte utili informazioni nel paragrafo 6 *Informazioni per i traduttori.*

[2] Le segnalazioni in italiano invece possono essere postate su it.comp.giochi.avventure.testuali

§3 Il Compilatore 6.30

Ecco elencati i cambiamenti apportati nella versione 6.30 del compilatore per Inform, che possono essere distinti in due categorie: *nuove caratteristiche* e *bug corretti*.

3.1 Nuove caratteristiche

Il compilatore 6.30 rispetto alla precedente versione 6.21 presenta alcune interessanti e nuove caratteristiche.

• Il compilatore definisce automaticamente la costante WORDSIZE, il cui valore è 2 quando si compila per la Z-machine, e 4 se, invece, si compila per Glulx. La costante specifica il numero di byte in una parola (word) nella VM; raccomandiamo caldamente di usare tale costante nelle poche occasioni dove tale numero è significativo. Il compilatore definisce anche la costante `TARGET_GLULX` se aggiungete l'opzione -G alla compilazione, o in alternativa la costante `TARGET_ZCODE`; in entrambe le occasioni il valore della costante è 0[3].

• La direttiva `Switches`, che permette di definire alcune opzioni del compilatore all'interno del sorgente piuttosto che nella linea di comando del compilatore. Tale caratteristica si è rivelata un meccanismo sorprendentemente straordinario. Lo speciale carattere di commento "!%", nelle primissime righe del file sorgente, vi permette di specificare comandi in Inform Command Language (ICL) per controllare la compilazione. Ad esempio:

```
!% -E1G                                  ! Glulx, 'Microsoft' errors
!% -~S                                   ! disable Strict mode
!% +include_path=./test,./,../lib/contrib
                                  ! che guarda nella libreria 'test'
!% $MAX_STATIC_MEMORY=20000
Constant STORY "RUINS";
```

[3] Per maggiori informazioni sull'uso di queste costanti si legga il paragrafo 5 *Il Supporto a Glulx*.

L'ICL è descritto nel paragrafo §39 dell'*Inform Designer's Manual*. In breve: ogni linea specifica un singolo comando, partendo con "-" per definire una o più opzioni, "+" per definire la variabile di un percorso, o "$" per definire l'impostazione di memoria. I commenti sono introdotti dal carattere "!". Il commando ICL "compile" non è permesso all'inizio del codice sorgente.

• Sono disponibili due nuove impostazioni della memoria; entrambe potevano precedentemente essere modificare solo ricompilando il compilatore. $MAX_SOURCE_FILES che ha di default il valore di 256 e $MAX_INCLUSION_DEPTH che invece ha un valore di default di 5.

• E' stata introdotta una nuova direttiva, simile ad Array ... string e Array... table:

```
Array array buffer N;
Array array buffer expr1 expr2 ... exprN;
Array array buffer "string";
```

Questa crea un array ibrido della forma usata da string.print_to_array () e la routine di libreria PrintToBuffer(), nella quale la prima **word** array-->0 contiene N e i seguenti N **byte** array->WORDSIZE, array->(WORDSIZE+1) ... array->(WORDSIZE+N-1) contengono i valori della espressione specificata o una stringa di caratteri.

• Una nuova regola di stampa (A) – simile a quella esistente (The) – stampa l'articolo indefinito con la prima lettera maiuscola. L'articolo stampato di default è "A" o "An", o viene preso dalla proprietà article dell'oggetto.

• La grandezza minima della testata della Z-code header extension table può essere impostata usando l'opzione della linea di comando -Wn. Ad esempio, -W6 rende la tabella di almeno sei parole di ampiezza.

• Sono ora supportati codici sorgente in set di caratteri diversi da ISO 8859-1 a 8859-9, purché il set di caratteri possa essere mappato in uno dei nove set ISO 8859. Un file di mappatura viene usato per definire come il codice sorgente deve essere processato. Questo file consiste di una direttiva indicante il set ISO 8859 che deve essere mappato, seguito dai 256 numeri che danno la mappa. Come esempio, sotto Microsoft Windows, il testo in Russo è codificato con il set di

caratteri definito Microsoft. Il seguente testo definisce una mappa al set ISO 8859-5:

```
! Windows Cyrillic (code page 1251) to ISO 8859-5
C5
0, 63, 63, 63, 63, 63, 63, 63, 63, 32, 10, 63, 10, 10, 63, 63
63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47
48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63
64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79
80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95
96, 97, 98, 99,100,101,102,103,104,105,106,107,108,109,110,111
112,113,114,115,116,117,118,119,120,121,122,123,124,125,126, 63
162,163, 44,243, 34, 46, 63, 63, 63, 63,169, 60,170,172,171,175
242, 39, 39, 34, 34, 46, 45, 45,152, 84,249, 62,250,252,251,255
32,174,254,168, 36, 63,124,253,161, 67,164, 60, 63, 45, 82,167
63, 63,166,246, 63, 63, 63, 46,241,240,244, 62,248,165,245,247
176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191
192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207
208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223
224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239
```

Le linee che cominciano con "`!`" sono considerate di commento. La linea successiva, che comincia con "`C`", definisce il set ISO da mappare nella medesima maniera in cui lo fa l'opzione `-Cn` nella linea di comando. Per usare la mappatura, Inform tratta ogni carattere nel codice sorgente come un numero tra 0 e 255, ed usa tale numero come indice della tabella di mappatura. Per esempio, supponete che il carattere letto da una Finestra di testo in Russo sia la piccola lettera cirillica "ya". Questo carattere è rappresentato nel set di caratteri del Windows Russo dal numero 255. Inform prende tale ingresso nella mappa su fornita, che corrisponde al numero 239. Perciò il carattere viene considerato come essere il numero 239 nel set ISO 8859-5.

Il nome del file di mappatura è specificato dalla nuova variabile di percorso del compilatore `+charset_map`. Se la suddetta mappa è inserita in un file di testo `win1251.map`, un gioco Russo potrebbe essere compilato con una linea di comando di questo tipo:

```
inform +charset_map=win1251.map +language_name=Russian mygame.inf
```

• Gli opcode `@check_unicode` e `@print_unicode`, introdotti negli *Z-Machine Standards Document* versione 1.0, possono ora essere chiamati per nome invece che attraverso la loro generica e poco chiara sintassi `@"EXT:11S"` e `@"EXT:12S"`. Per esempio:

```
@print_unicode $0401;
```

• La modalità "Strict" (che compila controllando l'integrità del gioco in esecuzione) è stata separata dalla modalità Debug (che definisce i verbi di debug come TRACE e SHOWOBJ). In tal modo non è più necessario disattivare la modalità "Strict" (per disabilitare i verbi di Debug) prima di rilasciare un gioco, sebbene sia consigliabile farlo dal momento che si salva spazio e si migliorano le prestazioni del gioco stesso. Di default, lo Strict mode è **abilitato** (potete disabilitarlo con l'opzione `-~S`) e la modalità Debug è **disabilitata** (potete attivarla con l'opzione `-D`).

• Il compilatore ora prevede un avvertimento (warning) se usate l' `array->n` su un array di parole, o `array-->n` su di un array di bytes. Usate la nuova direttiva `Array ... buffer` per creare array ibridi a cui si può accedere senza provocare un warning.

• Il compilatore prevede un ulteriore avvertimento (warning) se vi riferite ad una proprietà non qualificata all'interno di una routine[4], per esempio digitando `(number==0)` quando intendevate riferirvi a `(self.number==0)`.

• Un ultimo avvertimento (warning) appare se includete qualcosa di diverso da una parola di dizionario all'interno della proprietà `name` di un oggetto. Tale avvertimento vi sarà mostrato, per esempio, se provate ad aggiungere una singola lettera come parola di dizionario, digitando `name 'g' 'string'` quando invece avreste dovuto scrivere `name 'g//' 'string'`.

[4] Unqualified property name.

3.2 Bug corretti

Le correzioni ai bug noti che trovate qui di seguito possiedono un numero identificativo nella forma [C62127]. Tale identificativo è un riferimento corrisponde al numero del bug segnalato nell'elenco delle Patch della sezione 'Compiler'.

• E' stato corretto un problema con l'assegnazione multipla che coinvolgeva il puntatore aritmetico che dava errate risposte in modalità "Strict". [C62127]

• Dopo aver usato l'istruzione `Extend only` per separare un elemento dalla definizione esistente del Verbo i nuovi sinonimi che si danno a tale elemento si riferisco ad esso e non al residuo della definizione originale come accadeva precedentemente. [C62126]

• La modalità "Strict" ora fa il test per l'uso degli opcode `@put_prop` o `@get_prop` quando una proprietà comune è più lunga di due byte - gli *Z-Machine Standards Document* affermano che ciò è illegale, e che il risultato non è specificato. Il messaggio di errore è nella forma:

"[** Programming error: `obj` (object number N) has a property `prop`, but it is longer than 2 bytes so you cannot use "." to read **]".

Ciò significa che avete usato il costrutto `obj.prop` nella situazione dove `prop` è una proprietà **comune** (common) che contiene due o più valori; tipicamente `prop` viene esplicitata usando un array, o è una proprietà additiva e sia `obj` che il suo parent class hanno definito dei valori per essa. Il problema non si presenta se `prop` è una proprietà **individuale**. [C62125]

• La gestione delle virgolette Europee è (finalmente) stata corretta. Il simbolo "«" è prodotto da una delle seguenti combinazioni @<<, @@163 e @{00AB}, mentre @>>, @@162 e @{00BB} producono il rispettivo simbolo di chiusura "»". Nota, comunque, che il problema era originato da un precedente errore degli *Z-Machine Standards Document*, e quindi gli interpreti scritti basandosi su tali specifiche, o anche più recenti ma corretti per funzionare con Inform 6.12 (che aveva adottato una soluzione sbagliata), potrebbero ancora non funzionare correttamente. [C62124]

• Il messaggio di errore "no such constant" ora riporta l'appropriato numero di riga del codice che lo ha generato. [C62123]

• Le funzioni `metaclass()` e `ZRoutine()` non restituisco più numeri senza segno – riguardo l'area della memoria statica del gioco – come un tipo di stringa. Inoltre la costante `NULL (-1)` non viene più riportata come Stringa. [C62122]

• Espressioni complesse che combinano una chiamata ad una routine e gli operatori `ofclass` e `or` non generano più codice scorretto. [C62121]

• Costanti di valore negative in operazioni assembly - per esempio `@set_colour 4 (-1);` non causano più messaggi di espressioni inaspettate dal compilatore. [C62119]

• Sono stati risolti vari problemi con la gestione dei caratteri ISO 8859 nel blocco 128-255, ed anche nell'uso delle sequenze invio @@. [C62117, C62115, C6211, C62003]

• Una direttiva `Abbreviate` contente una sottostringa (substring) di "<unknown attribute>" non dovrebbe più mandare in crash il compilatore, speriamo. [C62116]

• Il limite di 320Kb posto da Inform su giochi v6 e v7 è stato alzato a 512Kb. [C62114]

• Gli accessi al dizionario che seguono una direttiva `Zcharacter` che rimpiazza l'intera tabella alfabetica, non dovrebbero più corrompersi. [C62113]

• Il compilatore ora genera parentesi condizionali che possono abbracciare 63 byte, alzando il precedente e non necessario limite di 31-byte, portando ad un piccolo risparmio di codice. [C62112]

• Mettere un oggetto all'interno di se stesso non produce più un ciclo infinito. [C62110]

• Vari problemi con gli opcode `@store`, `@inc_chk`, `@dec_chk`, `@not` e `@je` sono stati risolti. [C62109, C62108, C62105]

• I problemi con le direttive condizionali di compilazione `#Ifndef...#Ifnot...#Endif` sono stati risolti. [C62107]

• Una parola di dizionario lunga – come ad esempio `'rinoceronti//p'` – ora viene correttamente intesa come plurale. [C62103]

• Sono stati corretti i problemi con il constant folding – che consiste, nell'avere l'espressione valutata in fase di compilazione, che erano stati solo parzialmente risolti dalla precedente versione bipiattaforma del compilatore. [C62102]

• Quando si compila per Glulx, il compilatore usa gli opcode `@callf`, `@callfi`, `@callfii` o `@callfiii` dove possibile per generare chiamate invece di inserire sempre più argomenti nello stack e poi fare la chiamata `@call`.

• La presenza della direttiva `Switches G;` non causa più il blocco del compilatore.

• Il limite, inaspettato, di 1024 etichette (labels) per routine nel linguaggio assembly per Z-machine generato dal compilatore è stato elevato a 32768. Il modo più verosimile per incontrare tale limite è creando una istruzione switch con un numero estremamente elevato di casi.

• E' stato corretto un problema con l'istruzione read che generava l'errato opcode in giochi di versione 4.

• La dichiarazione di una classe dinamica come quella menzionata nel DM4 `Class Pebble(NUM_PEBBLES) … ;` non creerà più una misteriosa moltitudine di oggetti se il `NUM_PEBBLES` non è definito.

§4 La Libreria 6/11

Qui di seguito sono descritte le modifiche apportate nella versione 6/11 delle librerie di Inform. Anche qui i cambiamenti possono essere classificati in: *Nuove caratteristiche, Routine di libreria* e *Bug corretti.*

4.1 Nuove caratteristiche

La libreria 6/11 introduce rispetto alla sua precedente versione 6/10 numerose nuove caratteristiche:

• La libreria automaticamente definisce quattro costanti: LIBRARY_PARSER alla fine di Parser.h, LIBRARY_VERBLIB alla fine di VerbLib.h, LIBRARY_GRAMMAR alla fine di Grammar.h, e LIBRARY_ENGLISH alla fine di English.h. I contributi degli autori nella forma di estensioni alla libreria possono ora usare tali costanti per controllare che la loro estensione alla libreria venga inclusa (`Include`) nella giusta posizione. Una quinta costante `LIBRARY_VERSION`, al momento definita con il numero 611, può essere controllata dalle estensioni che richiedono questa particolare versione della libreria.

• La parola "muro" è stata tolta dall'oggetto `CompassDirection` definito nel file English.h, lasciando semplicemente i nomi delle direzioni "north", "south", etc.

• I verbi LOOK [TO THE] NORTH, LOOK DOWN, LOOK OUT[SIDE] etc – ma non LOOK IN[SIDE], che era già disponibile – sono stati aggiunti. Di default, la risposta è nella forma "Non vedi nulla di strano...", ma potete cambiarla per ogni singola direzione definendo una proprietà `compass_look`:

```
Room study "Il tuo studio"
with description "C'è una entrata ad est di questa stanza austera.",
compass_look [ obj;
        if (obj == e_obj) "Puoi vedere l'entrata alla stanza.";
        if (obj == n_obj or s_obj or w_obj) "Vedi il muro.";
],
e_to hallway;
```

Questa miglioria sfrutta il meccanismo descritto su:

`http://www.firthworks.com/roger/informfaq/ww.html#1`

`http://www.onyxring.com/InformGuide.aspx?article=23`

(con l'eccezione che la proprietà `compass_look` era precedentemente chiamata `compasslook`[5]), e comporta che non c'è più bisogno di fare una modifica della libreria nel punto indicato dalle precedenti pagine web.

• I verbi ASK `npc` TO `command` e TELL `npc` TO `command` – entrambi sinonimi di `npc`, `command` – sono ora stati definiti[6]. La nuova grammatica è:

```
Verb 'ask'
...
* creature 'to' topic -> AskTo
...
```

Nella quale il token `creature` corrisponde all' `npc` e il token `topic` rappresenta il `command`. `AskTo` non è una azione nel senso tradizionale del termine: viene intercettata dal parser e convertita nel formato originale `npc`, `command`. L' `npc` può intercettare il `command` se prevede una proprietà `orders` come fa usualmente e come è riportato nel paragrafo §18 dell'*Inform Designer's Manual*.

Questa miglioria comporta che non c'è più bisogno di inserire l'estensione `AskTellOrder.h` di Irene Callaci che in questo modo è stata inclusa nella libreria.

• I verbi REGISTRA [ON|OFF] e REPLAY sono ora sempre disponibili, indipendentemente dalla modalità DEBUG. Ciò potrebbe comportare errori di compilazione se avete già definito per conto vostro questi verbi.

• Sono stati aggiunti I verbi PRY, PRISE, PRIZE e LEVER[7]. Ciò potrebbe portare ad errori di compilazione se avete già definito tali verbi per vostro conto.

• Il parser tratta le linee di input che cominciano con "`*`" come un commento, senza tentare alcun altro parsing. Il carattere usato per introdurre commenti può essere modificato definendo COMMENT_CHARACTER prima dell'istruzione `Include Parser;`

---

[5] Attenzione il vecchio nome è ancora presente nella versione 2.5 di Infit, curata da Giovanni Riccardi, la libreria italiana per inform e dovrebbe essere corretto (NdT).

[6] Con command si intende un ordine del tipo "Chiedi/Di a Tizio di fare questo" che normalmente viene posto con la formula "Tizio, fai questo" (NdT).

[7] Impicciati, forza, premia e fai leva.

Ad esempio: `Constant COMMENT_CHARACTER '!';`

Dal momento che i commenti sono usati principalmente quando viene abilitata la trascrizione di ciò che accade nel gioco, sia in maniera completa (SCRIPT ON) o solo nella registrazione dei comandi impartiti (RECORDING ON), il parser risponde "[Commento registrato]" o "[Commento NON registrato]" se inappropriato[8].

• L'oggetto `selfobj`[9] ora include una proprietà vuota `add_to_scope`, che può essere sovrascritta dalla propria routine. Il suo uso tipico è quello di equipaggiare il giocatore con parti del proprio corpo. Per un oggetto singolo:

```
selfobj.add_to_scope = nose;
```

o per più oggetti:

```
[ IncludeBodyParts;
  PlaceInScope(nose);
  PlaceInScope(hands);
];
selfobj.add_to_scope = IncludeBodyParts;
```

• Il sistema di punteggi basato su obiettivi (task-based), descritto nel paragrafo §22 dell'*Inform Designer's Manual,* usa un byte array, che preclude l'assegnazione di punteggi molto alti o negativi. Per aggirare tale problema, potete rimpiazzare con l'istruzione `Replace` la routine di libreria `TaskScore()` come nel successivo esempio e quindi definire `task_scores` come un **word** array:

```
Replace TaskScore;
Array task_scores --> 100 200 300 400 (-50) 600;
[ TaskScore i; return task_scores-->i; ];
```

• Il sistema di punteggi viene completamente disabilitato se definite all'inizio del vostro gioco la costante `NO_SCORE`.

```
Constant NO_SCORE;
```

[8] Tale caratteristica praticamente ingloba l'estensione betatest.h (NdT).

[9] Ovvero il giocatore (NdT).

• E' disponibile una nuova proprietà denominata `before_implicit;` al momento questa viene usata solo dal parser, quando sta per eseguire implicitamente il verbo PRENDI (ad esempio, MANGIA LA MELA quando non si possiede la mela). Potete dare questa proprietà ad un oggetto se volete che controlli il comportamento del parser. Il valore della proprietà dovrebbe essere una costante o una routine che ritorni: il valore 0 per riportare "(prima prendi il/la...)" e quindi tentare di compiere l'azione (ovvero ciò che accade di default); il valore 1 per tentare di PRENDERE l'oggetto senza stampare alcun messaggio; il valore 2 per procedere con l'azione richiesta senza tentare di PRENDERE l'oggetto; o infine il valore 3 per stampare il messaggio "Non possiedi niente del genere!". L'oggetto può testare l' action_to_be per determinare quale azione ha dato il via al tentativo di PRENDERE l'oggetto:

```
before_implicit [;
        Take: if (action_to_be == ##Eat) return 2;
],
```

• Una nuova variabile di sistema chiamata `sys_statusline_flag` è inizialmente impostata al valore 1 se avete usato nel vostro programma la direttiva `statusline time;` per mostrare un orologio, o altrimenti al valore 0. Può essere usata dal programma.

• La proprietà `invent` di un oggetto – se ne definisce una – viene invocata sia quando si mostra l'inventario del giocatore **sia** quando si include l'oggetto nella descrizione di una locazione. `invent` viene richiamata nel modo usuale (con `inventory_stage` prima impostato a 1, e quindi a 2) sia quando si menziona l'oggetto nella descrizione della locazione sia quando viene elencato nell'inventario del giocatore. Di default avrete lo stesso output ogni volta. Se avete bisogno di distinguere tra le due occasioni, potete testare `(c_style&PARTINV_BIT)` – condizione vera durante la descrizione della locazione – o `(c_style&FULLINV_BIT)` – vera invece durante un inventario. Ecco un esempio:

```
Object -> "sacco"
        with name 'sacco',
        invent [;
        ! Quando si elencano gli oggetti nell'inventario del giocatore:
        if (c_style&FULLINV_BIT) rfalse;
        ! Quando si elencano gli oggetti al termine della descrizione della locazione:
        if (inventory_stage == 1) switch (children(self)) {
```

```
            0: print "un sacco vuoto";
            1: print "un sacco contenente ", (a) child(self);
            default: print "un assortimento di oggetti in un sacco.";
    }
    rtrue;
    ],
    has container open;
```

Questa miglioria utilizza il meccanismo descritto in questa pagina:

`http://www.firthworks.com/roger/informfaq/ww.html#4`

e comporta che non vi è più necessità dell'estesione `WriteList`.

• Il contatore di turni `turns` ha ora come valore iniziale 0, e non 1. Potete modificare tale impostazione definendo la costante `START_MOVE` verso l'inizio del vostro gioco. Ad esempio in questo modo: `Constant START_MOVE 1;`

• E' stato definito anche un nuovo oggetto chiamato `LibraryExtensions`, la cui funzione è di agire come parent degli oggetti creati dalle estensioni alla libreria in fase di avvio. Questi oggetti possono prevedere routine nelle proprietà `ext_initialise` e/o `ext_messages`, il cui ruolo è quello di aiutare ad integrare l'estensione nel gioco. Per spiegare tale funzione è meglio servirci di un esempio:

Considerate l'estensione alla libreria `SmartCantGo.h`, che sostituisce il messaggio "Non puoi andare in quella direzione" con un messaggio più esaustivo "Puoi andare solo a nord, sud ed ad est", e che può essere integrata nel gioco aggiungendo l'istruzione `ChangeDefault(cant_go, SmartCantGo)` nella vostra routine `Initialise()`. Invece di richiedere all'autore di fare questa aggiunta, l'estensione potrebbe integrarsi ora automaticamente definendo un oggetto in fase di avvio come child dell'oggetto `LibraryExtensions`, in questo modo:

```
Object "(SmartCantGo)" LibraryExtensions
with ext_initialise [;
        ChangeDefault(cant_go, SmartCantGo);
        ];
```

Appena prima di chiamare la funzione `Initialise()` del gioco, la libreria esegue un ciclo tra i children –se ve ne sono– dell'oggetto `LibraryExtensions`, ed

esegue le proprietà `ext_initialise` che vi trova. Le routine delle proprietà possono eseguire qualsiasi impostazione appropriata che, altrimenti, avrebbe dovuto essere inserita nella stessa funzione `Initialise();` ad esempio, far partire l'esecuzione di un daemon.

Un procedimento simile ha luogo nella visualizzazione dei messaggi di libreria (library messages). La libreria all'inizio controlla se l'autore ha provveduto a definire un oggetto chiamato `LibraryMessages` per intercettare i messaggi che si accinge a visualizzare. Se non è presente tale oggetto, allora esegue un ciclo tra i children di `LibraryExtensions`, ed esegue le proprietà `ext_messages` che vi trova. Se nessuna di queste routine restituisce true per segnalare che il messaggio è già stato previsto, viene stampato il messaggio della libreria standard nel solito modo. Ad esempio, ecco come una estensione potrebbe automaticamente intercettare i messaggi di inventario (a meno che il gioco non li abbia già gestiti attraverso `LibraryMessages`):

```
Object  "(UnaQualsiasiEstensioneInventario)"
LibraryExtensions
        with ext_messages [;
                Inv: switch(lm_n) {
                    1: "Sei a mani vuote.";
                    2: "Il tuo patrimonio consiste di ";
                    }
                ];
```

Nota che questa è una caratteristica in fase di collaudo, e potrebbe essere modificata o estesa alla luce dell'esperienza che si maturerà con essa.

4.2 Le routine della Libreria

Alcune nuove routine di libreria sono state aggiunte per armonizzare le differenze che si incontrano generalmente tra la Z-machine e la Glulx VM[10].

. . .

`KeyCharPrimitive()`

Attende la pressione di un singolo tasto, e restituisce il carattere da 1 a 255 (o, per Glulx, uno degli speciali codici per carattere delle Glk). Solo per Glulx è disponibile una forma estesa, si veda a tal proposito il paragrafo *Le Routine di Libreria disponibili solo in Glulx*.

. . .

`KeyDelay(time)`

Attende `time` decimi di secondo per la pressione di un tasto. Se nessun tasto viene premuto restituisce zero; altrimenti ritorna il carattere del tasto con un valore da 1-255.

. . .

`ClearScreen()`

`ClearScreen(selector)`

`ClearScreen()` pulisce sia la status line che la finestra principale (main window). Il cursore si sposta in cima allo schermo. La routine dovrebbe essere seguita da una chiamata alla funzione `MoveCursor ()` o `MainWindow()`. Alternativamente, può essere usata la routine `ClearScreen(selector)`: se `selector` è 0, allora entrambe sono pulite; se `selector` è 1 allora viene pulita solo la status line; se `selector` è pari a 2 allora viene pulita solo la finestra principale.

. . .

`MoveCursor ()`

`MoveCursor (line, column)`

`MoveCursor()` seleziona la status line come output. `MoveCursor (line, column)` seleziona la status line come output e sposta il cursore alla data linea `line` e colonna `column` all'interno della status area, dove la linea 1 è quella in cima e la colonna 1 è quella più a sinistra. (Tale specificazione è necessaria dal momento

[10] Ulteriori informazioni possono essere trovate al paragrafo 5.3.2 *Le Routine di Libreria disponibili solo in Glulx*.

che la convenzione Glk è invece di numerare sia le linee che le colonne partendo da 0 e non da 1)

. . .

`MainWindow()`

Seleziona la finestra principale di testo per l'output.

. . .

`StatusLineHeight (lines)`

Imposta l'altezza in linee della statusline. La routine standard `DrawStatusLine()` richiama questa funzione ad ogni turno, il che non è male, dal momento che `StatusLineHeight()` è una funzione veloce. Potete sostituire la `DrawStatusLine()`, mantenendo tale convenzione. (Le routine di libreria per i menu giocherellano con la status line, ed è importante che la funzione `DrawStatusLine()` la ridisegni dopo la chiusura del menu.)

Una nuova variabile di libreria chiamata `gg_statuswin_cursize` contiene l'impostazione della grandezza corrente della status line (in entrambe le VMs).

. . .

`ScreenWidth()`

Restituisce il numero di caratteri che possono essere stampati con un font monospace dal bordo destro a quello sinistro della finestra selezionata al momento. Solo per Glulx, la forma estesa `ScreenWidth(win)` funziona su di un id[11] di una finestra specificata; tenete conto che i risultati sono inaffidabili se lo stile normale del testo per la finestra prevede un font proporzionale.

. . .

`ScreenHeight()`

Restituisce l'altezza in linee della finestra principale.

. . .

`SetColour(fg, bg)`

`SetColour(fg, bg, selector)`

`SetColour(fg, bg)` imposta i colori correnti di primo piano e di sfondo del testo, usando gli stessi codici come gli opcode `@set_colour` nella Z-machine (1=default, 2=nero, 3=rosso, 4=verde etc.). Usando invece `SetColour(fg, bg, selector)`, i colori possono essere impostati separatamente per ogni finestra: se

[11] Nome identificativo. NdT.

`selector` è 0, entrambe sono impostate (la finestra in cima avrà i colori invertiti nella Z-machine); se `selector` è 1 solo la status line verrà modificata; se `selector` è 2 verrà impostata solo la finestra principale. Tutte le funzioni sul colore sono efficaci solo se la variabile di libreria `clr_on` è impostata ad un valore diverso da zero. Il vantaggio rispetto a `@set_colour` è che quando il giocatore carica una partita salvata precedentemente o digita UNDO, i colori saranno corretti esattamente come erano a quel punto del gioco.

Per Glulx, la routine produce un effetto appropriato se i suggerimenti di stile per il testo (style hints) sono abilitati dall'interprete; la funzione pulisce anche lo schermo. Per la Z-machine, è invece necessario richiamare separatamente anche la funzione `ClearScreen()`. Sono state definite le seguenti costanti per essere usate con `SetColour();` le ultime tre possono essere utili anche con la funzione `ClearScreen()`:

```
Constant CLR_DEFAULT 1;
Constant CLR_BLACK 2;
Constant CLR_RED 3;
Constant CLR_GREEN 4;
Constant CLR_YELLOW 5;
Constant CLR_BLUE 6;
Constant CLR_MAGENTA 7;
Constant CLR_CYAN 8;
Constant CLR_WHITE 9;
Constant CLR_PURPLE 7;
Constant CLR_AZURE 8;
Constant WIN_ALL 0;
Constant WIN_STATUS 1;
Constant WIN_MAIN 2;
```

. . .

```
DecimalNumber(num)
```

Stampa `num` come un numero decimale (è nei fatti identica a `print num;`). Può essere usata con profitto insieme a...

. . .

```
PrintToBuffer(array, arraylen, string)
PrintToBuffer(array, arraylen, obj)
PrintToBuffer(array, arraylen, obj, prop)
PrintToBuffer(array, arraylen, routine, arg1, arg2)
```

Tali funzioni stampano i loro argomenti – una stringa, il nome di un oggetto, il valore della proprietà di un oggetto, o una routine con più di due argomenti – nel buffer `array`. Il numero di caratteri scritti nel buffer si trova in `array-->0` (ed è il valore restituito dalla routine); i veri caratteri cominciano all'ingresso dell'array pari a `array->WORDSIZE`. Il numero massimo di caratteri è specificato in `arraylen`; per la Z-machine, una eccedenza causata dalla stampa di più caratteri di tale limite produce un messaggio di errore che informa l'autore che ha corrotto i contenuti della memoria che sta dietro all'array (per Glulx, l'output è automaticamente troncato al limite specificato in `arraylen`). Per Glulx, si vedano anche la routine `PrintAnyToArray()` nel paragrafo 5.3.2 *Routine di Libreria disponibili solo in Glulx*, che ha lievemente espanso le sue capacità, e che restituisce il numero di caratteri scritti invece che scriverli in `array-->0`.

. . .

```
Length(string)
Length(obj, prop)
```

Restituiscono il numero di caratteri presenti in una stringa. Considerate che il risultato viene scritto in uno degli array del parser, e che quindi è vostra responsabilità assicurarvi che la lunghezza di tale stringa **non sia maggiore di 160 caratteri**.

. . .

```
UpperCase(char)
LowerCase(char)
```

Restituiscono `char` in maiuscole o in minuscole (se si parla di lettere alfabetiche), o senza cambiamenti (in qualsiasi altro caso). I cambiamenti apportati alle lettere dalla A-Z e dalla a-z sono sempre affidabili. I cambiamenti apportati alle lettere accentate potrebbero non funzionare se avete selezionato l'opzione (switch) da `-C2` a `-C9`, o usato la direttiva `Zcharacter` per sistemare il set di caratteri standard ZSCII.

. . .

```
PrintCapitalised(obj, prop, flag, nocaps)
```

Si basa sulla funzione `PrintOrRun(obj, prop, flag)`. `PrintOrRun()` testa l'esistenza di `obj.prop`, e quindi la esegue (se è una Routine) o la stampa (se è una Stringa). Nel secondo caso, viene stampata anche una nuova linea a meno che `flag` non sia impostato a `true`. `PrintCapitalised()` fa tutto ciò con la differenza che la prima lettera di qualsiasi output è maiuscola a meno che `nocaps` sia impostato a `true`.

. . .

```
Cap(string, nocaps)
```

Stampa la stringa `string` con la prima lettera in maiuscolo, a meno che `nocaps` sia true. Può anche essere utilizzata come regola di stampa: `print ..., (Cap) myString, ...;`

. . .

```
Centre(string)
Centre(obj, prop)
```

Stampa una stringa `string` che occupa una singola linea con il testo approssimativamente centrato tra il bordo destro e quello sinistro dello schermo facendolo precedere da un quantitativo opportuno di spazi. La routine funziona unicamente con font monospace (cioè, dopo `font off;`), e funzionerà bene solo se nella finestra principale di testo di Glulx lo stile normale usa un font non proporzionale. E' comunque utile per centrare le informazioni nella status line. Può anche essere usata come regola di stampa: `print ..., (Centre) myString, ...;`

. . .

```
PrintOrRunVal(value, flag)
```

se `value` si riferisce ad un oggetto, stampa il nome dell'oggetto; se `value` si riferisce ad una routine, esegue tale routine; se `value` si riferisce ad una stringa, stampa tale stringa (terminandola con l'inserimento di una nuova linea a meno che `flag` sia true).

4.3 Correzione dei Bug

I numeri nella forma [L61036] sono riferimenti alla lista di Patch della sezione Libreria di Inform.

• Un comando come `EMPTYME` non replica più con un messaggio del tipo "Non puoi contenere delle cose". [L61036]

• I comandi PRENDI TUTTO DA X e TOGLI TUTTO DA X, dove X è un contenitore aperto o chiuso, ora producono messaggi più appropriati rispetto alle vecchie risposte, rispettivamente, "Non vedi niente del genere" e "Non puoi usare oggetti multipli con questo verbo". [L61035]

• E' stato corretto un problema con il malfunzionamento della proprietà `name` delle locazioni, usata insieme a THE. [L61034]

• Il comando METTI X DENTRO X ora produce correttamente la risposta "Non puoi mettere qualcosa destro se stessa", invece di "Non vedi niente del genere". [L61033]

• Gli errori in fase di esecuzione del gioco prodotti da `IndirectlyContains()` mentre tenta di trovare il parent di una Classe che supporta la creazione dinamica di oggetti sono stati risolti. [L61032]

• Il codice in `Parser__parse()` che si occupa di prevedere a quale oggetto indiretto ci si riferisce in casi come METTI TUTTO NELLA BORSA (un token MULTIEXCEPT)[12] e PRENDI TUTTO DALLA BORSA (un token MULTIINSIDE) ora correttamente imposta la variabile globale `advance_warning` (a BAG). [L61031, L61023]

• L'*Inform Designer's Manual* (p. 98) riporta che SHOWOBJ dovrebbe accettare il numero dell'oggetto; ora lo fa. [L61030]

[12] Ovvero deve prendere in considerazione tutti gli oggetti in scope con l'eccezione della borsa stessa. NdT.

• La routine `YesOrNo()` ora ristampa il prompt correttamente dopo un input scorretto. [L61029]

• Il parse buffer non è più dichiarato e inizializzato scorrettamente (sebbene fosse innocuo). [L61028,L60708]

• L'*Inform Designer's Manual* (p. 93) definisce l'ordine di chiamata delle routine e delle proprietà nella fase 'Before' come segue:

1. `GamePreRoutine()`
2. `orders` del giocatore
3. `react_before` di ogni oggetto in scope
4. `before` dell'attuale locazione
5. `before` del primo noun, se specificato

Nella libreria, invece, il punto tre e quattro erano eseguiti in ordine inverso. Ora funzionano come da documentazione. [L61027]

• Un oggetto fluttuante con la proprietà `found_in` che può essere preso dal giocatore (probabilmente a causa di un errore di programmazione) non viene più lasciato silenziosamente quando il giocatore ritorna in una delle locazioni previste nella proprietà. [L61026]

• E' stato corretto un piccolo problema con l'ereditarietà della proprietà `describe`. [L61025]

• La gestione standard dello schermo è stata implementata anche in giochi v6. [L61022]

• La gestione dei messaggi "Non puoi andare da quella parte" è stata resa più coerente.

Inoltre, l'istruzione `ChangeDefault(cant_go,myRoutine);` ora funziona correttamente. [L61020]

• Il tentativo di porre un oggetto dentro o sopra un altro oggetto dove già si trova restituisce il messaggio "E' già lì", invece di "Devi possederlo prima di poterlo mettere dentro qualcos'altro". [L61019]

• Un problema con un ingannevole elenco dell'inventario è stato chiarito. [L61018]

• Il commando VAI VIA DA X ora correttamente produce "Ma tu non sei dentro o sopra X", se appropriato. [L61017]

• La risposta a LEGGI era inappropriata quando un oggetto era mal digitato o fuori scope. [L61016]

• E' stato corretto un piccolo bug nella scelta dei messaggi di libreria per SPINGI e GIRA, che non era evidente se non si sovrascrivevano i messaggi per essere differenti da TIRA.. [L61015]

• Se siete in una locazione buia, non potete esaminare ciò che possedete. Tuttavia se aprite un contenitore che avete portato da una locazione illuminata, il messaggio standard "Aprite la scatola,rivelando un..." non era soppresso. [L61014]

• La routine `ScoreMatchL()` in Parserm.h decideva scorrettamente quale oggetto corrispondeva al descrittore [il qualificatore: tipo LIT LANTERN con LIT che è il descrittore] (descriptors). Come risultato, alcuni oggetti che non corrispondevano al descrittore (il qualificatore) non venivano rimossi del tutto dalla match list quando la libreria doveva decidere quale degli oggetti meglio incontrava l'input del giocatore. [L61013]

• Il problema di parsing di comandi di Infix contenenti virgole e punti è stato risolto. [L61010]

• Un problema con la descrizione di cosa è visibile dopo l'apertura di un contenitore è stato corretto. [L61009]

• E' stato modificato un messaggio inappropriato dopo il comando GO NORTH CIRCULAR. [L61008]

• Dopo il CARICAMENTO di una partita o il comando UNDO i colori di primo piano e di sfondo modificati ora vengono ripristinati correttamente. [L61007]

• La proprietà `grammar` ora funziona con giochi ampi il cui dizionario supera $8000. [L61006]

• Un conflitto di buffer tra disambiguation e UNDO è stato corretto. [L61004]

• Se un giocatore è all'interno di un contenitore chiuso e non trasparente, la libreria stampava un ulteriore linea Bianca tra il titolo "Il contenitore" e la prima linea della `inside_description`. Ora non più. [L61002]

• Le routine che scrivono elenchi non gestivano correttamente più contenitori. Se si avevano due scatole vuote, avrebbe dovuto essere riportato "due scatole (che sono chiuse)". E non "(che è chiusa)" come accadeva. Non solo ora anche se una delle due è chiusa e l'altra aperta esse comunque vengono raggruppate. [L61001]

• Un conflitto tra le routine `DrawStatusLine()` e `DisplayStatus()` su come debba essere determinato se visualizzare i turni o l'orologio è stato appianato con il controllo di una header flag. [L60709]

§5 Il supporto per Glulx

Una delle limitazioni della Z-machine è che la grandezza massima dei giochi più grandi è di 256 Kb se si compila per la versione 5, o di 512 KB se si compila per la versione 8. Se vi scontrate con tali limiti ed avete provato già tutti i normali trucchi per risparmiare qualche byte qui e la, allora è giunto il momento di passare a Glulx. Il che è piuttosto facile da fare: basta aggiungere l'opzione `-G` nella linea di comando e il compilatore genererà codice Glulx, che sarà possibile giocare su un qualsiasi interprete Glulx[13]. In realtà le cose non sono sempre così semplici. Qui di seguito andremo a vedere: *Cosa accade in Glulx*, *Le differenze tra Glulx e Z-machine* e *Cosa offre in più Glulx*.

5.1 Cosa accade in Glulx

Come menzionato in precedenza, il compilatore automaticamente definisce un paio di costanti utili. Se state compilando per la Z-machine allora viene definita la costante `TARGET_ZCODE`; se invece state compilando per Glulx allora sarà definita la costante `TARGET_GLULX`. Potete usare tali costanti con la direttiva `Ifdef`, in questo modo:

```
#Ifdef TARGET_ZCODE;
!       il codice per la Z-machine
#Endif;
#Ifdef TARGET_GLULX;
!       il codice equivalente per Glulx
#Endif;
```

O anche in quest'altro modo:

```
#Ifdef TARGET_ZCODE;
!       il codice per la Z-machine
#Ifnot;
!       il codice equivalente per Glulx
#Endif;
```

[13] Al momento esistono diversi interpreti per Glulx e coprono tutte le piattaforme più diffuse.

Se andate a scorrere il testo dei file della libreria troverete molte di queste direttive, ma ne avrete molto meno bisogno all'interno del vostro codice sorgente.

5.2 Le differenze tra Glulx e la Z-machine

Le due VM non sono identiche, e dovete fare attenzione alle differenze che intercorrono tra esse. In particolar modo vedremo che esse si incentrano su: *Grandezza delle parole*, *Direttive*, *Istruzioni*, *Gestione dei caratteri* e *Opzioni di compilazione*.

5.2.1 La grandezza delle parole (wordsize)

La principale differenza tra Glulx e la Z-machine è che le parole (word) sono lunghe quattro byte invece che due. Tutte le variabili sono valori a 32-bit, lo stack contiene valori a 32-bit, le proprietà sono valori a 32-bit, e così via.

Nella maggior parte della programmazione in Inform, non ci si deve preoccupare di questo cambiamento. Per esempio, se si ha un array

```
Array mylist --> 10;
```

... allora lo Z-code Inform alloca 10 parole (word) – ossia, venti bytes – e si può accedere a questi valori attraverso `mylist-->0` fino a `mylist-->9`. Se si compila lo stesso codice in Glulx Inform, il compilatore allocherà ancora una volta dieci parole (word), ovvero quaranta byte, e si potrà ancora accedere ai valori come mylist-->0 fino a `mylist-->9`. Ogni cosa funzionerà alla stessa maniera, con l'eccezione che l'array potrà contenere valori più grandi del limite della Z-machine pari a 65535.

Anche i Table array usano valori di parola (word) di due o quattro byte, e la prima parola (word) è la lunghezza dell'array. `String` e -> arrays, e la notazione -->, si riferiscono ancora a singoli byte. Non vi dovrebbe essere alcuna necessità di modificare il proprio codice qui.

Vi sono due casi importanti in cui **dovrete** modificare il vostro codice. Il primo è l'operatore `.#` L'espressione `obj.#prop` restituisce la lunghezza della proprietà **in byte**.

Dal momento che la maggior parte delle proprietà contiene parole, piuttosto che byte, è molto comune avere un codice per la Z-machine di questo tipo:

```
len = (obj.#prop) / 2;
for (i=0 : i<len : i++)
        print (string) (obj.&prop)-->i;
```

Nei programmi Glulx Inform, è necessario dividere per 4 invece che per 2. Dovreste quindi sostituire il precedente codice con quello seguente,che sfrutta la costante `WORDSIZE`:

```
len = (obj.#prop) / WORDSIZE;
for (i=0 : i<len : i++)
        print (string) (obj.&prop)-->i;
```

Questo codice verrà compilato ed eseguito correttamente in entrambe le VM.

La seconda circostanza in cui potrebbe essere necessario modificare il vostro codice è quando utilizzate la caratteristica 'print to array'. Un codice come questo:

```
! l'array deve essere grande abbastanza da contenere
! la stringa più lunga che intendete stampare:
Array mybuf buffer 100; !100 è un buon numero
mystr = "hello";
mystr.print_to_array (mybuf);
```

ha come risultato che la prima **parola** (word) di `mybuf` contiene 5 (il numero di caratteri in `mystr`), e i seguenti cinque **bytes** contengono le lettere 'h', 'e', 'l', 'l' e 'o'. Nella Z-machine, a questo punto potete estrarre i caratteri dall'array con uno dei seguenti frammenti di codice:

```
len = 2 + mybuf-->0
for (i=2 : i<len : i++)
        print (char) mybuf->i;

len = mybuf-->0
for (i=0 : i<len : i++)
        print (char) mybuf->(i+2);
```

Ancora una volta, per fare in modo che il codice funzioni correttamente con entrambe le VM, è necessario sostituire al "`2`" la costante "`WORDSIZE`":

```
len = WORDSIZE + mybuf-->0
for (i=WORDSIZE : i<len : i++)
        print (char) mybuf->i;

len = mybuf-->0
for (i=0 : i<len : i++)
        print (char) mybuf->(i+WORDSIZE);
```

Una versione estesa della forma `print_to_array` è descritta nel paragrafo 5.3.2 *Caratteristiche disponibili solo in Glulx*.

5.2.2 Le Direttive

Glulx è capace di gestire la maggior parte delle direttive di Inform, con due eccezioni:

• La direttiva `Zcharacter` che causa l'errore di compilazione "Glulx Inform does not handle Unicode yet"[14]. Il messaggio, sebbene corretto, potrebbe trarre in inganno; il suo significato reale è che Glulx non usa il set di caratteri ZSCII, che può essere in parte configurato in varie forme da `Zcharacter`. Il miglior approccio a tale situazione è quello di aggirare la direttiva quando si compila in Glulx:

```
#Ifdef TARGET_ZCODE;
        Zcharacter ... ;
#Endif;
```

• La direttiva (obsoleta) `Lowstring` causa un errore in fase di esecuzione quando è usata in questo modo:

```
Lowstring mystr "hello";
string 0 mystr;
print "@00 and goodbye.";
```

La successive forma, evitando l'uso di `Lowstring`, funziona egregiamente:

```
string 0 "hello";
print "@00 and goodbye.";
```

Per maggiori informazioni sulle variabili di stampa in Glulx si consulti il paragrafo *Caratteristiche disponibili solo in Glulx*.

[14] Recentemente Andrew Plotkin ha rilasciato un aggiornamento delle librerie Glk che supportano l'Unicode. C'è da aspettarsi che in un prossimo futuro tale supporto sarà esteso a Glulx ed ai suoi interpreti. NdT.

5.2.3 Le Istruzioni

Glulx è capace di gestire la maggior parte delle istruzioni (statement) di Inform, con poche eccezioni. Se si prova ad usare alcune di queste in Glulx, vi verrà segnalato un errore in fase di compilazione:

• `save`, `restore`: Tali procedure sono molto più complicate in Glulx rispetto allo Z-code, e non possono essere implementate senza non coinvolgere variabili e routine di libreria. Se volete fare qualcosa del genere dovrete modificare o copiare le routine di libreria chiamate `SaveSub()` e `RestoreSub()`.

• `read`: Allo stesso modo, leggere una linea di testo in Glulx coinvolge la libreria; il compilatore non può generare da solo codice indipendente per farlo. Si veda invece la routine `KeyboardPrimitive()`.

• `@opcode`: Il linguaggio assembly della Z-machine è completamente differente da quello di Glulx. Se avete usato delle istruzioni in assembly per Z-machine, dovrete nasconderle quando compilate per Glulx, come descritto nel paragrafo *Cosa accade in Glulx*. Si veda inoltre il paragrafo *Caratteristiche disponibili solo in Glulx* per i dettagli delle chiamate alla funzione `glk()`.

Glulx ora supporta anche l'istruzione style di Inform, dirottando le forme di stile della Z-machine agli stili delle Glk:

Istruzioni Inform	Equivalente stile in Glk
style roman;	style_Normal
style reverse;	style_Alert
style bold;	style_Subheader
style underline;	style_Emphasized
style fixed;	style_Preformatted

Comunque, è importante ricordare che, con Glulx, l'aspetto del gioco è sotto il controllo del giocatore. L'interprete Glulxe abilita i parametri dei font associati ad ogni stile in fase di esecuzione del gioco. Quindi, dovreste prestare attenzione all'uso degli stili, e se necessario informare il giocatore su quali impostazioni si aspetta il vostro gioco.

5.2.4 La gestione dei caratteri

Diversamente dalla Z-machine, che usa internamente il set di caratteri ZSCII (si veda la Tabella 2 dell'*Inform Designer's Manual* a p. 519), Glulx fa affidamento sulla codifica ISO 8859-1 (Latin-1). Questo schema a otto bit è lo stesso dei caratteri ZSCII per valori compresi tra 0-127, ma differisce per i restanti caratteri compresi tra i valori 128-255. Si ponga attenzione al fatto che Glulx è costruito sulla libreria I/O Glk, che al momento non supporta caratteri Unicode[15] (con l'eccezione che i punti $0000 fino a $00FF sono identici a ISO 8859). Le conseguenze di tale differenziazione sono le seguenti:

• `@escape_sequence`: le sequenze di caratteri (escape sequences) come `@:a` e `@LL` (per "ä" e "£" rispettivamente) sono accettate allo stesso modo da entrambe le VM (con l'eccezione che Glulx non gestisce `@oe` e `@OE`, dal momento che il carattere "oe" e "OE" formato da due lettere è al di fuori dal set di caratteri ISO 8859-1).

• `@@decnum`: Il numero è il valore decimale interno del carattere, così `@@65` è "A" in entrambe le VM, ma `@@165` è "í" nella Z-machine e "¥" in Glulx.

