Alma Mater Studiorum - Università degli studi di Bologna

FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica

SISTEMI DI CREAZIONE DINAMICA DI DOCUMENTI INTERATTIVI

Tesi di Laurea di:
Matteo Lancellotti

Relatore:
Prof. Paolo Ciancarini

Anno accademico 2003 – 2004 Sessione III

INDICE

1 SOMMARIO

Lo scopo di questa tesi è di analizzare il problema della generazione dinamica di documenti digitali, in particolare di documenti dotati di meccanismi di interazione con l'utente, proponendo alcuni approcci di soluzione. Questo ha comportato, in primo luogo, un'analisi dei principali formati digitali presenti sul mercato, considerando pregi e difetti delle loro caratteristiche tecniche in funzione della risoluzione del nostro problema.
Si è scelto come esempio dei contenuti dei documenti digitali prodotti il gioco degli scacchi, settore per il quale esiste già una fiorente editoria elettronica. Per integrare partite di scacchi nella documentazione è stato sfruttato un particolare formalismo, PGN, creato proprio per fornire una notazione delle partite semplice ed altamente portabile.
Alla luce del lavoro di ricerca svolto, quindi, sono stati sviluppati diversi applicativi software volti alla produzione di documenti digitali scacchistici in vari formati: ognuno di essi mette in luce aspetti diversi dello stesso problema, proponendo soluzioni altrettanto diversificate.
Il percorso logico che è stato seguito nella stesura di questa tesi può essere così riassunto:

- La nozione di documentazione elettronica ed i principali formati digitali (capitolo 2).

- Formulazione del problema di generazione di documenti digitali interattivi nell'ambito degli scacchi, illustrando alcune soluzioni già disponibili (capitolo 3).

- Introduzione al formalismo scacchistico PGN e come integrarlo in un software: analizzatore sintattico (capitoli 3 e 4).

- Creazione di documenti digitali interattivi a scopo di delivering: documenti statici e dinamici (capitoli 5 e 6).

- Creazione di documenti digitali interattivi automodificanti attraverso meccanismi di editing (capitolo 7).

- Conclusioni e sviluppi futuri (capitolo 8).

2 IL MONDO DELLA DOCUMENTAZIONE ELETTRONICA

2.1 Introduzione

Il termine documentazione elettronica, come noi lo intendiamo oggi, non identifica un oggetto univoco, ma una vasta famiglia di oggetti, aventi sì radici comuni, ma spesso con impieghi molto diversi tra loro.
Certamente le origini di questo ampio panorama vanno ricercate nel boom informatico degli anni ottanta, che ha portato ad un rapido sviluppo della tecnologia e di tutto ciò che vi ruotava attorno, fino ad arrivare ai giorni nostri. Esistono, però, idee e progetti inerenti all'argomento già nei primi anni sessanta:

- Nel 1962 Douglas Engelbart iniziò un progetto per lo sviluppo di un sistema in grado di aumentare la produttività degli operatori informatici, liberando il computer dalla schiavitù del calcolo numerico e dedicando le sue potenzialià anche all'elaborazione di testi (tra l'altro fu anche l'inventore del mouse).

- Nella seconda metà degli anni sessanta, Nelson lancia il suo progetto Xanadu [XAN] con l'obiettivo di costruire un sistema in grado di assicurare lo scambio di documenti in formato elettronico attraverso la comunicazione tra basi di dati, in modo assolutamente trasparente per l'utente. Nella visione di Nelson, Xanadu era la base di un universo informativo globale ed orizzontale - da lui definito *docuverse* (*docuverso*) - costituito da una sconfinata rete ipertestuale distribuita su una rete mondiale di computer. Allo stato attuale, il progetto Xanadu è ancora attivo e cerca di proporre una struttura alternativa per documenti ipertestuali, considerando l'attuale standard del World Wide Web carente sotto diversi aspetti.

- Alla fine degli anni '60 nascono i sistemi di gestione dei documenti digitali ed in questo panorama si inserisce IBM, progettando sistemi di catalogazione, archiviazione e pubblicazione di documenti di tipo legale. Nel tempo sorsero varie problematiche di intercomunicabilità tra i vari sistemi realizzati e questo introdusse l'esigenza, ai giorni nostri consolidata, di una standardizzazione dei documenti. Ciò

significa che ogni documento deve essere quanto più indipendente possibile dall'applicazione che lo manipola.

Nella lingua italiana una delle principali definizioni date alla parola documento è la seguente: "*Qualsiasi oggetto utilizzabile a fini di consultazione, ricerca, informazione; illustrazione, dimostrazione*". Alla luce di questa definizione vediamo che il panorama dei documenti elettronici può spaziare in molte direzioni:

- Immagini: sono indubbiamente l'esempio più semplice di documento, in quanto possono racchiudere in un unico blocco testi e rappresentazioni grafiche. Esistono numerosi standard di formati di immagine legati a diversi aspetti come la definizione (qualità dell'immagine stessa), la compressione (capacità di variare la memoria di massa occupata in funzione della definizione) e le tecnologie di rappresentazione grafica utilizzate (grafica vettoriale, rappresentazione a livelli, tecnologie 3D).

- Testi semplici: realizzati attraverso editor basilari, sono costituiti da sequenze di caratteri senza alcuna informazione aggiuntiva legata all'impaginazione o all'aspetto dei caratteri di stampa (font). Un esempio potrebbe essere il listato di un programma realizzato in un determinato linguaggio.

- Testi formattati: documenti che, oltre al testo, contengono informazioni aggiuntive legate all'impaginazione, alla specifica dei caratteri di stampa utilizzati ed agli effetti particolari dati al testo (grassetto, corsivo, colore, ecc.).
  Realizzati con editor avanzati, detti *word processors*, questi documenti possono essere spesso arricchiti da immagini, tabelle ed altre strutture più complesse. Un esempio pratico sono i libri elettronici (ebook) che attualmente stanno trovando larga distribuzione sulla rete.

- Multimedia: anche i suoni ed i filmati rientrano nella definizione di documento.

- Ipertesti: al momento sono forse la forma più diffusa di documento elettronico ed hanno come caratteristica principale la non linearità.

Un testo normale, infatti, deve essere letto dall'inizio alla fine perché abbia un senso logico, mentre all'interno di un ipertesto possiamo trovare collegamenti logici (link) ad altri ipertesti correlati e l'ordine di lettura può essere cambiato ad ogni interazione, senza perdere il significato generale (si segue un percorso logico anziché sequenziale).
Un altro punto di forza di questa tipologia di documenti sta nel fatto che, oltre ad informazioni testuali formattate, è possibile inserire immagini, suoni, filmati ed anche programmi realizzati con particolari linguaggi.

Ogni documento è caratterizzato da un proprio ciclo di vita, un insieme di fasi che il documento stesso attraversa nell'arco del tempo:

- Creazione: il documento viene generato attraverso un apposito editor.
- Distribuzione: il documento viene divulgato agli utenti per la consultazione.
- Consultazione: il contenuto informativo del documento viene preso in visione dall'utente finale.
- Modifica: il contenuto informativo del documento viene cambiato (l'eliminazione è un caso particolare di modifica).

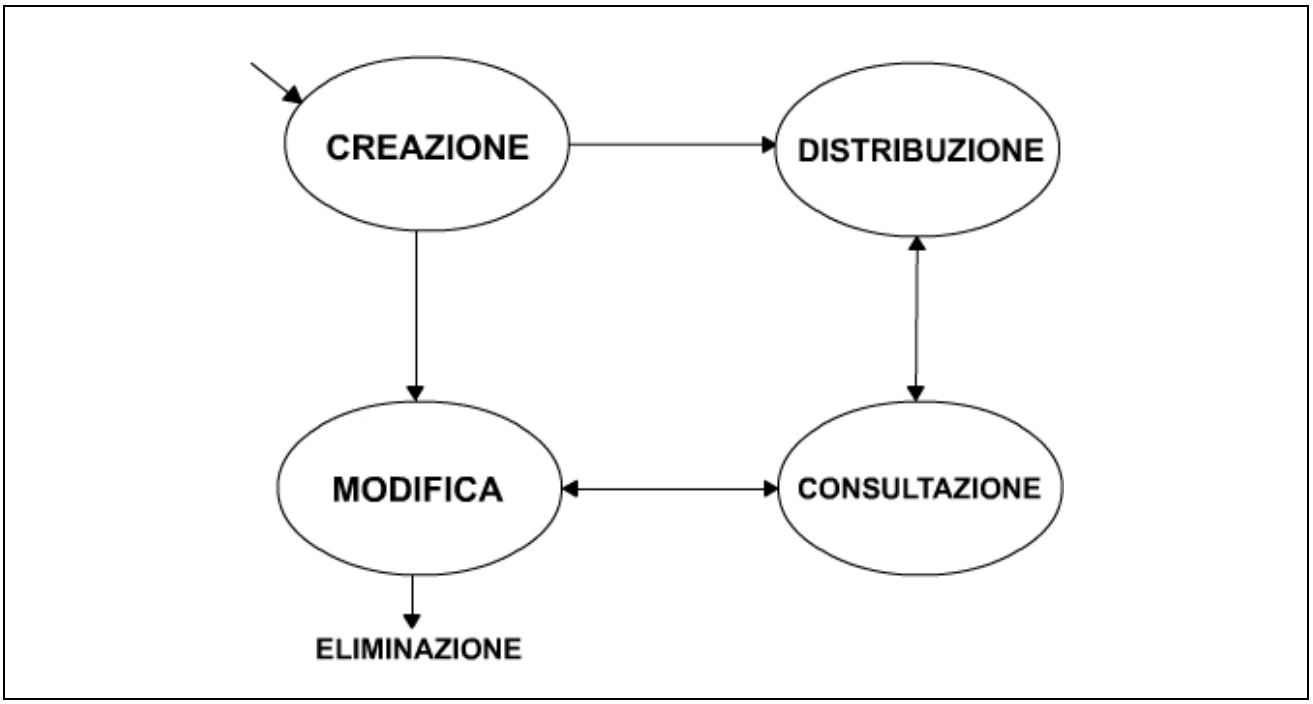


Figura 2.1: ciclo di vita di un documento.

2.2 Documenti attivi ed interattivi

2.2.1 Documenti attivi

Un documento attivo è un oggetto che presenta comportamenti autonomi, come ad esempio i filmati o le presentazioni Microsoft Power Point, senza però fornire all'utente la possibilità di interferire in tali comportamenti.
In definitiva, ciò che questi documenti offrono è una sequenza di informazioni che viene visualizzata in maniera predefinita e sempre uguale.

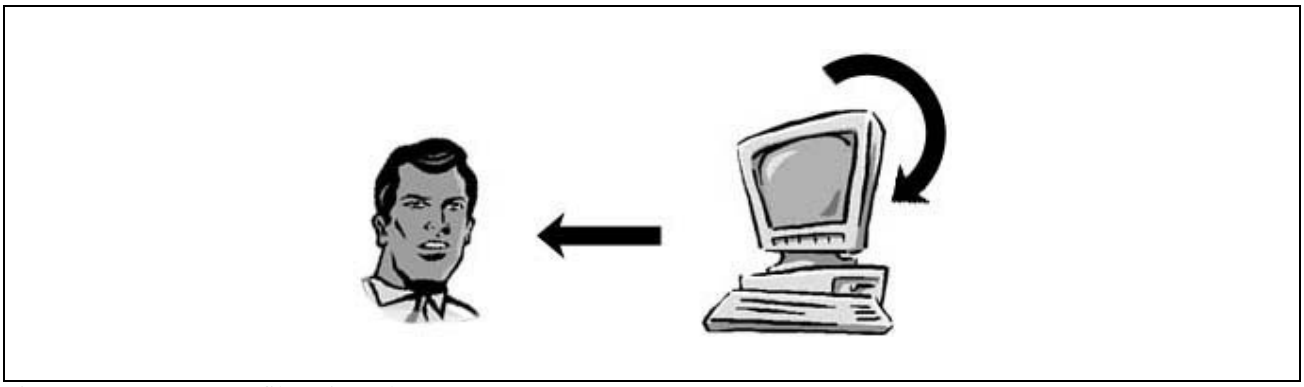


Figura 2.2: rapporto tra file attivo ed utente.

2.2.2 Documenti interattivi

Un documento si dice interattivo se permette la collaborazione dell'utente per portare a termine una determinata operazione, o se offre all'utente la possibilità di navigare attraverso le informazioni del documento stesso, visualizzando solo ciò che interessa e non forzatamente tutto il contenuto (come invece avviene nei documenti attivi).
Esistono vari livelli di interattività, vediamoli di seguito:

- Un primo livello è costituito dalla possibilità di accedere semplicemente ai contenuti, che a loro volta risultano essere documenti attivi. Un esempio è il servizio di Televideo: le pagine selezionate offrono una visualizzazione statica dei contenuti o, al più, una sequenza ciclica di sottopagine. Non esiste, per l'utente, alcun meccanismo di interazione coi contenuti delle pagine.

- Un secondo livello, invece, consente all'utente di scegliere un percorso di visita più complesso attraverso tutto il contenuto

informativo. Esempi noti possono essere i siti Web (attraverso link specifici si può saltare da una pagina all'altra del sito, richiedere dati specifici ad un database, ottenere o fornire informazioni attraverso l'utilizzo di form) o i DVD (comandi specifici per fermo immagine, avanzamento, indietreggiamento, scelta delle scene, ecc.).

- Una terza e più forte forma di interattività è legata al concetto di multi-utenza. Un esempio è costituito dai sistemi a database centralizzato: tutti gli operatori condividono tra loro i documenti, le fonti di dati, ed ogni volta che un operatore apporta una modifica a tali dati essa influisce anche sul lavoro di tutti gli altri operatori. Sistemi di questo tipo richiedono ovviamente rigide regole di condotta e frequenti sincronizzazioni tra gli operatori, onde garantire la stabilità del sistema (consistenza delle versioni).

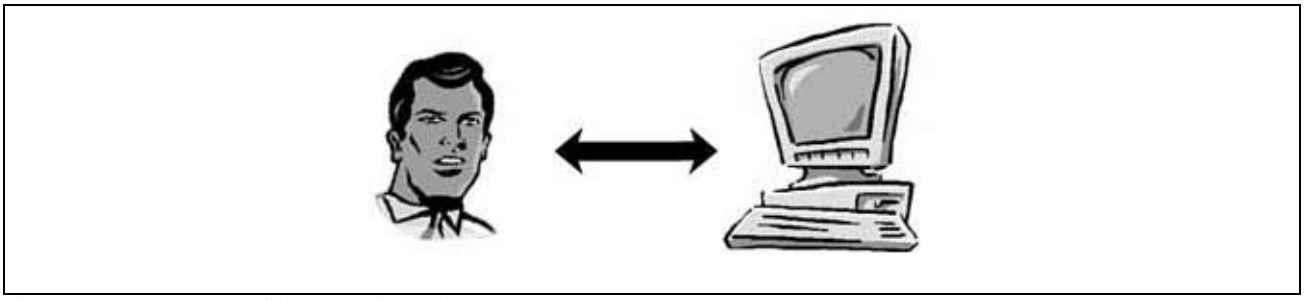


Figura 2.3: rapporto tra file interattivo ed utente.

2.3 Fonti e layout

Il layout di un documento definisce come questo si presenta agli occhi del lettore, cioè la disposizione, all'interno della pagina, dei testi e delle immagini che lo compongono.
L'aspetto finale di un documento va pianificato a priori, la disposizione di tutti i componenti e la scelta delle fonti tipografiche devono essere fatte in funzione della maggior nitidezza e leggibilità possibili.
Le moderne tecnologie ci offrono strumenti potenti per manipolare i nostri documenti, strumenti che hanno spesso una radice strutturale comune: un linguaggio di markup. Ogni documento digitale che contiene testo, infatti, viene rappresentato all'interno del sistema informatico in modo ben diverso da come noi lo vediamo:

- Un codice alfanumerico (lo standard dipende dalla nazione nella quale ci si trova) definisce la rappresentazione del testo del documento in forma di bit.

- Un codice di markup definisce la struttura del documento in funzione dell'applicazione che lo ha creato, oltre a riferire un sistema di fonti tipografiche. Questo codice è spesso inintelligibile, in quanto è in forma binaria.

- L'insieme di fonti tipografiche specificato definisce una rappresentazione intelligibile del documento in forma grafica, mediante caratteri stampabili a video o su carta. Siccome una fonte è visualizzabile solo se la stessa è installata sul computer in uso, bisogna scegliere fonti con una larga diffusione, per garantire la portabilità del documento (bisogna considerare, ad esempio, le differenze tra sistemi Windows e MacOS).

Per quanto riguarda le fonti tipografiche ne esistono numerose tipologie, ognuna delle quali prende il nome, ad esempio, dall'ideatore (Baskerville, Bodoni, Garamond, ecc.), dall'utilizzo originario (Times Roman fu inventato appositamente per la testata London Times), da caratteristiche particolari (Excelsior e Paragon furono introdotte per la loro alta leggibilità) e così via. L'unità di misura con la quale si stimano le dimensioni delle fonti è il punto tipografico, secondo la seguente uguaglianza:

1 pollice = 2,54 cm = 72 punti tipografici

Vediamo di seguito una rapida carrellata delle principali tecnologie e procedure che stanno alla base della creazione di una fonte:

- **Fonti bitmap**: ogni carattere è rappresentato attraverso una matrice bitmap. Questa struttura risulta essere per natura rigida ed infatti tali fonti non sono scalabili; per ogni dimensione richiesta esiste un archivio distinto che contiene la relativa fonte.

- **Fonti outline**: le forme geometriche che danno orgine alla fonte sono ricavate attraverso la combinazione di formule matematiche (in genere polinomi di secondo o terzo grado). Proprio per questa ragione le fonti outline risultano facilmente ridimensionabili e sono

più compatte in memoria, ma, in genere, necessitano di alcuni aggiustamenti per risultare tipograficamente gradevoli.
La figura 2.5 mostra un esempio dello studio geomentrico relativo ad una fonte outline.

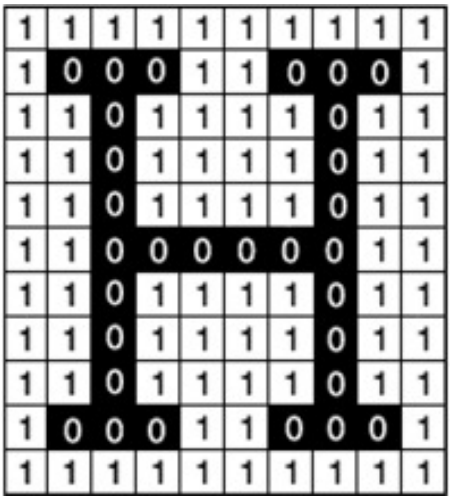


Figura 2.4: esempio di fonte bitmap.

- **Rasterizzazione**: questa operazione è utilizzata per ricavare dalla griglia dei pixel una fonte scalabile. Una volta individuata la forma del carattere (outline), infatti, si procede ad accendere i pixel che sono compresi all'interno dell'area delimitata dall'outline stessa.

- **Hinting**: la semplice rasterizzazione può non essere sufficiente a garantire una qualità grafica accettabile, specie nel caso di dimensioni piccole o basse risoluzioni dove si riscontra una scarsa densità di pixel disponibili per il rendering. Ecco perché le fonti sono accompagnate da informazioni aggiuntive, dette hints, che spiegano come modificare le forme dei caratteri sfruttando la griglia dei pixel disponibili. Queste informazioni quindi, introdotte nella fase di rasterizzazione, permettono di migliorare la forma e l'allineamento dei caratteri. In figura 2.6 è visibile la differenza nella visualizzazione di un carattere con o senza hinting.

- **Antialiasing**: questo procedimento è utile per ammorbidire i bordi delle immagini, eliminando il fastidioso effetto seghettato dovuto alla forma squadrata dei pixel. Normalmente si ottiene introducendo pixel di colori intermedi sui bordi interni ed esterni dell'immagine quindi,

nel caso dei caratteri di stampa dove ci troviamo a lavorare in bianco e nero, vengono aggiunti pixel grigi.

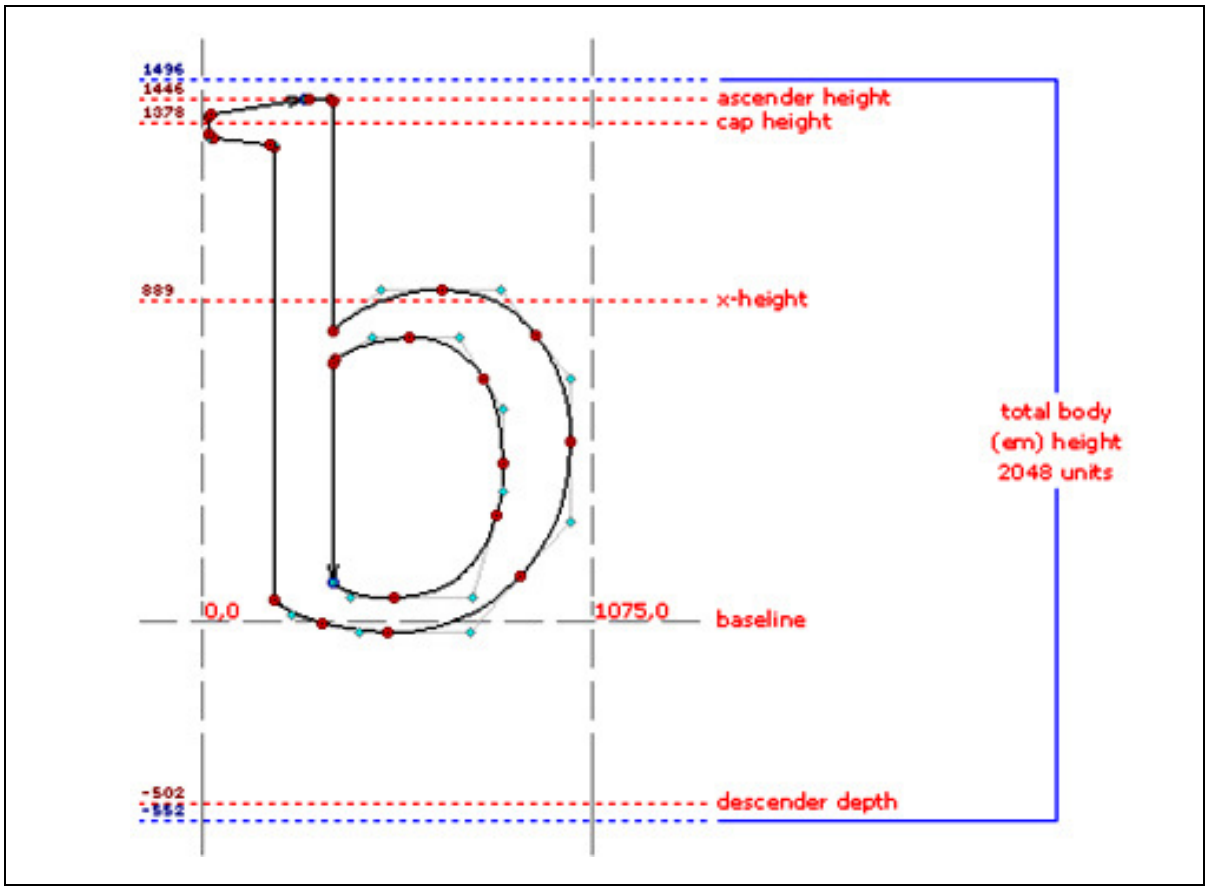


Figura 2.5: esempio di font outline.

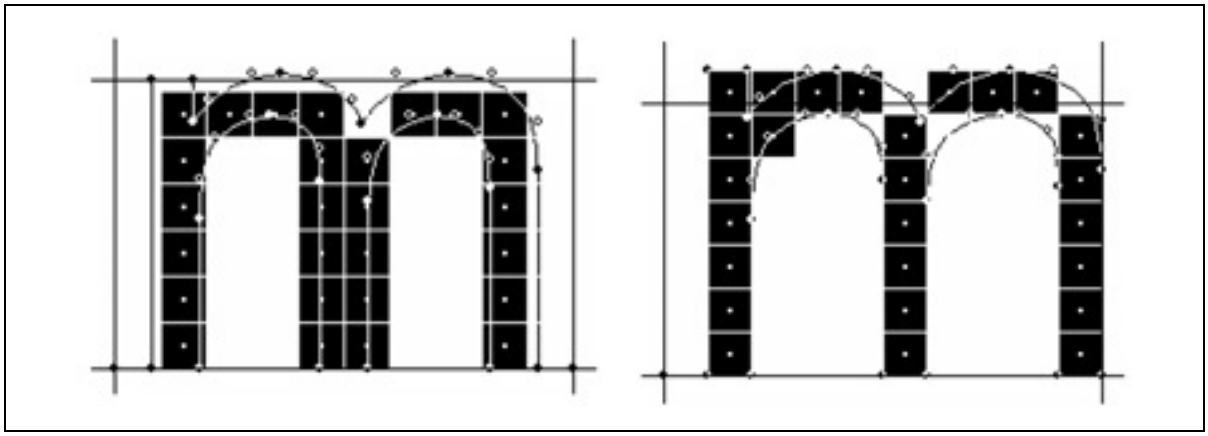


Figura 2.6: visualizzazione del carattere dopo la rasterizzazione: senza hinting (sx), con hinting (dx).

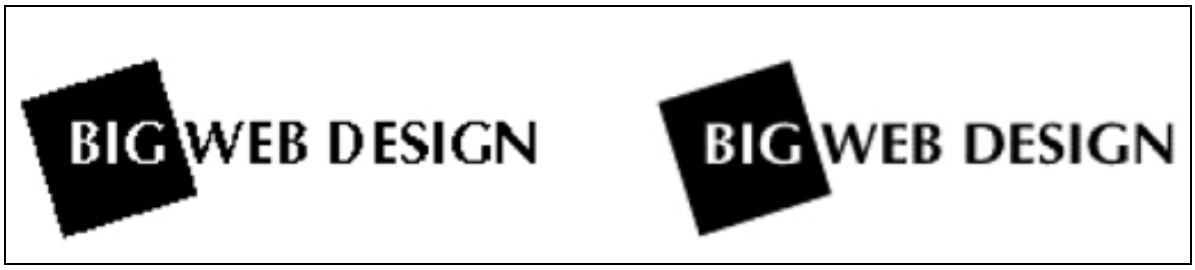


Figura 2.7: A sinistra visualizzazione affetta da aliasing, mentre a destra i bordi sono ammorbiditi dopo l'antialiasing.

2.4 Cosa offre il mercato

Vediamo di seguito una panoramica su alcuni dei formati di documenti digitali più diffusi nell'attuale mercato informatico, introducendo anche qualche cenno storico chiarificatore.

2.4.1 Il formato PDF

La prima comparizione di PDF avviene nel 1992 a cura di Adobe, che, sfruttando le già esistenti tecnologie PostScript ed Illustrator (editor grafico), presenta un prodotto nato dall'esigenza dell'azienda stessa di avere un formato di visualizzazione di documenti indipendente dall'hardware e dai sistemi operativi.
Il **Portable Document Format** (PDF) risulta quindi essere il successore di PostScript, o meglio un formato complementare a PostScript 3. Pur mantenendo la radice comune di indipendenza dai sistemi operativi, vanno sottolineate alcune differenze di fondo tra i due formati:

- Un file PostScript (.ps) è un programma, una sequenza di righe di comando volte alla stampa opportunamente interpretate da un RIP (Raster Image Processor) PostScript.
  Il fine ultimo di un file di questo tipo è la rasterizzazione di una bitmap su un qualche supporto di stampa, sia esso carta, pellicola o lastra. Gli elementi grafici e gli oggetti che li descrivono, contenuti nel documento, sono difficili da analizzare od estrarre fuori dal contesto del documento stesso: i documenti PostScript non sono fatti per essere editati.
  In buona sostanza PostScript è stato progettato per descrivere sequenze di pagine fisiche da stampare.

- Un file PDF è sostanzialmente un file PostScript già interpretato da un RIP e trasformato in oggetti grafici ben definiti, oggetti visibili sullo schermo, ma dal codice inintelligibile poiché binario. Siccome questi oggetti sono il risultato di una rasterizzazione, sono più affidabili in stampa rispetto ad un file .ps e sono, oltremodo, immediatamente visualizzabili su schermo, senza mandare effettivamente in stampa il documento.
  In generale, PDF è stato progettato per l'interscambio e la catalogazione di documenti.

Sottolineati quelli che sono i principi base vediamo ora le caratteristiche peculiari di PDF (intendendo la versione 1.5):

- **Codice PostScript semplificato**: il codice .ps ammesso in un documento PDF è semplificato, perché alcuni costrutti grafici originali del PostScript completo hanno un grado di complessità tale che molti rasterizzatori non sono in grado di gestirli.

- **Fonti Type 1 e TrueType**: la descrizione delle fonti e le relative informazioni di hinting sono incluse nel file, in modo che il fruitore del documento non abbia bisogno di nessuno strumento aggiuntivo per poterlo vedere od elaborare.

- **Bitmap pre-rasterizzate e compresse, e grafica vettoriale**: il fattore di compressione dei componenti del documento è alto; la grafica vettoriale riduce fino al 25%, senza che la nitidezza dell'immagine ne risenta e tra lo 0,5% ed il 75% per quanto riguarda la grafica bitmap. Tutti i documenti PDF sono scalabili (fino ad un massimo del 800%) e stampabili su stampanti sia PostScript che no.

- **Funzioni di indicizzazione**: in maniera immediata (link interni) consentono all'utente di visitare il contenuto informativo del documento in modo rapido e mirato. Utilizzati in modo più lungimirante (link esterni), permettono a PDF di diventare la base di un sistema integrato di gestione dei contenuti, poichè gli indici possono essere costruiti su più documenti gestiti da server diversi.

- **Funzioni di interazione (form)**: utili per raccogliere dati necessari allo svolgimento di una determinata operazione direttamente dall'utente.

- **Collegamenti a file sonori o Quicktime**: la suite software Acrobat di Adobe è un generatore e driver di presentazioni completo.

- **Collegamenti ipertestuali**: PDF supporta i link interattivi.

- **Descrizioni indipendenti di ciascuna pagina**: le pagine PDF entro lo stesso documento sono descritte indipendentemente le une dalle altre. In questo modo si possono combinare nello stesso documento pagine di diversa dimensione, si possono elaborare pagine singole e la trasmissione su rete può avvenire una pagina alla volta (il server in questione, però, deve saper gestire il byte-serving).

E' proprio grazie a queste sue caratteristiche di portabilità, integrità grafica, indipendenza delle pagine, compressione ed universalità che PDF si è diffuso così rapidamente e in una vasta gamma di applicazioni, tra le quali: trasferimento dei documenti, correzione di bozze, presentazioni multimediali, archiviazione, modulistica e sistemi di gestione di documenti digitali.
In considerazione del contesto di generazione dinamica di documenti interattivi nel quale questa tesi muove, è bene sottolineare i meccanismi di interattività offerti dal formato PDF. Alcuni di questi costrutti sono già stati menzionati precedentemente, ma vediamo comunque di analizzarli in maniera più approfondita:

- **Bookmarks**: i segnalibri sono un importante meccanismo di navigazione. Permettono sia di saltare da una parte all'altra del contenuto informativo di un documento (indice strutturato), sia di implementare collegamenti a fonti esterne al documento stesso (collegamenti ipertestuali).

- **Collegamenti ipertestuali**: possono essere incorporati direttamente nelle pagine del documento, senza l'utilizzo dei bookmark.

- **Form**: PDF dà la possibilità di inserire nei documenti campi testuali, caselle di scelta e quant'altro per la raccolta di dati o l'espressione di preferenze da parte dell'utente. Questi dati possono essere processati all'interno del documento stesso o da fonti esterne opportunamente specificate.

- **JavaScript**: funzioni scritte in questo linguaggio possono essere integrate all'interno dei documenti. La possibilità di utilizzare programmi attivi aumenta indubbiamente le capacità di interazione offerte, basti pensare all'uso congiunto coi costrutti form. E' necessario specificare, però, che JavaScript per PDF comporta alcune limitazioni e variazioni rispetto alla normale release del linguaggio.

Per concludere questa panoramica vediamo come un documento PDF può essere effettivamente generato. Ci sono molti metodi di creazione disponibili:

- **PDFWriter**: è un driver di stampante virtuale che genera documenti PDF a partire dal formato interno di una applicazione (GDI per Windows, QuickDraw per Macintosh). E' disponibile nella suite Acrobat di Adobe, ma esistono anche altri sofware disponibili gratuitamente che si basano sullo stesso principio, come ad esempio FreePDF [PDF1].

- **Acrobat Distiller**: sempre facente parte della suite Acrobat, questo programma è in grado di generare documenti PDF partendo da file PostScript.

- **Acrobat Capture**: software OCR che genera direttamente un output in formato PDF partendo da immagini di documenti acquisiti via scanner.

- **Linguaggi di programmazione**: data l'ampia diffusione di PDF, negli ultimi anni sono state progettate numerose librerie, per altrettanti linguaggi di programmazione, che implementano funzionalità mirate alla creazione e/o manipolazione di documenti PDF.

Per meglio comprendere l'evoluzione di PDF e delle sue caratteristiche tecniche è possibile osservare la tabella 2.1, che riporta la cronistoria del formato e dei software che ne hanno permesso la divulgazione.

