

Guida Introduttiva a **DMDX**

versione 0.4

Guida introduttiva a DMDX di Giorgio Arcara
è distribuito con Licenza [Creative Commons Attribuzione - Non commerciale 4.0 Internazionale](https://creativecommons.org/licenses/by-nc/4.0/).

Introduzione

DMDX è un programma freeware (cioè a distribuzione gratuita), per esperimenti di psicologia. L'utilizzo del programma è libero. La sola cosa che chiedono gli autori è che, nelle pubblicazioni, venga citato il suo utilizzo (es. *"The experiment was run using the DMDX software developed at Monash University and at the University of Arizona by K.I.Forster and J.C.Forster."*). Il programma può essere scaricato dal seguente sito <http://www.u.arizona.edu/~kforster/dmdx/download.htm>

Questa guida introduttiva è stata scritta a partire dalle altre guide e tutorial disponibili, dalle informazioni reperibili dall' Help del programma e dalla mia personale esperienza.

Note su Installazione

Dopo aver installato DMDX nel computer, non è possibile utilizzarlo immediatamente. Bisogna prima configurare il programma sulla base delle caratteristiche del computer su cui è installato (scheda audio, scheda video, refresh dello schermo etc.). Per fare ciò si deve accedere al programma TimeDX, il cui collegamento viene creato al momento dell'installazione del programma.

La prima volta che si accede a TimeDX, il programma automaticamente propone di effettuare delle configurazioni. Eseguirle tutte, evitando modalità manuali e selezionando sempre la modalità di configurazione automatica.

Terminata queste prime configurazioni bisogna effettuarne delle altre di base:

Dalla voce di menu "Basic test"

- Selezionare **"Select Video Mode"** e impostare la stessa risoluzione che attualmente è impostata sul monitor. Per vedere quale è la risoluzione del monitor basta fare click dx sul desktop vuoto, quindi andare in "proprietà", da lì nella scheda impostazioni e vedere quale è la risoluzione (es. 800x600) e quanti bit di profondità del colore (es. 16 bit). Confermare premendo "Done"
- Selezionare **"Refresh Rate"**, lì premere "Do Test", quindi aspettare che il pc termini il test e quindi confermare premendo "Done".

Queste sono le principali impostazioni NECESSARIE, per poter utilizzare il programma

Introduzione

I programmi in DMDX vengono scritti in file chiamati **Item files**. Il termine item può essere visto come sinonimo di “trial”. In questo file vengono specificate tutte le caratteristiche degli item, cioè dei trial dell’esperimento.

Gli item files nella pratica sono semplici file di testo RTF che contengono le righe di comando che dicono al programma principalmente queste informazioni:

- *Quali stimoli presentare al soggetto*
- *Per quanto tempo presentarli*
- *Quando cominciare a registrare i tempi di reazione (RT)*
- *Che tipo di risposta aspettarsi*

Gli **item files** sono divisi in due sezioni:

la **header line** che è rappresentata da alcune righe nelle quali sono scritte le istruzioni generiche valide per tutto l’esperimento

le **item lines** nelle quali sono specificate le caratteristiche di ciascun item

Non bisogna confondere gli **item files** con gli **stimulus files**. Gli item files sono i programmi che vengono fatti girare su DMDX. Gli stimulus file sono quei file (es. file audio .wav o immagini .bmp) che vengono utilizzati nell’esperimento.

Item files

Abbiamo detto che gli **item files** sono i file dei programmi di DMDX . Gli item file sono in formato RTF. Questo significa che possono essere scritti in WORD o con WORDPad. L’importante è che il file venga salvato in formato RT, altrimenti DMDX non lo riconosce. Per salvarlo in RTF basta fare Salva con nome, e nel formato scegliere RTF o RichTextFormat.

Nella pagina a seguire è mostrato c’è il tipico aspetto di un item file di DMDX

```
<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16,
0> <id tastiera> <mr +spazio> <mnr +CTRL di sinistra>
<mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1> <ss
<Randomize>>
```

```
$ 0 "PREMI LA BARRA PER INIZIARE L'ESPERIMENTO";$
```

```
+101 <fd 125 > "#####" <fd 25>/ *"BANCO"<fd 181>/;
-101 <fd 125 > "#####" <fd 25>/ *"CANFO"<fd 181>/;
-102 <fd 125 > "#####" <fd 25>/ *"ERLA"<fd 181>/;
+102 <fd 125 > "#####" <fd 25>/ *"POSTO"<fd 181>/;
+103 <fd 125 > "#####" <fd 25>/ *"CORSO"<fd 181>/;
-103 <fd 125 > "#####" <fd 25>/ *"ARZO"<fd 181>/;
+104 <fd 125 > "#####" <fd 25>/ *"PERLA"<fd 181>/;
-104 <fd 125 > "#####" <fd 25>/ *"TIRCO"<fd 181>/;
-105 <fd 125 > "#####" <fd 25>/ *"SELFIE"<fd 181>/;
+105 <fd 125 > "#####" <fd 25>/ *"OSTE"<fd 181>/;
```

```
$0 "FINE";$
```

La prima parte di ogni programma (o item file) è rappresentata dalla **header line**. Nota che non bisogna andare a capo, nello scrivere i comandi della header line.

Nella header line sono specificate le prime istruzioni che valgono per tutto l'esperimento. Fondamentalmente i comandi della header line possono essere distinti in due tipi. Un tipo di comando (es. <cr>, <nfb>) non hanno un 'oggetto', nel senso che non va specificato niente oltre al comando stesso. Nell'altro tipo di header line invece deve essere specificato un qualche parametro (ad esempio <fd XX>, dove XX è un numero che va specificato).

Questo è un esempio di header line:

```
<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0> <id tastiera> <mr +spazio>  
<mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1> <ss  
<Randomize>>
```

Ogni sequenza di caratteri racchiusa dai simboli <> è un'istruzione. Anche se può sembrare un bel casino, in realtà è molto semplice. L'esempio di header line riportato qui sopra può essere usato come canovaccio per ogni esperimento. Basta copiarlo e incollarlo nel file RTF dell'esperimento. Naturalmente andranno cambiati certi parametri, ma come scheletro può essere adattato a vari esperimenti.