• `@{hexnum}`: Il numero è il valore Unicode del carattere, quindi `@{41}` è "A" e `@{EF}` è "ï" in entrambe le VM. Naturalmente, `@{A5}` è "¥" in Glulx, ma causa un errore di compilazione nella Z-machine dal momento che "¥" non è un carattere compreso nel set ZSCII, mentre `@{152}` è "OE" nella Z-machine ma causa un errore di compilazione in Glulx dal momento che tale carattere non è compreso nel set ISO 8859-1.

• `-Cn`: Le opzioni di compilazione che vanno da `-C1` fino a `-C9` specificano che il file sorgente usa il set di caratteri definito rispettivamente dal set ISO 8859-1 fino al set 8859-9. Nella Z-machine, l'opzione inoltre imposta il migliore set di caratteri ZSCII appropriato a tale valore; questa caratteristica è irrilevante quando si compila per Glulx.

---

[15] Come si riportava in una nota precedente, recentemente Plotkin ha distribuito le librerie Glk 0.70 che a differenza delle precedenti supportano l'Unicode.

5.2.5 Opzioni di compilazione

Abbiamo già menzionato l'opzione -G per la linea di comando, che fa in modo che la compilazione generi codice per Glulx invece che per Z-machine. Vi sono pochi altri cambiamenti in questa area (si veda la tabella 3 dell'*Inform Designer's Manual* a p. 521).

Opzione	Valore	Significato
-k	off/on	incompatibile con -G (output debugging information)
-v*	da 3 a 8	incompatibile con -G (set Z-machine Version)
-G	off/on	compila per la Glulx VM
-H	on/off	usa la compressione Huffman sulle stringhe di Glulx (on di default)
-X	off/on	incompatibile con -G (include Infix debugger)

5.3 Cosa offre in più Glulx

Glulx offre alcune possibilità in più rispetto a quelle della Z-machine. Esse possono essere distinte in *Caratteristiche disponibili solo in Glulx*, *Routine di Libreria disponibili solo in Glulx* ed *Entry points disponibili solo in Glulx*.

5.3.1 Caratteristiche disponibili solo in Glulx

• La metaclass String di Inform prevede print_to_array:

```
str = "Questa è una stringa.";
len = str.print_to_array (arr);
```

Queste due istruzioni scriveranno il contenuto della stringa nell'array, a cominciare dal byte 2. Il primo termine (arr-->0) conterrà il numero di caratteri scritti, e lo passerà a len. In Glulx il funzionamento di print_to_array è il medesimo, con la solita eccezione che la stringa comincia al quarto byte.

In Glulx esiste però anche una forma estesa di `print_to_array`:

```
len = mystr.print_to_array (mybuf, 80);
```

Questo esempio scrive non più di 76 caratteri nell'array. Se `mybuf` è un array di 80 byte, avrete la sicurezza che la dimensione dell'array non verrà superata (overrun). (Non provate questo esempio con il secondo argomento impostato ad un valore inferiore a 4).

Il valore scritto in `mybuf-->0`, ed il valore restituito (posto in len), **non** sono limitati al numero di caratteri scritti nell'array, ma riportano il numero di caratteri della stringa al suo completo (anche se più lunga dell'array). Ciò significa che:

```
len = mystr.print_to_array (mybuf, 4);
```

...è un pessimo ma perfettamente valido modo di trovare la lunghezza di una stringa. (E in questo caso, `mybuf` ha solo bisogno di essere di almeno 4 byte di lunghezza).

• Lo Z-code di Inform supporta 32 variabili di stampa, da @00 a @31, che possono essere incluse nelle stringhe e impostate con l'istruzione:

```
string num "valore";
```

In Glulx, questo limite è stato elevato a 64. Inoltre, in Glulx è possibile impostare queste variabili perché puntino a Routine indipendenti esattamente come fanno per le stringhe:

```
[ routine; print "valore"; ];
string num routine;
```

In questo caso, la routine viene chiamata senza argomenti ed il valore restituito viene cancellato; dovrete quindi stampare l'output che desiderate all'interno della routine. In Glulx, a differenza dello Z-code, una variabile di stampa può contenere anche altri codici @.. , permettendo in tal modo la ricorsione. Potete nidificare la procedura quante volte volete. D'altra parte, è certamente una pessima idea quella di produrre una ricorsione infinta. Ad esempio, il seguente codice bloccherà sicuramente l'interprete:

```
string 3 "Questo è un @03!";
print "Cosa è un @03?";
```

• Molte cose che sono solitamente implementate con l'assembly dello Z-code ora vengono gestite dalle funzioni della Glk. Eseguire una chiamata alle funzioni della Glk da Inform è un po' stravagante, ma non difficile.

Ogni cosa nelle Glk è gestita dalla funzione `glk()`, già presente in Inform, che accetta uno più argomenti. Il primo argomento è un numero intero (integer); che dice **quale** funzione Glk si sta richiamando. I rimanenti argomenti sono semplicemente, nell'ordine, passati come argomenti alla funzione Glk.

Mettiamo, ad esempio, che vogliate impostare lo stile del testo su "preformatted". Il codice Inform per realizzare tale impostazione è:

```
glk($0086, 2);
```

Il valore esadecimale $0086 sta per `glk_set_style;` il valore "2" invece per `Preformatted`.

La tabella delle funzioni Glk, e i numeri interi che si riferiscono ad esse, sono nelle specifiche Glk (Ricordate che i valori lì presenti sono esadecimali). Si veda:

`http://www.eblong.com/zarf/glk/glk-spec-061_11.html#s.1.6`

Dal momento che i codici numerici non sono proprio semplici da leggere, vi raccomandiamo di scaricare la libreria aggiuntiva `infglk.h` di John Cater[16], che definisce delle funzioni contenitore e delle costanti. Così ad esempio, potreste chiamare con il medesimo effetto della precedente chiamata diretta alle Glk:

```
glk_set_style(style_Preformatted);
```

Quando leggete le specifiche Glk, tenete a mente che il valore `NULL` si riferisce al `NULL(0)` del linguaggio C, non all'omonimo `NULL(-1)` della Libreria di Inform.

`infglk.h` definisce una costante `GLK_NULL` (pari a 0) che potete usare dove ritenete appropriato.

---

[16] Questo volume offre un'ampia disamina della libreria infglk, sia attraverso i capitoli che vanno dal 7 al 10, sia attraverso una completa guida di riferimento alle funzioni messe a disposizione da infglk che trovate in appendice. L'autore di giochi in Glulx non dovrebbe trovare necessario rivolgersi alle (molto tecniche) specifiche glk a meno di non voler fare cose molto particolari.

• Di default, gli argomenti passati alle routine funzionano allo stesso modo sia in Glulx che in Zcode. Quando chiamate una routine, gli argomenti che passate sono scritti in ordine nelle variabili locali della routine stessa. Se passate troppi argomenti, quelli in più vengono scartati; se ne passate di meno, le variabili locali non utilizzate assumono valore zero.

In ogni caso, la VM Glulx supporta un secondo tipo di funzioni. Potete definire una funzione di questo tipo dando `_vararg_count` come nome del primo argomento della funzione. Ad esempio:

```
[ StackFunc _vararg_count ix pos len;
        ! Il codice Glulx
];
```

Se fate questo, gli argomenti della funzione **non** verranno scritti nelle variabili locali. Essi saranno inseriti nello stack, e dovrete usare l'assembly di Glulx per estrarli. Tutte le variabili locali saranno inizializzate a zero, con l'eccezione di `_vararg_count`, che (come potrete immaginare) contiene il numero di argomenti che vengono passati.

Notate che `_vararg_count` è una normale variabile locale, all'infuori del suo valore iniziale. Potrete assegnargli un valore, incrementarla o decrementarla, usarla nelle espressioni, e così via.

Le routine con gli argomenti nello stack (stack-argument routine) sono molto utili se si vuole una funzione con argomenti variabili, o se volete scrivere una funzione contenitore che passi i suoi argomenti ad una ulteriore funzione[17].

[17] I lettori più attenti potrebbero notare che questa non è esattamente la struttura che le specifiche della Glulx VM descrive. Una funzione C0-type comincia con un valore extra sullo stack, che specifica il numero di argomenti che lo seguono sullo stack. Naturalmente, ciò è nascosto ai vostri occhi dal compilatore. Quando scrivete una funzione _vararg_count il compilatore genera automaticamente il codice per far uscire questo valore dallo stack e inserirlo nella variabile locale _vararg_count.

5.3.2 Routine di Libreria disponibili solo in Glulx

```
KeyCharPrimitive(win, nostat);
```

Se `win` è diverso da zero, il carattere immesso va alla finestra Glk specificata (invece di `gg_mainwin`, finestra di default.) Se `nostat` è diverso da zero l'evento arrangiamento della finestra è restituito immediatamente come valore 80000000 (invece del comportamento di default, che consiste nel chiamare la `DrawStatusLine()` ed aspettare).

. . .

```
PrintAnything(thingie, ...);
```

Nella Z-machine, le stringhe e le routine sono indirizzi "packed" (impacchettati), le parole di dizionario sono indirizzi normali, e gli oggetti del gioco sono rappresentati come sequenze di numeri da 1 a `#top_object`. Questi intervalli [di indirizzi] si sovrappongono; una stringa, una parola di dizionario, ed un oggetto possono perfettamente essere tutti rappresentati dallo stesso valore numerico.

In Glulx, tutte le suddette situazioni sono rappresentate da indirizzi normali, pertanto cose diverse saranno sempre rappresentate da valori numerici diversi. Inoltre, il primo byte trovato all'indirizzo è un valore identificativo, che specifica il genere del contenuto che sta dietro l'indirizzo stesso.

`PrintAnything()` stampa qualsiasi `thingie` (cosa) – che sia una stringa, una routine (con eventuali argomenti), un oggetto, la proprietà di un oggetto (con eventuali argomenti), o una parola del dizionario – conosciuto alla libreria.

Chiamare:	E' equivalente a:
PrintAnything()	<non viene stampato niente>
PrintAnything(0)	<non viene stampato niente>
PrintAnything("string");	print (string) "string";
PrintAnything('word')	print (address) 'word';
PrintAnything(obj)	print (name) obj;
PrintAnything(obj, prop)	obj.prop();
PrintAnything(obj, prop, args...)	obj.prop(args...);
PrintAnything(func)	func();
PrintAnything(func, args...)	func(args...);

Argomenti extra dopo una stringa `string` o una parola del dizionario `word` sono sicuramente ignorati.

Il (primo) argomento che si passa alla routine è sempre interpretato come un riferimento ad una cosa (thingie), non come un numero intero (integer). Ciò spiega il perchè nessuna delle forme precedentemente mostrate stampa un numero intero (integer). Naturalmente, si può ottenere lo stesso effetto chiamando

```
PrintAnything(DecimalNumber, num);
```

...ed è qui che torna utile la funzione `DecimalNumber()`. Si possono anche usare, naturalmente, altre funzioni della libreria, e fare trucchi tipo

```
PrintAnything(EnglishNumber, num);
PrintAnything(DefArt, obj);
```

Nessuno di questi sembra molto utile. Dopo tutto, ci sono già dei modi per stampare tutte queste cose. Ma `PrintAnything()` è fondamentale per implementare la seguente funzione:

. . .

```
PrintAnyToArray (array, arraylen, thingie, ...);
```

Questa funzione lavora allo stesso modo, con l'eccezione che invece di stampare a schermo, l'output è trasferito nell'array nominato.

I primi due argomenti devono essere l'indirizzo dell'array e la sua massima lunghezza. Fino ad essa i caratteri verranno scritti nell'array; qualsiasi carattere extra sarà silenziosamente scartato. Ciò significa che non dovrete preoccuparvi di eventuali superamenti di limiti (overrun) dell'array.

La funzione `PrintAnyToArray()` restituisce il numero di caratteri generati. (Che potrebbero essere di più della lunghezza dell'array. Rappresenta l'intero testo che viene dato in output, non il numero limite scritto nell'array.)

Si possono annidare le fuzioni `PrintAnyToArray()`. Cioè, potete richiamare `PrintAnyToArray(routine)`, dove `routine()` è una funzione che richiama a sua volta `PrintAnyToArray()`. (Naturalmente, se entrambe dovessero cercare di scrivere lo **stesso array**, il caos sarebbe assicurato.)

E' consentito che `arraylen` sia pari a zero (nel qual caso l'`array` è ignorato, e può essere zero tranquillamente.) Ciò scarta **tutto** l'output, e semplicemente restituisce il numero di caratteri generati. Potete usare questo artifizio per trovare la lunghezza di qualsiasi cosa -- anche di una chiamata ad una funzione.

5.3.3 Entry points disponibili solo in Glulx

Una entry point è una funzione che si può inserire o meno nel proprio codice; la libreria la richiamerà solo se è presente, e la ignorerà nel caso opposto. (Vedi il paragrafo §21 dell'*Inform Designer's Manual*).

La libreria possiede alcune entry point che aiutano l'autore nella scrittura di interfacce più complicate del solito -- giochi con suoni, disegni, finestre supplementari, ed altri simpatici trucchetti offerti dalla Glk[18]. Se state scrivendo un gioco in semplice stile Infocom standard, **potete ignorare questa sezione**.

. . .

`HandleGlkEvent(ev, context, abortres)`

Questa entry point è chiamata ogni volta che accade un evento Glk[19]. L'evento potrebbe indicare qualunque cosa, una linea di input del giocatore, il ridimensionamento di una finestra o il ridisegno di una immagine, un rintocco dell'orologio, un click del mouse e così via.

La libreria gestisce tutti questi eventi necessari per un normale gioco in stile Infocom ed è necessario prevedere una funzione `HandleGlkEvent()` solo se si vuole aggiungere qualche funzionalità extra.

L'argomento `ev` è un array di quattro parole (word) che descrive l'evento. `ev-->0` è il tipo di evento; `ev-->1` è la finestra coinvolta (se rilevante); `ev-->2` ed `ev-->3` sono informazioni extra. L'argomento `context` è 0 se l'evento accade durante la linea di input (comandi normali, `YesOrNo()`, o qualche altro uso della funzione di libreria `KeyboardPrimitive();`) o 1 se l'evento è accaduto durante la pressione di un carattere input (qualsiasi uso della funzione di libreria `KeyCharPrimitive();`).

---

[18] Le funzioni descritte in questo paragrafo sono piuttosto complesse, un approfondimento sul loro funzionamento, utilizzo e composizione è disponibile in appendice.

[19] In appendice è disponibile una spiegazione di casa sia un evento glk ed un esauriente elenco degli stessi.

L'argomento `abortres` è usato solo se si vuole cancellare l'input del giocatore e forzare un particolare risultato; si veda sotto.

Se restituite 2 dalla `HandleGlkEvent()`, l'input del giocatore sarà immediatamente abortito[20]. Ma a seconda dell'evento è richiesto un po' di codice in più:

* Se l'evento è un input carattere (`context==1`), dovete richiamare la funzione Glk `cancel_char_event`, e quindi impostare `abortres-->0` al carattere che volete sia restituito e quindi restituite 2; `KeyCharPrimitive()` terminerà il proprio compito e restituirà il carattere, come se il giocatore lo avesse premuto.

* Se l'evento è un (line input) input di linea (`context == 0`), dovete richiamare la funzione Glk `cancel_line_event`. (Potete passare un argomento all'array per controllare cosa il giocatore ha premuto.) Quindi, inserire la lunghezza dell'input perchè restituisca `abortres-->0`. Se è diversa da zero, scrivete i caratteri di input in sequenza nell'array cominciando da `abortres->WORDSIZE`, sino a (ma non includendolo) `abortres->(WORDSIZE+len)`. Non superate i 256 caratteri. Quindi restituite 2; `KeyboardPrimitive()` terminerà e restituirà la linea.

Se restituite -1 dalla funzione `HandleGlkEvent ()`, l'input del giocatore continuerà anche dopo aver premuto un tasto (per l'input di un carattere) o dopo il tasto invio (per l'input di linea). (Non so se è utile, ma potrebbe esserlo). In questo caso dovete ri-richiedere l'input chiamando la funzione `request_char_input` o `request_line_input`.

Ogni altro valore restituito dalla funzione `HandleGlkEvent ()` (un normale `return`, `rfalse`, o `rtrue`) sicuramente non condizionerà l'input del giocatore.

. . .

`InitGlkWindow(winrock)`

Questa entry point è chiamata dalla libreria quando imposta le finestre standard: la finestra principale (story window), la finestra per la riga di stato (status window), e (se usate riquadri per le citazioni) le finestre delle citazioni (quote box window). Le prime due finestre sono create alla partenza del gioco, prima della funzione `Initialise()`. Il terzo tipo di finestre sono create e distrutte quando è necessario. `InitGlkWindow ()` è chiamata in cinque fasi (phase):

[20] Si consiglia caldamente di leggere l'approfondimento dedicato alla materia presente in appendice.

1 La libreria richiama `InitGlkWindow(0)`. Il che avviene proprio all'inizio dell'esecuzione, anche prima della routine `Initialise()`. Potete impostare qualsiasi situazione che vogliate. (Naturalmente, ricordate che le finestre story e status potrebbero già esistere -- ad esempio, se il giocatore ha semplicemente dato il comando RICOMINCIA). Questo è un buon momento per impostare `gg_statuswin_size` ad un valore diverso da 1. Restituire 0 per procedere con l'impostazione standard delle finestre della libreria, o 1 se avete creato per conto vostro tutte le finestre.

2 La libreria richiama `InitGlkWindow(GG_MAINWIN_ROCK)`, prima di creare la finestra story. Questo è un buon momento per impostare gli stili di testo per la finestra story. Restituite 0 per lasciare che la libreria crei la finestra; restituite 1 se avete creato per conto vostro una finestra e l'avete posta in `gg_mainwin`.

3 La libreria richiama `InitGlkWindow(GG_STATUSWIN_ROCK)`, prima di creare la finestra status. Ancora una volta, restituite 0 per lasciare che la libreria lo faccia per voi; restituite 1 se avete creato una vostra finestra e l'avete posta in `gg_statuswin`.

4 La libreria richiama `InitGlkWindow(1)`. Che è la fine dell'impostazione delle finestre; potete cogliere questa opportunità per aprire altre finestre. (Altrimenti potete farlo nella vostra routine `Initialise()`. Non cambia molto).

5 La libreria richiama `InitGlkWindow(GG_QUOTEWIN_ROCK)`, prima di creare la finestra per le citazioni (quote box). Ciò non accade durante l'inizializzazione del gioco; la finestra per le citazioni viene creata durante il gioco, quando stampate una citazione, e distrutta al turno successivo. Come al solito, restituite 1 per indicare che avete creato una finestra in `gg_quotewin`. (Il numero desiderato di linee per la finestra può essere trovato in `gg_arguments-->0`.)

In ogni modo gestiate l'inizializzazione delle finestre, ricordate che la libreria richiede una `gg_mainwin`. Se non ne create una, e non lasciate che la libreria lo faccia per voi, il gioco verrà terminato. Viceversa, le finestre status e citazioni sono opzionali; la libreria può tranquillamente funzionare senza di esse.

. . .

`IdentifyGlkObject(phase, type, ref, rock)`

Questa entry point è richiamata dalla libreria per farvi sapere quali oggetti Glk esistono. Dovrete inserire questa funzione nel caso in cui decidiate di creare una qualsiasi finestra, fileref (riferimento a file), file stream (flusso), o canale sonoro oltre a quelli standard della libreria. (Ciò si rende necessario dal momento che dopo un comando di caricamento, di reinizio o di undo le vostre variabili globali che contengono gli oggetti Glk potrebbero essere inesatte).

`IdentifyGlkObject()` viene richiamata in tre fasi (phases):

1 La libreria richiama la funzione `IdentifyGlkObject()` con `phase==0`. Qui dovreste impostare a zero tutti i riferimenti ai vostri oggetti Glk.

2 La libreria richiama la funzione `IdentifyGlkObject()` con `phase==1`. Ricorre una volta per ogni finestra, stream, e fileref che la libreria non riconosce. (La libreria gestisce le due finestre standard, e i file e gli stream che hanno a che fare con i comandi di salvataggio, copia e registrazione. Dovrete, quindi, trattare unicamente con gli oggetti che create.) Dovreste impostare qualsiasi riferimento in modo appropriato all'oggetto. Per ogni oggetto: `type` sarà 0, 1, 2 per finestre, flussi e fileref rispettivamente; `ref` sarà il riferimento all'oggetto; e `rock` sarà un numero identificativo, la pietra miliare dell'oggetto, dalla quale potrete riconoscerlo[21].

3 La libreria richiama la funzione `IdentifyGlkObject ()` con `phase==2`. Il che accade una sola volta, dopo tutte le altre funzioni, e che vi da la possibilità di riconoscere gli oggetti che non sono né finestre, né stream né filerefs. Se non create nessun oggetto di questo tipo, potete ignorarla. Dovreste però cogliere l'occasione per aggiornare tutti i vostri oggetti Glk allo stato del gioco, che sia appena cominciato o che sia stato caricato da un precedente salvataggio. (Per esempio, ridisegnare le finestre, o reimpostare la giusta musica di sottofondo.)

[21] Il rock value è un numero intero che deve essere associate ad ogni oggetto glk presente nel codice. Il valore può essere scelto dall'autore che però deve tenere conto che alcuni valori sono utilizzati dalla libreria parsem.h Pertanto sarà bene, per evitare problemi seguire le seguenti semplici regole: valori compresi tra 210 e 299 per le finestre, tra 310 e 399 per i flussi ed infine valori superiori a 410 per i riferimenti ai File.

§6 Informazioni per i traduttori

La libreria è progettata per essere facilmente tradotta, ed al momento esistono versioni in Francese, Tedesco ed alcune altre lingue[22]. Se curi il mantenimento di una di queste traduzioni, le seguenti informazioni potrebbero tornarti utili.

§6.1 English.h

Questo è il "file di definizione del linguaggio", e nella traduzione dovrebbe essere sostituito da un analogo file chiamato `French.h`, `German.h`, etc. Sono stati apportati i seguenti cambiamenti:

• La Classe `CompassDirection` e i suoi oggetti sono stati ampiamente rivisitati.

• La Routine `LanguageVerb()` è stata modificata.

• Le Routines `LanguageVerbIsDebugging()`, `LanguageVerbLikesAdverb()` e `LanguageVerbMayBeName()` sono state aggiunte per isolare i test specifici per la lingua che prima erano in `parserm.h`.

• Le Constanti `YOU__TX` e `COMMA__TX` sono state aggiunte.

• La Routine `LanguageLM()` è stata messa in ordine alfabetico.

• In questa routine, `CommandsOff`, `CommandsOn` e `CommandsRead` sono state aggiunte.

• Sempre in tale routine, `Exit`, `Inv`, `Look`, `Miscellany`, `Places`, `Pronouns` e `Score` sono stati estesi, e `Go` è stato modificato.

• La Constante `LIBRARY_ENGLISH` è stata aggiunta. Suggeriamo che le versioni tradotte definiscano invece `LIBRARY_DUTCH`, `LIBRARY_FRENCH`, `LIBRARY_GERMAN`, `LIBRARY_ITALIAN`, `LIBRARY_SPANISH`, `LIBRARY_SWEDISH` etc, nel caso diventi utile in un qualche momento determinare quale lingua si sta utilizzando.

[22] Le librerie di inform sono state tradotte in italiano in due occasioni, una molto tempo fa da Nardinocchi ed una invece più recentemente ad opera di Riccardi. INFIT la libreria italiana curata da quest'ultimo è senza dubbio, al momento, la scelta migliore per l'autore italiano.

§6.2 Grammar.h

Questo file contiene la grammatica per i verbi Inglesi come TAKE e DROP, e nella traduzione viene sostituito dal file `FrenchG.h`, `GermanG.h`, etc. Sono state apportate le seguenti modifiche:

• Le grammatiche sono state poste in ordine alfabetico.

• Le definizioni dei verbi `'recording'` e `'replay'` non sono più comprese nella condizione DEBUG.

• Le definizioni dei verbi `'showobj'`, `'ask'`, `'exit'`, `'look'` e `'tell'` sono state estese.

• Le definizioni dei verbi `'pry'` `'prise'` etc sono state aggiunte.

• E' stata aggiunta la Constante `LIBRARY_GRAMMAR`.

§7 Gull: una guida multimediale per Inform-Glulx

7.1 Introduzione

Ora che abbiamo terminato di vedere le differenze introdotte nell'ultima versione di inform6 è arrivato il momento di approfondire Glulx e guidare il lettore verso il suo corretto uso in una serie di semplici passi.

Il presente ed i prossimi tre capitoli rappresentano tre sezioni tutte abbastanza indipendenti tra loro. Se non avete intenzione di programmare qualcosa in Inform nel prossimo futuro, ma siete curiosi di sapere che cosa è questo Glk di cui avete sentito parlare, ad esempio, potete semplicemente leggere il presente capitolo. Se già sapete che cosa è Glulx Inform e volete solo sapere come fare a suonarci della musica, potete cominciare con il capitolo 8. E se tutto quello che volete è vedere alcune delle cose che Glulx Inform può fare, potete andare direttamente ai giochi di esempio presentati nel capitolo 10. Sta a voi...

Comunque all'interno di ogni capitolo è una buona idea cominciare a leggere dall'inizio, infatti alcuni degli ultimi paragrafi di ogni capitolo danno per scontato che abbiate familiarità con gli argomenti trattati precedentemente, così se saltate direttamente alla parte che parla degli input del mouse, potreste restare confusi dalla segnalazione della routine `HandleGlkEvent()`, che viene spiegata alcune sezioni prima, nella parte che riguarda la gestione delle finestre.

Per questo motivo i prossimi tre capitoli possono essere visti principalmente come un tutorial. È anche una collezione di codice che potete usare ed adattare: sentitevi liberi di copiare ed incollare il codice `IdentifyGlkObject()` e modificatelo, invece di cercare di impararlo a memoria e cercare di ricostruirlo riga per riga[23].

Gull, comunque, non è certamente un manuale di riferimento. Sebbene la ragion d'esistere principale di Gull sia quella di fornire una fonte di informazioni più amichevole rispetto alle specifiche di Glk (Glk spec), certamente non sostituisce quel documento. Gull non cerca in nessun modo di essere completo od autorevole, od anche ben organizzato. Alla fine farete riferimento più spesso alle

[23] Se proprio volete capire cosa accade in ogni funzione qui citata, in appendice sono presenti approfondimenti su di ognuna di esse.

Specifiche Glk, le specifiche di Glulx o ancora alla guida per l'autore di Glulx Inform piuttosto che a Gull[24].

```
http://www.eblong.com/zarf/glk/glk-spec-toc.html
http://eblong.com/zarf/glulx/glulx-spec.html
http://eblong.com/zarf/glulx/inform-guide.txt
```

Ma questi documenti non sono molto comprensibili al programmatore dilettante che non abbia almeno delle nozioni di base sul materiale che trattano. Lo scopo di Gull è quello di fornire queste nozioni.

7.2 Riconoscimenti

Gull l'ho scritto io, Adam Cadre. In generale si tratta di una nuova esposizione di quello che ho imparato riguardo questa materia da altre persone. Un sacco di persone mi hanno spiegato delle cose per la sezione uno, tra cui John Cater, David Glasser, Stephen Granade, Iain Merrick, Dan Shiovitz, e senza dubbio altre persone che mi sono dimenticato di indicare. Il materiale della sezione due, d'altra parte, l'ho imparato quasi completamente da Andrew Plotkin, sia direttamente (tramite domande dirette su un Wiki Web configurato da David Glasser) che indirettamente (con la lettura delle Glk spec). Ulteriori commenti sono stati forniti durante la fase di compilazione da Ross Raszewski e Evin Robertson.

L'argomento di Gull è Glulx Inform. Glulx è stato creato da Andrew Plotkin; Inform da Graham Nelson. Questo è indicato da altre parti in Gull, ma qui sembra un buon posto per dare il giusto risalto alla cosa[25].

[24] Si ricorda che Gull è una serie di articoli che fino ad ora è sempre stata diffusa separatamente dagli altri contenuti. In questo volume dovreste trovare tutto ciò di cui avete bisogno per sviluppare un gioco in inform glulx. Nel presente volume non sono state incluse solo le specifiche glk. Esse per la loro funzione vanno al di la delle necessità (e spesso della comprensione) del semplice autore di avventure testuali. Difficilmente dovrete ricorrere ad esse. NdT.

[25] Per questo volume il documento è stata completamente rivisto ed aggiornato. NdT.

§8 Un'introduzione a Glulx Inform

8.1 Cosa è l'interactive fiction?

Interactive fiction, o IF, è un modo di raccontare delle storie in cui il pubblico è un partecipante attivo. Un brano di interactive fiction (al quale ci si riferisce come a un "gioco", sebbene ci sia poco di giocoso) generalmente comincia presentando al giocatore un breve scenario ("È uno stretto passaggio senza uscita, con i muri che si ergono oppressivamente alti in tre direzioni. Una liscia porta di metallo ti è di fronte ad est, vicino la fine del corridoio. È decisamente chiusa.") A questo punto il giocatore decide che cosa vuole che il proprio personaggio del gioco faccia -- potrebbe lasciare il corridoio, o bussare alla porta. Vede quello che accade, indica cosa fare, e così via, fino a che la storia non arriva ad una fine (spesso coinvolgendo il personaggio in una morte raccapricciante). È divertente!

Ora, se l'autore stesse lì, come il master in un gioco di D&D, il giocatore potrebbe dirgli che cosa vuole fare, e lui potrebbe pensarci per un momento, lanciando dei dadi nel frattempo, e dicendogli quali sono i risultati. Ma se le centinaia e centinaia di persone che regolarmente visitano l'IF archive avessero bisogno di avere gli autori presso le loro case per poter giocare, ci sono serie possibilità che debbano attendere un po'. Fortunatamente il media non dipende da un contatto faccia-a-faccia. Invece l'autore codifica tutta la faccenda -- tutte le descrizioni delle stanze, ogni possibile risposta ai comandi del giocatore -- in un file per computer. I giocatori copiano il file sul proprio computer, leggono il testo che scorre sui loro schermi, attendono la comparsa di un **prompt** (di solito un segno di maggiore: ">"), inseriscono i loro comandi, leggono le risposte, attendono un altro prompt, inseriscono un altro comando, e via di seguito. Le sezioni che seguono illustrano in dettaglio il processo mediante il quale un gioco di IF passa dalla testa di un autore allo schermo del computer di un giocatore.

8.2 Dall'idea al codice sorgente

Dopo aver giocato a qualche IF, molte persone decidono che vogliono provare a scrivere qualcosa di proprio. Il primo passo è quello di tradurre le vostre idee in un **codice sorgente**. Il codice sorgente è un insieme di istruzioni, scritte in un linguaggio per computer, che definiscono cose come quali oggetti ci sono in un gioco, che cosa fanno, come si comportano quando un giocatore cerca di manipolarli, e così via. Ci sono molti linguaggi per computer progettati specificatamente per scrivere IF, come TADS, Hugo e Alan. Ma il linguaggio per IF più usato in questo momento è Inform.

Inform è un linguaggio abbastanza ad alto livello, il che significa che è più vicino al modo in cui gli esseri umani processano le informazioni ("Se il personaggio del giocatore cerca di prendere il cracker al pappagallo, il pappagallo può morderlo") che a quello dei computer ("001001101110010"). Un frammento di codice Inform che agisca in questo modo potrebbe apparire come quello che segue:

```
Object pappagallo "Polly il pappagallo"
     with name 'polly' 'pappagallo' 'uccello',
        before [;
           LetGo: if (noun == cracker)
                     "Polly morde la tua mano mentre cerchi di
                     sgraffignargli il cracker! ~Awwwk! Giù le mani!~
                     dichiara.";
        ],
     has animate proper;
```

Anche se non conoscente Inform, potete dare un'occhiata a questo codice e farvi un'idea abbastanza precisa di quello che fa. È certamente più facile rendere le idee in questa forma che direttamente attraverso una stringa di numeri.

Alcune parti di un determinato gioco in Inform possono essere proprie di tale gioco -- non in tutti i giochi ci sono dei pappagalli, ad esempio. Ma una gran parte può ripetersi da gioco a gioco. Sicuramente la maggior parte dei giochi di IF ha bisogno di un **parser**, la parte del gioco che prende i comandi complessi come "METTI TUTTO TRANNE IL CRACKER NELLA GABBIA DEL PAPPAGALLO" e li scompone in una lista di cose da fare ("esegui il comando Inserisci sul pappagallo, il mangime ed il giornale, ma non il cracker, con la gabbia come destinazione"). Scrivere un parser dall'inizio può essere estremamente difficile -- la maggior parte dei parser "fatti in casa" (ed una manciata di questi viene fuori ad ogni IF competition) sono terribili. Inoltre è una perdita di tempo: se una persona ne ha codificato uno veramente buono, e

volete che il vostro parser faccia quasi le stesse cose, perché non potete semplicemente usare il codice di quella persona? Risposta: potete. Il parser, il sistema di gestione degli oggetti e quasi ogni elemento di interactive fiction che si ripete da gioco a gioco è presente nella **libreria** di Inform.

Una libreria è un gruppo di file, scritto nello stesso linguaggio con il quale state programmando, che potete includere nel vostro programma per eseguire certe funzioni senza aver bisogno di ricodificarle. In Inform questa cosa è facile come scrivere:

```
Include "Parser";
Include "VerbLib";
Include "Grammar";
```

Codificate gli oggetti e le routine proprie del vostro gioco, aggiungete quelle tre linee al posto giusto ed il vostro programma è completo. D'altra parte nessun computer può eseguire direttamente il codice sorgente: il codice sorgente ha bisogno di essere tradotto in codice macchina.

8.3 Dal codice sorgente al codice della VM

Al cuore di ogni computer c'è la **central processing unit**, o CPU. Il lavoro di una CPU è quello di eseguire le istruzioni che le vengono fornite in **codice macchina**, che è semplicemente una stringa di cifre binarie: "1101011010010001" potrebbe voler dire "vai alla 106esima locazione di memoria di questo computer ed aggiungi 1 a qualunque sia il numero che ci trovi". Come fa la CPU a conoscere il significato di questa stringa di numeri? Il motivo è che i chip di silicio sono costruiti in modo che l'invio di una serie di impulsi elettrici che rappresentano quel numero all'interno del chip portano al risultato voluto. E come fa il programmatore a sapere che cosa significa quella stringa di numeri? A meno che si stia lavorando con un linguaggio molto a basso livello, non è possibile saperlo. È per questo che ci sono i **compilatori**[26].

[26] Agli ideologi del "torniamo alle origini" piace sparare a zero contro la "Nuova Matematica" dai tempi dell'educazione hippy. L'enfasi della Nuova Matematica sulle basi alternative è uno dei bersagli preferiti: la Nuova Matematica ritiene importante insegnare ai ragazzi che la base 10 è solo una convenzione arbitraria e che i numeri possono essere espressi anche come posizioni che rappresentano le potenze di altri numeri, come 2. Questa è una inutile perdita di tempo, come dichiarano i critici? Beh, No. Non solo l'insegnamento di basi alternative porta ad una più profonda conoscenza dell'aritmetica in generale, ma è anche utile quando si ha a che fare con i computer. Per i computer la matematica non è decimale, ma binaria.

La matematica in base 10 usa dieci cifre differenti, da 0 a 9. Ma i moderni computer non fanno corrispondere dieci differenti livelli di voltaggio alle varie cifre: c'è un alto voltaggio che scorre in un certo punto, od un basso voltaggio. Due cifre tra cui scegliere. 1 o 0. Questo rende la base 2 l'ordine del giorno. Così, nel caso in cui non avete mai incontrato o vi siete dimenticati della matematica con basi alternative, qui c'è una rapida veduta d'insieme.

Prima di tutto gli esponenti (o potenze). La frase "alla potenza di" significa "moltiplicare insieme il numero per il numero di volte che segue". Così "tre alla potenza di quattro" (o "tre alla quarta potenza") significa prendere quattro 3 e moltiplicarli tra di loro: 3 per 3 per 3 per 3. Un numero alla potenza di 1 è se stesso. Un numero alla potenza di 0 è sempre 1.

Ogni numero ha un certo numero di posizioni. Ogni posizione rappresenta un numero che viene aggiunto agli altri numeri rappresentati dalle altre posizioni, ed il totale è la quantità rappresentata dall'intero numero. Il numero rappresentato da ogni data posizione è la base che viene usata per la potenza di quante posizioni la posizione è distante dalla posizione più a destra, per la cifra indicata nella posizione stessa. Questa cosa è molto contorta da illustrare in astratto, ma è molto semplice nella pratica. Ecco alcuni esempi

Cominciamo con la base 10. Prendiamo ad esempio il numero 1906

- La prima posizione è distante 3 posizioni dalla posizione più a destra. 10 alla potenza di 3 è 10 per 10 per 10, che è 1000. La cifra in questa posizione è 1. 1000 per 1 è 1000. Il numero rappresentato da questa colonna è 1000.
- La seconda posizione è distante 2 posizioni dalla posizione più a destra. 10 alla potenza di 2 è 10 per 10, che è 100. La cifra in questa posizione è 9. 100 per 9 è 900. Il numero rappresentato da questa posizione è 900.
- La terza posizione è distante 1 posizione dalla posizione più a destra. 10 alla potenza di 1 è 10. La cifra indicata in questa posizione è 0. 10 per 0 è 0. Il numero rappresentato da questa posizione è 0.
- La quarta posizione è distante 0 posizioni dalla posizione più a destra. 10 alla potenza di 0 è 1. La cifra in questa posizione è 6. 1 per 6 è 6. Il numero rappresentato da questa posizione è 6.

Ora sommiamo insieme le posizioni. 1000 + 900 + 0 + 6 = 1906. Che è quello con il quale abbiamo cominciato, ed è quello che ci si aspetta quando si converte un numero decimale (base 10) in un decimale. Ora che abbiamo visto un esempio facile, cambiamo base e convertiamo da base 2, quella che usa un computer, in base 10, quella in cui la maggior parte degli umani pensa.

Prendiamo un numero binario (cioè, un numero in base 2). Scegliamo 10101.

- La prima posizione è distante 4 posizioni dalla posizione più a destra. 2 alla potenza di 4 è 2 per 2 per 2 per 2, che è 16. La cifra in questa posizione è 1 (le cifre binarie sono chiamate "bit" (da binary digit, cifra binaria), ed è così che le chiameremo da adesso in poi). 16 per 1 è 16. Il numero rappresentato da questa posizione è il decimale 16.
- La posizione successiva è distante 3 posizioni dalla posizione più a destra. 2 alla potenza di 3 è 2 per 2 per 2, che è 8. Il bit in questa posizione è 0. 8 per 0 è 0. Il numero rappresentato in questa posizione è il decimale 0.
- La posizione successiva è distante 2 posizioni dalla posizione più a destra. 2 alla potenza di 2 è 2 per 2, che è 4. Il bit in questa posizione è 1. 4 per 1 è 4. Il numero rappresentato da questa posizione è il decimale 4.
- La posizione successiva è distante 1 posizione dalla posizione più a destra. 2 alla potenza di 1 è 2. Il bit in questa posizione è 0. 2 per 0 è 0. Il numero rappresentato da questa posizione è 0.
- L'ultima posizione è distante 0 posizioni dalla posizione più a destra, è la posizione più a destra. 2 alla potenza di 0 è 1. Il bit in questa posizione è 1. 1 per 1 è 1. Il numero rappresentato in questa posizione è il decimale 1.

Sommiamo insieme le posizioni. 16 + 0 + 4 + 0 + 1 = 21 decimale. 10101 binario = 21 decimale.

I numeri binari diventano molto lunghi molto in fretta. Prima abbiamo usato il decimale 1906 come esempio; in binario questo è 11101110010. Così molti programmatori scelgono di usare una base più efficiente. Base 10? No, è molto più facile usare una potenza di 2, e questo è il motivo per il quale l'esadecimale, base 16, è la soluzione consueta. L'esadecimale (spesso chiamato "hex") usa le cifre da 0 a 9 come in decimale, e poi usa le lettere da A ad F a rappresentare i decimali da 10 a 15. È conveniente per i programmatori perché i numeri binari possono essere convertiti in numeri esadecimali quattro bit alla volta. Ad esempio prendiamo un numero binario lungo, come 1001110100110101. Per convertirlo in esadecimale lo suddividiamo in unità di quattro bit:

1001 1101 0011 0101

e convertiamo ognuna di queste unità in una cifra esadecimale. 1001 = 9, 1101 = D (13 decimale), 0011 = 3, 0101 = 5 Così la versione esadecimale di quel numero binario è 9D35. Ora, per verificare che sia veramente così facile, assicuriamoci che corrispondano alla stessa cosa in decimale (comprimendo un po' il processo, questa volta):

32768 per 1 è 32768 ;	16384 per 0 è 0;	8192 per 0 è 0;	4096 per 1 è 4096;
2048 per 1 è 2048 ;	1024 per 1 è 1024;	512 per 0 è 0;	256 per 1 è 256;
128 per 0 è 0;	64 per 0 è 0;	32 per 1 è 32;	16 per 1 è 16;
8 per 0 è 0;	4 per 1 è 4;	2 per 0 è 0;	1 per 1 è 1;

32768 + 0 + 0 + 4096 + 2048 + 1024 + 0 + 256 + 0 + 0 + 32 + 16 + 0 + 4 + 0 + 1 = 40245 decimale. Ora l'esadecimale.

- Prima posizione: 16 alla 3 è 4096. La cifra in questa posizione è 9. 4096 per 9 è 36864.
- Seconda posizione: 16 alla 2 è 256. La cifra in questa posizione è D (13 decimale). 256 per 13 è 3328.
- Terza posizione: 16 alla 1 è 16. La cifra in questa posizione è 3. 16 per 3 + 48.
- Quarta posizione: 16 alla 0 è 1. La cifra in questa posizione è 5. 1 per 5 è 5.

36864 + 3328 + 48 + 5 = 40245. Ta-da! Ha funzionato. Perché funziona? Vediamola in questo modo:

- Il significato delle posizioni in un numero decimale a cinque cifre: 10000x, 1000x, 100x, 10x, 1x (cioè, quanti 10000 ci sono più quanti 1000 ci sono più ecc., x non è una variabile, è un moltiplicatore.)
- Il significato delle posizioni in un numero binario a sedici cifre: 32768x, 16384x, 8192x, 4096x, 2048x, 1024x, 512x, 256x, 128x, 64x, 32x, 16x, 8x, 4x, 2x, 1x
- Il significato delle posizioni in un numero esadecimale a quattro cifre: 4096x, 256x, 16x, 1x

Ora, prendiamo una cifra esadecimale e convertiamola nel binario equivalente: 1x hex = (8x, 4x, 2x, 1x) binario. Adesso moltiplichiamolo per ognuna delle posizioni esadecimali:

- 4096x per (8x, 4x, 2x, 1x) = 32768x, 16384x, 8192x, 4096x
- 256x per (8x, 4x, 2x, 1x) = 2048x, 1024x, 512x, 256x
- 16x per (8x, 4x, 2x, 1x) = 128x, 64x, 32x, 16x
- 1x per (8x, 4x, 2x, 1x) = 8x, 4x, 2x, 1x

Come si può vedere, sono esattamente le stesse delle posizioni binarie. L'unica posizione in comune tra i numeri binari/esadecimali e quelli decimali è la prima, quindi quella decimale non è una base molto conveniente da usare con i computer.

Un compilatore è un programma che prende il codice sorgente di un linguaggio ad alto livello, abbastanza comprensibile come Inform, e lo converte in codice macchina. Comunque è qui che cominciano i problemi. Ogni tipo di macchina è differente. Una stringa di numeri che significa "somma tra di loro questi due numeri" sulla CPU di una macchina Intel, può significare qualcosa di diverso -- o non significare nulla -- su una macchina Apple. Una soluzione a questo problema potrebbe essere quella di prendere il vostro codice sorgente, trovare una macchina su cui gira Windows, e compilare un **file eseguibile** -- cioè , un programma che può girare per proprio conto, come ogni altro programma che usate -- per Windows. Poi cercare un Mac e compilare un eseguibile per Mac. Poi prendete un Palm Pilot e compilate una versione per il Palm. E una per l'Acorn, una per l'Amiga, e poi una nuova ogni volta che esce un nuovo tipo di macchina... è facile vedere come questa non sia la soluzione migliore se volete che persone differenti con tipi differenti di macchine siano in grado di usare il vostro gioco.

Invece la soluzione adottata dalla comunità IF è quella di compilare i propri programmi per delle **macchine virtuali** (virtual machine). Invece che compilare per il Mac o per DOS, gli utenti di TADS compilano i file per la "VM TADS", quelli Hugo per la "VM Hugo", e gli utenti Inform compilano per la "Z-machine". I documenti con le specifiche per la Z-machine sono come i documenti con le specifiche per ogni altra macchina: dicono agli scrittori di compilatori come appare il codice macchina interpretabile dalla Z-machine. L'unica differenza è che nessuna Z-machine è stata realmente costruita: nessuna fabbrica connette Z-chip-di-silicio a Z-circuiti-stampati, e non vedrete Z-laptop in vendita da CompUSA. Ma potete ancora compilare programmi per queste.

Così avete avuto un'idea per un gioco, avete imparato Inform, avete scritto il vostro codice sorgente e lo avete compilato. Quello che avete adesso è il file di un gioco progettato per essere eseguito su una macchina che non esiste. Cosa c'è di buono in tutto questo? Continuate a leggere.

8.4 Dal codice della VM al vostro schermo

Dovreste avere familiarità con il concetto di **emulatore**. Un emulatore è un software che vi consente di eseguire programmi compilati per una piattaforma differente sul vostro computer, traducendo il codice macchina di tale piattaforma nel codice macchina nativo del vostro computer. Molti utenti Mac possiedono un emulatore chiamato Virtual PC che consente loro di eseguire Microsoft Windows, ad esempio. L'IF fa un uso molto spinto di questo concetto: ogni macchina virtuale ha un certo numero di **interpreti** associati, che sono programmi che consentono ai giochi compilati per quella VM di essere giocati su un particolare tipo di computer. MaxTADS, ad esempio, è un

interprete che prende i giochi per la VM TADS e li esegue sul Macintosh. WinFrotz è un interprete che prende i giochi per la Z-machine e li esegue sulle macchine Windows. La Z-machine, in particolare, ha interpreti per uno spaventoso numero di piattaforme, compreso il GameBoy (!). Così, sebbene alcuni giochi vengano rilasciati come file eseguibili (usando un programma come JZEXE, che unisce il file del gioco ad un interprete specifico per la piattaforma considerata), il modo più comune di giocare con un'IF è quello di scaricare separatamente il file del gioco che avete intenzione di usare ed un interprete che può girare sulla vostra macchina, eseguire l'interprete, e cominciare a giocare. Ed è così che i giochi passano dalla testa dell'autore al vostro schermo.

Potreste chiedervi se il passo intermedio dello specificare una virtual machine non richieda più lavoro. Vero, significa che non dovete compilare il vostro gioco per dozzine di piattaforme differenti -- ma *dovete* compilare degli interpreti per tutte quelle piattaforme differenti, come può questa cosa essere più efficiente? La risposta è: non può, se un solo gioco verrà scritto per la VM in questione. Ma non appena scrivete un secondo gioco, diventa ovvio il perché valga la pena di usare una VM: compilate il file del gioco per quella VM, ed il gioco può essere usato su tutti gli interpreti che avete già creato. Così se scrivete un gioco in Inform non avete bisogno di preoccuparvi di compilarlo per Windows e per Macintosh e per Acorn e per GameBoy -- lo compilate semplicemente per la Z-machine, e le persone che usano Windows e Macintosh e Acorn e GameBoy possono giocarci con degli interpreti scritti anni fa.

E fino a poco tempo fa la cosa sarebbe finita qui. Ma di recente è stato introdotto un nuovo passo salva-fatica, che va sotto il (molto discusso) nome di Glk.

8.5 Glk

C'è una grande quantità di linguaggi per IF, ognuno con la propria virtual machine, e molti altri se ne stanno scrivendo. Ci sono anche un sacco di piattaforme differenti là fuori, e nuovi computer e sistemi operativi stanno vedendo la luce. Questo ha portato ad un problema che dovrebbe suonarvi familiare se avete letto le sezioni precedenti. Ogni volta che arriva un nuovo sistema per la programmazione di IF, le persone hanno la necessità di scrivere un interprete per ogni piattaforma su cui vogliono essere in grado di far girare i giochi del vostro sistema: Windows, DOS, Mac, Acorn, Amiga, BeOS, Unix, e moltissime altre. Ed ogni volta che una piattaforma su cui sia possibile giocare con l'IF arriva -- ad esempio delle persone che voglio giocare su un MUD, o su

un telefono cellulare abilitato ad Internet -- le nuove piattaforme hanno bisogno di un interprete Z-code ed un interprete TADS ed un interprete Hugo ed un interprete Alan e così via.