Anno	Ver	Caratteristiche	Reader	Tool di creazione e manipolazione
1992	**1.0**	- Link interni - Bookmarks - Fonti incorporate - Supporto solo per RGB	Reader 1.0 esce nel 1993 al costo di 50$, successivamente distribuito gratuitamente.	Acrobat Distiller + Acrobat Exchange venduti a 695$ e 2.495$ rispettivamente versione personal e network.
1994	**1.1**	- Link esterni - Article threads - Funzionalità di sicurezza - Indipendenza dei colori dal dispositivo - Note	Reader 2.	Acrobat 2: Acrobat Exchange viene dotato di supporto per i plug-in e capacità di ricerca dei file PDF. Acrobat 2.1: supporto multimediale per aggiungere audio e video ai documenti PDF.
1996	**1.2**	- Supporto per la specifica OPI 1.3. - Supporto per CMYK. - Spot colours preservati nei documenti. - Funzioni per i mezzi toni ed istruzioni di overprinting integrabili nei documenti	Reader 3: viene realizzato un plug-in per visualizzare documenti PDF nel browser Web Netscape. Ora è possibile collegre file PDF a pagine HTML e viceversa.	Acrobat 3: Acrobat Exchange viene migliorato con numerosi strumenti che facilitano il processo pre-stampa.
1999	**1.3**	- Supporto fonti 2-byte CID. - Supporto per la specifica OPI 2.0. - Definito DeviceN, uno spazio colori che migliora il supporto per gli spot colours. - Smooth shading. - Annotazioni.	Reader 4.	Acrobat Exchange viene rinominato Acrobat. Acrobat 4: - Supporto per pagine fino a 5080 x 5080 mm di dimensione. - Webcapture. - Acrobat Distiller arricchito di configurazioni che facilitano la creazione di file PDF validi. - Integrazione in Microsoft Office.
2000	**1.4**	- Gestione trasparenze. - Sicurezza migliorata. - Supporto per JavaScript, inclusa la versione 1.5 e	Reader 5.	Illustrator 9 è l'unica applicazione capace di gestire PDF 1.4. Maggio 2001 esce Acrobat 5:

		miglior integrazione coi database. - "Tagged PDF", che fornisce informazioni strutturali sul contenuto del documento (meta-informazioni su titoli, blocchi di testo, ecc.)		- Visualizzazione corretta delle sovrapposizioni di stampa. - Esportazione da PDF a EPS-es in batch mode. - Distiller 5 può comprimere immagini che usano colori DeviceN. - Migliorato il motore di gestione dei colori sia di Acrobat che di Distiller. - Potenziamento delle funzionalità dei form. - Aumentati e migliorati i filtri per l'esportazione in altri formati: PDF to RTF ad esempio.
2003	**1.5**	- Miglioramento tecniche di compressione, includendo object streams e JPEG 2000. - Supporto per i layers. - Supporto migliorato per "tagged PDF"	Reader 6: incluse funzioni per il reperimento e la lettura di libri elettronici proprie di Adobe eBook Reader.	Acrobat 6 Professional: - Preflighting incorporato. - Ottimizzatore PDF. - Righelli e guide. - Job tickets. - Supporto per PDF/X. - Separation output e separation preview. - Transparency flattener. - Layers. - Measurement tool e loup tool. - Interfaccia grafica nettamente migliorata.

Tabella 2.1: cronistoria di PDF.

2.4.2 Il formato .LIT di Microsoft Reader

Il formato di file .LIT è legato ad un progetto che Microsoft ha sviluppato, e sta tuttora sviluppando, negli ultimi anni e che prende il nome di **Microsoft Reader** [LIT]. Questo software fa la sua prima comparsa sul mercato nell'aprile del 2000 e sostanzialmente è volto alla distribuzione e catalogazione di libri e documenti digitali su PC, noteBook e Pocket PC.
Questo lettore presenta diverse caratteristiche che lo rendono particolarmente interessante agli occhi degli utenti:

- E' gratuito.

- La sua interfaccia è appositamente progettata per permettere la lettura degli ebook online.

- Si basa sulla tecnologia di fonti digitali **ClearType**, attraverso le quali è possibile ottenere un'elevata nitidezza nella visualizzazione pur mantenendo dimensioni del documento finale molto contenute in termini di allocazione di memoria. Basti pensare che un libro di trecento pagine può essere contenuto in un file di 250 Kb anche se, ovviamente, la dimensione sarà sempre influenzata dal numero di immagini contenute nel documento stesso.
  ClearType è una tecnologia studiata appositamente da Microsoft per migliorare la leggibilità dei caratteri su schermi LCD. Si basa sul

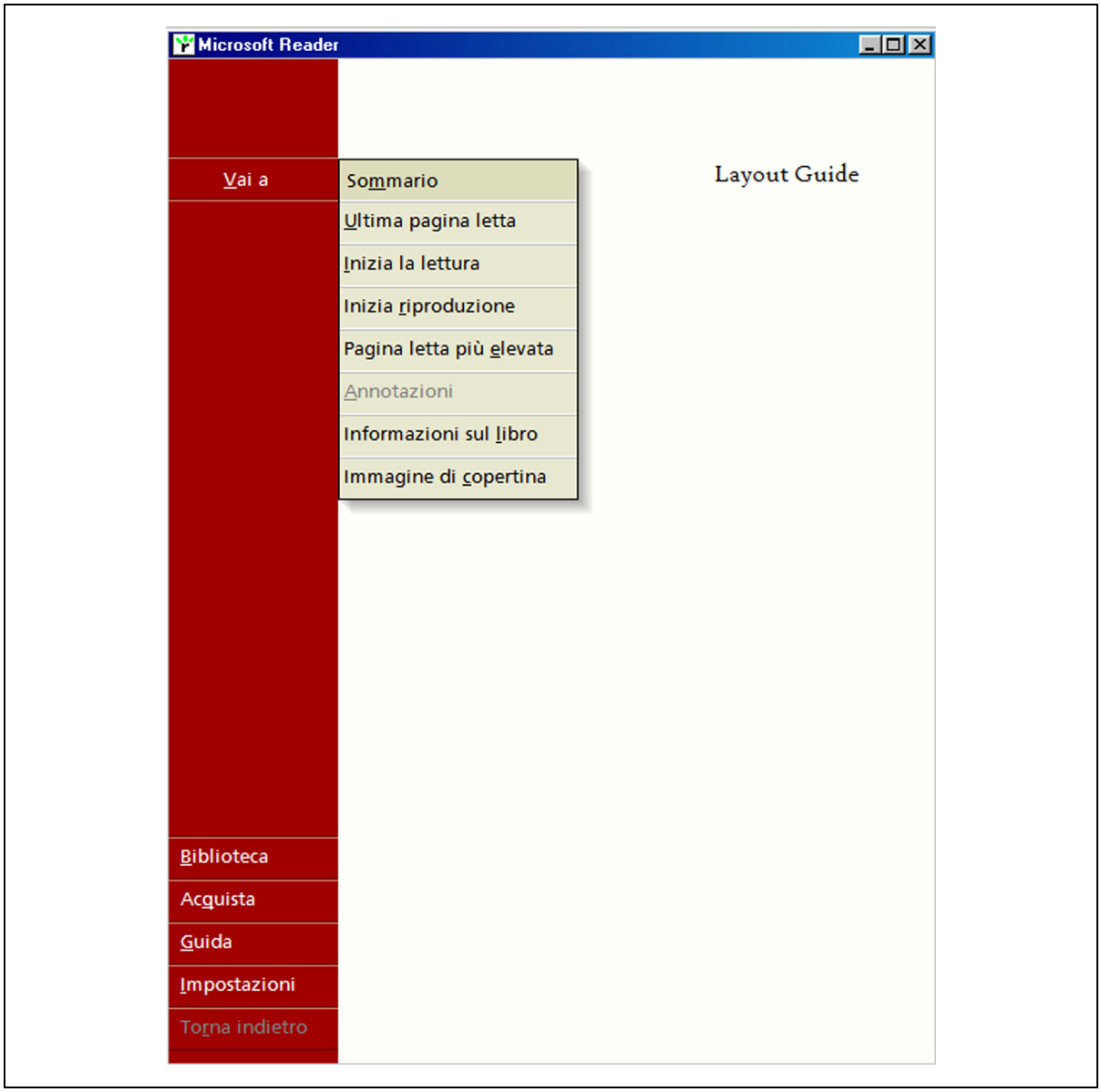


Figura 2.8: schermata di apertura di Microsoft Reader.

concetto di "sub-pixel", ovvero ogni pixel su uno schermo LCD è suddiviso in tre componenti: uno rosso, uno verde ed uno blu.

Se, prima di ClearType, il più piccolo livello di dettaglio che un computer poteva visualizzare era il pixel, ora è possibile visualizzare elementi di testo piccoli come frazioni di pixel (fino a 1/3 in larghezza). Questa caratteristica garantisce un incremento della risoluzione, specialmente per quanto riguarda i dettagli più piccoli.

- Permette di marcare ed annotare un ebook.

- Permette all'utente di personalizzare l'ambiente di lettura, fornendo opzioni inerenti alla grandezza dei caratteri, all'impaginazione, alla visualizzazione o meno delle immagini.

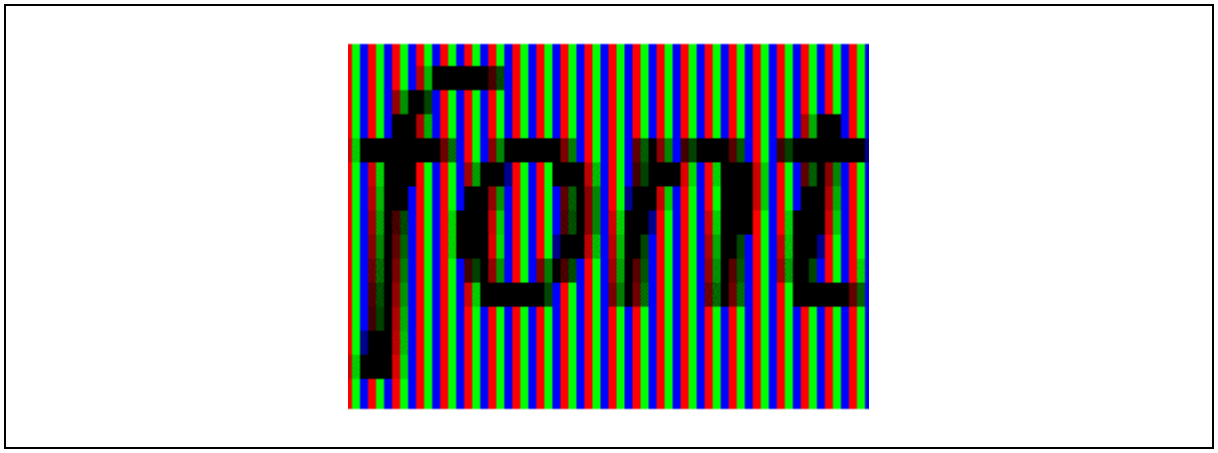


Figura 2.9: ingrandimento di font ClearType visualizzati su uno schermo LCD.

MS Reader si colloca in un panorama dove fondamentalmente non esistono standard, né per quanto riguarda la realizzazionone degli ebook, né riguardo le caratteristiche dei relativi lettori. I produttori di ebook sanno che la mancanza di un preciso standard a cui fare riferimento rallenta la diffusione delle pubblicazioni digitali, anche perché, allo stato attuale, esistono molteplici formati di pubblicazione tutti diversi tra loro e, nella maggior parte dei casi, tra loro incompatibili. Tutto ciò crea inevitabilmente confusione.
E' per questi motivi che Microsoft appoggia il progetto OeB (Open eBook), che da diversi anni sta cercando di proporre un valido standard per la rappresentazione delle pubblicazioni digitali e le metodologie di distribuzione relative. Il formato LIT ("Literature"), infatti, utilizza una specifica XML-based chiamata OeBPS (Open eBook Publication Standard) [OeB] curata da Open eBook Forum (OeBF), un'associazione composta da produttori hardware e software, editori, autori ed organizzazioni varie legate al mondo dell'editoria.

OeBPS consente di creare una singola rappresentazione elettronica di una pubblicazione, che può essere renderizzata da diversi dispositivi o in modo diretto, o attraverso una serie di trasformazioni intermedie, che portano alla creazione di un formato interno utilizzato dal particolare dispositivo. Questa specifica propone anche una propria terminologia relativa alle pubblicazioni ed ai dispositivi connessi, onde evitare le ambiguità introdotte dall'uso troppo generico del termine ebook:

- *OeB publication*: il contenuto digitale leggibile, cioè una versione "paperless" di un libro, un articolo od un qualsiasi altro documento.

- *Reading device*: applicativi o dispositivi utilizzati per renderizzare una pubblicazione OeB. Come esempio si possono citare i lettori dedicati agli ebook (come MS Reader), personal computer ed affini o palmari.

- *Reading system*: la combinazione di hardware e software volta a processare contenuti OeBPS e presentarli all'utente. Il reading system non deve per forza essere completamente contenuto nel reading device.

La specifica OeBPS, da sola, non è in grado di gestire le politiche sui diritti d'autore e la protezione delle copie, una caratteristica però molto importante nel campo dell'editoria, in particolare quando si parla di commercializzazione. E' proprio per questo motivo che OeBF si fonde, nel 2001, con EBX, un consorzio dedito proprio alla protezione del copyright nei libri elettronici.
In linea con queste necessità di salvaguardia dei diritti d'autore, il formato LIT è in grado di fornire diversi livelli di protezione dei documenti digitali, a seconda dell'utilizzo che ne deve essere fatto:

- *ebook sigillato*: normale livello di criptazione, standard in ogni documento. Il contenuto è accessibile a chiunque possieda MS Reader.

- *ebook autografato*: oltre ad essere sigillato, al documento vengono legati i dati del relativo acquirente. Per la lettura deve entrare in gioco un DAS (Digital Asset Server).

- *ebook a proprietà esclusiva*: oltre ad essere autografato, al documento viene unita la licenza d'uso, sempre attraverso l'utilizzo di un DAS, per fare in modo che il libro sia leggibile solo dal proprietario su macchine appositamente abilitate. L'abilitazione avviene attraverso la rete ed ogni utente può abilitare al massimo due lettori; tutti i documenti a proprietà esclusiva potranno quindi essere letti solo ed esclusivamente sui lettori abilitati.

A questo punto è necessario chiarire il concetto di Digital Asset Server: il DAS è il primo server che utilizza il linguaggio XrML, ovvero l'eXtensible rights Management Language, un linguaggio che serve a dare un metodo unico per specificare e gestire i diritti e le condizioni associate ad un contenuto digitale. Dal punto di vista dell'utente finale, fruire del sistema DAS è estremamente semplice, ma dal punto di vista dell'editore appare un sistema estremamente sofisticato e costoso. Questo è il motivo per cui, a tutt'oggi in Italia, solo la Mondatori ha attivato un proprio DAS center dedicato esclusivamente alla produzione dei suoi ebook, mentre le altre case editrici si appoggiano a società esterne direttamente collegate alla Microsoft.
Questi accorgimenti garantiscono l'integrità dei contenuti dei documenti ed oltremodo forniscono una piattaforma sicura per la commercializzazione dei libri elettronici.

Vediamo ora un esempio pratico di formalizzazione di un documento digitale secondo l'OeBPS: il **package file**. Le risorse necessarie alla realizzazione di un libro elettronico possono essere di diversa natura e provenienza, principalmente immagini e testi.
Tutto il materiale utilizzato per la creazione del documento LIT viene elencato e mappato in un unico file descrittore che prende appunto il nome di package file, realizzato utilizzando la formalizzazione XML. Il software di creazione del libro elettronico utilizza questo descrittore per reperire velocemente tutte le risorse necessarie e convertirle nel formato finale, considerando che l'ordine di comparizione di ogni singolo elemento nel package influisce sul relativo ordine di impaginazione.
I formati di immagine accettati dal software di conversione sono generalmente GIF, JPEG e PNG mentre, per quanto riguarda i testi, sono accettati file di testo semplice (.txt), file HTML a struttura semplice (gli script non vengono accettati, così come le tabelle annidate o i fogli di stile) e documenti Word.

```
<package unique-identifier="isbn">
  <metadata>
     <dc-metadata
       xmlns:dc="http://purl.org/metadata/dublin_core"
       xmlns:oebpackage="http://openebook.org/namespaces/oeb-package/1.0/">
          <dc:Title>Great Expectations</dc:Title>
          <dc:Contributor role="aui">Stanley Weintraub</dc:Contributor>
          <dc:Creator file-as="Dickens, Charles" role="aut">Charles
            Dickens</dc:Creator>
          <dc:Coverage>19th Century England</dc:Coverage>
          <dc:Date>July, 2000</dc:Date>
          <dc:Identifier id="isbn">0-451-52671-6</dc:Identifier>
          <dc:Language>en</dc:Language>
          <dc:Publisher>Penguin Putnam Inc.</dc:Publisher>
    </dc-metadata>
    <x-metadata>
         <meta name="ms-chaptertour" content="chaptertour"/>
    </x-metadata>
   </metadata>
   <manifest>
    <item id="content" href="Expect.htm" media-type="text/x-oeb1-document" />
    <item id="about" href="About.htm" media-type="text/x-oeb1-document" />
    <item id="coverstandard" href="GEcover.jpg" media-type="image/jpeg" />
    <item id="thumbstandard" href="GEthumbnail.jpg" media-type="image/jpeg" />
    <item id="titlestandard" href="GEtitle.jpg" media-type="image/jpeg" />
   </manifest>
   <spine>
     <itemref idref="content" />
     <itemref idref="about" />
   </spine>
   <guide>
     <reference type="toc" title="Table of Contents" href="Expect.htm#TOC" />
     <reference type="other.ms-coverimage-standard" title="cover image
       standard" href="GEcover.jpg" />
     <reference type="other.ms-thumbimage-standard" title="thumb image
       standard" href="GEthumbnail.jpg" />
     <reference type="other.ms-titleimage-standard" title="title image
       standard" href="Getitle.jpg" />
     <reference type="other.ms-firstpage" title="" href="Expect.htm#p1"/>
     <reference type="title-page" title="" href="Dickens_About.htm"/>
   </guide>
   <tours>
     <tour id="chaptertour" title="chapter tour">
         <site title="ch1" href="Expect.htm#Ch1"/>
         <site title="ch2" href=" Expect.htm#Ch2"/>
         …
        <site title="epilogue" href=" Expect.htm #Epilogue"/>
      </tour>
   </tours>
</package>
```
Figura 2.10: esempio di listato di package file

Consideriamo le voci principali che compaiono all'interno del package file:

- **Metadata**: forniscono informazioni relative al libro, come ad esempio l'autore, l'editore, la lingua e così via. Solo alcuni degli elementi metadata sono necessari in ogni package file, gli altri sono

opzionali; quelli obbligatori sono *identifier* (stringa alfanumerica che identifica univocamente il documento), *creator*, *title* e *xmlns* (identifica il namespace XML dove gli altri tag sono definiti).

- **Manifest**: tutti i file utilizzati per la creazione dell'eBook sono elencati qui, ognuno munito di un particolare identificatore (*id*) ed un attributo che ne specifica il tipo. Tali identificatori sono nomi assegnati a piacere, tranne per alcuni elementi particolari che vengono contrassegnati in modo ben preciso, come ad esempio l'immagine di copertina che verrà sempre identificata da *coverstandard*.

- **Spine**: riporta l'ordine di comparizione delle risorse nel libro. Tipicamente è una semplice sequenza di capitoli, ognuno dei quali può essere costituito da uno o più file. Tutte le risorse che compaiono nello spine (e solo quelle) vengono concatenate esattamente nell'ordine di elencazione nel libro finale (se il capitolo 10 fosse elencato prima del capitolo 2, nel documento finale si manterrebbe esattamente questo ordine).

- **Tours**: l'utilizzo di questo tag varia a seconda della versione di client MS Reader che si utilizza. MS Reader 1.0, ad esempio, non lo supporta affatto, mentre con MS Reader 2.0 si possono gestire anche "tours" multipli. Nello specifico, questo tag è utile per elencare i capitoli presenti nel libro e per poter saltare da un capitolo all'altro senza leggere tutti i precedenti. Il suo utilizzo, però, non è vincolato ed un editore potrebbe utilizzarlo per collegare una serie di illustrazioni, poemi o quant'altro al libro in analisi.

Ricollegandoci all'argomento interattività, vediamo che i meccanismi offerti da MS Reader sono ancora limitati. Allo stato attuale delle cose, infatti, un utente può interagire col documento LIT solamente attraverso tre costrutti:

- Collegamenti ipertestuali attraverso i quali è possibile raggiungere fonti esterne al documento. Questi possono essere integrati direttamente dentro le pagine del documento.

- Indici strutturati nei quali è possibile inserire anche collegamenti ipertestuali (vedere tag “Tours” sopracitato).

- Segnalibri inseribili in qualsiasi pagina del documento, attraverso i quali è possibile creare un collegamento promemoria ad una particolare pagina.

Per approfondimenti si invita a consultare la guida al layout di MS Reader reperibile sul sito ufficiale della Microsoft [LIT].

Per concludere, giunti a questo punto nasce spontanea una domanda: “Come si costruiscono i file LIT ? ”. Ci sono diverse possibilità in merito:

- Utilizzare MS Reader SDK (Software Development Kit) per integrare nelle proprie applicazioni la possibilità di creare file LIT. Microsoft, infatti, ha messo a disposizione degli sviluppatori la libreria *litgen.dll*, che offre tutte le funzionalità C++ indispensabili per la creazione di documenti digitali in formato LIT.

- Integrare a MS Word il plug-in, ovviamente sviluppato da Microsoft, che consente di salvare i documenti creati con il word processor direttamente in formato LIT.

- Utilizzare un software già esistente, come ad esempio ReaderWorks.

ReaderWorks [ORW] è un programma sviluppato dalla Overdrive (partner Microsoft) che consente di implementare libri elettronici in formato LIT attraverso un semplice layout grafico.
Esistono principalmente due metodi di creazione di un ebook con questo software:

1. Attraverso la tecnica “drag & drop” (o i menù standard) si selezionano tutti i file sorgente che andranno a far parte del libro. Come già visto in precedenza, sono ben accetti i file di testo semplice o anche HTML a struttura semplice; nel caso di file HTML contenenti immagini, queste ultime vengono estrapolate direttamente dal codice della pagina ed inserite nella lista delle risorse necessarie alla pubblicazione. Sono presenti anche menù per la compilazione

dei metadata, per l'inserimento di una copertina e per tutto ciò che normalmente compare in un package file.

2. Importare direttamente un package file (.opf) opportunamente scritto.

Una volta individuate le risorse, ReaderWorks può procedere alla conversione nel formato finale, individuando eventuali errori dovuti a risorse mancanti o collegamenti errati (ad esempio in una pagina HTML è presente un link ad una risorsa che non è stata elencata nella fase iniziale del processo).

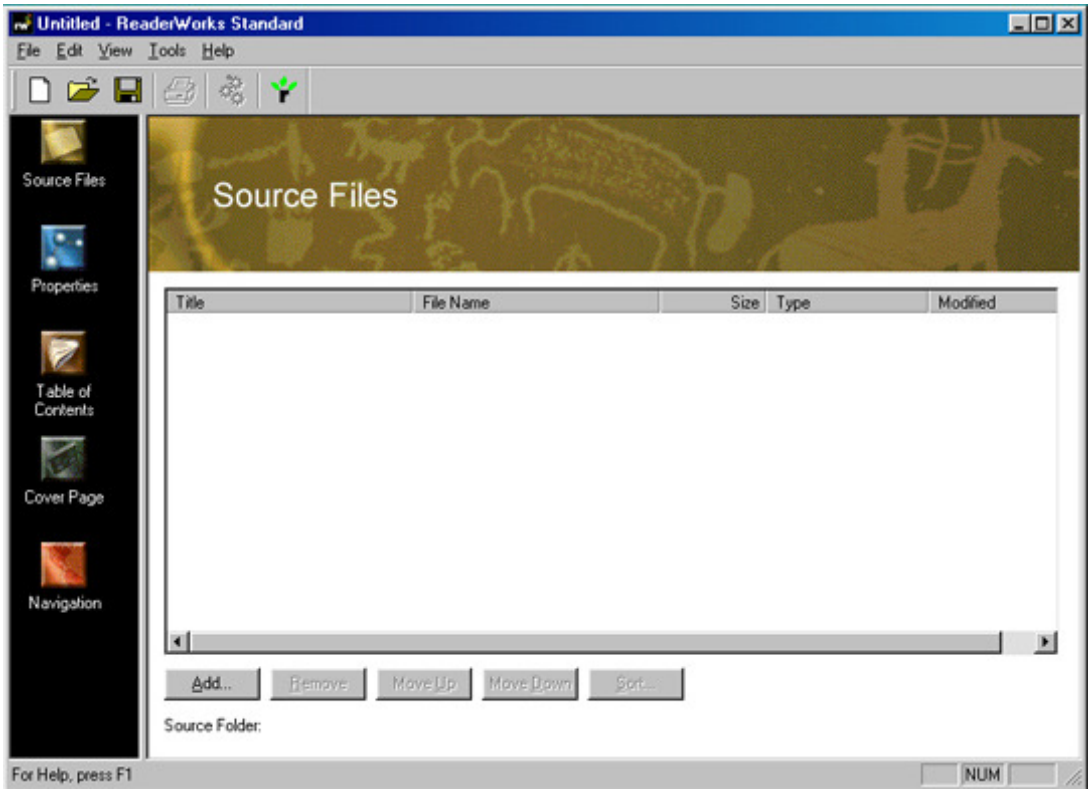


Figura 2.11: schermata di apertura di ReaderWorks della Overdrive.

2.4.3 Il formato HTML

Nel 1986 venne rilasciato un nuovo standard ISO (ISO 8879), con lo scopo di rendere irrilevanti le differenze di piattaforma e visualizzazione nella divulgazione ed il rendering dei documenti digitali. Questo standard forniva la descrizione dettagliata di un linguaggio chiamato SGML (Standard Generalizad Markup Language).
Nel 1989, Tim Barners-Lee propose un sistema di documenti ipertestuali da utilizzare all'interno della comunità CERN (Consiglio Europeo per la

Ricerca Nucleare). Il suo scopo era quello di avere a disposizione un potente e veloce metodo di comunicazione che favorisse le collaborazioni a lunga distanza, il reclutamento veloce di nuovi membri e la conservazione ed il reperimento veloce delle informazioni in modo pratico ed in grado di far fronte al veloce avvicendamento dei membri.
Nell' ottobre del 1990 Barners-Lee nominò il suo sistema "The World Wide Web" formalizzandone le componenti fondamentali, componenti che sono tuttora la base del moderno WWW:

- Deve essere indipendente dalla piattaforma.

- Deve essere capace di utilizzare la maggior parte dei sistemi informativi esistenti, permettendo anche, nel tempo, l'integrazione di nuovi sistemi in modo semplice.

- E' necessario un meccanismo di trasporto per spostare i documenti attraverso la rete (ciò che successivamente divenne il protocollo HTTP).

- E' necessario uno schema di identificazione ed indirizzamento dei documenti ipertestuali remoti (successivamente noto come indirizzamento URL).

- E' necessario un linguaggio di formattazione per gli ipertesti (HTML).

Barners-Lee sviluppò il linguaggio HTML partendo da SGML e pubblicò il suo lavoro, compresa una prima versione di browser e server web, su Internet stessa nell'estate del 1991. In questo modo si formò una sempre più popolosa comunità di sviluppatori che consentì, nell'arco di pochi anni, la creazione di numerosi progetti ed applicativi basati su HTML.
Gli standard HTML sono attualmente curati e sviluppati da un consorzio noto come il W3C.
Analizziamo ora più da vicino i meccanismi che entrano in gioco nell'interscambio di informazioni sul WWW e per farlo prendiamo in considerazione lo schema riassuntivo di figura 2.12.
Il concetto che risiede alla base della consultazione in rete di un documento HTML è una comunicazione di tipo client-server. Un sito internet, cioè l'insieme di uno o più documenti HTML correlati tra loro, è

fisicamente residente su un determinato server ed ogni server può contenere più siti, ad ognuno dei quali assegna un dominio (indirizzo internet, ad es: www.nonesiste.net). Ogni server è mappato sulla rete, in maniera univoca, da un indirizzo numerico detto IP ed ogni sito residente sul server è raggiungibile attraverso questo IP.

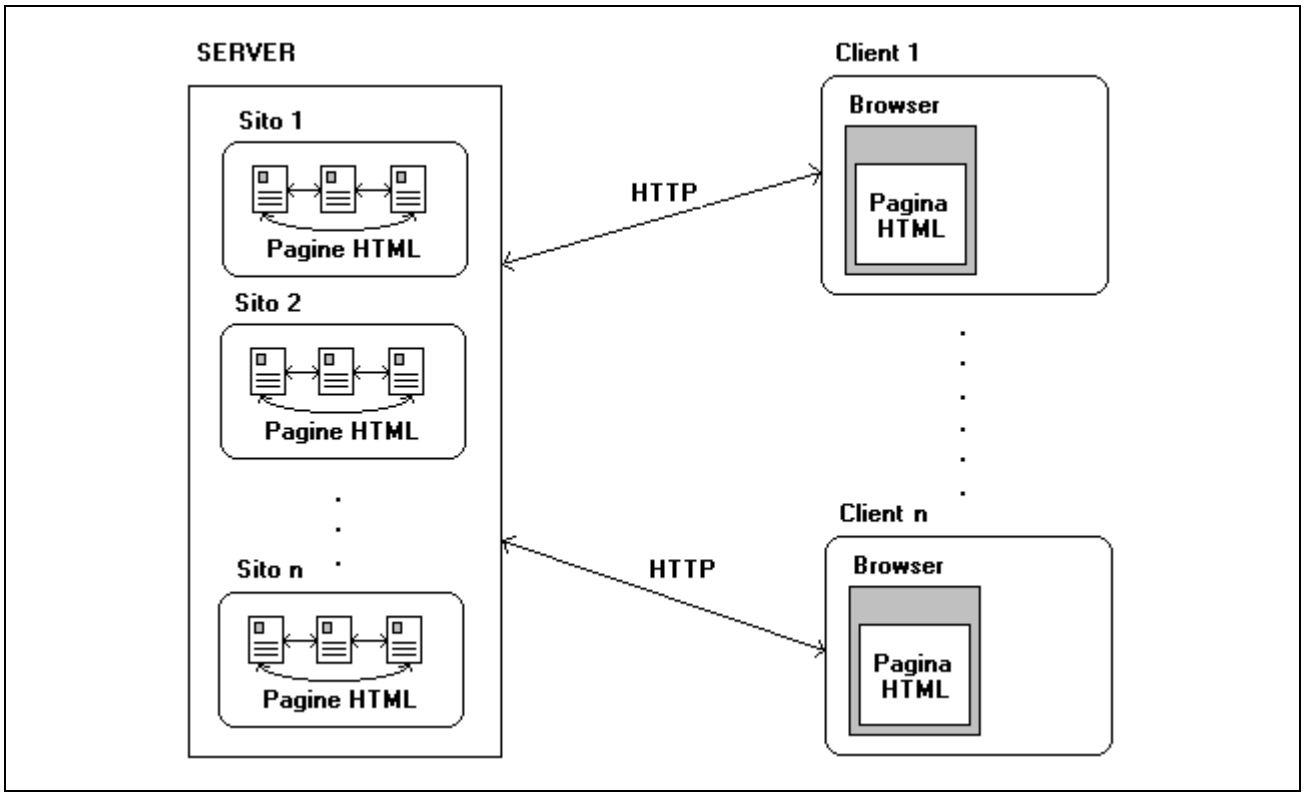


Figura 2.12: schema delle interazioni client-server per l'acquisizione di sorgenti HTML.

L'utente che vuole visionare il sito utilizza il suo web browser, richiedendo, però, non l'indirizzo IP del server (comunque difficile da ricordare), ma direttamente l'indirizzo internet del sito. Attraverso un particolare servizio, detto DNS (Domain Name System), l'indirizzo richiesto dall'utente viene convertito nel relativo indirizzo IP del server da contattare: sarà quest'ultimo ad individuare il dominio richiesto tra tutti i siti residenti sui suoi dischi.
A questo punto la connessione tra il server ed il client dell'utente, rappresentato dal web browser, è stabilita: attraverso il protocollo HTTP l'utente può richiedere al server la visione di determinati documenti ipertestuali ed il server risponderà inviando al browser le pagine HTML richieste.

Scendendo ancora più nel particolare vediamo come si presenta un file HTML sotto forma di semplice testo, considerando come esempio il listato riportato in figura 2.13.

```
<html>
<head>
 <title>Esempio di HTML</title>
 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<div align="center">
 <p><b><font face="Verdana,Arial,Helvetica,sans-serif"> Hello World !!</font>
 </b></p>
 <p><b><font face="Verdana,Arial,Helvetica,sans-serif">
       <a href="www.nonesiste.net">Link</a></font></b></p></div>
</body>
</html>
```

Figura 2.13: esempio elementare di listato HTML.

Come è ben visibile, ogni parte del documento (titolo, corpo, ecc.) ed ogni elemento che vi compare sono identificati da precisi tag che ne definiscono l'inizio e la fine, ad esempio: tutto ciò che compare tra *<body>* e *</body>* fa parte del corpo del documento, mentre i caratteri compresi tra *<title>* e *</title>* rappresentano il titolo del documento. In un comune web browser una struttura così scritta verrebbe visualizzata come in figura 2.14.
Attraverso specifici tag, scritti nella forma vista sopra, è possibile definire strutture di visualizzazione molto complesse, gestendo tabelle (anche annidate), immagini, fonti tipografiche e così via.
Nell'esempio visualizzato in figura 2.14 è possibile anche vedere ciò che maggiormente caratterizza un documento digitale di questo tipo: il collegamento ipertestuale, mappato in questo caso sulla scritta "Link", che consente all'utente di raggiungere altri documenti ipertestuali partendo da quello in visione. Utilizzando il sistema di identificazione univoca delle risorse (URL), HTML mette a disposizione un tag appositamente creato per la gestione dei collegamenti ipertestuali che, come è visibile nell'esempio precedentemente riportato, appare nella tipica forma

```
<a href="www.nonesiste.net">Link</a>.
```

Nell'ultima versione di HTML (Html 4.01 del 1999), formalizzata dal W3C, è stata introdotta una nuova caratteristica di grande utilità rispetto alle versioni precedenti: la capacità di gestire gli oggetti. Questa novità ha notevolmente rivoluzionato il mondo della rete, aprendo una porta verso il sempre più affermato universo delle tecnologie O-O: i documenti

ipertestuali, al giorno d'oggi, sono in grado di integrare animazioni avanzate, filmati, suoni ed applicativi appositamente creati per il web.