Adesso andiamo a vedere cosa significa ciascuna di queste istruzioni.

<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0> <id tastiera> <mr +spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1> <ss <Randomize>>

Il comando <azk> indica semplicemente che i file di Output del programma saranno file di testo ASCII. Prendete questa istruzione come oro colato e limitatevi a riportarla. Non ha un gran peso.

<azk> **<cr>** <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0> <id tastiera> <mr +spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1> <ss <Randomize>>

Il comando <cr> sta per Continuous Running. Senza questo comando DMDX effettuerà una pausa dopo ogni item (trial) e aspetterà un comando (es. premere la barra spaziatrice), per passare al successivo item (cioè al successivo trial). In alcuni casi può essere preferibile togliere l'istruzione <cr>, specie se i soggetti sono molto lenti a rispondere o si distraggono facilmente.

Ricapitolando se è presente questa istruzione il programma passerà automaticamente da un trial a quello successivo, se non è presente attenderà un'ulteriore istruzione per passare al trial successivo.

```
<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0> <id tastiera> <mr  
+spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1>  
<ss <Randomize>>
```

<fd XXX> è una sigla per “Standard Frame Duration measured in ticks”

Questo comando indica per quanto tempo ciascuno stimolo sarà presentato nello schermo. Quando viene immesso questo comando bisogna scrivere < fd XXX>, dove XXX è un numero. È importante notare che il numero che segue **non indica i millisecondi di presentazione**. Il numero rappresenta invece il numero di “tick” che andranno presentati.

Che cosa è un tick?

Un **tick** è un’unità utilizzata da DMDX per definire il tempo di presentazione nello schermo. Esso è il tempo di presentazione che impiega il raggio catodico per effettuare uno scan del monitor.

Siccome dipende dal monitor utilizzato, **la durata di un tick dipende dall’apparecchio in cui il programma gira!**

Per sapere quanto dura un tick (in un dato computer), si può

- Guardare la prima riga del file .azk creato dal programma accanto la voce “refresh”.
- Andare in TimeDX , tra i basic test selezionare “refresh rate”, lasciar effettuare il test e vedere il valore.

Supponiamo di avere un computer in cui la refresh rate (misurata tramite TimeDX) è di 16.61 ms.

In questo computer 1 tick = 16.61 ms, 2 tick = 33.22 ms, 3 tick = 49.83 etc. Quindi, se vogliamo che il nostro stimolo appaia un certo numero di ms (supponiamo 100 ms), per sapere quanti tick dobbiamo impostare basta che dividiamo il numero di ms desiderati per la refresh rate del monitor. Nel nostro caso $(100 \text{ ms} / 16.61 \text{ ms}) = 6.03$. Nel programma però dovremo mettere un numero intero. Nel nostro caso il comando sarà <fd 6>. Segue che il numero di millisecondi per cui sarà visibile la schermata non sarà esattamente 100 ms, ma un multiplo della refresh rate del monitor (nel nostro caso 99.66 ms) e quindi un’approssimazione.

```
<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0> <id tastiera> <mr  
+spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1>  
<ss <Randomize>>
```

“d” sta per Delay. Il valore di d indica il ritardo dalla fine di un item (trial) all’inizio del successivo. Anche in questo caso il numero non indica i millisecondi, ma i ticks. Nel caso qui sopra <d 0> indica che al termine di un trial, inizia immediatamente il successivo.

```
<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0> <id tastiera> <mr  
+spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1>  
<ss <Randomize>>
```

“t” indica il tempo di attesa di una risposta (più avanti si vedrà come se determina quando far partire il cronometro. Questa volta il tempo va espresso in millisecondi (e non in ticks). Nel nostro esempio il <t 1500> indica che DMDX aspetterà 1.5 secondi che il soggetto risponda. A seconda dell’esperimento potrebbe essere necessario impostare un tempo di attesa maggiore.

<azk> <cr> <fd 20> <d 0> <t 15000>

<vm 800, 600, 600, 16, 0> <id tastiera> <mr +spazio> <mnr +CTRL di sinistra>
<mpr +CTRL di destra> <nfb> <dbc 255255255> <s 1> <ss <Randomize>>

Questo comando indica le impostazioni di risoluzione del monitor. Il significato di questi numeri è il seguente:

- 800** numero di pixel orizzontali
- 600** numero di pixel verticali
- 600** numero di scan lines (va messo uguale al numero di pixel verticali)
- 16** numero di bit di colore
- 0** questo valore specifica il refresh rate impostato. Se è lasciato a 0, allora DMDX utilizzerà il refresh rate utilizzato per default dalla scheda video (Si suggerisce di lasciarlo a 0).

È molto importante che questi valori corrispondano a quelli impostati. Per vedere quali sono quelli correntemente impostati basta fare click dx sul desktop vuoto, quindi andare in “proprietà”, da lì nella scheda impostazioni e vedere quale è la risoluzione (es. 800x600) e quanti bit di profondità del colore (es. 16 bit). Ricordate che il numero di scan lines è uguale al secondo numero della risoluzione (es. 600 nel caso di una risoluzione 800x600).

<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0>

<id tastiera> <mr +spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb>
<dbc 255255255> <s 1> <ss <Randomize>>

Il comando “id” indica il nome del dispositivo che viene rilevato per l’immissione di input. Nel nostro caso il soggetto premerà i pulsanti della tastiera quindi il nome di “id” è “tastiera”. Il fatto che il nome sia “tastiera”, non è casuale. Potrebbe essere che il vostro computer la riconosca invece come “keyboard”. Per vedere con che nome viene riconosciuta la periferica basta andare su TimeDX e tra i basic test vedere “input test”. Se accedete vedrete i nomi delle periferiche riconosciute dal programma (probabilmente saranno “mouse” e “tastiera”). Se voleste utilizzare il mouse, come dispositivo per le risposte, basta inserire <id mouse> nella header line. Se voleste sia il mouse, che la tastiera, basterebbe inserire <id mouse> <id tastiera>.