Inoltre molto del lavoro che riguarda la scrittura di un interprete è fatica sprecata. Vedete, un interprete deve fare due cose più o meno indipendenti: il trattamento dei dati -- interpretazione dei comandi (parsing), tenere traccia degli oggetti nel gioco, e cose così -- e l'input/output (abbreviato in I/O per semplicità), che ha a che fare con l'esattezza con cui le cose il programma dice di stampare vengono visualizzate. E con qualche limitata eccezione (di cui ci occuperemo tra breve), la parte di trattamento dati non si preoccupa di ciò che fa la parte di I/O. Dice alla porzione di I/O l'equivalente in codice macchina di "Scrivi queste tre frasi", ma alla parte di trattamento dati non interessa se queste frasi appaiono in testo piano su una shell Unix o se appaiono con un bel font in una finestra grafica con delle barre di scorrimento. La stessa cosa per la parte di I/O, che si preoccupa solo di mostrare le cose che gli è stato detto di mostrare. Non si preoccupa di tutti gli algoritmi che la parte di trattamento dati ha usato per arrivare alla decisione di cosa stampare. Si preoccupa solamente di fare la visualizzazione. Questo significa che mentre la porzione di trattamento dati di tutti gli interpreti Z-machine è vicina ad essere identica, la porzione di I/O differisce drasticamente: DOS Frotz, progettato per essere usato in un ambiente senza supporto per font differenti, richiede delle routine di stampa del testo molto differenti da quelle usate in WinFrotz. Allo stesso modo, mentre la parte di trattamento dei dati dei vari interpreti Mac può essere piuttosto diversa -- TADS e Inform e AGT ed altri semplicemente non macinano i numeri allo stesso modo -- la parte dell'interprete che ha a che fare con il look and feel, l'I/O, può essere più o meno la stessa indipendentemente dalla VM che si sta usando. Ma visto che ogni potenziale coppia deve essere messa insieme, il lavoro finisce per essere rifatto tutte le volte.

Glk è il tentativo di rendere tutto questo processo più efficiente separando le due funzioni degli interpreti IF tradizionali. Lo fa definendo una serie di routine, scritte nel linguaggio per computer C, e raccogliendole nella libreria Glk. I programmatori che hanno familiarità con il C possono scrivere delle **applicazioni Glk** (Glk applications), che fanno il trattamento dei dati e restituiscono i risultati in un formato che Glk può capire -- GlkInform, GlkTADS, e così via -- e delle **librerie Glk** (Glk libraries, chiamate anche "implementazioni Glk", Glk implementations), che adattano le funzioni Glk ad una specifica piattaforma: WinGlk, XGlk, MacGlk, ecc. Lo scrittore di interpreti non deve far altro che unire tra di loro applicazioni e librerie.

Se non è chiaro come questa cosa sia utile, immaginiamola in questo modo. Diciamo che invece di cercare di adattare differenti linguaggi per IF a differenti piattaforme, stiamo cercando di costruire un sistema ferroviario che trasporti le persone da alcune città della costa ovest degli Stati Uniti -- Seattle, Portland, San Francisco, Los Angeles, ecc. -- alle città della costa est: Boston, New York, Philadelphia, Washington, ed altre. Il sistema pre-Glk di collegamenti diretti

avrebbe comportato la costruzione di rotaie da Seattle a Boston, da Seattle a New York, da Seattle a Philadelphia, ecc., poi da Portland a Boston, da Portland a New York, ecc. Se fosse venuto fuori un nuovo punto di partenza, avremmo dovuto costruire un intero nuovo insieme di rotaie: da San Diego a Boston, da San Diego a New York, ecc. E lo stesso sarebbe accaduto per ogni nuova destinazione: avremmo dovuto costruire da Seattle a Miami, da Portland a Miami, da San Francisco a Miami, e così via. Glk stabilisce un centro, un concentratore. Nella nostra analogia diciamo che sia Omaha nel Nebraska. Così, invece di costruire rotaie direttamente dai punti di partenza alle destinazioni, noi costruiamo da Seattle a Omaha, da Portland a Omaha, da San Francisco a Omaha, ecc. E poi costruiamo da Omaha a Boston, da Omaha a New York, ecc. Questo riduce drasticamente il numero di rotaie che dobbiamo costruire. E se viene fuori un nuovo punto di partenza, dobbiamo costruire solo una rotaia: da San Diego a Omaha. Una nuova destinazione? Una rotaia: da Omaha a Miami. E le persone che arrivano da San Diego possono cambiare treno a Omaha per andare ad ogni altra destinazione, e le persone che arrivano da qualunque parte della costa ovest possono cambiare treno a Omaha ed andare a Miami.

Così diciamo che qualcuno scrive un nuovo linguaggio per IF chiamato ABC. Non ha bisogno di scrivere degli interpreti ABC-per-Windows, ABC-per-Mac, ABC-per-Palm; invece scrive semplicemente un interprete ABC-per-Glk, e le persone di tutte le piattaforme con le librerie Glk possono giocare ai giochi ABC. Allo stesso modo se la Toastent Technologies rilascia il Cybertoaster, tutto quello che deve essere scritto è una libreria Glk-per-Cybertoaster, ed il Cybertoaster è istantaneamente pronto per usare i giochi Inform, TADS, Hugo, o giochi che usino qualunque altro sistema per il quale ci sia una applicazione Glk -- tutto quello di cui hanno bisogno gli utenti Cybertoaster è qualcuno che metta insieme le librerie e le applicazioni per creare degli interpreti.

Potreste chiedervi: le applicazioni Glk ricevono dalle virtual machine e riformattano il loro output così che Glk possa capirlo. Esistono virtual machine il cui output sia già adatto all'uso con Glk? La risposta: ci potete scommettere. Questo è Glulx.

8.6 Glulx e Glulx Inform

Ci sono, come discusso nelle precedenti sezioni, molte virtual machine là fuori progettate per l'IF. Alcune di queste -- la VM TADS e la VM Hugo, ad esempio -- che possono fare cose che la Z-machine non può fare. Possono gestire giochi più grandi di 512 kilobyte, ad esempio, far mostrare loro delle immagini od emettere suoni è molto meno scomodo che cercare di fare lo stesso con la Z-machine. Comunque, per usare una di quelle VM, avete bisogno di imparare il

corrispondente linguaggio: TADS o Hugo. Molti programmatori Inform hanno investito una significante quantità di tempo ed energia per imparare Inform, e preferirebbero non dover ricominciare da capo per imparare come si programmano IF.

Glulx è stato progettato per risolvere questo problema. Glulx è una virtual machine a 32 bit i cui compilatori sono stati disegnati per prendere il codice Inform e convertirlo in codice Glulx. In questo modo i programmatori Inform possono lasciarsi alle spalle le restrizioni della Z-machine ed approfittare dei vantaggi delle capacità extra di Glulx senza avere la necessità di imparare un nuovo linguaggio. O quasi[27].

---

[27] Cosa è una VM a 32 bit? Come indicato nella nota su binario ed esadecimale, un **bit** è una cifra binaria: 0 o 1. Un numero a 32 bit sarebbe quindi un numero con 32 cifre binarie, da qualche parte tra 0 e 4.294.967.295 decimale. E delle virtual machine si è già parlato. Ma sapere cosa semplicemente "32 bit" e "virtual machine" significano non necessariamente getta molta luce su cosa possa essere una "virtual machine a 32 bit". Con un po' di fortuna, questa nota lo farà.

Ogni computer ha una certa quantità di **memoria**. La memoria è dove il computer conserva le informazioni. Fisicamente è fatta di chip di silicio, con ogni chip che contiene molti minuscoli pezzi di macchinario che possono variare tra due configurazioni, una che rappresenta lo 0, l'altra che rappresenta l'1: cioè un bit. La maggior parte dei computer ha dei chip di memoria che raggruppano questi bit in insiemi di otto; ogni gruppo di otto bit viene chiamato **byte**, e può contenere un valore da 00000000 a 11111111, o da 0 a 255 in decimale. Oltre al proprio valore, ogni bit ha un **indirizzo**. Così quando la CPU riceve un'istruzione che dice "memorizza il numero 73 nella locazione 12", comincia dall'inizio della propria memoria e conta, "Questa è la locazione 0, questa è la locazione 1, questa è la locazione 2..." e ad un certo punto "Ah ah, questa è la locazione 12". Quindi imposta i bit di quella locazione a 01001001 (73 decimale).

Quando eseguite una virtual machine sul vostro computer, la VM chiede al computer di allocare della memoria che poi userà per proprio conto. La VM chiama ogni locazione in memoria con il suo proprio nome. Il nome della VM ed il vero nome nel computer possono non corrispondere; l'interprete si occupa della traduzione. Così il comando della VM "memorizza il numero 73 nella locazione virtuale 12" viene tradotto nel comando del sistema operativo "memorizza il numero 73 nella locazione di memoria reale 12345" (o qualunque sia l'equivalente reale della locazione virtuale 12) e la virtual machine non ha bisogno di sapere quale sia il reale hardware che si sta usando -- può semplicemente usare i propri nomi e lasciare che il SO si occupi dei dettagli.

Le istruzioni della VM sono composte di **opcode** (la cosa da fare) e **operandi** (la cosa o le cose a cui farla). Potreste avere una VM con un opcode "@add", ad esempio, che richiede due operandi: una locazione virtuale ed un numero da sommare al numero in quella locazione. Così "@add(10101001,00010111)" significherebbe "vai alla locazione 10101001 (169 decimale, A9 esadecimale) ed aggiungi 00010111 (23 decimale, 17 esadecimale) a qualunque cosa ci sia là dentro". Ora, la Z-machine è una VM a 16 bit, che significa che i suoi operandi sono stati disegnati per accettare operatori che sono lunghi 16 bit. Questo significa che le locazioni non possono avere un indirizzo più grande di 1111111111111111, o 65535 decimale (FFFF hex). Anche se il vostro computer ha moltissime altre locazioni disponibili nella sua memoria, la Z-machine non le può usare perché i suoi operatori non sono in grado di nominarle.

Un'analogia: negli Stati Uniti i numeri di telefono di ogni area hanno il formato XXX-XXXX. Il che significa che ci possono essere, al massimo, dieci milioni di numeri possibili in un'area: da 000-0000 a 999-9999. (In pratica ce ne sono meno, visto che alcuni numeri sono riservati). Ora diciamo che tutti i numeri di una certa area siano stati già assegnati. Potete ancora costruire un edificio in quell'area. Potete ancora comprare un telefono e metterlo in quella casa. Ma non potete

Dovreste ricordare dalla sezione Glk che c'è una importante eccezione alla regola riguardante la separazione tra trattamento dei dati ed input/output nella programmazione IF. L'eccezione capita quando i programmatori inseriscono **opcode** nei loro programmi.

Un opcode è un'istruzione che salta alcuni dei consueti passi di un processo di programmazione. Immaginate questo processo come una torre di robot, ognuno sulle spalle di un altro. Ogni robot parla solo con il robot sotto di lui. Così un robot vicino la cima della torre parla con il robot sotto di lui, "Ecco, stampa questa frase". E quel robot, il robot di I/O, dice al prossimo robot più sotto, "Okay, scrivi la lettera A a questa posizione dello schermo con questo colore, poi stampa la lettera B a *questa* posizione con *questo* colore..." E *quel* robot la traduce in "Metti un punto bianco qui, qui, qui, qui, qui..." Alla fine vedrete dei piccoli puntini bianchi sullo schermo a formare le parole che interpretate come parte di una storia.

Ma cosa accade se non volete lasciare ai robot il compito di determinare dove le lettere devono andare e come devono apparire? Cosa accade se volete levarvi dai piedi uno dei robot e dire direttamente al robot di I/O, "Okay, voglio la lettera Q, in grassetto, alla posizione dello schermo (1,5), e in verde chiaro"? È qui che entrano in gioco gli opcode. Normalmente se il vostro gioco è in Inform, scrivete tutte le vostre istruzioni in Inform, e poi lasciate al compilatore il compito di convertirle in codice macchina. Ma qualche volta, quando volete un controllo più diretto -- come quando state personalizzando la vostra riga di stato, o quando volete inserire effetti speciali nel gioco come delle parole che ballano nello schermo dei titoli -- avete la necessità di inserire degli opcode. Se state compilando per la Z-machine, questi opcode sono scritti in Z-assembly, e sono cose come "@set_colour 2 4;" (che significa "stampa il testo in nero con lo sfondo verde") o "@read_char 1 8 Dummy i;" (che significa "aspetta che venga premuto un tasto o per otto decimi di secondo, chiunque dei due arrivi prima"). Gli opcode Z-assembly vengono trattati nell'Inform designer's manual, e per un sacco di tempo sono stati considerati dai più come un'altra parte di Inform.

D'altra parte l'arrivo di Glulx ha cambiato le cose. I normali compilatori Inform sanno come trasformare il codice sorgente Inform in codice Z-machine e come trasformare gli opcode Z-assembly in codice Z-machine. Ma mentre i

---

chiamare quel telefono, perché ogni numero che avreste potuto assegnarli è già stato preso da un altro telefono. L'unica soluzione è quella di cambiare il formato di numerazione che i numeri di telefono assumono in quel sistema, in modo da aggiungere più cifre.

Questo è quello che le VM a 32 bit come Glulx fanno. Gli opcode di Glulx accettano operandi che sono lunghi 32 bit, in questo modo i giochi che usano Glulx sono in grado di richiamare, nominare ed usare non solo 65536 locazioni di memoria, ma più di quattro miliardi di locazioni, con i nomi che vanno da 00000000 hex a FFFFFFFF hex. Così con Glulx possono essere creati dei giochi che sono molto più grandi di quelli che possono essere creati per la Z-machine.

compilatori Glulx Inform sanno come convertire il codice sorgente Inform in codice Glulx, lo Z-assembly appare loro incomprensibile. I compilatori hanno bisogno che tutti gli opcode siano scritti per Glk. Questo comprende anche quelli inseriti nella libreria di Inform, così che la libreria 6/10 standard di Inform non funziona con Glulx: avete la necessità di usare la libreria Inform 6/11 (che tra le sue caratteristiche ha quella di poter essere usata con i giochi indirizzati alla Z-machine o a Glulx).

Questo significa che ogni conoscenza gli autori Inform hanno riguardo gli opcode per la Z-machine è inutile nella creazione di giochi Glulx. Se vogliono compilare per Glulx, hanno bisogno di imparare un nuovo insieme di procedure per specificare gli stili del testo, creare effetti speciali, visualizzare immagini, suonare musica ed altro. Ed è a questo che serve il prossimo capitolo.

§9 Come fare delle cose con Glulx Inform

9.1 Di cosa avete bisogno per usare Glulx Inform

Per compilare un gioco Glulx Inform avete bisogno di:

- Il codice sorgente da compilare. Lo fate voi stessi, con un editor di testi (*non* con un word processor) consigliamo il JIF di Alessandro Schillaci[28].
- La librerie di Inform 6/11.
- Infglk, un'aggiunta alla libreria, che fornisce una serie di routine che consentono agli autori Glulx Inform di avvantaggiarsi delle capacità multimediali di Glk senza dover conoscere niente riguardo gli opcode @glk o la tabella che associa i numeri alle funzioni. Semplicemente scaricate l'ultima versione del package Infglk[29] dalla sezione Glulx dell'IF archive, estraete i file nella directory in cui state facendo i vostri lavori in Glulx Inform, ed includete la riga che segue nel vostro codice sorgente (subito dopo il punto in cui includete le librerie parser e verblib): `Include "Infglk";` Ogni sezione che segue presuppone che stiate usando Infglk, così consideratela una estensione obbligatoria.
- Il compilatore Inform 6.30 o successivo per la vostra piattaforma.

E per eseguire il programma che avete compilato, avete bisogno di:

- Un interprete Glulx per la vostra piattaforma[30].
- Una libreria Glk per la vostra piattaforma, per fornire all'interprete le risorse di cui ha bisogno per essere eseguito.

Ora, se volete includere suono e grafica nel vostro gioco, avete bisogno di aggiungere un'altra trovata a questa configurazione: **Blorb**. Che cos'è? Continuate a leggere...

---

[28] E' possibile trovarlo all'indirizzo http://www.slade.altervista.org/JIF/.

[29] Vanno bene le Infglk dalla versione 0.61 in poi.

[30] Al momento ne esistono per tutte le principali piattaforme, il più classico è Glulxe, il più recente (al momento) Gargoyle.

9.2 Blorb

Come indicato nelle sezioni precedenti, la comunità IF tende a dare molta importanza al fatto che i giochi e gli strumenti IF siano usabili dalle persone su una grande varietà di piattaforme. Ma questa accessibilità sconta un prezzo da pagare. Anche solo per giocare con l'IF c'è bisogno di scaricare un paio di pezzi differenti, un interprete appropriato per una determinata macchina ed il file del gioco; ora immaginate di mettere delle risorse multimediali nel tutto. Sarebbe abbastanza spiacevole se si trattasse semplicemente di prendere qualche centinaio di immagini condite da qualche file musicale e comprimere il tutto in un file zip insieme al file del gioco, ma la cosa è più complicata. Anche se riuscite a portare tutti i vostri file dal punto A al punto B senza perdere niente lungo la strada, chi vi assicura che il computer al punto B sia in grado di capire? Anche i nomi dei file possono essere un problema quando si cerca di condividere i dati con tipi differenti di piattaforme. Le macchine DOS e Windows hanno bisogno di avere nei nomi di file un'estensione di tre lettere appropriata per renderli usabili; le macchine Acorn (come quelle usate per sviluppare Inform) non *permettono* proprio le estensioni. Essere in grado di compilare un singolo file, che racchiuda il file del gioco e tutte le risorse associate, che sia giocabile dagli interpreti di molte piattaforme differenti, sarebbe un significativo passo avanti. È per questo che è stato creato Blorb.

Per creare un file Blorb sono necessari:

- un file di risorse Blorb (che è in pratica una lista dei file che volete mettere insieme)
- i file di immagini e/o suoni che state usando
- il codice sorgente del vostro gioco e le librerie che gli occorrono
- il compilatore Inform 6.30 per la vostra piattaforma
- un Blorbifier (Blorbificatore) per la vostra piattaforma

Sfortunatamente Blorb è abbastanza nuovo perché gli strumenti non siano completamente standardizzati, e non sono neanche largamente supportati. Fino a quel momento dovrete avere a che fare con degli strumenti specifici per ogni piattaforma. Una buona scelta è Iblorb per DOS[31].

[31] Una scelta alternativa può essere gblorb di Simon Baldwin, rispetto a iblorb è meno sofisticato e più spartano, ma presenta l'indubbio vantaggio di essere multipiattaforma. E' infatti scritto in glulx e pertanto per creare il file blorb basterà redigere il file risorse e poi lanciare gblorb con il proprio interprete per glulx. Gblorb (aggiornato alle specifiche 2 di Blorb) è stato tradotto in italiano ed è disponibile sul sito www.ifitalia.info. La sua versione originale è sull'if archive. Si consiglia di consultare quest'ultimo per controllare se ne esistono versioni più aggiornate.

9.3 Blorb su DOS/Windows

Il Blorbifier (Blorbificatore) per DOS/Windows è un pacchetto chiamato "iblorb" di Ross Raszewski[32].

```
http://www.cs.loyola.edu/%7Elraszews/if/
```

Una volta che avete scaricato l'archivio di iblorb, estraete i file all'interno della directory in cui il vostro codice sorgente ed il compilatore si trovano.

Avrete, inoltre, bisogno di creare un **file delle risorse** (resource file) per Blorb. Questo deve avere lo stesso nome del vostro codice sorgente, solo con l'estensione .res invece che .inf (ad es., se il vostro codice sorgente si chiama "chef.inf", chiamate il file delle risorse di Blorb "chef.res". Createlo con qualunque programma usiate per scrivere il vostro codice sorgente: un text editor di qualche tipo, non un word processor).

Il file delle risorse è semplicemente una lista dei file da includere. Questa lista deve essere compilata secondo un particolare formato, con ogni riga che individua il tipo di file da includere, il nome con il quale chiamerete il file all'interno del codice sorgente, e la posizione del file. Il primo file che vorrete includere sarà il file del vostro gioco, seguito da qualunque immagine o suono vorrete. Un gioco molto semplice potrebbe avere un file delle risorse di Blorb come questo:

```
CODE C:\IF\Glulx\chef.ulx
PICTURE Monkey C:\Graphics\macaque2.jpg
PICTURE Stapler C:\Graphics\Drawings\stapler.png
```

Una volta che il vostro file delle risorse è pronto, dovrete includere una citazione dello stesso nel vostro codice sorgente. Inserire il comando `Include` per il file delle risorse di Blorb subito dopo l'inclusione di Infglk funziona benissimo. D'altra parte quello che realmente viene incluso non è il file delle risorse che avete scritto, ma una versione modificata generata da iblorb. Questo file ha l'estensione `.bli`, così che la vera istruzione `Include` nel vostro codice sarà:

```
Include "chef.bli";
```

[32] La versione .5 di Iblorb è stata tradotta in italiano ed è disponibile su www.ifitalia.info.

L'ultimo passo prima di cominciare: visto che la compilazione e la Blorbificazione hanno luogo nello stesso momento con questo metodo, iblorb ha bisogno di sapere come si chiama il vostro compilatore. Per dirglielo dovete creare un file chiamato **infb.rc**, più o meno allo stesso modo di in cui avete creato il file .res[33]. Deve andare nella stessa directory di iblorb, e deve contenere una riga. La riga è la parola "INFORM" seguita dal nome (senza l'estensione) del compilatore che state usando, più tutti gli switch che volete. Il vostro compilatore potrebbe chiamarsi inform630.exe, ad esempio, e potreste aver bisogno di usare lo switch `-G` per compilare per Glulx invece che per la Z-machine. Così il vostro file infb.rc potrebbe essere[34]:

```
INFORM inform630 -G
```

Infine, aprite una finestra DOS, posizionatevi nella directory dove si trovano il vostro codice sorgente, il vostro compilatore, il vostro file delle risorse Blorb, il vostro file infb.rc, ed i file di iblorb. Il comando che digiterete avrà il seguente formato: la parola "front", uno spazio, il nome del file delle risorse (senza l'estensione .res), un altro spazio, il nome del file del vostro codice sorgente (compresa l'estensione .inf), ancora un altro spazio, ed il nome del file di gioco compilato volete includere che nel file Blorb (questo avrà un'estensione .ulx, che dovete indicare). Sembra complicato? Questo significa che tutto quello che dovete digitare è:

```
front chef chef.inf chef.ulx
```

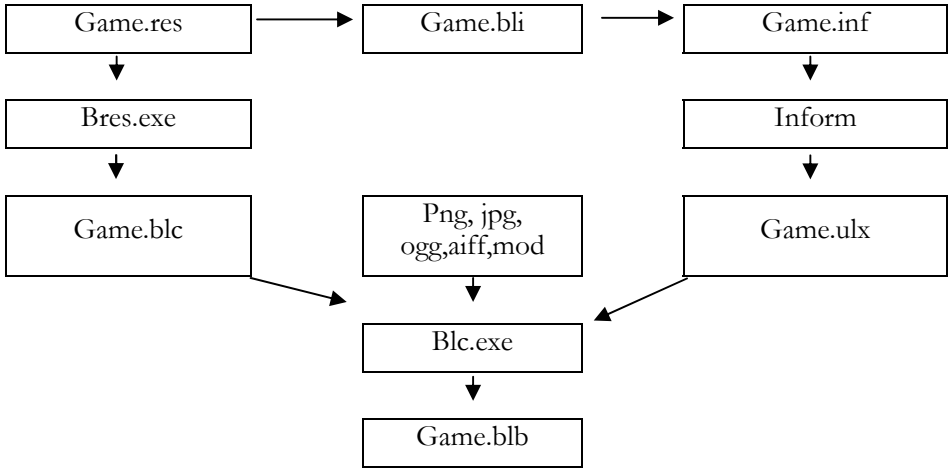
...sostituendo a "chef" qualunque sia il nome comune i vostri file avranno.

Ora, potrebbe essere il caso in cui ci siano degli errori nel vostro programma, e vi verranno indicati proprio come lo sarebbero stati se aveste compilato nel vecchio modo "inform -G chef". Ma se il vostro programma non ha errori di compilazione, quello che avrete alla fine sarà un file con estensione .blb (chef.blb, in questo esempio) che potete eseguire con il vostro interprete Glulx. Caricate semplicemente chef.blb al posto di chef.ulx e godetevi i frutti del vostro lavoro.

---

[33] La creazione del file infb.rc non si rende necessaria se utilizzate l'edito JIF di Alessandro Schillaci che supporta iblorb al suo interno. Per maggiori informazioni si consulti la guida in linea di JIF.

[34] Alla riga chiaramente vanno aggiunti tutti gli switch di cui l'autore ha bisogno, compreso il +language_name=italian per compilare con le librerie italiane.



Lo schema mostra il processo che porta alla creazione del gioco. Si parte dal sorgente del gioco [game.inf] e dal file dove sono elencate le risorse [game.res]. La compilazione del secondo con bres.exe crea due file: [game.bli] che deve essere incluso nel sorgente [game.inf] e [game.blc]. A questo punto si compila il sorgente del gioco e si ottiene il file [game.ulx]. Che può già essere giocato così come è. Per ottenere un file che inglobi tutto usiamo blc.exe che comprimerà [game.blc] le immagini ed i suoni ed il file [game.ulx] in un unico file [game.blb]. Il file che distribuiremo e che sarà giocato dagli utenti.

La nuova versione .5 di Iblorb contiene al suo interno anche altre applicazioni utili all'autore. In particolare sono presenti i programmi Bmerge e Bdiff, essi permettono di unire due o più file blorb assieme o di estrarre le differenze tra due versioni successive del medesimo gioco "blorbato". Dal momento che i giochi distribuiti nel formato possono essere piuttosto pesanti in termini di KB i due programmi si rivelano particolarmente utili per il rilascio di patch al gioco (o anche di distribuzioni frazionate in più capitoli o con risorse diverse, tipo solo sonoro, solo grafica) che non costringano il giocatore a riscaricare l'intero pacchetto del gioco. Per maggiori informazioni su queste e altre utilità si rinvia alla documentazione contenuta in Iblorb.

9.4 Test delle capacità

Immaginate di avere un gioco ambientato in una cittadina del vecchio West, e quando il protagonista entra in città passa vicino ad un cartello che dice "Dead Dog, Kansas, Popolazione 213" scritto con vernice scolorita su del legno stagionato a cui hanno sparato. Il giocatore non sa cosa c'è scritto fino a che non digita "ESAMINA IL CARTELLO" -- la descrizione della stanza ha solo la riga "Vedi un cartello qui", o simile, ad indicare la sua presenza. Ed immaginate di avere una grande immagine di un cartello stagionato che ha un paio di fori provocati da un fucile. Idealmente vorreste mostrare l'immagine se l'interprete del giocatore fosse in grado di gestire la grafica, o scrivere un paragrafo che descrive il cartello in caso contrario. È per questo che esiste **`glk_gestalt()`**.

Ci sono più di una dozzina di cose che potete testare[35]. In questo caso, la costante da usare deve essere `gestalt_Graphics`, e l'oggetto cartello dovrebbe essere come questo:

```
Object cartello "cartello stagionato"
       with name 'stagionato' 'cartello',
          description [;
               if (glk_gestalt (gestalt_Graphics, 0)) {
  ! inserire il codice per mostrare l'immagine del cartello
                    }
                    else "Il cartello dice ~DEAD DOG, KANSAS, POP.
                    213.~ O forse potrebbero essere 2130, o 21300 --
                    qualcuno ha fatto saltar via l'angolo del
                    cartello con un fucile.";
                ],
       has static;
```

(Fate caso allo 0 nella chiamata a `glk_gestalt` -- è obbligatorio. `glk_gestalt` richiede due argomenti, anche nel caso in cui il secondo non serva a niente). Qui ci sono le diverse costanti che si possono usare con glk_gestalt:

[35] Un ulteriore approfondimento al tema, a dire il vero qui già spiegato molto bene, è disponibile in appendice.

Costante	2ndo arg	Cosa restuisce
gestalt_Version	0	Un numero a 32 bit con il numero di versione di Glk codificato (primi sedici bit = numero di versione major, successivi otto = numero minor, ultimi otto = numero sub-minor, così 0.6.1 sarebbe 00000601 hex.)
gestalt_CharOutput	il codice del carattere da testare[36]	Serve a verificare se un particolare carattere (come una "n" con una tilde sopra) può essere stampato sulla piattaforma del giocatore. Restituisce una di queste tre costanti: gestalt_CharOutput_ExactPrint se può, gestalt_CharOutput_CannotPrint se non può, o gestalt_CharOutput_ApproxPrint se lo converte in qualcos'altro (come una normale "n" per la n-con-tilde, o "ae" per ae-con-legatura.)
gestalt_LineInput	il codice del carattere da testare	Come sopra, ma per vedere se un carattere può essere accettato durante l'input di una riga (cioè, l'input che viene trattato quando si preme il tasto Invio.)
gestalt_CharInput	il codice del carattere da testare	Come sopra, ma per vedere se un carattere può essere accettato durante l'input di un carattere (cioè, l'input che viene trattato un carattere alla volta.)
gestalt_MouseInput	il tipo di finestra da testare	1 se l'input da mouse è supportato, 0 se no.
gestalt_Timer	0	1 se il tempo reale è supportato, 0 se no.
gestalt_Graphics	0	1 se la grafica è supportata, 0 se no.
gestalt_DrawImage	il tipo di finestra da testare	Serve a testare in maniera specifica se le immagini possono essere posizionate in un particolare tipo di finestra (wintype_TextBuffer o wintype_Graphics). 1 se è possibile, 0 se no.
gestalt_GraphicsTransparency	0	Serve a testare se le immagini PNG con aree trasparenti vengono realmente mostrate con la trasparenza funzionante come dovrebbe. 1 se funziona, 0 se no.

[36] Ricordate che quando comunicate con un computer, tutto finisce per diventare zero e uno, comprese le lettere dell'alfabeto. Per quanto ne sa il computer, ogni lettera, ogni numero ed ogni segno di punteggiatura è semplicemente un numero. Un **set di caratteri** è solo un sistema di corrispondenza tra i numeri ed i caratteri che rappresentano. Ad esempio, in molti set di caratteri, la lettera A maiuscola viene rappresentata dal numero 65 (01000001 binario, 41 hex). Questo è vero nel famoso set di caratteri ASCII, ed è anche vero per il set di caratteri ISO 8859 Latin-1, che è quello che usa Glk. I caratteri che vanno da 32 a 126 dell'ISO 8859 Latin-1 sono garantiti funzionare su ogni piattaforma. (Consultate la tabella ISO 8859 Latin-1 per vedere a cosa corrispondono i vari numeri). Alcune piattaforme supporteranno degli altri caratteri. Altre no. Per determinare cosa è disponibile ci sono le costanti gestalt_LineInput, gestalt_CharInput e gestalt_CharOutput.

gestalt_Sound	0	1 se il suono è disponibile, 0 se no.
gestalt_SoundMusic	0	1 se possono essere suonati i MOD, 0 se no. (Può valere 0 anche se glk_gestalt (gestalt_Sound, 0); restituisce 1 -- in questo caso solo i suoni AIFF sono supportati.)
gestalt_SoundVolume	0	1 se glk_schannel_set_volume() è supportato, 0 se no.
gestalt_SoundNotify	0	1 se la routine HandleGlkEvent() può eseguire codice personalizzato nel momento in cui un suono termina l'esecuzione, 0 se no.
gestalt_Hyperlinks	0	1 se i collegamenti ipertestuali sono supportati, 0 se no.
gestalt_HyperlinkInput	il tipo di finestra da controllare	1 se i collegamenti ipertestuali sono supportati all'interno di un particolare tipo di finestra, 0 se no.

Le sezioni che seguono spiegano come avvantaggiarsi di queste di queste capacità -- ma mentre leggete queste sezioni, ricordate che prima di poter visualizzare la grafica, o suonare della musica, o dichiarare un collegamento ipertestuale, o quello che volete, dovete prima usare `glk_gestalt()` per verificare se l'interprete del giocatore *può* fare queste cose. Se non può, dovete trovare dei metodi alternativi -- descrizioni testuali delle immagini, e cose così -- o almeno visualizzare un messaggio che dica al giocatore di quali caratteristiche l'interprete ha bisogno per poter usare quel gioco. Idealmente dovreste controllare il risultato di gestalt *tutte le volte* che usate le capacità opzionali di Glk -- se fate questo controllo solo in `Initialise()`, vi esponete a problemi come quello che potrebbe accadere nel caso in cui il giocatore inizi a giocare con il vostro gioco sulla sua macchina desktop, poi salvi il gioco e prosegua la partita sul proprio palmare. (Questo non significa per forza che dovete digitare un sacco di chiamate a `glk_gestalt()` -- potreste, ad esempio, creare una routine `MyGlkImageDraw()` che esegua i controlli gestalt appropriati e poi visualizzi l'immagine che volete).

9.5 Finestre Glk

Quando avviate un interprete Glulx, che chiameremo "Glulxe" da adesso in poi, il sistema operativo del vostro computer gli riserverà una certa parte dello schermo. Questa potrebbe essere semplicemente un'altra finestra tra le tante, come su Windows o sul Mac; oppure potrebbe essere l'intero schermo, come in DOS, che esegue un solo programma alla volta. Ad ogni modo, per quel che interessa al gioco IF, lo spazio allocato a Glulxe è il mondo intero. Il gioco non sa niente di quello che c'è fuori dallo spazio di schermo di Glulxe, e non può alterare nulla al di fuori dello stesso.

All'inizio, all'interno dello spazio di schermo di Glulxe non c'è niente. E niente *può* esserci messo, fino a che una **finestra Glk** viene aperta. Una finestra Glk è una porzione dello spazio di schermo di Glulxe in cui si possono visualizzare cose e da cui, a volte, è possibile accettare degli input. Ci sono tre tipi di finestre che il programmatore deve conoscere:

- **Finestre di testo (Text buffer windows).** Questo tipo di finestra è in grado di visualizzare del testo ed anche le immagini, ma lo fa solo in maniera **streaming**. Cioè, quando il gioco dice alla finestra "visualizza questa frase" o "mostra questa immagine", la frase o l'immagine vengono accodate *dopo* qualunque testo o immagine sia già presente. Quando la finestra di testo si riempie, la roba vecchia scorre fuori dalla stessa e scompare: in alcuni interpreti quel materiale se ne è andato dallo schermo per sempre, mentre altri possono fornire una barra di scorrimento, così che si possa vedere quello che è sparito tornando indietro di una pagina. La finestra dove i giocatori digiteranno i propri comandi sarà una finestra di testo, tranne nei casi più diversi dal solito.
- **Finestre a griglia di testo (Text grid windows).** Anche questo tipo di finestra è in grado di visualizzare del testo, ma lo fa in maniera differente dalla finestra di testo. Le finestre a griglia di testo consentono al programmatore di stampare i caratteri dove vogliono, come quando si mettono le lettere sulla tavola dello Scarabeo. Questo le rende ideali nel caso in cui vogliate sovrascrivere le vecchie informazioni, come nella riga di stato usata dalla maggior parte dei giochi IF. Le finestre a griglia di testo possono accettare gli input dalla tastiera -- la pressione di freccia-sù e freccia-giù per muovere la freccia di selezione all'interno di un menu, ad esempio -- ma qualunque cosa di più avanzato richiede qualche modifica alla libreria.
- **Finestre grafiche (Graphics windows).** Questo tipo di finestra non accetta affatto il testo, ma consente al programmatore di giochi di visualizzare immagini (nel formato PNG o JPEG) che non scorre via mentre il gioco procede. Possono anche creare le proprie immagini usando i comandi di disegno di Glulx Inform, se ne hanno la pazienza. Le finestre grafiche non accettano input da tastiera, ma *possono*

riconoscere i click del mouse (una capacità molto utile, condivisa con le finestre a griglia di testo).

I dettagli su come usare questi tipi di finestra verranno illustrati nelle sezioni seguenti. Il resto di questa sezione riguarderà il come crearle e come costruire un layout personalizzato per il vostro gioco.

Due finestre vengono create automaticamente dalla libreria (sebbene questo possa essere evitato usando il punto di partenza (entry point) `InitGlkWindow()`. Fra poco ne sapremo di più). La prima finestra è una finestra di testo chiamata "`gg_mainwin`", che riempie l'intero spazio di schermo di Glulxe. Questa è suddivisa in due, con una finestra a griglia di testo chiamata "`gg_statuswin`" che occupa una riga alla cima dello schermo di Glulxe, e `gg_mainwin` che occupa il resto.

Se volete aggiungere altre finestre, qui è spiegato come farlo. Prima di tutto bisogna decidere che tipo di finestra deve essere. Forse volete una mappa disegnata del mondo di gioco che sia sempre visualizzata sullo schermo. Questo porta a scegliere una finestra grafica. Poi bisogna scegliere un nome per la finestra. La potete chiamare come volete, ma visto che la libreria chiama le finestre tutte con il prefisso "gg_", forse è il caso che lo facciate anche voi. Così la finestra si chiamerà "gg_mapwin".

Poi dobbiamo scegliere un `rock value` per questa finestra. Questo deve essere un numero pari a 210 o superiore. (Se non sapete ancora che cosa è un `rock value` non vi scoraggiate -- scegliete semplicemente il numero 210 o superiore, che non sia stato usato in un'altra finestra, ed andate avanti)[37]. Per questo esempio sceglieremo il numero 210. Il passo è quello di aggiungere un paio di righe al nostro codice. All'interno della vostra lista di costanti, aggiungete questa riga:

```
Constant GG_MAPWIN_ROCK 210;
```

ed all'interno della vostra lista di variabili globali, questa:

```
Global gg_mapwin = 0;
```

[37] La spiegazione di cosa sia un Rock value oltre che diffusa nel testo è approfondita in appendice.

Con queste righe aggiunte al codice siamo ora in grado di creare la nostra nuova finestra. Il comando chiave è `glk_window_open()`. All'interno delle parentesi ci vanno cinque parametri:

1 Il primo è il **nome** della finestra da dividere in due parti. In questo caso stiamo dividendo la finestra principale, quindi usiamo `gg_mainwin`.

2 Poi ci va il **metodo** che usiamo per dividere la finestra. Questo si ottiene con un processo in due parti. Prima dobbiamo scegliere una delle seguenti quattro opzioni:

- `winmethod_Above`: divide la finestra in due parti, una sopra l'altra, ed imposta la parte superiore come la nuova finestra
- `winmethod_Below`: divide la finestra in due parti, una sopra l'altra, ed imposta la parte inferiore come la nuova finestra
- `winmethod_Left`: divide la finestra in due parti, affiancate, ed imposta la parte di sinistra come la nuova finestra
- `winmethod_Right`: divide la finestra in due parti, affiancate, ed imposta la parte di destra come la nuova finestra

Poi dobbiamo scegliere una delle seguenti opzioni:

- `winmethod_Fixed`: riserva un certo numero di righe o colonne (per le finestre di testo) o di pixel (per le finestre grafiche) per la nuova finestra
- `winmethod_Proportional`: riserva una certa *percentuale* della vecchia finestra per la nuova

Unitele con un segno di addizione (+). Così se abbiamo deciso che la mappa deve stare permanentemente nella parte sinistra dello spazio di schermo di Glulxe, e sappiamo che la mappa è larga 240 pixel, probabilmente andremo a scegliere

```
(winmethod_Left+winmethod_Fixed).
```

E il "240" dove va? Beh, va nella...

3 terza posizione, dove mettiamo: il numero di righe o colonne da riservare per la nuova finestra (se è una finestra di testo, ed abbiamo scelto l'opzione `winmethod_Fixed`); il numero di pixel da riservare per la nuova finestra (se è una finestra grafica, ed abbiamo scelto `winmethod_Fixed`); o, se abbiamo scelto `winmethod_Proportional`, la percentuale della vecchia finestra da riservare per la nuova (ad esempio, avremmo messo "50" se avessimo voluto dividere la vecchia finestra a metà, oppure "33" se avessimo voluto riservare 1/3 dello spazio di schermo della vecchia finestra per la nuova).

4 Qui inseriamo il tipo di finestra che vogliamo: `wintype_TextBuffer`, `wintype_TextGrid` o `wintype_Graphics`. Nel nostro esempio vogliamo usare l'ultimo dei tre.

5 L'ultima posizione riguarda la lista delle costanti che abbiamo dichiarato in precedenza. In questo caso deve essere `GG_MAPWIN_ROCK`.

Questo ci porta ad avere la seguente chiamata:

```
gg_mapwin = glk_window_open(gg_mainwin, (winmethod_Left+winmethod_Fixed),
                                    240, wintype_Graphics, GG_MAPWIN_ROCK);
```

Naturalmente il semplice inserimento di una riga come questa nel vostro programma tutta da sola è una pessima idea -- ci sono una serie di condizioni che dovete controllare. Tanto per dirne una, prima di aprire una finestra è necessario assicurarsi che la finestra non sia già aperta. Perché dovrebbe? Forse perché il giocatore ha appena caricato un gioco salvato (che *non* ripristina automaticamente la configurazione della finestra dal momento del salvataggio: dovete pensarci per conto vostro. Come fare verrà spiegato tra breve). Così inseriamo il tutto in una condizione:

```
if (gg_mapwin == 0) {
   gg_mapwin = glk_window_open(gg_mainwin,
      (winmethod_Left+winmethod_Fixed), 240,
      wintype_Graphics, GG_MAPWIN_ROCK);
}
```

Inoltre bisogna tenere presente che la richiesta di apertura di una finestra non è detto che vada a buon fine. Il giocatore potrebbe stare usando un interprete che non supporta la grafica, ad esempio, nel qual caso il tentativo di aprire una finestra grafica è destinato a fallire. O forse l'interprete supporta la grafica, ma il giocatore ha impostato l'intero spazio di schermo di Glulxe ad essere largo meno di 240 pixel, così la finestra che ottenete non è esattamente quella alla quale avevate pensato. Certamente vorrete controllare questo tipo di cose; scegliere cosa fare in base ai risultati di questi controlli sta a voi. Ad esempio, se il vostro gioco è incomprensibile senza grafica, allora dovreste controllare se la finestra grafica è stata aperta realmente (con un test tipo `if (gg_graphics1win == 0)` dopo la chiamata a `glk_window_open()`) e, se non lo è, visualizzare un messaggio come "Questo gioco ha bisogno di un interprete in grado di visualizzare la grafica. Spiacente!" ed uscire dal gioco. Ma se la grafica è semplicemente una divertente aggiunta, allora dovreste far continuare il vostro gioco, ma dovreste assicurarvi di non cercare di usare `glk_image_draw()` per

mettere un'immagine in quella (inesistente) finestra dopo non essere riusciti ad aprirla -- eseguite un controllo tutte le volte.

Per chiudere una finestra si usa il comando `glk_window_close()`. Questo richiede due argomenti. Il primo è il nome della finestra da chiudere. Il secondo è estremamente esoterico; metteteci semplicemente 0 e vivete felici.

9.6 Stili del testo

Come indicato in precedenza, chi desidera compilare i propri codici sorgenti scritti in Inform per Glulx, deve assicurarsi di sostituire tutte le istruzioni Z-assembly in istruzioni Glulx assembly (o chiamare le routine di Glulx Inform che fanno quello che faceva lo Z-assembly). L'altra importante sostituzione che i programmatori di giochi devono fare, riguarda le chiamate tipo `style bold;` e `style underline;` -- Glulx non le riconosce. Al loro posto dovete usare le chiamate di stile di Glulx Inform[38].

Le chiamate di stile di Glulx Inform funzionano quasi allo stesso modo delle normali chiamate di stile di Inform. Si usa il comando `glk_set_style()` con lo stile che si vuole applicare tra parentesi, ed ogni testo visualizzato dopo tale comando apparirà nello stile specificato. Non c'è una "disattivazione" dello stile, come in HTML con i suoi tag </style> -- per ricominciare a visualizzare il testo con lo stile normale, si usa semplicemente la riga `glk_set_style(style_Normal);` Le altre opzioni includono:

- `style_Emphasized`: per il testo in risalto.
- `style_Preformatted`: per le immagini formate da caratteri, le tabelle, o qualunque altra cosa abbia bisogno di una particolare sistemazione di caratteri a spaziatura fissa.
- `style_Header`: per introdurre delle sezioni ampie (es., "ATTO I").
- `style_Subheader`: per introdurre delle sezioni inferiori all'interno delle sezioni ampie indicate sopra (es., "scena i").
- `style_Alert`: per degli avvisi molto importati (es., "*** Sei morto ***").
- `style_Note`: per avvisi meno importanti (es., "[Il tuo punteggio è aumentato di 5 punti.]")
- `style_BlockQuote`: per iscrizioni e cose simili.
- `style_Input`: per il testo che il giocatore inserisce.

[38] Con Inform 6.30 è ora possibile usare le chiamate "style" anche in giochi Glulx Inform, rimane quindi necessario sostituirle solo se si usa una versione precedente di Inform.

Come appaiono esattamente questi stili? Varia in base all'interprete. `style_Emphasized` potrebbe essere in grassetto, o in corsivo, o semplicemente chiaro. Potete, naturalmente, fornire dei suggerimenti riguardo i vari aspetti di un particolare stile (compresi i due stili che sono lasciati al programmatore da definire: `style_User1` e `style_User2`). Alcuni interpreti terranno conto di questi suggerimenti; altri no. (Ed altri ne terranno conto in maniera errata: DOS e Allegro Glulxe al momento prendono i suggerimenti per i colori di primo piano e sfondo al contrario!) Ad oggi non esiste una `gestalt_StyleHints` o qualcosa di simile per verificare se i suggerimenti verranno o meno ignorati[39]. Ma se vi sentite fortunati, usate la chiamata a `glk_stylehint_set()`, che richiede quattro argomenti:

1 Il tipo di finestra alla quale il suggerimento di stile si dovrebbe applicare. Le opzioni sono `wintype_TextBuffer`, `wintype_TextGrid` o `wintype_AllTypes` (ma non wintype_Graphics, visto che non è possibile visualizzare il testo in una finestra grafica).

2 Lo stile al quale il suggerimento di stile andrebbe applicato.

3 L'aspetto dello stile da modificare (segue la lista).

4 Il numero o la costante che indica come quell'aspetto dello stile dovrebbe essere modificato.

Il significato del quarto argomento cambia drasticamente in base a cosa c'è nel terzo argomento: "1" potrebbe significare "indenta di una piccola quantità", "grassetto", "quasi nero", o molte altre cose a seconda del contesto. Così il terzo ed il quarto argomento non dovrebbero essere considerati separatamente, ma in tandem. Segue una tabella delle combinazioni attualmente supportate:

ASPETTO **(terzo argomento)**	**VALORE** **(quarto argomento)**
stylehint_Indentation	Questo dovrebbe essere un numero, il cui significato è "sposta il seguente blocco di testo di queste unità a destra" (i numeri negativi sposteranno il testo a sinistra.) Quanto è grande un'unità? Varia a seconda dell'interprete.

[39] Alcuni interpreti solitamente sono accompagnati da un file di configurazione per gli stili di testo, la grandezza della finestra dell'applicazione etc... vale la pena darvi un'occhiata.

stylehint_ParaIndentation	È come stylehint_Indentation, ma riguarda solo la prima riga di ogni paragrafo.
stylehint_Justification	Ci sono quattro costanti tra cui scegliere: stylehint_just_LeftFlush (allineato a sinistra), stylehint_just_RightFlush (allineato a destra), stylehint_just_LeftRight (giustificato), e stylehint_just_Centered (centrato).
stylehint_Size	Dovrebbe essere un numero, ma non un numero assoluto (come la dimensione di un punto); invece, 0 significa "usa la dimensione di default del font", mentre i numeri positivi aumentano la dimensione del font di un certo numero di incrementi ed i numeri negativi la diminuiscono. Gli incrementi non sono necessariamente della stessa dimensione: se 0 è 12 punti, e +1 è 14 punti, +2 potrebbe essere 18 punti.
stylehint_Weight	0 = normale; 1 = grassetto; -1 = sottile.
stylehint_Oblique	0 = non corsivo; 1 = corsivo.
stylehint_Proportional	0 = font a larghezza fissa; 1 = font proporzionale.
stylehint_TextColor	Dovrebbe essere un numero a 32 bit indicante il colore da usare. Il modo di gran lunga più semplice di indicarlo è quello di farlo in esadecimale[40], che viene fatto come segue. Prima si digita il simbolo del dollaro (il quale indica che quello che segue è un numero esadecimale.) Poi, si scrive un numero a due cifre esadecimali, da 00 a FF, che indica la quantità di rosso da applicare al colore. Quindi viene un numero a due cifre esadecimali che rappresenta la quantità di verde, ed infine un numero a due cifre esadecimali che indica la quantità di blu. Così, $000000 sarà nero, $FFFFFF sarà bianco, $FF000000 sarà rosso chiaro, $FFC000 sarà un gradevole dorato, $C0C0FF sarà blu chiaro, e così via.
stylehint_BackColor	Questo è un numero a 32 bit proprio come sopra, solo che questa volta si sceglie il colore che deve apparire *dietro* il testo. D'altra parte questo non è il colore dello sfondo della finestra, i risultati possono arrivare ad essere estremamente sgradevoli su alcuni interpreti. Ad oggi non c'è un suggerimento di stile per indicare il colore di sfondo della finestra; con qualche fortuna verrà aggiunto tra non troppo tempo.
stylehint_Reverse	0 = scrivi normalmente; 1 = scrivi con il colore di sfondo sopra il colore di primo piano.