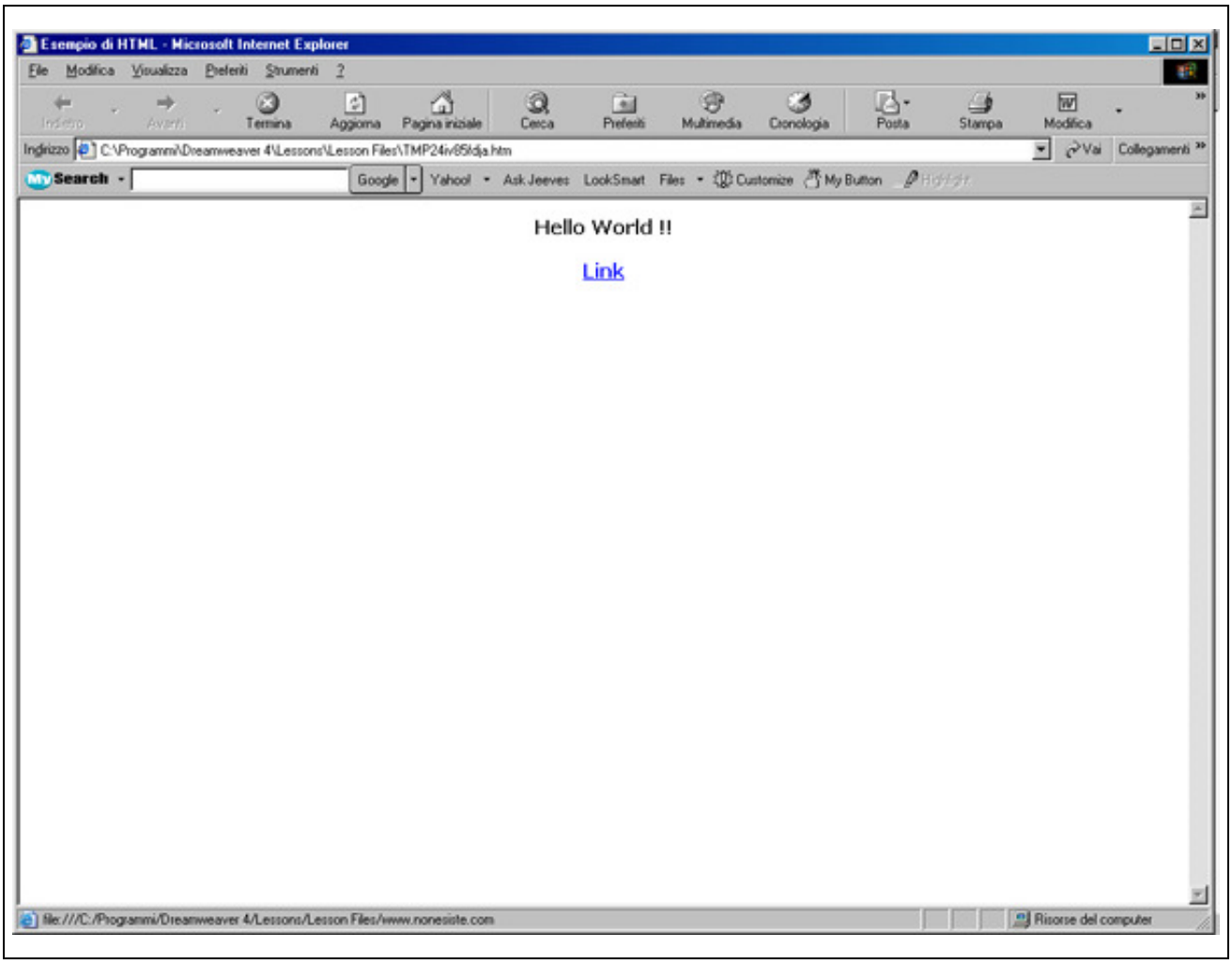


Figura 2.14: documento HTML di figura 2.13 renderizzato sul browser web IE 6.0.

Esistono diversi modi di generare un documento HTML:

- Editor di testo: attraverso un editor di testo basilare è possibile scrivere tutto il codice di un documento ipertestuale, riportando tutti i tag necessari alla visualizzazione tramite browser web od applicativi compatibili.

- Editor avanzati: in commercio esistono diversi sistemi di tipo WYSIWYG, applicativi ad oggetti che consentono di trattare in maniera grafica la composizione di un documento HTML in tutte le sue parti, generando automaticamente il codice relativo.

2.5 Conclusioni

Lo studio di sistemi che offrono la possibilità di interscambiare e catalogare informazioni in modo rapido e ampio trova le sue radici in progetti nati già negli anni sessanta. Allo stato attuale delle cose esistono numerosi formati digitali a disposizione del pubblico, ognuno dei quali è stato a suo tempo ideato per soddisfare particolari esigenze. PDF, ad esempio, risulta particolarmente interessante nel campo della divulgazione e catalogazione di documenti, mentre HTML è stato creato esclusivamente per la rete, per fornire un supporto leggero e potente all'implementazione di ipertesti. LIT di Microsoft Reader è stato ideato principalmente per la divulgazione dei libri elettronici e per questo presenta caratteristiche di estrema nitidezza dei contenuti e dimensioni limitate in termini di memoria.
La lista potrebbe proseguire ancora per molto, ma l'aspetto fondamentale che accomuna la maggior parte dei formati digitali è la risposta ad una precisa esigenza: l'indipendenza dalla piattaforma. Questo significa che un documento deve essere quanto più indipendente possibile dall'applicazione che lo ha creato e, in particolar modo, deve essere leggibile su qualsiasi dispositivo, indipendentemente dal sistema operativo su di esso installato.

3 FORMULAZIONE DEL PROBLEMA

Come già anticipato, lo scopo prefissato di questa tesi è l'analisi di soluzioni software atte alla generazione dinamica di documenti interattivi a carattere scacchistico. Questa scelta è stata basata su precise considerazioni:

- Il gioco degli scacchi è ampiamente diffuso nel mondo.
- Esiste già un settore di pubblicazioni digitali inerente all'argomento.
- Esistono già precise formalizzazioni volte alla rappresentazione di partite di scacchi, formalizzazioni sviluppate proprio per consentire la crescita degli applicativi software di settore.
- Esistono già numerose fonti tipografiche digitali per la rappresentazione dei pezzi sulla scacchiera.
- La letteratura inerente al gioco degli scacchi ha, per sua natura, un aspetto intrinsecamente interattivo. L'evoluzione statica dei diagrammi scacchistici in una pubblicazione porta immediatamente a pensare ai vantaggi di una rappresentazione dinamica della medesima sequenza, in cui i passaggi di visualizzazione sono scelti dal lettore stesso (concetto che si avvicina maggiormente all'idea di gioco).

Analizzando il problema generale in maniera più dettagliata è possibile evincere che esso è scomponibile in diversi sottoproblemi, ognuno dei quali può essere affrontato e risolto in maniera separata:

- Interpretazione dei problemi scacchistici o, più precisamente, integrazione dei formalismi che li rappresentano.
- Rappresentazione grafica dei problemi scacchistici, dipendente dai dispositivi o dagli applicativi scelti per la visualizzazione.
- Creazione di meccanismi di interazione con l'utente all'interno dei documenti stessi.

- Generazione dinamica dei documenti finiti e grado di dinamicità offerto dal prodotto finale (spesso legato alle limitazioni degli applicativi di visualizzazione).

Di seguito vengono introdotti alcuni concetti e delucidazioni per comprendere meglio la portata del problema e, di conseguenza, per capire meglio i ragionamenti e le scelte progettuali alla base delle soluzioni successivamente proposte.

3.1 Introduzione a PGN

PGN (Portable Game Notation) è una notazione standard per la rappresentazione di partite di scacchi che utilizza file di testo ASCII. Il suo scopo non è quello di fornire uno standard universale, ma piuttosto di creare una solida base volta a facilitare lo sviluppo di applicazioni mirate, in grado di trattare e manipolare in modo semplice e veloce partite e problemi scacchistici. In figura 3.1 è stato riportato un esempio di file PGN.

```
--INTESTAZIONE--
[Event "Open"]
[Site "?"]
[Date "1996.??.??"]
[Round "0"]
[White "Grigor Minchev"]
[Black "Nejad"]
[Result "1-0"]

--CORPO--
1. e4 e5 2. Nf3 Nc6 3. Nc3 Nf6 4. Nxe5 Nxe5 5. d4 Ng6 6. e5 Qe7 7. Bg5 Qe6 8.
Qf3 c6 9. O-O-O Ng8 10. h4 Bb4 11. d5 Qxe5 12. Bc4 Bxc3 13. bxc3 f6 14. Rde1
Kf8 15. Rxe5 Nxe5 16. Qe4 Nxc4 17. Qxc4 fxg5 18. hxg5 Ne7 19. Qf4+ Ke8 20. d6
Ng6 21. Qe4+ Kf7 22. Rh3 Rf8 23. Rxh7 Nf4 24. 2g3 1-0
```

Figura 3.1: come si presenta un file PGN: intestazione e corpo.

Come è ben visibile, i dati forniti sono scomponibili in due blocchi principali: intestazione e corpo. Il primo è costituito da un insieme di tag racchiusi tra parentesi quadre, il secondo da una sequenza di caratteri ASCII che rappresenta l' evoluzione delle mosse dei pezzi sulla scacchiera.

3.1.1 Insieme dei tag PGN

Lo scopo di questi tag è di fornire tutte le informazioni ambientali che caratterizzano la partita descritta nel file PGN come, ad esempio, il luogo dove si è svolta, i nomi dei giocatori e così via.
Analizziamoli nello specifico:

- `[Event "Open"]` : specifica il nome dell'evento durante il quale la partita è stata giocata, come ad esempio un torneo, un campionato e così via. Nel caso questo dato non sia conosciuto viene normalmente utilizzato il simbolo "?".

- `[Site "?"]` : indica il luogo dove la partita è stata disputata. Anche in questo caso il simbolo "?" indica che il luogo è sconosciuto. Un altro esempio corretto potrebbe essere `[Site "New York City, NY USA"]`.

- `[Date "1996.??.??"]`: indica la data in cui la partita è stata giocata nel formato "AAAA.MM.GG". Come nel caso precedente, i simboli "?" indicano che mese e giorno non sono noti.

- `[Round "0"]` : specifica il round di gioco (o numero delle partite sinora disputate). E' possibile ci siano anche round multipli, che si possono indicare come `[Round "3.1"]`, dove 3 indica il round principale e 1 il "sotto-round".

- `[White "G.Minchev"]`: specifica il nome del giocatore bianco.

- `[Black "Nejad"]` : specifica il nome del giocatore nero.

- `[Result "1-0"]` : indica il risultato della partita; in questo caso bianco vince su nero, ma altri valori attribuibili sono "0-1", "1/2-1/2" (caso di parità), "*" (caso di

partita ancora in corso, abbandonata o di risultato sconosciuto).

I sette tag qui presentati sono quelli fondamentali che non devono mai mancare in ogni file PGN, ma ne esistono anche altri supplementari. Per eventuali approfondimenti sull'argomento si consiglia di consultare [PGN1].

3.1.2 Rappresentazione delle mosse

Le mosse in un file PGN vengono rappresentate secondo la formalizzazione SAN (Standard Algebric Notation). In ogni mossa devono essere presenti tutti i dati necessari ad individuare il pezzo che deve muovere, la relativa locazione di destinazione e le informazioni aggiuntive eventualmente necessarie alla disambiguazione (il pezzo candidato alla mossa deve essere univoco). La simbologia utilizzata per indicare i vari pezzi sulla scacchiera deriva dalla terminologia inglese:

P (pawn) = pedone (in genere è omesso);
N (knight) = cavallo;
B (bishop) = alfiere;
R (rook) = torre;
Q (queen) = regina;
K (king) = re;

Gli ultimi due caratteri della mossa indicano la locazione di destinazione del pezzo sulla scacchiera, quindi:

Nf3 = cavallo muove in f3.

Come è ben visibile, non esiste alcun simbolo che ci faccia capire se a muovere sia il cavallo bianco oppure quello nero. Questa informazione ci viene fornita intrinsecamente dal posto che la mossa occupa all'interno di ogni singolo turno di gioco: il primo a muovere è sempre il bianco, il secondo il nero. Consideriamo un esempio tratto dalla figura 3.1:

2. Nf3 Nc6 = nel secondo turno di gioco (2.) il cavallo bianco muove in f3, mentre quello nero in c6 (le mosse sono sempre separate da uno spazio bianco).

Va ricordato che il simbolo del pedone viene spesso omesso ("e3" = "Pe3" = pedone muove in e3).
Esiste la possibilità che più pezzi dello stesso tipo possano muovere nella medesima casella di destinazione e a tal proposito sussistono precise regole di disambiguazione:

1. Se il pezzo è individuabile attraverso l'indicazione della colonna dove risiede, tale indicazione viene inserita subito dopo la lettera che indica il pezzo stesso (es.: "Bce3" = l'alfiere nella colonna "c" muove in e3).
2. Se la regola 1 fallisce, allora si indica la riga di appartenenza del pezzo (es.: "B1e3" = l'alfiere nella riga "1" muove in e3).

3. Se entrambe le regole 1 e 2 non hanno buon esito, allora si indicano sia la riga che la colonna di appartenenza del pezzo (es.: "Bc1e3" = l'alfiere nella colonna "c" riga "1" muove in e3).

La formalizzazione SAN fornisce anche un insieme di caratteri speciali (o sequenze di caratteri), utili ad individuare altrettanti casi specifici o terminologie del gioco degli scacchi. Vediamoli di seguito elencati tra apici:

- **"O-O"** : arrocco corto (lato del re).

- **"O-O-O"** : arrocco lungo (lato della regina).

- **"x"** : mangiata o presa (es.: Nxd4 = cavallo prende in d4; exd5 = pedone nella colonna "e" prende in d5).

- **"="** : promozione di pedone (es.: a8 = Q indica che il pedone muove in a8 e viene promosso a regina).

- **"+"** : posto al termine della mossa indica che è una mossa di scacco.

- **"#"** : posto al termine della mossa indica che è una mossa di scacco matto.

Per concludere, alla luce di quanto detto sinora, possiamo osservare che una mossa scacchistica descritta nel formato SAN è una stringa di lunghezza variabile tra i due caratteri (es.: "e4") ed i sette caratteri (es.: "Ra6xb6#" = la torre in "a6" mangia in "b6" ed è scacco matto; "fxg1=Q+" = il pedone nella colonna "f" prende in "g1", viene promosso a regina ed è scacco).

3.2 Generazione dinamica di documenti

La generazione dinamica di documenti può essere vista, fondamentalmente, come un processo di trasformazione. L'oggetto di tale trasformazione, in genere, è una sorgente di informazioni scritta in un formato che non ne consente una visualizzazione di per sé gradevole, spesso perché i contenuti della sorgente stessa sono strutturati in modo scarno, minimale, più consono alla programmazione che non alla visualizzazione grafica. Lo scopo della trasformazione, quindi, è proprio quello di tradurre la sorgente in un formato adatto alla visualizzazione, generando uno o più documenti a seconda delle necessità.
Nel nostro caso, come vedremo successivamente, la sorgente informativa di partenza sarà un file PGN, che verrà visivamente interpretato e tradotto nei formati LIT, HTML e PDF.
Il processo di trasformazione, e quindi relativa generazione dei documenti, può avvenire in due modi differenti:

- Generazione del documento direttamente nel suo formato finale.

- Passaggio attraverso una sequenza di trasformazioni che conducono dalla fonte iniziale al formato finale toccando una serie di stadi intermedi. I passaggi possono essere integrati all'interno di uno stesso software o possono comportare l'utilizzo di più programmi differenti.

Per entrambi i metodi si possono ottenere, a loro volta, diverse forme di dinamicità, legate a precise scelte progettuali o, più spesso, vincolate dai limiti tecnologici derivanti dai vari software di gestione dei documenti stessi:

- Tutte le pagine del documento vengono generate in un unico passaggio. In questo modo l'utente ha a disposizione l'intero blocco informativo da subito, sia che il suo interesse copra tutto il documento o solo porzioni limitate.

  Vantaggi: le interazioni con l'eventuale software di trasformazione sono ridotte al minimo. Il documento può essere direttamente distribuito, poiché non sono necessarie ulteriori trasformazioni.

  Svantaggi: vengono prodotte anche quelle porzioni di documento che possono essere non interessanti per l'utente; il documento finale risulta essere più pesante sia in termini di gestione che di allocazione di memoria.

- Le pagine del documento vengono generate solo ed esclusivamente su specifica richiesta dell'utente. Questo approccio è indubbiamente più potente ed efficace del precedente, in quanto vengono visualizzate soltanto le informazioni strettamente necessarie a coprire l'interesse dell'utente stesso.

  Vantaggi: visualizzazione mirata delle informazioni; le singole pagine prodotte risultano essere più veloci da gestire e meno pesanti da allocare in memoria.

  Svantaggi: un elevato numero di interazioni con il software di trasformazione potrebbe portare a cali prestazionali (es.: l'utente potrebbe dover aspettare troppo tra la generazione di una pagina e la successiva).

3.3 Interattività

Rendere un documento interattivo significa dotarlo di meccanismi che consentano all'utente di agire sui contenuti, il chè può significare diverse cose:

1. L'utente può cambiare i contenuti del documento (editing) a suo piacimento.

2. Sussistono meccanismi interni al documento per modificarne i contenuti in modo pianificato, espandendo o riducendo la quantità di informazioni visualizzate (ad esempio menù a scomparsa, finestre di popup, ecc.).

3. L'utente può navigare attraverso il contenuto informativo scegliendo ogni volta un percorso diverso, visualizzando solo ciò che gli interessa ed ignorando il resto (esempio tipico sono i collegamenti ipertestuali).

4. Integrazione di soluzioni che permettano all'utente di compiere delle interrogazioni, visualizzando solo informazioni mirate a dare risposta all'interrogazione fatta (form, database quering, ecc.).

Rapportando quanto detto alle soluzioni software illustrate nei capitoli successivi, i meccanismi di interazione scelti per i documenti generati sono principalmente di tipo 2 e 3. Nel caso di tipo 3 la soluzione consiste nell'integrare in ogni pagina del documento un menù, le cui voci sono collegamenti a tutte le pagine presenti nel documento stesso, siamo esse statiche o dinamiche. Nel caso di tipo 2, le funzionalità di interazione sono in grado di modificare dinamicamente il contenuto del documento in base alle richieste dell'utente, senza la mediazione di meccanismi di trasformazione esterni.

3.4 Soluzioni esistenti

Vediamo ora alcuni esempi di soluzioni già esistenti, in merito alle problematiche precedentemente discusse, disponibili al pubblico come realizzazioni per il web o come applicativi stand-alone. C'è da premettere che esistono una miriade di realizzazioni diverse volte alla costruzione e/o manipolazione di partite di scacchi sfruttando la formalizzazione PGN, ma la maggior parte di esse non cura l'aspetto del salvataggio in formati diversi da PGN stesso.

3.4.1 SVG Chess

Questo progetto è stato sviluppato da Peter Watkins e si pone come scopo principale la validazione di mosse scacchistiche attraverso XLST,

utilizzando un'interfaccia grafica (scacchiera interattiva) realizzata in SVG.
Facciamo una piccola digressione sulle tecnologie che entrano in gioco in questo applicativo, onde capirne meglio le caratteristiche:

- **SVG**: significa Scalable Vector Graphics ed è un nuovo formato di file grafico realizzato per applicativi web basati su XML. Consente di generare elementi grafici ad alta definizione dinamicamente, partendo da dati real-time, con precisi comandi strutturali e visuali.
  I punti di forza di questo formato sono la compattezza in memoria e l'alta risoluzione delle immagini, garantita ad ogni livello di ingrandimento.

- **XSLT**: è un linguaggio di programmazione sviluppato dal W3C che si sta affermando come uno standard sul World Wide Web. La sua nascita è legata ad XML e XSL, in particolare XSLT sta per "XSL Transformation".
  Ogni documento XML appare strutturato come un albero e necessita di un particolare meccanismo di formattazione dei dati che ne consenta una corretta visualizzazione come pagina web. Tale meccanismo è fornito da XSL (eXtensible Stylesheet Language), il cui componente fondamentale è proprio XSLT, un linguaggio autonomo utilizzato per convertire un documento XML in altri formati come HTML, PDF, RTF od ancora XML. Per ulteriori approfondimenti si consiglia di consultare [XSL].

SVG Chess consente all'utente di muovere i pezzi sulla scacchiera in accordo con le regole degli scacchi, fornendo opzioni visuali per suggerire le eventuali mosse consentite o meno. L'interfaccia grafica di SVG Chess è rappresentata in figura 2.1, dove è possibile identificare la scacchiera interattiva sulla quale l'utente può effettuare le mosse, una legenda che spiega come vengono illuminate le caselle in base alle possibili mosse da effettuare ed un quadro comandi per settare le possibili opzioni di visualizzazione.

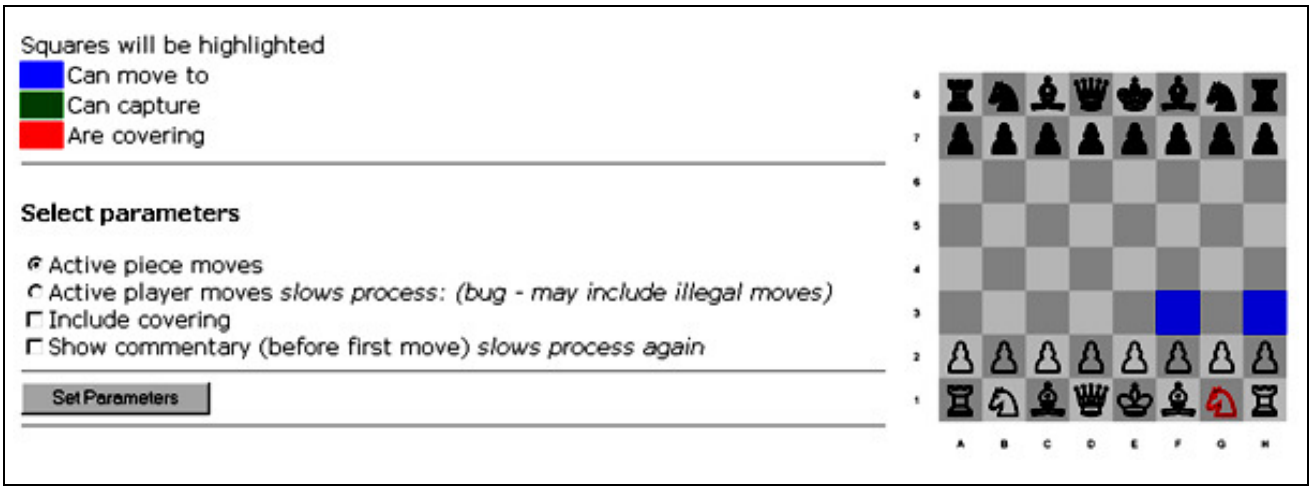


Figura 3.2: interfaccia grafica di SVG Chess.

Il codice XML e le relative trasformazioni che entrano in gioco si basano fondamentalmente sulla gestione di tre elementi:

- **Moves**: rappresentazione delle mosse.
- **Squares**: rappresentazione dello stato corrente della scacchiera.
- **Validation**: serve per convalidare o meno le mosse effettuate. Ha un attributo *go* che identifica il giocatore corrente ed un attributo *error* che permette di individuare errori nelle mosse effettuate (mosse illegali). Nell'albero strutturale possiede dei nodi figli contenenti le istruzioni necessarie all'aggiornamento della scacchiera SVG.

Quando un utente clicca e seleziona un pezzo da muovere subentrano determinate trasformazioni:

- Moves e Squares rimangono inalterati.
- Validation prende due liste di nodi, una contenente le istruzioni per aggiornare la scacchiera (caselle colorate per indicare le mosse possibili) e l'altra contenente la rappresentazione di tutte le mosse possibili relative al pezzo selezionato.

A questo punto l'utente può o scegliere di cambiare il pezzo da muovere, ripetendo il passo precedente, o selezionare una casella di destinazione sulla scacchiera, innescando un nuovo processo di trasformazione:

- Un nodo *move* (mossa) viene collegato all'elemento Moves.

- L'elemento Squares viene aggiornato in modo da rappresentare il nuovo stato della scacchiera dopo la mossa.

- Validation controlla che la mossa effettuata sia legale: in caso contrario il suo attributo *error* permette di segnalare l'impossibilità di effettuare la mossa, bloccandone la prosecuzione. Se non sussistono illegalità, invece, l'attributo *go* indicherà il nuovo giocatore attivo (White o Black) e la relativa lista di comandi verrà opportunamente compilata con le istruzioni necessarie all'aggiornamento grafico della scacchiera.

- Per ogni mossa effettuata viene indicata, in un apposito spazio, la scrittura PGN relativa alla mossa stessa.

SVG Chess è stato chiaramente sviluppato per il web ed allo stato attuale non consente l'esportazione delle rappresentazioni delle partite in alcun formato. In particolare non v'è possibilità di acquisire partite già scritte da terzi in PGN (o formalizzazioni affini) o di correggere eventuali errori subentrati nella movimentazione dei pezzi.
Questa soluzione, che ha scopi chiaramente di puro editing, presenta un elevato grado di dinamicità, poiché il documento è in grado di modificare i propri contenuti in maniera autonoma.
Essendo il progetto ancora in via di sviluppo, però, ed essendo state utilizzate tecnologie di realizzazione potenti e versatili, non si esclude che in futuro possano essere sviluppate ed integrate nuove funzionalità, volte proprio alla creazione di documentazione digitale in formati di ampia diffusione, come ad esempio PDF.

3.4.2 RenderX

RenderX è un'azienda leader nel settore delle soluzioni per la produzione di documentazione digitale, specialmente per quanto riguarda le trasformazioni "XML to PDF" e "XML to PostScript". In particolare ha sviluppato un motore, XEP, dedicato alla conversione di documenti da XSL a PDF realizzato in Java. Per dimostrare la potenza di XSL unitamente a XEP, gli sviluppatori di RenderX hanno realizzato un tool completo per la visualizzazione di partite di scacchi, con la conseguente capacità di esportazione dei dati in formato PDF.