<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0>

<id tastiera> **<mr +spazio>** **<mnr +CTRL di sinistra>** **<mpr +CTRL di destra>** <nfb> <dbc 255255255> <s 1> <ss <Randomize>>

I tre comandi evidenziati sopra si riferiscono ai tasti utilizzati nell'esperimento. Quando si programma un esperimento con DMDX, occorre specificare quali tasti verranno utilizzati e in che modo. Andiamo per ordine.

Il primo comando è "mr". Questo comando sta per map request. Il tasto specificato in questo punto sarà il tasto da premere per avviare l'esperimento e se non è presente il comando <cr> (vedi sopra), per passare al trial successivo. Il comando è nella forma <mr xxxx>, dove xxxx è il nome del tasto scelto, preceduto da "+" o "-". Nel nostro esempio il tasto scelto è "spazio". Il "+" che precede spazio significa che il programma rileverà la risposta nel momento in cui il tasto verrà schiacciato. Se prima di "spazio" ci fosse "-", il programma rileverebbe la risposta nel momento in cui il tasto verrebbe rilasciato. Normalmente la risposta è considerata nel momento in cui il tasto è premuto quindi il più delle volte il comando sarà nella forma <mr +xxx>. Per vedere quale è il nome del tasto non possiamo sparare a caso, ma dobbiamo ancora una volta andare a vedere da TimeDX. Aprendo dai Basic Test, Input Test, selezionare il dispositivo di Input (es. tastiera) quindi premere Test. Dovrebbe apparire l'elenco dei nomi dei tasti. Per sapere che nome corrisponde al tasto che ci interessa basta premerlo sulla nostra tastiera. A quel punto sullo schermo verrà evidenziato il tasto che abbiamo premuto e il nome corrispondente. Il comando "mr" serve solo per passare da un trial all'altro o per iniziare l'esperimento. Le risposte date dal soggetto vanno date mappando altri tasti.

I comandi "mpr" e "mnr", stanno per "Map positive response", "Map negative response". Fondamentalmente DMDX è progettato per esperimenti dove esistono due tipi di risposte: positiva e negativa. Questo non significa che un tasto indica la risposta corretta e l'altro quella sbagliata. Indica semplicemente che DMDX organizza le risposte in una maniera dicotomica. Il programma in genere prevede che il soggetto dia una risposta a scelta tra due (che potrebbe essere "destra o sinistra", "alto o basso", "congruente o incongruente"). Se vogliamo un esperimento con tempi di reazione nel quale è previsto l'utilizzo di un solo tasto, basta inserire solo uno dei due comandi ("mpr" o "mnr"). Anche in questo caso dobbiamo specificare il tasto mappato (vedi sopra) e quando verrà rilevata la risposta scrivendo, prima del nome del tasto "+" o "-" (vedi sopra). Le procedure da seguire sono analoghe a quelle già spiegate per "mr".

NOTA: è possibile mappare più di un tasto per la stessa funzione, ma non lo stesso tasto per funzioni diverse. Esiste anche la possibilità di prevedere un tipo di risposta non dicotomica (es. quattro possibilità), ma richiede un procedimento un po' più complesso.

<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0>

<id tastiera> <mr +spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> **<nfb>** <dbc 255255255> <s 1> <ss <Randomize>>

Quando il comando "nfb" ("no feedback") viene incluso nella header line, allora non viene dato al soggetto un feedback su tempi ed errori. Se questo comando non è incluso dopo ogni trial apparirà una schermata che informa sul tempo di reazione appena effettuato e sull'accuratezza.

In particolare sullo schermo apparirà:

Correct 550

Per una risposta corretta con un RT di 550 ms

Wrong Per una risposta sbagliata

Siccome queste informazioni sono raramente utili (e distraggono il soggetto) il più delle volte il comando “nfb” sarà incluso.

```
<azk> <cr> <fd 20> <d 0> <t 1500> <vm 800, 600, 600, 16, 0>  
<id tastiera> <mr +spazio> <mnr +CTRL di sinistra> <mpr +CTRL di destra> <nfb> <dbc  
255255255> <s 1> <ss <Randomize>>
```

Il comando “dbc”, serve per definire il colore di sfondo di base per tutti i trial (“default background color”). Il comando dbc va seguito da un numero che indica (secondo il codice RGB) il colore. Apriamo una piccola parentesi il codice RGB è un codice per la definizione dei colori, L’acronimo RGB sta per RedGreenBlue. Quando si definisce un colore tramite il codice RGB si definiscono tre triplette, una per ciascun canale. Il valore che può assumere ciascun canale va da 000 a 255, all’aumentare della luminosità del colore (000 luminosità minima, 255 luminosità massima)

Quindi 255000000 è **rosso**
 000255000 è **verde**
 255255255 è **bianco**
 000000000 è **nero**
 e così via...

NOTA: Anche se nell’esempio non è specificato, è possibile utilizzare il comando <dwc xxxxxxxxx> per stabilire il colore di default con cui sono scritti i caratteri. Quindi se nell’header line si inserisce <dwc 000000000>, allora i caratteri degli stimoli saranno di colore nero (questo aspetto sarà più chiaro quando si parlerà delle item lines).

Item lines

Abbiamo visto come nella header line vanno specificate tutte le caratteristiche dell'esperimento.

L'altra parte, fondante un programma di DMDX, sono le cosiddette item lines. Le linee in cui sono specificati i trial dell'esperimento.

Cos'è un item?

Un item consiste in una serie di frame. Lunghezza totale di un item non può superare i 500 caratteri (sono più che sufficienti!). Ciascuna frame è delimitata dall'utilizzo di una slash ("/"). La slash indica al programma di cancellare la frame. Ogni frame indica una schermata separata.

La fine di un item line è segnata da un punto e virgola (",")

"Queste"/ "sono"/ "cinque"/ "frame"/ "separate"/;

NOTA: l'ultima "/" dell'item line (in questo caso quella dopo la parola "separate"), serve per cancellare l'ultima frame. Se non viene inserita l'ultima frame rimane nello schermo fino all'inizio della successiva item line.