Così, mettendo tutto insieme, diciamo che volete definire style_User1 come testo rosso su uno sfondo nero. Questo si ottiene mediante il seguente codice:

```
glk_stylehint_set (wintype_TextBuffer, style_User1,
    stylehint_TextColor, $FF0000);
glk_stylehint_set (wintype_TextBuffer, style_User1,
    stylehint_BackColor, $000000);
```

[40] Vedi nota sugli esadecimali.

Comunque i suggerimenti di stile hanno effetto solo sulle *finestre create in seguito*. Questo significa che se volete usarli, dovete definirli prima della creazione della finestra nella quale volete appaiano. E visto che `gg_mainwin` viene creata dalla libreria, se volete che appaiano nella normale finestra di gioco, mettere i vostri suggerimenti di stile in `Initialise()` è troppo tardi -- dovete usare il punto d'ingresso **`InitGlkWindow()`**. `InitGlkWindow()` viene chiamata diverse volte, passandole un parametro differente ogni volta; nel nostro caso, se volete dichiarare i vostri suggerimenti di stile e se il valore è uguale al rock value di `gg_mainwin`, avremo:

```
[ InitGlkWindow winrock;
   switch (winrock) {
      GG_MAINWIN_ROCK:
          glk_stylehint_set (wintype_TextBuffer,
             style_User1,stylehint_TextColor,$FF0000);
          glk_stylehint_set (wintype_TextBuffer,
             style_User1, stylehint_BackColor, $000000);
   }
   rfalse; ! leaving out this line will lead to a messy crash!
];
```

Un'ultima nota prima di proseguire. Potrebbe sembrarvi che per stampare a schermo la frase “Io dico che la violenza **non** è la risposta giusta.” del codice come:

```
print "Io dico che la violenza ";
glk_set_style(style_Emphasized);
print "non";
glk_set_style(style_Normal);
print " è la risposta giusta.";
```

potrebbe essere un po' scomodo. Lo è. D'altra parte, non è molto peggio di

```
print "Io dico che la violenza ";
style bold;
print "non";
style roman;
print " è la risposta giusta.";
```

È per questo che molti autori di IF usano già delle piccole routine che rendono questo tipo di cose più comodo. Non c'è bisogno di digitare `style_Emphasized` un centinaio di volte; qualcosa come questo funzionerà tranquillamente:

```
[ b text;
     glk_set_style(style_Emphasized);
     print (string) text;
     glk_set_style(style_Normal);
  ];
```

E una volta che avete creato quella routine, potete rendere la frase dell'esempio con:

```
print "Io dico che la violenza ", (b) "non", " è la risposta giusta.";
```

9.7 Righe di stato (Status Lines)

Uno degli usi più comuni dello Z-assembly nei giochi Inform è quello di creare delle righe di stato personalizzate. Questa cosa non funziona in Glulx Inform, visto che non riconosce lo Z-assembly. Invece dovrete usare gli equivalenti Glk che non sono opcode diretti, ma sono delle routine come altre.

La riga di stato è una tale tradizione IF da richiedere l'impegno di qualche sforzo. Un gioco standard Inform divide una porzione dello schermo per mostrare la stanza in cui si trova il personaggio, il punteggio del giocatore ed il numero di mosse fatte. Per cambiare la cosa, il programmatore deve inserire la riga `Replace DrawStatusLine;` all'inizio del programma (metterla dopo la lista delle costanti funziona, come in qualunque altro posto). Quindi il programmatore deve fornire una routine `DrawStatusLine()` alternativa. Ma qui è dove le cose differiscono un po'.

In Inform nelle librerie 6/10, il comando per creare la finestra della riga di stato `@split_window` si trova all'interno della routine `DrawStatusLine()` della libreria. Se volete sostituirla con una vostra, dovete inserire il comando `@split_window` da voi. Se volete *eliminarla*, tutto quello che dovete fare è sostituire `DrawStatusLine()` con una routine vuota, come:

```
[ DrawStatusLine; ];
```

Non è abbastanza in Glulx Inform. Nella sezione Glulx della libreria 6/11, la finestra di stato viene aperta in `GGInitialise()` e non in `DrawStatusLine()`. Questo significa che se usate una `DrawStatusLine()` vuota, la riga di stato sarà vuota, ma la finestra sarà ancora presente, facendosi beffe di voi. Così dovrete intercettare il comando che crea la finestra della riga di stato usando il punto di ingresso `InitGlkWindow()`, come segue:

```
[ InitGlkWindow winrock;
   switch (winrock) {
      GG_STATUSWIN_ROCK: rtrue;
   }
   rfalse; ! levare questa riga porta ad uno sporco crash!
];
```

Se volete mantenere la riga di stato, ma volete fornire le vostre istruzioni, ci sono una serie di linee guida da seguire. Prima di tutto, se volete allocare più di una riga per la finestra di stato, fatelo in `InitGlkWindow()`, così:

```
[ InitGlkWindow winrock;
   switch (winrock) {
      GG_STATUSWIN_ROCK:
         gg_statuswin_size = 2; ! o quante righe volete
   }
   rfalse; ! levare questa riga porta ad uno sporco crash!
];
```

Poi scrivete una routine che sostituisca `DrawStatusLine()`, che inizi con il codice seguente:

```
! Se non abbiamo una riga di stato, non dobbiamo cercare di
! ridisegnarla.
   if (gg_statuswin == 0)
      return;

! Non dobbiamo farlo neanche se non esiste una locazione
! per il giocatore
   if (location == nothing || parent(player) == nothing)
      return;
```

La prima riga dopo quella deve essere `glk_set_window(gg_statuswin);` per impostare la finestra nella quale si vuole visualizzare la riga di stato, e l'ultima riga della routine deve essere "`glk_set_window(gg_mainwin);` per essere sicuri che il programma continui a mettere il testo della storia nel posto giusto, ovvero la finestra principale. In mezzo potete visualizzare quello che volete, sebbene il processo da seguire sia leggermente differente da quello che si usa per una finestra di testo.

Ricordate che una finestra a griglia di testo è una specie di tavola dello Scarabeo: sebbene i bordi tra le caselle non siano visibili, la finestra è composta da una griglia di caselle, con ognuna che può contenere un carattere -- lettere, numeri, punteggiatura. Ogni casella ha un paio di coordinate associate, una coordinata X ed una coordinata Y. La coordinata X indica a quante caselle di distanza si trova la casella corrente dal bordo sinistro della finestra; la coordinata X delle caselle *sul* bordo sinistro è 0. La coordinata Y indica a quante caselle di distanza si trova la casella corrente dal bordo superiore della finestra; la coordinata Y delle caselle *sul* bordo superiore è 0. (Questa cosa differisce dal normale Inform, dove l'angolo in alto a sinistra non è (0,0), ma (1,1)). Per selezionare il posto in cui cominciare a stampare si usa il comando `glk_window_move_cursor()`, che richiede tre argomenti: il nome della finestra in cui stampare, la coordinata X della casella da cui si comincia a stampare e la coordinata Y della casella da cui si comincia a stampare. Mentre stampate il cursore si sposta automaticamente -- non avete bisogno di spostarlo voi per stampare la lettera successiva di una parola. Così, il codice seguente:

```
glk_window_move_cursor(gg_statuswin, 3, 0);
print "Ora: "
```

metterà la lettera "O" alle coordinate (3,0), "r" a (4,0), "a" a (5,0), i due punti a (6,0) ed uno spazio a (7,0), lasciando il cursore a (8,0). Probabilmente dopo verrà del codice che stamperà l'ora corrente.

Una cosa da tenere presente è che non tutti quelli che giocheranno il vostro gioco avranno la stessa quantità di spazio di schermo allocata da Glulxe: la finestra della riga di stato potrebbe essere lunga 120 caratteri sulla macchina di una persona, e 40 su un'altra. O qualcuno potrebbe ridimensionare la propria finestra di Glulxe da una larghezza di 120 caratteri a 40 nel corso di una partita. Potete, comunque, mettere del codice in `DrawStatusLine()` per sistemare queste cose (come, in realtà, viene già fatto nella routine standard). Semplicemente mettete delle variabili chiamate `width` e `height` nella dichiarazione della vostra `DrawStatusLine()`, ed inserite il seguente pezzo di codice nella routine:

```
glk_window_get_size (gg_statuswin, gg_arguments, gg_arguments+4);
        width = gg_arguments-->0;
        height = gg_arguments-->1;
```

Quindi potete fare tutti gli aggiustamenti che ritenete opportuni[41]. Diciamo che state lavorando ad un gioco chiamato *Crunch, Crumple and Stomp* e volete mettere un indicatore di fame nella riga di stato, iniziando da colonna 40, che visualizzi "PIENO", "SAZIO", "AFFAMATO", "VORACE". Questo ci porta fino a colonna 47. Alcune persone, però, potrebbero giocare con meno di 48 colonne, così volete che l'indicatore compaia solo se c'è abbastanza spazio (visto che non volete che l'indicatore tagli a metà le parole). Si usi semplicemente del codice come questo:

```
   if (width > 48) { ! imponiamo un po' di chiarezza
      glk_window_move_cursor(gg_statuswin, 40, 0);
      switch (gojira.fame) {
         0: print "PIENO";
         1: print "SAZIO";
         2: print "AFFAMATO";
         3: print "VORACE";
      }
   }
```

Notate che le possibilità hanno differenti lunghezze. Se cercate di sovrascrivere "AFFAMATO" con "PIENO" ottenete "PIENOATO", a meno che riempiate a spazi ogni possibilità per ottenere per tutti la stessa lunghezza, o inserite la riga `glk_window_clear(gg_statuswin);` subito dopo la vostra chiamata a `glk_set_window(gg_statuswin);`.

Sono possibili degli aggiustamenti migliori che il non stampare del tutto le voci che non entrano -- potremmo aver scritto un codice elaborato che abbrevia le cose quando la riga di stato diventa più piccola, ad esempio. La cosa simpatica è che, visto che `DrawStatusLine()` viene chiamata ad ogni turno, non dovrete stargli dietro attraverso la `HandleGlkEvent()` per avere a che fare con i ridimensionamenti, come invece accade per la grafica. Troverete altro su questo argomento nelle prossime sezioni[42].

---

[41] Per sapere cosa è gg_arguments e comprendere meglio questa istruzione leggete la parte introduttiva della appendice dedicata alle funzioni della libreria infglk.

[42] Per una esposizione completa del funzionamento delle entrypoint come HandleGlkEvent si veda in appendice.

9.8 Il codice della Routine DrawStatusLine

Di seguito riportiamo il codice commentato riga per riga della `DrawStatusLine()` come da libreria 6/11, pensando sia un valido supporto per chi desidera personalizzare la riga di stato nel proprio gioco.

```
! la status line si può rimpiazzare
#Ifndef DrawStatusLine;

! la drawstatusline è Una normale funzione in Inform, con
! tre variabili locali.



[ DrawStatusLine width posa posb;
    #Ifdef TARGET_GLULX;

    ! Se non c'è una status line, non dobbiamo tentare di
    ! disegnarla.
    if (gg_statuswin == 0)
        return;
    #Endif;

! gg_statuswin è la variabile globale della libreria che
! contiene il riferimento alla finestra status. Alcune
! librerie non supportano una finestra status; se non ve ne
! è una, gg_statuswin contiene zero. E quindi, non c'è
! nulla da fare qui e si può operare un return subito.

    ! Se il giocatore non è nella locazione, non dovremmo
    ! provare a  disegnare la status line
    if (location == nothing || parent(player) == nothing)
        return;

! L'evenienza su riportata non capiterà in molti giochi, ma
! è meglio prevederla. (un modo in cui potrebbe verificarsi
! è se si pone una domanda YesOrNo() nella procedura
! Initialise().)
```

```
    StatusLineHeight (gg_statuswin_size);
    MoveCursor (1, 1);

!StatusLineHeight() è una funzione della libreria che semplicemente
!imposta l'altezza della status line ad un determinato valore. E'
!meglio ricordare la grandezza corrente, e lasciare che la finestra
!la decida da sola se non si necessita di cambiamenti. La variabile
!globale gg_statuswin_size di default è posta ad 1 (una linea), e
!la libreria non la cambia in nessun punto.
!MoveCursor imposta come finestra di lavoro la status win e quindi
!muove il cursore alle coordinate 1 1. Da alla variabile
!statuswin_current in valore 1 (che significa che ci si sta
!muovendo nella finestra di stato.

    width = ScreenWidth();

!Questa istruzione misura la grandezza della finestra status.

    posa = width-26; posb = width-13;

!Una volta che sappiamo la larghezza, ne sfrutteremo il valore in
!due variabili locali:  posa e posb, ovvero la posizione orizzontale
!degli indicatori del punteggio e del conto dei turni di gioco.

    spaces width;
    MoveCursor (1, 2);

! stampiamo tanti spazi quanti è la larghezza della finestra e
! quindi spostiamo il cursore[43] e poi stampiamo il nome della locazione:
    if (location == thedark) {
        print (name) location;
    }
    else {
        FindVisibilityLevels();
        if (visibility_ceiling == location)
```

[43] Sposta il cursore nella posizione (1,2) -- che corrisponde al secondo carattere della prima linea. La posizione in alto a sinistra nella finestra Glk è infatti numerata come (0,0). Il che è differente dalle finestre in Z-code, dove la posizione in alto a sinistra corrisponde a (1,1). Altra differenza, solo come importanza, è che la MoveCursor () richiede due coordinate nell'ordine (X, Y). Lo Z-code @set_cursor opcode usa invece (Y, X).

```
            print (name) location;
        else
            print (The) visibility_ceiling;
    }
!Se preferiamo l'orologio ai turni e punti:
    if (sys_statusline_flag && width > 53) {
        MoveCursor(1, posa);
        print(string) TIME__TX;
        LanguageTimeOfDay(sline1, sline2);
    }
!a meno di eliminare il punteggio
!ecco il classico punti: e turni:
    else {
        if (width > 66) {
            #Ifndef NO_SCORE;
            MoveCursor(1, posa);
            print (string) SCORE__TX, sline1;
            #Endif;
            MoveCursor(1, posb);
            print (string) MOVES__TX, sline2;
        }

!oppure se c'è poco spazio punti/turni:
        #Ifndef NO_SCORE;
        if (width > 53 && width <= 66) {
            MoveCursor(1, posb);
            print sline1, "/", sline2;
        }
        #Endif;
    }

!obbligatorio tornare alla finestra principale:
    MainWindow(); ! set_window
];
#Endif;
```

9.9 Visualizzare immagini PNG e JPEG

Se volete che il vostro gioco visualizzi un'immagine dovrete fare un po' di lavoro preparatorio. Il nome del file dell'immagine deve essere elencato nel file di risorse Blorb associato al vostro programma (consultate il paragrafo dedicato a Blorb) e gli deve essere dato un nome con il quale riferirsi alla stessa all'interno del vostro sorgente.

Glk supporta due formati di immagine: PNG e JPEG. JPEG è un formato lossy (a perdita d'informazione), che usa un algoritmo di compressione che riduce considerevolmente la dimensione del file a scapito di una leggera infedeltà dell'immagine rispetto all'originale. PNG è un formato non-lossy (senza perdita d'informazione) con supporto per i pixel trasparenti, molto simile a GIF, solo che è senza le potenziali controversie legali che potrebbero nascere dall'uso di GIF (che utilizza un algoritmo di conversione brevettato). JPEG va meglio per le foto, mentre PNG va bene per cose come disegni a mano, immagini di testo e cose simili. Se la vostra immagine non è in uno di questi due formati, dovete usare un programma di grafica per convertirla prima di poterla usare nel vostro gioco Glulx Inform.

Poi dovete decidere dove deve andare la vostra immagine. La porzione di spazio di schermo data all'interprete Glulx sarà divisa in una serie di finestre (consultate il paragrafo dedicato alle Finestre Glk), e dovete scegliere quale di queste sarà la destinazione dell'immagine in questione. Quello che dovrete fare poi dipende dal tipo della finestra di destinazione.

- Se volete inframmezzare di immagini il testo della vostra storia, allora volete mettere la grafica in una finestra di testo.
- Se volete visualizzare un'immagine in un'area dello schermo riservata allo scopo, allora volete mettere la grafica in una finestra grafica.

9.10 Grafica in una finestra di testo

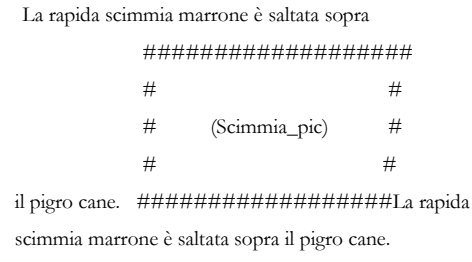
Così avete deciso che volete mettere un'immagine direttamente nel testo della vostra storia IF. In realtà questa cosa è relativamente semplice: si chiama semplicemente la routine `glk_image_draw()` con quattro argomenti:

1 Il nome della finestra alla quale l'immagine deve essere inviata. A meno che stiate facendo qualcosa di molto strano con il layout della vostra finestra, mischiare le immagini con il testo significa inviarle alla finestra `gg_mainwin`.

2 Il nome dell'immagine da visualizzare, come l'avete chiamata nel vostro file di risorse Blorb. Se è l'immagine di una scimmia, ad esempio, potreste averla chiamata `Scimmia_pic`.

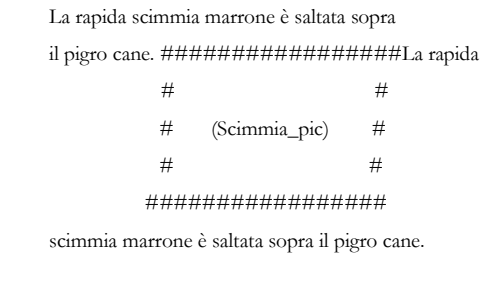
3 Il modo in cui volete che il testo scorra intorno all'oggetto. Qui avete cinque opzioni. Le prime tre servono se volete che l'immagine venga posizionata come se ci fosse un'altra parola sulla riga di testo dove appare. Forse il modo migliore di dimostrare quello che fa ogni opzione, è quello di usare dei diagrammi:

`imagealign_InlineUp`:



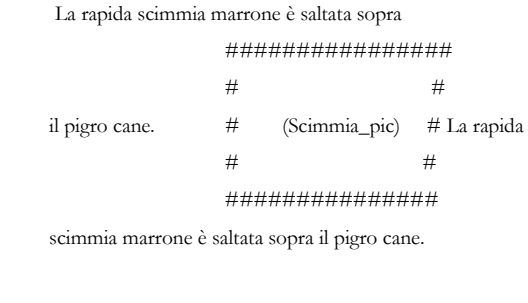
. . .

`imagealign_InlineDown`:



. . .

`imagealign_InlineCenter`:



Naturalmente se l'immagine è l'unica cosa sulla riga in questione, potete scegliere una qualunque di queste opzioni per avere lo stesso risultato.

. . .

Le successive due opzioni funzionano solo se l'immagine è la prima cosa sulla riga (ad es., se segue un carattere ^ in una stringa od una chiamata a "new_line;"). Qualunque testo che segue verrà disposto accanto all'immagine:

`imagealign_MarginLeft`:

```
 ##################La rapida
#                          # scimmia marrone
#     (Scimmia_pic)        # è saltata sopra
#                          # il pigro cane.
 ##################La rapida
scimmia marrone è saltata sopra
il pigro cane. La rapida scimmia
marrone è saltata sopra il pigro
```

. . .

`imagealign_MarginRight`:

```
La rapida            ################
scimmia marrone      #              #
è saltata sopra      #  (Scimmia_pic)  #
il pigro cane.       #              #
La rapida            ###############
scimmia marrone è saltata sopra
il pigro cane. La rapida scimmia
marrone è saltata sopra il pigro
```

. . .

4 Il quarto argomento è 0. Questo argomento ha significato solo quando si disegnano immagini su una finestra grafica; comunque non possiamo semplicemente ignorarlo, perché, a differenza delle normali funzioni di Inform, le funzioni Infglk non sono in grado di avere a che fare con meno argomenti di quelli che si aspettano.

Così diciamo che vogliamo che l'immagine di questa scimmia venga visualizzata quando il giocatore decide che il proprio personaggio vada allo zoo e gli dica >GUARDA LA SCIMMIA. Potremmo codificare la scimmia come segue:

```
Object scimmia "Bobo la scimmia"
   with name 'scimmia' 'bobo',
      description [;
         print "La scimmia somiglia a questo:^";
         glk_image_draw (gg_mainwin, Scimmia_pic,
            imagealign_InlineUp, 0);
         print "^"; rtrue;
      ],
   has animate proper;
```

Naturalmente c'è sempre la possibilità che il giocatore stia usando un interprete che non supporta la grafica, in questo caso il caricamento dell'immagine andrà in errore ed il giocatore rimarrà con una riga vuota dopo "a questo:". Per informazioni su come trattare questa possibilità, consultate la sezione sul Test delle capacità.

9.11 Grafica in una finestra grafica

Come con le finestre di testo, lo strumento principale per il posizionamento delle immagini PNG e JPEG in una finestra grafica è `glk_image_window()`. Questa volta richiede i seguenti quattro argomenti:

1) Il nome della finestra in cui deve essere posizionata l'immagine (gg_mapwin, ad esempio)

2) L'immagine da posizionarvi (ma seguite la discussione che segue la lista degli argomenti)

3 e 4) Le coordinate di dove volete che vada l'angolo in alto a sinistra dell'immagine. Il terzo argomento rappresenta quanti pixel dal bordo sinistro della finestra deve andare, ed il quarto è quanti pixel dal bordo superiore della finestra deve andare, quindi se volete che l'immagine si appoggi a questi bordi, mettete 0 in entrambi. (Perché vorreste poter mettere le immagini da una parte che non sia 0,0? Forse volete prima mettere l'immagine di un bel bordo decorato, e caricare un'immagine più piccola all'interno del bordo, magari a 10,10). Non dovete preoccuparvi che l'immagine esca dal bordo destro o inferiore della finestra -- tutto quello che eccede i limiti viene tagliato.

La parte delicata è il secondo argomento. Quando inserite le immagini in una finestra di testo, queste diventano un'altra parte del flusso che il programma invia alla finestra, così quando un `restore` od un `undo` cambiano la posizione del giocatore nel gioco, la libreria gestisce automaticamente le immagini. Questo non accade se inserite le immagini in una finestra separata. Ecco un esempio. Diciamo che avete un gioco con una finestra grafica fissa dove visualizzate un'immagine della stanza nella quale il giocatore si trova. E diciamo che scegliete di farlo inserendo direttamente l'immagine che volete visualizzare quando il giocatore entra nella stanza. Così, se la cucina è a sud della sala da pranzo, potreste avere un blocco di codice nell'oggetto sala da pranzo come questo:

```
s_to [;
      if (gg_picwin) { ! verifichiamo che esista la finestra
         glk_image_draw (gg_picwin, Cucina_pic, 0, 0);
      }
      PlayerTo(Cucina);
   ],
```

Questo causa, in effetti, la visualizzazione dell'immagine della cucina nella finestra appropriata quando il personaggio del giocatore entra nella cucina. Ma se il giocatore decide di fare un restore od un undo al punto in cui si trovava nella sala da pranzo? Nella finestra della storia sarà tornato nella sala da pranzo -

- ma la finestra grafica continuerà a mostrare un'immagine della cucina! La libreria non aggiorna automaticamente le finestre grafiche. Dovete farlo voi.

9.12 La gestione delle finestre grafiche

La libreria di inform offre una serie di **punti di ingresso** (entry point) che rendono la gestione delle finestre un po' più facile. Un punto di ingresso, come abbiamo visto prima, è una specie di routine opzionale che i programmatori possono inserire nel loro codice sorgente. Molto spesso gli scrittori di librerie arrivano ad un punto nel codice della libreria nel quale sospettano che i programmatori *potrebbero* voler eseguire delle istruzioni personalizzate, oppure no. Così lo scrittore di librerie crea una routine vuota che non fa nulla, ed inserisce la chiamata alla routine in quel punto. Ora il programmatore può scrivere una routine con lo stesso nome, e quando la libreria arriva a quel punto, esegue il codice della routine del programmatore del gioco, che fa qualcosa. O il programmatore del gioco può lasciar perdere, e quando la libreria arriva a quel punto esegue la routine vuota e continua tranquillamente per la sua strada.

Così se non state creando nessuna finestra speciale, o nessun canale sonoro, o nessun riferimento a file, o nessun'altra cosa simile, non dovete preoccuparvi di nessuno dei punti di ingresso che seguono. La libreria si occuperà di `gg_mainwin` e `gg_statuswin` per voi, così come di ogni file di salvataggio di gioco che generate. Ma se create i vostri oggetti Glk, come una finestra grafica, è vostra responsabilità tenerne conto. Qui scopriamo come[44].

Prima di tutto dovete creare una routine per il punto di ingresso che viene chiamato dopo ogni restart, restore e undo. Questa è **`IdentifyGlkObject()`**. La ragione per la quale dovete avere una routine `IdentifyGlkObject` se create le vostre finestre grafiche è che dopo ogni restart, restore o undo, le variabili globali che puntano a queste finestre possono contenere valori errati, e le cose possono diventare brutte se non le azzerate.

La routine `IdentifyGlkObject()` viene chiamata, in realtà, tre volte, inviando un numero differente alla variabile locale `phase` (fase) ogni volta: prima 0, poi 1, infine 2. In `phase 0`, il vostro compito è quello di impostare tutte le vostre variabili di puntamento ai vostri oggetti Glk a 0. In `phase 1`, dovete resettare tutte le vostre finestre, i vostri stream ed i vostri riferimenti ai file. In `phase 2` resettate tutti gli altri oggetti che avete creato. La `phase 2` è anche quella nella quale dovete aggiornare le vostre finestre per mostrare le immagini giuste, i vostri canali sonori per suonare la musica giusta, e così di seguito.

---

[44] In appendice trovate un approfondimento sull'uso delle entrypoint di Inform Glulx.

Così continuando con gli esempi delle sezioni precedenti, diciamo che abbiamo una finestra grafica, `gg_picwin`, nella quale mostriamo un'immagine della stanza nella quale il personaggio del giocatore si trova. Ma, diversamente dalle sezioni precedenti, non disegniamo l'immagine sulla finestra con comandi come `glk_image_draw(gg_picwin, Cucina_pic, 0, 0)`. Invece creiamo una variabile globale chiamata `current_pic` (immagine corrente), ed una routine che somiglia a questa:

```
[ MyRedrawGraphicsWindows;
   glk_image_draw(gg_picwin, current_pic, 0, 0);
];
```

Ora, quando scriviamo la routine che sposta il giocatore dalla sala da pranzo a sud, in cucina, dobbiamo scrivere una cosa così:

```
s_to [;
    current_pic = Cucina_pic;
    MyRedrawGraphicsWindows();
    PlayerTo(Cucina);
 ],
```

Ed alla fine inseriamo il nostro punto d'ingresso `IdentifyGlkObject()`:

```
[ IdentifyGlkObject phase type ref rock;
      if (phase == 0) { ! Si azzerano i riferimenti a tutti i nostri oggetti.
         gg_picwin = 0;
         return;
      }
      if (phase == 1) { ! Reset le nostre finestre, stream, e rif. ai file.
         switch (type) {
            0: ! È una finestra.
               switch (rock) {
                  GG_PICWIN_ROCK: gg_picwin = ref;
               }
            1: ! È uno stream. Ma non ne abbiamo creato nessuno.
            2: ! È un rifer. a file. Ma non ne abbiamo creato nessuno.
         }
         return;
      }
      if (phase == 2) { ! Aggiorniamo i nostri oggetti.
         MyRedrawGraphicsWindows();
      }
];
```

Che cosa abbiamo fatto? All'inizio abbiamo preso la variabile globale per la finestra grafica che abbiamo creato in precedenza (che adesso è riempita con dati sporchi, grazie al reset) e l'abbiamo impostata a 0 -- è la `phase 0`. Poi la libreria ha trovato questo costrutto, non l'ha riconosciuto, e così ce lo ha passato per determinare che cosa è. Noi abbiamo detto "È il `'rock value'` per questo oggetto (che *non* si è perso durante il reset)" ed è lo stesso del `rock value` che abbiamo impostato per la nostra finestra grafica, quindi deve essere la nostra finestra grafica! Perciò faremo puntare nuovamente la variabile della finestra grafica a questa cosa. È la `phase 1`. Poi nella `phase 2` abbiamo fatto un aggiornamento della visualizzazione: visto che adesso sappiamo dove inviare l'immagine che deve essere visualizzata a questo punto, possiamo mostrarla.

La finestra grafica, adesso, dovrebbe reagire correttamente ai vari restore, undo e cose simili: se il giocatore sposta il proprio personaggio nella cucina, poi digita "UNDO", non solo il personaggio tornerà nella sala da pranzo, ma la finestra sostituirà l'immagine della cucina con l'immagine della sala da pranzo.

E per quanto riguarda gli eventi esterni che influiscono sul gioco? Ad esempio, il giocatore potrebbe ridimensionare la finestra di Glulxe, o cambiare la risoluzione del monitor -- come affrontare la cosa? Risposta: la libreria ha un ciclo che tiene traccia di queste cose, e questo ciclo fornisce un punto d'ingresso, chiamato **`HandleGlkEvent()`**. `HandleGlkEvent()` richiede due argomenti: `ev` è un array che contiene le informazioni su quello che è appena successo (una finestra ridimensionata? Un click del mouse? La fine di un effetto sonoro?)[45], e `context` (contesto) che vale 0 se la cosa è accaduta durante l'input di una riga (come nei normali comandi o in un prompt `YesOrNo()` -- ogni volta che il programma deve attendere che il giocatore prema Invio prima di poter rispondere all'input) o 1 se l'evento è avvenuto durante l'input di un carattere (come nei menu, dove il gioco risponde ad ogni pressione di tasto). Ad esempio possiamo gestire la cosa con un routine molto piccola:

```
[ HandleGlkEvent ev context;
   context = 0; ! sopprime gli avvertimenti ignorati
   switch (ev-->0) {
      evtype_Redraw, evtype_Arrange:
         MyRedrawGraphicsWindows ();
   }
];
```

Questo codice si riduce essenzialmente a "se accade qualcosa che richiede il ridisegno delle finestre grafiche, vai avanti a ridisegnarle usando le istruzioni che abbiamo impartito in precedenza".

---

[45] In appendice è presente un approfondimento sul tipo di eventi che si possono generare e gestire in un gioco scritto in inform glulx.

E questo è tutto quello di cui avete bisogno per rendere le vostre finestre grafiche robuste abbastanza da gestire la maggior parte degli avvenimenti che possono accadere.

9.13 Problemi con la grafica

D: Sono abbastanza sicuro di aver fatto tutto correttamente, ma il mio gioco si rifiuta di visualizzare questa JPEG! Non va in crash e non resta appeso, ed il controllo a `gestalt_Graphics` è passato, ma non visualizza l'immagine. Che succede?

R: Il problema potrebbe essere nel file JPEG. JPEG è, in realtà, un nome dato ad un insieme di formati di immagine differenti, e l'implementazione Glk che state usando potrebbe essere in grado di decodificare solo una parte di questi formati. Salvare nuovamente l'immagine JPEG con delle impostazioni differenti, o con un programma grafico diverso, può risolvere la cosa. Se continua a non funzionare, potreste usare PNG al suo posto.

9.14 Disegnate le vostre immagini

Potete fare altro oltre a poter visualizzare immagini PNG e JPEG in una finestra grafica; potete disegnarci sopra. Il comando principale per farlo è `glk_window_fill_rect()`, che richiede sei argomenti:

1) Il nome della finestra nella quale volete disegnare.

2) Il colore che volete il vostro rettangolo assuma. Viene codificato come un numero esadecimale[46], come segue. All'inizio inserite il simbolo del dollaro (che indica che quello che segue è un numero esadecimale). Poi digitate un numero esadecimale a due cifre, da 00 a FF, che indica la quantità di colore rosso da apportare al colore. Poi viene un numero esadecimale a due cifre che indica la quantità di verde, ed, infine, un numero esadecimale a due cifre che indica la quantità di blu. Così, $000000 sarà nero, $FFFFFF sarà bianco, $FF000000 sarà rosso chiaro, $FFC000 sarà un gradevole dorato, $C0C0FF sarà blu chiaro, e così via.

3) La coordinata X dell'angolo in alto a sinistra del rettangolo.

4) La coordinata Y dell'angolo in alto a sinistra del rettangolo.

[46] Vedi nota sugli esadecimali.

5) La larghezza del rettangolo. Se volete disegnare un solo pixel o una riga verticale, deve essere 1.

6) L'altezza del rettangolo. Se volete disegnare un solo pixel o una riga orizzontale, deve essere 1.

E questo è, più o meno, tutto. L'altro trucco che potete usare è quello di riempire una finestra con un certo colore, usando `glk_window_set_background_color()` (che richiede due argomenti, la finestra in questione ed il colore da usare per lo sfondo, codificato come sopra) seguito da un `glk_window_clear()` (che richiede un argomento, la finestra da pulire). Ma se volete disegnare delle immagini più complesse di punti, righe e rettangoli, dovete, ad oggi, ottenerle da punti, righe e rettangoli. Nella maggior parte dei casi, la cosa migliore è quella di usare un programma di grafica per creare delle immagini PNG o JPEG.

9.15 Suono e musica

Il processo di creazione di un canale sonoro, per fargli emettere effetti sonori o musica, e l'uso dei punti d'ingresso per la gestione, dovrebbe apparire molto familiare a chi ha letto la sezione riguardante l'implementazione delle finestre grafiche. A quelli che sono passati direttamente a questa sezione si consiglia vivamente di leggere le sezioni riguardanti la grafica, prima di proseguire, specialmente la parte che riguarda la gestione delle finestre grafiche.

Un'altra lettura obbligatoria è la sezione su Blorb, dal momento che userete Blorb per rendere i file sonori disponibili al vostro gioco, proprio come viene fatto per i file grafici. I formati sonori supportati al momento da Glk e, quindi, da Glulx Inform sono AIFF, MOD e OGG. Il primo è un formato sonoro non compresso, una specie di Microsoft WAV multi piattaforma, che produce file, usando le giuste impostazioni, abbastanza fedeli al suono originale che può essere inciso su un CD. (L'inconveniente è che quando vengono confrontati con un formato lossy come l'MP3, sono abbastanza enormi). MOD è decisamente differente. I file MOD vengono creati usando dei programmi chiamati **tracker**, nei quali caricate alcuni piccoli campioni sonori (un colpo di tamburo, la nota di un pianoforte, un corda di chitarra pizzicata, un cane che abbaia, quello che volete) e poi componete una specie di spartito musicale, inserendo le note su una griglia con il tono appropriato, selezionando il tempo, applicandovi effetti speciali, e così via. Visto che i file non memorizzano l'intero pezzo musicale, ma semplicemente i mattoni per costruirlo e le istruzioni sul come suonarli, i file MOD sono decisamente piccoli. (È anche molto divertente metterli insieme e suonarli. Se state usando una macchina Windows, vi consiglio caldamente Modplug Tracker). **Ogg** è un formato open source per il trasporto di flussi di bit progettato per permettere sia lo streaming che l'archiviazione in maniera

efficiente. Il nome "Ogg" si riferisce al formato di file, che include un numero di codec indipendenti per il video, l'audio ed il testo (ad esempio, per i sottotitoli). I file con l'estensione ".ogg" possono contenere uno qualsiasi dei formati supportati, e poiché il formato è liberamente implementabile, i vari codec ogg sono stati incorporati in molti riproduttori multimediali, sia proprietari, sia liberi. E' facile trovare programmi gratuiti e open source per convertire qualsiasi tipo di formato musicale in ogg. E' un formato estremamente consigliato.

Diciamo che state codificando un ascensore, e decidete che quando il giocatore vi entra, la musica dell'ascensore deve suonare. Il processo da seguire è questo. Per cominciare dobbiamo creare un canale sonoro, e dobbiamo dargli un rock value pari a 410 o superiore (in questo caso scegliamo 410)[47]. Inseriamo le righe seguenti nel programma:

```
Constant GG_MUSICCHAN_ROCK 410;

Global gg_musicchan = 0;
```

E visto che ci siamo, inseriamo una variabile globale per memorizzare la musica che deve essere suonata ad ogni punto determinato:

```
Global current_music = 0;
```

E, naturalmente, ad un certo punto abbiamo modificare il file di risorse Blorb per dare al file sonoro un nome con il quale possiamo usarlo nel nostro codice sorgente. Diciamo che vogliamo farci beffe delle leggi sul diritto d'autore e chiamiamo il nostro file "Muzak". Come indicarlo può variare; se state usando iblorb, la riga potrebbe essere:

```
SOUND Muzak elevmus2.mod
```

Con il lavoro di preparazione terminato, possiamo concentrarci sull'apertura del canale musicale e l'invio della musica allo stesso. L'apertura è facile: basta inserire le righe seguenti in `Initialise()`:

```
if (gg_musicchan == 0) {
  gg_musicchan = glk_schannel_create(GG_MUSICCHAN_ROCK);
}
```

(Personalizzatele per i vostri programmi nella maniera solita). Il canale, adesso, è aperto. Prossimo passo: suonarci la musica. Questa volta il comando chiave è `glk_schannel_play_ext()`, che richiede quattro argomenti:

1) Il nome del canale sonoro. Notate che potete avere diversi canali sonori -- ad esempio, uno per gli effetti speciali, ed uno per la musica -- ma su ogni

[47] Il perché della scelta del numero è approfondibile in appendice.

piattaforma, alla fine incontrerete alcune limitazioni. Suonare più MOD insieme è una prospettiva dall'esito particolarmente dubbio.

2) Il nome del suono da riprodurre (leggete più avanti)

3) Il numero di volte che il suono deve essere ripetuto. Se volete venga ripetuto per sempre fino a quando non gli dite di cessare (o uscite dal gioco), mettete -1.

4) Deve valere 0, a meno che vogliate inviare un evento `evtype_SoundNotify` alla vostra routine `HanldeGlkEvent()` quando termina la riproduzione del suono. Ad esempio, potreste voler implementare una casa infestata con dei suoni paurosi riprodotti a caso. Il codice potrebbe essere:

```
        [ HandleGlkEvent ev context new_sound;
            context = 0; ! sopprime gli avvertimenti ignorati
            switch (ev-->0) {
              evtype_SoundNotify:
              new_sound = random(Ululato, Urlo, Ringhio);
              glk_schannel_play_ext(gg_musicchan, new_sound, 1, 1);
              }
        ];
```

Ma siamo andati un po' troppo oltre. Torniamo indietro.

Come quando disegniamo le immagini in una finestra grafica, non vogliamo riprodurre un commento sonoro direttamente attraverso un canale sonoro, specialmente se deve essere suonato indefinitamente. Se lo facessimo, il giocatore potrebbe far entrare il proprio personaggio nell'ascensore, poi ricaricare un gioco salvato nel punto in cui il personaggio era nel mezzo di una zona di guerra -- e la musica dell'ascensore starebbe ancora suonando! Per evitare questi potenziali errori, usiamo la variabile globale che abbiamo creato in precedenza, e scriviamo la seguente routine:

```
[ MyRestartMusicChannel;
    if (gg_musicchan) {
        if (current_music == 0)
           glk_schannel_stop(gg_musicchan);
        else glk_schannel_play_ext(gg_musicchan, current_music, -1, 0);
      }
];
```

Il codice all'interno delle parentesi graffe riproduce qualunque motivo deve essere suonato al momento, o smette di riprodurre musica nel caso in cui la variabile current_music sia impostata a 0; naturalmente, se non ci sono canali sonori aperti, non viene riprodotto nulla. Ora possiamo creare un oggetto ascensore, con un blocco `before` come questo:

```
before [;
   Enter: current_music = Muzak;
               MyRestartMusicChannel ();
               print "Entri nell'ascensore. La selezione
                      musicale di oggi è:
                      Arrangiamento di corno per xylofono e
                      flauto.^";
            PlayerTo(In_Ascensore);
   ],
```

Ed, alla fine, una routine `IdentifyGlkObject()` per assicurarsi che tutto torni a posto correttamente dopo un restore od un undo. Se ne avete già una, aggiungetevi questo:

```
[ IdentifyGlkObject phase type ref rock res id;
   if (phase == 0) { ! Azzera i riferimenti ai nostri oggetti.
       gg_musicchan = 0;
       return;
   }
   if (phase == 1) { ! niente suoni qui
        return;
   }
   if (phase == 2) {
   ! Itera attraverso tutti i canali esistenti, potrebbe essercene
   ! uno o nessuno, ed identifica i nostri.
         id = glk_schannel_iterate(0, gg_arguments);
         while (id) {
            switch (gg_arguments-->0) {
               GG_MUSICCHAN_ROCK: gg_musicchan = id;
            }
            id = glk_schannel_iterate(id, gg_arguments);
         }
         ! Abbiamo appena cambiato lo stato del gioco, così
         ! dobbiamo cambiare musica o spegnerla del tutto.
         MyRestartMusicChannel ();
      }
];
```

In questo caso il codice in `phase 2` fa più o meno le stesse cose del codice in `phase 1` nella sezione sulle finestre grafiche; la differenza è che un canale sonoro non è né una finestra, né uno stream o un riferimento a file, così la phase 1 non ha bisogno di sapere come deve trattarlo.

9.16 Input del mouse

Le finestre grafiche e quelle a griglia di testo hanno la capacità di riconoscere i click del mouse e restituire le coordinate del pixel o del carattere selezionato. Per usufruire di questa capacità, per prima cosa dovete avvertire il programma di mettersi in ascolto dei click del mouse nella finestra o nelle finestre nelle quali li volete. Diciamo che avete una finestra grafica, `gg_compasswin`, in cui inserite l'immagine di un'elegante bussola ad otto raggi. Volete che il giocatore sia in grado di fare click su qualunque raggio, per averlo convertito nel comando per andare nella determinata direzione. Per prima cosa è necessario inserire la seguente riga in `Initialise()`:

```
glk_request_mouse_event (gg_compasswin);
```

Il gioco si porrà in ascolto di uno, ed uno soltanto, click del mouse nella finestra con la bussola. Se non ce ne sono, va bene così; il programma continua ad accettare le righe di input come al solito. Ma se c'è un click, verrà richiamata `HandleGlkEvent()`.

`HandleGlkEvent()` è stata illustrata nelle sezioni precedenti (consultate Grafica in una finestra grafica, ad esempio, o Suono e musica) ma questa è una spiegazione particolarmente chiara di come la routine funzioni[48]. Come al solito, `HandleGlkEvent()` richiede gli argomenti `ev` e `context`; questa volta `ev`, un array, richiede impegno. Quando il programma intercetta un click del mouse, riempie l'array con le informazioni seguenti:

- `ev-->0`: viene impostato con la costante `evtype_MouseInput`, indicante che è stato il click del mouse (e non, ad esempio, il ridimensionamento di una finestra) a chiamare `HandleGlkEvent()`.

- `ev-->1`: viene impostato con la finestra nella quale è stato fatto il click. Nel nostro esempio sarà sempre `gg_compasswin`.

[48] Un'ulteriore approfondimento è disponibile in appendice.

- `ev-->2`: la coordinata X del pixel su cui si è fatto click. (Se fosse stata una finestra a griglia di testo, sarebbe stata la coordinata X del carattere su cui si è fatto click).

- `ev-->3`: la coordinata Y dello stesso.

La vostra routine `HandleGlkEvent`, sarà quindi (almeno in parte):

```
[ HandleGlkEvent ev context;
   switch (ev-->0) {
      evtype_MouseInput:
         glk_request_mouse_event (gg_compasswin);
         ! Qui dobbiamo inserire il codice che analizza il
         ! risultato e genera le nuove istruzioni. Ad esempio,
         ! potremmo dire che se ev-->2 è compreso tra
         ! 50 e 100, e ev-->3 è tra 0 e 50, allora deve essere
         ! il raggio nord della bussola, e dobbiamo generare il
         ! comando go north (vai a nord).
   }
];
```

Notate che dopo il riconoscimento, e relativa reazione, di un click del mouse, il gioco smette di ascoltare altri input del mouse a meno che gli venga detto esplicitamente di ascoltarne un altro, da qui la seconda chiamata a `glk_request_mouse_event()`.

Ora, visto che il gioco è in attesa di un input da riga, non potete semplicemente generare un comando come `<<Go n_obj>>;` Dovete, invece, fare quello che segue. Prima di tutto tornate alla riga nella quale avete dichiarato `HandleGlkEvent()` e dichiarate tre nuove variabili locali: `abortres`, `newcmd` e `cmdlen`. Poi, tornate nel punto in cui eravate e cancellate la riga di input con:

```
glk_cancel_line_event(gg_mainwin, 0);
```

Quindi arriviamo al comando che faccia la stessa cosa. Nel nostro esempio, "go north" è quello giusto. Impostatelo come il nuovo comando in questo modo:

```
newcmd = "go north";
```

Poi copiate pari pari il seguente codice:

```
cmdlen = PrintAnyToArray(abortres+WORDSIZE,
   INPUT_BUFFER_LEN-WORDSIZE, newcmd);
abortres-->0 = cmdlen;
```

Quello che fa è completare il processo con il quale si dice al computer che in questo caso particolare, il click su quella particolare area della finestra grafica è

perfettamente equivalente all'aver digitato "go north" ed aver premuto Invio. Ora potete inserire quella frase direttamente al prompt, per far sapere al giocatore quello che è stato fatto, o potete fare qualcosa come questo:

```
glk_set_style(style_Input);
print "(mouse click)";
glk_set_style(style_Normal);
new_line;
```

Quale sia il metodo migliore dipende dalle preferenze personali e dal particolare caso considerato. Comunque, l'ultima riga in ognuno di questi blocchi di codice deve essere `return 2;`, indicante il termine di questo turno. Et voilà -- avete avuto a che fare con gli input del mouse.

9.17 Collegamenti ipertestuali

I collegamenti ipertestuali sono immagini o parti di testo che, una volta cliccate, inviano un messaggio a `HandleGlkEvent()` in cui si dice che qualcuno ha fatto click su di loro, consentendovi di eseguire del codice speciale. Potreste, ad esempio, inserire delle note cliccabili nel vostro gioco, o consentire al giocatore di spostarsi facendo click sulle voci dell'elenco delle uscite dalla stanza... ed altre possibilità senza limiti. I collegamenti ipertestuali vengono impostati allo stesso modo degli input del mouse nelle finestre grafiche o a griglia di testo. Prima di tutto dovete avvertire il programma di mettersi in ascolto di un collegamento ipertestuale cliccabile:

```
glk_request_hyperlink_event(gg_mainwin);
```

(Cambiate il nome della finestra con quello nella quale i collegamenti ipertestuali devono apparire, se non è `gg_mainwin`. Se volete i collegamenti in diverse finestre, dovete effettuare questa chiamata diverse volte). Adesso che il programma è pronto a rispondere ad un collegamento, dovete costruire i collegamenti veri e propri. Questo viene fatto con il comando `glk_set_hyperlink()`. La routine viene chiamata con un argomento, un numero univoco diverso da 0. Probabilmente farete delle costanti per i valori dei collegamenti che scegliete: "GO_NORTH_LINK" rende il codice molto più leggibile di "67". Ogni testo o immagine che segue una chiamata a `glk_set_hyperlink()` diventa parte del collegamento, fino a che si raggiunge una chiamata a `glk_set_hyperlink(0);`. (Potete anche andare direttamente da un collegamento all'altro, se volete -- il primo collegamento verrà completato ed il secondo comincerà con la stessa chiamata).

Ora i vostri collegamenti sono pronti, e quello che dovete fare è modificare la routine `HandleGlkEvent()` per rispondere agli stessi quando il giocatore vi fa click sopra. Dovrebbe essere qualcosa come:

```
[ HandleGlkEvent ev context abortres newcmd cmdlen;
   switch (ev-->0) {
      evtype_Hyperlink:
            glk_request_hyperlink_event (gg_mainwin);
            ! Inserite il codice qui, allo stesso modo di
            ! come avete fatto per gli input del mouse.
            ! L'unica differenza è che ora ev-->1
            ! memorizza la finestra sorgente e ev-->2 il
            ! valore del collegamento.
   }
];
```

E proprio come nella sezione precedente, tenete conto del fatto che dopo aver risposto ad un collegamento selezionato, il programma non risponderà a nessun altro a meno che gli venga detto esplicitamente.