Il processo parte dalla rappresentazione dei dati di una partita in formato XML, utilizzando una struttura ed una notazione molto simili alla formalizzazione PGN. Vediamone un esempio considerando la figura 3.3.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
  - <chessgame event="IBM Kasparov vs. Deep Blue Rematch" site="New York,
   NY USA" date="1997.05.03" round="1" white="Kasparov, Garry"
   black="Deep Blue" opening="Reti-King's Indian attack,Keres variation"
   result="1-0">
     <move>
         <white>N g1-f3</white>
         <black>P d7-d5</black>
     </move>
     <move>
         <white>P g2-g3</white>
         <black>B c8-g4</black>
     </move>
     ...
     ...
 </chessgame>
```

Figura 3.3: formalizzazione XML di una partita di scacchi creata per XEP.

A questo punto entra in gioco XSL, che converte il semplice testo della rappresentazione della partita in una sequenza di immagini grafiche che rappresentano l'evoluzione delle mosse (viene utilizzata una particolare fonte scacchistica per la rappresentazione dei pezzi). Il risultato di questa operazione è uno stream FO (Formatted Object), cioè un documento dai contenuti formattati e con una precisa struttura di visualizzazione.
Il passo finale, ora, consiste nel convertire il risultato precedentemente ottenuto nel formato PDF. Per farlo si utilizza XEP che, essendo in grado di implementare la compressione PDF, produce un documento finale di dimensioni estremamente contenute.
Tutto il procedimento può essere schematizzato come in figura 3.4, nella quale sono riportate anche possibili variazioni sul tema riguardo i formati di output (HTML, WML e TXT).
Il documento finale PDF si presenta come in figura 3.5, con un'intestazione riportante i dati principali della partita (un parallelo coi tag PGN) ed una sequenza di diagrammi scacchistici rappresentanti l'evoluzione delle mosse nella partita stessa.
Questa realizzazione risulta completa per quanto riguarda la generazione dinamica di documenti digitali statici, ma non presenta alcuna capacità di editing. Oltremodo, allo stato attuale, non v'è alcuna possibilità di inserire nei documenti prodotti un meccanismo di interazione con l'utente.

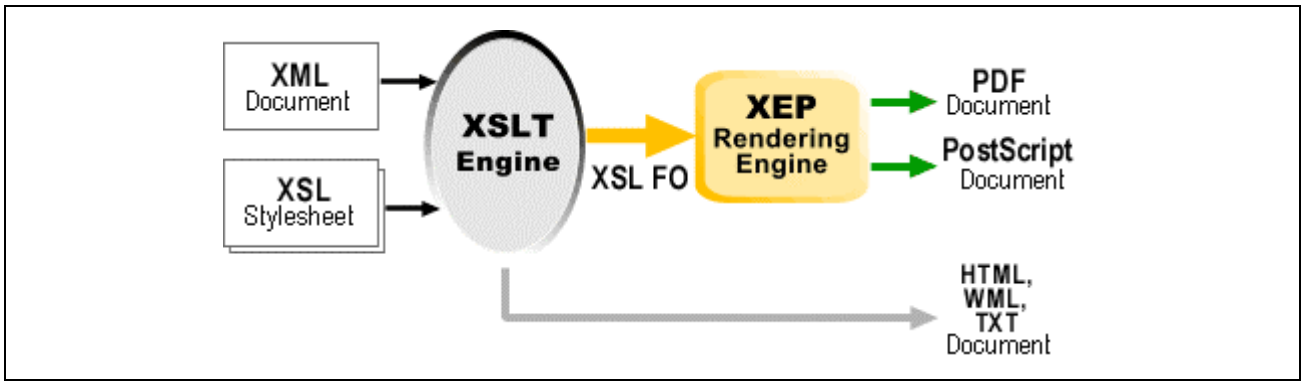
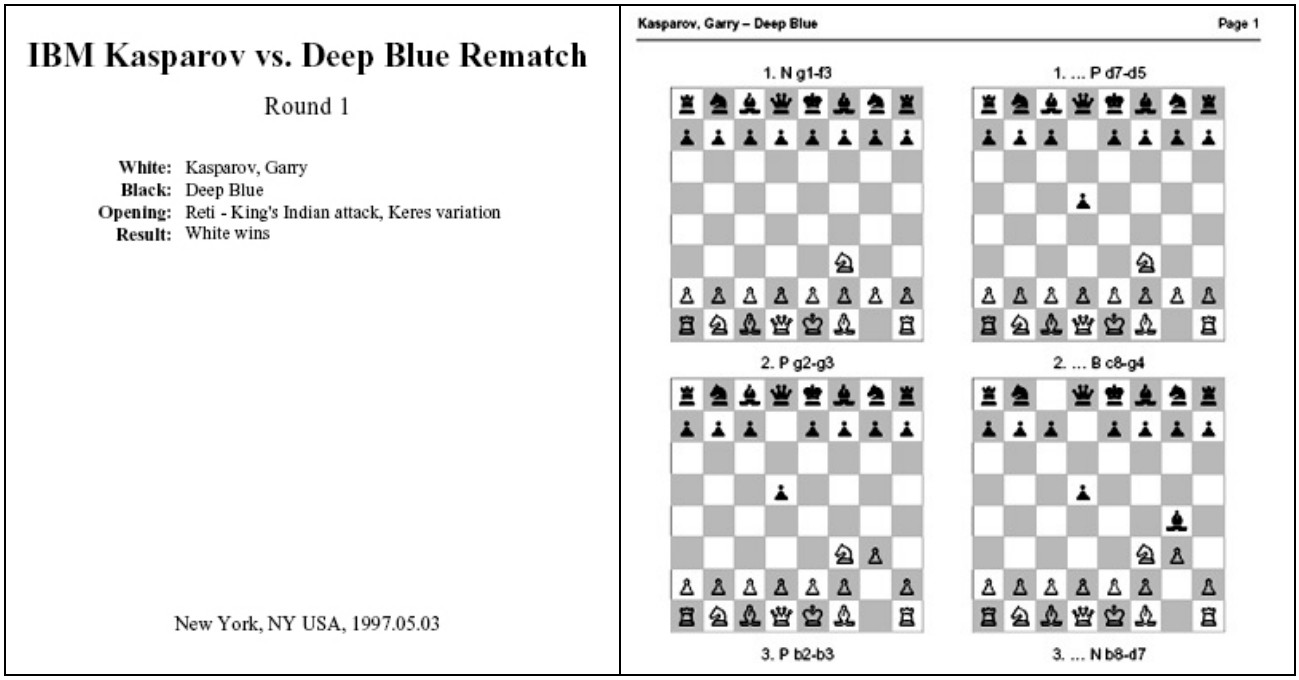


Figura 3.4: processo di trasformazione di un documento XML in PDF attraverso l'utilizzo di XEP.

La formalizzazione XML della partita, infatti, viene presentata come un qualcosa di già costruito, di dato in partenza, senza specifiche riguardo come sia stata implementata.
Non è difficile pensare, comunque, che si possa ovviare al problema realizzando un modulo aggiuntivo al programma, in grado di convertire un file PGN in una equivalente specifica XML o, ancor più completo, che possa offrire un tool visuale per la costruzione di una partita (come la scacchiera virtuale di SVG Chess) e quindi della relativa formalizzazione scritta.


IBM Kasparov vs. Deep Blue Rematch

Round 1

White: Kasparov, Garry
Black: Deep Blue
Opening: Reti - King's Indian attack, Keres variation
Result: White wins

New York, NY USA, 1997.05.03

Kasparov, Garry – Deep Blue Page 1

1. N g1-f3 1. ... P d7-d5

2. P g2-g3 2. ... B c8-g4

3. P b2-b3 3. ... N b8-d7

Figura 3.5: documento PDF risultante in uscita a XEP. A sinistra l'intestazione con i dati della partita, a destra l'evoluzione dei diagrammi scacchistici e le relative mosse riportate sopra ogni diagramma.

3.4.3 PGN to JS

Questo tool consente di convertire file scritti in notazione PGN in pagine HTML, con diagrammi e comandi per osservare l'evoluzione delle mosse realizzati in JavaScript. Vediamo una panoramica delle numerose caratteristiche del software:

- Creazione di partite ex-novo attraverso l'utilizzo della scacchiera virtuale. Le mosse compiute vengono direttamente scritte in notazione PGN ed è possibile apportare modifiche alla sequenza in ogni momento.

- Possibilità di gestire mosse alternative, caratteristica che consente anche l'implementazione di quesiti scacchistici, non solo partite complete. Ciò è facilitato dalla capacità del software di gestire il FEN, un particolare tag PGN utilizzato per riportare le posizioni di apertura del gioco sulla scacchiera.
- Inserimento di commenti alle mosse effettuate ("unica mossa possibile", "compensazione" e così via).

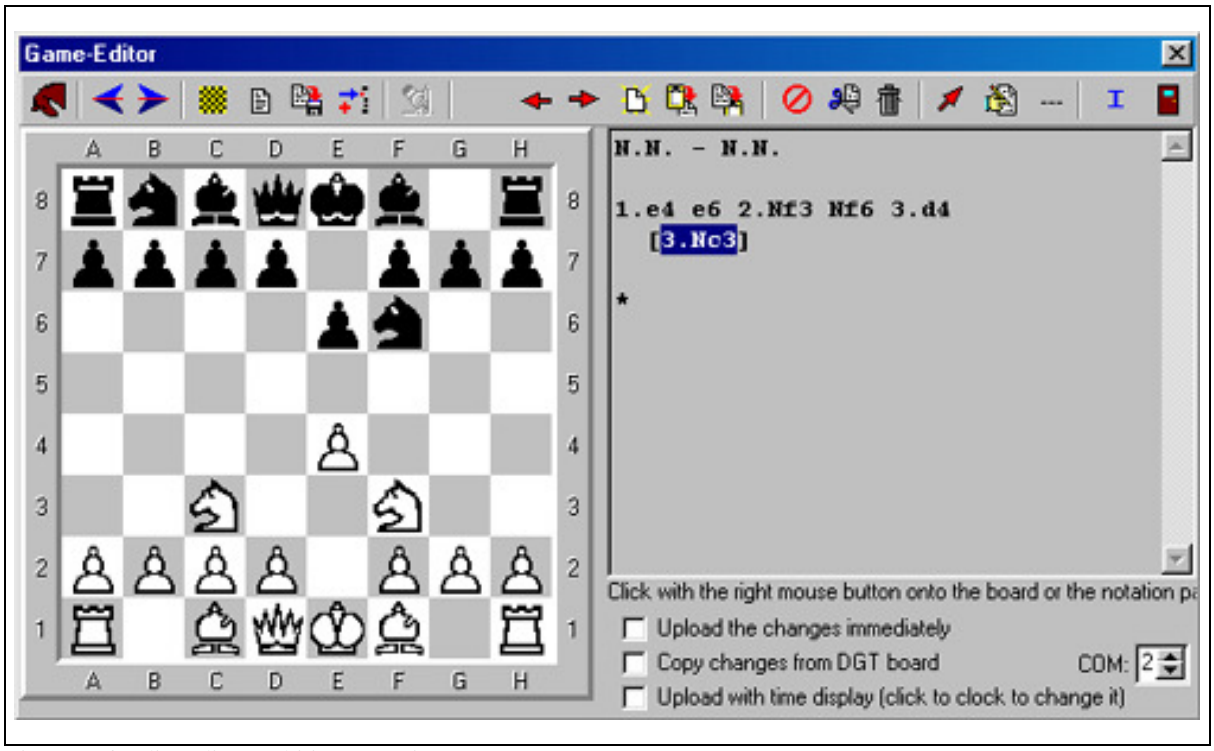


Figura 3.6: editor di scacchi integrato in PGNtoJS.

- Capacità di salvare lo stato della scacchiera, in ogni momento, come immagine (JPEG o Bitmap) o come sequenza di fonti scacchistiche. Quest'ultima opzione consente, attraverso la tecnica di copia/incolla, di riportare il diagramma in un qualsiasi word processor o programma affine.

- Acquisizione di file PGN già esistenti, con possibilità di manipolazione del relativo contenuto attraverso le utilities viste sopra. Un file PGN può contenere più partite, costituendo di fatto un vero e proprio database di giochi: PGNtoJS permette di visualizzare questo database e caricare nell'editor una singola partita selezionata tra le tante, così come permette la cancellazione di una partita dal database in caso di necessità.
  Il file-database scelto nella schermata di apertura del software serve anche come riferimento per i salvataggi delle partite create con l'editor. In questo modo è possibile salvare i passaggi intermedi e completare comodamente il lavoro in più tempi.

- Editor HTML integrato, le cui funzioni basilari sono simili a quella dell'editor standard, ma volte a scopi differenti. Consente la generazione diretta di codice PGN muovendo i pezzi sulla scacchiera virtuale e permettendo le modifiche alla sequenza delle mosse. Il testo PGN può successivamente essere manualmente corredato di commenti od integrato in un testo di diversa natura. In alternativa, nel testo del documento possono essere inserite le copie della configurazione della scacchiera virtuale sotto forma di sequenza di fonti scacchistiche. E' possibile così creare documenti completi, che possono essere visti in anteprima di stampa, stampati direttamente o esportati in formato RTF.

- Generazione dinamica di documenti HTML contenenti comandi JavaScript per la visualizzazione della partita esportata. E' possibile, attraverso il settaggio di alcune opzioni, inserire pulsanti per lo spostamento avanti ed indietro nella sequenza delle mosse, od un comando per l'autoplay della partita in oggetto.

- Menù avanzato per la gestione delle preferenze di visualizzazione inerenti all'aspetto dei documenti HTML generati.

Figura 3.7: schermata di esempio di un file HTML generato con PGNtoJS.

Senza dubbio, questo software è completo sia dal punto di vista dell'editing che dal punto di vista del delivering, cioè della consegna, della diffusione della documentazione digitale generata dal programma. Con PGNtoJS, infatti, è possibile creare documenti adatti sia al Worl Wide Web che a pubblicazioni su cartaceo.
Anche per quanto riguarda le funzionalità di interazione con l'utente fornite all'interno dei documenti prodotti, risulta essere più che soddisfacente, anche se limitato all'ambiente Web (il formato RTF non presenta alcun aspetto interattivo).

3.4.4 DocBook

DocBook [DOB] è una realizzazione molto simile, sotto certi aspetti, al precedentemente menzionato RenderX. Si tratta di un vero e proprio linguaggio di markup per scrivere documenti strutturati utilizzando SGML o XML. In poche parole DocBook è un DTD già utilizzato da migliaia di utenti in tutto il mondo per scrivere documenti in vari formati, sia per la stampa che per ambiente Web (anche SUN lo utilizza per la documentazione tecnica).
Vediamo di chiarire alcuni concetti tecnici legati agli acronimi sopra citati:

- SGML è un framework per descrivere singoli linguaggi di markup: HTML e DocBook stessi sono istanziazioni di questo linguaggio.

- DTD (Document Type Definition): è la specifica SGML/XML di un particolare linguaggio di markup utilizzato all'interno di un

documento. Definisce tutti i tag che vengono utilizzati dal linguaggio e la loro eventuale gerarchia.

Strumenti come DocBook hanno cambiato il modo di vedere la produzione di documenti. Con editor di tipo WYSIWYG o sistemi di type setting come TeX, la maggior preoccupazione nell'utilizzo del linguaggio di markup è l'ottimizzazione dell'aspetto del documento in funzione della stampa. In ambiente DocBook, invece, la preoccupazione ricade su come utilizzare il linguaggio di markup per enfatizzare il contenuto semantico, in modo da poter riconoscere ed estrarre le informazioni utili.
Il linguaggio, infatti, fornisce un ricco insieme di tag strettamente relazionato alla struttura ed alla natura dei contenuti. Ad esempio vediamo come sono strutturate le due principali tipologie di documentazione ottenibili con DocBook, consultando la figura 3.8.

Books	**Articles**
book meta information chapter sect1 sect1 chapter sect1 appendix sect1 appendix sect1 ... glossary	article meta information sect1 sect1 sect2 sect1 ...

Figura 3.8: struttura dei documenti in DocBook. A sinistra il tipo "Book", a destra il tipo "Article".

Esiste sempre un elemento principale che identifica la natura del documento: `book` per i libri e `article` per gli articoli. A seconda della tipologia di documento, poi, la struttura interna viene organizzata in semplici sezioni (caso di articolo) o con costrutti più complessi come capitoli suddivisi in sezioni e così via (caso di libro).
A questo punto rimane solo da vedere come appare il listato di un documento vero e proprio scritto secondo il formalismo DocBook e lo troviamo in figura 3.9.
Detto questo vediamo che sussiste una base per creare anche documenti a carattere scacchistico. In particolare, se consideriamo il DTD in XML e le

eventuali trasformazioni applicabili attraverso XLS, è possibile ricondursi ad un processo di pubblicazione molto simile a quello analizzato per RenderX. Guardando figura 3.10 ce ne rendiamo conto meglio.

```
<!doctype book PUBLIC "-//Davenport//DTD DocBook V3.0//EN" []>
<book>
  <bookinfo>
    <date>1997-10-11</date>
    <title>My first booklet</title>
    <subtitle>it even has a subtitle</subtitle>
  </bookinfo>
  <toc></toc>
  <!-- We are done with the preliminaries, now we can start with
          the body of the document -->
  <chapter>
    <title>My first chapter</title>
    <para>Here's a paragraph of text because it is …</para>
    <sect1>
      <title>A section in that first chapter</title>
      <para>All I need is a single paragraph of text …</para>
    </sect1>
  </chapter>
  <appendix>
    <title>Remaining details</title>
    <para>Although this booklet is quite complete …</para>
    <sect1>
      <title>Use of the word dude</title>
      <para>Here's an example of how to say
        <emphasis>dude</emphasis>: DUDE.</para>
    </sect1>
  </appendix>
</book>
```

Figura 3.9: documento di tipo “book” scritto nel formalismo DocBook.

La specifica del documento redatta in XML viene sottoposta ad un processo di trasformazione tramite XSLT. La creazione di opportuni fogli di stile consente di ottenere diversi tipi di output, tra i quali HTML (successivamente riconducibile a testo semplice) ed oggetti formattati (FO). Questi ultimi possono essere utilizzati per generare documenti in formato PDF o utilizzando XEP di RenderX o passando per il formato Tex (il percorso di pubblicazione XML → XSLT → FO → XEP → PDF è esattamente quello incontrato per RenderX).
Anche in questo caso, comunque, il risultato finale è un documento statico senza alcuna possibilità di interazione (a meno di costruire fogli di stile XSL opportuni) o di editing. Per sua natura, infatti, DocBook risulta essere un ottimo sistema di delivering, ma nulla di più.

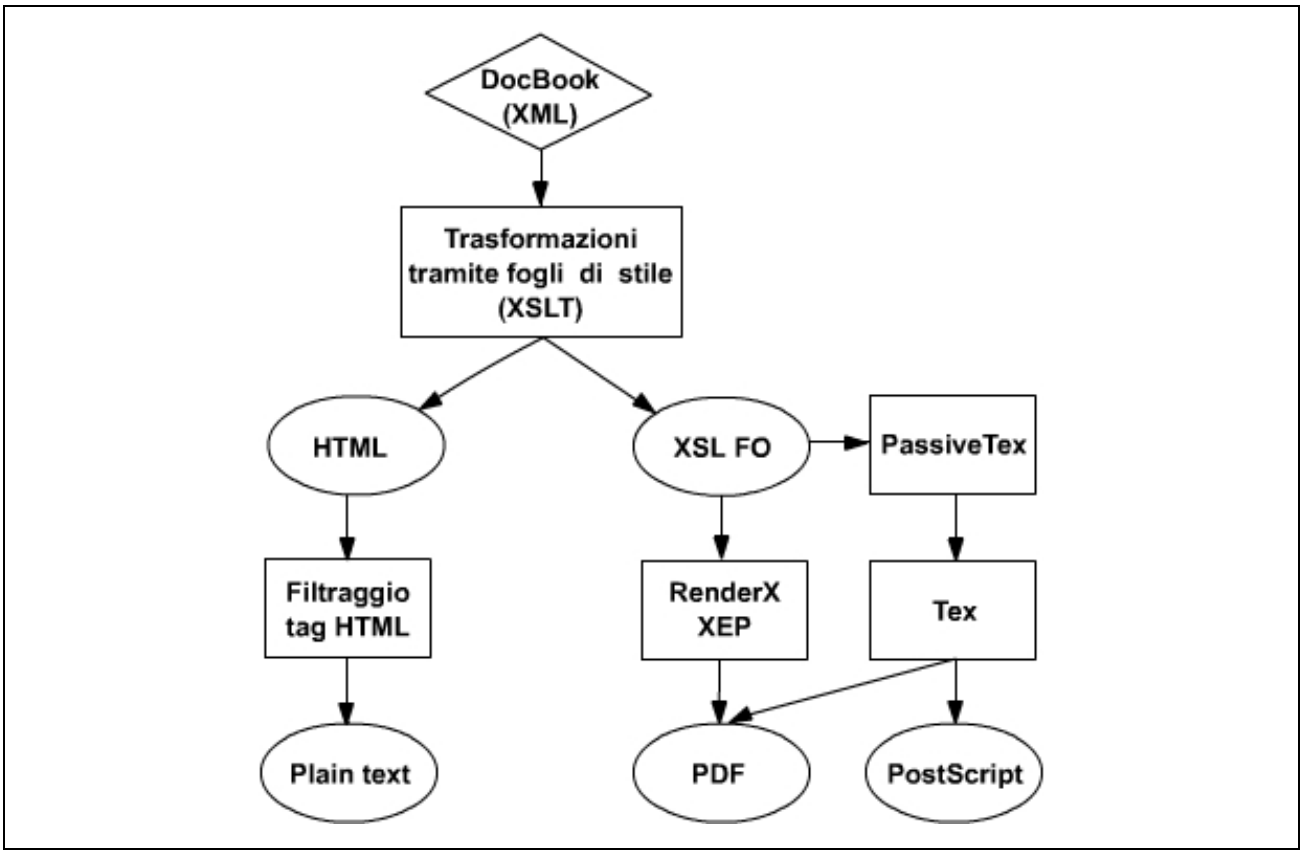


Figura 3.10: processo di pubblicazione con DocBook.

3.5 Conclusioni

Analizzando tutto quando detto sinora è possibile fare un primo punto della situazione. Da un lato abbiamo il problema della generazione dinamica di documenti interattivi, dall'altro una vasta gamma di formati digitali utilizzabili per risolvere il detto problema.
Come è già stato anticipato, tra tutti i formati di documenti digitali disponibili sul mercato la scelta è caduta principalmente su tre: LIT di Microsoft Reader, PDF e HTML. Confrontiamoli tra loro, illustrando pregi e difetti delle loro caratteristiche in funzione della risoluzione del nostro problema.

LIT risulta essere un formato chiuso, cioè una volta che il documento è stato creato non ci sono possibilità di apportare modifiche al contenuto, se non ripetendo tutto il processo di creazione. Questo aspetto influisce inesorabilmente sul grado di dinamicità che un applicativo sviluppato per questo ambiente può offrire.
Le caratteristiche di compattezza in memoria e nitidezza di visualizzazione rendono questo formato un ottimo strumento di delivering (compito per il quale nasce).

Le funzionalità di gestione dei collegamenti sia interni che a fonti esterne costituiscono buoni strumenti di navigazione dei contenuti, garantendo il livello di interattività richiesto per la risoluzione del nostro problema.
PDF può essere generato direttamente a livello di programmazione, senza che subentrino stadi intermedi di trasformazione (al contrario di LIT). I meccanismi offerti per la navigazione dei contenuti (link interni al documento, bookmark e JavaScript integrabile) coprono perfettamente le esigenze progettuali precedentemente evidenziate.
L'ottima gestione del layout, e quindi degli elementi visivi, rendono PDF un altrettanto ottimo strumento di delivering, ma è da escludersi qualsiasi funzionalità di editing degna di nota (per apportare modifiche è necessario ricreare il documento), a meno di utilizzare strumenti appositi come Acrobat.
Per quanto riguarda la dinamicità nella creazione dei documenti non esistono particolari vincoli tecnologici.

HTML è indubbiamente il più versatile tra tutti i formati visti, in quanto è semplice da generare (direttamente a livello di programmazione) e modificare (esistono numerosi editor in grado di farlo). Un listato HTML è renderizzabile attraverso svariati software, che possono quindi essere utilizzati dai sistemi di generazione documenti come piattaforme di visualizzazione, di conversione e come strumenti di appoggio per la realizzazione del grado di interattività e dinamicità desiderati.
L'unico problema di questo formato è riconducibile alla visualizzazione, in quanto non sussistono costrutti "forti" per la gestione del layout di ogni singola pagina. In un contesto di delivering quindi, dove il documento prodotto deve essere accettabile per la stampa su cartaceo, HTML non è utilizzabile, ma si rivela un ottimo strumento di passaggio, un formato intermedio dal quale si possono ottenere altri formati.

Gli applicativi visionati (SVG Chess, RenderX, PGNtoJS e DocBook) sono stati scelti per fornire una panoramica esaustiva, un sunto di ciò che il mercato offre per la produzione di documenti digitali, in particolare documenti a carattere scacchistico. I parametri di valutazione sono legati a quattro aspetti fondamentali, gli stessi che caratterizzano il problema più generale della creazione dinamica dei documenti: dinamicità, interattività, delivering ed editing. Queste quattro caratteristiche sono difficili da trovare in un unico software, spesso perché il fine ultimo del software stesso non le richiede tutte insieme. Prendiamo, ad esempio, RenderX e

DocBook: lo scopo della loro creazione è di fornire un meccanismo potente per generare voluminosi quantitativi di documenti digitali a carattere generale, dalla documentazione tecnica ai libri, passando anche per le pubblicazioni scacchistiche. Il loro fine ultimo, quindi, è il delivering di documenti statici, senza alcuna necessità di interattività.
SVG Chess, invece, trova impiego solo in ambiente Web e presenta alti livelli di interattività (scacchiera virtuale) e dinamicità (il documento autogestisce le proprie modifiche internamente), ma ha scopo puramente di editing.
PGNtoJS è forse l'esempio più completo, in quanto racchiude sia capacità di editing che di pubblicazione di documenti dinamici interattivi, con la limitazione, però, a documenti adatti solo al Web (le pubblicazioni in RTF sono statiche e prive di interattività).
Il traguardo finale al quale aspirare, a mio avviso, è la creazione di documenti automodificanti, cioè in grado di modellarsi autonomamente sulle esigenze dell'utente: in parole povere il documento ideale è editor di sè stesso. Cose di questo tipo esistono già e sono realizzate con metodi di programmazione embedded, cioè esistono costrutti in grado di integrare all'interno del documento programmi in vari linguaggi che ne gestiscono i contenuti. Basti pensare alle pagine Web create dinamicamente in risposta o a query dell'utente, o a valori salvati in un qualche cookie, oppure a particolari documenti interattivi PDF dotati di funzionalità JavaScript integrate.

4 ANALISI SINTATTICA DI TESTI SCACCHISTICI

Come è stato detto nel capitolo precedente, uno dei principali problemi da risolvere per raggiungere l'obiettivo finale è l'interpretazione delle partite di scacchi. Questo passo può essere riformulato, alla luce di quanto visto sinora, come l'integrazione via software della notazione PGN attraverso particolari strutture dati. In definitiva è necessaria un'interpretazione sintattica della notazione PGN e questo implica la presenza di un parser apposito.
E' proprio a questo scopo che è stata realizzata la libreria Perl *PGN_parser.pm*, che costituisce un vero e proprio ponte tra il file PGN e la rappresentazione grafica dei giochi scacchistici e per questo viene utilizzata dalla maggior parte dei software di creazione documenti implementati.
L'interfaccia della libreria è costituita fondamentalmente da due funzioni:

- ***PGN_Parse*** *("file.pgn")*: si basa sulla libreria Perl *Chess::PGN::Parse* [PGN2]; come argomento prende il nome del file PGN dal quale si vuole estrapolare la partita di scacchi, mentre in output fornisce un hash contenente tutti i dati relativi alla partita stessa (i sette tag descrittivi ed un array contenente tutte le mosse prive dell'indicativo di turno).

- ***Parse_Game*** *(elenco mosse, turno, giocatore)* : questa funzione è in grado di sviluppare la partita, e quindi di gestire la scacchiera, fino al punto specificato dai parametri di input. In particolare, *elenco mosse* è l'array contenente la lista delle mosse fornito da *PGN_Parse()*, *turno* specifica fino a quale turno la partita deve essere sviluppata e *giocatore* specifica fino a che mossa (giocatore bianco o nero) deve essere sviluppato l'ultimo turno di gioco indicato.
  Ad esempio *Parse_Game(mosse, 12, W)* si interpreta come: "Considerando la sequenza di mosse contenuta nell'array *mosse*, sviluppa la partita fino al dodicesimo turno ed in quest'ultimo limitati alla mossa del giocatore bianco (W = bianco, B = nero)" .
  L'output consiste di due variabili ambientali *%ENV_scacchi* ed *%ENV_scacchiera*, due hash che riportano lo stato della scacchiera e dei pezzi su di essa al termine dell'esecuzione di *Parse_Game()*.

4.1 Strutture dati

Affrontiamo ora un'analisi delle principali strutture dati che vengono utilizzate dal nostro parser. Si tratta di due tabelle hash, ideate appositamente per snellire gli algoritmi di gestione della configurazione della scacchiera e di reperimento dei pezzi sulla stessa. Per maggior chiarezza nella notazione, chiameremo queste due tabelle rispettivamente `Scacchiera` (in origine *%ENV_scacchiera*) e `Scacchi` (in origine *%ENV_scacchi*)

`Scacchiera` : tabella hash che rappresenta lo stato della scacchiera in ogni momento. Le chiavi sono costituite dalle lettere che identificano le colonne della scacchiera, mentre i valori associati alle chiavi sono puntatori a liste di otto elementi i quali, a loro volta, corrispondono alle otto caselle di ogni colonna. Ogni lista contiene, in ordine, i simboli dei pezzi che sulla scacchiera risiedono sulla colonna corrispondente alla chiave ("**" corrisponde a casella vuota, "WR" identifica una torre bianca e così via).

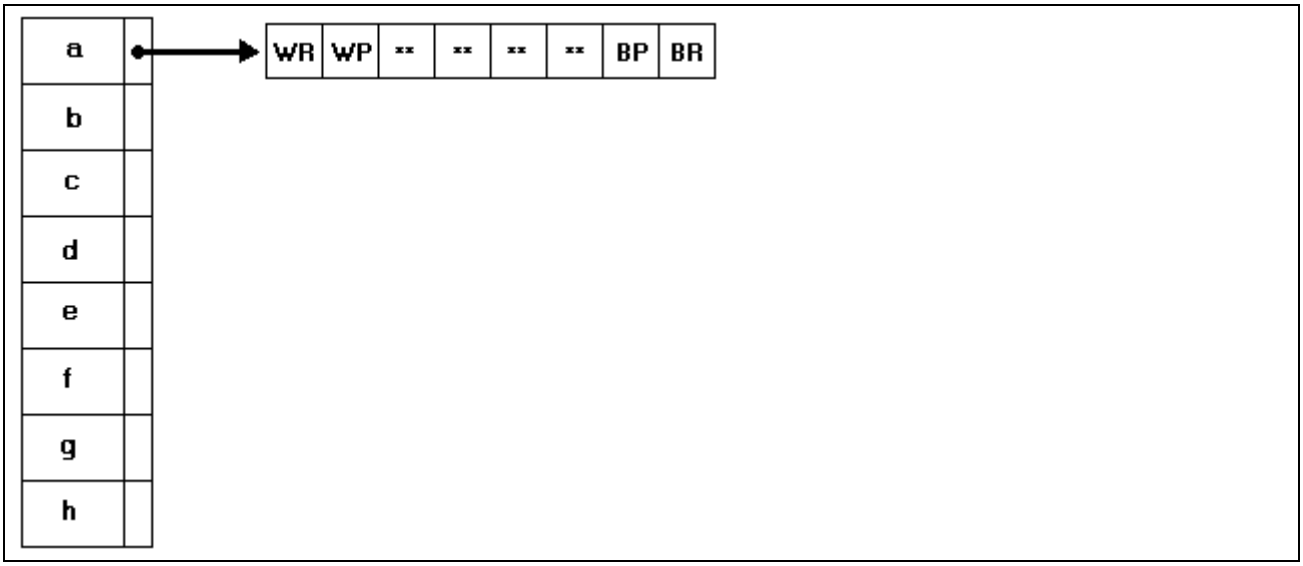


Figura 4.1: schema strutturale della tabella hash Scacchiera.

In questo modo è possibile interagire velocemente con qualsiasi casella della scacchiera, controllandone o modificandone il contenuto in maniera immediata. La formula utilizzata per interrogare la struttura dati è `Scacchiera{'lettera colonna'}→[n-1]`, dove n è il numero di riga desiderato.

`Scacchi` : tabella hash che riporta tutte le posizioni delle tipologie dei pezzi presenti sul piano di gioco in ogni momento. Utile per sapere esattamente dove si trova un pezzo senza esplorare tutta la scacchiera, o per sapere quanti pezzi di un certo tipo sono ancora in gioco.
Le chiavi della tabella sono costituite dai nomi dei pezzi, mentre i corrispondenti valori sono puntatori a liste libere. Gli elementi di queste liste contengono stringhe di due caratteri che indicano le posizioni dei rispettivi pezzi secondo le cordinate colonna-riga (es.: "d5"). In questo modo, ad esempio, scrivendo `Scacchi{'WP'}` ottengo la lista delle posizioni di tutti i pedoni bianchi sulla scacchiera.

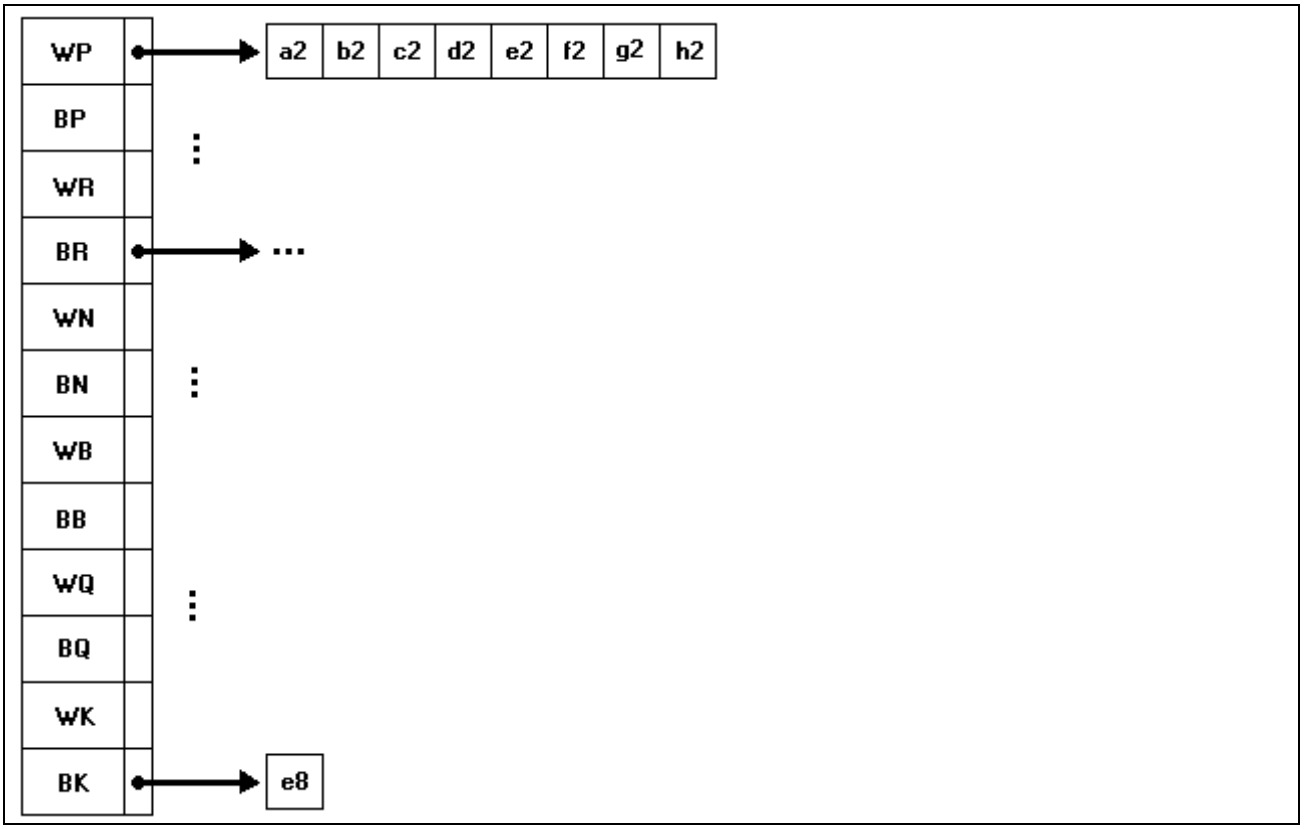


Figura 4.2: schema strutturale della tabella hash Scacchi.

Come è ben visibile nelle figure 4.1 e 4.2, i nomi dei pezzi scacchistici sono costituiti da stringhe di due caratteri: il primo carattere rappresenta il colore, mentre il secondo la tipologia in base alla notazione anglosassone (K=king, Q=queen, ecc.).

4.2 Algoritmi

Come detto in precedenza, il cuore della libreria *PGN_parser.pm* è costituito dalla funzione *Parse_Game*, le cui mansioni principali sono:

- Inizializzazione delle strutture dati: viene inserita la configurazione di apertura della partita in `Scacchiera` e `Scacchi`.

- Gestione dei token: le mosse in notazione SAN vengono scandite una ad una e trasformate in token, successivamente utilizzati per le operazioni di parsing.

- Parsing bottom-up di ogni token, con relativa gestione dei simboli speciali "P", "K", "Q", "N", "R", "B", "x", "O-O", "O-O-O", "=" e gestione delle informazioni di disambiguazione delle mosse.

- Gestione separata di arrocco e promozione.

Vediamo nello specifico ognuno di questi compiti, analizzando gli algoritmi che ne consentono l'assoluzione.

4.2.1 Generazione dei token

Ogni mossa di n caratteri estrapolata dal file PGN viene convertita in una lista `Token` di n elementi, dove ogni elemento contiene un singolo carattere della mossa stessa. In testa al token viene inserito un carattere "P", in previsione di eventuali mosse del pedone ed in considerazione del fatto che spesso tale carattere, nella notazione SAN, viene omesso.
Dal token vengono eliminati eventuali caratteri "+" e "#", che hanno scopo puramente di commento.

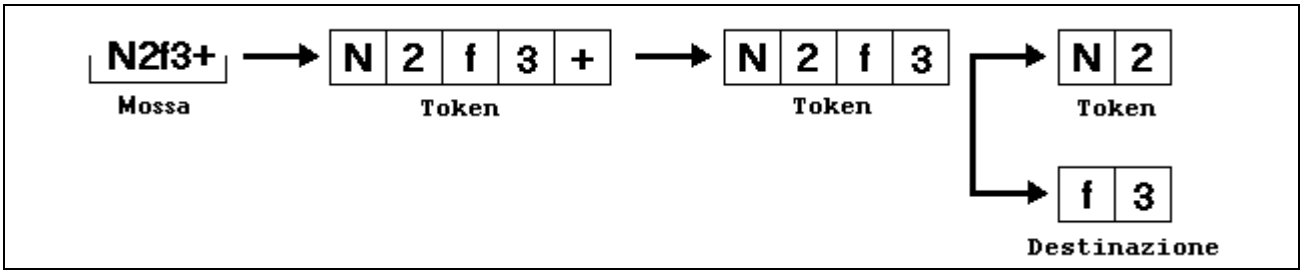


Figura 4.3: la mossa in formato SAN viene trasformata in token, dal quale vengono successivamente epurati i simboli "+" e "#". Al termine del processo Token contiene solo il simbolo del pezzo da muovere e le informazioni di disambiguazione, mentre Destinazione contiene le coordinate colonna-riga della casella di destinazione della mossa.

Allo stato attuale, `Token` contiene il simbolo del pezzo da muovere, eventuali informazioni di disambiguazione e la locazione di destinazione della mossa. In particolare, la destinazione è identificata dagli ultimi due

elementi del token, che forniscono le coordinate colonna-riga della casella dove il pezzo in analisi deve essere spostato.
Per velocizzare il processo di parsing e facilitare il reperimento delle informazioni di destinazione, queste ultime vengono estratte da `Token` e memorizzate in una struttura dedicata che chiameremo `Destinazione`.
In figura 4.3 è possibile vedere la schematizzazione del processo sopra illustrato.

4.2.2 Operazione di parsing

Il parsing viene effettuato su ogni singolo token utilizzando un approccio bottom-up, cioè iniziando la scansione dall'ultimo elemento e risalendo via via fino al primo.
Durante questa operazione si possono verificare due possibilità:

- Viene incontrato un simbolo speciale "P", "K", "Q", "N", "R", "B" o "x": in questo caso viene richiamata la relativa routine di gestione del simbolo.

- Viene incontrato un simbolo che non appartiene all'insieme dei simboli speciali: si tratta indubbiamente di un'informazione di disambiguazione, che viene estratta dal token ed inserita in un'apposita struttura che chiameremo `Partenza`. L'operazione di parsing riprende dal successivo elemento del token.

Vediamo quindi come si comportano le varie routine di gestione dei simboli speciali. Per quanto riguarda i pezzi scacchistici, le funzioni hanno tutte una struttura comune:

1. Le regole di movimento del pezzo sono state formalizzate attraverso precise formule matematiche, scritte in funzione delle coordinate colonna-riga della scacchiera.

2. In funzione del pezzo in oggetto e del giocatore detentore del turno di gioco (W o B) è possibile reperire la lista di tutti i probabili candidati alla mossa (le loro coordinate), sfruttando la tabella `Scacchi`.

3. Per ogni candidato si effettua un controllo utilizzando le formule introdotte al punto 1: data la posizione attuale del pezzo si verifica se questo è in grado di raggiungere o meno la posizione indicata da `Destinazione`. In questo modo si ottiene una prima scrematura, restringendo il numero dei candidati possibili.

4. Il controllo viene passato alla funzione `Move`, che si occupa della vera e propria movimentazione dei pezzi applicando anche, dove necessario, le regole di disambiguazione.

5. La computazione si conclude con una chiamata alla funzione `Resume`, il cui compito è quello di resettare tutte le variabili globali utilizzate sinora (Token, `Destinazione`, `Partenza`) e di cambiare il turno dei giocatori, preparando il programma all'analisi di un nuovo token.

Alla luce di quanto detto vediamo che merita particolare attenzione la funzione `Move`, solamente menzionata al punto 4, la cui interfaccia si presenta come segue: *Move (pezzo, candidati, non_candidati)*, dove *pezzo* indica il simbolo del pezzo da muovere sulla scacchiera, *candidati* è la lista delle posizioni degli scacchi di tipo *pezzo* che possono concorrere alla mossa e *non_candidati* è la lista delle posizioni degli scacchi di tipo *pezzo* che invece sono esclusi a priori dalla mossa.
Quando `Move` viene invocata si possono verificare due casi:

1. Il candidato alla mossa è univoco, quindi si procede direttamente all'aggiornamento delle strutture dati. In `Scacchiera` viene riportata la mossa effettuata, sfruttando le informazioni in `Destinazione` e le coordinate del pezzo candidato. In `Scacchi` viene sostituita la lista delle posizioni inerenti al pezzo mosso: la nuova lista è composta da *non_candidati* (le cui posizioni sono indubbiamente invariate) e dalla nuova posizione assunta dal pezzo mosso.

2. I candidati alla mossa sono molteplici, quindi è necessario applicare le regole di disambiguazione. Si sfruttano le informazioni aggiuntive estrapolate dal token e memorizzate in `Partenza` per eliminare i candidati che non rispondono ai requisiti. La funzione si richiama ricorsivamente aggiornando opportunamente le liste *candidati* e

non_candidati: un candidato scartato dalle regole di disambiguazione diverrà un non candidato. Il procedimento si ripete fintantoché non è possibile ricondursi al punto 1, dove il candidato è univoco.

Rimane ora da analizzare la routine di gestione del simbolo speciale "x", che identifica una mossa di presa. La sua funzione si riduce ad un semplice aggiornamento delle strutture dati: è necessario eliminare un pezzo dal piano di gioco. `Destinazione` contiene le coordinate del pezzo che viene preso, quindi la routine identifica tale pezzo e lo elimina da `Scacchiera`: il simbolo del pezzo viene sostituito da "**", che rappresenta la casella vuota. Successivamente viene aggiornata anche la tabella `Scacchi`, dove le coordinate del pezzo eliminato vengono tolte dalla rispettiva lista. A questo punto il parsing continua ed i rimanenti elementi del token vengono interpretati come uno spostamento normale di un pezzo su di una casella vuota.

4.2.3 Gestione arrocco e promozione

Le sequenze di caratteri "O-O", "O-O-O" e le sequenze contenenti il simbolo "=" vengono trattate separatamente dal parser. Una mossa di arrocco, sia esso lungo o corto, viene gestita nel seguente modo:

1. Vengono eliminati i caratteri "+" e "#" eventualmente presenti.

2. A seconda che il giocatore di turno sia W o B si considerano le rispettive coordinate delle torri e del re sulla scacchiera.

3. Si riconosce il tipo di arrocco, lungo o corto, ed in base a questa informazione si identifica univocamente la torre interessata allo spostamento.

4. I pezzi vengono spostati attraverso due differenti chiamate a `Move`, una per il re ed una per la torre, considerando il fatto che le posizioni di destinazione sono fisse, dettate dalla regola stessa di arrocco.

5. Il token viene così analizzato in un unico passaggio ("O-O" e "O-O-O" sono trattati come simboli unici) e quindi è necessaria una chiamata a `Resume` per preparare il parser all'analisi del token successivo.

Anche una stringa inerente alla una promozione di un pedone viene trattata come un simbolo unico (token con un solo elemento). Vediamo come essa viene analizzata dal parser:

1. Vengono eliminati i caratteri "+" e "#" eventualmente presenti.

2. Dalla stringa vengono estrapolate tutte le informazioni necessarie all'aggiornamento delle strutture dati, ad esempio: si supponga di avere la sequenza g8=Q, che identifica un pedone bianco che muove in g8 e viene promosso a regina. In questo caso g8 costituisce la `Destinazione`, mentre Q il nuovo pezzo che dovrà comparire sulla scacchiera.

3. Viene invocata una procedura "P", che si occupa di muovere effettivamente il pedone in `Destinazione`.

4. Viene invocata una procedura "x", che elimina il pedone mosso in `Destinazione`, aggiornando opportunamente le strutture dati.

5. Il pezzo a cui il pedone è stato promosso viene sistemato sulla scacchiera nella posizione `Destinazione` e la sua presenza viene segnalata introducendo in `Scacchi` la sua attuale posizione.

In figura 4.4 viene illustrato il diagramma di stato che rappresenta l'algoritmo di parsing così come è stato precedentemente descritto: la notazione di rappresentazione è UML, scelta per la sua chiarezza descrittiva, anche se la realizzazione non è stata sviluppata con tecnologie O-O.

4.3 Conclusioni

L'algoritmo di parsing così elaborato risulta essere veloce, ma presenta alcune limitazioni, come ad esempio l'incapacità di gestire mosse alternative o di elaborare un eventuale tag FEN presente nel file PGN in input.

Tutto il codice sviluppato in merito a questo software è consultabile all'appendice A.

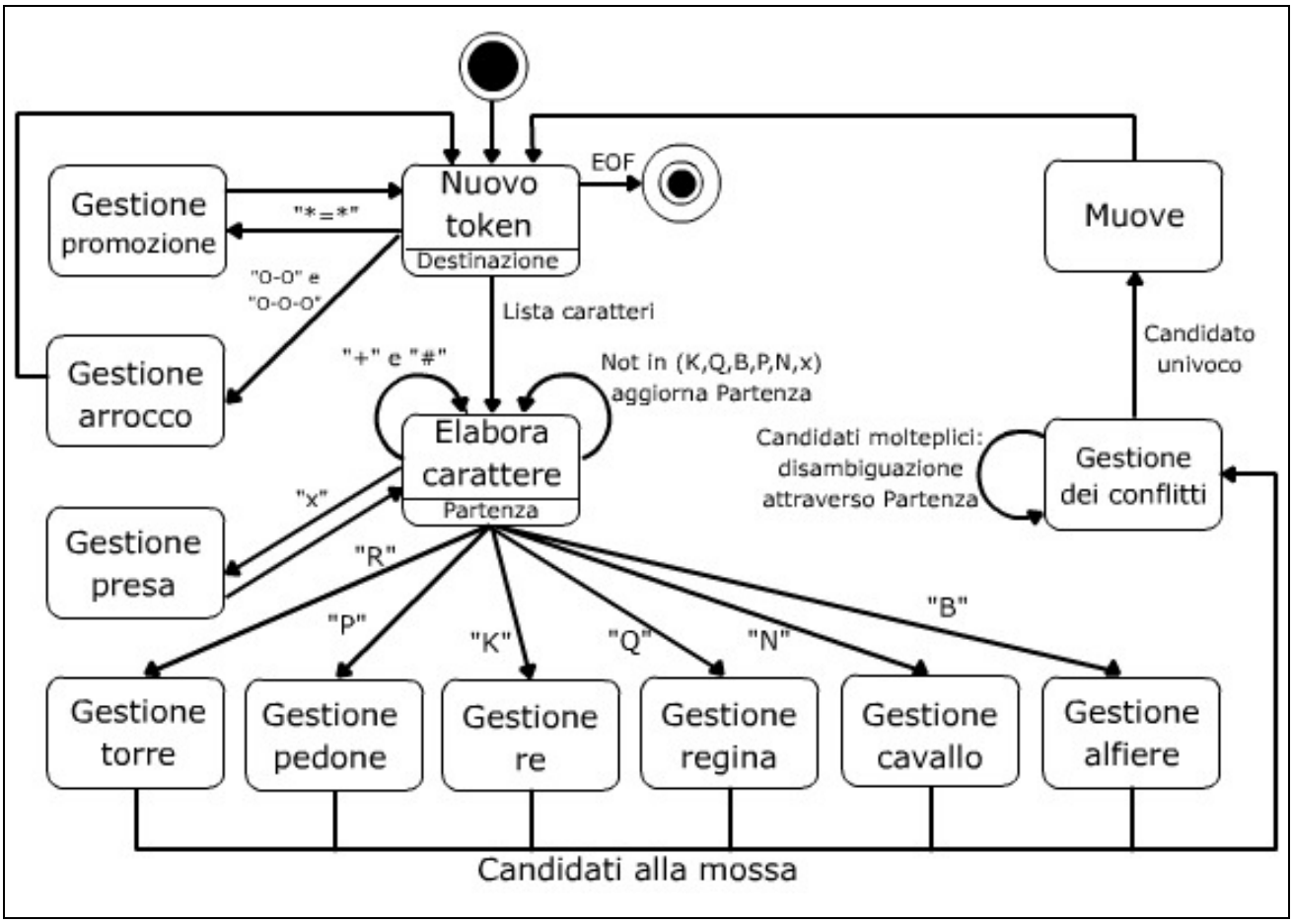


Figura 4.4: diagramma di stato rappresentante l'algoritmo di parsing secondo la notazione UML.

5 INTERAZIONE CON DOCUMENTI STATICI

La soluzione di seguito proposta ha come obiettivo la generazione di documenti digitali statici, contenenti diagrammi scacchistici e dotati di un meccanismo di interazione che permette all'utente di esplorare l'evoluzione di una partita a proprio piacimento. In particolare, il prodotto finale sarà un file in formato LIT di Microsoft Reader, che soddisfa a pieno l'aspetto di delivering, ma che preclude qualsiasi possibilità di editing.
Il passaggio dalla notazione PGN al formato finale LIT avviene attraverso una serie di trasformazioni intermedie, operate da più software interpellati in maniera sequenziale.
La figura 5.1 illustra lo schema globale del processo di elaborazione, evidenziando tutti i moduli che entrano in gioco.

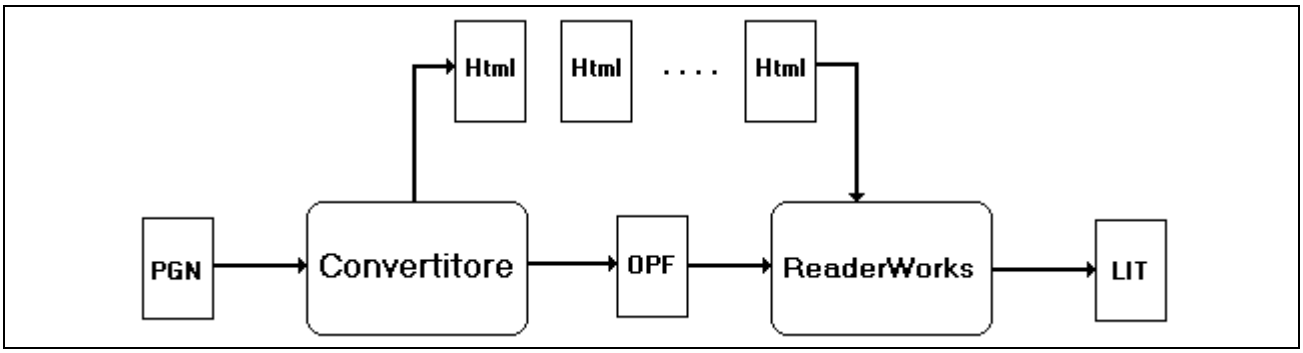


Figura 5.1: schema di funzionamento del sistema di creazione dinamica di documenti LIT.

Analizzando il diagramma è possibile vedere come il file PGN venga preso in input dal blocco *Convertitore*, il quale fornisce in output una sequenza di file HTML ed un package file (OPF). Quest'ultimo viene dato in input a ReaderWorks, il quale è così in grado di mappare tutti i sorgenti necessari alla produzione del libro elettronico e generare il file LIT finale.
La scelta di utilizzare HTML come formato intermedio di trasformazione è stata dettata da diversi fattori:

- E' uno dei pochi formati accettati da ReaderWorks per la generazione di ebook.

- Un file HTML è semplice da costruire, specialmente attraverso le potenzialità di Perl (codice generato direttamente a livello di programmazione).

- Consente di integrare nel documento elementi grafici e collegamenti ipertestuali in maniera semplice e diretta.

Analizziamo più a fondo ciò che finora è apparso solo come una scatola nera, il vero e proprio cuore del sistema e cioè il blocco *Convertitore*. La maggior parte delle operazioni compiute da questo modulo software, implementato nel file *main.pl*, si basano sull'uso intensivo della libreria *PGN_Parser.pm*:

1. Utilizzando la funzione *PGN_Parse()* acquisisce ed analizza il file PGN contenente la partita di scacchi. Questo file deve essere denominato "Game.pgn" e collocato nella medesima directory di lavoro del software.

2. Attraverso la funzione *Parse_Game()* scandisce ogni mossa, analizzando la configurazione della scacchiera passo per passo. Per ogni turno di gioco crea due pagine HTML nell'apposita directory "Output", una per la mossa del giocatore bianco ed una per la mossa del giocatore nero (ad esempio "Pagina10W.htm" rappresenta lo sviluppo della partita fino al decimo turno dopo la mossa del giocatore bianco). Ogni pagina riporta tre sezioni: un'intestazione contenente i dati della partita (nomi dei giocatori, luogo, data, ecc.), la rappresentazione grafica dello stato della scacchiera per la data mossa ed un menù per la scelta delle mosse (ogni mossa elencata è un link alla corrispondente pagina che la rappresenta). In questo modo, da ogni pagina creata è possibile saltare ad un'altra qualsiasi, osservando lo sviluppo della partita ad ogni passaggio. Per la rappresentazione dei pezzi sul piano di gioco è stato utilizzata una particolare fonte scacchistica: "Alpha.ttf", di tipo True Type.

3. Sfruttando un modello predefinito costruisce il package file che descrive il nome e la locazione delle pagine HTML generate al passo 2. Il modello generico del file descrittore, *modello_package-file.opf*, ricalca la struttura classica di un file di questo tipo, ma in esso sono state inserite delle ancore in corrispondenza dei tag "manifest" e "spine", che ne consentono una semplice e rapida compilazione. Vediamo uno stralcio di codice tratto dal modello stesso e riportato in figura 5.2 come esempio dell'utilizzo delle ancore.

Attraverso le potenti funzionalità di sostituzione di pattern offerte da Perl, il software è in grado di sostituire le ancore con le informazioni vere e proprie in un'unica riga di comando.