All'interno di ogni frame può esserci del testo o dei comandi. Se i caratteri sono tra virgolette allora il programma li considererà come testo. Altrimenti li considererà come comandi. Il testo viene automaticamente centrato da DMDX.

Supponiamo di volere fare apparire nello schermo, in tre schermate separate, le parole "Evviva – la - neuropsicologia". Dovremo scrivere

"Evviva"/ "la"/ "Neuropsicologia"/;

Il testo può essere formattato direttamente utilizzando i comandi RTF. Se vogliamo quindi che la parola "neuropsicologia", sia diversa (ad esempio in grassetto e più grande) dalle altre schermate basterà formattarla opportunamente nel file RTF.

Es.

"Evviva"/ "la"/ "**Neuropsicologia**" /;

Se si vuole scrivere in più linee di testo, allora si deve aggiungere il comando @n dove n è il numero della linea di testo. 0 indica il centro dello schermo (@0), numeri negativi, le righe al di sopra dello 0 (@-1,@-2,@-3, etc.), numeri positivi (@, righe al di sotto. Ogni riga di testo va scritta tra virgolette ("") e deve essere separata dall'altra da una virgola.

Ad esempio

"Evviva" @0, "la" @1, "**Neuropsicologia**" @2;

Importante: il testo da mostrare deve essere separato da uno spazio dal codice in cui viene indicata la riga in cui il testo verrà mostrato (ad esempio “Evviva”@1) altrimenti il programma non va!!.

INDICATORE DELLA RISPOSTA CORRETTA (CR)

Il primo elemento di un item line è sempre l'indicatore di risposta corretta. Tramite questo simbolo diciamo al programma che risposta deve considerare come giusta o sbagliata (nota: per impostare a quale tasto corrisponde ciascuna risposta, vedi spiegazione della header line).

Esistono varie possibilità

+	La risposta mappata come positiva (<mpr xxx> è la risposta corretta
-	La risposta mappata come negativa <mnr xxx> è la risposta corretta
^	Non deve essere data nessuna risposta (es. compito go-no go)
=	Sia la risposta positiva che quella negativa sono corrette
!	Questa non è un vero e proprio indicatore di risposta. Se precede una item line allora DMDX ignora quella item line, che può essere utilizzata per commento.

NUMERO DELL'ITEM

Dopo l'indicatore della risposta corretta, la seconda informazione che dobbiamo mettere è il numero dell'item. Nel numerare gli item è importante utilizzare un criterio che permetta di distinguere item appartenenti a categorie diverse. Questo è particolarmente importante perché altrimenti diventa estremamente complicato fare le analisi ei dati a partire dall' output.

NOTA: Dopo il numero di item va *sempre* uno spazio bianco

Supponiamo di avere due condizioni sperimentali “parola” e “non-parola”.

Nel numerare gli item conviene fare in modo che item appartenenti a categorie diverse siano facilmente distinguibili.

Ad esempio potremmo numerare tutti gli item appartenenti al gruppo “parola”:

+101

+102

+103

+104

...

Mentre tutti gli item appartenenti al gruppo “non-parola”:

-201
-203
-204
-205

Si noti che in questi esempi di numerazione ho già incluso l'indicatore di risposta (in questo caso il programma si aspetterebbe una risposta positiva nel caso di item appartenenti al gruppo "parola" e una risposta negativa agli item appartenenti al gruppo "non parola")

La sequenza di item (casuale) verrebbe quindi qualcosa del genere:

+101
-201
-202
+102
-203
+103
+104
...

NOTA: Non numerate categorie diverse in questa maniera: 01, 02, 03 per il gruppo "A" e 001, 002, 003 per il gruppo "B". Gli "0" prima del numero non vengono considerati e quindi le due numerazioni sarebbero equivalenti (nello specifico 1, 2, 3).

PARTENZA DEL CRONOMETRO

Ovviamente l'aspetto più importante del programma è la possibilità di registrare tempi di reazione. Per segnare il momento in cui partono i tempi di reazione basta mettere (nella frame prescelta) un asterisco (*). Questo dirà al programma di aspettarsi la risposta da quel momento e di far partire il cronometro all'inizio di quella frame.

Esempio

+ 01 /"rispondi"/*"ORA"/;

In questo caso ci sarebbe una prima schermata in cui compare la parola "rispondi". E una seconda schermata in cui compare la "ORA". Nel momento in cui appare la seconda schermata ("ORA"), parte il cronometro e DMDX attende la risposta (in questo esempio positiva).

ESEMPIO DI ITEM LINES

A questo punto possiamo fare un breve esempio di item lines di un esperimento di scelta lessicale (parola o non parola).

+101 /"XXXX"/*"CASA"/;
+102 /"XXXX"/*"PANE"/;
- 201 /"XXXX"/*"FACO"/;

```
+103 /"XXXX"/*"MENO"/;  
-202 /"XXXX"/*"FISO"/;  
-203 /"XXXX"/*"PIRU"/;
```

In questo caso ogni trial consisterà in una prima schermata con una serie di XXXX. Quindi una schermata con la parola target, concomitante alla partenza del cronometro e all'attesa della risposta.

COMANDI NELLE ITEM LINES

L'esempio di sopra può essere un ottimo punto di partenza per spiegare l'utilizzo dei comandi nelle item lines. Nell'esempio di sopra la durata delle frame "XXXX" e "CASA", "PANE" etc. è sempre uguale e stabilita dai parametri dell'header line. Supponiamo che nella header line ci sia il comando <fd 30> allora tutte le frame dell'esperimento dureranno 30 ticks.

Supponiamo di volere che le frame con le XXXX durino un po' di più. Basterà inserire il comando all'interno della frame stessa. Tale comando varrà solo per la frame in questione.

Esempio

```
+101 / <fd 50> "XXXX"/*"CASA"/;  
+102 / <fd 50> "XXXX"/*"PANE"/;  
- 201 / <fd 50> "XXXX"/*"FACO"/;  
+103 / <fd 50> "XXXX"/*"MENO"/;  
-202 / <fd 50> "XXXX"/*"FISO"/;  
-203 / <fd 50> "XXXX"/*"PIRU"/;
```

NOTA: Alcuni comandi, anche se inseriti in una frame, valgono per tutto l'esperimento da quel punto in poi. Ad esempio il comando <cr>, può essere messo in una frame affinché valga da quell'item in avanti per il resto dell'esperimento.