9.18 Le pause ed il real time

Sebbene l'IF sia generalmente basata su una sequenza di turni, è possibile -- e, se avete familiarità con gli input del mouse ed i collegamenti ipertestuali, anche abbastanza semplice -- incorporare eventi in real time nel vostro gioco. La cosa può essere utile, ad esempio, per muovere i personaggi tra le stanze mentre il giocatore sta pensando alla prossima mossa da fare, o per implementare una bomba che il personaggio del giocatore deve veramente disinnescare in pochi secondi. Può essere usato anche in congiunzione alla grafica per creare delle animazioni.

Il comando per far partire un evento in real time è `glk_request_timer_events()`, con un numero tra le parentesi che indica il numero di millisecondi che trascorrono tra le chiamate a `HanldeGlkEvent()`. Questo significa che, ad esempio, quando il programma raggiunge la riga `glk_request_timer_events(1000);`, andrà al posto appropriato in `HandleGlkEvent()` ogni 1000 millisecondi -- cioè, ogni secondo -- ed eseguirà il codice che vi troverà. L'inserimento del codice è molto simile a quello che viene fatto per gli input del mouse o i collegamenti ipertestuali -- semplicemente aggiungete alla vostra routine `HandleGlkEvent()` qualcosa di simile a questo:

```
[ HandleGlkEvent ev context;
   switch (ev-->0) {
      evtype_Timer:
         ! qui ci va il codice da eseguire ad ogni intervallo
   }
];
```

La differenza principale dagli eventi del mouse e dei collegamenti ipertestuali è che non dovete effettuare la richiesta di un nuovo evento del timer ogni volta che sono passati N millisecondi: il programma continuerà ad andare a HandleGlkEvent() ripetutamente fino a che non gli dite esplicitamente di smettere con la riga `glk_request_timer_events(0);`.

Allora, come si fanno le animazioni? Il punto chiave consiste nell'approfittare del fatto che se elencate i frame di ogni animazione in sequenza in un file di risorse Blorb, alle costanti che scegliete verranno assegnati dei numeri progressivi. Se avete un'animazione di sette frame, ai vostri frame potrebbero essere assegnati i numeri di risorsa Blorb 3, 4, 5, 6, 7, 8 e 9. Ma non avete bisogno di sapere quali siano i numeri -- dovete, invece, creare una variabile globale che conti i frame, ed inizializzarla con il primo di questi:

```
frame_count = Primo_Frame;
```

Poi inserite un codice come questo nella vostra routine `HandleGlkEvent()`:

```
switch (ev-->0) {
    evtype_Timer:
       if (frame_count > Ultimo_Frame)
             glk_request_timer_events(0);
       else {
          glk_image_draw (gg_moviewin, frame_count, 0, 0);
          frame_count++;
       }
 }
```

Questo codice visualizza prima il frame uno, poi il frame due, quindi il frame tre, ecc., uno dopo l'altro fino al frame sette, e poi si ferma. È compito vostro assicurarvi che la cosa avvenga ad una velocità ragionevole -- provate con numeri di millisecondi differenti per la vostra chiamata a `glk_request_timer_events()` e vedete quello che funziona meglio.

Adesso se volete semplicemente fermare le cose fino a che il giocatore preme un tasto, la cosa è molto più facile. C'è un comando chiamato **`KeyCharPrimitive()`** che legge la pressione di un tasto, se inserite la riga `foo = KeyCharPrimitive();` nel vostro codice, il gioco aspetta che il giocatore prema un tasto, e poi mette il valore del tasto premuto nella variabile `foo`. Un simpatico effetto collaterale è che potete usare `KeyCharPrimitive()` da sola per fermare il flusso di output ed attendere che il giocatore prema un tasto prima di continuare. Ad esempio, diciamo che vogliate inserire una pausa prima di una rivelazione drammatica:

```
print "~Ho preso in considerazione tutte le prove,~ dice
     il detective, ~e tutte portano in una direzione. L'omicidio è
     stato commesso da...";
KeyCharPrimitive();
```

```
print " me! Mi dispiace, non accadrà più.~";
```

9.19 File I/O (Input/Output)

Quasi tutti i sistemi per creare Narrativa Interattiva prevedono la gestione dei file in input/output, anche se solo per creare e leggere il file di salvataggio di una precedente partita. Glulx Inform permette al programmatore di creare e leggere file con qualsiasi tipo di contenuto si desideri. Ad esempio, si potrebbe desiderare scrivere una serie di giochi che hanno come protagonista lo stesso personaggio, e trasferire le informazioni ed i cambiamenti dello stesso da un gioco all'altro. L'implementazione di tale possibilità è piuttosto semplice facendo in modo che il primo episodio della serie crei un file contenente le informazioni necessarie e poi facendo in modo che il secondo episodio vada ad estrarle da detto file. Andiamo a vedere come si fa.

Per creare un file dobbiamo, innanzi tutto, creare un oggetto di riferimento al file **`(fileref)`** così che il programma abbia sempre un modo sicuro di riferirsi al file. Si possono seguire due strade. Se volete specificare il nome del file all'interno del programma del gioco, vi avvarrete del comando **`glk_fileref_create_by_name`**; se invece preferite che sia il giocatore a selezionare il nome del file, dovrete avvalervi del comando **`glk_fileref_create_by_prompt`**. Entrambe le funzioni richiedono tre argomenti.

Il *primo* argomento è l'informazione sul tipo di file che si desidera creare. L'argomento consiste di due costanti, unite dal simbolo di addizione. La prima costante sarà `fileusage_Data` (le alternative qui sono solo usate dalle routine di salvataggio delle partite e similari). La seconda può essere o `fileusage_BinaryMode` o `fileusage_TextMode`; `BinaryMode` è per i file che saranno letti dai giochi fatti in Glulx (come nella registrazione delle qualità di un personaggio nell'esempio precedente) mentre `TextMode` è per i file che possono essere letti dall'utente (una lista di comandi divertenti da provare, o un certificato che attesti la vittoria alla fine della partita o qualunque altra informazione vogliate).

Il *terzo* argomento (no non ne abbiamo saltato uno, lasciamo un attimo da parte il secondo) è il valore `rock` che assegniamo al file da creare. Tale valore può essere considerato come un segnale, una pietra miliare. Tali valori, o pietre se vi piace l'analogia, per il `filerefs` sono gestiti esattamente allo stesso modo di come lo sono per le finestre ed i canali audio. Naturalmente, se vi serve solo creare un file, scrivere informazioni al suo interno, e poi richiuderlo, non avete bisogno della variabile `rock`, dal momento che tali variabili servono per tenere traccia degli oggetti e non può accadere nulla nel breve periodo necessario a compiere tali operazioni che possa far perdere la bussola a Glulx. In tal caso quindi potete assegnare anche il valore 0; se invece decidete di lasciare aperto il

`fileref` per più turni, allora, assegnate un valore alla variabile `rock` come al solito[49].

Il *secondo* argomento dipende dal tipo di file che si desidera creare. Se state creando un file con un nome che avete deciso voi, dovete inserire tale nome come secondo argomento. Tuttavia si presentano un paio di complicazioni. Prima di tutto, volete che il vostro nome di file funzioni su tutte le piattaforme, quindi la scelta più sicura è quella di usare un nome che sia lungo al massimo otto caratteri (dal momento che non tutte le piattaforme gestiscono nomi di file più lunghi). In secondo luogo, non potete semplicemente inserire il nome del file in quel punto, ma dovete inserirlo all'interno di una speciale funzione chiamata, **`ConvertAnyToCString()`**. In tal modo, per creare un file con il nome INFO per contenere i nostri dati, dovremo scrivere una istruzione del genere:

```
fref = glk_fileref_create_by_name (fileusage_Data+fileusage_BinaryMode,
ConvertAnyToCString("INFO"), 0);
```

Se invece decidiamo di permettere al giocatore di scegliere il nome del file, il secondo argomento della funzione conterrà l'informazione indicante quali tipi di file deve creare. Infatti, dal momento in cui questi comandi sono usati per selezionare file da leggere o caricare oltre che per creare file su cui scrivere, vi sono quattro ulteriori opzioni da tenere in conto:

- `filemode_Read`: Il file deve già esistere; al giocatore sarà chiesto di selezionare da un elenco di file già esistenti quale fa al caso suo.

- `filemode_Write`: Il file non deve esistere; se il giocatore seleziona un file esistente, comparirà un avvertimento che lo informerà che il file sarà sostituito.

- `filemode_ReadWrite`: Il file può o non può esistere; se esiste, il giocatore sarà avvisato che esso verrà modificato.

- `filemode_WriteAppend`: ha lo stesso comportamento di `filemode_ReadWrite`. (Utilizzate questa costante per aggiungere informazioni al file, se già esiste).

In questo caso quindi alla fine avremo una istruzione di questo tipo:

```
fref =
glk_fileref_create_by_prompt(fileusage_Data+fileusage_BinaryMode,
filemode_Write, 0);
```

Una volta che avete definito il `fileref`, potete aprire il file.

---

[49] Si veda in appendice per i dettagli sulle convenzioni dei numeri rock da assegnare ai vari tipi di oggetti.

```
str = glk_stream_open_file (fref, filemode_Write, 0);
```

Il secondo argomento è uno dei metodi (filemode) visti sopra. Nel caso abbiate chiesto al giocatore per il nome del file in questa seconda istruzione dovrete usare lo stesso metodo (filemode) che avete scelto prima.

Ora che avete aperto il file, possiamo liberarci del `fileref`:

```
glk_fileref_destroy(fref);
```

Per scrivere, indirizziamo lo stream di output corrente[50] a questo file e usiamo l'istruzione print come al solito:

```
glk_stream_set_current (str);
print "EXPERIENCE POINTS: ", player.exp, "^";
```

Quindi torniamo indietro alla finestra story principale, e chiudiamo il flusso verso il file:

```
glk_set_window(gg_mainwin);
glk_stream_close(str, 0);
```

Il secondo argomento della funzione di chiusura del file `glk_stream_close()`, esattamente come `glk_window_close()`, è di un qualche interesse solo se volete contare quanti caratteri vengono scritti o letti. Se tale informazione non vi interessa ponete tale argomento pari a zero.

Per leggere un file, il procedimento da seguire è praticamente lo stesso, usando però la modalità `filemode_Read`. Le funzioni per leggere i dati sono `glk_get_char_stream()` per leggere un singolo carattere, `glk_get_buffer_stream()` per leggere un array di byte, e `glk_get_line_stream()` per leggere una linea di byte fino ad incontrate il successivo termine della linea (linebreak)[51].

---

[50] Current output stream, ovvero il corrente flusso di dati in uscita.

[51] Consultate gli esempi "Ork" nel prossimo capitolo per maggiori informazioni. Ork 1, infatti, genera un file con informazioni sul carattere del giocatore mentre Ork 2 permette al giocatore, se lo vuole, di caricare in questo secondo gioco il carattere così come è stato determinato in Ork 1, o altrimenti cominciare come nuovo.

§10 Codice in Glulx

Siamo arrivati infine al terzo ed ultimo capitolo dedicato a Gull. Qui potete trovare alcuni esempi di codice per sfruttare le potenzialità multimediali Glk in giochi scritti in Glulx Inform . Gli esempi sono stati realizzati da Adam Cadre e per questo documento portati all'italiano da Marco Falcinelli.

10.1 Esempio di personalizzazione del testo.

In questo esempio vedremo come è possibile stampare del testo sullo schermo impostandone il colore…

```
Constant Story "Caricamento!";
Constant Headline "^una dimostrazione di come si possa personalizzare
                   il testo in Glulx Inform di Adam Cadre^";

Global pct = 94; !percentuale di caricamento del gioco all'inizio
Replace DrawStatusLine;    ! rimpiazziamo la status line
Include "Parser";

Object LibraryMessages
   with
      before [; Score: rtrue; ]; ! nessun messaggio se prendi punti

Include "VerbLib";
Include "Infglk";
Include "Replace";

! la locazione
Object Den "Den"
   with description
        "Sei davanti al tuo nuovissimo computer da 600 cucuzze e sei
        impegnato a ~caricare~ un gioco per computer chiamato ~Lunar
        Lander~. Sarà circa un'ora, ma è quasi fatta!",
   has light;

!il computer nella locazione
Object -> computer "computer"
   with article "tuo",
      name 'computer' 'schermo' 'verdi' 'fosfori',
```

```
      describe "^Lo schermo a fosfori verdi del tuo computer risplende
amichevolmente.",
      description [;
         !(g) permette di "verdificare" il testo
         print (g) "                                ^";
         print (g) "   CARICAMENTO: LUNRLNDR.EXE   ^";
         if (pct < 98) {
            print (g) "   ";
!per rendere verde una variabile però dobbiamo usare la forma estesa:
            glk_set_style(style_User1);
            print pct;
            glk_set_style(style_Normal);
            print (g) "% completato...                ^";
            print (g) "                               ^";
         } else {
            print (g) "  98% completato...$H*#Q&P%%%  ^";
            print (g) "   %                           ^";
            print (g) "   %%% CARICAMENTO FALLITO     ^";
            print (g) "                               ^";
         }
         if (pct == 98) {
            deadflag = 3;
            "^Hmm. Sembra che tu abbia appena gettato il ~monitor~
fuori dalla finestra.";
         }
         rtrue;
      ],
      each_turn [; !ad ogni turno si incrementa pct
         if (pct < 98) pct++;
         if (pct == 98) "^Il computer emette un beep!.";
!non sul serio quello lo faremo in un altro esempio
         rtrue;
      ],
   has static;

[ InitGlkWindow winrock; !qui definiamo gli stili
   switch (winrock) {
      GG_MAINWIN_ROCK:
         glk_stylehint_set(wintype_TextBuffer, style_User1,
            stylehint_TextColor, $00FF00);   ! testo verde
```

```
         glk_stylehint_set(wintype_TextBuffer, style_User1,
            stylehint_BackColor, $000000);   ! se sfondo nero
         glk_stylehint_set(wintype_TextBuffer, style_User1,
            stylehint_Proportional, 0);      !testo proporzionale
   }
   rfalse;
];

[ Initialise;
   location = Den;
   rtrue;
];

! implementiamo una semplicissima status line personalizzata:
[ DrawStatusLine;
   ! se non c'è la finestra per la riga di stato, non dobbiamo crearla
   if (gg_statuswin == 0)
      return;
   ! se il giocatore non è presente, neanche
   if (location == nothing || parent(player) == nothing)
      return;
   glk_set_window(gg_statuswin);      !andiamo alla finestra della
statusline
   glk_window_clear(gg_statuswin);    !puliamola
   glk_window_move_cursor(gg_statuswin, 5, 0);
!spostiamo opportunamente il cursore
   print "Caricamento: ", pct-1, "% completato";
!mettiamoci la percentuale di caricamento del gioco
   glk_set_window(gg_mainwin); !e torniamo alla finestra principale
];

[ DeathMessage; !conclusione
   switch (deadflag) {
      3: print "Non sei riuscito a caricare con successo Lunar Lander";
   }
];

[ g text; !creiamo una funzione che ci semplifichi stampare testo verde
   glk_set_style(style_User1); !testo verde
   print (string) text;
```

```
!stampa il testo successivo alla funzione (g) "..."
   glk_set_style(style_Normal); !testo normale
];

[ PrintRank; rtrue; ];
! per sopprimere frasi inutili quando il gioco termina

Include "ItalianG";
```

. . .

10.2 Come gestire le finestre grafiche

In questo secondo esempio vedremo come mostrare, in una finestra grafica, una immagine della locazione dove si trova il giocatore e come modificarla nei suoi spostamenti.

Cominciamo con il creare un file chiamato bipolar.res nel quale porremo le seguenti istruzioni:

```
CODE C:\IF\GiocoInGlulx\bipolar.ulx
PICTURE Artico_pic artico.jpg
PICTURE Antartico_pic antartico.jpg
```

La prima riga riporta a seguito del comando CODE il percorso assoluto di dove si trova il file del gioco compilato. Mentre le successive righe definiscono due immagini, il primo argomento (ad es. Artico_pic) è il nome con cui potremo riferirci alla risorsa nel nostro file sorgente, il secondo argomento è il percorso relativo (rispetto al file.res) del file che contiene l'immagine. In questo caso stiamo assumendo che l'immagine sia nella stessa cartella del file.res. Compilando tale file con Bres.exe (o con un altro blorbificatore[52]) otterremo un file chiamato bipolar.bli ed uno bipolar.blc. Vediamo ora il sorgente dell'esempio:

```
Constant Story "L'orso Bipolare";
Constant Headline "^una dimostrazione della gestione delle finestre
grafiche in Glulx Inform di Adam Cadre. Traduzione di Marco
Falcinelli.^";


! definiamo due rock value per due finestre una grande ed una piccola
!(ricordiamoci che i valori devono essere diversi)
Constant GG_BIGWIN_ROCK 210;
Constant GG_SMALLWIN_ROCK 211;
```

[52] Se non li è fatto si legga il paragrafo nel capitolo 7 dedicato a Blorb.

```
Replace DrawStatusLine; !no alla statusline di libreria

!le variabili delle due finestre
Global gg_bigwin = 0;
Global gg_smallwin = 0;

!e una per l'immagine che vogliamo visualizzare
Global curr_pic;

Include "Parser";
Include "VerbLib";
Include "Replace";
Include "Infglk";
!includiamo il file.bli del gioco
Include "bipolar.bli";

!locazione
Object Arctic "L'Artico"
   with description
      "Fa dannatamente freddo qui. Temperature più temperate temperano
il freddo a sud.",
      s_to [;!andando verso sud
         curr_pic = Antartico_pic;    !impostiamo l'immagine per il sud
         MyRedrawGraphicsWindows();   !disegnamola
         PlayerTo(Antarctic);         !spostiamo il giocatore a sud
         rtrue;
      ],
   has light;

!locazione
Object Antarctic "L'Antartico"
   with description
      "Whoops! Hai camminato troppo e sei arrivato in Antartico. Quasi
faceva più caldo al nord.",
      n_to [; !verso nord
         curr_pic = Artico_pic;       !impostiamo l'immagine peril nord
         MyRedrawGraphicsWindows();   !disegnamola
         PlayerTo(Arctic);            !spostiamo il giocatore in artico
         rtrue;
      ],
```

```
   has light;

[ Initialise;
   location = Arctic;
   !controlliamo che l'interprete supporti le finestre grafiche
   if (~~glk_gestalt(gestalt_Graphics, 0)) {
   !altrimenti avvisiamo il giocatore
      print "^^^Questo interprete non supporta le finestre grafiche!
Per giocare a questo esempio esse sono necessarie.^";
   }

   ! Creiamo la finestra grande se ancora non esiste
   if (gg_bigwin == 0) {
      gg_bigwin = glk_window_open(gg_mainwin,
         (winmethod_Above+winmethod_Fixed), 279, wintype_Graphics,
         GG_BIGWIN_ROCK);
   }

   ! Creiamo quella piccola dalla grande, ma solo se esiste
   if (gg_smallwin == 0 && gg_bigwin) {
      gg_smallwin = glk_window_open(gg_bigwin,
      (winmethod_Left+winmethod_Fixed), 425, wintype_Graphics,
      GG_SMALLWIN_ROCK);
   }

   !la riga di stato non ci interessa e la chiudiamo
   if (gg_statuswin) {
      glk_window_close(gg_statuswin, 0);
   }
   curr_pic = Arctic_pic;            !Partiamo dall'artico
   MyRedrawGraphicsWindows();
];

!Entry point necessaria per gestire correttamente undo,restore,save
[ IdentifyGlkObject phase type ref rock;
    if (phase == 0) {                ! azzeriamo i nostri oggetti.
        gg_bigwin = 0;
        gg_smallwin = 0;
        return;
    }
```

```
    if (phase == 1) {
        switch (type) {
            0:                              ! se è una finestra
                switch (rock) {
                    GG_BIGWIN_ROCK: gg_bigwin = ref;
                    GG_SMALLWIN_ROCK: gg_smallwin = ref;
                }
            1:                              ! qui vanno i flussi
                                            ! ma non ne abbiamo.
            2:                              ! è un fileref
                                            ! ma non ne abbiamo.
        }
        return;
    }
    if (phase == 2) {
! se abbiamo cambiato lo stato del gioco, e la grafica delle finestre
! probabilmente è sbagiata. Inseriamo il codice per ridisegnarli.
        MyRedrawGraphicsWindows();
    }
];

[ HandleGlkEvent ev context;
    context = 0; ! sopprimiamo gli avvertimenti ignorati
    switch (ev-->0) {
      evtype_Redraw, evtype_Arrange:
          MyRedrawGraphicsWindows();
    }
];

!disegnamo l'immagine nella finestra piccola
[ MyRedrawGraphicsWindows;
   if (gg_smallwin && glk_gestalt(gestalt_Graphics, 0)) {
      glk_image_draw(gg_smallwin, curr_pic, 0, 0);
   }
];

[ DrawStatusLine; ];!niente status line

Include "ItalianG";
```

10.3 Una dimostrazione del real time

Il prossimo esempio mostra come poter implementare il real time in un vostro gioco.

Partiamo con il definire le risorse del nostro gioco, nel file acman.res, dove non troveremo nulla di nuovo rispetto ai precedenti esempi se non la definizione di sette immagini. Esse rappresentano lo stesso soggetto, hanno lo stesso nome con un suffisso numerato saranno utilizzati come fotogrammi per una animazione[53].

```
CODE C:\IF\Glulx\acman.ulx
PICTURE First_Frame Acman\acman1.png
PICTURE Frame_2 Acman\acman2.png
PICTURE Frame_3 Acman\acman3.png
PICTURE Frame_4 Acman\acman4.png
PICTURE Frame_5 Acman\acman5.png
PICTURE Frame_6 Acman\acman6.png
PICTURE Last_Frame Acman\acman7.png
```

Ed ora passiamo al codice sorgente:

```
Constant Story "Acman Maniar";
Constant Headline "^una dimostrazione di realtime in Glulx Inform
                   by Adam Cadre.^Traduzione di Marco Falcinelli.";

!abbiamo bisogno di un unica finestra grafica, quindi un rock value
Constant GG_ACMANWIN_ROCK 210;

!due variabili una per la finestra
Global gg_acmanwin = 0;

!un'altra per contare i fotogrammi
Global acman_count = 0;

Include "Parser";
Include "VerbLib";
```

[53] Notate come le immagini a differenza dell'esempio precedente siano questa volta in una cartella a se stante.

```
Include "Infglk";
Include "Replace";
!il file.bli delle risorse, compilate con Bres.exe:
Include "acman.bli";

!locazione
Object Arcade "Arcade"
   with description
      "Hai dovuto aspettare in linea per ore, ma alla fine è arrivato
il tuo turno. Hai indovinato -- tocca a te giocare ad Acman.",
   has light;

Object -> button "bottone"
   with name 'bottone' 'pulsante',
      describe "^C'è un bottone qui. Premilo per far partire  Acman
         (se non lo hai già fatto.)",
      description "Non c'è altro da fare una volta che è partito.",
      before [;
         Push: if (self has general) "Acman ha lasciato il palazzo.";
            give self general;
            !facciamo partire il timer
            glk_request_timer_events(200);
            "Vai, Acman, vai!";
      ],
   has static;

[ Initialise;
    !controlliamo se l'interprete supporta la grafica:
   if (~~glk_gestalt(gestalt_Graphics, 0)) {
     print "^^^Questo interprete non supporta le finestre grafiche! Per
giocare a questo esempio esse sono necessarie.^";
      KeyCharPrimitive();
               !in caso contrario aspettiamo la pressione di un tasto
      quit;    !prima di uscire
   }

   !ed anche il realtime:
   if (~~glk_gestalt(gestalt_Timer, 0)) {
     print "^^^Questo interprete non supporta il real time! Per giocare
                              a questo esempio ne hai bisogno.^";
```

```
      KeyCharPrimitive();
      quit;
   }

   ! se la finestra non esiste creala
   if (~~gg_acmanwin) {
      gg_acmanwin = glk_window_open(gg_mainwin,
         (winmethod_Above+winmethod_Fixed), 100,
         wintype_Graphics, GG_ACMANWIN_ROCK);
   }

   MyRedrawGraphicsWindows();         !disegna le immagini
   location = Arcade;                 !impostazioni di partenza
   acman_count = First_Frame;
];

[ IdentifyGlkObject phase type ref rock;
   if (phase == 0) {                  ! Azzeriamo gli oggetti
      gg_acmanwin = 0;
      return;
   }

   if (phase == 1) { ! Resettiamo le finestre, i flussi e i filerefs.
      switch (type) {
         0:                   ! se è una finestra
            switch (rock) {
               GG_ACMANWIN_ROCK: gg_acmanwin = ref;
            }
         1:                   ! non abbiamo flussi
         2:                   ! non abbiamo fileref
      }
      return;
   }
   if (phase == 2) { ! aggiorniamo gli oggetti.
      MyRedrawGraphicsWindows();
   }
];
```

```
 [ HandleGlkEvent ev context;
   switch (ev-->0) {          !gestione del realtime
!il tipo di evento da eseguire è la stampa di una immagine
      evtype_Redraw, evtype_Arrange:
         MyRedrawGraphicsWindows();
      evtype_Timer:
!al passare del tempo se il conto supera l'ultimo frame allora fermati
if (acman_count > Last_Frame) glk_request_timer_events(0);
!altrimenti stampa i fotogrammi uno per volta in successione[54]
         else {
            glk_image_draw(gg_acmanwin, acman_count, 0, 0);
            acman_count++;
         }
   }
];

 [ MyRedrawGraphicsWindows;   !disegna le immagini
!se la finestra non esiste non fare nulla.
   if (gg_acmanwin==0) return;
!altrimenti imposta lo sfondo della finestra come bianco
   glk_window_set_background_color(gg_acmanwin, $FFFFFF);
   glk_window_clear(gg_acmanwin);     !e puliscila
   if (acman_count == 0)      !se il conto dei fotogrammi è a 0
                              !mostra il fotogramma di partenza
      glk_image_draw(gg_acmanwin, First_Frame, 0, 0);
   else
               !altrimenti quello di arrivo  dove non c'è acman
      glk_image_draw(gg_acmanwin, Last_Frame, 0, 0);
];

Include "ItalianG";
```

. . .

[54] Il programma sfrutta il fatto che ad ogni immagini è associato un numero 1,2,3,4... quindi stamperà l'immagine associata a tale numero. Si legga il paragrafo 9.18 per maggiori informazioni.

10.4 Una dimostrazione musicale

Andiamo ora a vedere come si gestisce una colonna sonora di una avventura in Glulx. Come al solito cominciamo con il definire il file delle risorse `jukebox.res`:

```
CODE C:\IF\Glulx\jukebox.ulx

SOUND Brano1 audio\brano1.ogg

SOUND Brano2 audio\brano2.ogg

SOUND Brano4 audio\brano4.ogg

SOUND Brano6 audio\brano6.ogg
```

Come potete vedere niente di particolarmente difficile. `SOUND` definisce un file audio e quindi segue il nome con cui chiameremo il file nel sorgente ed il percorso del file. Anche questa volta useremo il blorbificatore[55] per creare il file `jukebox.bli` da importare nel file sorgente del gioco.

```
Constant Story "Da Seppo";
Constant Headline "^una dimostrazione musicale in  Glulx Inform
                   by Adam Cadre^Traduzione di Marco Falcinelli.";

! il valore identificativo del canale audio
Constant GG_MUSICCHAN_ROCK 410;

! la variabile del canale
Global gg_musicchan = 0;

! quale musica suoniamo?
Global current_music = 0;

Include "Parser";
Include "VerbLib";
Include "Infglk";
Include "Replace";
Include "jukebox.bli"; !le nostre risorse musicali

!locazione
Object Diner "Da Seppo"
```

[55] Se non lo avete già fatto leggete il paragrafo dedicato a blorb nel capitolo 7.

```
   with description "Saranno anni che Seppo ha chiuso i battenti del
suo ristorante. Sembra solo ieri che il vecchio Seppo stava dietro i
fornelli -- ogni giorno potevi entrare e si era già scottato un altro
dito, e vi facevate grasse risate... ma poi venne l'incidente della
falciatrice... povero, povero Seppo. Niente più grill e forno, come
tutto il resto... tutto ciò che rimane è un jukebox pieno di musica
techno Finnica, ed anche questo si regge in piedi per miracolo.",
   has light;


Object -> jukebox "jukebox"
   with name 'jukebox',
      number 0,
      describe [;
         new_line;
         switch (self.number) {
            0: "Il jukebox è spento.";
            1: "Il jukebox sta suonando il sinistro ~Brano 1~ di Riku
               Nuottajarvi.";
            2: "Il jukebox sta suonando lo strano ~Brano 2~ di Riku
               Nuottajarvi.";
            3: "Il jukebox sta suonando il vivace ~Brano 4~ di Riku
               Nuottajarvi.";
            4: "Il jukebox sta suonando l'eccitante ~Brano 6~ di Riku
               Nuottajarvi.";
         }
      ],
      description "Il jukebox si regge in piedi per miracolo. Non
accetta più monete; il solo modo per farlo suonare è colpirlo, tipo
Fonzie. E non saprai mai che brano suonerà, sebbene ci siano buone
probabilità che sia un pezzo di Riku Nuottajarvi, l'unico musicista che
piaceva a Seppo.",
      before [x;
         Attack:                              !scelta random
            x = random(5) - 1;
            while (x == self.number) x = random(5) - 1;
            self.number = x;
            switch (self.number) {
               0: current_music = 0;          !nessun brano
                  MyRestartMusicChannel();
                  "Il jukebox si è spento. Speriamo di non averlo rotto
del tutto!";
               1: current_music = Brano1;     !primo brano
                  MyRestartMusicChannel();!suona!
```

```
                    "Il jukebox sta suonando il sinistro ~Brano 1~
                    di Riku Nuottajarvi.";
            2: current_music = Brano2;
               MyRestartMusicChannel();
               "Il jukebox sta suonando lo strano ~Brano 2~ di Riku
               Nuottajarvi.";
        3: current_music = Brano4;
               MyRestartMusicChannel();
               "Il jukebox sta suonando il vivace ~Brano 4~ di Riku
               Nuottajarvi.";
        4: current_music = Brano6;
               MyRestartMusicChannel();
               "Il jukebox sta suonando l'eccitante ~Brano 6~ di Riku
               Nuottajarvi.";
           }
      ],
   has static;

[ Initialise;
   if (~~gg_musicchan)        !se non c'è apri il canale audio
      gg_musicchan = glk_schannel_create(GG_MUSICCHAN_ROCK);
   location = Diner;
];

[ MyRestartMusicChannel;      !suoniamo un disco...
   if (gg_musicchan) {
      if (current_music == 0) glk_schannel_stop(gg_musicchan);
      else glk_schannel_play_ext(gg_musicchan, current_music, -1, 0);
      !facciamo suonare il brano all'infinito a meno di nuovi ordini
   }
];

[ IdentifyGlkObject phase type ref rock res id;
    if (phase == 0) {          ! Azzeriamo gli oggetti.
        gg_musicchan = 0;
        return;
    }
    if (phase == 1) { return;} !niente finestre
    if (phase == 2) {
! controlliamo tutti i canali esistenti -- potrebbe essercene uno
! nessuno o più -- e identifichiamo i nostri.
```

```
        id = glk_schannel_iterate(0, gg_arguments);
        while (id) {
            switch (gg_arguments-->0) {
                GG_MUSICCHAN_ROCK: gg_musicchan = id;
            }
            id = glk_schannel_iterate(id, gg_arguments);
        }
! se cambiamo stato facciamo ripartire la musica che c'era prima
        MyRestartMusicChannel();
    }
];

Include "ItalianG";
```

. . .

10.5 Una dimostrazione degli input del mouse

In questo esempio simuleremo una partita di tris, alla sua mossa decisiva, che naturalmente gestiremo attraverso il mouse. Vediamo il file risorse:

```
        CODE C:\IF\Glulx\ttt.ulx
        PICTURE TTX ttx.png
        PICTURE TTO tto.png
```

Con solo due immagini definite, e quindi il file sorgente:

```
Constant Story "Giochiamo a Tris";
Constant Headline "^una dimostrazione della gestione degli input via
                mouse in Glulx Inform by Adam Cadre^Traduzione di Marco
                Falcinelli^";

! l'id di una finestra glk
Constant GG_TTTWIN_ROCK 210;

! la variabile della finestra
Global gg_tttwin = 0;
```

```
Include "Parser";

Object LibraryMessages

       with

       before [;Score: if (deadflag == 0) "Tu sui gli O. Non hai
       realizzato punti.";else rtrue;];


Include "VerbLib";

Include "Infglk";

Include "Replace";

Include "ttt.bli"; !includiamo un paio di immagini


!la locazione

Object Boardroom "La sala riunioni"

   with description

      "E così siamo arrivati a questo: non hai che due scelte. Metti il
tuo ultimo O nella casella giusta, e vincerai; mettilo nel posto
sbagliato, e .... beh, rabbrividisci solo al pensarci.",

   has light;


Object -> block       !se scegliamo di bloccare l'avversario

   with name 'blocco' 'perdere' 'blocca' 'sconfitta',

      before [;

         Try: glk_image_draw(gg_tttwin, TTO, 207, 105);

              glk_image_draw(gg_tttwin, TTX, 107, 205);

               deadflag = 1;

               "^Decidi di bloccare la casella nel centro destra -- e
               eXecutore allegramente ti risponde scegliendo quella al
               centro in basso, assicurandosi la vittoria. Avendo
               perso, la tua vita non ha più valore. L'eXecutore tira
               fuori un'ascia; è l'ultima cosa che vedi.";

      ],

   has scenery;


Object -> win

   with name 'vittoria' 'vincere' 'vinci',

      before [;

         Try: glk_image_draw(gg_tttwin, TTO, 107, 205);

              deadflag = 2;

               "^Allegramente segni la casella in basso al centro,
               assicurandoti la vittoria. Uno squillo di trombe
               annuncia il tuo trionfo. Huzzah!";

      ],

   has scenery female;
```

```
Object -> invalid
   with name 'invalido',
      before [;Try: "^Non è una casella vuota!";],
   has scenery;


[ Initialise;
    !controlliamo se l'interprete supporta la grafica:
   if (~~glk_gestalt(gestalt_Graphics, 0)) {
        print "^^^Questo interprete non supporta le finestre grafiche!
        Per giocare a questo esempio esse sono necessarie.^";
        KeyCharPrimitive();    !in caso contrario aspettiamo la
                               !pressione di un tasto
        quit;                  !prima di uscire
   }
   if (~~glk_gestalt(gestalt_MouseInput, wintype_Graphics)) {
        print "^^^Questo interprete non supporta gli input del mouse!
        Per giocare a questo esempio esse sono necessarie.^";
        KeyCharPrimitive();    !in caso contrario aspettiamo la
                               !pressione di un tasto
      quit;!prima di uscire
   }
!se non esiste creiamo la finestra per il tris
   if (~~gg_tttwin) {
      gg_tttwin = glk_window_open(gg_mainwin,
         (winmethod_Left+winmethod_Fixed), 300,
         wintype_Graphics, GG_TTTWIN_ROCK);
   }
!avviamo la gestione per l'attesa di un click del mouse
   glk_request_mouse_event(gg_tttwin);
   MyRedrawGraphicsWindows();         ! disegnamo le immagini
   location = Boardroom;
];


[ IdentifyGlkObject phase type ref rock;
   if (phase == 0) {   ! Azzeriamo gli oggetti
      gg_tttwin = 0;
      return;
   }
   if (phase == 1) { ! Resettiamo la finestra, i flussi e i filerefs.
      switch (type) {
```

```
         0:                          ! le finestre
            switch (rock) {
               GG_TTTWIN_ROCK: gg_tttwin = ref;
            }
         1:                          ! non abbiamo flussi
         2:                          ! non abbiamo fileref
      }
      return;
   }
   if (phase == 2) {                 ! Aggiorniamo gli oggetti.
      MyRedrawGraphicsWindows();
   }
];

!ecco la gestione dell'input del mouse:
[ HandleGlkEvent ev context abortres newcmd cmdlen;
   switch (ev-->0) {
      evtype_Redraw, evtype_Arrange:
         MyRedrawGraphicsWindows();
!se la pressione del mouse è nella finestra del tris
      evtype_MouseInput:
         glk_request_mouse_event(gg_tttwin);
 !tra queste coordinate:
         if (ev-->2 >= 200 && ev-->2 < 300
            && ev-->3 >= 100 && ev-->3 < 200) {
            glk_cancel_line_event(gg_mainwin, 0);
!allora stai provando il centrodestra
            newcmd = "try block";
!leggete il paragrafo 9.16 per sapere cosa significa questo comando:
            cmdlen = PrintAnyToArray(abortres+WORDSIZE,
               INPUT_BUFFER_LEN-WORDSIZE, newcmd);
            abortres-->0 = cmdlen;
!avverti che stai eseguendo un click del mouse:
            glk_set_style(style_Input);
            print "(mouse click)";
            glk_set_style(style_Normal);
            new_line;
            return 2;
         }
!oppure  se premi quello al centro in basso:
```

```
         if (ev-->2 >= 100 && ev-->2 < 200
            && ev-->3 >= 200 && ev-->3 < 300) {
            glk_cancel_line_event(gg_mainwin, 0);
            newcmd = "try win";
            cmdlen = PrintAnyToArray(abortres+WORDSIZE,
               INPUT_BUFFER_LEN-WORDSIZE, newcmd);
            abortres-->0 = cmdlen;
            glk_set_style(style_Input);
            print "(mouse click)";
            glk_set_style(style_Normal);
            new_line;
            return 2;
         }
!o ancora in qualsiasi altro posto della finestra del tris:
         else {
            glk_cancel_line_event(gg_mainwin, 0);
            newcmd = "try invalid";
            cmdlen = PrintAnyToArray(abortres+WORDSIZE,
               INPUT_BUFFER_LEN-WORDSIZE, newcmd);
            abortres-->0 = cmdlen;
            glk_set_style(style_Input);
            print "(mouse click)";
            glk_set_style(style_Normal);
            new_line;
            return 2;
         }
   }
];

!questra routine disegna le immagini
[ MyRedrawGraphicsWindows;
   if (gg_tttwin==0) rtrue;
!imposta il colore di sfondo:
   glk_window_set_background_color(gg_tttwin, $000000);
   glk_window_clear(gg_tttwin);
!disegna dei quadrati con le seguenti coordiante:
   glk_window_fill_rect(gg_tttwin, $FFFFFF, 0, 0, 300, 300);
!nero e poi qeulli bianchi:
   glk_window_fill_rect(gg_tttwin, $000000, 99, 4, 2, 292);
   glk_window_fill_rect(gg_tttwin, $000000, 199, 4, 2, 292);
```

```
   glk_window_fill_rect(gg_tttwin, $000000, 4, 99, 292, 2);
   glk_window_fill_rect(gg_tttwin, $000000, 4, 199, 292, 2);
!disegna le immagini   usando le immagini risorse
   glk_image_draw(gg_tttwin, TTX, 107, 5);
   glk_image_draw(gg_tttwin, TTX, 207, 5);
   glk_image_draw(gg_tttwin, TTX, 7, 105);
   glk_image_draw(gg_tttwin, TTX, 107, 105);
   glk_image_draw(gg_tttwin, TTO, 7, 5);
   glk_image_draw(gg_tttwin, TTO, 7, 205);
   glk_image_draw(gg_tttwin, TTO, 207, 205);
];

[ PrintRank; rtrue; ]; ! niente frasi in più alla fine

Include "ItalianG";
[ TrySub; "Non provarci nemmeno."; ];
Verb "try" * noun -> Try;
```

. . .

10.6 Una dimostrazione dei collegamenti ipertestuali

Passiamo ora a vedere concretamente una implementazione della gestione dei collegamenti ipertestuali. In questo esempio faremo in modo di mostrare delle immagini e di manipolarle a seconda della pressione da parte del giocatore sui link ad esse associati. Vediamo il file risorse, che non presenta nulla di nuovo rispetto agli altri esempi.

```
CODE C:\IF\Glulx\monkey.ulx
PICTURE Dorso cardback.jpg
PICTURE Fiori club.jpg
PICTURE Cuori hearts7.jpg
PICTURE Picche spades7.jpg
```

Ed ora andiamo a visionare il file sorgente:

```
Constant Story "La scimmia ed il gioco delle tre carte";
Constant Headline "^una dimostrazione dei collegamenti ipertestuali in
        Glulx Inform by Adam Cadre^Traduzione di Marco Falcinelli.";
```

```
Include "Parser";
Object LibraryMessages
   with before [;Score: rtrue;];

Include "VerbLib";
Include "Infglk";
Include "Replace";
Include "monkey.bli";         ! le immagini delle carte

!locazione
Object House "La case del primate"
   with description
                    "La scimmia fa sembrare il gioco così semplice.
              Due sette e una regina, mischiati tra loro un poco, e
              tutto ciò che devi fare è trovare la signora. Il gorilla
              che era in fila prima di te riusciva trovarla senza mai
              sbagliare, e ha vinto casco di banane. Non vorrai essere
              da meno!?",
   has light;

Object -> cards       ! presentiamo le carte
   with describe [;
      print "^Hai tre carte tra cui scegliere:";
! controlla che la finestra grafica esiste:
      if (glk_gestalt(gestalt_Graphics, 0)) {       !se esiste:
         new_line;
!crea tre collegamenti associati alle immagini
         glk_set_hyperlink(1);                      !inizio link
         glk_image_draw(gg_mainwin, Dorso, imagealign_InlineUp, 0);
         glk_set_hyperlink(0);                      !fine link
         print " ";
         glk_set_hyperlink(2);
         glk_image_draw(gg_mainwin, Dorso, imagealign_InlineUp, 0);
         glk_set_hyperlink(0);
         print " ";
         glk_set_hyperlink(3);
         glk_image_draw(gg_mainwin, Dorso, imagealign_InlineUp, 0);
         glk_set_hyperlink(0);
      }
      else {  ! altrimenti crea tre collegamenti associati al testo
         print " la carta a ";
```

```
            glk_set_hyperlink(1);
            print "sinistra,";
            glk_set_hyperlink(0);
            print " la carta al";
            glk_set_hyperlink(2);
            print "centro";
            glk_set_hyperlink(0);
            print " , e quella a ";
            glk_set_hyperlink(3);
            print "destra";
            glk_set_hyperlink(0);
            ".";
         }
         "^^Fai la tua scelta.";
      ];

   Object -> left "carta a sinistra"
      with name 'sinistra' 'carta',
         before [; !provi quella a sinistra ma è il sette di fiori
            Try: if (glk_gestalt(gestalt_Graphics, 0)) {
                   glk_image_draw(gg_mainwin,Fiori,imagealign_InlineUp,0);
                   print " ";
                   glk_image_draw(gg_mainwin,Dorso,imagealign_InlineUp,0);
                   print " ";
                   glk_image_draw(gg_mainwin,Dorso,imagealign_InlineUp,0);
                   new_line;
                  }
                  deadflag = 3;
                  "^^Il sette di fiori! La scimmia ti ha fregato ancora!";
         ],
      has scenery female;

   Object -> center "carta al centro"
      with name 'centro' 'carta',
         before [;
            Try: if (glk_gestalt(gestalt_Graphics, 0)) {
                   glk_image_draw(gg_mainwin,Dorso,imagealign_InlineUp,0);
                   print " ";
                   glk_image_draw(gg_mainwin,Cuori,imagealign_InlineUp,0);
                   print " ";
```

```
                glk_image_draw(gg_mainwin,Dorso,imagealign_InlineUp,0);
                new_line;
               }
               deadflag = 3;
            "^^Il sette di cuori! La scimmia ti ha fregato ancora!";
      ],
   has scenery female;

Object -> right "carta a destra"
   with name 'destra' 'carta',
      before [;
         Try: if (glk_gestalt(gestalt_Graphics, 0)) {
               glk_image_draw(gg_mainwin,Dorso,imagealign_InlineUp,0);
               print " ";
               glk_image_draw(gg_mainwin,Dorso,imagealign_InlineUp,0);
               print " ";
               glk_image_draw(gg_mainwin,Picche,imagealign_InlineUp,0);
               new_line;
               }
               deadflag = 3;
               "^^Il sette di picche La scimmia ti ha fregato ancora!";
      ],
   has scenery female;

!Avrete capito che non potete vincere....
!Si poteva rendere il tutto più casuale, ma questo è solo un esempio

[ Initialise;
!controlliamo se l'interprete supporta gli hyperlink
   if (~~glk_gestalt(gestalt_Hyperlinks, 0)) {
      print "^^^Questo interprete non supporta gli hyperlink.^";
      KeyCharPrimitive();
      quit;
   }
   glk_request_hyperlink_event(gg_mainwin);  ! avviamo gli hyperlink
   location = House;
];

[ HandleGlkEvent ev context abortres newcmd cmdlen;
   switch (ev-->0) {   ! gestione dei collegamenti
```

```
      evtype_Hyperlink:
         glk_cancel_line_event(gg_mainwin, 0);
         switch (ev-->2) {
                     1: newcmd = "try left";
            2: newcmd = "try center";
            3: newcmd = "try right";
         }
         cmdlen = PrintAnyToArray(abortres+WORDSIZE,
            INPUT_BUFFER_LEN-WORDSIZE, newcmd);
         abortres-->0 = cmdlen;
         new_line;
         return 2;
   }
];

[ DeathMessage; switch (deadflag) {3: print "Hai perso";}];

[ PrintRank; rtrue; ]; ! sopprimiamo i messaggi superflui alla fine

Include "ItalianG";

[ TrySub; "Non provarci nemmeno."; ];
Verb "prova" * noun -> Try;
```

Nel caso in cui la routine `HandleGlkEvent` registri un click su un collegamento ipertestuale (`ev-->2`) allora, a seconda se ha scelto il primo il secondo o il terzo, esegue l'azione try per il corrispondente oggetto carta. Per una spiegazione più dettagliata della precedente routine si rimanda al paragrafo 9.17 e all'approfondimento dedicatogli in appendice.

. . .

10.7 Una dimostrazione dei file I/O

Come ultimo esempio associato a Gull, la sezione di questo volume che vi ha tenuto compagnia dal capitolo sette fino a qui, andiamo a vedere come è possibile gestire i flussi input e output da e verso un file esterno. Gestiremo due sorgenti che potranno essere compilati separatamente, Ork1 e Ork2, dove nel primo registreremo delle informazioni su di un file che poi verrà caricato nel secondo gioco. Per questo esempio non useremo immagini o suoni pertanto non abbiamo bisogno di creare un file delle risorse e possiamo subito concentrarci sui due sorgenti:

```
Constant Story "Ork 1";
Constant Headline "^una dimostrazione di un file I/O in Glulx Inform
                   di Adam Cadre^Traduzione di Marco Falcinelli.^";

!creiamo due variabili di riferimento del file
Global fref;   !una relativa alla modalità (lettura scrittura)
Global str;    !una relativa allo stato del file (Aperto chiuso)

Include "Parser";
Object LibraryMessages with before [; Score: rtrue;];
Include "VerbLib";
Include "Infglk";
Include "Replace";

!creiamo i crediti del gioco come locazione, tanto si vince prima ☺
Object Credits "Ork 1"
   with description
      "^una dimostrazione di un file I/O in Glulx Inform
                   di Adam Cadre^Traduzione di Marco Falcinelli.^",
   has light;

[ Initialise;
   location = Credits;
   print "^^^Hai appena trucidato il drago con un solo colpo della tua
magnifica ascia da battaglia! L'oro del drago è tuo!";
   KeyCharPrimitive();        !aspetta la pressione di un tasto
   print "^^Seleziona un nome per il file su cui salvare il tuo
personaggio e i tuoi possedimenti.";
```

```
!lasciamo sia il giocatore a scegliere il nome del file
   fref =
glk_fileref_create_by_prompt(fileusage_Data+fileusage_BinaryMode,
             filemode_Write, 0);
!apriamo il file
   str = glk_stream_open_file(fref, filemode_Write, 0);
!il riferimento non ci serve più e lo distruggiamo:
   glk_fileref_destroy(fref);
!indirizziamo il flusso di output verso il file:
   glk_stream_set_current(str);
!stampiamo del testo nel file
   print "ORO: 5^";
!torniamo al flusso verso la finestra principale:
   glk_set_window(gg_mainwin);
!chiudiamo il file
   glk_stream_close(str, 0);
   print "^^hai appena completato....";
   deadflag = 2;
   return 2;
];
Include "ItalianG";
```

. . .