```
<manifest>##manifest##</manifest>
<spine>##spine##</spine>
```

Figura 5.2: porzione del codice contenuto nel modello di package file che illustra l'utilizzo delle ancore per la compilazione.

Il package file ultimato viene memorizzato nella medesima directory "Output" dei file HTML.

Come anticipato precedentemente, per ottenere il file finale in formato LIT viene sfruttato l'operato di ReaderWorks, applicato al file .opf generato dal blocco di conversione.
La figura 5.3 illustra la trasformazione di uno dei file HTML generati in una pagina dell'ebook prodotto da ReaderWorks: sono visibili alcune differenze grafiche, specialmente nella conversione di alcuni colori.

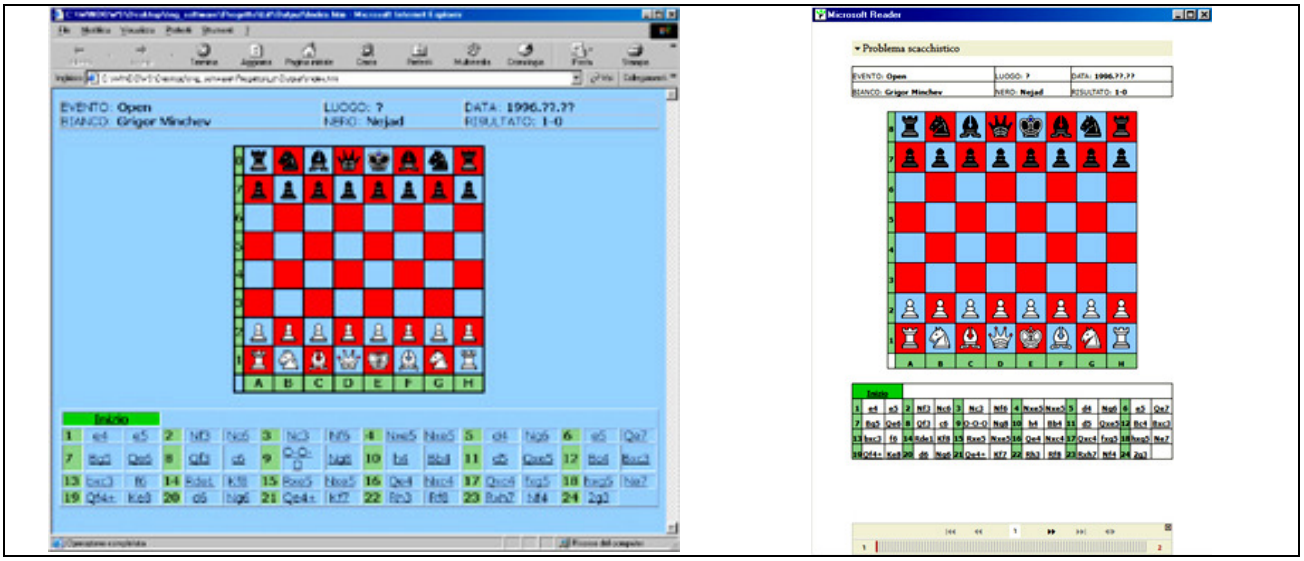


Figura 5.3: a sinistra una pagina HTML prodotta dal convertitore. A destra la medesima pagina convertita da ReaderWorks in LIT.

Un'ultima precisazione va fatta per quanto riguarda la generazione dinamica dei file HTML, che finora è stata data per scontata. A questo proposito è stata realizzata un'apposita libreria, *HtmlTools.pm*, costituita da un insieme di routine specifiche per la gestione separata

dell'intestazione, della scacchiera e del menù scelte di ogni pagina. Vediamo di seguito le funzioni che ne compongono l'interfaccia:

- *NuovaPagina()*: quando invocata restituisce la struttura basilare del documento HTML munita di un'apposita ancora per il successivo inserimento dei contenuti, come mostrato in figura 5.4.