SCHERMATE INIZIALI E DI PAUSA

Durante l'esperimento saranno necessarie schermate iniziali e di pausa. Queste schermate sono caratterizzate da avere, come item number uno "0". Per uscire da questo tipo di schermate DMDX si aspetta che venga premuto il pulsante mappato come "request" (<map request XXX>).

Esempio

0 "SCHERMATA INIZIALE";

```
+101 / <fd 50> "XXXX"/*"CASA"/;  
+102 / <fd 50> "XXXX"/*"PANE"/;  
- 201 / <fd 50> "XXXX"/*"FACO"/;
```

0 "PAUSA";

```
+103 / <fd 50>"XXXX"/*"MENO"/;  
-202 / <fd 50>"XXXX"/*"FISO"/;  
-203 / <fd 50>"XXXX"/*"PIRU"/;
```

NOTA: nelle schermate iniziali e di pausa non ho messo la "/" finale, altrimenti la scritta rimarrebbe sullo schermo solo per la <fd xxx> impostata nella header line. Inoltre non è necessario specificare altro perché il programma attenderà all'infinito che venga emessa la risposta prima di passare all'item line successiva.

BRANCHING E SUBROUTINE

Nei paragrafi di seguito sono elencate alcune classiche problematiche sperimentali che sono risolte, in DMDX, tramite l'utilizzo di subroutine e del branching. In breve le subroutine sono linee di codice che vengono eseguite solo in certe condizioni, mentre il branching è l'insieme di funzioni che rimanda a queste subroutine.

FEEDBACK

DMDX, per default, fornisce, dopo ogni item un feedback sulla risposta fornita dal soggetto. Questo feedback può essere eliminato con il comando <nfb> inserito nella header line.

Il feedback di base è:

Correct XXXX	(Risposta corretta a xxxx ms)
Wrong	(Risposta sbagliata)
No response	(nessuna risposta entro il tempo limite)

E' possibile utilizzare vari comandi (vedi elenco Feedback Keywords in DMDXHelp) per editare a piacimento il proprio feedback. In questa maniera si possono modificare vari parametri, come il testo che compare sullo schermo, la sua durata, la posizione
NOTA: gran parte di questi comandi NON VANNO INSERITI NELLA HEADER LINE, MA NELLE ITEM LINE! Una volta inseriti lì, (ad esempio nel primo item) saranno attivi per tutto il resto del programma.

FEEDBACK ACUSTICO

Se si vuole utilizzare un feedback acustico (ad esempio un "beep" per la risposta sbagliata), bisogna ricorrere ad un procedimento un po' più complesso che è quello di utilizzare delle *subroutine*.

Utilizzare delle subroutine significa utilizzare delle piccole parti del programma che verranno chiamate in causa solo quando richiesto.

Il seguente è un esempio per fornire un feedback acustico a seconda del tipo di risposta. Immaginiamo di volere un suono diverso a seconda che la risposta del soggetto sia giusta, sbagliata, o troppo in ritardo.

PASSI PRELIMINARI...

Innanzitutto dobbiamo avere i tre file wav dei suoni che vogliamo utilizzare e dobbiamo mettere questi file nella stessa cartella in cui si trova il programma (supponiamo che questi file si chiamino “corretta”, “errata”, “no risposta”).

Nella header line occorre lasciare <nfb> (no feedback). Questo perché non utilizziamo le procedure classiche per ottenere un feedback. Consideriamo un'ipotetica item line iniziale dell'esperimento

es. “ 0 **“PREMI SPAZIO PER INIZIARE”**;

occorre aggiungere il comando <bu XXX>, dove xxx indica il numero della prima item line (del primo trial) dell'esperimento. Verrà fuori quindi una cosa del genere:

“ 0 **“PREMI SPAZIO PER INIZIARE”** <bu XXX>;

Questo comando è molto importante perché dice al programma di andare direttamente alla item line con il numero XXX. Supponiamo che il primo item sia 101 dovreste mettere <bu 101>.

A questo punto bisogna creare una subroutine, cioè quella parte del programma che dice cosa fare in base.

In riferimento ai tre file wave prima citati (“corretta”, “errata”, “no risposta”), una possibile subroutine è la seguente.

```
~10 <biw 20>;  
0 <wav> "corretta" <return>;  
~20 <binr 30>;  
0 <wav> "errata" <return>;  
30 <wav> "no risposta" <return>;
```

Analizziamola passo per passo.

Il primo item è il numero 10, preceduto dal simbolo ~. Il simbolo ~ indica che lo schermo non deve mostrare nulla. Il <biw 20> ci dà un'indicazione condizionale. In particolare il comando <biw xxx> indica “se la risposta non è corretta vai all'item line numero xxx”. Oltre al comando <biw xxx>, esistono i comandi <bic xxx> e <binr xxx>, che rispettivamente dicono in che item line si deve spostare il programma se la risposta è corretta o se non è data in tempo.

Nel nostro caso il programma dice quindi “se la risposta non è corretta vai all'item line 20, altrimenti vai avanti. Supponiamo di avere dato la risposta corretta. Il programma andrà avanti e troverà la seguente item line:

```
0 <wav> "corretta" <return>;
```

Questa item line dice: “suona il file wav dal nome “corretta”, quindi ritorna da dove sei venuto”. (Per capire cosa significa ritorna da dove sei venuto, vedi dopo).

Se la risposta è sbagliata il programma dice di andare nella item line 20. Ma nella item line 20 c'è scritto “se non c'è risposta vai all'item line 30 altrimenti vai avanti”.

Arriviamo quindi alle altre due possibilità, perché se non abbiamo risposto il file andrà all'item line 30:

```
30 <wav> "no risposta" <return>;
```

che significa “suona il file wav dal nome “no risposta”, quindi ritorna da dove sei venuto”. Se invece la risposta è proprio sbagliata allora il programma va avanti alla item line

```
0 <wav> "errata" <return>;
```

che significa “suona il file wav dal nome “errata”, quindi ritorna da dove sei venuto”.