Prima di andare a vedere il secondo sorgente compilate il primo Ork e giocateci. Ora aprite il file in cui avete salvato l'oro guadagnato e controllate come Inform abbia svolto il suo lavoro egregiamente. Vediamo ora il secondo episodio della saga, dove caricheremo l'informazione sull'oro precedentemente salvata.

```
Constant Story "Ork 2";
Constant Headline "^una dimostrazione di un file I/O in Glulx Inform
                   di Adam Cadre^Traduzione di Marco Falcinelli.";

Global fref;          !ifileref
Global str;           !stato del file (aperto/chiuso)
Global gold;   !variabile in cui riversare l'informazione sull'oro
Global ch;     !qui andremo a porre ciò che è scritto nel file

Include "Parser";
```

```
Object LibraryMessages  with  before [;Score: rtrue;];


Include "VerbLib";
Include "Infglk";
Include "Replace";


!locazione
Object Showroom "Salone delle armi"
   with description
        "Le Asce di Guerra Usate di Grignr è il negozio dove acquisti
        tutte le tue bardature. Non vedi nessun inserviente, però.
        Forse è un self-serve.",
   has light;


!armi in vendita...
Object -> goodaxe "Ammazza-Draghi 8000"
   with name 'ascia-battaglia' 'battaglia' 'ascia' 'axe' 'ammazza-
draghi' 'ammazzadraghi' 'dragi' 'ammazza' '8000',
      price 8,         !il prezzo dell'arma
      describe "^L'Ammazza-Draghi 8000 cattura il tuo sguardo -- ma
potrebbe avere un prezzo che va al dila delle tue tasche.",
      description "Sembra nuova di zecca. Forse era di proprietà di una
vecchina che la usava solo di Domenica.",
      before [;
         Take, Buy:
            if (gold < self.price) "Non te la puoi permettere.";
            else {
!hey come fa il giocatore ad avere più di 8 pezzi d'oro?
               gold = gold - self.price;
               deadflag = 3;
               "Un inserviente compare appena prendi l'ascia e gli
               consegni i tuoi pezzi d'oro. ~Hai acherato il file dei
               dati eh?~ ti dice. ~Bene, non ti dirò che non lo avresti
               dovuto fare. Divertiti con la tua ascia!";
            }
      ];


Object -> okayaxe "Trincia-Elfi 5000"
    with name 'ascia-battaglia' 'battaglia' 'ascia' 'trincia-elfi'
'trinciaelfi' 'trincia' 'elfi' '5000',
       price 4,       !questo ce lo possiamo permettere
       describe "^Un'altra scelta è il Trincia-Elfi 5000.",
```

```
        description "Sembra abbastanza usata. Forse è per questo che la
        popolazione di elfi sta scemando da queste parti.",
        before [;
          Take, Buy:
             if (gold < self.price) "Non te lo puoi permettere.";
             else {
                gold = gold - self.price;
                deadflag = 3;
                "Un inserviente compare non appena prendi l'ascia e gli
                consegni i tuoi pezzi d'oro. ~Un ottima scelta,~ ti
                dice. ~Si diverta con la sua ascia!~";
             }
      ];

!questo è gratis, mettiamo che non si sia voluto caricare il file...
Object -> stick "bastone"
   with name 'bastone',
      description "Solo un bastone.",
      before [;
         Take, Buy:
            deadflag = 3;
            "Ah, la scelta economica. Bene, mi accontenterò.";
      ];

[ Initialise;
   location = Showroom;
   print "^^^Vuoi caricare un personaggio? >";
   if (YesOrNo()) { !scelta si o no
      print "^^Seleziona il file da caricare.";
!modalità lettura:
      fref =
glk_fileref_create_by_prompt(fileusage_Data+fileusage_BinaryMode,
            filemode_Read, 0);
!apriamo il file:
      str = glk_stream_open_file(fref, filemode_Read, 0);
!chiudiamo il fileref:
      glk_fileref_destroy(fref);
!andiamo a leggere il il flusso che arriva dal file:
      ch = glk_get_char_stream(str);
! '0' corrisponde al codice carettere 48, '1' corrisponde al codice
! carattere 49, etc. '9' al 57; mentre la lettura della stringa non
```

```
! riporta tale carattere continua a leggere
      while (ch < 48 || ch > 57)
                  ch = glk_get_char_stream(str);
! quando incontra un numero tra 0 e 9 si ferma il ciclo e...
! l'oro non è altro che il valore di ch letto in corrispondenza del
! numero e il valore di ch dello 0 :
      gold = ch - 48;
      glk_stream_close(str, 0);      ! chiudiamo il file.
   }
   else gold = 2;    !diamo comunque un paio di pezzi doro al giocatore
   "^E' tempo di un'altra grande avventura per te! Ma prima, bisogna
fare rifornimento.^";
];

[ DeathMessage;
   switch (deadflag) {3: print "Hai acquistato la tua arma.";}
];

Include "ItalianG";
```

. . .

11 Simple Glulx Wrapper *Versione 1.6.1 2006-03-11*

Se avete già letto il capitolo dedicato a GULL, allora questa sezione non sarà nulla di nuovo, dal momento che parla di **sgw.h** una piccola e semplice libreria per Glulx scritta da Alessandro Schillaci. Essa è particolarmente adatta ai principianti o a coloro che non hanno bisogno di implementare nel proprio gioco molti effetti multimediali. La libreria `infglk.h` da la possibilità di sfruttare appieno le possibilità di Glulx ma richiede anche una certa dose di impegno nella costruzione delle routine necessarie a maneggiare gli elementi multimediali. `sgw.h`, si può dire sia una metalibreria dal momento che si basa anch'essa sulla `infglk.h` Rispetto a questa la sgw.h offre all'autore poche funzioni con il vantaggio, però, di essere già completamente implementate. Pertanto con `sgw.h` non vi sarà richiesto di costruire complicate funzioni di controllo su quali finestre sono aperte o meno, ma vi si fornirà la possibilità di inserire alcuni elementi multimediali spesso con una sola istruzione. Naturalmente se si desidera creare qualcosa di più sarà necessario rivolgersi direttamente alla libreria `infglk.h` e leggere approfonditamente i capitoli precedenti e le appendici di approfondimento di questo volume.

11.1 Cosa permette di fare il Simple Glulx Wrapper?

Essenzialmente permette di visualizza un'immagine oppure di riprodurre suoni e una musica di sottofondo. Prevede inoltre la possibilità di intervenire su alcuni stili di testo. Le istruzioni da seguire sono molto semplici, pertanto è facile anche convertire una avventura già scritta in Z-code e convertirla in formato Glulx (con immagini e suoni) semplicemente importando la libreria nel file INF principale e utilizzando le funzioni disponibili, senza doversi curarsi di aspetti tecnici quali ad esempio la sincronizzazione degli elementi multimediali nel caso di un RESTORE della partita in corso o di un RESTART.

11.2 Come installare la libreria sgw

Come primo passo sarà necessario includere la libreria nel vostro sorgente, tale inclusione deve essere posta prima di quella dedicata al parser:

```
Include "sgw.h";
Include "Parser";
Include "infglk";
```

E' poi necessario avviare la libreria attraverso la funzione `initializeSGW(y)` nella funzione `Initialise()` del vostro gioco. Ad Esempio:

```
[Initialise;
        initializeSGW(240);
        location=prima_stanza;
];
```

Questa funzione controlla gli oggetti "Glulx" della libreria e fa il reset, controlla che l'interprete supporti Glulx ed in più creerà una finestra grafica per le vostre immagini con altezza uguale a 240 pixel. Essa sarà in alto sullo schermo al di sotto della status line e sopra la finestra di testo. Tale finestra potrà contenere unicamente immagini. Ovviamente è possibile utilizzare un altro valore come argomento della routine per cambiare le dimensioni della finestra dedicata alle immagini[56].

Se non si vuole sfruttare la possibilità di inserire immagini nel proprio gioco, ma non si vuole rinunciare alle altre caratteristiche di `sgw.h` allora è possibile definire la costante `NOGRAPHICS` prima dell'inclusione della libreria `sgw.h` ed essa ignorerà tutte le immagini e le funzioni ad esse dedicate[57]. Avete a questo punto completato l'installazione della libreria.

11.3 Le risorse del gioco

Se non avete letto GULL, e non sapete cosa è un file Blorb, allora andate a dare un'occhiata al paragrafo dedicato a quest'ultimo ed in particolare a iblorb[58]. Troverete tutte le istruzioni che vi porteranno a creare un unico file contente la vostra avventure e i vari file multimediali ad essa associati. Vi verrà in particolare spiegato come scrivere un file.res quale elenco dei vostri file, la funzione del file.bli e di quello.blc frutto della compilazione del programma iblorb e di come si arrivi a produrre il file.blb ovvero il vostro gioco. E' più semplice di quanto possiate immaginare.

Eccovi di ritorno... ora che avete creato il vostro file di risorse possiamo andare a vedere.

---

[56] Le immagini non saranno ridimensionate a seconda della grandezza della finestra pertanto ponete attenzione alle dimensioni della finestra e delle immagini che vi volete inserire.

[57] Se si erano già inserite delle immagini non è necessario rimuovere le istruzioni ad esse dedicate, la libreria semplicemente non ne terrà conto.

[58] E' sufficiente leggere il funzionamento dei file blorb e la loro costruzione.

11.4 Come gestire la grafica

Glulx supporta due formati di immagini, ma niente paura essi sono tra i più diffusi e tra quelli che meglio rendono su di uno schermo. Avrete la possibilità di usare sia immagini JPG che immagini PNG. Sono considerati degli standard e troverete moltissime informazioni su tali formati in Internet se volete approfondire. Inform e Glulx comunque non prevedono strumenti per la creazione di immagini pertanto dovrete utilizzare uno dei tanti programmi di grafica disponibili per la loro creazione e/o modifica[59]. La libreria sgw.h mette a vostra disposizione tre funzioni per mostrare delle immagini nel vostro gioco, esse potranno essere usate nel vostro gioco per mostrare l'immagine nel momento opportuno nella finestra grafica creata nella routine initialise[60].

```
viewImageLeft(image)
```

che visualizza l'immagine (image) nella finestra dedicata alla grafica ed allineata al bordo sinistro.

```
viewImageCenter(image, image_width)
```

che visualizza l'immagine (image) centrata nello schermo.

```
viewImageRight(image, image_width)
```

che visualizza l'immagine (image) allineata a destra.

Con il termine `image` si intende il nome che avete assegnato alla risorsa nel vostro file.res ad esempio `cane_pic` .

Da notare che per centrare o per allineare a destra un'immagine, è necessario

---

[59] Tali programmi normalmente vi forniranno anche alcune informazioni di cui avrete bisogno come la grandezza in pixel dell'immagine. Per quanto riguarda la definizione dell'immagine stessa espressa in dpi considerate che i monitor raramente superano i 90dpi e comunemente si assestano sui 72dpi. Pertanto non è necessario includere immagini ad altissima risoluzione e molto pesanti.

[60] I vantaggi di avere un finestra grafica sempre in evidenza sono molti, nell'esempio allegato `sgw_test_it.inf` è quello di poter mostrare al giocatore in ogni momento una immagine della locazione in cui si trova il giocatore. Per far si che tale caratteristica sia esaltata l'esempio, come potrete vedere, imposta il `lookmode` a 2 di modo che la descrizione della locazione (dove sono dichiarate le funzioni per la stampa di immagini) venga chiamata ogni volta che è opportuno si stampi anche l'immagine a schermo.

passare un secondo parametro alla funzione `viewImageRight` indicato qui con `image_width`. Questo parametro è la larghezza (in pixel) dell'immagine da centrare o da allineare a destra.

Oltre alle su indicate funzioni che servono per disegnare nuove immagini sullo schermo, avete a disposizione altre tre funzioni per gestire l'aspetto del vostro gioco:

```
clearMainWindow ();
```

che come si intuisce dal nome ripulisce la finestra principale.

```
closeAllWindows ();
```

che chiude la finestra grafica, ovvero quella dove compaiono solo le immagini. Se avete aperto altre finestre grafiche per conto vostro usando le funzioni glk verranno chiuse anch'esse.

```
initializeSGW (h);
```

che imposta l'altezza della finestra grafica ad h pixel.

Sebbene non riportato nelle spiegazioni allegate alla libreria è bene ricordare che è comunque possibile usare le funzioni della libreria `infglk` anche avendo importato `sgw` purchè si approntino gli opportuni accorgimenti. Una funzione grafica che potete usare, come qui di seguito descritto, senza preoccuparvi troppo è

```
glk_image_draw (gg_mainwin,image,align,0);
```

essa vi permette di stampare una immagine non nella finestra grafica, ma in quella principale ovvero in mezzo al testo che legge il giocatore. I quattro argomenti identificano rispettivamente, il nome della finestra principale[61] (gg_mainwin), il nome con cui avete dichiarato l'immagine nel file risorse, l'allineamento con cui si presenterà l'immagine a schermo che potrete impostare ad uno dei seguenti valori: `imagealign_InlineUp`, `imagealign_InlineDown`, `imagealign_InlineCenter`, `imagealign_MarginLeft`, `imagealign_MarginRight`[62].

[61] Non cambiate questo parametro se usate questa funzione assieme alle sgw.h.

[62] Per vedere dei diagrammi su come cambia il layout si faccia riferimento a Gull §9.11.

Infine il quarto parametro della funzione che potrete lasciare posto uguale a 0.

11.5 Aggiungere il sonoro al proprio gioco

Glulx supporta diversi formati audio. Ed in particolare AIFF, MOD ed OGG. Il primo è molto simile ai più conosciuti WAV ed è adatto a riprodurre effetti sonori dal momento che non è un formato compresso e occupa notevole spazio. MOD è un sistema molto particolare che permette di riprodurre brani musicali anche complessi in un file molto piccolo ed è pertanto pensato per la musica, purtroppo non è un gran che supportato e non ci sono molti programmi per la creazione o conversione di tali file ad eccezione di ModTracker per Windows. La scelta consigliata pertanto è il formato OGG, non solo è open source, ma è largamente utilizzato. La qualità di riproduzione e compressione è ottima e può essere usato sia per brani musicali che per effetti sonori. Non farete fatica a reperire su Internet gli strumenti per arricchire con musiche e effetti sonori la vostra avventura.

Questa libreria prevede 3 canali audio da utilizzare nelle vostre avventure:

- **`music`** : è il canale dedicato alla musica
- **`chan1`** : è il canale 1 per i suoni (effetti)
- **`chan2`** : è il canale 2 per i suoni (effetti)

Dal momento che ogni canale può riprodurre un file per volta sarà possibile inserire una colonna sonora al gioco e produrre uno o due effetti sonori contemporaneamente (ma questo accade raramente, normalmente due canali sono più che sufficienti).

La libreria `sgw.h`, per riprodurre i nostri file audio sia che siano brani musicali o effetti sonori, prevede una unica funzione:

```
playSound(channel,sound,lenght,volume);
```

dove nel primo argomento qui indicato con `channel` dovere specificare il valore `music` oppure `chan1` oppure `chan2` a seconda di quale canale intendiate utilizzare per riprodurre il file audio;

al secondo argomento ovvero `sound` dovrete porre il nome identificativo del file audio così come lo avete definito tra le risorse, ad esempio `bau_snd;`

al terzo parametro qui chiamato lenght indicherete il numero di volte che il suono dovrà essere riprodotto, il valore -1 farà si che il file venga riprodotto all'infinito, a meno che non si invii successivamente una nuova istruzione sullo stesso canale.

Infine nel quarto argomento della funzione, `volume`, potrete inserire una delle seguenti tre costanti per settare il volume della riproduzione del file: `VOLUME_HIGH` , `VOLUME_NORMAL` oppure `VOLUME_LOW` .

La libreria prevede inoltre altre tre funzioni per la gestione dei canali sonori:

`silenceAll()`	che azzera tutti i canali audio,
`silenceChannel(channel)`	che azzera il solo canale `channel` audio specificato,
`setVolume(val, channel)`	che infine imposta il valore `val` per il volume del canale specificato. Anche qui si possono usare le tre costanti per il volume.

11.6 Un po' di colore

Questa libreria definisce i colori dei vari tipi di font. Ma è possibile cambiarli nel caso non siano di gradimento. Basta definire le seguenti Costanti prima di includere la libreria sgw (ovvero prima della direttiva `Include "sgw.h";`):

Ecco un esempio:

```
Constant SCBACK $110101;
Constant SCTEXT $DDBB99;
Constant SCSOFT $665544;
Constant SCEMPH $FFFFDD;
Constant SCHEAD $EEDDAA;
Constant SCINPU $DDEEAA;
```

Le istruzioni mostrano sei costanti con al fianco di ognuna un numero esadecimale che corrisponde ad un colore.

Vediamo a quale tipo di testo si riferiscono le costanti:

SCBACK, indica il colore dello sfondo;

SCTEXT, indica i colori per il testo normale e nello stile alert e reverse;

SCSOFT, indica i colori degli stili personalizzati;

SCEMPH, indica i colori per il testo emphasized, bold e header;

SCHEAD, indica i colori per il testo subheader;

SCINPU, indica i colori per il testo preformatted, fixed, note, underline, blockquote e input;

Per i colori è possibile definirli come sopra e la scelta è praticamente infinita o, se si preferisce, sfruttare alcune costanti opportunamente definite dalla libreria per non combattere con gli esadecimali. Vediamole:

ARANCIONE	CLR_GG_ORANGE
AZZURRO	CLR_GG_AZURE
BIANCO	CLR_GG_WHITE
BLU	CLR_GG_BLUE
CIANO	CLR_GG_CYAN
GIALLO	CLR_GG_YELLOW
GRIGIO	CLR_GG_GREY
MAGENTA	CLR_GG_MAGENTA
MARRONE	CLR_GG_BROWN
NERO	CLR_GG_BLACK
ROSA	CLR_GG_PINK
ROSSO	CLR_GG_RED
VIOLA	CLR_GG_PURPLE
VERDE	CLR_GG_GREEN

Pertanto per il nero scrivere CLR_GG_BLACK o $000000 è la medesima cosa. E le due istruzioni qui di seguito sono equivalenti:

```
Constant SCTEXT $000000;
Constant SCTEXT CLR_GG_BLACK;
```

Come fare ora a stampare a schermo il testo con un particolare stile? La libreria prevede delle funzioni di facile utilizzo, basta premetterle tra parentesi al testo su cui si vuole agire come nell'esempio associato alla libreria `sgw_test_it.inf`:

```
print "Testo con lo stile ", (s_emph) "Emphasized", "^";
print "Testo con lo stile ", (s_bold) "Bold", " (come in Inform)^";
print "Testo con lo stile ", (s_pref) "Preformatted", "^";
print "Testo con lo stile ", (s_fixed) "Fixed", " (come in Inform)^";
print "Testo con lo stile ", (s_head) "Header", "^";
print "Testo con lo stile ", (s_subhead) "Subheader", "^";
print "Testo con lo stile ", (s_alert) "Alert", "^";
print "Testo con lo stile ", (s_reverse) "Reverse", " (come in Inform)^";
print "Testo con lo stile ", (s_note) "Note", "^";
print "Testo con lo stile ", (s_underline)"Underline/Italic";
print "Testo con lo stile ", (s_block) "BlockQuote", "^";
print "Testo con lo stile ", (s_input) "Input", "^";
```

Si noti che il testo su cui agisce lo stile è solo quello subito successivo (la prima istruzione print), dal momento ogni funzione dopo l'esecuzione della prima stampa di testo riportano lo stile a quello normale. Nell'esempio pertanto la scritta "come in Inform" non viene formattata[63].

11.7 Altre possibilità?

Al momento la libreria sgw.h non supporta altre caratteristiche delle glk come la possibilità di gestire finestre a griglia di testo, gli eventi generati dal mouse, i collegamenti ipertestuali e i file I/O. D'altra parte la caratteristica di questa libreria è proprio la semplicità e leggerezza.

[63] Si faccia anche attenzione a che `(s_bold) "blabla";` non ritorna `true`, ma è come aver scritto `print "blabla";` ma in grassetto.

APPENDICE A – LA LIBRERIA INFGLK – RIFERIMENTI

La libreria infglk 0.6.1 o anche 0.70 (che implementa solo l'Unicode rispetto alla precedente) permette di usare nei propri sorgenti di Inform le funzioni glk. Nella seguente guida di riferimento[64] sono state escluse quelle variabili, costanti e funzioni che dovrebbero rimanere ad uso esclusivo della libreria.

E' bene ricordare, in questa occasione, che normalmente le funzioni restituiscono valori del tipo 1 = True, 0 = False, GLK_NULL = 0. Quest'ultima è una costante definita da infglk.h, necessaria per evitare conflitti con Inform. Infatti in Inform NULL è pari a -1 mentre in Glulx è pari a 0. Pertanto la libreria infglk invece di riportare NULL (confusione con Inform) riporta GLK_NULL.

Prima di andare a vedere le funzioni messe a disposizione dalla libreria infglk.h andiamo a vedere quali costanti e funzioni sono definite dalla libreria di Inform:

Valori ROCK di libreria:

I seguenti valori sono riservati all'uso della libreria e vanno quindi evitati nella scelta del Rock value per i propri oggetti glk.

```
Constant GG_MAINWIN_ROCK     201;
Constant GG_STATUSWIN_ROCK   202;
Constant GG_QUOTEWIN_ROCK    203;
Constant GG_SAVESTR_ROCK     301;
Constant GG_SCRIPTSTR_ROCK   302;
Constant GG_COMMANDWSTR_ROCK 303;
Constant GG_COMMANDRSTR_ROCK 304;
Constant GG_SCRIPTFREF_ROCK  401;
```

Al fine di evitare confusioni quindi sarà bene scegliere il valore rock per ogni oggetto seguendo il seguente schema: numeri maggiori di 210 per le finestre, numeri maggiori di 310 per i flussi, numeri maggiori di 410 per i Riferimenti ai File.

La libreria di default definisce le seguenti finestre:

```
gg_mainwin     ! finestra di testo principale
gg_statuswin   ! finestra a griglia di testo per la statusline
gg_quotewin    ! finestra di testo per le citazioni
```

[64] Questa guida è stata direttamente tratta tradotta e aggiornata da quella scritta da Marnie Parker aka Doe. `http://members.aol.com/doepage/summary.txt`.

I flussi (stream) di default della libreria sono invece i seguenti:

```
gg_savestr     ! per il flusso di salvataggio del gioco
gg_scriptstr   ! per la trascrizione del gioco ovvero di ciò
che viene stampato a schermo durante una partita
gg_commandstr  ! per il flusso creato dallo script che registra
tutti i comandi impartiti in fase di debug
```

Viene invece definito un solo file di riferimento, ovvero, `gg_scriptfref`, per il file di trascrizione del gioco

Passiamo ora ad elencare **le costanti** e **le funzioni** definite nella libreria **infglk.h**:

Costanti per il controllo Gestalt

Constant gestalt_Version 0;
Constant gestalt_CharInput 1;
Constant gestalt_LineInput 2;
Constant gestalt_CharOutput 3;
Constant gestalt_CharOutput_CannotPrint 0;
Constant gestalt_CharOutput_ApproxPrint 1;
Constant gestalt_CharOutput_ExactPrint 2;
Constant gestalt_MouseInput 4;
Constant gestalt_Timer 5;
Constant gestalt_Graphics 6;
Constant gestalt_DrawImage 7;
Constant gestalt_Sound 8;
Constant gestalt_SoundVolume 9;
Constant gestalt_SoundNotify 10;
Constant gestalt_Hyperlinks 11;
Constant gestalt_HyperlinkInput 12;
Constant gestalt_SoundMusic 13;
Constant gestalt_GraphicsTransparency 14;

Costanti per il tipo di evento

Constant evtype_None 0;
Constant evtype_Timer 1;
Constant evtype_CharInput 2;
Constant evtype_LineInput 3;
Constant evtype_MouseInput 4;
Constant evtype_Arrange 5;
Constant evtype_Redraw 6;
Constant evtype_SoundNotify 7;
Constant evtype_Hyperlink 8;

Costanti per immagini grafiche

Constant imagealign_InlineUp $01;
Constant imagealign_InlineDown $02;
Constant imagealign_InlineCenter $03;
Constant imagealign_MarginLeft $04;
Constant imagealign_MarginRight $05;

Costanti per i tasti

```
Constant keycode_Unknown $ffffffff;
Constant keycode_Left $fffffffe;
Constant keycode_Right $fffffffd;
Constant keycode_Up $fffffffc;
Constant keycode_Down $fffffffb;
Constant keycode_Return $fffffffa;
Constant keycode_Delete $fffffff9;
Constant keycode_Escape $fffffff8;
Constant keycode_Tab $fffffff7;
Constant keycode_PageUp $fffffff6;
Constant keycode_PageDown $fffffff5;
Constant keycode_Home $fffffff4;
Constant keycode_End $fffffff3;
Constant keycode_Func1 $ffffffef;
Constant keycode_Func2 $ffffffee;
Constant keycode_Func3 $ffffffed;
Constant keycode_Func4 $ffffffec;
Constant keycode_Func5 $ffffffeb;
Constant keycode_Func6 $ffffffea;
Constant keycode_Func7 $ffffffe9;
Constant keycode_Func8 $ffffffe8;
Constant keycode_Func9 $ffffffe7;
Constant keycode_Func10 $ffffffe6;
Constant keycode_Func11 $ffffffe5;
Constant keycode_Func12 $ffffffe4;
```

Costanti per gli stili di testo

```
Constant style_Normal 0;
Constant style_Emphasized 1;
Constant style_Preformatted 2;
Constant style_Header 3;
Constant style_Subheader 4;
Constant style_Alert 5;
Constant style_Note 6;
Constant style_BlockQuote 7;
Constant style_Input 8;
Constant style_User1 9;
Constant style_User2 10;
```

Costanti per le finestre

```
Constant wintype_AllTypes 0;
Constant wintype_Pair 1;
Constant wintype_Blank 2;
Constant wintype_TextBuffer 3;
Constant wintype_TextGrid 4;
Constant wintype_Graphics 5;
Constant winmethod_Left $00;
Constant winmethod_Right $01;
Constant winmethod_Above $02;
Constant winmethod_Below $03;
Constant winmethod_Fixed $10;
Constant winmethod_Proportional $20;
```

Costanti per i File I/O	Costanti per gli Hint di testo
Constant fileusage_Data $00;	Constant stylehint_Indentation 0;
Constant fileusage_SavedGame $01;	Constant stylehint_ParaIndentation 1;
Constant fileusage_Transcript $02;	Constant stylehint_Justification 2;
Constant fileusage_InputRecord $03;	Constant stylehint_Size 3;
Constant fileusage_TypeMask $0f;	Constant stylehint_Weight 4;
Constant fileusage_TextMode $100;	Constant stylehint_Oblique 5;
Constant fileusage_BinaryMode $000;	Constant stylehint_Proportional 6;
Constant filemode_Write $01;	Constant stylehint_TextColor 7;
Constant filemode_Read $02;	Constant stylehint_BackColor 8;
Constant filemode_ReadWrite $03;	Constant stylehint_ReverseColor 9;
Constant filemode_WriteAppend $05;	Constant stylehint_just_LeftFlush 0;
Constant seekmode_Start 0;	Constant stylehint_just_LeftRight 1;
Constant seekmode_Current 1;	Constant stylehint_just_Centered 2;
Constant seekmode_End 2;	Constant stylehint_just_RightFlush 3;

Le funzioni della libreria `INFGLK.H`:

Funzioni Principali	Funzioni finestra
`glk_exit`	`glk_window_get_root`
`glk_set_interrupt_handler`	`glk_window_open`
`glk_tick`	`glk_window_close`
	`glk_window_get_size`
Funzioni Gestalt	`glk_window_set_arrangement`
`glk_gestalt`	`glk_window_get_arrangement`
`glk_gestalt_ext`	`glk_window_iterate`
	`glk_window_get_rock`
Funzioni utili	`glk_window_get_type`
`glk_char_to_lower`	`glk_window_get_parent`
`glk_char_to_upper`	`glk_window_get_sibling`
	`glk_window_clear`
	`glk_window_move_cursor`
	`glk_window_get_stream`
	`glk_window_set_echo_stream`
	`glk_window_get_echo_stream`
	`glk_set_window`

Funzioni flusso

```
glk_stream_open_file
glk_stream_open_memory
glk_stream_close
glk_stream_iterate
glk_stream_get_rock
glk_stream_set_position
glk_stream_get_position
glk_stream_set_current
glk_stream_get_current
glk_put_char
glk_put_char_stream
glk_put_string
glk_put_string_stream
glk_put_buffer
glk_put_buffer_stream
glk_set_style
glk_set_style_stream
glk_get_char_stream
glk_get_line_stream
glk_get_buffer_stream
glk_stylehint_set
glk_stylehint_clear
glk_style_distinguish
glk_style_measure
```

Funzioni immagini

```
glk_image_draw
glk_image_draw_scaled
glk_image_get_info
glk_window_flow_break
glk_window_erase_rect
glk_window_fill_rect
glk_window_set_background_color
```

Funzioni File I/O

```
glk_fileref_create_temp
glk_fileref_create_by_name
glk_fileref_create_by_prompt
glk_fileref_create_from_fileref
glk_fileref_destroy
glk_fileref_iterate
glk_fileref_get_rock
glk_fileref_delete_file
glk_fileref_does_file_exist
```

Funzioni eventi

```
glk_select
glk_select_poll
glk_request_timer_events
glk_request_line_event
glk_request_char_event
glk_request_mouse_event
glk_cancel_line_event
glk_cancel_char_event
glk_cancel_mouse_event
```

Funzioni suoni

```
glk_schannel_create
glk_schannel_destroy
glk_schannel_iterate
glk_schannel_get_rock
glk_schannel_play
glk_schannel_play_ext
glk_schannel_stop
glk_schannel_set_volume
glk_sound_load_hint
```

Funzioni hyperlink

```
glk_set_hyperlink
glk_set_hyperlink_stream
glk_request_hyperlink_event
glk_cancel_hyperlink_event
```

Prima di andare a vedere nel dettaglio tutte le funzioni glk disponibili è bene spendere alcune parole su alcune di esse che invece di restituire un unico valore come accade generalmente possono restituire valori multipli stipati in un apposito array. Se queste parole vi suonano strane, passate pure avanti avendo però l'accortezza di tornare qui nel caso incontriate nella guida l'array chiamato **gg_arguments**.

Ogni routine di Inform ha la capacità di restituire esattamente un valore, nella libreria infglk tale valore è generalmente usato per identificare/controllare/avviare un componente glk. Ad esempio:

```
gg_mioCanale = glk_schannel_create(rock);
```

o provvedere ad una condizione true/false per controllare uno stato:

```
if (glk_gestalt(sel, val))
...
```

Alcune funzioni definite nella libreria infglk devono restituire però più di un valore/indirizzo alla routine che le ha chiamate per completare i propri argomenti. Nel seguito di questa guida di riferimento gli argomenti coinvolti in una simile evenienza saranno indicati con il suffisso ptr (puntatore), come ad esempio:

```
glk_window_get_size(win, widthptr, heightptr);
```

Se alcune routine infglk hanno bisogno di restituire **più di un valore** significa che è necessario un approccio differente da quello normale. Il modo in cui una routine può restituire più valori è restituendo un solo array nella quale avrà immagazzinato, a diverse entrate, diversi valori. Se tale operazione riesce, allora il programma che l'ha chiamata potrà accedere ai diversi valori immagazzinati nell'array e la routine è riuscita a restituire più valori contemporaneamente. Un esempio chiarirà meglio il meccanismo:

```
! Esempio di routine che restituisce il quadrato del valore indicato nel primo
! argomento nel suo secondo argomento, e il cubo del numero indicato nel
```

! primo argomento nel suo terzo argomento.

```
[ miaRoutine x x_quadro x_cubo;
    x_quadro-->0 = x * x;
    x_cubo-->0   = x * x * x;
];
```

! Usando un array word con due entrate per ricevere le risposte della routine,

! avremo:

```
array mieiRisultati 2;
miaRoutine(5, mieiRisultati, mieiRisultati+4);
print "Il quadrato è ", mieiRisultati -->0, "^";
print "Il cubo è ",   mieiRisultati -->1, "^";
```

Nel chiamare miaRoutine, il secondo argomento è l'indirizzo che punta al primo termine di entrata dell'array mieiRisultati. La corrispondente istruzione "x_quadro-->0 = x * x;" (secondo argomento della miaRoutine) immagazzina un valore di ritorno in tale entrata dell'array. In modo simile, il terzo argomento, mieiRisultati+4, è l'indirizzo del secondo termine di entrata all'array mieiRisultati (dal momento che in Glulx vi sono quattro byte a termine). L'istruzione "x_cubo-->0=x * x * x;" immagazzina il secondo risultato nella seconda entrata dell'array. Nei fatti, non c'è bisogno di dichiarare il proprio array per ricevere gli argomenti di ritorno dalle routine. Un array, usato per diversi scopi chiamato gg_arguments, è già disponibile nella libreria di Inform per essere usato con tale tipo di routine.

Il primo argomento dell'array può esser indicato semplicemente usando il nome della stessa, ovvero gg_arguments; il secondo elemento scrivendo gg_arguments+4. Se avrete mai bisogno di un terzo o quarto elemento, essi saranno gg_arguments+8 e gg_arguments+12.

Per accedere agli elementi dell'array, è di gran lunga preferibile usare la costante WORDSIZE, in modo che il vostro gioco mantenga tutti i crismi di compatibilità con Glulx e la Z-machine[65]. Pertanto si userà per passare gli argomenti alle entrate di un array una istruzione di questo tipo:

... gg_arguments, gg_arguments+WORDSIZE, gg_arguments+WORDSIZE*2, ...

Ecco come si presenterà quindi una vostra routine personalizzata tipo GetWindowSize() per restituire l'attuale larghezza ed altezza della finestra specificata.

[65] La costante wordsize assume valore 2 per lo z-code e 4 per glulx. L'argomento è trattato diffusamente in questo libro.

```
[ GetWindowSize win widthptr heightptr;
       ! win      finestra da controllare
       ! widthptr   restituisce la larghezza della finestra
       ! heightptr  restituisce l'altezza della finestra.
        if (win == 0) rfalse;
        return glk_window_get_size(win, widthptr, heightptr);
    ];
```

Ed ecco come dovrebbe essere chiamata la routine GetWindowSize() nel proprio gioco per trovare la grandezza della finestra principale:

```
GetWindowSize(gg_mainwin,gg_arguments,gg_arguments+WORDSIZE);
    width  = gg_arguments-->0;
    height = gg_arguments-->1;
```

Qui di seguito riportiamo in ordine alfabetico le succitate funzioni spiegate nel dettaglio.

`glk_cancel_char_event(win);` *Evento*

Cancella la richiesta di input di carattere nella finestra indicata nel parametro (win). E' perfettamente lecito cancellare la richiesta di input di carattere nella data finestra quando non si è in attesa di un evento.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_cancel_char_event(gg_miatext_win);`

`glk_cancel_hyperlink_event(win);` *Hyperlink*

Cancella la richiesta di input di hyperlink nella finestra indicata nel parametro (win). E' perfettamente lecito cancellare la richiesta di una selezione di hyperlink quando non ve ne è in attesa.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_cancel_hyperlink(gg_miaText_win);`

`glk_cancel_line_event(win,event);` *Evento*

Cancella la richiesta di input di linea di comando nella finestra specificata nel primo parametro (win). E' perfettamente lecito cancellare una richiesta di input di linea di comando in una data finestra quando non c'è un evento in attesa.

Il secondo parametro (event) corrisponde al tipo di evento (event type) in attesa. Quando si cancella spesso non è necessario usare le costanti evtype_(specificare). Se pari a GLK_NULL il secondo parametro indica che non interessa il tipo di evento.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: glk_cancel_line_event(gg_miatext_win, GLK_NULL);

Nota: cancellare un evento linea di comando è un'operazione che si consiglia fare in ogni caso all'interno della routine HandleGlkEvent, dove in ogni caso il tipo di evento è automaticamente recuperato.

`glk_cancel_mouse_event(win);` *Evento*

Cancella la richiesta di input di mouse nella finestra indicata nel parametro (win). E' perfettamente legale cancellare una richiesta di input di mouse nella detta finestra quando un evento non è in attesa.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_cancel_mouse_event(gg_mygraph_win);`

`glk_char_to_lower(ch);` *Utilità*

Converte il carattere in minuscolo. L'unico parametro è ch e corrisponde al carattere da trasformare. Restituisce il carattere `ch`.

`glk_char_to_upper(ch);` *Utilità*

Converte il carattere in maiuscolo. L'unico parametro è `ch` e corrisponde al carattere da trasformare. Restituisce il carattere ch.

`glk_exit();` *Principale*

esce dal sistema glk, senza alcun messaggio di avvertimento, sebbene si possa usare anche nel proprio sorgente se ne sconsiglia l'uso. Non ha argomenti e chiaramente non ritorna alcun valore.

glk_fileref_create_by_name(usage,name,rock); *File*

Crea un file con il nome specificato.

Il primo parametro (usage) è costituito di due addendi data_usage+format_usage, il primo indica il tipo di dati da registrare che può essere dati/salvataggi di partita/trascrizioni/registrazione di input. Il secondo se si usa la modalità testo o binaria. Per scegliere l'uso desiderato si individuino le costanti fileusage_(specificare) più opportune.

Il secondo parametro (name) indica il nome del file. Qui è necessario fare attenzione al nome inserito che può essere non letto correttamente su alcune piattaforme. Si suggerisce di non usare nomi più lunghi di otto caratteri, l'estensione viene aggiunta automaticamente dall'interprete a seconda del tipo di file creato.

Il terzo parametro (rock) indica il numero identificativo.

Restituisce il riferimento al file o GLK_NULL in caso di fallimento.

Esempio: `gg_miowrite_fref =`

```
glk_fileref_create_by_name(fileusage_Data+fileusage_BinaryMode,
"PROVA",410);
```

glk_fileref_create_by_prompt(usage,fmode,rock); *File*

Crea un file chiedendo al giocatore di inserire un nome per esso, o digitandolo o selezionandolo da un menu di navigazione aperto dall'interprete.

Il primo parametro (usage) è costituito di due addendi data_usage+format_usage, il primo indica il tipo di dati da registrare che può essere dati/salvataggi di partita/trascrizioni/registrazione di input. Il secondo se si usa la modalità testo o binaria. Per scegliere l'uso desiderato si individuino le costanti fileusage_(specificare) più opportune.

Il secondo (fmode) corrisponde alla modalità di accesso al file (lettura/scrittura – lettura/lettura – scrittura/aggiunta) e usa le costanti definite come Constant filemode_(specificare). Per la lettura (Read) il file deve esistere, per le altre modalità se il file non esiste viene creato: Scrittura (Write) se il file esiste viene riscritto da zero, ScritturaAggiunta (WriteAppend) aggiunge il nuovo flusso alla fine dei dati già registrati sul file.

Il terzo parametro (rock) indica il numero identificativo.

Restituisce il riferimento al file o GLK_NULL in caso di fallimento.

Esempi:

```
gg_mywrite_fref=
glk_fileref_create_by_prompt(fileusage_Data+fileusage_BinaryMod
e, filemode_Write, 410);
```

```
gg_myread_fref=
glk_fileref_create_by_prompt(fileusage_Data+fileusage_BinaryMod
e, filemode_Read, 411);
```

glk_fileref_create_from_fileref(usage,fref,rock);

Copia il riferimento al file esistente, ma cambia il suo uso. Il file originale non viene modificato.

Il primo parametro (usage) è costituito di due addendi data_usage+format_usage, il primo indica il tipo di dati da registrare che può essere dati/salvataggi di partita/trascrizioni/registrazione di input. Il secondo se si usa la modalità testo o binaria. Per scegliere l'uso desiderato si individuino le costanti fileusage_(specificare) più opportune.

Il secondo parametro (fref) indica il riferimento al file mentre il terzo il numero identificativo (rock).

Restituisce il riferimento al file o GLK_NULL in caso di fallimento.

Esempio: `gg_miotrans_fref=`
```
glk_fileref_from_fileref(fileusage_Transcript+fileusage_BinaryM
ode, gg_myread_fref, 414);
```

glk_fileref_create_temp(usage, rock); *File*

Crea un nuovo file temporaneo, che però non deve già esistere. Può essere cancellato automaticamente all'uscita o alla fine del gioco.

Il primo parametro (usage) è pari all'uso_dati+uso_formato:

L'uso dei dati (data usage) può essere dati/salvataggi/trascrizioni/registrazione input, mentre l'uso formato (format usage) può essere la modalità testo o binaria. Si utilizzino le costanti fileusage_(specificare).

Il secondo parametro (rock) è il numero identificativo.

Restituisce il riferimento al file o GLK_NULL in caso di fallimento.

Esempio: `gg_mytemp_fref =`
```
glk_fileref_create_temp(fileusage_Data+fileusage_TextMode,
413);
```

glk_fileref_delete_file(fref); *File*

Al momento cancella il file indicato nel parametro (fref).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_fileref_delete_file(gg_myFile_ref);`

`glk_fileref_destroy(fref);` *File*

Distrugge il riferimento al file (nota: distrugge il riferimento al file non il file stesso), indicato nel primo parametro (fref). I riferimenti ai file sono usati per aprire gli stessi, non per accedere al flusso del file, pertanto è perfettamente lecito distruggere un riferimento al file mentre il file è ancora aperto.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato=glk_fileref_destroy(gg_mioRiferimento_file);`

`glk_fileref_does_file_exist(fref);` *File*

Controlla se il file indicato nel parametro (fref) esiste.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato=glk_fileref_does_file_exist(gg_mioFileRead_ref);
```

`glk_fileref_get_rock(fref);` *File*

Individua il numero identificativo (rock) per il riferimento al file indicato nel primo parametro.

Restituisce il numero rock o GLK_NULL in caso di fallimento.

Esempio: `fref_rock = glk_fileref_get_rock(gg_mioFile_ref);`

`glk_fileref_iterate(fref, rockptr);` *File*

Esegue un ciclo per tutti i riferimenti ai file (ovvero gli oggetti della classe fileref). Il primo parametro (fref) se pari a GLK_NULL restituisce il primo oggetto riferimento a file, altrimenti restituisce fref ovvero il successivo oggetto riferimento a file. Il secondo parametro (rockptr) è un puntatore/indirizzo che sarà riempito con il numero identificativo (rock) appropriato per ogni oggetto fref. Se pari a GLK_NULL indica che non si desidera conoscere i valori rock.

Restituisce il riferimento al file o GLK_NULL quando raggiunge la fine del ciclo.

Esempio:

```
obj = glk_fileref_iterate(GLK_NULL, GLK_NULL);
while (obj) {
    ! fai qualcosa con l'oggetto fref
      obj = glk_fileref_iterate(obj, GLK_NULL);
    }
```

`glk_gestalt(sel, val);` *Gestalt*

Determina la configurazione della libreria e dell'interprete. Il primo parametro individua la caratteristica che deve essere testata e deve corrispondere ad una delle costanti: constant gestalt_(specificare); Il secondo paramento è un valore (normalmente una delle costanti, speso wintype)[66].

Restituisce il valore true (1) o false (0). Cosa può testare?

LA VERSIONE: `glk_gestalt(gestalt_Version, 0);`

Restituisce un numero di versione a 32 bit.

Esempio: ($004E020B = 78.2.11)

LA GRAFICA: `glk_gestalt(gestalt_Graphics, 0);`

Restituisce 1 se sono disponibili le funzioni per la grafica tipo glk_image_draw, etc.

`glk_gestalt(gestalt_DrawImage, wintype);`

Restituisce 1 se le immagini possono essere disegnate nella finestra di quel tipo.

`glk_gestalt(gestalt_GraphicsTransparency,0)`

Restituisce 1 se la grafica trasparente è supportata.

L'AUDIO: `glk_gestalt(gestalt_Sound, 0);`

Restituisce 1 se sono supportate le funzioni audio.

`glk_gestalt(gestalt_SoundVolume, 0);`

Restituisce 1 se le funzioni per impostare il volume funzionano.

`glk_gestalt(gestalt_SoundNotify, 0);`

Restituisce 1 se l'evento notifica del suono è supportato.

`glk_gestalt(gestalt_SoundMusic, 0);`

Restituisce 1 se è capace di eseguire anche risorse musicali di formato MOD. Restituisce 0 se può solo eseguire file in formato AIFF e OGG.

IL MOUSE: `glk_gestalt(gestalt_MouseInput, wintype);`

[66] Si veda anche la tabella al paragrafo 9.4.

Restituisce 1 se la finestra del tipo specificato supporta l'input via mouse.

GLI HYPERLINKS: `glk_gestalt(gestalt_Hyperlinks, 0);`

Restituisce 1 se le funzioni per i collegamenti ipertestuali sono funzionanti.

`glk_gestalt(gestalt_HyperlinkInput,wintype)`

Restituisce 1 se la finestra del tipo indicato supporta gli hyperlink.

IL TIMER: `glk_gestalt(gestalt_Timer, 0);`

Restituisce 1 se gli eventi glk timer sono supportati.

`glk_gestalt_ext(sel, val, arr, arrlen);` *Gestalt*

Questa funzione serve per determinare la configurazione della libreria glk dell'interprete. Può anche essere usata per testare se un particolare carattere può essere stampato. Il primo parametro (sel) è la caratteristica che si vuole testare e corrisponde a una delle costanti Constant gestalt_(specificare). Il secondo parametro è un valore (il carattere di output), mentre il terzo è un array (carattere di output, il puntatore o l'indirizzo che deve essere riempito con un numero di scritte, gg_arguments). Infine il quarto parametro è il numero dei caratteri che deve essere testato (normalmente 1)[67].

Restituisce True (1) o False (0)

Ad esempio:

```
glk_gestalt_ext(gestalt_CharOutput,ch,gg_arguments,1);
```

Può restituire:

`gestalt_CharOutput_CannotPrint` non può essere stampata con significato

`gestalt_CharOutput_ExactPrint` può essere stampata esattamente

`gestalt_CharOutput_ApproxPrint` sarà stampata con approssimazione

glyphs = gg_arguments-->0;

Numero di glyphs che verrà usato per rappresentare il carattere.

---

[67] Nota: i caratteri possono essere da 0 a 255. I valori da 32 a 126 sono caratteri standard di stampa in ASCII. Quelli tra 0 e 31 e tra 127 e 159 sono caratteri di controllo e non hanno un equivalente stampabile. Quando un testo viene stampato in una finestra di testo o in un flusso, glk può stampare qualsiasi carattere Latin-1 in 32-126, in 160-255, e il carattere nuova linea. La nuova versione 0.7.0 può essere estesa ai caratteri UNICODE.

`glk_get_buffer_stream(str, buf, len);` *Flusso*

Legge il numero len di caratteri dal dato flusso nel buffer, a meno che la fine del flusso non sia raggiunta prima. Nessun NULL è piazzato alla fine del buffer.

Il primo parametro (str) è il flusso, il secondo (buf) è l'array buffer e il terzo (len) è il numero di caratteri da leggere.

Restituisce il numero di caratteri realmente letti.

Esempio: `char_count=glk_get_buffer_stream(gg_mioFlusso_scrivi, mio_buffer, mio_buffer_len);`

`glk_get_char_stream(str);` *Flusso*

Legge un carattere dal flusso indicato nel primo parametro.

Restituisce un carattere da 0 a 255, può usare le costanti keycode_(specific).

Esempio: `ch = glk_get_char(gg_mioFlusso_read);`

`glk_get_line_stream(str, buf, len);` *Flusso*

Legge i caratteri da un dato flusso fino a len-1 o fino a quando non raggiunge un new line. Inserisce al termine nel buffer il valore NULL, '\0'. Il primo parametro (str) è il flusso, il secondo (buf) è l'array buffer e il terzo (len) è il numero di caratteri da leggere.

Restituisce il numero di caratteri veramente letti compreso il new line se presente, ma non NULL.

Esempio: `char_count = glk_get_line(gg_mioFlusso_read, mio_buffer+WORDSIZE, mio_buffer_len);`

Nota: `glk_get_line_stream` non mette da parte i primi quattro byte dell'array che si passa (e che indicano la lunghezza della stessa), il che spiega perchè l'indirizzo (secondo parametro) deve essere manipolato. Anche la libreria di Inform alcune volte ricorre a questa funzione con il buffer di input. Si leggano le note anche per `glk_request_line_event`.

`glk_image_draw(win, image, val1, val2);` *Immagine*

Disegna l'immagine indicata nel secondo parametro (image) nella finestra indicata nel primo parametro (win). Il terzo parametro può indicare a seconda che la finestra sia di tipo grafico o a griglia di testo o di testo rispettivamente la coordinata dell'ascissa (pixel) o l'allineamento della immagine (questo avviene attraverso l'uso delle costanti imagealign_(specific)) nelle seconde due. Il terzo

parametro allo stesso modo può indicare la coordinata delle ordinate o semplicemente 0 nel caso della griglia di testo o della finestra di testo.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato=glk_image_draw(gg_miaGraph_win,mia_pic,0,0);
```

`glk_image_draw_scaled(win,image,v1,v2,wid,h);` *Imm*

Disegna l'immagine indicata al secondo parametro (image) in modo che sia scalata alla finestra indicata nel primo parametro (win).