```
<HTML><BODY bgcolor=#99ccff>##contenuto##</BODY></HTML>
```

Figura 5.4: codice HTML generato dalla funzione NuovaPagina().

- *Intestazione(game)*: costruisce una tabella di sei elementi, successivamente compilati con i dati della partita trattata (giocatore bianco, giocatore nero, data, ecc.). Questi dati sono contenuti nella tabella hash il cui puntatore, *game*, è passato come argomento alla funzione stessa.

- *Scacchiera()*: riproduce graficamente il piano di gioco attraverso una tabella. La posizione dei pezzi sulla scacchiera viene ricavata scandendo tutta la struttura dati `Scacchiera` opportunamente compilata dal parser PGN. Come già detto in precedenza, la rappresentazione degli scacchi è implementata attraverso l'utilizzo di una particolare fonte digitale.

- *MenuScelte(mosse)*: consultando la lista delle mosse passata come parametro di input costruisce una tabella, le cui dimensioni sono direttamente proporzionali alla lunghezza della lista stessa. In questa tabella vengono collocati gli indicativi numerici dei turni di gioco e, conseguentemente, le relative mosse effettuate. Ogni mossa è un link ipertestuale alla pagina HTML che la rappresenta.

Figura 5.5: tabella di navigazione dei contenuti generata da HtmlTools.pm

5.1 Considerazioni

Questa soluzione presenta un buon grado di interattività, in quanto l'utente può esplorare tutto il contenuto del documento finale a suo piacimento, in modo veloce e chiaro. Le dimensioni del documento LIT finale risultano sempre molto contenute, anche quando ci si trova di fronte a partite con un elevato numero di mosse e quindi ad un ebook con molte pagine. Tuttavia non vengono offerte opportunità di editing, se non limitate alla variazione manuale dei file PGN sorgenti.
Il documento finale è statico, cioè le pagine vengono generate tutte in una volta: questo fatto è strettamente legato alla natura del formato LIT, la cui creazione richiede assolutamente tutti i componenti a priori, senza permettere mutazioni successive (a meno di ricreare da capo il documento).
Facendo un parallelo con alcune delle soluzioni viste nel capitolo 3, questa realizzazione è concettualmente paragonabile a RenderX o DocBook, anche se in più offre quei meccanismi interattivi che gli esempi citati non riportavano.
Una nota di demerito va registrata a carico dei lunghi tempi computazionali necessari a ReaderWorks per creare il libro elettronico. In base a numerose prove effettuate, è stato stimato che il software di conversione della Overdrive impiega mediamente sette secondi per processare una singola pagina HTML e tempi anche più lunghi se il sorgente è un documento .doc di Microsoft Word. E' stato appurato che la maggior parte del tempo computazionale viene impiegato per il controllo e la traduzione dei collegamenti ipertestuali tra le pagine: la certezza ne deriva dal fatto che costruendo un documento Word contenente solo fonti scacchistiche (analogo sarebbe stato per un file HTML), il tempo impiegato per analisi e trasformazione si è abbassato drasticamente al di sotto del secondo.

Per una descrizione completa del codice sviluppato si consiglia di consultare l'appendice B.

6 INTERAZIONE CON DOCUMENTI DINAMICI

Compiamo ora un passo avanti, analizzando l'aspetto della dinamicità nella creazione di documenti digitali scacchistici. Lo scopo della soluzione di seguito proposta è quello di fornire un meccanismo tale per cui i documenti vengano fisicamente generati solo su richiesta dell'utente e contengano solo le informazioni che interessano all'utente stesso.
Per realizzare quanto prefissato si prende in analisi uno dei più diffusi word processor sul mercato: Microsoft Word. Il concetto alla base di questa realizzazione è l'utilizzo di Word come browser, sfruttando la capacità del software di leggere file in formato HTML.
In figura 6.1 è illustrato lo schema di funzionamento del sistema in analisi.

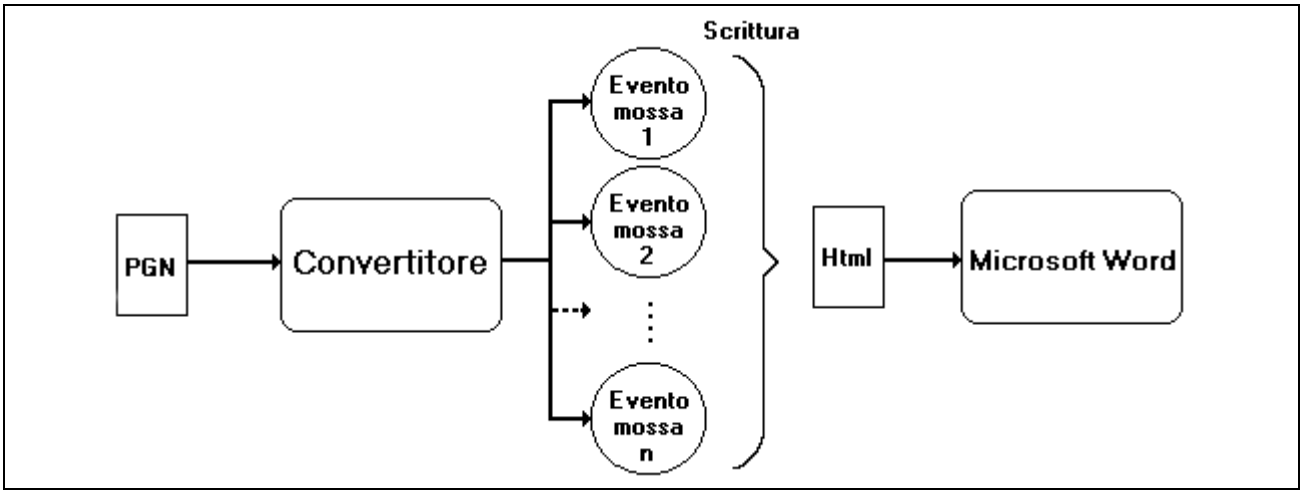


Figura 6.1: schema di funzionamento del sistema di creazione dinamica di documenti per MS Word.

Facendo un parallelo con la precedente realizzazione per MS Reader, vediamo che anche in questo caso sussiste un processo di trasformazione che conduce dal sorgente PGN al formato finale, in questo caso un file HTML. Il protagonista di questa trasformazione è il modulo **Convertitore**, implementato nel file *main.pl*, il quale cattura in input un file PGN generando in risposta una serie di gestori di eventi. In particolare, ogni gestore è un programma Perl indipendente e ne vengono creati tanti quante sono le mosse contenute nella partita acquisita, ad esempio: *Evento10B.pl* è il gestore assegnato alla mossa del giocatore nero al decimo turno di gioco.

6.1 Interfaccia grafica

Il convertitore implementato offre una succinta, ma esaustiva interfaccia grafica realizzata attraverso la release della libreria Tk per Perl [PTK]. L'aspetto dell'interfaccia viene riportato in figura 6.2, dove sono evidenziate e numerate tutte le parti che la compongono.

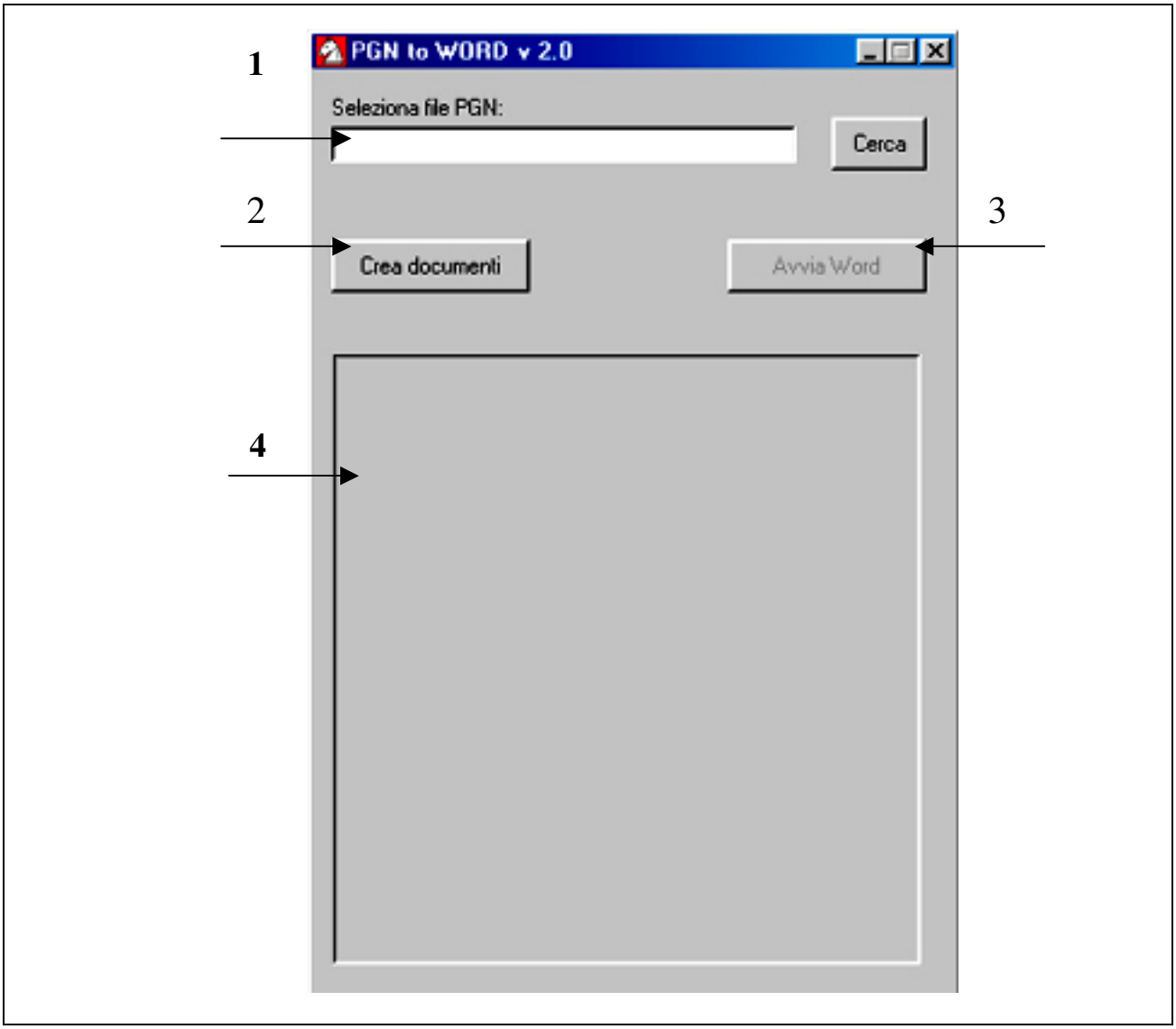


Figura 6.2: interfaccia del convertitore per MS Word.

Analizziamo le funzionalità in base alla numerazione di figura:

1. Casella dove viene indicato il nome del file PGN da acquisire. Se nessun percorso viene specificato la ricerca è eseguita nell'attuale directory di lavoro, altrimenti è possibile usufruire del tasto "Cerca" per esplorare le risorse del computer.

2. Pulsante per la creazione dei gestori di eventi inerenti alle mosse trovate nel file PGN. I file generati vengono salvati nella directory "Output".

3. Pulsante per avviare la lettura del documento in Word. Questo oggetto si attiva solo ed esclusivamente dopo che i gestori sono stati creati.

4. Finestra di riepilogo: mostra l'evoluzione delle operazioni di creazione dei gestori.

6.2 Modulo Convertitore

E' stato implementato attraverso la funzione *Crea()* presente nel file *main.pl*. Questa funzione viene chiamata in risposta ad un evento di pressione del pulsante 2 di figura 6.2 e compie le seguenti azioni:

1. Individua la directory di lavoro ed acquisisce la lista delle mosse dal file PGN utilizzando la libreria *PGN_Parser.pm*.

2. Acquisisce il modello base per la creazione dei gestori (*Modello_evento.txt*), il quale è costituito dal listato Perl del gestore arricchito con alcune ancore per semplificarne la compilazione.

3. Per ogni mossa riscontrata nel file PGN crea un gestore di evento utilizzando il relativo modello. Le ancore presenti in quest'ultimo vengono sostituite con dati fondamentali che permettono ad ogni gestore di conoscere le directory di lavoro e di output, il nome del file PGN a cui si fa riferimento ed i paramentri necessari al parsing. Come già anticipato, il nome di ogni file Perl generato è inerente alla mossa di cui esso è gestore, secondo il seguente schema:
   "Evento" + "*numero turno di gioco*" + "*colore giocatore*".

6.3 Gestori di eventi

Un gestore di evento è un programma Perl in grado di operare in maniera completamente indipendente. Il suo scopo è quello di creare una visualizzazione grafica di una certa mossa scacchistica, utilizzando una normale sessione di lavoro in Microsoft Word. L'evento da gestire è la

scelta, da parte dell'utente, di visualizzare una mossa ben precisa, individuata in un apposito menù che riporta tutte le mosse disponibli. Vediamo, nello specifico, cosa effettivamente un gestore di evento fa:

1. Utilizzando la libreria *PGN_Parser.pm* acquisisce ed analizza il file PGN contenente la partita di scacchi. In particolare, il gioco viene sviluppato fino alla mossa a cui l'evento è legato.

2. Costruisce la pagina HTML con la rappresentazione grafica della partita fino al punto sviluppato in 1. La struttura di ogni pagina riporta un'intestazione contenente tutti i dati della partita, un diagramma scacchistico ed un menù, ordinato per turni di gioco, con riferimenti a tutte le mosse. Ogni voce del menù, quindi, è un collegamento al gestore di evento relativo alla mossa descritta dalla voce stessa, una struttura del tutto simile a quella vista in figura 5.5.
   La generazione del codice HTML si ottiene tramite la medesima libreria *HtmlTools.pm* realizzata per la soluzione in MS Reader, nella quale viene utilizzata la fonte scacchistica "Alpha.ttf" per la rappresentazione dei pezzi sul piano di gioco.

3. Interagisce con Word: se esiste una finestra del word processor già aperta la cattura e ne chiude il contenuto, altrimenti ne apre una nuova.

4. Scrive il listato HTML costruito in 2 in un file temporaneo denominato *Source.tmp*.

5. Carica il file temporaneo in Word, sfruttando la finestra dell'applicazione aperta in 3.

L'interazione tra il sistema e l'utente può essere schematizzata come riportato in figura 6.3. In essa vediamo che l'utente può scegliere una qualsiasi delle mosse della partita, rappresentate nel file HTML da dei comuni link. Con un semplice click del mouse viene richiamato il gestore di evento corrispondente alla mossa selezionata, il quale chiude l'attuale sessione di "Source.tmp", aggiorna il suddetto file e lo riapre nella medesima finestra di Word. In altre parole Word viene utilizzato come un browser web, dove tutti i documenti vengono aperti e consultati in un'unica finestra.

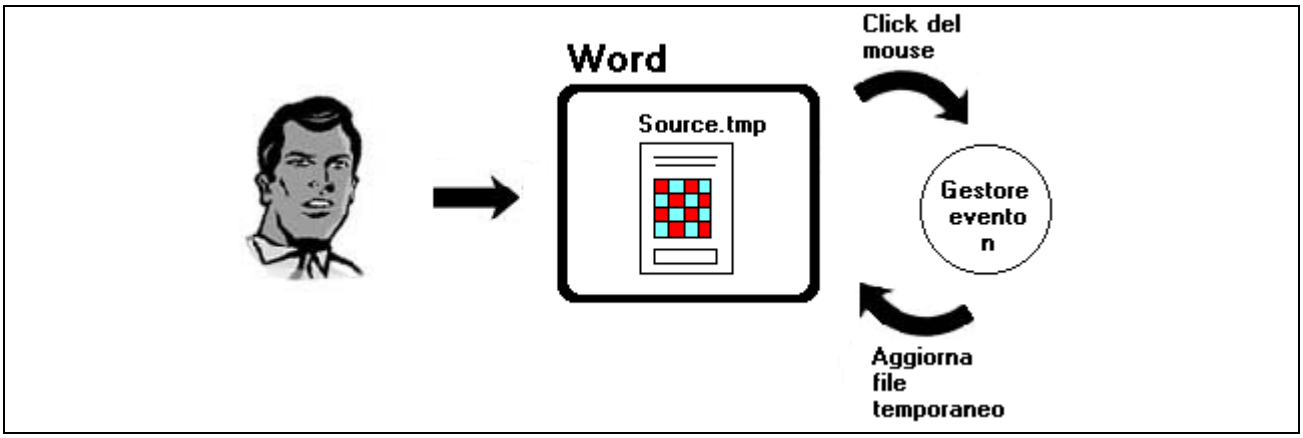


Figura 6.3: schema di interazione tra l'utente ed il sistema di generazione dei documenti elettronici in ambiente Word.

6.4 Osservazioni

6.4.1 Microsoft Word vs Microsoft Reader

Si può notare un'analogia tra il sistema sviluppato in ambiente Word e quello sviluppato per MS Reader: in entrambi i casi viene prodotto un documento per ogni mossa scacchistica che compare nel file PGN (pagine HTML per MS Reader e gestori di eventi per Word), ma mentre in ambiente MS Reader viene creata una pagina fisica per ogni mossa, in ambiente Word la pagina fisica è sempre e solo una. I gestori di eventi sono indubbiamente più leggeri, in termini di allocazione di memoria, di un documento HTML completo e consentono all'utente di analizzare solo le informazioni che più lo interessano, in quanto esse non esistono fisicamente fintantoché non è l'utente stesso a richiederle.
Il paragone fatto con la realizzazione per MS Reader può essere esteso a qualsiasi sistema software in grado di produrre solo documenti statici: basti pensare alla miriade di pubblicazioni scacchistiche in formato RTF reperibili in rete e renderizzabili anche in MS Word.
In definitiva, questa soluzione presenta un alto livello di dinamicità, senza soffrire di particolari tempi di ritardo dovuti alle frequenti interazioni col modulo di conversione.

6.4.2 Considerazioni tecniche

Il sistema realizzato in ambiente Word è da considerarsi più un esperimento, una dimostrazione di ciò che è possibile fare, piuttosto che una soluzione definitiva al problema della generazione dinamica di

documenti interattivi. Questo perché è proprio la natura stessa del word processor a porre dei limiti di realizzazione:

- Il sistema di sicurezza integrato in Word chiede una conferma ogni qualvolta si tenti di aprire un file esterno al documento. Questo significa dare una conferma manuale ogni volta che si attiva un gestore di evento.

- Avere n gestori per n mosse scacchistiche può creare situazioni difficili da gestire, specialmente per quanto riguarda la divulgazione dei documenti. Era stata affrontata l'idea di creare un unico gestore generico, in grado di visualizzare e gestire qualsiasi file PGN, ma l'impossibilità di effettuare passaggio di parametri dall'ambiente Word a programmi esterni ha posto un pesante vincolo di progettazione (il passaggio di parametri a programmi locali è impossibile, mentre in rete attraverso collegamenti ipertestuali è possibile).

L'aspetto interessante dell'esperimento sta nel poter utilizzare le funzionalità del word processor sui contenuti generati, potendo così editare, salvare in formato diverso o quant'altro le pagine HTML visualizzate.

La gestione di un oggetto Windows come l'applicativo Word, attraverso un linguaggio tipicamente non ad oggetti come Perl, è stata possibile attraverso la libreria *Win32-OLE* [MSW]. Tale libreria si comporta come un vero e proprio middleware, fornendo funzioni che si interfacciano alle primitive Visual Basic per la gestione degli oggetti (anche nella forma assomigliano a funzioni Visual Basic).

Tutto il codice sviluppato per il sistema di creazione dinamica di documenti interattivi in ambiente Microsoft Word è visionabile all'appendice C.

7 EDITING DI DOCUMENTI INTERATTIVI AUTOMODIFICANTI

Il percorso logico che ci ha sin qui condotti, è basato sull'analisi dei principali aspetti di cui un sistema di generazione dinamica di documenti digitali interattivi deve tener conto: dinamicità (creazione di documenti statici o dinamici), interattività (documenti dotati di meccanismi di interazione), delivering (portabilità dei documenti per la loro divulgazione), editing (meccanismi per la modifica dei contenuti di un documento).
Considerando che le realizzazioni sinora incontrate soddisfano solo sottoinsiemi delle caratteristiche sopra citate, è naturale pensare che il passo conclusivo del nostro percorso consista nel fornire una soluzione in grado di soddisfarle tutte.
In figura 7.1 è possibile vedere lo schema che descrive l'interazione tra l'utente ed un sistema di creazione di documenti interattivi sviluppato a scopo prettamente di delivering (come quelli descritti nei capitoli 5 e 6).

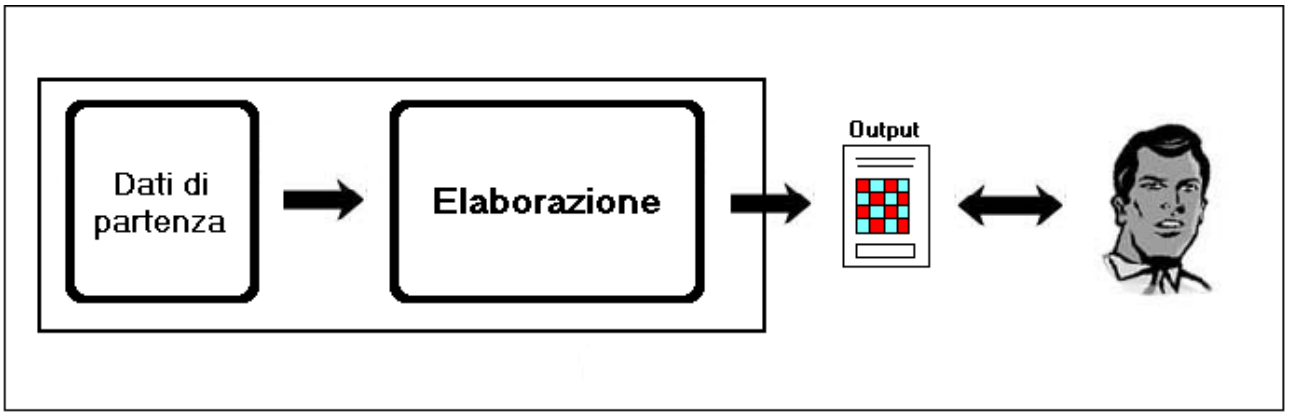


Figura 7.1: schema delle interazioni per un sistema di delivering.

L'interazione avviene solamente con il documento finale, mentre i *dati di partenza* (nel nostro caso i file PGN generati da terze parti) e le relative *elaborazioni* su di essi effettuate appaiono all'utente come una scatola nera, di cui non conosce il contenuto.
La soluzione di seguito presentata, denominata Chess Editor, è stata realizzata sfruttando la tecnologia Perl/Tk, già incontrata nella realizzazione per Word, che fornisce tutti i meccanismi per la gestione delle finestre e degli oggetti ad esse correlati (caselle di testo, menu, ecc.)

relativamente ai principali sistemi operativi in commercio (in questo caso il software è stato sviluppato su piattaforma Windows). Vediamo di seguito una panoramica delle caratteristiche più interessanti del programma, sottolineando le capacità di editing e di esportazione in vari formati:

- Interfaccia grafica per la gestione delle mosse scacchistiche (scacchiera virtuale), comprese anche le mosse particolari come arrocchi e promozioni.

- Le mosse effettuate sulla scacchiera virtuale vengono direttamente interpretate e formalizzate secondo lo standard SAN.

- Possibilità di correggere la sequenza delle mosse in qualsiasi momento.

- Possibilità di salvare la partita in formato PGN e/o PDF.

- Possibilità di caricare una partita da un file PGN esterno (ai dati caricati è possibile applicare tutti i precedenti punti).

In figura 7.2 viene mostrato come cambiano le interazioni tra utente e sistema quando si ha a disposizione uno strumento come Chess Editor.

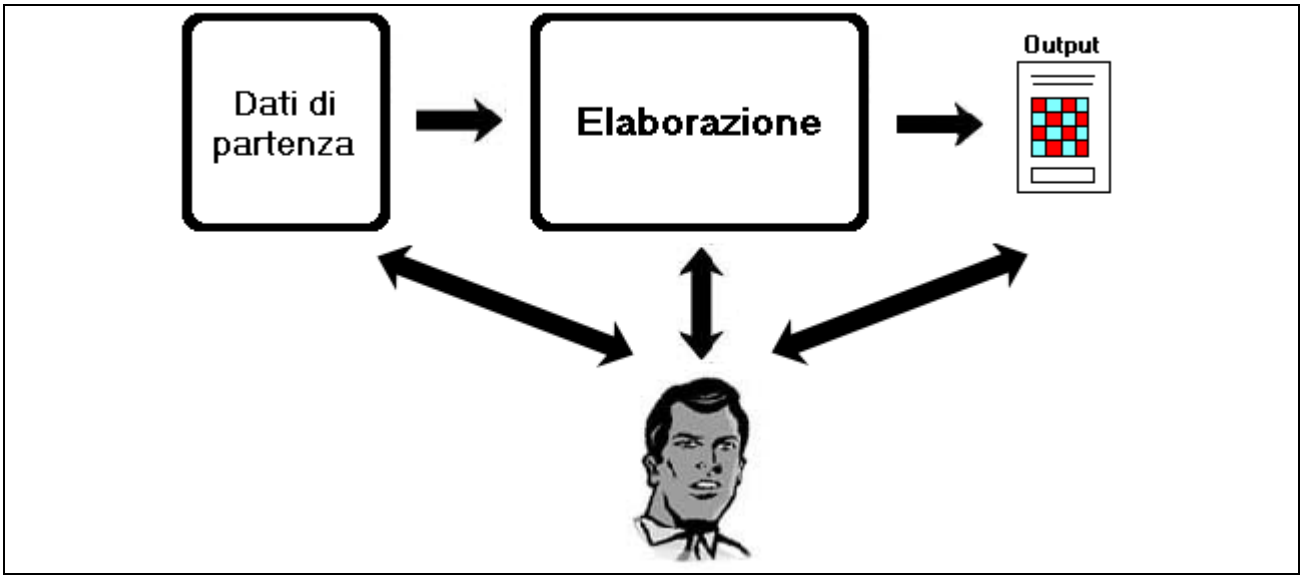


Figura 7.2: schema delle interazioni per un sistema di editing e delivering.

7.1 Interfaccia

L'interfaccia grafica di Chess Editor risulta essere semplice ed immediata, come è visibile anche in figura 7.3, nella quale sono distinguibili cinque diverse regioni d'interesse.

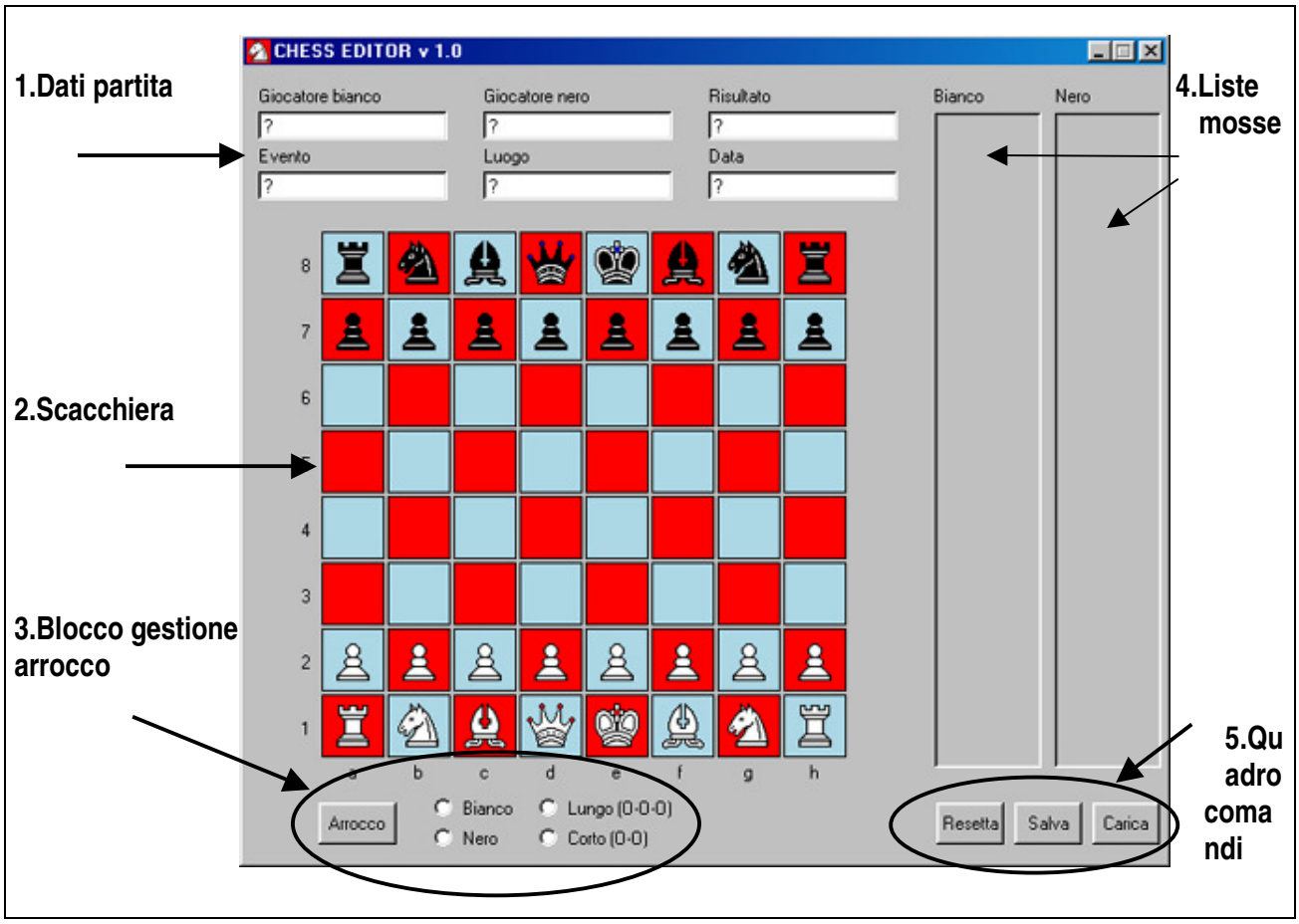


Figura 7.3: schermata di presentazione di Chess Editor.

Analizziamo i componenti dell'interfaccia in base alla numerazione fornita in figura:

1. Dati partita: rappresentano fondamentalmente i sei principali tag PGN contenenti i dati descrittivi della partita. Quando il programma viene lanciato, queste caselle appaiono pre-compilate con il simbolo "?", che è il simbolo standard utilizzato in PGN laddove i dati in questione siano assenti.

2. Scacchiera: è la scacchiera virtuale sulla quale i pezzi vengono spostati in osservanza delle relative regole di movimentazione.

3. Blocco gestione arrocco: in base alla combinazione scelta (es.: giocatore bianco, arrocco lungo) gestisce le mosse di arrocco sulla scacchiera.

4. Liste mosse: in queste liste, una per il giocatore bianco ed una per il giocatore nero, vengono riportate e numerate le mosse effettuate sulla scacchiera, direttamente nel formato SAN.

5. Quadro comandi: insieme dei pulsanti di comando che gestiscono il reset della scacchiera (la scacchiera viene riportata alla configurazione iniziale), i salvataggi (apre la finestra di dialogo per salvare la partita in corso in formato PGN e/o PDF) ed i caricamenti (apre la finestra di dialogo per scegliere un file PGN dal quale caricare i dati).

7.1.1 Scacchiera virtuale

La movimentazione di un pezzo sulla scacchiera è realizzata in maniera visivamente semplice per l'utente, ma in realtà comporta una serie corposa di passaggi:

1. L'utente seleziona il pezzo da muovere cliccandoci sopra con il mouse: la casella sulla quale il pezzo selezionato risiede viene contrassegnata con un colore particolare.

2. Il programma riconosce la tipologia del pezzo selezionato ed in base a questa ne calcola le possibili mosse, tenendo anche conto di eventuali ostacoli lungo il cammino. Le caselle della scacchiera che potrebbero costituire un valido punto di arrivo per il pezzo in movimento vengono contrassegnate con un colore specifico.

3. Basandosi sulle caselle evidenziate sulla scacchiera l'utente può infine scegliere dove spostare il pezzo precedentemente selezionato, od eventualmente può selezionare un altro pezzo ripartendo così dal punto 1. Qualsiasi casella che non sia contrassegnata debitamente non viene ovviamente accettata come destinazione della mossa.

4. La scacchiera viene aggiornata tenendo conto dello spostamento avvenuto e la mossa viene riportata nella lista delle mosse in base al giocatore, bianco o nero, che l'ha effettuata.

5. Viene aggiornato il turno di gioco, passando la mano dal giocatore bianco a quello nero o viceversa a seconda di chi ha effettuato l'ultima mossa. Il programma inibisce la selezione dei pezzi che non appartengono al giocatore detentore del turno di gioco.

In figura 7.4 sono illustrati alcuni esempi di movimentazione sulla scacchiera virtuale, nei quali sono visibili le caselle evidenziate per sottolineare le possibilità di movimento dei pezzi selezionati.

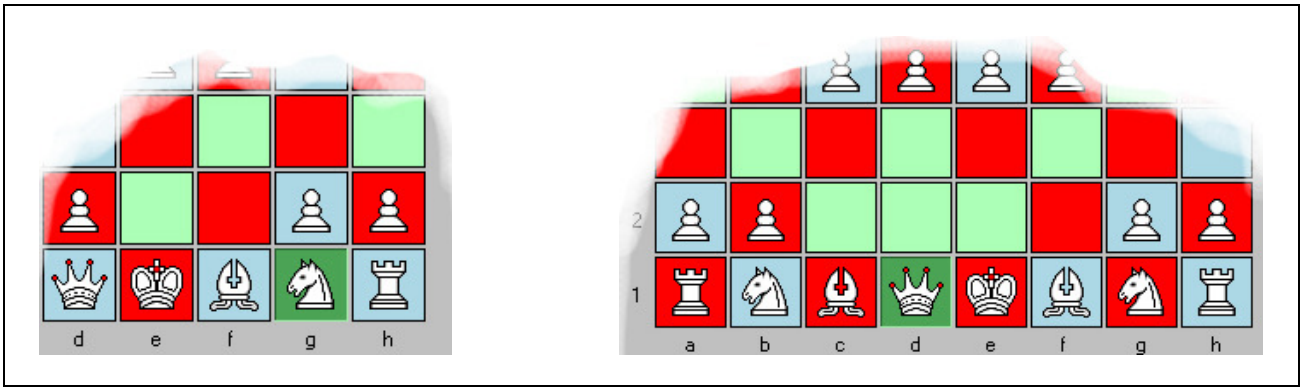


Figura 7.4: esempi di mosse. A sinistra è stato selezionato un cavallo, a destra una regina.

7.1.2 Gestione mosse di arrocco e promozione

Esistono alcune tipologie di mosse sulla scacchiera che richiedono un trattamento particolare: vediamo di seguito come Chess Editor gestisce arrocchi e promozioni.

Arrocco: viene gestito in maniera esplicita, cioè esiste un comando fisico apposito nella finestra del programma (figura 7.5). Questo comando permette di scegliere tra varie combinazioni, attraverso le quali l'utente decide il giocatore e la tipologia di arrocco coinvolti nella mossa (arrocco lungo o corto).

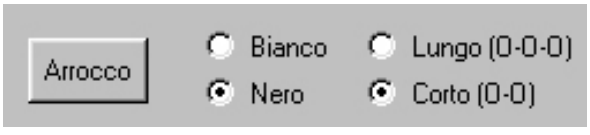


Figura 7.5: quadro di comando gestione arrocco.

Le caselle di selezione (radio button) che specificano una determinata voce sono mutuamente esclusive (es.: non posso scegliere contemporaneamente nero e bianco o lungo e corto).

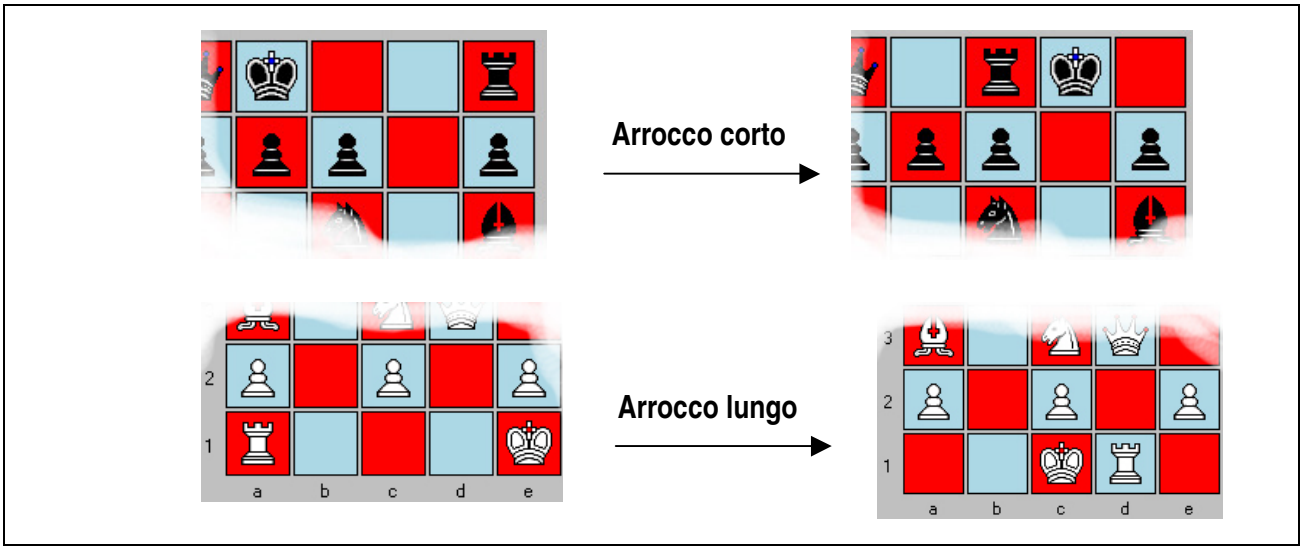


Figura 7.6: in alto il giocatore nero esegue un arrocco corto; in basso il giocatore bianco esegue un arrocco lungo.

Una volta impostati i dati corretti si può utilizzare il pulsante "Arrocco" per effettuare la mossa sulla scacchiera, ovviamente previo controllo della presenza o meno di ostacoli sul cammino dei pezzi: nel caso i prerequisiti richiesti per la mossa non siano presenti (non vi deve essere alcun ostacolo tra il re e la torre coinvolti nell'arrocco) il programma non compie alcuna azione e si mette in attesa di un ulteriore comando da parte dell'utente.
La mossa di arrocco viene riportata nella lista delle mosse dei giocatori con le stesse modalità di qualsiasi altra mossa.
In figura 7.6 sono rappresentati alcuni esempi delle tipologie di arrocco.

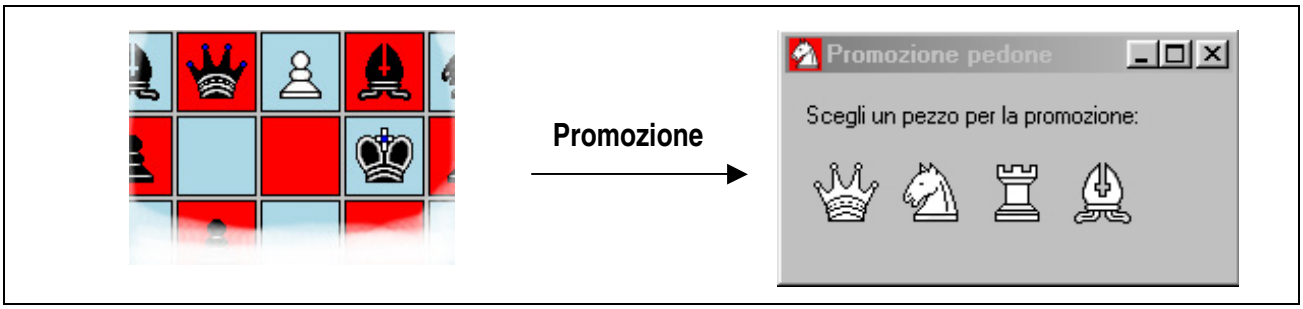


Figura 7.7: il pedone raggiunge l'estremità opposta della scacchiera (sinistra), si apre la finestra delle promozioni (destra) dove è possibile selezionare un pezzo per la relativa promozione.

Promozione: viene gestita in maniera implicita, cioè il programma è in grado di individuare automaticamente le casistiche in cui la promozione di un pedone entra in gioco. Quando un pedone raggiunge l'estremità opposta

della scacchiera (lato dell'avversario) si apre una finestra di dialogo nella quale è possibile selezionare il pezzo con il quale promuovere il pedone stesso.
Effettuata la promozione, questa viene formalizzata secondo la notazione SAN ed inserita nella relativa lista delle mosse.

7.1.3 Liste delle mosse

Le liste delle mosse risultano essere un elemento fondamentale dell'interfaccia di Chess Editor poiché, oltre a riportare l'elenco delle mosse effettuate sulla scacchiera, gestiscono l'accesso allo stack delle configurazioni di sistema.
Vediamo schematicamente le operazioni effettuate dal programma ad ogni mossa, in ordine cronologico:

1. A seconda del pezzo mosso costruisce la relativa formalizzazione SAN, che viene successivamente visualizzata come nuova riga nella lista corrispondente al giocatore che ha effettuato la mossa. Ogni riga è contrassegnata da un indice progressivo, che rappresenta il turno di gioco nel quale la mossa è stata eseguita.

2. "Congela" la configurazione della scacchiera, inserendola in un apposito stack.

3. Crea una relazione tra la nuova riga inserita nella lista delle mosse e la corrispondente configurazione della scacchiera nello stack.

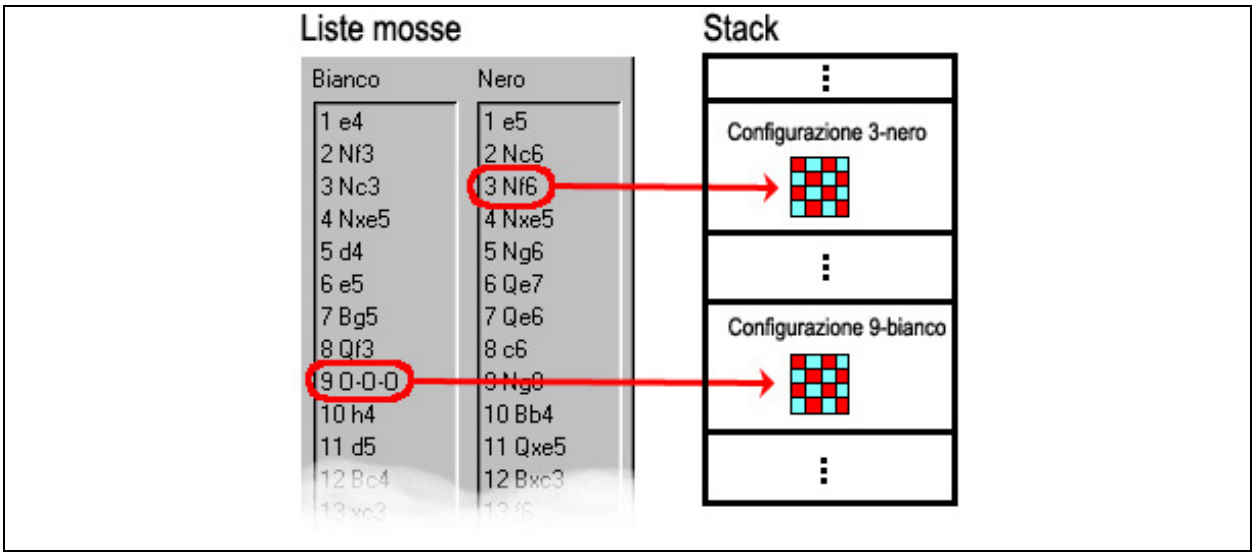


Figura 7.8: relazione tra liste delle mosse e stack configurazioni.

Attraverso questo meccanismo di "congelamento" delle configurazioni è possibile richiamare in qualsiasi momento lo stato della scacchiera corrispondente ad una determinata mossa. All'utente, infatti, è sufficiente eseguire un doppio click del mouse, su una qualsiasi delle mosse che compaiono nelle liste, per riportare la scacchiera indietro o avanti nella partita fino a quel momento costruita.
Vediamo come questa caratteristica offra un potente mezzo di correzione, analizzando un ipotetico scenario d'uso:

- L'utente si rende conto di aver commesso un errore nell'esecuzione della decima mossa del giocatore nero.

- Per correggere l'errore l'utente seleziona la mossa che precede quella errata nella sequenza e cioè la decima del giocatore bianco. La selezione avviene effettuando un doppio click sulla mossa in questione.

- Il programma carica la configurazione della scacchiera inerente alla decima mossa del giocatore bianco, portandosi così al passo precedente all'errore commesso.

- L'utente può ora ripetere la mossa del giocatore nero nel modo che ritiene più opportuno.

- La nuova mossa viene riportata nella lista delle mosse del giocatore nero, mentre tutte le mosse successive, sia di bianco che di nero, vengono cancellate (ovviamente anche lo stack viene aggiornato). Questo comportamento è giustificato dal fatto che il cambiamento effettuato a monte potrebbe portare a delle incongruenze se si lasciassero le mosse successive inalterate.

7.1.4 Quadro comandi

Il quadro comandi di Chess Editor è costituito da tre pulsanti, utili per compiere altrettante azioni di rilevante importanza. Vediamone di seguito una descrizione.

Resetta: risulta particolarmente utile in situazioni in cui il lavoro eseguito con Chess Editor risulti non soddisfacente od errato. La sua funzione è

proprio quella di resettare l'interfaccia riportandola allo stato iniziale, cancellando le liste delle mosse e gli eventuali tag PGN, ovviamente aggiornando le relative strutture dati.

Salva: quando utilizzato apre una finestra di dialogo, nella quale è possibile gestire tutte le opzioni relative al salvataggio della sessione di lavoro nei vari formati supportati. Vediamo di seguito una panoramica dei componenti che compaiono nella finestra di salvataggio:

1. Due caselle di controllo che permettono all'utente di selezionare il formato di output, attraverso diverse combinazioni possibili: solo PGN, solo PDF, PGN e PDF insieme.

2. Un pulsante "Cerca" che permette di allacciarsi al meccanismo di navigazione del file system presente sul sistema operativo in uso. In questo modo è più semplice indicare un percorso di salvataggio per il file di output.

3. Una casella di testo (etichettata come "Nome file:") nella quale è possibile specificare il nome del file di salvataggio ed il suo eventuale percorso. Se viene specificato solo il nome, il programma prende come directory di default per il salvataggio quella dove Chess Editor risiede. In particolare, se si è scelto di salvare in entrambi i formati i file di output avranno lo stesso nome, ma estensione diversa.

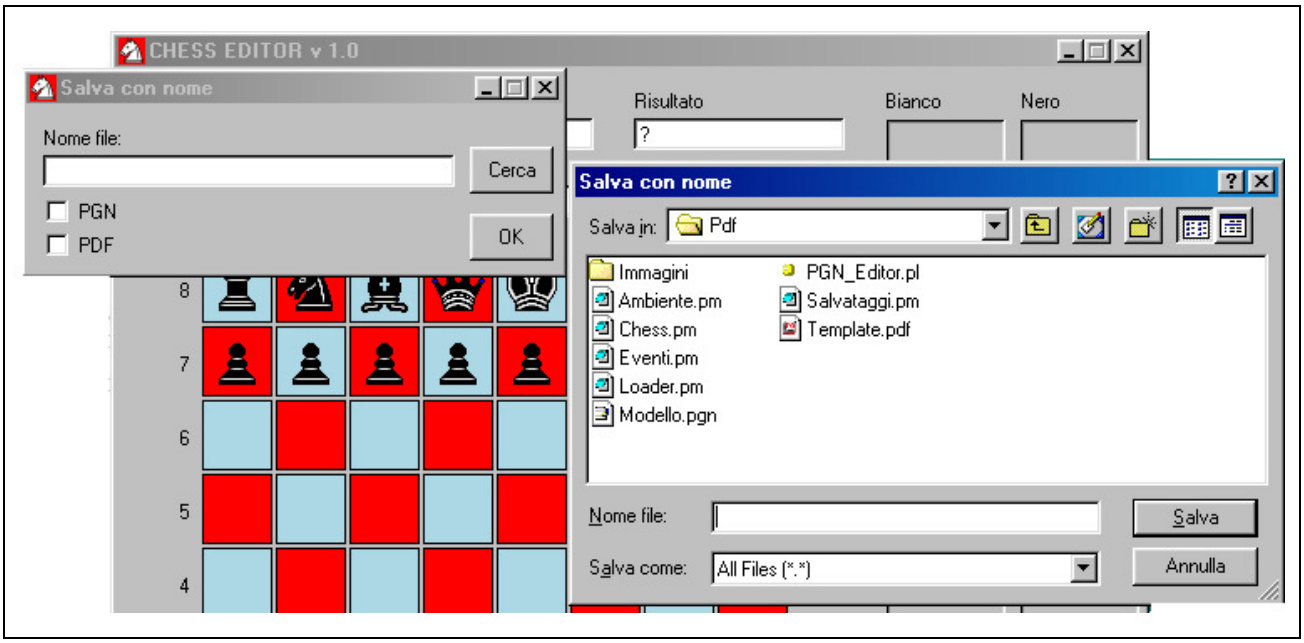


Figura 7.9: interfaccia per il salvataggio dati di Chess Editor.

4. Un pulsante “OK” che da inizio alla procedura di salvataggio. Se nessun nome di file viene specificato il programma non compie alcuna azione, restituendo il controllo alla finestra principale dell’applicazione.

Carica: la sezione di caricamento di Chess Editor consente all’utente di importare i dati di una partita da un file PGN esterno. Questa capacità fornisce diversi vantaggi:

- L’utente può interrompere la sua sessione di lavoro in qualsiasi momento, potendo salvare e successivamente ricaricare i dati della partita fino a quel momento sviluppata.

- L’utente può sfruttare una porzione della partita sviluppata da qualcun altro per crearne una propria: è sufficiente caricare i dati dall’esterno ed, utilizzando le potenzialità di Chess Editor viste sopra, apportare la correzione dal punto desiderato in poi.

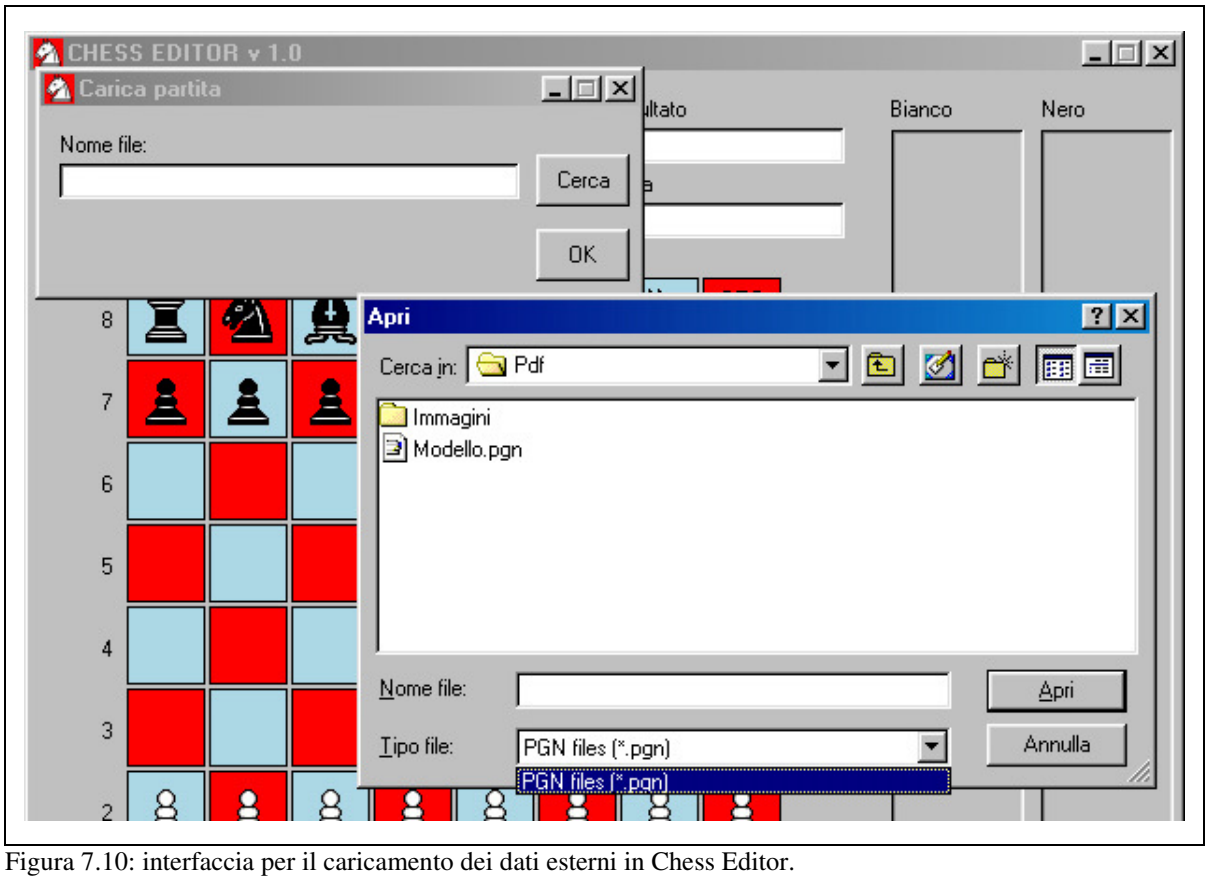


Figura 7.10: interfaccia per il caricamento dei dati esterni in Chess Editor.

- Capita spesso che nella sequenza di mosse in formato SAN vi siano dei dati superflui, ridondanti. Può accadere, ad esempio, che vengano

specificate righe e/o colonne di pezzi da muovere che non sono necessarie, poiché non vi sono ambiguità sulla mossa in questione. Chess Editor elimina automaticamente questi esuberi nel momento stesso della lettura del file di input.

In figura 7.10 è visibile come si presenta la sezione di caricamento di Chess Editor. Anche in questo caso si sfrutta il meccanismo di navigazione del file system del sistema operativo in uso, in modo da rintracciare velocemente il file che interessa. In particolare viene messo in evidenza il fatto che la possibile scelta dal file system si riduce ai soli file che hanno come estensione ".pgn".
La casella di testo etichettata come "Nome file:" permette eventualmente di inserire direttamente il nome del file da caricare, nel caso esso ed il relativo percorso siano già noti a priori.
Come nel caso del salvataggio, il pulsante "OK" da inizio all'operazione di caricamento: se il programma rileva che non è stato specificato alcun nome di file restituisce il controllo alla finestra principale, senza compiere alcuna altra azione.

7.2 Principali strutture dati

La struttura dati principale alla base di Chess Editor è una tabella hash utilizzata per la rappresentazione dello stato della scacchiera e che chiameremo appunto, per ragioni di semplicità, `Scacchiera`. La sua morfologia è simile a quella dell'analoga rappresentazione incontrata nel parser PGN, ma con alcune modifiche che la rendono più sofisticata e potente. In questa tabella le chiavi sono costituite dalle lettere delle colonne della scacchiera, mentre i valori associati alle chiavi sono puntatori a liste di otto elementi. In questo modo si copre l'intera struttura della scacchiera fisica, con otto caselle per ognuna delle otto colonne presenti. Ogni elemento delle liste associate alle chiavi è costituito da un puntatore ad una tabella hash, che a sua volta contiene i dati di descrizione della casella della scacchiera corrispondente.
Come è visibile in figura 7.11 ogni casella è caratterizzata da diversi attributi:

- **Pezzo**: riporta la stringa che identifica il pezzo contenuto nella casella. La notazione è identica a quella vista per il parser PGN ("WR", "BQ", "**", ecc).

- **Ptr**: puntatore al vero e proprio oggetto casella nell'interfaccia grafica (scacchiera virtuale).

- **Bg**: colore dello sfondo nella forma #RRGGBB. Identifica il colore di riempimento della casella nell'interfaccia grafica.

- **Sel**: indica se la casella è stata selezionata o meno durante l'operazione di movimentazione pezzi (1 | 0).

- **Mosso**: indica se il pezzo sulla casella è già stato mosso almeno una volta o no (1 | 0).

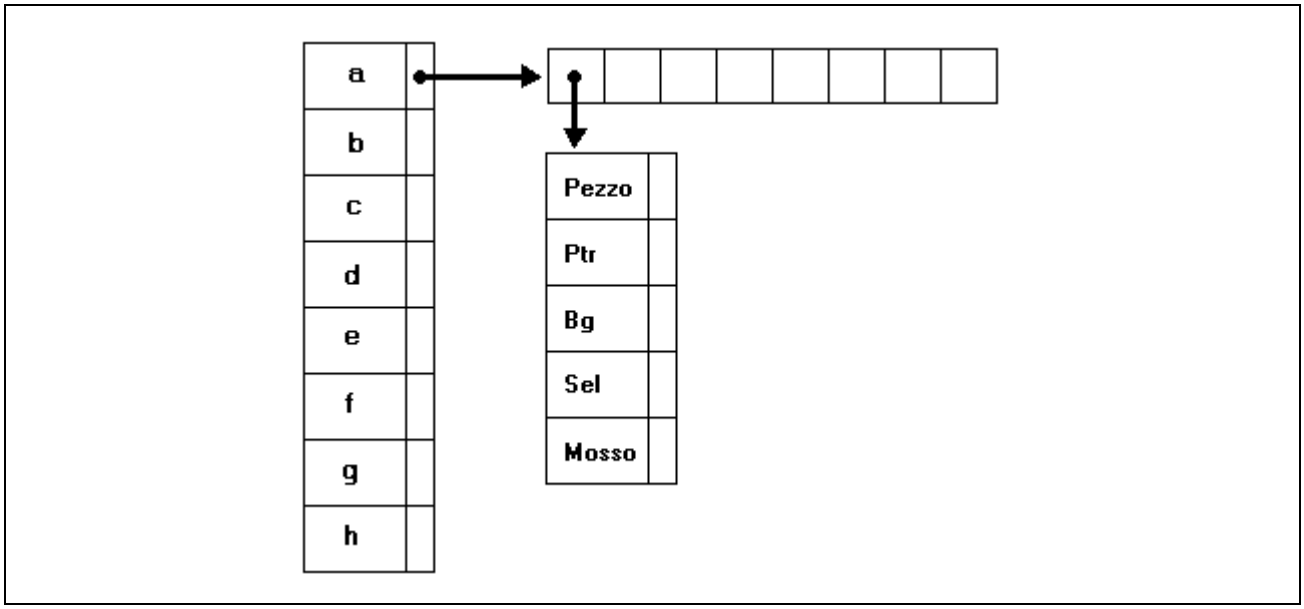


Figura 7.11: Struttura della tabella hash `Scacchiera`.

In questo modo reperire qualsiasi informazione riguardante una cella risulta immediato. Mettiamo, ad esempio, che si voglia conoscere il pezzo che risiede sulla casella di coordinate colonna-riga (x,y), in questo caso la formula da utilizzare è:

```
Scacchiera {'x'} → [y-1] → {'Pezzo'}
```

La seconda struttura dati che ricopre un ruolo fondamentale nel funzionamento di Chess Editor è lo `Stack`.
Come abbiamo già visto nella presentazione dell'interfaccia grafica, il suo ruolo è quello di memorizzare tutte le configurazioni della scacchiera man mano che esse evolvono nel corso della costruzione della partita.

Anche in questo caso viene utilizzata una tabella hash le cui chiavi sono stringhe del tipo *ns*, dove *n* è il numero del turno di gioco ed *s* il colore del giocatore che ha mosso (es.: "7B" identifica la mossa del giocatore nero al settimo turno di gioco). I valori associati alle chiavi sono puntatori a strutture dati di tipo `Scacchiera`. In questo modo è semplice reperire la configurazione del piano di gioco inerente a qualsiasi mossa sia stata effettuata durante la costruzione della partita.

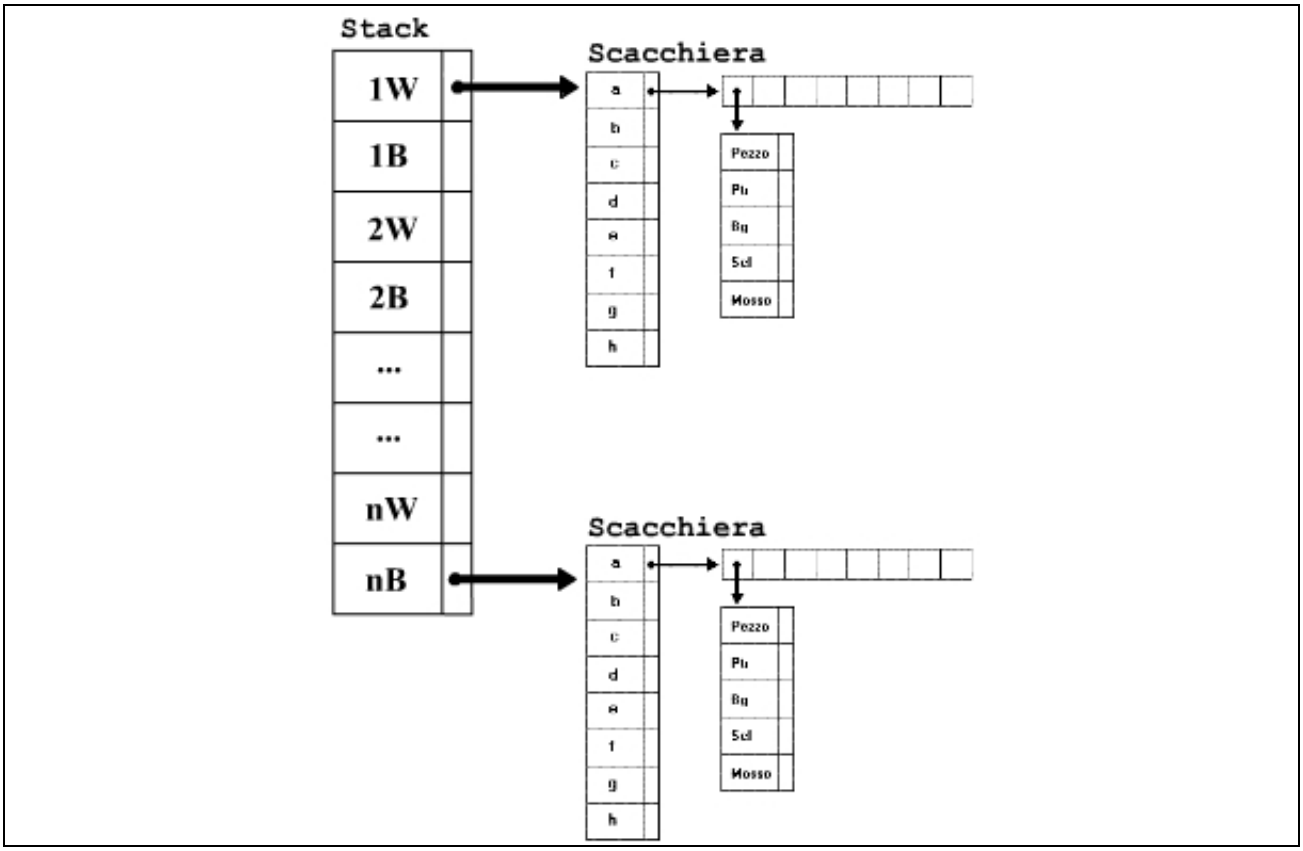


Figura 7.12: struttura di `Stack`.

7.3 Algoritmi

Una volta individuati i requisiti del progetto attraverso un'analisi delle caratteristiche e dei casi d'uso, un punto focale della progettazione è stata l'ideazione di un'interfaccia grafica semplice e completa che coprisse tutte le esigenze.
Come già anticipato sono state sfruttate intensivamente le funzionalità di Perl/Tk, che consentono di assegnare un gestore di eventi ad ognuno dei componenti che compaiono nella finestra dell'applicativo, in particolare è possibile creare una routine di gestione per ogni evento (eventi mouse e tastiera).

Chess Editor è stato sviluppato seguendo un approccio modulare, che ha consentito la suddivisione del processo produttivo globale in tante parti più piccole e quindi più facilmente gestibili. La modularità del software facilita anche le eventuali correzioni e le future espansioni.
Vediamo quanti e quali sono i moduli che compongono Chess Editor, evidenziando tra parentesi i nomi dei file fisici che li implementano:

- Modulo visualizzazione interfaccia grafica (*Chess_Editor.pl*).
- Modulo gestione movimentazione scacchi (*Chess.pm*).
- Modulo gestione eventi interfaccia grafica (*Eventi.pm*).
- Modulo gestione caricamento dati esterni (*Loader.pm*).
- Modulo gestione esportazioni PGN/PDF (*Salvataggi.pm*).

Nello sviluppo modulare è stata prestata particolare attenzione a ciò che in ingegneria del software viene definita **indipendenza funzionale**. Tutti i moduli sviluppati, infatti, svolgono mansioni specializzate ed indipendenti dagli altri moduli, interagendo tra loro solo attraverso un numero molto limitato di variabili d'ambiente (ogni azione compiuta dal modulo comincia e finisce all'interno del modulo stesso). Questo consente un buon livello di **information hiding**, volto a facilitare le operazioni di debugging ed evitare la propagazione degli effetti collaterali dovuti alle eventuali correzioni al codice.

7.3.1 Modulo gestione movimentazione scacchi

L'idea di funzionamento alla base di questo modulo ricalca, sotto alcuni aspetti, quella sviluppata per il parser PGN. La differenza tra i due approcci consiste nel fatto che in Chess Editor la mossa viene costruita da zero, decidendo a priori sia la partenza che la destinazione di un pezzo noto, mentre il parser si trova ad analizzare mosse delle quali conosce solo la destinazione e la tipologia di pezzo che muove.
L'interfaccia del modulo è costituita principalmente da due insiemi di funzioni:

- Funzioni per la gestione delle mosse di ogni singola tipologia di pezzo.

- Funzioni per la gestione dei conflitti che eventualmente si possono creare durante la movimentazione: due pezzi dello stesso tipo che possono muovere nella medesima casella. Non tutti i pezzi necessitano di questi controlli, ad esempio elementi singoli come re e regina o gli alfieri non presentano alcuna possibilità di conflitto.

Le regole di spostamento dei pezzi sono state formalizzate attraverso formule matematiche, le quali lavorano sulle coordinate colonna-riga di `Scacchiera`. Alla luce di questo vediamo di seguito le operazioni svolte da una funzione di movimentazione:

1. Utilizzando le regole di spostamento relative al pezzo selezionato, individua tutte le caselle sulla scacchiera candidate come possibili destinazioni della mossa.

2. Per ogni candidato trovato al punto 1 invoca la funzione *Candidato()*, che si occupa di controllare l'eventuale presenza di ostacoli sulla casella, considerando il caso particolare del pedone (mossa normale o di presa). Se la casella è libera od è occupata da pezzi dell'avversario allora è una candidata valida, altrimenti viene scartata.

3. Viene invocata la funzione di servizio *Accendi()*, la quale si occupa di evidenziare su `Scacchiera` tutte le caselle candidate allo spostamento, portando a 1 il loro flag *Sel* e cambiandone il colore di sfondo, in modo che tale marcatura sia evidente anche graficamente.

Analogamente vediamo come opera una funzione di controllo dei conflitti:

1. Data la tipologia del pezzo A in movimento trova tutti gli altri pezzi di analogo tipo sulla scacchiera.

2. Per ogni pezzo "concorrente" di A trovato al punto 1 ne invoca la relativa funzione di movimentazione che, come spiegato precedentemente, marca tutte le caselle in cui il concorrente può spostarsi.

3. Se la casella di destinazione di A risulta essere tra quelle marcate al punto 2 allora esiste un conflitto ed è quindi necessario creare le

informazioni di disambiguazione che verranno successivamente riportate nella notazione SAN.

7.3.2 Modulo gestione eventi interfaccia grafica

In questo modulo sono implementate tutte le routine di gestione degli eventi legati ai componenti dell'interfaccia grafica, le cui funzionalità sono già state ampiamente discusse. Vediamo l'aspetto della realizzazione che risulta essere più interessante da analizzare, cioè le modalità di interazione tra i vari componenti ed il modo in cui essi sfruttano le strutture dati. Particolare attenzione viene dedicata alla scacchiera virtuale ed alle liste delle mosse, poiché i pulsanti generalmente compiono azioni minimali, come ad esempio l'apertura di un'ulteriore finestra.

Azioni liste mosse - Questa routine accetta in input tre elementi: azione, che specifica se la lista è da manipolare in lettura o scrittura, coordinate di partenza e di arrivo della mossa effettuata. L'accesso in scrittura si ha quando una mossa è stata portata a termine sulla scacchiera virtuale e deve essere formalizzata. Questa opzione comporta i seguenti passaggi:

1. Riconoscimento del pezzo che ha mosso, attraverso le coordinate di arrivo applicate a `Scacchiera`.

2. Costruzione della notazione SAN che descrive la mossa, ottenuta utilizzando il simbolo del pezzo e le coordinate di partenza. Vengono sfruttate le funzioni di gestione dei conflitti del modulo *Chess.pm* per inserire eventuali informazioni di disambiguazione. Qui avvengono anche il riconoscimento e la gestione del caso particolare di promozione.

3. Lo stato attuale della scacchiera viene "congelato" ed inserito nello `Stack`, creando un nuovo indice.

4. La mossa così formalizzata viene scritta nell'apposita lista (B|W), preceduta dall'indice che la lega alla relativa configurazione nello `Stack` (indice = "turno di gioco" + "colore giocatore", ad esempio "10B").

L'accesso in lettura, invece, viene invocato ogni volta che si compie un doppio click del mouse su una delle mosse riportate nelle liste ed implica il richiamo della configurazione della scacchiera legata alla mossa selezionata. Questa opzione comporta la seguente sequenza di azioni:

1. Riconoscimento della voce selezionata dall'utente nelle liste delle mosse e conseguente estrapolazione del relativo indice di `Stack`.

2. La configurazione individuata nello `Stack` viene caricata in `Scacchiera`, procedendo all'aggiornamento opportuno della grafica sul piano di gioco virtuale.

3. L'indice individuato al punto 1 viene riportato nella variabile di ambiente `Change`. Questo consentirà al programma di capire che l'eventuale mossa successivamente effettuata comporterà una variazione alla sequenza di mosse già esistente (correzione).

Azioni scacchiera virtuale - Ogni casella della scacchiera è stata costruita per rispondere all'evento di pressione del tasto del mouse. Quando questa azione viene perpetrata dall'utente, l'elemento grafico richiama la relativa routine di gestione comunicando le proprie coordinate riga-colonna: attraverso queste ultime la routine è in grado di reperire tutte le informazioni inerenti alla casella che l'ha invocata. A questo punto si possono delineare quattro differenti scenari:

1. Non esistono precedenti selezioni sulla scacchiera e la casella che ha invocato la routine contiene un pezzo appartenente al giocatore detentore del turno di gioco. Utilizzando le funzioni di gestione della movimentazione, contenute nel modulo *Chess.pm*, vengono individuate le possibili caselle di destinazione del pezzo selezionato. La casella che ha invocato la routine viene considerata come punto di partenza della mossa.

2. Esiste una precedente selezione sulla scacchiera e la casella che ha invocato la routine contiene un pezzo appartenente al giocatore detentore del turno di gioco: si tratta di un cambio di selezione. La scelta precedentemente effettuata viene annullata e ci si riconduce al punto 1.

3. Esiste una precedente selezione sulla scacchiera, la casella che ha invocato la routine risulta essere tra quelle selezionate come destinazione della mossa e la variabile `Change` non riporta alcun indice: si tratta di effettuare una mossa semplice. Sulla scacchiera virtuale il pezzo viene graficamente spostato dalla locazione di partenza a quella di arrivo, ponendo attenzione ad identificare e gestire eventuali prese.
   A questo punto viene richiamata la routine di gestione delle liste delle mosse in modalità "scrittura", passando tutti i dati inerenti alla mossa necessari per la formalizzazione SAN.

4. Esiste una precedente selezione sulla scacchiera, la casella che ha invocato la routine risulta essere tra quelle selezionate come destinazione della mossa e la variabile `Change` riporta un indice: si tratta di effettuare una mossa correttiva nella sequenza già esistente. L'indice riportato da `Change`, come abbiamo già visto, identifica una voce nella lista delle mosse, in particolare indica la mossa dopo la quale si vuole effettuare la correzione. La routine elimina tutte le voci successive a quella selezionata in entrambe le liste, preoccupandosi di aggiornare debitamente anche lo `Stack`. Eliminando qualsiasi indicazione riportata da `Change`, quindi, siamo ora in grado di ricondurci al punto 3.

7.3.3 Modulo caricamento dati esterni

La funzione di questo modulo è indubbiamente simile a quella del parser PGN incontrato precedentemente, anche se il suo operato risulta più snello e veloce del suo predecessore, in quanto può usufruire di costrutti più complessi e potenti.
Il suo scopo principale è quello di acquisire una partita di scacchi da un file PGN esterno, interpretarla ed integrarla nell'interfaccia grafica in modo che i dati possano essere manipolati dall'utente attraverso le funzionalità di Chess Editor.
L'idea base è quella di simulare, per ogni mossa trovata nel file, gli eventi che si verificherebbero normalmente se fosse proprio l'utente a creare la mossa, interagendo con la scacchiera virtuale.
L'interfaccia del modulo è costituita dall'unica funzione *Loader()*, che esegue le seguenti operazioni:

1. Invoca la funzione di servizio *Init()*, il cui compito è di resettare i costrutti inerenti alla scacchiera, sia per quanto riguarda le strutture dati che la grafica, acquisire la partita dal file PGN e riportare i dati descrittivi nelle apposite caselle di testo dell'interfaccia. Al momento dell'acquisizione vengono eliminati anche eventuali commenti presenti nella sequenza delle mosse (testi racchiusi tra parentesi graffe).

2. Per ogni mossa estrapolata dal file PGN ne esegue il parsing bottom-up, invocando la funzione *Token()*. Quest'ultima trasforma la stringa SAN di n caratteri in un array di n elementi, eliminando i caratteri "+" e "#" e separando gli ultimi due caratteri che identificano la `Destinazione` della mossa. Le mosse di arrocco e promozione sono trattate separatamente dalle altre.
3. Per ogni simbolo speciale "K", "Q", "B", "P", "N", "R" richiama la funzione di servizio *Move()* passandogli come argomento proprio il simbolo: questa funzione si occupa della vera e propria movimentazione dei pezzi. Le informazioni di disambiguazione eventualmente presenti nel token vengono riconosciute e salvate in `Partenza`.

A questo punto merita particolare attenzione la citata funzione *Move()*, che risulta essere il vero e proprio fulcro del modulo. Le operazione che essa esegue sono di seguito riportate:

1. Data la tipologia di pezzo da muovere ricerca su `Scacchiera` tutti i pezzi dello stesso tipo, ricavandone una lista di coordinate riga-colonna (caselle sulle quali i pezzi risiedono).

2. Per ognuno dei pezzi individuati al punto 1 viene simulato un evento di click del mouse sulla casella che lo contiene nella scacchiera virtuale. L'effetto ottenuto è che tutte le caselle che possono costituire una possibile destinazione della mossa vengono marcate (flag *Sel* = 1): se la casella `Destinazione` risulta essere tra quelle marcate significa che il pezzo può effettivamente esservi spostato sopra. In caso di conflitti troveremmo le relative informazioni di disambiguazione in `Partenza`, attraverso la quale sarebbe possibile scremare tutti i candidati indesiderati e ricondursi ad una scelta univoca.

3. Una volta individuato il pezzo da spostare, viene simulato un secondo evento di click sulla scacchiera virtuale, questa volta sulla casella individuata da `Destinazione`. Questa operazione innesca il procedimento di spostamento del pezzo sulla scacchiera stessa ed il relativo inserimento della mossa nelle liste in formato SAN.

Le mosse di arrocco e promozione sono gestite sempre attraverso il meccanismo di simulazione: per l'arrocco viene semplicemente invocata la routine di gestione dell'omonimo pulsante nell'interfaccia (previo il settaggio dei parametri che identificano il giocatore ed il tipo di arrocco eseguito), mentre il meccanismo di promozione è innescato automaticamente dal movimento opportuno del pedone (una corretta impostazione delle variabili d'ambiente consente di bypassare la visualizzazione della finestra di popup che gestisce la promozione).

7.3.4 Modulo gestione esportazioni PGN/PDF

Per quanto riguarda il salvataggio dei dati in formato PGN viene utilizzato un modello (*Modello.pgn*), la cui struttura viene riportata in figura 7.13.