Ricapitolando, con un albero di scelte condizionali abbiamo previsto tutte le possibili opzioni.

Manca però qualcosa. Nella item lines che indicano i trial dell'esperimento, dobbiamo indicare di andare nella subroutine.

Supponiamo questa sia la prima item line

```
^101 “+” <fd 60>/ * ”9” / ;
```

occorrerà aggiungere un altro piccolo comando

```
^101 “+” <fd 60>/ * ”9” / <call -10> ;
```

Questo <call xxx> indica al programma di andare alla subroutine la cui item line iniziale ha il numero xxx. In questo caso era 10. Il – (‘meno’) posto prima del 10 sta ad indicare che il programma deve cercare all'indietro la item line 10.

Un risultato analogo è raggiungibile inserendo il comando <call xxx> nell'ultima frame della item line.

```
^101 “+” <fd 60>/ * ”9” <call -10> /;
```

In realtà non so bene quale è la differenza tra le due modalità, ma nella seconda la durata del trial dovrebbe essere ridotta.

NOTA IMPORTANTE 1: Nel generare subroutine bisogna stare molto attenti che le item line delle subotoutine siano numerate diversamente dalle item line del programma vero e proprio!!!

NOTA IMPORTANTE 2: Non bisogna scordarsi che anche le subroutine sono fatte da item line e che quindi risentono dei parametri indicati nella header line che se vogliono essere cambiati vanno specificate (fate molta attenzione al delay!!!)

ALTRO ESEMPIO:


```
~10 <biw 30>;  
20 <return>;  
30 <wav> "errata" <return>;
```

Questa subroutine (ovviamente inserita insieme agli altri opportuni comandi), fornisce un feedback facendo suonare il file “errata”, solo se la risposta è sbagliata. Se la risposta è giusta o non c’è risposta, non c’è alcun feedback

.

INSERIRE PAUSE DOPO UN CERTO NUMERO DI TRIAL

Un problema che si pone spesso nella preparazione di un esperimento è quello di mettere delle pause dopo un certo numero di trial. Questo non è un problema se gli item sono in una sequenza fissa. In questo caso basta mettere un' item lines preceduta dallo 0 con la schermata di pausa. Ad esempio:

```
+101 / <fd 50> "XXXX"/*"CASA"/;  
+102 / <fd 50> "XXXX"/*"PANE"/;  
- 201 / <fd 50> "XXXX"/*"FACO"/;  
0 "PAUSA – PREMI LA BARRA PER RIPRENDERE L'ESPERIMENTO";  
+103 / <fd 50> "XXXX"/*"MENO"/;  
-202 / <fd 50> "XXXX"/*"FISO"/;  
-203 / <fd 50> "XXXX"/*"PIRU"/;
```

La situazione diventa più problematica se invece si vogliono mettere delle pause dopo un certo numero di trial e al contempo si vuole che i trial siano randomizzati. In sostanza è necessario che si tenga conto di quanti trial sono passati per poi impostare una pausa dopo un certo numero di trial. Per fare ciò è necessario impostare un **contatore**, un numero che verrà modificato man mano durante l'esperimento. (Nota che i contatori potrebbero essere utilizzati anche in altri contesti, non solo per contare il numero di trial eseguiti). Oltre ad impostare il contatore è necessario utilizzare del codice che in parte è analogo a quello utilizzato per fornire dei feedback acustici. Nello specifico si crea una subroutine, che si fermerà nella pausa solo quando una determinata condizione verrà soddisfatta. Nello specifico faremo in modo che la schermata verrà mostrata quando il contatore avrà raggiunto certi valori.

La soluzione va fatta tramite tre passi:

- 1) Mettere una riga di codice all'inizio (ma non nell'header line) che faccia saltare le righe di subroutine (vedi dopo).
- 2) Scrivere la subroutine per la pausa.
- 3) Segnalare il punto in cui comincerà l'esperimento
- 4) Mettere in ogni item line alcune aggiunte di codice.

Di seguito c'è un piccolo esempio di file .rtf di un esperimento in cui ci sono delle pause dopo un certo numero di trial.

2009 - giorgio.arcara@unipd.it

```
<azk> <cr> <t 3000> <fd 50> <vm 1280, 800, 800, 32, 60> <id
RecordVocal> <id Mouse> <id Tastiera> <mr +Spazio> <mpr +Pulsante
0> <mnr +Pulsante 1> <nfb> <aikn +Pulsante 2> <dbc 200200200> <dwc
0000000000> <s 1> <ss <Randomize>>
```

```
$ 0 "BENVENUTO ALL'ESPERIMENTO" <set c1=0> <bu 100>;
```



```
~10 <bi 30, (c1 .ne. 3) .and. (c1 .ne. 6) >;
```

```
0 "pausa" <set c1=0> <return>;
```



```
30 <return>;
```


```
100 "" ; $
```



```
+101/<fd 50>/ <fd 100>"+"/*"UNO" <inc 1> <call -10>;
```

```
+102/<fd 50>/ <fd 100>"+"/*"DUE" <inc 1> <call -10>;
```

```
+103/<fd 50>/ <fd 100>"+"/*"TRE" <inc 1> <call -10>;
```



```
+104/<fd 50>/ <fd 100>"+"/*"QUATTRO" <inc 1> <call -10>;
```

```
+105/<fd 50>/ <fd 100>"+"/*"CINQUE" <inc 1> <call -10>;
```

```
+106/<fd 50>/ <fd 100>"+"/*"SEI" <inc 1> <call -10>;
```

```
+107/<fd 50>/ <fd 100>"+"/*"SEI" <inc 1> <call -10>;
```

```
$ 0 "FINE"; $
```

PUNTO 1)

Queste prime linee di codice effettuano due passi: settare il counter a 0, al valore 0. Siccome si possono settare più counter chiamiamo il counter c1 (che sta per counter 0). In codice <set c1=0>. Ricapitolando il counter avrà inizialmente il valore 0.