Il terzo parametro (v1) se la finestra è di tipo grafico indica la coordinate delle ascisse altrimenti per gli altri tipi di finestre indica l'allineamento delle immagini (usando le costanti imagealign_(specificare)).

Il quarto parametro (v2) per le finestre grafiche indica la coordinata delle ordinate, altrimenti può essere posto a 0.

Il quinto (wid) e sesto (h) parametro indicano la larghezza e l'altezza dell'immagine.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato = glk_image_draw_scaled(gg_miaGraph_win, mess_pic, 0,
0, 200, 200);
```

`glk_image_get_info(image, widthptr, heightptr);` *Imm*

Individua le informazioni dell'immagine indicata al primo parametro (image).

Il secondo (widthptr) ed il terzo parametro (heightptr) sono due puntatori/indirizzi che saranno riempiti con la larghezza e l'altezza dell'immagine (gg_arguments).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato=glk_image_get_info(mia_pic,gg_arguments,
gg_arguments+WORDSIZE);
            mia_width  = gg_arguments-->0;
            mia_height = gg_arguments-->1;
```

Nota: Sebbene infglk non mostri i suffissi indicanti l'altezza e la larghezza, essi sono puntatori (si vedano le specifiche glk).

`glk_put_buffer(buf, len);` *Flusso*

Stampa blocchi di caratteri nel flusso (stream) corrente, ed è equivalente alle istruzioni:

```
for (i = 0: i < len: i++)
    print (char) buf->(i);
```

Il primo parametro (buf) è l'array buffer, mentre il secondo (len) corrisponde alla lunghezza del blocco da stampare.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato=glk_put_buffer(mio_buffer, mio_buffer_len);`

`glk_put_buffer_stream(str, buf, len);` *Flusso*

Stampa un blocco di caratteri nel flusso (stream) indicato. La funzione è equivalente alle seguenti istruzioni:

```
for (i = 0: i < len: i++)
    print (char) buf->(i);
```

Il primo parametro (str) è il flusso, il secondo (buf) è l'array buffer mentre il terzo (len) è la lunghezza del blocco da stampare.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato= glk_put_buffer_stream(gg_mioFlusso_scrivi,
mio_buffer, mio_buffer_len);
```

`glk_put_char(ch);` *Flusso*

Stampa un carattere nel flusso (stream) corrente. L'unico parametro passato è il carattere che si vuole stampare nel flusso.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_put_char('M');`

`glk_put_char_stream(str, ch);` *Flusso*

Stampa un carattere nel flusso (stream) indicato. Il primo parametro individua il flusso, il secondo specifica il carattere.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_put_char_stream(gg_mioFlusso, 'M');`

`glk_put_string(s);` *Flusso*

Stampa la stringa in fondo al flusso (stream) corrente seguita dal valore NULL.

Il Parametro s è la stringa.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: risultato= glk_put_string("Ciao.");

`glk_put_string_stream(str, s);` *Flusso*

Stampa la stringa in fondo al flusso (stream) indicato al primo parametro (str) seguita dal valore NULL. Il primo parametro individua il flusso il secondo la stringa di testo.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato=glk_put_string_stream(gg_mioFlusso_write, "Ciao.");
```

`glk_request_line_event(win,buf,maxlen,initlen);`

Richiede l'inserimento di un comando di linea nella finestra indicata al primo parametro (win). Il secondo parametro (buf) indica l'array buffer ed il terzo (maxlen) la massima lunghezza in byte della linea di comando. Il quarto parametro (initlen) se diverso da zero verrà inserito all'inizio del buffer come input preesistente, come se il giocatore lo avesse digitato lui stesso..

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: si inserisca la linea in un buffer creato dall'autore.

```
risultato = glk_request_line_event(gg_miaText_win,
mio_buffer, mio_buffer_len, GLK_NULL);
```

Nota: Questa funzione viene normalmente usata dalla libreria nell'input buffer pertanto se l'autore del gioco usa initlen per inserire un comando nell'input buffer come se il giocatore stesso lo avesse digitato, ciò potrebbe portare a un conflitto. Se invece initlen è usato con un buffer creato dall'autore, non ci sono controindicazioni. Si raccomanda pertanto, per evitare problemi, di utilizzare HandleGlkEvent per raggiungere il medesimo risultato, inserendo le seguenti istruzioni[68]:

```
newcmd = "command";
cmdlen = PrintAnyToArray(abortes+WORDSIZE,
         INPUT_BUFFER_LEN-WORDSIZE,newcmd);
```

[68] Si legga l'appendice dedicata all'approfondimento delle entry point.

```
abortres-->0 = cmdlen;
return 2;
```

PrintAnyToArray non mette da parte i primi quattro byte per la lunghezza del buffer, il che spiega la necessità di metter mano all'indirizzo/puntatore. INPUT_BUFFER_LEN è dichiarato anche in parser.h, per varie ragioni, come un buffer più lungo di uno dell'attuale lunghezza, pertanto è necessario metter mano anche a tale indirizzo/puntatore. PrintAnyToArry restituirà l'esatta lunghezza in byte, pertanto può essere usata per trovare la lunghezza di qualsiasi cosa comprese le funzioni.

`glk_request_char_event(win);` *Evento*

Richiede l'input di un carattere nella finestra indicata nel parametro (win).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_request_char_event(gg_miaText_win);`

`glk_request_hyperlink_event(win);` *Evento*

Richiede l'input di un hyperlink nella finestra indicata nel parametro (win).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_request_hyperlink(gg_miaText_win);`

`glk_request_mouse_event(win);` *Evento*

Richiede un input di mouse nella finestra indicata nel parametro (win).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato= glk_request_mouse_event(gg_miaGraph_win);`

`glk_request_timer_events(millisecs);` *Evento*

Richiede un evento temporizzato. A differenza degli eventi di input, gli eventi timer possono essere controllati con `glk_select_poll` o `glk_select`.

L'unico parametro sono i millisecondi che devono passare prima dell'evento.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_request_timer_events(300);`

`glk_schannel_create(rock);` *Suono*

Crea un canale audio. Il parametro indica il valore identificativo (rock) per il canale. Restituisce il canale audio o GLK_NULL in caso di fallimento.

Esempio: `risultato = glk_schannel_create(510);`

`glk_schannel_destroy(chan);` *Suono*

Distrugge il canale audio indicato nel primo parametro.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_schannel_destroy(gg_miocanale);`

`glk_schannel_get_rock(chan);` *Suono*

Individua il numero identificativo (rock) per il canale audio indicato nel parametro (chan).

Restituisce il canale audio o GLK_NULL in caso di fallimento.

Esempio: `chan_rock = glk_schannel_get_rock(gg_miocanale);`

`glk_schannel_iterate(chan, rockptr);` *Suono*

Esegue un ciclo su tutti i canali audio aperti (ovvero gli oggetti glk facenti parte della classe canali audio). Il primo parametro (chan) se pari a GLK_NULL restituisce il primo oggetto canale, altrimenti restituisce `chan` ovvero l'oggetto canale audio successivo. Il secondo parametro (rockptr) è un puntatore/indirizzo che viene riempito per ogni canale audio dal corrispettivo numero identificativo rock (gg_arguments). Se pari a GLK_NULL indica che non interessano i numeri rock.

Restituisce il canale audio o GLK_NULL quando termina il ciclo.

Esempio:
```
obj = glk_schannel_iterate(GLK_NULL, GLK_NULL);
    while (obj) {
        ! fai qualcosa con l'oggetto canale
        obj = glk_schannel_iterate(obj, GLK_NULL);
    }
```

`glk_schannel_play(chan, snd);` *Suono*

Fa partire l'esecuzione del file audio indicato nel secondo parametro (snd) nel canale audio indicato nel primo (chan).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato = glk_schannel_play(gg_miocanale, mormorio_snd);
```

`glk_schannel_play_ext(chan,snd,repeats,notify);`

Fa partire l'esecuzione di un file audio indicato nel secondo parametro (snd) nel canale audio indicato nel primo parametro (chan). Il terzo parametro (repeats) indica il numero di volte che deve essere ripetuta l'esecuzione del file audio (con 0 neanche una volta, con -1 all'infinito). Il quarto parametro (notify) se diverso da zero richiede che l'evento notifica si avvii nel momento in cui la esecuzione del file termina.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato = glk_schannel_play_ext(gg_mychan, babble_snd, 2, 1);
```

`glk_schannel_set_volume(chan, vol);` *Suono*

Imposta il volume del canale indicato al primo parametro con effetto immediato ed al valore indicato nel secondo parametro (vol) in valori decimali/esadecimali.

Il volume parte al massimo 65536 ($10000), tre quarti di volume sono espressi come 49152 ($C000), metà volume corrisponde a 32768 ($8000).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato = glk_schannel_set_volume(gg_miocanale, $C000);
```

`glk_schannel_stop(chan);` *Suono*

Termina l'esecuzione di qualsiasi file sul canale indicato nel parametro (chan).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_schannel_stop(gg_mychan);`

`glk_select(event);` *Principale*

Tutta l'interfaccia utente è realizzata attraverso funzioni glk_select, la funzione glk_main chiama glk_select e aspetta per un evento, quindi lo restituisce in una struttura o ferma il gioco fino al momento in cui arrivano gli eventi. I programmatori di Inform dovrebbero usare HandleGlkEvent piuttosto che questa funzione che è rivolta più ai programmatori di interpreti.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

`glk_select_poll(event);` *Principale*

Sonda (e trova) se qualche evento è in attesa, questa funzione è principalmente usata dai programmatori di interpreti a cui si raccomanda di non usarla spesso. I programmatori di Inform dovrebbero limitare l'uso di questa funzione agli eventi legati al tempo e solo se vi sono molte operazioni che si eseguono contemporaneamente. Il parametro è un puntatore di struttura che viene riempito con le informazioni sull'evento (gg_event).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

`glk_set_hyperlink(linkval);` *Hyperlink*

Attiva o disattiva i collegamenti ipertestuali nella finestra corrente. Il parametro (linkval) è un valore che se diverso da zero indica che il testo stampato successivamente sia un hyperlink attivo (normalmente sottolineati), ogni hyperlink dovrebbe avere un numero unico in questo parametro. Un valore invece pari a 0 disattiva i collegamenti ipertestuali.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
glk_set_hyperlink(1);
print "Ecco una prova.";
glk_set_hyperlink(0);
print "E tra poco....";
glk_set_hyperlink(2);
print " un'altra prova.";
glk_set_hyperlink(0);
```

`glk_set_hyperlink_stream(str,linkval);` *Hyperlink*

Attiva o disattiva i collegamenti ipertestuali nel flusso (stream) indicato nel primo parametro (str). Il secondo parametro (linkval) è un valore che se diverso da zero indica che il testo stampato successivamente sia un hyperlink attivo (normalmente sottolineati), ogni hyperlink dovrebbe avere un numero unico in questo parametro. Un valore invece pari a 0 disattiva i collegamenti ipertestuali.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:
```
glk_set_hyperlink_stream(gg_mioFlusso_scrivi, 1);
print "Questa è una prova.";
glk_set_hyperlink_stream(gg_mioFlusso_scrivi, 0);
```

`glk_set_interrupt_handler(func);` *Principale*

Imposta il gestore di interruzioni glk (interruzioni hardware), viene chiamata dalla funzione glk_main e non dovrebbe essere usata dai programmatori di Inform. Il parametro (func) è un puntatore/indirizzo della funzione di gestione delle interruzioni, che non ha argomenti e non restituisce alcun risultato.

`glk_set_style(styl);` *Stile*

Imposta lo stile di testo indicato. Il parametro (styl) è lo stile che si vuole impostare, a tal fine bisogna usare le costanti definite come Constant style_(specificare)

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:
```
risultato = glk_set_style_stream(style_Emphasized);
```

`glk_set_style_stream(str, styl);` *Stile*

Imposta lo stile di testo indicato per il flusso (stream) indicato. Il primo parametro è il flusso mentre il secondo è lo stile da impostare usando le cosanti style_(specificare).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:
```
risultato =
glk_set_style_stream(gg_mioFlusso_scrivi, style_Emphasized);
```

`glk_set_window(win);` *Finestra*

Indirizza il flusso (stream) corrente alla finestra indicata nel parametro, esattamente allo stesso modo di glk_stream_set_current (scambi di date finestre). Qualsiasi comando di stampa successivo verrà indirizzato alla finestra corrente.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato=glk_set_window(gg_miaFinestraDiTesto);`

`glk_sound_load_hint(snd, flag);` *Suono*

Suggerisce se uno specifico file audio debba essere caricato. Il primo parametro (snd) è il suono. Il secondo è una flag che può essere o True (1) o False (0).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_sound_load_hint(mormorio_snd, 1);`

`glk_stream_close(str, result);` *Flusso*

Chiude il flusso specificato nel primo parametro. Il secondo parametro (result) è un puntatore alla "struttura" che contiene il conteggio finale dei caratteri

(gg_arguments, viene passato solo gg_argument, e non gg_arguments+4).

gg_arguments-->0 == conto in lettura

gg_arguments-->1 == conto in scrittura

GLK_NULL == il conto dei caratteri non è richiesto

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Se il flusso è al momento un flusso in uscita (output), il flusso è impostato a GLK_NULL.

Esempio:

```
risultato=glk_stream_close(gg_mioFlussoDiTesto, gg_arguments);
            read_count  = gg_arguments-->0;
            write_count = gg_arguments-->1;
```

`glk_stream_get_current();` *Flusso*

Individua il flusso (stream) corrente. Non ha parametri e restituisce il flusso o GLK_NULL se non ne trova nessuno.

Esempio: `curr_str = glk_stream_get_current();`

`glk_stream_get_position(str);` *Flusso*

Individua la posizione corrente (segno) di lettura/scrittura in un dato flusso (stream), indicato nel primo parametro.

Restituisce la posizione del punto (segno) che corrisponde:

Nella Memoria e nei File Binari all'esatto numero di byte letti o scritti dall'inizio del flusso. Nei File di Testo è ambiguo, dal momento che si può essere sicuri solamente che pos aumenterà dopo la scrittura e lettura.

Esempio:

```
curr_pos = glk_stream_get_position(gg_mioFlusso_scrivi);
```

`glk_stream_get_rock str` *Flusso*

Individua il numero identificativo rock di un dato flusso (stream) (quello indicato al primo parametro).

Restituisce il numero identificativo rock o GLK_NULL in caso di fallimento.

Esempio: `str_rock = glk_stream_get_rock(gg_mioFlusso_write);`

`glk_stream_iterate (str, rockptr);` *Flusso*

Esegue un ciclo su tutti i flussi (stream) aperti (ovvero gli oggetti glk della classe stream). Il primo parametro può essere una variabile per restituire il successivo flusso o GLK_NULL per restituire il primo oggetto flusso. Il secondo parametro (rockptr) è una variabile indicatrice/puntatore che andrà ad essere riempita con ognuno dei numeri identificativi rock dei flussi (gg_arguments), se pari a GLK_NULL indica che non si desidera conoscere i numeri rock.

Restituisce il flusso o GLK_NULL quando raggiunge la fine del ciclo.

Esempio:

```
obj = glk_stream_iterate(GLK_NULL, GLK_NULL);
while (obj){
        ! fai qualcosa con l'oggetto flusso.
        obj = glk_stream_iterate(obj, GLK_NULL);
}
```

`glk_stream_open_file (fileref, fmode, rock);` *Flusso*

Apre un nuovo flusso (stream) che legge o scrive su di un file. Il primo parametro (fileref) è il riferimento al file. Il secondo (fmode) corrisponde alla modalità di accesso al file (lettura/scrittura – lettura/lettura – scrittura/aggiunta) e usa le costanti definite come Constant filemode_(specificare). Per la lettura (Read) il file deve esistere, per le altre modalità se il file non esiste viene creato:

Scrittura (Write) se il file esiste viene riscritto da zero, ScritturaAggiunta (WriteAppend) aggiunge il nuovo flusso alla fine dei dati già registrati sul file. Infine l'ultimo parametro assegna un numero identificativo (rock) all'oggetto.

La funzione restituisce il flusso o GLK_NULL in caso di fallimento.

Esempio

```
gg_MioFlusso_scrivi =
glk_stream_open_file(gg_MioFlusso_scrivi_ref,filemode_Write,310
);
gg_MioFlusso_leggi =
glk_stream_open_file(gg_MioFlusso_leggi_ref,filemode_Read,311);
```

glk_stream_open_memory(buf,buflen,fmode,rock);

Apre un nuovo flusso (stream) che legge/scrive in uno spazio in memoria. Il primo parametro (buf) è un array buffer dove l'output sarà letto/scritto. Il secondo parametro (buflen) indica la lunghezza del buffer, se vengono scritti più caratteri nel buffer di quanti indicati in buflen (outputting) allora l'eccesso di questi verrà tagliato via e non vi sarà alcuna sovrascrittura nel buffer. Se invece vengono letti più caratteri dal flusso di quanti specificati in buflen (inputting), allora il flusso risponderà restituendo -1 (fine del file). Il terzo parametro (fmode) deve essere una costante tra quelle definite come Constant fmode_(specificare) ed indicare la modalità lettura/scrittura/lettura-scrittura del file. Infine il quarto parametro indica un numero identificativo dell'oggetto flusso (rock).

Restituisce il flusso o GLK_NULL in caso di fallimento.

Esempio:

```
gg_miobufferout_str=glk_stream_open_memory(mio_buffer,
mio_buffer_len,filemode_Write, 312);


gg_miobufferin_str = glk_stream_open_memory(mio_buffer,
mio_buffer_len, filemode_Read, 313);
```

glk_stream_set_current(str); *Flusso*

Imposta il flusso (stream) corrente al flusso indicato, deve essere un flusso in uscita (output) (e corrisponde ad uno scambio tra flussi dati). Il parametro indica il flusso. Restituisce True (1) in caso di successo e GLK_NULL se il flusso corrente è impostato su nothing.

Esempio:

```
risultato=glk_stream_set_current(gg_mioFlusso_scrivi);
```

`glk_stream_set_position(str, pos, seekmode);` *Flusso*

Imposta il punto (segno) da cui cominciare a leggere/scrivere nel flusso (stream) indicato. Il primo parametro individua il flusso, il secondo la posizione che a seconda del seekmode può essere:

Start → pos è il numero di caratteri cominciando a contare dall'inizio del file,

Current → pos è il numero di caratteri successivi alla posizione corrente,

End → pos è il numero di caratteri dopo la fine del file,

Nei file Binari pos corrisponde esattamente al numero di caratteri letti/scritti,

nei file di testo pos può variare, a causa della conversione delle linee e dell'impostazione dei caratteri, pertanto è più sicuro usare solamente un punto di riferimento dall'inizio o dalla fine del file o da una posizione ricavata con la funzione get_position.

L'ultimo parametro è il seekmode e può essere pari a start/current/end usando le rispettive costanti come definite in Constant seekmode_(specificare).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato=glk_stream_set_position(gg_mioFlusso_write,
0,seekmode_Start);
```

`glk_style_distinguish(win, styl1, styl2);` *Stile*

Controlla se i due stili sono visivamente distinguibili nella finestra indicata.

Il primo parametro (win) individua la finestra, il secondo (styl1) la categoria del primo stile attraverso le costanti style_(specificare) ed il terzo parametro (style2) la categoria del secondo stile di testo sempre attraverso le costanti style_(specificare).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato = glk_style_distinguish(gg_mytext_win, style_Normal,
style_Emphasized);
```

`glk_style_measure(win, styl, hint, result);` *Stile*

Determina gli attributi dello stile indicato nella data finestra.

Il primo parametro (win) individua la finestra, il secondo (styl) la categoria dello stile attraverso le costanti style_(specificare). Il terzo parametro (hint) l'attributo di stile (hint) usando le costanti stylehint_(specificare). Infine il quarto

parametro (result) è una variabile puntatore/indirizzo che sarà riempita con il risultato (gg_arguments). Essa varia a seconda del suggerimento di stile testato:

Indentation/ParaIndentation – misura dell'indentazione,

Justification – giustificazione, che può essere just_LeftFlush, just_LeftRight,etc.

Size – ovvero la grandezza dei font,

Weight – spessore del font (1=grassetto, 0=normale,-1 = fino),

Oblique – corsivo (1) o normale (0),

Proportional – proporzionale (1) o larghezza fissa (0),

TextColor – numero decimale/esadecimale del colore del carattere,

Background – numero decimale/esadecimale del colore di sfondo,

ReverseColor – impostato a 1 se i colori sono invertiti, a 0 se normali.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato = glk_style_measure(gg_mytext_win, style_Normal, stylehint_Oblique, gg_arguments);
normal_oblique = gg_arguments-->0;
```

`glk_stylehint_clear(wintype, styl, hint);` *Stile*

Rimuove i suggerimenti di stile per la categoria di stile indicata.

Il primo parametro (wintype) indica il tipo di finestra, attraverso le costanti wintype_(specificare) e può essere anche usata la costante wintype_AllTypes per avere effetto su tutti i tipi di finestra.

Il secondo parametro (styl) è la categoria di stile che vogliamo ripulire dai suggerimenti di stile, ed è individuata attraverso l'uso delle costanti style_(specificare).

Il terzo ed ultimo parametro (hint) è il suggerimento di stile da togliere, identificato con le costanti stylehint_(specificare).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempi: `risultato = glk_stylehint_clear(wintype_TextBuffer, style_Note, stylehint_Proportional);`

`risultato = glk_stylehint_clear(wintype_TextBuffer, style_Note, stylehint_just_FlushLeft);`

`glk_stylehint_set(wintype, styl, hint, val);` *Stile*

Suggerisce il suggerimento di stile (style hint) indicato per la data categoria di stile. Ha effetto solo sulle finestre create successivamente. I parametri sono:

Primo parametro è (wintype), il tipo di finestra, specificato usando le costanti wintype_(specificare). Può essere usata anche la costante wintype_AllTypes per avere effetto su tutte le finestre.

Secondo parametro è (styl) che suggerisce su quale categoria di stile deve avere effetto la funzione, attraverso l'uso delle costanti style_(specificare).

Terzo parametro è (hint), ovvero il suggerimento di stile attraverso l'indicazione di una delle costanti stylehint_(specificare).

Il quarto ed ultimo parametro è (val), ovvero il valore dello specifico hint. Varia a seconda che l'impostazione del suggerimento di stile sia impostato su:

Indentation/ParaIndentation – misura dell'indentazione,

Justification – giustificazione, che può essere just_LeftFlush, just_LeftRight,etc.

Size – ovvero la grandezza dei font,

Weight – spessore del font (1=grassetto, 0=normale,-1 = fino),

Oblique – corsivo (1) o normale (0),

Proportional – proporzionale (1) o larghezza fissa (0),

TextColor – numero decimale/esadecimale del colore del testo,

Background – numero decimale/esadecimale del colore di sfondo,

ReverseColor – impostato a 1 se i colori sono invertiti, a 0 se normali.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempi:

Suggeriamo che il testo normale sia bianco su sfondo nero:

```
glk_stylehint_set(wintype_TextBuffer, style_Normal,
stylehint_BackColor, $000000);

glk_stylehint_set(wintype_TextBuffer, style_Normal,
stylehint_TextColor, $ffffff);
```

Suggeriamo che il testo delle note usi un font proporzionale con giustificazione a sinistra:

```
glk_stylehint_set(wintype_TextBuffer, style_Note,
stylehint_Proportional, 0);

glk_stylehint_set(wintype_TextBuffer, style_Note,
stylehint_just_FlushLeft, 0);
```

`glk_tick();` *Principale*

Questa funzione restituisce temporaneamente il controllo al sistema operativo per consentirgli di effettuare le sue operazioni (in un sistema operativo con un multitasking cooperativo – come windows 3.11 o il vecchio Mac) – è

un'operazione indispensabile, perché altrimenti non sono in grado di riprendere il controllo.

Non ha argomenti e chiaramente non ritorna alcun valore.

`glk_window_clear(win);` *Finestra*

Cancella il contenuto della finestra indicata nel primo parametro. Se tale finestra è una finestra di testo può cancellare tutto il testo presente o stampare delle linee bianche etc… se invece è una finestra a griglia di testo tutte le caselle della griglia vengono riempite di spazi bianchi e il cursore viene posizionato alle coordinate 0,0. Infine per una finestra grafica essa viene ripulita con il corrente colore di sfondo.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato = glk_window_clear(gg_miaFinestra);`

`glk_window_close(win, result);` *Finestra*

Chiudere la finestra indicata. Il primo parametro individua la finestra da chiudere, mentre il secondo il puntatore o indirizzo da riempire con il conto dei caratteri in output (gg_arguments) o GLK_NULL se non si desidera sapere quanti caratteri in output ci sono.

Restituisce True (1) in caso di successo o altrimenti False (0) in caso di fallimento.

Esempio:

```
glk_window_close(gg_mytext_win, gg_arguments);
      char_count = gg_arguments-->0;
```

`glk_window_erase_rect(win,left,top,width,height);`

Cancella nella finestra indicata al primo parametro (win) una area rettangolare (ovvero la pulisce usando il corrente colore di sfondo).

I parametri successivi al primo indicano i valori necessari per disegnare il rettangolo: left = coordinata lato sinistro del rettangolo; top = coordinata lato superiore del rettangolo; width = larghezza del rettangolo; height = altezza del rettangolo.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato = glk_window_erase_rect(gg_Graph_win, 0, 5, 50, 50);
```

`glk_window_fill_rect(win,color,left,top,width,h);`

Riempie una area rettangolare nella finestra grafica indicata nel primo parametro (win) con il colore indicato nel secondo parametro (color) attraverso un numero decimale/esadecimale. I parametri successivi indicano i valori necessari per disegnare il rettangolo: left = coordinata lato sinistro del rettangolo; top = coordinata lato superiore del rettangolo; width = larghezza del rettangolo; h = altezza del rettangolo.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
risultato=glk_window_fill_rect(gg_Graph_win,$ff0000,0,5,50,50);
```

`glk_window_flow_break(win);` *Finestra*

Inserisce, nella finestra indicata al primo parametro (win), una nuova linea di testo sotto una immagine allineata al margine, trova il numero di nuove linee che sono necessarie (a seconda del font in uso) per inserire testo sotto l'immagine. Essenzialmente inserisce un segno invisibile nel testo.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

`glk_window_get_arrangement(wi,metpt,sizept,keywpt);`

Individua il cambiamento della finestra indicata. Il primo parametro (wi) indica la finestra, il secondo (metpt) è un puntatore/indirizzo che sarà riempito con il metodo adottato dal cambiamento mentre il terzo parametro (sizept) con la grandezza del cambiamento. L'ultimo parametro (keywpt) sarà sempre un puntatore/indirizzo che sarà riempito con il keywin (vedi funzione `glk_window_set_arrangement`) (gg_arguments).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:

```
glk_window_get_arrangement(gg_miaFinestraGrafica,
gg_arguments, gg_arguments+WORDSIZE,
gg_arguments+(WORDSIZE*2))
miaFinestra_metodo == gg_arguments-->0;
miaFinestra_grandezza  == gg_arguments-->1;
miaFinestra_keywin == gg_arguments-->2;
```

`glk_window_get_echo_stream (win);` *Finestra*

Individua il flusso eco per la finestra indicata nel parametro e lo restituisce. Se la data finestra non ha al momento della chiamata alla funzione un flusso eco, allora restituisce il valore GLK_NULL (il flusso eco non esiste).

Esempio:

```
gg_mioEco_str=glk_window_get_echo_stream(gg_mainwin);
```

`glk_window_get_parent(win);` *Finestra*

Individua la finestra genitrice (parent) di quella indicata nel primo argomento.

Restituisce la finestra o GLK_NULL in caso di fallimento.

Esempio:

```
parent_finestra =
glk_window_get_parent(gg_miaFinestra);
```

`glk_window_get_rock(win);` *Finestra*

Individua il numero identificativo (rock) della finestra indicata al primo ed unico parametro.

Restituisce il numero rock o GLK_NULL in caso di fallimento.

Esempio: Grafica_rock = glk_window_get_rock(gg_miaFinestraGrafica);

`glk_window_get_root();` *Finestra*

Individua la root dell'albero delle finestre (la finestra originale). Non ha parametri. Restituisce la finestra root o GLK_NULL (niente)

Esempio

```
root = glk_window_get_root();
```

`glk_window_get_sibling(win);` *Finestra*

Individua le finestre sorelle (sibling) di quella indicata al primo parametro.

Restituisce la finestra o GLK_NULL se non vi sono finestre sorelle.

Esempio:

```
finestra_sorella =
glk_window_get_sibling(gg_miaFinestra);
```

`glk_window_get_size(win,widthptr,heightptr);`

Individua la grandezza della data finestra. Viene chiamata con tre parametri: il primo (win) è il nome della finestra che si vuole misurare, il secondo è una

variabile che verrà riempita con la misura della larghezza della finestra mentre il terzo parametro misurerà l'altezza (gg_arguments).

Restituisce True (1) se ha successo e False (0) in caso di fallimento.

Esempio:

```
glk_window_get_size(gg_miaFinestraGrafica,
gg_arguments,gg_arguments+WORDSIZE);

FinestraGrafica_larghezza == gg_arguments-->0;

FinestraGrafica_altezza == gg_arguments-->1;
```

`glk_window_get_stream(win);` *Finestra*

Individua il flusso (stream) associato alla finestra indicata nel parametro win.

Restituisce il flusso o GLK_NULL in caso di fallimento.

Esempio: `gg_mioFlusso_testo = glk_window_get_stream(gg_miaFinestraTesto);`

`glk_window_get_type(win);` *Finestra*

Individua il tipo della finestra indicata al primo parametro.

Restituisce il tipo di finestra (wintype) o GLK_NULL in caso di fallimento.

Esempio `tipo_fin = glk_window_get_type(gg_miaFinestraGrafica);`

`glk_window_iterate(win, rockptr);` *Finestra*

Esegue un ciclo su tutte le finestre aperte (ovvero gli oggetti glk della classe finestre). Ha due parametri, il primo può assumere i valori GLK_NULL che fa restituire alla funzione il primo oggetto finestra oppure WIN che restituisce la successiva finestra dopo questa. Il secondo parametro rockptr è un puntatore/indirizzo che sarà riempito con ognuno dei numeri identificativi (rock) corrispondenti a ciascuna finestra (gg_arguments). In caso assuma il valore GLK_NULL allora indica che non si desidera conoscere i numeri rock.

Restituisce la finestra o GLK_NULL quando raggiunge la fine del ciclo.

Esempio:

```
obj = glk_window_iterate(GLK_NULL, GLK_NULL);

while (obj) {

      ! fai qualcosa con l'oggetto finestra
      obj = glk_window_iterate(obj, GLK_NULL);

}
```

`glk_window_move_cursor(win, xpos, ypos);` *Finestra*

Posiziona il cursore nella finestra indicata al primo parametro ed alle coordinate indicate nel secondo e terzo parametro. (0, 0) sono le coordinate dell'angolo superiore a sinistra.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio

```
risultato=glk_window_move_cursor(gg_miaFinGriglia,0,0);
```

`glk_window_open(split,method,size,wintype,rock);`

Apre una nuova finestra. Il primo argomento individua la finestra originaria da cui deve essere ricavata la nuova. Il secondo argomento è il metodo con il quale deve essere ricavata la nuova finestra. Il metodo è composto da due fattori uniti dal segno di addizione. Il primo fattore è la direzione di ritaglio della nuova finestra (sopra/sotto/destra/sinistra), il secondo il metodo per stabilire la grandezza della finestra (proporzionale/fisso). A tal fine si utilizzano le costanti Constant winmethod_(specificare). Il terzo argomento indica il tipo di finestra che si vuole creare (di testo, a griglia di testo, grafica). Infine il quarto argomento indica il numero identificativo della nuova finestra (rock value).

Restituisce: la finestra o GLK_NULL (aperture della finestra fallita)

Ad esempio, se si volesse aprire una finestra a griglia di testo di misura proporzionale in percentuale in alto rispetto la finestra principale gg_mainwin, useremmo i seguenti argomenti (finestra principale, ritaglio verso l'alto, grandezza in proporzione all'altezza, numero di linee di testo, tipo di finestra, valore identificativo).

```
gg_mytext_win =

glk_window_open(gg_mainwin,winmethod_Above+winmethod_Pro
portional, 30, wintype_TextBuffer, 210);
```

Se invece volessimo aprirla di una grandezza fissa (con altezza precisata in numero di pixel) allora avremmo:

```
gg_mygraph_win =

glk_window_open(gg_mainwin,winmethod_Above+winmethod_Fix
ed,200, wintype_Graphics, 210);
```

`glk_window_set_arrangement(win,method,size,keywin);`

Questa funzione permette di cambiare la configurazione della finestra indicata.

Il primo parametro indica la finestra da configurare, il secondo il metodo di ritaglio nella direzione (sopra/sotto/destra/sinistra) e misura (fissa o proporzionale) usando le costanti winmethod_(specificare). Il terzo parametro indica la misura (dell'altezza se la direzione è sopra/sotto o della larghezza se è

destra/sinistra) ed è espressa in linee, colonne o pixel a seconda del metodo scelto. L'ultimo parametro è keywin ed indica la prima finestra delle due da ritagliare o GLK_NULL che lascia la finestra chiave immutata.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:
```
glk_window_set_arrangement(gg_miaFinestraGrafica,
        winmethod_Below+winmethod_Fixed, 200, 0);
```

`glk_window_set_background_color(win,color);` *Finestra*

Imposta il colore di sfondo indicato nel secondo parametro (color) attraverso un numero decimale/esadecimale nella finestra indicata nel primo parametro (win). Questa istruzione ha effetto solo in conseguenza di una pulizia o ridimensionamento della finestra stessa.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:
```
risult=glk_window_set_background_color(gg_mygraph_win,$C0C0C0)
```

`glk_window_set_echo_stream(win,str);` *Finestra*

Crea un eco del flusso diretto alla finestra, ovvero lo duplica. Tutto ciò che è diretto alla finestra viene registrato in un flusso detto eco. E' possibile anche scrivere direttamente nel flusso eco, ma non ci si deve aspettare che il flusso diretto all'eco vada anche alla finestra. E' a senso unico: flusso→finestra→eco.

Il flusso può essere di qualsiasi tipo, anche un flusso di un'altra finestra.

Il primo parametro è la finestra da monitorare, il secondo o è una variabile str che contiene il flusso eco o GLK_NULL che serve a fermare la copia eco del flusso.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio:
```
risult=glk_window_set_echo_stream(gg_mainwin,gg_mioEcho_str);
```

. . .

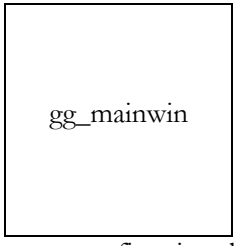
APPENDICE B – RELAZIONI TRA LE FINESTRE GLK CREATE

La procedura per creare nuove finestre è ben spiegata nel relativo paragrafo di Gull, il testo di Adam Cadre. Ma qui ci occuperemo di specificare per bene cosa accade nel momento in cui si crea una nuova finestra e come questa si colloca nell'albero degli oggetti glk. Posto che la finestra principale viene definita dalla libreria con il nome gg_mainwin, una nuova finestra non si crea dal nulla ma sempre dalla divisione di una già esistente. La finestra che si crea pertanto è figlia (child) di quella da cui l'abbiamo separata; e quest'ultima è la genitrice (parent) della nuova finestra. E' una buona idea aprire la propria finestra principale nella routine Initialise del sorgente del gioco in modo da controllare che l'interprete sia in grado di gestire tale finestra non appena parte.

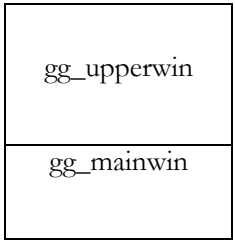
Una finestra può essere ricavata ritagliando l'area a destra/sinistra/sopra/sotto della sua finestra parent. Se la nuova finestra viene tagliata in senso verticale da quella vecchia la sua grandezza viene determinata dalla larghezza (che sarà il parametro che passeremo alla funzione) mentre se viene tagliata in senso orizzontale la propria grandezza sarà ricavata dall'altezza. I parametri possono essere dati sia in valore assoluto (pixel) per le finestre grafiche o in linee e colonne per le finestre di testo, che proporzionalmente in percentuale per quelle grafiche.

Nel momento in cui decidete di chiudere una finestra lo spazio da essa occupato tornerà alla sua genitrice. Occorre fare attenzione a chiudere le finestre figlie prima di chiudere le loro genitrici.

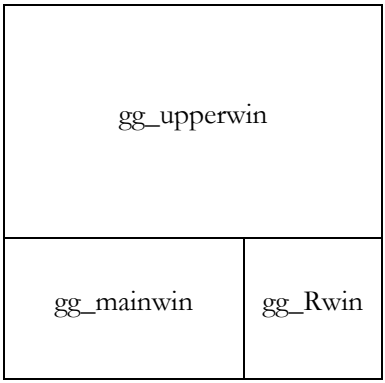
Vediamo con un aiuto visuale il processo di creazione delle finestre:



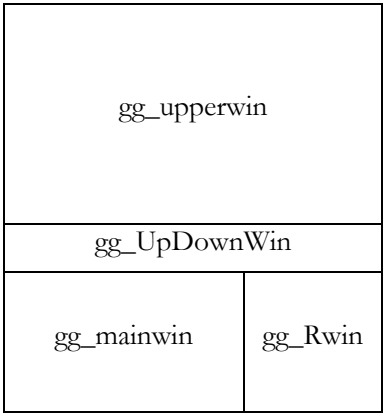
Da questa creiamo una finestra grafica in alto sullo schermo, chiamata gg_upperwin. gg_mainwin risulterà essere la parent di gg_upperwin.



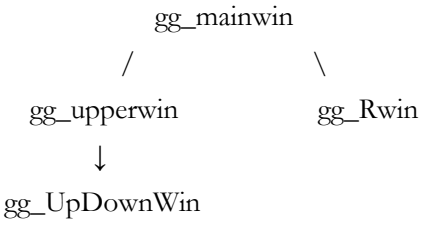
Ora possiamo dividere ancora la gg_mainwin, ma anche la gg_upperwin, sia in senso verticale che orizzontale. Seguiamo questo secondo approccio e dividiamo la gg_mainwin ancora ma in senso verticale.



Ora la situazione è che la gg_mainwin è la genitrice di due nuove finestre che sono tra loro sorelle (sibling). Dividiamo ora la gg_upperwin in due finestre in senso orizzontale e creiamo la nuova finestra gg_UpDownWin.



Ora la situazione è che la gg_mainwin è genitrice di gg_Rwin e di gg_upperwin. Quest'ultima è a sua volta genitrice di gg_UpDownWin. L'albero degli oggetti pertanto sarà:



APPENDICE C – LE ENTRY POINT DI GLULX

`InitGlkWindow()`

La funzione `InitGlkWindow()` viene chiamata dalla libreria subito dopo l'inizio del gioco, prima della funzione `Initialise` e dopo la funzione `IdentifyGlkObject()`. La libreria crea tre finestre principali: `gg_mainwin`, `gg_statuswin` e `gg_quotewin`. E' possibile definire la `InitGlkWindow()` nel proprio codice in modo da cambiare le impostazioni di queste finestre di default, gli stili di testo predefiniti ed i colori. Si ricorda che gli hint di stile hanno effetto solo sul buffer di testo delle finestre che sono aperte successivamente, come l'impostazione dei colori di sfondo delle finestre grafiche ha effetto solo dopo un successivo evento di pulizia della finestra (clear) o ridimensionamento della stessa.

Se si include una propria `InitGlkWindow()` allora sara necessario provvedere a definire la `gg_mainwin`, altrimenti il gioco verrà terminato.

Si tenga però presente che se si vuole sostituire semplicemente la riga di status, per personalizzarla a proprio piacimento, allora non sarà necessario usare l'entrypoint in oggetto, ma come nel classico Inform basterà porre l'istruzione `Replace DrawStatusLine;` nel proprio codice subito dopo aver incluso `parser.h` e prima di includere `verblib.h`. E quindi poi definire la propria routine `DrawStatusLine()`.

. . .

`IdentifyGlkObject()`

Nel momento si decide di creare un qualsiasi oggetto glk[69] per il proprio gioco, allora si rende **necessario** inserire anche una propria routine `IdentifyGlkObject()` perché essi possano essere correttamente identificati dopo un comando tipo UNDO, RICOMINCIA o CARICA. Dal momento che la routine della libreria non può sapere dell'esistenza degli oggetti da voi creati e non li identificherà[70] se non provvederete a comunicarglielo attraverso questa entry point.

---

[69] Al di fuori degli oggetti glk gestiti dalla libreria, come la finestra principale.

[70] Un riferimento è una variabile globale che punta ad un oggetto glk. Quando un oggetto glk viene aperto o creato il suo riferimento viene riempito con un numero che punta all'indirizzo dell'oggetto nella memoria (semplificando molto, in realtà esso punta ad un ingresso in una hashing table o qualcosa del genere). Dopo un comando tipo undo,carica,ricomincia i riferimenti

La funzione `IdentifyGlkObject()` viene chiamata dalla libreria in due momenti: all'inizio del gioco, prima di `Initialise` e `InitGlkWindow()`, e dopo i comandi UNDO, RICOMINCIA e CARICA.

`IdentifyGlkObject()` viene chiamata una volta per ognuna delle seguenti tre fasi (phase):

* Fase zero: Questa fase dovrebbe ripulire tutto, impostando ogni variabile che punti ad un oggetto glk a zero. Ciò si rende necessario perché le variabili recuperate da una precedente posizione potrebbero essere fuorvianti, magari dicendo al sistema, ad esempio, che una finestra è aperta mentre invece è chiusa[71].

* Fase uno: Questa fase dialoga con alcuni degli oggetti glk quali le finestre, i riferimenti ai file ed i flussi. Esegue un ciclo su tutti questi oggetti nella memoria, identificando per ognuno il proprio valore id (rock value), e reimpostando le variabili in modo che corrispondano al loro vero stato nella memoria.

* Fase due: Questa fase è invece dedicata a tutti gli oggetti glk che non sono stati rilevati nella fase uno, ovvero tutto ciò che non è una finestra, un flusso o un riferimento a file. Al momento l'unico oggetto che non ricade in tale classificazione sono i canali audio. Anche qui dovete porre le istruzioni che permettano agli oggetti Glk di reimpostarsi al valore delle variabili ripristinate. Se non avete previsto audio per il vostro gioco non significa che possiate saltare questa fase. Difatti essa serve anche per correggere i riferimenti agli oggetti che hanno modificato il loro stato nel gioco in modo da non generare errori dopo i comandi undo, carica e ricomincia. Ciò viene fatto chiamando in questo punto le funzioni che creano/ridisegnano/modificano i vostri oggetti glk a seconda dei riferimenti ad essi associati.

---

potrebbero essere ricaricati con i loro precedenti valori che a loro volta potrebbero non corrispondere con l'attuale stato in cui sono registrati al momento in memoria. Per esempio, gg_miaFinetraGrafica potrebbe essere ripristinata con un numero, indicante che la finestra a cui punta è aperta, ma dal momento che il gioco inizia con quella finestra chiusa, ecco verificarsi un errore. Dal momento che era stata creata e registrata in memoria, il ripristino non ricreerà la finestra, ma semplicemente la variabile che punta ad essa. Toccherà quindi all'autore attraverso questa entrypoint evitare tali evenienze.

[71] Esempio: Si supponga che dopo una decina di turni nel gioco, si apra una finestra in griglia di testo nella quale bisogna inserire una password per proseguire. Facciamo conto che il giocatore decida di salvare la partita prima di inserire tale password. Poi prosegue con l'inserimento e la finestra a griglia di testo, avendo assolto al suo compito, viene chiusa. Dopo qualche turno ancora il giocatore decide di caricare la partita precedente, ma la finestra per la password non compare, perché? Il motivo è che quando il gioco è stato ricaricato la variabile globale associata alla finestra per la password non è stata reimpostata ma punta ancora ad un valore modificato successivamente. Tale valore non è più valido, poiché punta ora ad un indirizzo in memoria non più corrispondente alla situazione restaurata. Leggi anche la nota precedente su questo problema. La fase zero quindi è prevista perché i riferimenti agli oggetti vengano azzerati. Le fasi successive prevedono invece come ricreare gli oggetti nel giusto modo.

Vediamo un esempio della funzione, che viene chiamata con cinque argomenti:

```
[ IdentifyGlkObject phase type ref rock id;
               ! phase ovvero le fasi su indicate
               ! type  ovvero il tipo di oggetto Glk
               ! ref   il riferimento alla variabile che punta ad esso
               ! rock  il proprio numero identificativo
               ! id    che è una variabile locale


! Fate attenzione di inserire tutti gli oggetti glk creati nel gioco.
       if (phase == 0) {
              gg_miaFinestra        = 0;
              gg_miaAltraFinestra   = 0;
               gg_mioCanaleMusica   = 0;
               gg_mieiEffettiSonori = 0;
               gg_mioFileRef        = 0;
               gg_mioFlussoFile     = 0;
               return;       !return in modo che il processo continui
       }
!Bisogna poi trovare quale è l'attuale stato degli oggetti glk
!giacenti in memoria. Essi vengono identificati attraverso il loro
!numero identificativo (Rock Value) Si tenga presente che 0 o GLK_NULL
!comporta la non esistenza dell'oggetto un valore diverso da zero ne
!comporta l'esistenza.
       if (phase == 1){      ! Ciclo Glulx sui vostri oggetti glk
         switch (type) {
              0:             ! type==0 → è una finestra
                switch (rock){
                   GG_MIAFINESTRA_ROCK: gg_miaFinestra=ref;
                   GG_MIAALTRAFINESTRA_ROCK:gg_miaAltraFinestra=ref;
                }
            1:               ! type==1 → è un flusso
              switch (rock){
                     GG_MIOFLUSSSO_ROCK : gg_mioFlussoFile=ref;
              }
            2:               ! type==2 → è un riferimento a un file
              switch (rock){
                     GG_MIOFILEREF_ROCK : gg_mioFileRef=ref;
              }
```

```
            }          ! fine switch(type)
        return;        ! il processo continua...
    } !fine fase 1


! Ora gli unici oggetti che non ancora identificati sono i canali
!audio. Controlliamo quindi lo stato di questi oggetti, L'operazione
!deve essere fatta nella fase due e non nella uno.


    if (phase == 2){
        ! eseguiamo un ciclo sui canali audio esistenti, dal primo (0)
        ! in avanti (gg_arguments)...
        id = glk_schannel_iterate(0, gg_arguments);
        while (id) {
                switch (gg_arguments-->0) {
                        GG_MYCANALEMUSICA_ROCK : gg_myforechannel = id;
                        GG_MIEIEFFETTISONORI : gg_mybackchannel = id;
                }
                id = glk_schannel_iterate(id, gg_arguments);
        }
! Al successivo caricamento di una partita tutto ciò che è stato
!salvato precedentemente sullo stato del gioco deve essere
!correttamente ripristinato: se una finestra era aperta la apriamo e ci
!disegniamo dentro l'opportuna immagine/i, se era chiusa la chiudiamo.
!Allo stesso modo i suoni. Questo va fatto richiamando le vostre
!funzioni per creare/modificare le finestre disegnare le immagini,
!suonare i suoni.
        CreaMieFinestre();
        DisegnaMieImmagini();
        SuonaMiaMusica();
    } !fine fase 2
]; ! fine della routine
```

Si ricorda che le funzioni personalizzate per creare, distruggere, modificare gli oggetti glk dovrebbero sempre prevedere al loro interno una routine gestalt di controllo. Per degli esempi si faccia riferimento a quelli dedicati all'argomento da Gull, tra il capitolo 7 ed il 10 di questo volume.

. . .

`HandleGlkEvent()`

Glulx Inform, come il classico Inform, gestisce automaticamente gli input del giocatore siano essi delle linee di comando o la pressione di un singolo carattere. Tale gestione però avviene attraverso l'uso di due nuove routine: `KeyboardPrimitive`, e `KeyCharPrimitive`[72]. Di base `KeyboardPrimitive` (o `KeyCharPrimitive` per i menu o per le domande Si/No) funziona come un ciclo continuo in cui una richiesta di comando è sempre in attesa (prompt intermittente). Pertanto quando il giocatore inserisce la linea di comando, un carattere, fa click con il mouse (se attivo) o seleziona un collegamento ipertestuale (se attivo) qualsiasi delle due routine in quel momento è attiva chiama la routine `HandleGlkEvent` per sapere come gestirlo. O meglio riconoscono l'evento come diverso dal classico comando, ed invocano l'entrypoint. Di conseguenza, `HandleGlkEvent` viene chiamata dopo ogni evento e deve gestire ogni evento che non sia un input di carattere o di comando mentre si attende uno di questi. Ciò comporta che se attivate un qualsiasi evento Glk, allora dovrete includere nel vostro sorgente la funzione `HandleGlkEvent()` per gestirlo. La libreria, infatti, non può sapere quali tipi di eventi volete usare e cosa volete associarvi, dovrete essere voi a dirglielo.

Nel dettaglio la funzione `HandleGlkEvent()` viene chiamata dalla libreria quando:

- il giocatore cambia le dimensioni dell'applicazione (la finestra dell'interprete),
- il giocatore cambia le impostazione dei colori,
- un brano musicale arriva alla sua conclusione (se la notifica dell'evento è stata richiesta) ,
- un timer Glk emette un "tick" (se il timer è stato attivato),
- il giocatore inserisce una linea di comando (se l'evento è stato attivato),
- il giocatore inserisce un carattere (se l'evento è stato attivato),
- il giocatore invia un input del mouse (se attivato),
- il giocatore seleziona un collegamento ipertestuale (se attivato).

Riassumendo la routine `HandleGlkEvent()` di base consiste in una funzione che deve gestire gli eventi e dialogare con le routine `KeyboardPrimitive()` e `KeyCharPrimitive()` (le due routine solitamente usate dalla libreria per catturare gli input del giocatore). Quando un giocatore inserisce un comando od un carattere, `HandleGlkEvent()` è chiamata prima che tale comando sia

[72] Si veda a tal proposito il paragrafo 4.2.

mandato al parser. Ciò avviene anche per gli tutti altri eventi in attesa (ridimensionamento, reset, timer, notifica dei sound, etc...).

Un input del mouse o un hyperlink non può soppiantare l'evento input generato da una linea di comando o da un carattere. Allo stesso modo nessun testo può essere inserito dal giocatore prima o mentre `HandleGlkEvents` sta gestendo uno di questi eventi. Pertanto, dal momento che sono anch'essi eventi di input, quando il giocatore li attiva, è una buona idea cancellare ogni possibile comando o carattere in attesa prima di rispondere. L'argomento context della funzione `HandleGlkEvent()` controlla se le funzioni `KeyCharPrimitive()` o `KeyboardPrimitive()` sono attive quando l'input di mouse o l'hyperlink vengono attivati.

Pertanto, se vi sono comandi inseriti dal giocatore, e l'input del giocatore non è interrotto (aborted), i comandi sono mandati al parser di Inform. Se vi sono eventi generati dall'autore, e l'input del giocatore viene interrotto (aborted), gli eventi sono mandati al parser. Se non ci sono comandi (normalmente perché gli eventi non sono di tipo input), nulla viene mandato al parser.

Andiamo a vedere nel dettaglio come è strutturata l'entry point.

La routine è chiamata con alcune variabili come argomenti:

```
[ HandleGlkEvent ev context abortres newcmd cmdlen;
```

Le prime tre variabili sono argomenti passati alla funzione dalla libreria, `ev context` e `abortres`, mentre le ultime due sono variabili locali con nomi convenzionali, `newcmd cmdlen`.

Il **primo argomento è `ev`**, ovvero l'array "evento". E' definito all'interno di parserm.h come `gg_events`, viene usato con tale nome nella libreria, ma quando viene passato alla routine `HandleGlkEvent` è semplicemente chiamato `ev`. I suoi valori indicano:

ev-->0 il tipo di evento (event type)

ev-->1 la finestra coinvolta dall'evento (se rilevante)

ev-->2 è un valore, a seconda dell'uso

ev-->3 è un valore, a seconda dell'uso

Il primo termine dell'array `ev-->0`, ovvero il tipo di evento (event type), è definito da una costante nella libreria `infglk.h`. Ogni tipo di evento ha la radice `evtype_` e la desinenza:

`None`	- nessuno
`Timer`	- evento ripetuto a determinati intervalli di tempo
`CharInput`	- l'input di un carattere in una finestra

`LineInput`	- l'input in linea di comando in una finestra
`MouseInput`	- l'input generato dal mouse in una finestra
`Arrange`	- le dimensioni di una o più finestre sono cambiate
`Redraw`	- le finestre grafiche devono essere ridisegnate
`SoundNotify`	- si è conclusa l'esecuzione del suono
`Hyperlink`	- è stato selezionato in una finestra

`evtype_CharInput` e `evtype_LineInput`, sono eventi che dovrebbero essere lasciati gestire alla libreria a meno di non voler cercare guai. Inoltre se pensate di dover intervenire in tal senso per il controllo dell'inserimento di un carattere date un'occhiata alla funzione `KeyCharPrimitive` probabilmente fa al caso vostro.

`evtype_Arrange` segnala che il giocatore ha ridimensionato l'applicazione, sempre che l'interprete lo permetta, il che ha probabilmente cambiato le dimensioni di tutte le finestre interne. Il ridimensionamento di una finestra di testo non comporta molti problemi dal momento che il buffer del testo non fa che scorrere su linee più o meno lunghe. Diverso discorso va fatto per le finestre grafiche e le finestre a griglia di testo, per le quali un ridimensionamento dello spazio a loro disposizione potrebbe comportare una perdita di informazioni o ad un loro diverso impatto sul layout. Pertanto diventa fondamentale in tali casi prevedere l'evento e approntare gli opportuni cambiamenti agli oggetti.

`evtype_Redraw` è un evento riservato alle finestre grafiche. Dove oltre che al controllo se viene ridimensionata, una finestra grafica può essere scombussolata se il giocatore decide di giocherellare con la risoluzione o l'impostazione dei colori del proprio monitor, con l'effetto di provocare la sparizione delle immagini a schermo o il cambiamento del loro aspetto. In questi casi è necessario ridisegnare tali immagini nella finestra grafica.

`evtype_SoundNotify` notifica quando l'esecuzione di un file audio è terminata.

`evtype_MouseInput` e `evtype_Hyperlink` si spiegano da sole, il giocatore seleziona con il mouse un'area o un collegamento ipertestuale, il che origina l'evento.

Il secondo termine dell'array `ev ovvero ev-->1` indica la finestra coinvolta dall'evento, si ricorda che alcuni eventi possono avvenire solo in determinati tipi di finestre. Tutti gli eventi, con l'eccezione dei timer glk e dei canali audio, sono associati ad una particolare finestra.

Tipo di finestra	Tipo di Input supportato	Non supportato
Finestra di testo	Input di linea di comando Input di carattere Selezione di Hyperlink	Input di mouse
Finestra a griglia di testo	Input di linea di comando Input di carattere Selezione di Hyperlink Input di mouse	
Finestra grafica	Input di mouse	Input di linea di comando Input di carattere Selezione di Hyperlink

Solo le finestre a griglia di testo supportano tutti i tipi di input. Ma vi sono altre limitazioni in ogni caso. E' illegale richiedere un input di linea di comando ed un input di carattere nella medesima finestra contemporaneamente (qualunque sia il tipo di finestra). Dal momento che la maggior parte degli interpreti non possono organizzare secondo priorità le richieste di input del mouse o di un hyperlink è sconsigliato usare nella medesima finestra entrambe le richieste contemporaneamente.

Il terzo e quarto termine dell'array `ev`, `ev-->2` e `ev-->3` sono invece due valori che vengono assegnati a seconda dell'input. Per gli input del mouse, indicano le coordinate x (`ev-->2`) e y (`ev-->3`)[73] dell'area nella finestra dove il mouse può generare l'evento. Per l'evento hyperlink, il primo valore (`ev-->2`) è il numero associato al particolare hyperlink. Per la notifica dei suoni, il primo valore (`ev-->2`) è il numero identificativo rock del canale audio mentre il secondo valore (`ev-->3`) se diverso da zero indica che si desidera che vi sia un evento notifica di cessata esecuzione del suono al suo termine. Naturalmente dal momento che i canali audio non coinvolgono finestre non saranno presenti in alcun modo nel secondo termine dell'array (ev-->1) o meglio saranno pari a GLK_NULL.

[73] Le coordinate sono espresse con le seguenti unità di misura: numero di caratteri in una finestra a griglia di testo, pixel in una finestra grafica.

Il **secondo argomento** della funzione `HandleGlkEvent` è `context` che controlla se l'evento è stato scatenato durante l'attesa di input da linea di comando (`context==0`) o da una richiesta di input di carattere (`context==1`). Dal momento che HandleGlkEvent viene chiamata sia durante l'esecuzione di KeyboardPrimitive che di KeyCharPrimitive, si applicherà l'una o l'altra. Normalmente, ma non sempre, le richieste di input sono nella finestra principale di testo. E normalmente la richiesta di input in attesa è quella della linea di comando (prompt lampeggiante). Se si vuole interrompere l'input del giocatore per eseguire l'evento sopraggiunto allora, quando `context==0`, dovete cancellare la richiesta di input in linea di comando. Quando invece `context==1` dovete cancellare la richiesta di input di carattere. In ogni caso sarà necessario determinare in quale finestra è avvenuta la richiesta sopraindicata.

Il **terzo argomento** della funzione `HandleGlkEvent` è `abortres` ovvero il buffer dell'input. Può restituire un comando come se il giocatore lo avesse digitato direttamente sulla tastiera. Se non usato per tale scopo può essere ignorato.

Gli **altri argomenti** della funzione `HandleGlkEvent` sono variabili locali di supporto per permettere di riportare a seguito dell'evento un'azione come se fosse digitata direttamente da tastiera dal giocatore. In particolare newcmd dovrebbe essere posto uguale al comando che si vuole passare al parser.

```
newcmd = "command";
```

Mentre `cmdlen` dovrebbe contenere le seguenti istruzioni che permettono di scrivere in `abortres` il nuovo comando.

```
cmdlen = PrintAnyToArray(abortes+WORDSIZE,
                         INPUT_BUFFER_LEN-WORDSIZE,newcmd);
                       abortres-->0 = cmdlen;
```

L'istruzione precedente inserisce una stringa nel buffer dell'input così che possa essere inviato al parser di Inform come un comando generato dall'autore (Se tale operazione viene eseguita `HandleGlkEvent` dovrebbe ritornare il valore 2 per interrompere l'input del giocatore). Tale istruzione mostra anche un piccolo ma elegante trucchetto, `PrintAnyToArray` infatti restituirà l'esatta lunghezza in byte di qualunque cosa[74].

Il valore di ritorno (return value) di `HandleGlkEvent` può essere: `return` (nothing), `rtrue` (1), o `rfalse` (0); Tali valori non influiscono sull'input del giocatore. `Return 2`, invece interromperà l'input del giocatore in modo che

[74] Si approfondisca al paragrafo §5.3.2.

`abortres` (o meglio il comando ivi contenuto) possa essere trattato come se il giocatore lo avesse digitato da tastiera. Infine può ritornare `Return -1`, che permetterà al giocatore di continuare anche dopo la pressione dell'invio (input di linea) o della pressione di un tasto (input di carattere). Comunque `return -1` non è normalmente usato.

Infine come ultimo accorgimento nella scrittura della propria `HandleGlkEvent` è necessario considerare che il testo non può essere scritto in una finestra quando una richiesta di input di linea o di carattere è ancora in attesa nella finestra stessa. Pertanto cancellare qualunque dei due sia in attesa quando un `HandleGlkEvent` risponde ad un input di mouse o hyperlink non è una cattiva idea. Se non avete intenzione di stampare del testo nella finestra, la cancellazione della richiesta in attesa non è necessaria, ma ancora una volta potrebbe non essere una cattiva idea eseguirla ugualmente.

A questo punto per completare questo approfondimento non ci rimane che vedere un esempio:

```
[ HandleGlkEvent ev context abortres newcmd cmdlen;
    switch (ev-->0){

! Le finestre sono ridimensionate dal giocatore, bisogna aggiornarle
!assieme alle immagini ivi disegnate.
        evtype_Arrange :      ModificaMieFinestre ();
                              DisegnaMieImmagini ();

! Le impostazioni di Colore/risoluzione vengono cambiate dall'utente,
!controlla la grafica
        evtype_Redraw :       DisegnaMieImmagini ();

! La musica/effettoSonoro ha terminato di suonare, fai qualcosa di
!appropriato.
        evtype_SoundNotify:   RespondiNotifica();

! E' arrivato un input del mouse, avvia qualche azione. Nota: l'input
!di Mouse e gli hyperlink sono associate a particolari finestre. Vedi a
!pagina.. ..
        evtype_MouseInput :
                if (ev-->1 == gg_miaFinestraGrafica) {
! l'input canale,linea o carattere dipende dal contesto
                if (context == 1) CancelCharInput(gg_mainwin);
            else CancelLineInput(gg_mainwin);
```

```
            newcmd = "open door";
            cmdlen = PrintAnyToArray(abortres+WORDSIZE,
                     INPUT_BUFFER_LEN-WORDSIZE, newcmd);
            abortres-->0 = cmdlen;
            return 2;
        }
```

!Restituisce il comando come se fosse stato digitato dal giocatore, facendolo !passare attraverso il parser e quindi verso l'appropriato oggetto e all'appropriata !locazione. Un valore di ritorno pari a 2 corrisponde al comando come digitato !dal giocatore.

!Il commando "open door" o "n" viene inserito nell'array abortres (input !buffer) per essere passata al parser e processata. PrintAnyToArray in realtà !copia il commando nell'input buffer, e ritorna l'esatta lunghezza byte della !stringa rappresentate il comando inserito dall'autore (e non dal giocatore), !Quindi la lunghezza è inserita nel primo termine dell'input buffer. Quindi viene !restituito il valore 2 per passare far tornare il comando al parser esattamente !come se il giocatore lo avesse digitato sulla tastiera.

```
       evtype_HyperLink :
         if (ev-->1 == gg_mainwin){
```

!cancella linea o carattere, l'input dipende dal contesto context

```
            if (context == 1) CancelCharInput(gg_mainwin);
            else CancelLineInput(gg_mainwin);
```

! ev-->2 in questo caso è lo specifico numero dell'hyperlink.

```
            switch(ev-->2){
                1 : newcmd = "open door";
                2 : newcmd = "n";
            }
            cmdlen = PrintAnyToArray(abortres+WORDSIZE,
                      INPUT_BUFFER_LEN-WORDSIZE, newcmd);
            abortres-->0 = cmdlen;
            return 2;
        }
      evtype_Timer :
        if (timer_counter == NULL) StopGlkTimer();
        else AvviaMiaRoutineTime();
]; ! fine
```

APPENDICE D – Evitare gli errori nell'uso degli oggetti glk

Un normale oggetto in Inform, come una locazione, è del tutto testuale indipendente dalla piattaforma, pertanto può essere salvato su un file tranquillamente. Gli oggetti glk invece, come ad esempio una finestra grafica con delle immagini all'interno, viene aperta e disegnata dinamicamente dall'interprete glulxe come permesso dalla piattaforma del giocatore. Pertanto quest'ultima, assieme al sistema operativo ed alla risoluzione dello schermo, può influenzare il comportamento del gioco. Visto che gli oggetti glk sono dipendenti dalle piattaforme e gestiti in memoria, non possono essere salvati su disco.

Allora cosa viene salvato? La variabile globale che "punta" ad ogni oggetto glk. In Glulx ciò viene chiamato riferimento. Per esempio, per creare una finestra grafica, avrete bisogno di una variabile globale e di una costante che individui un numero identificativo (rock number) per essa.

```
Constant GG_MIAFINESTRAGRAFICA_ROCK 210;
Global gg_miaFinestraGrafica = 0;
```

Da qualche parte nel codice, normalmente nella funzione `Initialise`, dovremo aprire la finestra:

```
gg_miaFinestraGrafica = glk_window_open(gg_mainwin,
        winmethod_Above+winmethod_Proportional,
        30, wintype_Graphics, GG_MIAFINESTRAGRAFICA_ROCK);
```

La variabile, `gg_miaFinestraGrafica`, è ciò che viene salvato su disco e potrà essere pari a:

0	→ non esistente o chiusa
# (numero di riferimento dell'oggetto)	→ creata/aperta.

Poiché Initialise è chiamata prima del primo turno, al giocatore sembra che il gioco sia cominciato con la finestra grafica già aperta. Ma in realtà è stata creata nella memoria dall'interprete e dalle sue librerie glk subito dopo il caricamento del gioco.

Una variabile globale di un oggetto glk deve essere controllata per assicurarsi che non sia pari a zero prima di cercare di usare l'oggetto stesso, dal momento che potrebbe non essere aperta con successo (a causa di memoria insufficiente, grandezza dello schermo, etc...). E' necessario controllarla anche prima dei

successivi usi, dal momento che qualcosa potrebbe essere cambiato nel frattempo.

```
if (gg_miaFinestraGrafica ==0)
        print "Ooops! Abbiamo un problema, la finestra
        grafica non è disponibile per le immagini. La
        figura non verrà mostrata. ^";
glk_image_draw(gg_miaFinestraGrafica,imagine_grande,0,0);
```

Esiste poi l'eventualità che l'interprete utilizzato dal giocatore non supporti alcuni degli oggetti glk presenti nel gioco. Allo scopo di controllare l'adeguatezza dell'interprete la libreria ci mette a disposizione le funzioni Glk Gestalt. Esse permettono di evitare che l'interpreti vada in crash, se le aggiungerete in modo corretto. Le gestalt possono controllare che l'interprete del giocatore supporti: le immagini grafiche, gli input via mouse, i collegamenti ipertestuali e i suoni. Inoltre in alcune piattaforme potrebbero essere supportati solo alcuni formati audio.

La stessa funzione, glk_gestalt, è usata per controllare tutte queste caratteristiche. Il primo argomento della funzione determina quale caratteristica glk si vuole testare. Come ad esempio i suoni:

```
result = glk_gestalt(gestalt_Sound, 0);
```

Se l'interprete del giocatore non supporta una caratteristica, potete includere nel codice: un'uscita dal gioco, chiedendo al giocatore di usare un interprete più consono; l'impostazione di una variabile che attraverso istruzioni if/else includa o escluda la caratteristica dal gioco ed avvertire il giocatore dell'inconveniente.

```
if ( glk_gestalt(gestalt_Sound,0) && gg_mioCanaleAudio )
        glk_schannel_play(gg_mioCanaleAudio, mormorio_snd);
else print "Puoi udire un mormorio di voci. ^";
```

Il normale controllo degli errori di Inform può essere usato in combinazione con la funzione gestalt per essere certi che tutto sia pronto. Non è una cattiva idea richiamare la gestalt ogni qual volta si accede ad un oggetto glk. In modo da prevenire problemi anche nel caso il giocatore carichi una precedente partita salvata su di un altro computer o giocata con un diverso interprete.

I modi in cui può essere chiamata la gestalt, oltre ad essere ben spiegati nel § 9.4 sono mostrati nella tabella di riferimento alla fine di questo volume.

Un altro possibile tipo di errore a cui si può andare incontro è conosciuto con il termine "zero errors". In Inform molti degli "zero errors[75]" che vengono generati dipendono dall'uso, come condizioni in una routine, di variabili non assegnate. Vediamo un esempio, supponiamo che non venga assegnato alcun valore alla variabile `second`:

```
if (second == qualcosa) fai quacosa;
```

Non tutte le routine della grammatica di Inform prevedono un valore per `second`. La soluzione consiste nel controllare prima che second esista e poi vedere se corrisponde a qualcosa.

```
if ((second ~= 0) && (second == qualcosa ))fai qualcosa;
```

Glulx da questo punto di vista può costituire un'altra fonte di questo tipo di errori. Supponiamo di chiudere una finestra Glk, e quindi più tardi di usare la variabile ad essa associata come condizione per avviare una azione. Alcuni interpreti riporteranno un errore Glk.

```
CloseWindow(gg_mywin);
```

In questo momento la variabile gg_mywin non è più un valido riferimento e l'istruzione seguente può essere fonte di errori.

```
if (gg_mywin) fai qualcosa;
```

Comunque, un riferimento non valido ad un oggetto Glk **non** è zero, ma punta ad un indirizzo in memoria che non esiste più. La soluzione più semplice è quella di assegnare zero come valore della variabile subito dopo l'istruzione con cui abbiamo chiuso o distrutto l'oggetto glk. In tal modo la variabile potrà essere utilizzata più tardi per controllare la situazione dell'oggetto stesso. Ad esempio, le seguenti istruzioni non comportano errori:

```
CloseWindow(gg_mywin);
gg_mywin = 0;
if (gg_mywin) fai qualcosa;
if (gg_mywin == 0) fai qualcosa;
```

[75] Ovvero gli errori che non sono evidenti in compilazione ma crashano il sistema su alcuni interpreti.

APPENDICE E – Gli Eventi

In questa appendice andremo a elencare tutti i tipi di eventi che possono essere richiesti o controllati in Glulx.

Il primo tipo di eventi è quello legato al **ridisegno delle immagini** e delle finestre. E' il più semplice degli eventi glk. La maggior parte del codice sarà inserito all'interno della funzione `HandleGlkEvents();` dove si gestirà per mezzo di routine che già esistono nel gioco l'apertura/chiusura e disegno delle finestre.

Quando il giocatore modificherà la grandezza dell'applicazione, la `HandleGlkEvents();` gestirà il ridimensionamento di tutte le finestre e del loro contenuto facendo leva sul codice opportunamente predisposto dall'autore del gioco[76]. Allo stesso modo sarà possibile gestire anche il cambiamento di risoluzione del monitor da parte del giocatore.

Gli eventi "ridisegno delle finestre" non sono attivati dall'autore del gioco, ma dal giocatore quando compie determinate azioni (ridimensionamento, reset) che coinvolgono le finestre.

Un secondo tipo di eventi è quello legato alle **notifiche dei suoni**. Quando l'esecuzione di un suono termina si genera (se richiesto dall'autore) una notifica del fatto. L'evento può essere usato per inserire nel proprio gioco alcuni effetti speciali.

Se richiedete la notifica dell'evento sound, chiamando la funzione PlaySound() e ponendo a True il parametro Notify, allora la routine `HandleGlkEvents();` aspetterà tale evento e quindi eseguirà il codice che avrete predisposto al suo accadere (sempre che l'interprete supporti tale caratteristica).

Un terzo tipo di eventi sono quelli legati al **tempo**. Le Glk hanno un singolo timer globale, molto simile all'orologio di sistema. Pertanto, diversamente dal timer di Inform, il timer Glk è basato sul "tempo reale", non sul tempo del gioco. Inoltre, la "fetta minima di tempo" permessa dalle Glk è un semplice e singolo "tick". Una volta che il timer è cominciato ad un certo tasso di avanzamento, esso lo mantiene fino a quando viene fermato. Ogni volta che un tick viene raggiunto, il codice previsto dall'autore può essere eseguito. Anche questo evento può essere usato per aggiungere effetti speciali al vostro gioco, come ad esempio l'inserimento di immagini animate[77].

---

[76] Una disamina approfondita della funzione è disponibile nell'appendice sulle entry point.

[77] Si veda l'esempio sul realtime nel paragrafo 10.

Un timer glk può essere attivato o disattivato, ma non c'è modo di sapere se al momento è in fase di avanzamento o meno. Dal momento che può solo andare avanti allo stesso ritmo di un orologio, usare una flag globale per esso è un buon sistema per tenere traccia di quando è attivo o disattivo. Il timer glk può tenere conto di più eventi legati al tempo contemporaneamente, ma ogni evento temporale avrà il medesimo tasso di avanzamento. Pertanto se si desidera implementare più eventi temporali che avanzino a velocità diverse, dovrete separarli nel codice in modo che non si sovrappongano.

Un ulteriore tipo di evento, fondamentale nei giochi, è quello legato all'**input del giocatore**. Il che può avvenire tramite una linea di comando, un carattere, un click del mouse o la selezione di un hyperlink. Stiamo parlando della base dell'interazione del giocatore con il gioco.

L'autore deve attivare tutti gli eventi input prima che essi siano usati, altrimenti non produrranno effetti. Un evento input viene attivato per una singola finestra. Quando il giocatore incontra la richiesta di evento e lo genera allora verrà richiamata la funzione `HandleGlkEvent()` per processarlo.

Tutti gli eventi sono associati a particolari finestre. E non tutti i tipi di finestre supportano qualsiasi tipo di evento:

Tipo finestra	Input supportati	Input non supportati
Testo	Linea di comando Inserimento carattere Hyperlink	Mouse
Griglia di testo	Linea di comando Inserimento carattere Mouse Hyperlink	
Grafica	Mouse	Linea di comando Inserimento carattere Hyperlink

Inoltre un input di linea di comando ed uno di carattere non possono essere attivi nel medesimo momento nella medesima finestra. Allo stesso modo anche

un input di mouse ed un hyperlink non possono essere attivi nel medesimo istante nella medesima finestra. Naturalmente altre combinazioni sono invece possibili ad esempio: un input di linea e di mouse possono essere attivati nella medesima finestra contemporaneamente. Tutte le combinazioni sono possibili poi se si usano finestre differenti.

L'evento più classico tra quelli di input è chiaramente quello che coinvolge la richiesta di un carattere o la linea di comando. Entrambi questi eventi normalmente "fermano" il gioco e aspettano che il giocatore risponda. Ciò potrebbe non essere immediatamente evidente, dal momento che mentre si aspetta per tali eventi, il giocatore potrebbe rispondere con un altro tipo di input.

L'input di **linea di comando** richiede al giocatore di inserire un ordine di seguito al prompt lampeggiante. La libreria di Inform gestirà i normali input di linea e, a meno che non diventiate grandi esperti di Inform, non dovreste interferire con la libreria in questo argomento. Comunque, se lo si desidera, la routine `KeyboardPrimitive();` è la funzione che normalmente viene usata dalla libreria per individuare l'input di linea del giocatore.

L'input di **carattere** è anch'esso gestito principalmente dalla libreria attraverso la routine `KeyCharPrimitive()`, che individuerà la pressione di un tasto da parte del giocatore catturandone il valore (`key = KeyCharPrimitive();`) e confrontandolo con i codici dei tasti (keycode) definiti dalle costanti in infglk.h.

Gli eventi linea/carattere sono normalmente attivati dalla libreria di Inform, attraverso le routine `KeyboardPrimitive()` o `KeyCharPrimitive()`. Gli input di linea rimangono in attesa fino al momento che il giocatore preme INVIO per terminare la linea di comando. Gli input di carattere rimangono in attesa fino a quando il giocatore non preme un tasto.

Gli ultimi tipi di eventi di input glk gestiti sono quelli generati dal **mouse** o dalla selezione di un **hyperlink**. Normalmente sono previsti come scorciatoie ai normali comandi di linea in Inform. Sarà necessario inserire del codice nella routine `HandleGlkEvent()` per passare tali comandi al parser di inform, che li processerà abitualmente.

Ad esempio: Un hyperlink sulla parola "Nord" potrebbe generare un normale commando "n" da tastiera e passarlo in tal modo al parser come se fosse stato digitato dal giocatore.

L'attesa per gli input di mouse o di selezione di un link, normalmente, non fermano il gioco per aspettare la risposta del giocatore. Se e quando il giocatore li attiva essi vengono considerati completi e passati alla funzione `HandleGlkEvent()` per essere processati, altrimenti possono rimanere incompleti o sempre in attesa.

Un input di Mouse si genera quando si preme il pulsante del mouse su un'area sensibile (così come definite dall'autore). Allo stesso modo un evento di selezione di un hyperlink si genera nel momento in cui il giocatore preme il pulsante del mouse sul testo o immagine associata al link.

Entrambi gli eventi coinvolgono il mouse, e sono attivabili solo in determinate finestre. Devono inoltre essere attivati ogni volta che si intende permettere al giocatore di usarli.

APPENDICE F – UNICODE

Unicode è un sistema di codifica che assegna un numero (o meglio, una combinazione di bit) ad ogni carattere in maniera indipendente dal programma, piattaforma e dalla lingua (e relativo alfabeto).

Unicode si basa sulla vecchia codifica ASCII esteso (ISO 8859), che consentiva la rappresentazione di 256 caratteri ed era sufficiente per gli alfabeti dell'Europa Occidentale e del Nord America.

Unicode va molto oltre, codificando i caratteri usati in quasi tutte le lingue vive e in alcune lingue morte, nonché simboli matematici e chimici, cartografici, l'alfabeto Braille, ideogrammi etc.

L'ASCII (anche nella sua forma estesa), non possedeva un numero di caratteri sufficienti per tutte le lingue e le necessità di comunicazione in qualsiasi ambito disciplinare. Per ciascuna esigenza, era necessario utilizzare una tabella apposita, molto spesso proprietaria. Attualmente lo standard Unicode non rappresenta ancora tutti i caratteri che potrebbe rappresentare; essendo ancora in evoluzione, però, forse in futuro arriverà a coprire tutti i caratteri rappresentabili. Una curiosità: tra le lingue coperte da Unicode vi è il Klingon, ben conosciuto dai fan di Star Trek.

Al momento lo standard Unicode (e l'ISO/IEC 10646) supporta tre forme di codifica che condividono un repertorio comune di caratteri, ma possono essere estese fino a rappresentarne circa un milione. Ciò appare sufficiente a coprire anche i fabbisogni di codifica di scritti del patrimonio storico dell'umanità, nelle diverse lingue e negli svariati sistemi di segni utilizzati.

L'Unicode viene supportato dai moderni standard della programmazione e del markup come XML, Java, JavaScript, da vari sistemi operativi, ed ora anche dalle Glk e quindi da Glulx Inform.

In particolare le Glk hanno due Api, separate ma parallele, per gestire gli input ed output di testo. Le funzioni base trattano unicamente caratteri in 8-bit, ed i loro argomenti sono array di byte (ottetti). Queste funzioni assumono tutte la codifica di caratteri denominata Latin-1[78].

Latin-1 è una codifica di caratteri in 8-bit; essa prevede una mappa di codici numerici che vanno da 0 a 255 che corrispondono a altrettanti caratteri di stampa. I valori da 32 a 126 sono i caratteri di stampa standard chiamati ASCII (' ' to '~'). I valori da 0 a 31 e da 127 a 159 sono riservati per i caratteri di controllo, e non hanno equivalenti caratteri di stampa.

Le funzioni Glk estese, o "Unicode", trattano i caratteri interamente in parole (word) a 32-bit. Esse hanno come argomenti array di parole e non byte. Esse

[78] Equivalentemente, si può dire che usino i punti di codice da U+00 a U+FF di Unicode.

possono quindi far fronte a qualsiasi codice Unicode. Le funzioni estese, che supportano Unicode, hanno quale ultima appendice del proprio nome la desinenza "_uni".

Andiamo a vedere qui di seguito cosa comporta l'introduzione della codifica Unicode nel proprio programma.

Come per tutte le altre caratteristiche Glk, sarà necessario testare se l'interprete con cui viene avviato il programma supporta la codifica Unicode. Le funzioni di testo di base saranno disponibili in ogni libreria Glk. Mentre le funzioni Unicode potrebbero essere o non essere disponibili[79]. Prima di chiamarle pertanto sarà necessario eseguire una gestalt appropriata:

```
resut = glk_gestalt(gestalt_Unicode, 0);
```

La libreria infglk.h dalla versione 0.70 prevede infatti l'aggiunta dell'apposita costante:

```
Constant gestalt_Unicode 15;
```

La gestalt restituisce True (1) se le funzioni Unicode sono disponibili. Se invece restituisce False (0) allora tali funzioni non dovrebbero essere chiamate nel proseguo del programma. Esse infatti potrebbero non stampare nulla, stampare in una lingua incomprensibile o peggio causare un errore nell'esecuzione del gioco. Le funzioni Glk incluse nella libreria per la gestione dell'Unicode sono:

```
glk_buffer_to_lower_case_uni
glk_buffer_to_upper_case_uni
glk_buffer_to_title_case_uni
glk_put_char_uni
glk_put_string_uni
glk_put_buffer_uni
glk_put_char_stream_uni
glk_put_string_stream_uni
glk_put_buffer_stream_uni
glk_get_char_stream_uni
glk_get_buffer_stream_uni
glk_get_line_stream_uni
glk_request_char_event_uni
glk_request_line_event_uni,
glk_stream_open_file_uni
glk_stream_open_memory_uni
```

[79] Al momento in cui scriviamo non vi sono interpreti che supportano l'Unicode. E' plausibile pensare che in breve i maggiori interpreti verranno aggiornati per adeguarsi ai nuovi standard glk. Si fa comunque presente che il supporto Unicode da parte di un interprete non è banale, diventa quindi fondamentale controllare attraverso le gestalt tale caratteristica.

Quando si decide di inviare del testo in una finestra, o in un file aperto in modalità testuale, è possibile stampare un qualsiasi tipo di carattere Latin-1: da 32 a 126, da 160 a 255. E' anche possibile stampare una nuova linea.

Non è legale stampare qualsiasi altro tipo di carattere di controllo (da 0 a 9, da 11 a 31, da 127 a 159). Non è possibile stampare neanche caratteri di formattazione quali il tab (control-I), il tasto invio (control-M), o l'interruzione di pagina (control-L).

Stampare caratteri Unicode superiori a 255 complica le cose, le complicazioni sono tali da non poter essere affrontate in questa trattazione . Fate riferimento pertanto alle specifiche Unicode e buona fortuna[80].

Qui di seguito riportiamo una guida di riferimento alle funzioni glk unicode.

`glk_buffer_to_lower_case_uni (buf,len,numchars);` *Unicode*

E' l'equivalente di `glk_char_to_lower(ch)` per i caratteri Unicode, e converte in minuscolo qualsiasi carattere sia presente nel buffer indicato al primo parametro `buf`.

La conversione dei caratteri Unicode è leggermente più complessa di quella normale, e deve essere applicata agli array di caratteri, e non ai singoli caratteri.

La funzione prevede tre argomenti. Il primo come detto individua il buffer a cui ci si rivolge, il secondo parametro `len` è la lunghezza disponibile del buffer, mentre `numchars`, il terzo parametro, è il numero di caratteri nel buffer all'inizio[81]. Pertanto `numchars` deve essere minore o uguale a `len`. Il contenuto del buffer successivo a `numchars` non viene considerato nell'operazione.

La funzione restituisce il numero di caratteri dopo la conversione. Se questo è maggiore di `len`, i caratteri nell'array saranno certamente troncati a `len`, ma il conto reale sarà ritornato. (I contenuti del buffer dopo che il conto viene restituito dalla funzione sono indefiniti).

---

[80] Mettere assieme caratteri Unicode è particolarmente fastidioso. La stampa di una combinazione di caratteri può alterare i caratteri precedentemente stampati. La libreria dovrebbe essere preparata a tale scopo, anche se i caratteri vengono stampati attraverso due distinte chiamate alla funzione glk_put_char_uni();

[81] Sono presenti due parametri per la lunghezza perché nel momento in cui una stringa di caratteri Unicode viene cambiata anche solo tra maiuscolo e minuscolo essa può espandersi o contrarsi.

`glk_buffer_to_upper_case_uni(buf,len,numchars);` *Unicode*

E' l'equivalente di `glk_char_to_upper(ch)` per i caratteri Unicode, e converte in maiuscolo qualsiasi carattere sia presente nel buffer indicato al primo parametro `buf`.

La conversione dei caratteri Unicode è leggermente più complessa di quella normale, e deve essere applicata agli array di caratteri, e non ai singoli caratteri.

La funzione prevede tre argomenti. Il primo come detto individua il buffer a cui ci si rivolge, il secondo parametro `len` è la lunghezza disponibile del buffer, mentre `numchars`, il terzo parametro, è il numero di caratteri nel buffer all'inizio[82]. Pertanto `numchars` deve essere minore o uguale a `len`. Il contenuto del buffer successivo a `numchars` non viene considerato nell'operazione.

La funzione restituisce il numero di caratteri dopo la conversione. Se questo è maggiore di `len`, i caratteri nell'array saranno certamente troncati a `len`, ma il conto reale sarà ritornato. (I contenuti del buffer dopo che il conto viene restituito dalla funzione sono indefiniti).

`glk_buffer_to_title_case_uni(buf,len,numchars,lowerest);`

Converte in maiuscolo il primo carattere presente nel buffer indicato al primo parametro `buf` e lascia il resto del buffer invariato.

La conversione dei caratteri Unicode è leggermente più complessa di quella normale, e deve essere applicata agli array di caratteri, e non ai singoli caratteri.

La funzione prevede quattro argomenti. Il primo come detto individua il buffer a cui ci si rivolge, il secondo parametro `len` è la lunghezza disponibile del buffer, mentre `numchars`, il terzo parametro, è il numero di caratteri nel buffer all'inizio[83]. Pertanto `numchars` deve essere minore o uguale a `len`. Il contenuto del buffer successivo a `numchars` non viene considerato nell'operazione.

Il quarto parametro `lowerrest` è di tipo booleano. Se è pari a True (1), modifica il resto della stringa in minuscolo (invece di lasciarla come era prima).

La funzione restituisce il numero di caratteri dopo la conversione. Se questo è maggiore di `len`, i caratteri nell'array saranno certamente troncati a `len`, ma il conto reale sarà ritornato. (I contenuti del buffer dopo che il conto viene restituito dalla funzione sono indefiniti).

[82] Sono presenti due parametri per la lunghezza perché nel momento in cui una stringa di caratteri Unicode viene cambiata anche solo tra maiuscolo e minuscolo essa può espandersi o contrarsi.

[83] Sono presenti due parametri per la lunghezza perché nel momento in cui una stringa di caratteri Unicode viene cambiata anche solo tra maiuscolo e minuscolo essa può espandersi o contrarsi.

`glk_get_buffer_stream_uni(str,buf,len);` *Unicode*

La funzione legge `len` caratteri Unicode dal dato flusso `str`, a meno che si raggiunga prima la fine del flusso. Nessun valore NULL è posto alla fine del flusso. Restituisce il numero di caratteri Unicode realmente letti.

`glk_get_char_stream_uni (str);` *Unicode*

Legge un carattere Unicode dal dato flusso. Il risultato sarà compreso tra 0 e 0x7fffffff. Se si raggiunge la fine del flusso, il risultato sarà pari a -1.

`glk_get_line_stream_uni(str,buf,len);` *Unicode*

La funzione legge i caratteri Unicode dal dato flusso `str`, fino a che o siano stati letti `len-1` caratteri Unicode o si sia incontrata una nuova linea. Viene quindi inserito un carattere finale NULL (un valore zero) alla fine del flusso. Restituisce il numero di caratteri realmente letti, inclusa la nuova linea (se ve ne è una) ma non includendo il valore finale NULL.

`glk_put_buffer_uni(buf, len);` *Flusso Unicode*

Stampa blocchi di caratteri Unicode nel flusso corrente, ed è equivalente ad una serie di chiamate alla funzione `glk_put_char_uni()`.

`glk_put_buffer_stream_uni(buf,len);` *Flusso Unicode*

Analoga alla funzione `glk_put_buffer_stream()` con l'accorgimento che len è un glui32 per la codifica di caratteri Unicode.

`glk_put_char_uni(ch);` *Flusso Unicode*

Stampa un carattere nel flusso corrente. Il carattere è di tipo Unicode (Unicode code point).

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

`glk_put_char_stream_uni(str,ch);` *Flusso Unicode*

Analoga alla funzione `glk_put_char_stream()` con l'accorgimento che ch è un glui32 per la codifica di caratteri Unicode.

`glk_put_string_uni(s);` *Flusso Unicode*

Stampa una stringa Unicode in coda al flusso corrente seguita da un glui32 il cui valore è pari a 0. Il parametro s è la stringa Unicode. E' equivalente ad una serie di chiamate alla funzione `glk_put_char_uni()`.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

`glk_put_string_stream_uni(str,buf,len);` *Unicode*

Analoga alla funzione `glk_put_string_stream()` con l'accorgimento che len e buf sono glui32 per la codifica di caratteri Unicode.

`glk_request_char_event_uni(win);` *Evento Unicode*

Richiede l'input di un carattere Unicode o di un tasto speciale nella finestra indicata nel parametro (win). Tale richiesta elimina qualsiasi precedente richiesta di carattere in attesa.

Restituisce True (1) in caso di successo o False (0) in caso di fallimento.

Esempio: `risultato=glk_request_char_event_uni (gg_miaText_win);`

`glk_request_line_event_uni(win,buf,maxlen,initlen);`

Richiede l'inserimento di un comando di linea nella finestra indicata al primo parametro (win), questa funzione è analoga alla glk_request_line_event(), con l'eccezione che il risultato viene registrato in un array di valori glui32 invece che in un array di caratteri, e i valori possono essere qualsiasi codice valido di Unicode (valid Unicode code points).

`glk_stream_open_file_uni(fileref,fmode,rock);`

La funzione è analoga a `glk_stream_open_file()`, con l'eccezione che in modalità binaria (binary mode), i caratteri sono scritti e letti come valori a quattro byte (big-endian). Ciò vi permette di scrivere e leggere qualsiasi carattere Unicode.

In modalità testo (text mode), il file è scritto e letto in un modo che varia a seconda della piattaforma, che potrebbe o meno gestire tutti o caratteri Unicode. Un file creato in modalità testo con `glk_stream_open_file_uni()` può avere lo stesso formato di un file creato in modalità testo con `glk_stream_open_file();` o può usare un formato più Unicode-friendly.

```
glk_stream_open_memory_uni(buf,buflen,fmode,rock);
```

La funzione è analoga a `glk_stream_open_memory()`, con l'eccezione che il buffer è un array di parole a 32 bit, invece che di byte. Ciò vi permette di scrivere e leggere qualsiasi carattere Unicode. Il parametro `buflen` è il numero di parole, non il numero di byte. [Se il buffer contiene il valore 0xFFFFFFFF, ed è aperto in lettura, il lettore non può distinguere tale valore da -1 (fine del file). Fortunatamente 0xFFFFFFFF non è un codice valido per un carattere Unicode].

GLOSSARIO

Baco

Vedi Bug.

Bug

Letteralmente insetto, in Italiano solitamente viene tradotto come Baco. Si tratta di un errore di programma non individuato in fase di compilazione e/o test.

Classi Glk

Ogni oggetto glk è membro di una classe. Una finestra glk fa parte della classe finestre glk, un canale audio fa parte della classe glk dei canali audio, etc… Una funzione di ripetizione (iteration) è stata prevista in modo tale da poter eseguire cicli su tutti i membri di una classe glk ed avere effetto su di ognuno di essi. Ciò non solo permette di risparmiare righe di codice, ma diventa necessario quando si interagisce con i canali audio nella funzione `IdentifyGlkObject`.

```
obj=glk_schannel_iterate(GLK_NULL, GLK_NULL);
while (obj){
     !fai qualcosa con il !canale audio obj
    obj=glk_schannel_iterate(obj, GLK_NULL);
    }
```

La funzione è disponibile per ogni classe, ed il suo nome è composto dal suffisso `glk_` il nome della classe e l'appendice `_iterate`. Ad esempio: `glk_windows_iterate`. Il primo argomento se pari a `GLK_NULL`, farà in modo che la funzione glk_class_iterate ritorni il primo oggetto della classe.

Compilatore

Vedi Compiler.

Compiler

Si tratta di un programma che si occupa di convertire quanto scritto in un linguaggio ad alto livello (più comprensibile ad una persona, nel nostro caso Inform) nel linguaggio specifico del computer (nel nostro caso lo Z-Code della Z-Machine). Di solito la parte scritta nel linguaggio ad alto livello viene chiamata "codice sorgente", mentre il risultato della compilazione viene chiamato "codice oggetto".

DM4

L'Inform Designer's Manual versione 4: il libro su Inform scritto da Graham Nelson.

Emulatore
È un programma che consente di eseguire dei programmi compilati per una macchina/piattaforma differente sul vostro computer. Nel nostro caso si tratta degli Interpreti che consentono di eseguire il codice Z-Code o Glulx nel sistema operativo delle nostre macchine. Vedi il paragrafo "8.4 Dal codice della VM al vostro schermo" per una trattazione più completa.

Entry Point
Letteralmente Punto d'Ingresso. Si tratta di chiamate specifiche a routine di librerie predisposte da qualcuno, ma nel caso di Inform con entry point si intendono delle routine che potete scrivere o meno e che vengono chiamate in determinate occasioni dalla libreria di Inform.

Evento GLK
Si consulti l'appendice E.

Finestra
Vedi Window.

Flussi (stream)
I flussi sono essenzialmente un "torrente" di informazioni, normalmente testo. Tutti i caratteri in uscita (output) in Inform Glulx sono gestiti dai flussi. Vi sono: flussi di finestra (ogni finestra ha un flusso output di default associato ad essa), flussi di memoria, e flussi nei file.
I flussi di file sono molto usati. Essi realizzano la vera scrittura o lettura del file su disco. Aprire un flusso di file apre il file; chiuderlo chiude il file. La libreria gestisce la normale scrittura su file: salvataggi delle partite, trascrizione, e la registrazione dei comandi impartiti in fase di debug (i flussi gg_savestr, gg_scriptstr, e ggcommand_str sono creati dalla libreria). Pertanto non avete bisogno di lavorare con i flussi a meno che non abbiate intenzione di creare file particolari per ottenere qualche effetto speciale.

GIP
Guida ad Inform per Principianti: è la traduzione in Italiano del libro di Roger Firth Inform Beginner's Guide.

GLK Spec
Sono le specifiche (le regole di funzionamento) della libreria GLK scritte da Andrew Plotkin.

Gruppo di discussione
Vedi newsgroup.

IBG
Inform Beginner's Guide: la guida ad Inform per i principianti scritta da Roger Firth. Ne esiste una traduzione in Italiano (vedi GIP).

Interprete
Vedi Emulatore.

Library
Un insieme di routine e funzioni che si occupano di risolvere una serie di problemi comuni. Nel caso di Inform con libreria si intendono quei file che contengono le routine che si occupano di gestire il mondo nel quale si svolge il gioco, implementare il parser (l'analizzatore dei comandi del giocatore) e così via.

Libreria
Vedi Library.

Macchina Virtuale
Vedi Virtual Machine.

Newsgroup
In Italiano Gruppo di discussione, ma anche Forum. Sono una serie di "contenitori" che contengono messaggi inviati da utenti di Internet da varie parti del mondo. Ci sono diversi gruppi di discussione, ognuno legato ad un certo argomento, le persone condividono vi opinioni e passioni discutendo dei loro interessi.

Opcode
Codice operativo, si tratta di un'istruzione elementare (a basso livello, comprensibile dal computer) che effettua una determinata operazione.

Parola
Vedi Word.

Patch
Si tratta di correzioni apportate ad un programma o ad un libreria per risolvere un problema causato da un bug.

Punto d'ingresso
Vedi Entry Point.

Riferimenti ai File
I riferimenti ai file puntano a file esterni che in tal modo possono essere aperti/modificati/consultati/creati. Seguono un poco la stessa forma degli oggetti di Inform, quasi statica, sebbene siano creati dinamicamente in memoria come tutti gli altri oggetti Glk. Di base possono operare come contenitori di diverse variabili: il nome del file, la sua locazione, il tipo di file (di testo o binario). Un riferimento a File di default è gg_scriptref, ed è creato dalla libreria di Inform per la trascrizione del gioco.

Rock

Ogni oggetto glk definito nel gioco deve possedere un numero identificativo unico e personale, una pietra miliare, che permetta di riconoscerlo. Il loro uso è ampiamente spiegato in questo volume. Alcune costanti rock sono però riservate agli oggetti glk di libreria e non vanno usate.

Costanti	Valore	Assegnato a	Tipo di Oggetto
GG_MAINWIN_ROCK	201	main text window	windows
GG_STATUSWIN_ROCK	202	status line	
GG_QUOTEWIN_ROCK	203	quote box	
GG_SAVESTR_ROCK	301	save file	file streams
GG_SCRIPTSTR_ROCK	302	transcript	
GG_COMMANDWSTR_ROCK	303	command file (debugging)	
GG_COMMANDRSTR_ROCK	304	(ditto)	
GG_SCRIPTFREF_ROCK	401	script file	file reference

Per seguire la convenzione della numerazione, nello scegliere nel vostro gioco i numeri rock per i vostri oggetti cominciate con la seguente numerazione: da 210 in poi per le finestra, da 310 in poi per i flussi e da 410 in avanti per i riferimenti ai file.

Specifiche GLK
Vedi GLK Spec.

Stream
Vedi Flussi.

Virtual Machine
In Italiano Macchina Virtuale, abbreviato in VM. Si tratta di un programma che girando su una determinata macchina si occupa di simulare il funzionamento di

un'altra macchina. Nel caso di Inform è, solitamente, identificato dall'Emulatore.

VM
Vedi Virtual Machine.

Window
Letteralmente Finestra. E' la zona dello schermo del computer nel quale il nostro programma è in grado di mostrare del testo o di disegnare delle immagini.

Word
E' la dimensione nativa dei dati usati da un computer. Si tratta di un dato di dimensioni fisse, composto da una serie di bit. Ci sono macchine (microprocessori) che lavorano con word di 8, 16, 32 o 64 bit.

Z-assembly
E' il linguaggio con il quale sono scritti gli opcode per la Z-machine.

Z-code
Sono le istruzioni della Z-machine. Vedi anche Z-assembly.

Z-machine
La macchina virtuale immaginaria creata dalla Infocom per far girare i propri giochi.

Z-Machine Standards Document
Si tratta di una serie di documenti che provano a dettare uno standard per l'implementazione degli interpreti della Z-machine.

Indice