Segnalare a che punto andare. Con questa riga diciamo di andare direttamente all'esperimento, saltando le righe di subroutine. <bu 100>, sta per "vai alla item line 100". Questa riga serve per saltare la subroutine al primo giro.

PUNTO 2)

Queste righe di codice sono la famosa "subroutine" e servono per mostrare la pausa. Sostanzialmente queste righe dicono la seguente cosa: Se il counter non è pari a 3 o a 6, allora vai alla item line 30, altrimenti, vai alla item line successiva. Quindi quanto il counter assumerà un qualsiasi valore che non sia 3 o 6, la item line della pausa verrà saltata e si andrà direttamente alla item line 30. La riga <return> alla fine serve per ritornare alla item line da cui si è venuti prima di andare alla subroutine (vedi anche punto 3).

La parte di codice con ".ne." è un operatore logico, nello specifico è NOT (vedi nell'help di DMDX, sotto *setcounter* gli altri operatori logici).

PUNTO 3)

Nel punto 1) è stato messo il codice <bu 100> che manda a questa item line. Questa item line serve semplicemente per saltare, al primo ciclo, la subroutine. Da notare che è semplicemente item line in cui non viene mostrato nulla. In realtà per il codice dell'esempio questo passo è superfluo. Se la subroutine venisse effettuata comunque non accadrebbe nulla (si salterebbe all'item line 30).

PUNTO 4)

In ciascuna itemline dell'esperimento sono presenti due elementi chiave.

<inc 1> indica di aumentare il contatore di un'unità. Se all'inizio il contatore era stato settato a 0, dopo che verrà valutata la prima item line, il contatore sarà pari a 1. Dopo la seconda item line, il contatore sarà pari a 2 e così via. La seconda parte <call -10> dice di tornare indietro alla item line 10 e a eseguire la item line da lì (nel nostro caso, nella subroutine). Il "-" (meno) dopo "call" sta ad indicare che occorre tornare indietro alla item line 10.

Ricapitolando quello che accadrà sono le cose seguenti. Alla fine di ogni item line il contatore aumenta di uno, e si viene mandati alla subroutine. Se il contatore è un numero diverso da 3 o 6 allora si viene mandati direttamente all'item line 30, che rimanda da dove si è venuti. Se invece il contatore è 3 o 6 si va alla riga successiva che è quella della pausa.

UTILITIES

Le utilities sono piccoli software di complemento che semplificano l'utilizzo di DMDX e la gestione di alcuni aspetti.

Analyze

Analyze è un software che facilita la gestione dei file di output per l'analisi dei dati. DMDX genera un file di output che non permette un'immediata analisi dei risultati. Prima è necessario che i dati vengano rimaneggiati in maniera opportuna (tramite fogli di calcolo tipo Excel). Analyze semplifica questi passaggi.

Analyze lavora quindi sui file .azk (cioè i file di output degli esperimenti)

Caratteristiche di Analyze

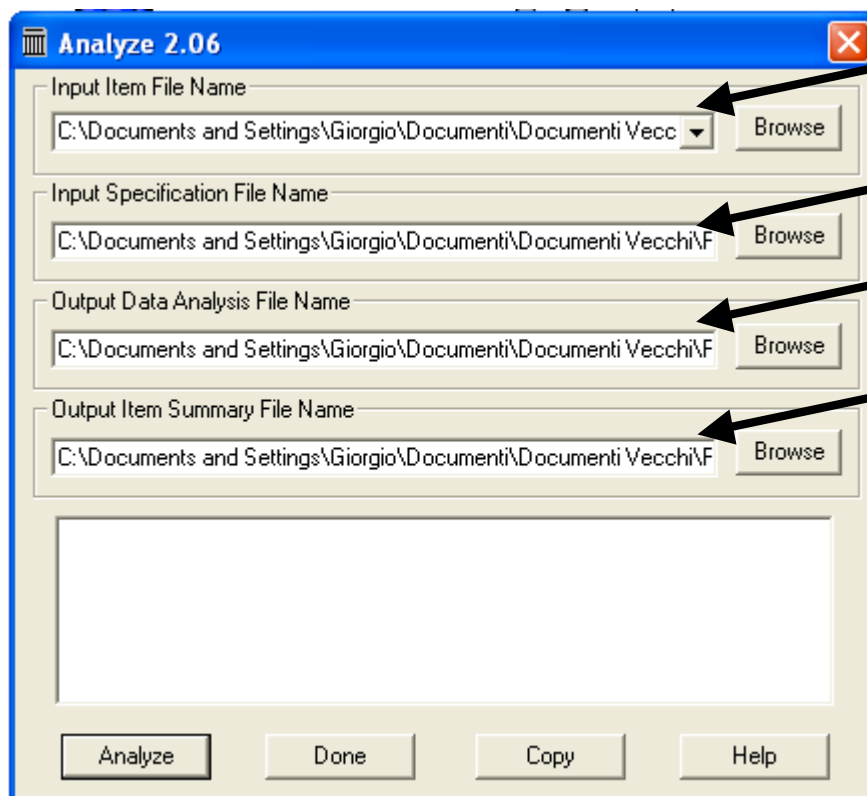
Analyze vuole due file di input ed ha due file di output

Se aprite il programma Analyze vedrete che ci sono quattro finestre. Nelle prime due deve essere specificato:

1. **Input Item file name:** qui va specificato (Premi Browse e specifica il percorso), il percorso del file del quale va fatta l'analisi (cioè il percorso del file dell'esperimento in questione)
2. **Input Specification file name:** qui va specificato (Premi Browse e specifica il percorso), il percorso del file .spc nel quale è specificato che tipo di analisi vanno fatte.

È possibile (ma non indispensabile) specificare quali siano i percorsi dei file di output.

3. **Output Data Analysis file name:** è un file di output per l'analisi dei dati
4. **Output Item Summary file name:** è un file di output che fa anche analisi separate per ciascun item



Se si vuole utilizzare Analyze il file dell'esperimento deve rispettare un requisito: **tutte le item line devono avere un numero diverso**. Ad esempio non è possibile utilizzare un file dove tutti gli item di un certo tipo sono contrassegnati dal numero 101. Se fosse così analyze prenderebbe solo il primo item con quel numero e tralascerebbe tutti gli altri. Se avete file con gli item contrassegnati in questo modo dovete modificarli, assegnando ad ogni item line un numero diverso (es. 101, 102, 103, 104)

I file .spc

Analyze è un programma che facilita l'analisi dei dati. Bisogna però specificare che cosa Analyze deve fare del file '.azk'. Questo viene specificato nel file .spc.

Per comprendere come utilizzare Analyze è utile rifarsi ad un esempio. Supponiamo di avere un item file di un esperimento con due condizioni sperimentali (A e B, Go-no go etc.).

```
Title: Analisi preliminari RT go-nogo
condition: 1
Low_cutoff:200
name: c1
description: item go
items: 101-103
condition: 2
Low_cutoff:200
name: c2
description: items nogo
items:01 02 03 04
```

Analizziamo una per una, le righe di questo file

Title: qui va specificato il titolo che si vuole dare al file. Non esistono vincoli per questa scelta

Condition: in questa riga si specifica la condizione che, da adesso in poi si vuole definire. Tutte le righe sotto questa si riferiranno a questa condizione, fino a che non verrà specificata una nuova condizione. Nel nostro esempio è stato messo condition: 1. I comandi sottostanti (low_cutoff, name, description, items) si riferiscono a condition:1.

Name: il nome della condizione. Anche in questo caso non esistono vincoli (deve essere solo breve)

Description: la descrizione della condizione. Anche in questo caso non ci sono vincoli

Low_cutoff: filtro associato alla condizione. Il numero che specifichiamo è il parametro del filtro taglia-basso. Se mettiamo low_cutoff: 200, tutti gli RT inferiori a 200 saranno esclusi dalle analisi

Items: tramite questo comando specifichiamo gli item che appartengono alla condizione in questione. Possiamo metterli uno per uno (es. 101 102 103) separando ciascun item da uno spazio. Oppure tramite intervalli (es. 01-04)

Per salvare un file nel formato .spc

Niente di più semplice. Abbiamo scritto i nostri comandi in un file normalissimo del blocco note. Fare salva con nome. Quindi come tipo selezionare tutti i file e nel nome file scrivere il nome del file prescelto aggiungendo '.spc'.

NOTA: Il nome del file .spc non deve necessariamente corrispondere con il nome dell'item, file dell'esperimento che si vuole "analizzare".

E Adesso?

Una volta creato il file .spc è possibile utilizzare Analyze.

Basterà selezionare nel percorso "Input Item file name", il file azk che vogliamo trattare.

Nel percorso "Input specification file name", il file spc che abbiamo creato.

Se vogliamo possiamo specificare destinazione e nome dei file .das e .ism che verranno creati come output.

Quindi premiamo il pulsante 'Analyze?', se non abbiamo commesso errori il programma ci dirà che i file di output sono stati creati correttamente.

Abbiamo detto che ci sono due file di output, **quello che ci interessa è il file .das**, nel quale ci saranno i file, divisi per soggetti e per condizioni. La prima colonna sarà riferita alla prima condizione, la seconda colonna alla seconda condizione e così via.

Questo è un esempio di file .das molto semplice, con tre soggetti e due condizioni sperimentali

SUBJECT RT Analisi preliminari RT go-nogo prova1 per analisi

1356 3000
1298 3000
1309 3000

SUBJ % ERRS

0.0 0.0

33.3 0.0

0.0 0.0

ITEM RT

1379 3000

1274 3000

1301 3000

**** 3000

ITEM % ERRS

0.0 0.0

33.3 0.0

0.0 0.0

**** 0.0

Quello a cui dovete prestare attenzione è la parte nel riquadro. Quelli sono gli RT per i soggetti 1, 2 e 3. La prima colonna è relativa alla prima condizione la seconda alla seconda condizione.

In questo formato sarà molto più semplice importare i dati in un programma per le elaborazioni statistiche.

CHECK VOCAL

Check vocal è un ottima utility che facilita la raccolta dati in esperimenti di denominazione in cui vengano registrati i tempi di reazione vocale. Check vocal (e una guida al suo utilizzo) può essere trovato in questo link:

<http://www.ilsp.gr/homepages/protopapas/checkvocal.html>

Per usare Check vocal è necessario avere le seguenti cose, che **devono essere nella stessa cartella in cui è presente il file CheckVocal.exe**:

- 1) Il file .azk contenente i risultati dell'esperimento.
- 2) Il file .rtf dell'esperimento.
- 3) Tutti i file .wav dei soggetti, contenenti le registrazioni all'esperimento.
- 4) un file che termina con l'estensione -ans.txt che contenga la risposta corretta agli stimoli e i corrispondenti delle item lines degli stimoli

BUG RISCONTRATI

BUG 1 Ho riscontrato un bug nell'utilizzare contemporaneamente linee di commento (quelle precedute dal punto esclamativo '!') e la randomizzazione (scramble).

In questo caso ho riscontrato dei problemi nella generazione della sequenza random di item, che tendeva ad omettere sistematicamente alcune item line.

Consiglio pertanto, prima di far partire un esperimento, di dare una scorsa alla sequenza per vedere se per caso c'è stato qualche problema nella generazione della sequenza random.

BUG2 *Questo bug sembra risolto nell'ultima versione di DMDX disponibile (14/06/2010).*

Se non si inserisce la funzione "scramble" (ad esempio < s 80 >, nella header line). Si potrebbero avere problemi nell'utilizzo del simbolo "\$" (è capitato di dare come errore "missing item number". Per risolvere il problema basta togliere questi simboli. Si noti che questi simboli sono irrilevanti, dal momento che comunque non c'è randomizzazione e che quindi le righe di comando vengono lette in ordine dalla prima all'ultima.

BUG3 Ho trovato qualche problema con l'utilizzo del simbolo \$ per non randomizzare delle item lines dello scrambling se le linee da non randomizzare sono contigue. In questo caso funziona il blocco della randomizzazione funziona solo se si mette un simbolo all'inizio delle linee da non randomizzare e uno alla fine. Non funziona mettendo i simboli all'inizio e alla fine di ogni riga.