```
[Event "##evento##"]
[Site "##site##"]
[Date "##date##"]
[Round "0"]
[White "##white##"]
[Black "##black##"]
[Result "##result##"]
[ICCResult ""]
[WhiteElo "0"]
[BlackElo "0"]
[ECO "C47"]
[NIC ""]
[Time "00:00:00"]
[TimeControl ""]
##mosse##
```

Figura 7.13: struttura di Modello.pgn.

Al momento dell'operazione, Chess Editor sostituisce alle ancore presenti nel modello tutti i dati riportati nell'interfaccia grafica.
L'elenco delle mosse viene costruito consultando le relative liste dei giocatori, sfruttando gli indicativi dei turni di gioco riportati in ogni mossa (per il turno n vengono prese le n-esime mosse di bianco e nero).

Per il salvataggio in formato PDF, invece, il procedimento risulta più complicato. Allo scopo è stata sfruttata la libreria Perl *PDF::Reuse* [PDF2], che fornisce tutte le funzionalità necessarie a creare e gestire ogni componente del documento.
La caratteristica più interessante di questa libreria è costituita da alcune particolari funzioni, attraverso le quali si possono integrare programmi JavaScript nel documento PDF generato. Sfruttando questa tecnica di embedded programming è possibile creare documenti interattivi in grado di manipolare i loro stessi contenuti, da qui la definizione automodificanti.
Il risultato raggiunto è stato di generare un documento PDF di una sola pagina, dotato di comandi interattivi (pulsanti) per scorrere in avanti ed indietro un insieme di diagrammi scacchistici rappresentanti una partita di n mosse.

Figura 7.14: frammento del modello di form PDF Modello.pdf

Per raggiungere questo obiettivo è stato inizialmente creato un modello di form PDF (*Modello.pdf*) con l'ausilio di Adobe Acrobat 6.0. Questo modello riporta l'intestazione generica della partita, contenente i campi

testo per l'inserimento dei nomi dei giocatori, il luogo e così via, ed un campo testo a più linee dove verranno successivamente riportate le configurazioni della scacchiera, come è visibile in figura 7.14.
Quest'ultimo campo è stato creato in modo da visualizzare al suo interno testi scritti col font "Chess Alpha", una fonte digitale scacchistica di tipo TrueType già vista in precedenza.
Ulteriormente, sono stati aggiunti quattro pulsanti per lo spostamento avanti ed indietro nelle visualizzazioni dei diagrammi scacchistici successivamente prodotti e due campi nascosti, `indice` e `MAX`, il cui compito è quello di memorizzare valori globali utilizzati dalle funzioni JavaScript di seguito descritte:

- *Slide (target)*: gestisce le azioni dei pulsanti, le quali sono definite dal parametro *target*. Ad esempio "*target* = *avanti*" indica che deve essere visualizzato il diagramma scacchistico successivo a quello attualmente in visione. La pressione di ogni pulsante comporta una corrispondente variazione del valore del campo nascosto `indice`.

- *Compile (nome_nero, nome_bianco, risultato, evento, luogo, data, MAX)*: funzione che compila l'intestazione del form con i rispettivi dati descrittivi della partita ed i campi nascosti `indice` (inizializzato a 0) e `MAX`. Il primo serve, come suggerisce il nome, ad indirizzare le configurazioni nell'array `Scacchiera[]` ($0 \leq$ `indice` $\leq$ `MAX`), mentre il secondo identifica il numero di mosse della partita descritta dal documento.

Le configurazioni della scacchiera sono rappresentate attraverso sequenze di sessantaquattro caratteri: ogni carattere identifica una casella della scacchiera ed ovviamente è la rappresentazione "Chess Alpha" del contenuto della casella stessa. Tutte le configurazioni vengono memorizzate in una variabile di tipo array denominata `Scacchiera[]`, quindi una partita di n mosse corrisponde ad un array di n elementi, ognuno composto da stringhe di sessantaquattro caratteri.
La programmazione Perl inerente alla generazione del documento finale consiste principalmente nell'inizializzare correttamente tutte le variabili e le strutture dati che vengono integrate nel documento stesso.

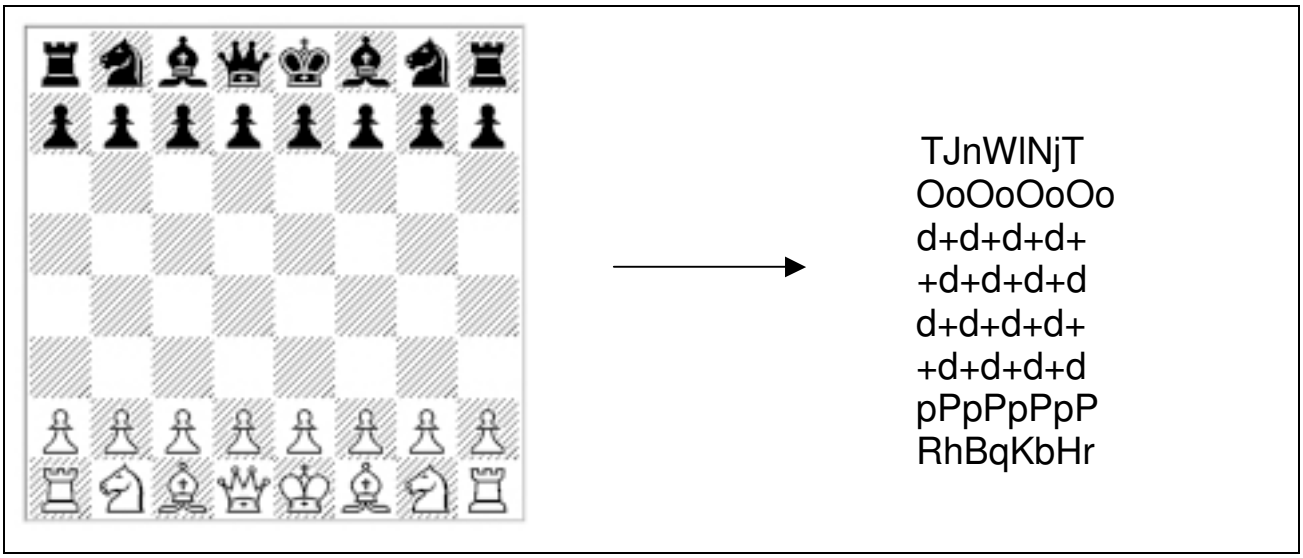


Figura 7.15: corrispondenza tra il diagramma renderizzato coi caratteri "Chess Alpha" (sinistra) e la relativa stringa di sessantaquattro caratteri (destra).

Le operazioni effettivamente compiute dal modulo di salvataggio vengono di seguito illustrate, in ordine cronologico:

1. Viene scandito lo `Stack` di sistema e, ad una ad una, vengono analizzate tutte le configurazioni della scacchiera in esso contenute.

2. Ogni configurazione trovata al punto 1 deve essere tradotta in un diagramma da inserire nel documento. Per fare ciò è necessario analizzare ogni casella e tradurne il contenuto nell'opportuno carattere ASCII che ne consenta una corretta rappresentazione con "Chess Alpha": in questo modo si genera una stringa di sessantaquattro caratteri per ogni configurazione. Tutte le stringhe man mano costruite vengono utilizzate per generare il listato JavaScript inerente alla struttura dati `Scacchiera[]`.

3. Viene acquisito il listato JavaScript dal file *JavaScript.js*, che viene completato inserendo l'array `Scacchiera[]` all'interno della funzione *Slide()*.

4. Viene preparato il listato della funzione di inizializzazione JavaScript, che riporta tutte le operazioni da effettuare al momento del caricamento del documento PDF in Acrobat Reader:
   - viene richiamata la funzione *Compile()*, alla quale vengono passati tutti i dati descrittivi della partita ed il numero di mosse di cui la partita stessa si compone.

- Viene richiamata la funzione *Slide()* con parametro “indietro” per visualizzare la configurazione di apertura della scacchiera.

A questo punto il funzionamento del meccanismo interattivo integrato nel documento PDF risulta facilmente comprensibile: ogni volta che un pulsante di spostamento viene premuto si verifica una variazione del valore di `indice`, che implica il prelevamento della stringa individuata da `Scacchiera[indice]`. Questa stringa viene immessa nel campo del form preposto alla visualizzazione dei diagrammi, che la renderizza utilizzando il font “Chess Alpha”.
In figura 7.16 è possibile vedere come si presenta un documento PDF generato da Chess Editor.

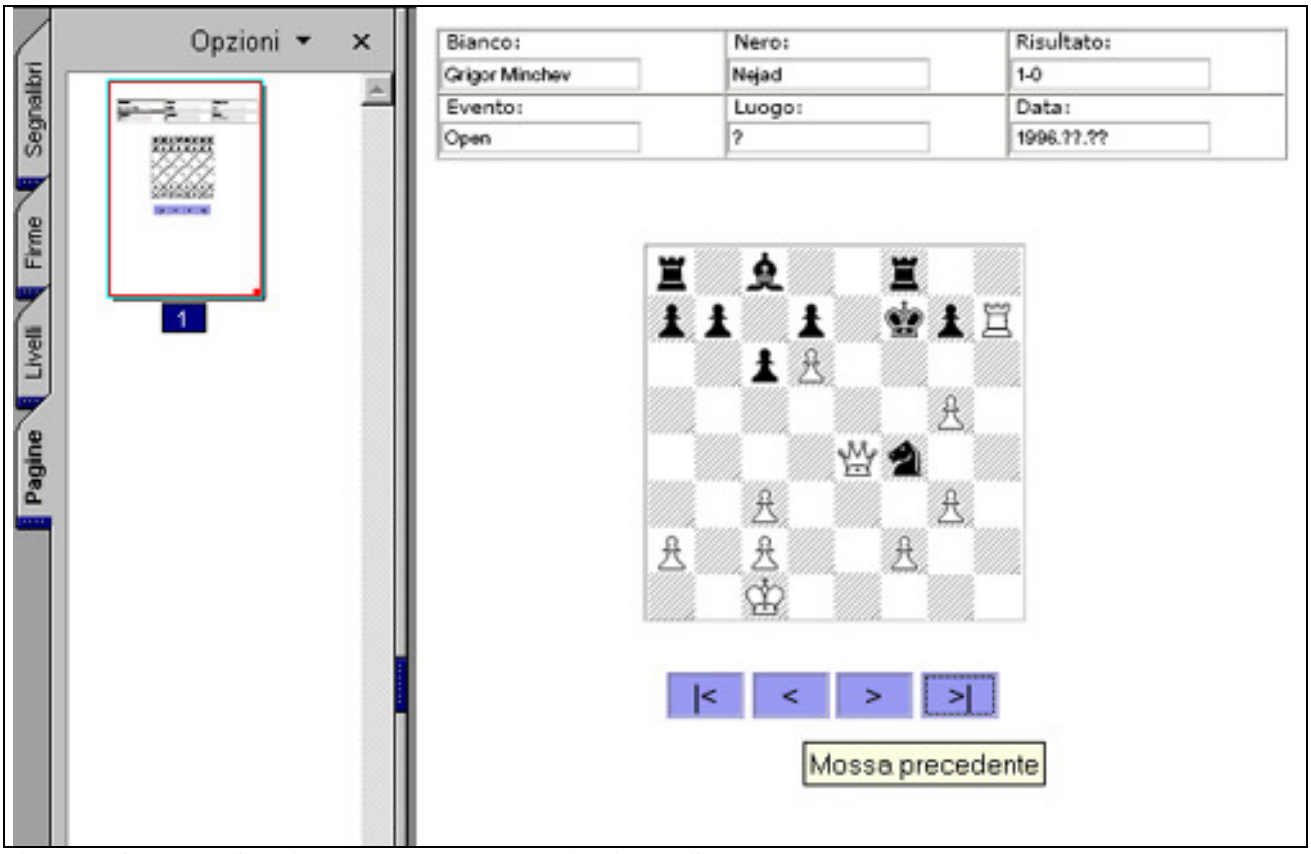


Figura 7.16: aspetto di un documento PDF generato da Chess Editor.

Va fatta una precisazione legata al corretto funzionamento della soluzione presentata: date le funzionalità avanzate di embedded programming utilizzate nell’implementazione è necessario servirsi Acrobat Reader 6.0 (o successivi) per una corretta visualizzazione dei documenti generati.

7.4 Conclusioni

La soluzione proposta ricopre tutte le caratteristiche che un sistema ideale di generazione di documenti digitali interattivi dovrebbe presentare:

Editing: il software fornisce un'interfaccia apposita per la creazione di partite di scacchi, con relativa memorizzazione delle mosse in formato SAN per uso successivo.

Delivering: i formati digitali in cui i dati delle partite possono essere esportati sono altamente diffusi e portabili. PGN offre un valido supporto per l'interscambio veloce di dati, mentre PDF garantisce un'ottima qualità di visualizzazione su qualsiasi piattaforma.

Dinamicità: i documenti PDF prodotti sono automodificanti, sono cioè in grado di modificare i propri contenuti su richiesta dell'utente. Agli occhi di un lettore il documento appare di una sola pagina ed i dati vengono aggiornati all'interno della stessa, senza l'ausilio di fonti esterne.

Interattività: nei documenti PDF prodotti vengono forniti all'utente quattro pulsanti per la navigazione dei contenuti. Attraverso di essi è possibile osservare l'evoluzione dei diagrammi scacchistici procedendo passo per passo (sia in avanti che indietro), o saltando direttamente agli estremi della slide-show (primo o ultimo diagramma della sequenza).

Se consideriamo gli esempi di soluzioni esistenti, analizzati nel capitolo 3, come un riassunto di ciò che il mercato offre, vediamo che la soddisfazione dei requisiti è sempre parziale. Questo è legato alla ristrettezza del campo d'uso a cui ogni software degli esempi è soggetto: Chess Editor è stato sviluppato appositamente per dimostrare che è possibile integrare, in un'unica soluzione, tutti i concetti che caratterizzano un sistema di generazione dinamica di documenti e per questo risulta completo sotto ogni aspetto.

Tutto il codice sviluppato per l'implementazione di Chess Editor è visionabile all'appendice D.

8 CONCLUSIONI

Il panorama della documentazione digitale si presenta molto ampio e variegato, con una consistente gamma di formati frutto delle evoluzioni tecnologiche iniziate nei primi anni sessanta e progredite fino ad oggi.
Ogni tipologia di documento elettronico nasce con un preciso scopo: divulgazione e catalogazione di informazioni in modo indipendente dalla piattaforma (ad esempio PDF e LIT), visualizzazione di ipertesti sulla rete (HTML), visualizzazione di contenuti a carattere grafico (formati immagine o video) e così via.
Quando si progetta un sistema di creazione dinamica di documenti bisogna innanzitutto individuare le caratteristiche che la documentazione prodotta dovrà presentare, scegliendo poi il formato, od i formati, le cui peculiarità meglio si addicono alla soddisfazione dei requisiti.
Nel nostro caso si è scelto di produrre documenti a carattere scacchistico, dotati di sistemi di interazione che consentono all'utente di esplorare il contenuto informativo (evoluzione di diagrammi scacchistici).
Analizzando il problema così formulato è stato possibile scomporlo in diversi sottoproblemi, più semplici da risolvere:

- Formalizzazione del gioco degli scacchi: è necessario avere un formalismo che descriva le partite di scacchi in modo completo e non ambiguo. Allo scopo è stata utilizzata la notazione PGN (Portable Game Notation).

- Interattività: è necessario stabilire come l'utente deve interagire con il documento finale, per integrare in quest'ultimo meccanismi appropriati. Spesso limitazioni in questo senso derivano proprio dalle caratteristiche tecniche dei formati digitali o dei software che li visualizzano.

- Dinamicità: il documento prodotto può essere statico o dinamico. Nel primo caso tutto il contenuto informativo viene inserito in fase di creazione del documento, mentre nel secondo caso il contenuto informativo viene generato su richieste dell'utente che partono dal documento stesso.

- Delivering: il documento viene prodotto a scopo di divulgazione di informazioni al pubblico. In questo caso è necessario che il formato prescelto presenti ottime caratteristiche di leggibilità, pur mantenendo dimensioni limitate in termini di memoria.

- Editing: devono essere fornite funzionalità per modificare il contenuto dei documenti prodotti.

Risulta difficile far convivere tutte queste caratteristiche in un'unica soluzione ed è per questo che si è scelto di procedere a piccoli passi, sviluppando prima soluzioni che soddisfacessero sottoinsiemi delle caratteristiche sopra elencate, per poi arrivare ad una soluzione completa sotto ogni aspetto.

8.1 Risultati e sviluppi futuri

Per risolvere il primo sottoproblema individuato è stato implementato un analizzatore sintattico per PGN (capitolo 4), utilizzato come libreria dai software successivamente realizzati. Le sue funzioni sono legate all'interpretazione dei dati contenuti in un file PGN passatogli come input:

- Estrapolazione dei dati descrittivi di una partita.

- Costruzione di strutture dati in grado di riportare l'evoluzione dello stato della scacchiera dopo un numero *n* qualsiasi di mosse effettuate.

Il risultato ottenuto è funzionale, anche se presenta alcune limitazioni come, ad esempio, l'incapacità di gestire un eventuale tag FEN o il costrutto delle mosse alternative. Sviluppi futuri potrebbero essere proprio legati all'eliminazione di queste lacune.

La prima soluzione presentata in risposta al problema della generazione dinamica di documenti interattivi (capitolo 5) è legata al formato digitale LIT di Microsoft Reader. Tramite l'utilizzo di *PGN_Parser.pm* si è potuto operare un procedimento di trasformazione che ha portato dalla notazione PGN ad una sequenza di documenti HTML, ognuno contenente un diagramma scacchistico (un documento per ogni mossa) ed un menù che permette lo spostamento da una pagina all'altra. I riferimenti a questi

documenti sono stati tutti riportati in un package file scritto in notazione XML, il quale è stato successivamente passato in input a ReaderWorks, software della Overdrive. Le pagine HTML precedentemente generate sono state così integrate in un unico file di tipo LIT.
Il risultato è quindi un documento statico dotato di collegamenti che permettono l'esplorazione del contenuto informativo in modo rapido e mirato, senza alcuna possibilità di editing se non ripetendo tutto il processo di generazione.
Sviluppi futuri inerenti a questa soluzione sono legati a future evoluzioni di Microsoft Reader e del relativo formato LIT che, ora come ora, non consentono più di tanto di spaziare con la fantasia.

La seconda soluzione sviluppata (capitolo 6) ha lo scopo di analizzare l'aspetto della dinamicità all'interno dei documenti prodotti e per farlo utilizza uno dei word processor più diffusi sul mercato: Microsoft Word.
L'idea è di creare, sempre con l'ausilio del parser PGN, una sequenza di gestori di eventi (uno per ogni mossa) in grado di visualizzare diagrammi scacchistici sfruttando una normale sessione di lavoro in Word.
In particolare, ogni gestore è un programma Perl indipendente in grado di sviluppare la partita fino ad una certa mossa, disegnarne il relativo diagramma e visualizzarlo con Word in formato HTML.
Il risultato è un documento di una sola pagina il cui contenuto informativo viene aggiornato dinamicamente su richiesta dell'utente, avvalendosi del sistema di trasformazione esterno costituito dai gestori di eventi.
Un'interessante sviluppo futuro potrebbe consistere in un analogo sistema, costituito però da un unico gestore in grado di visualizzare da solo un'intera partita. Questo consentirebbe di migliorare notevolmente l'aspetto di delivering del software.

La terza ed ultima realizzazione racchiude in sé tutte le caratteristiche che un sistema ideale di generazione dinamica di documenti interattivi dovrebbe presentare: dinamicità, interattività, funzionalità di editing e idoneità al delivering.
Si tratta di un editor scacchistico attraverso il quale è possibile creare partite od acquisirle da fonti PGN esterne. Tutte le mosse effettuate sulla scacchiera virtuale, inclusa nell'interfaccia, vengono automaticamente formalizzate secondo la notazione SAN. La sequenza delle mosse è modificabile in qualsiasi momento e la partita creata è salvabile in formato PGN e/o PDF. La particolarità dei documenti PDF generati è che sono

automodificanti, sono cioè dotati di meccanismi attraverso i quali possono modificare i loro stessi contenuti: in un'unica pagina è quindi possibile visualizzare tutti i diagrammi scacchistici che rappresentano una partita di *n* mosse creata con l'editor, senza l'ausilio di fonti esterne al documento.
Sviluppi futuri di questa soluzione potrebbero essere la gestione del tag FEN e dei costrutti per le mosse alternative, che attualmente non sono supportati: in questo modo sarebbe possibile creare anche quiz scacchistici.
Ulteriormente, la possibilità di gestire costrutti come i layers, vera novità di PDF 1.5, con JavaScript potrebbe aprire la via verso nuovi utilizzi dei documenti PDF, come ad esempio presentazioni animate ed interattive.

A - Parser PGN

Ambiente.pm

```
#!/usr/bin/perl

@ENV_destinazione=();               # Destinazione del pezzo in esame: x,y
@ENV_partenza=();                   # Informazioni aggiuntive sulla
                                    # locazione di partenza del pezzo
$ENV_stato_mossa="normale";         # normale|scacco|scacco matto
$ENV_mangiata=0;                    # 0 la mossa corrente non implica una
                                    # mangiata, 1 altrimenti
$ENV_player="W";                    # Giocatore corrente
@ENV_token=();                      # Token in analisi
%ENV_mappa_x=('a',1,'b',2,'c',3,'d',4,
              'e',5,'f',6,'g',7,'h',8);# Traduce le lettere in numeri
```

PGN_parser.pm

```
#!/usr/bin/perl

use Chess::PGN::Parse;
require ("Ambiente.pm");

#########################################################################
# Prende in input un file PGN e restituisce un array contenente tutti i #
# dati della partita                                                    #
#########################################################################

sub PGN_Parse {
local($pgn_file,$pgn,@game);

 $pgn_file=@_[0];

 $pgn = new Chess::PGN::Parse $pgn_file or die "can't open $pgn_file\n";

 while ($pgn->read_game()) {
  push(@game,"Bianco",$pgn->white,"Nero",$pgn->black,"Data",$pgn->date,
       "Evento",$pgn->event,"Round",$pgn->round,"Luogo",$pgn->site,
       "Risultato",$pgn->result,"Mosse",$pgn->smart_parse_game());
 }

 return(@game);
}


#########################################################################
# Data una mossa di n caratteri la trasforma in un array di n celle.    #
# Il token ottenuto viene invertito in modo da permetterne il parsing   #
# bottom-up. In questa fase vengono trattati i caratteri speciali "+"   #
# e "#" e viene definita la destinazione della mossa.                   #
#########################################################################

sub Token {
local($mossa,$i,@token,$length,$x,$y);

 $mossa=@_[0];

 push(@token,"P");                      # Preparazione a eventuali mosse pedone
```

```
 for($i=0;$i<length($mossa);$i++){
  push(@token,substr $mossa,$i,1);
 }
 $length=scalar @token -1;

 if(@token[$length] eq "#"){$ENV_stato_mossa="scacco matto";pop @token;}
 if(@token[$length] eq "+"){$ENV_stato_mossa="scacco";pop @token;}
 if((@token[$length] ne "+")&&(@token[$length] ne "#"))
  {$ENV_stato_mossa="normale";}
 push (@ENV_destinazione,pop @token,pop @token);     # Si identifica la
                                                     # destinazione del pezzo
 @ENV_destinazione=reverse @ENV_destinazione;

 return(@token);
}


#########################################################################
# Funzione di aggiornamento variabili ambiente al termine dell'analisi #
# di un token                                                           #
#########################################################################

sub Resume {

 @ENV_destinazione=();
 @ENV_partenza=();
 @ENV_token=();
 $ENV_mangiata=0;

 if($ENV_player eq "W"){$ENV_player="B";}
 else{$ENV_player="W";}
}


#####################################################
# Funzione che gestisce la movimentazione dei pezzi #
#####################################################

sub Move {
local($candidati,@candidati,$non_candidati,@non_candidati,
      $pezzo,$item,@new,$candidato);

 $pezzo=@_[0];                                  # Tipologia pezzo da spostare
 $candidati=@_[1];
 $non_candidati=@_[2];

 @candidati=@$candidati;                        # Lista pezzi candidati allo
                                                # spostamento
 @non_candidati=@$non_candidati;                # Lista pezzi da non muovere

 if(scalar @candidati==1){                      # Il candidato è univoco
  $ENV_scacchiera{substr @candidati[0],0,1}->
   [(substr @candidati[0],1,1)-1]="**";         # Aggiorna scacchiera
  $ENV_scacchiera{@ENV_destinazione[0]}->
  [(@ENV_destinazione[1])-1]=$pezzo;            #
  push(@non_candidati,
  @ENV_destinazione[0].@ENV_destinazione[1]);   # Aggiorna scacchi
  $ENV_scacchi{$pezzo}=\@non_candidati;
 }

 else{                                  # I candidati sono molteplici
  while(@ENV_partenza){                 # Analizza informazioni aggiuntive
   $item=pop(@ENV_partenza);
```

```
   if($item=~ m/[a-h]/){                  # L'informazione è una colonna
    while(@candidati){
     $candidato=pop(@candidati);
     if($candidato=~ m/$item[1-8]/){      # Il candidato è idoneo
      push(@new,$candidato);
     }
     else{                                # Il candidato non è idoneo
      push(@non_candidati,$candidato);
     }
    }
   }

   else{                                  # L'informazione è una riga
    while(@candidati){
     $candidato=pop(@candidati);
     if($candidato=~ m/[a-h]$item/){      # Il candidato è idoneo
      push(@new,$candidato);
     }
     else{                                # Il candidato non è idoneo
      push(@non_candidati,$candidato);
     }
    }
   }

   &Move($pezzo,\@new,\@non_candidati);
  }
 }

}


#############################################################
# Funzione che stabilisce se un numero è pari o dispari #
#############################################################

sub Is_pari{

 if(@_[0]==1 || @_[0]==3 || @_[0]==5 || @_[0]==7){
  return(0);
 }
 else{
  return(1);
 }

}
#####################################################################
# Funzioni di gestione caratteri speciali nella fase di parsing  #
# (O,x,P,N,B,R,Q,K)                                                 #
#####################################################################

### Gestione arrocco ###
sub O {
local($re_start_colonna,$re_start_riga,$torre_start_colonna,
      $torre_start_riga,$re_end_colonna,$re_end_riga,$torre_end_colonna,
      $torre_end_riga,$torre,@torre,$tipo,@new);

 $tipo=@_[0];                                    # Tipo di arrocco
 $tipo=~ s/\+//;$tipo=~ s/\#//;
 $torre=$ENV_scacchi{$ENV_player."R"};
 @torre=@$torre;                                 # Identifica posizione torri

 if($ENV_player eq "W"){                         # Arrocco bianco
  $re_start_colonna="e";$re_start_riga="1";
  if($tipo eq "O-O"){                            # Arrocco corto
```

```
   $re_end_colonna="g";$re_end_riga="1";
   $torre_start_colonna="h";$torre_start_riga="1";
   $torre_end_colonna="f";$torre_end_riga="1";
  }
  else{                                        # Arrocco lungo
   $re_end_colonna="c";$re_end_riga="1";
   $torre_start_colonna="a";$torre_start_riga="1";
   $torre_end_colonna="d";$torre_end_riga="1";
  }
 }
 else{                                         # Arrocco nero
  $re_start_colonna="e";$re_start_riga="8";
  if($tipo eq "O-O"){                          # Arrocco corto
   $re_end_colonna="g";$re_end_riga="8";
   $torre_start_colonna="h";$torre_start_riga="8";
   $torre_end_colonna="f";$torre_end_riga="8";
  }
  else{                                        # Arrocco lungo
   $re_end_colonna="c";$re_end_riga="8";
   $torre_start_colonna="a";$torre_start_riga="8";
   $torre_end_colonna="d";$torre_end_riga="8";
  }
 }
 @ENV_destinazione=($re_end_colonna,$re_end_riga);              # Aggiorna re
 &Move($ENV_player."K",[$re_start_colonna.$re_start_riga],[]);  #
 @ENV_destinazione=($torre_end_colonna,$torre_end_riga);        #Aggiorna torri
 @ENV_partenza=($torre_start_colonna,$torre_start_riga);        #
 &Move($ENV_player."R",$ENV_scacchi{$ENV_player."R"},[]);        #

 &Resume();
}


### Gestione promozione ###
sub Promote {
local($mossa,@item,$pezzo,$new,$i);

 $mossa=@_[0];
 $mossa=~ s/\+//;$mossa=~ s/\#//;
 @item=split("=",$mossa);
 for($i=0;$i<length(@item[0]);$i++){     # Individua colonna-riga destinazione
  push(@ENV_destinazione,substr @item[0],$i,1);}
  $pezzo=$ENV_player.@item[1];           # Pezzo da sostituire al pedone
  $pedone=$ENV_player."P";

  &P();
  for($i=0;$i<length(@item[0]);$i++){    # Individua colonna-riga destinazione
   push(@ENV_destinazione,substr @item[0],$i,1);}
  &x();
  $ENV_scacchiera{@ENV_destinazione[0]}->
  [(@ENV_destinazione[1])-1]=$pezzo;     # Aggiorna la scacchiera
  $new=$ENV_scacchi{$pezzo};
  push(@$new,@item[0]);                  # Aggiunge nuova posizione alla lista
                                         # del pezzo

  &Resume();
}


### Gestione mangiata ###
sub x {
local($pezzo,$posizioni,@posizioni,$destinazione,@new);

 $ENV_mangiata=1;
```

```
 $pezzo=$ENV_scacchiera{@ENV_destinazione[0]}->
           [(@ENV_destinazione[1])-1];        # Identifico il pezzo mangiato
 $ENV_scacchiera{@ENV_destinazione[0]}->
    [(@ENV_destinazione[1])-1]="**";
 $posizioni=$ENV_scacchi{$pezzo};
 @posizioni=@$posizioni;
 $destinazione=@ENV_destinazione[0].@ENV_destinazione[1];
 while(@posizioni){                                          # Il pezzo mangiato
  $posizioni=pop(@posizioni);                                # viene cercato ed
  if($destinazione ne $posizioni){push(@new,$posizioni);}    # eliminato
 }                                                           #
 $ENV_scacchi{$pezzo}=\@new;                                 #

}


### Gestione pedone ###
sub P {
local(@posizioni,$posizioni,$pezzo,@candidati,@non_candidati,$candidato,
      $x_start,$y_start,$x_end,$y_end,$x,$y,$ostacolo,$i,$start,$end);

 $pezzo=$ENV_player."P";                          # Individua WP o BP
 $posizioni=$ENV_scacchi{$pezzo};                 # Posizioni di tutti i pedoni
 @posizioni=@$posizioni;                          #
 $x_end=$ENV_mappa_x{@ENV_destinazione[0]};       # x di arrivo
 $y_end=@ENV_destinazione[1];                     # y di arrivo

 if($ENV_mangiata==1){                            # Il pedone mangia
  while(@posizioni){                              # Scandisco tutte le posizioni
                                                  # dei pedoni alla ricerca di
                                                  # quella giusta
   $candidato=pop(@posizioni);
   $x_start=$ENV_mappa_x{substr $candidato,0,1};# x di partenza
   $y_start=substr $candidato,1,1;                # y di partenza
   $x=abs($x_start-$x_end);
   $y=abs($y_start-$y_end);
   if($x==1 && $y==1){       # Il candidato in analisi può essere quello giusto
    push(@candidati,$candidato);
   }
   else{                     # Il candidato in analisi non è quello giusto
    push(@non_candidati,$candidato);
   }
  }
 }

 else{                                            # Il pedone muove normalmente

  while(@posizioni){                              # Scandisco tutte le posizioni
                                                  # dei pedoni alla ricerca di
                                                  # quella giusta
   $candidato=pop(@posizioni);
   $x_start=$ENV_mappa_x{substr $candidato,0,1};# x di partenza
   $y_start=substr $candidato,1,1;                # y di partenza

   if($x_start-$x_end==0){    # Il candidato in analisi può essere quello giusto

    if($y_start<=$y_end && $ENV_player eq "W"){      # Decide la direzione della
                                                    # scansione ostacoli
                                                    # considerando che il
                                                    # pedone può muovere solo
                                                    # avanti
     $start=$y_start+1;$end=$y_end;
    }
    if($y_start>=$y_end && $ENV_player eq "B"){$start=$y_end;$end=$y_start-1;}
```

```
    for($i=$start;$i<=$end;$i++){
     if($ENV_scacchiera{substr $candidato,0,1}->[$i-1] ne "**"){
      $ostacolo=1;}                # Rileva ostacoli sul percorso del pedone
    }
    if($start==0 && $end==0){$ostacolo=1;}     # Elimina i pedoni che non
                                               # devono muovere all'indietro
    if($ostacolo==0){push(@candidati,$candidato);}
    else{push(@non_candidati,$candidato);}
    $ostacolo=0;
   }

   else{                           #Il candidato in analisi non è quello giusto
    push(@non_candidati,$candidato);
   }
  }
 }

 &Move($pezzo,\@candidati,\@non_candidati);
 &Resume();
}


### Gestione cavallo ###
sub N {
local(@posizioni,$posizioni,$pezzo,@candidati,@non_candidati,$candidato,
      $x_start,$y_start,$x_end,$y_end,$x,$y);

 $pezzo=$ENV_player."N";                       # Individua WN o BN
 $posizioni=$ENV_scacchi{$pezzo};              # Posizioni di tutti i cavalli
 @posizioni=@$posizioni;                       #
 $x_end=$ENV_mappa_x{@ENV_destinazione[0]};    # x di arrivo
 $y_end=@ENV_destinazione[1];                  # y di arrivo

 while(@posizioni){                            # Scandisco tutte le posizioni
                                               # dei cavalli alla ricerca di
                                               # quella giusta
  $candidato=pop(@posizioni);
  $x_start=$ENV_mappa_x{substr $candidato,0,1};# x di partenza
  $y_start=substr $candidato,1,1;              # y di partenza

  $x=abs($x_start-$x_end);
  $y=abs($y_start-$y_end);
  if(($x==1 && $y==2)||($x==2 && $y==1)){      # Il candidato è idoneo
   push(@candidati,$candidato);
  }
  else{                                        # Il candidato non è idoneo
      push(@non_candidati,$candidato);
  }
 }

 &Move($pezzo,\@candidati,\@non_candidati);
 &Resume();
}


### Gestione alfiere ###
sub B {
local(@posizioni,$posizioni,$pezzo,@candidati,@non_candidati,$candidato,
      $x_start,$y_start,$x_end,$y_end);

 $pezzo=$ENV_player."B";                       # Individua WB o BB
 $posizioni=$ENV_scacchi{$pezzo};              # Posizioni di tutti gli alfieri
 @posizioni=@$posizioni;                       #
```

```
 $x_end=&Is_pari($ENV_mappa_x{@ENV_destinazione[0]});
 $y_end=&Is_pari(@ENV_destinazione[1]);

 while(@posizioni){                              # Scandisco tutte le posizioni
                                                 # degli alfieri alla ricerca di
                                                 # quella giusta
  $candidato=pop(@posizioni);
  $x_start=&Is_pari($ENV_mappa_x{substr $candidato,0,1});
  $y_start=&Is_pari(substr $candidato,1,1);

  if($x_end==$y_end){         # Devo trovare alfiere che muove su p/p o d/d
   if($x_start==$y_start){push(@candidati,$candidato);}
   else{push(@non_candidati,$candidato);}
  }
  else{                       # Devo trovare alfiere che muove su p/d o d/p
   if(($x_start==1 && $y_start ==0)||($x_start==0 && $y_start ==1)){
    push(@candidati,$candidato);
   }
   else{push(@non_candidati,$candidato);}
  }
 }

 &Move($pezzo,\@candidati,\@non_candidati);
 &Resume();
}


### Gestione torre ###
sub R {
local(@posizioni,$posizioni,$pezzo,@candidati,@non_candidati,$candidato,
      $x_start,$y_start,$x_end,$y_end,$x,$y,$i,$ostacolo,$start,$end,
      %rimappa_x);

 $pezzo=$ENV_player."R";                         # Individua WR o BR
 $posizioni=$ENV_scacchi{$pezzo};                # Posizioni di tutti le torri
 @posizioni=@$posizioni;                         #
 $x_end=$ENV_mappa_x{@ENV_destinazione[0]};      # x di arrivo
 $y_end=@ENV_destinazione[1];                    # y di arrivo

 while(@posizioni){                              # Scandisco tutte le posizioni
                                                 # delle torri alla ricerca di
                                                 # quella giusta
  $candidato=pop(@posizioni);
  $x_start=$ENV_mappa_x{substr $candidato,0,1};# x di partenza
  $y_start=substr $candidato,1,1;                # y di partenza

  $x=abs($x_start-$x_end);
  $y=abs($y_start-$y_end);
  if(($x==0 && $y<=7)||($x<=7 && $y==0)){       # Il candidato è idoneo

  if($x==0){                                     # Si muove su una colonna
    if($y_start<=$y_end){                        # Decide la direzione della
                                                 # scansione ostacoli
     $start=$y_start+1;$end=$y_end;
    }
    else{$start=$y_end;$end=$y_start-1;}
    for($i=$start;$i<=$end;$i++){
     if($ENV_scacchiera{substr $candidato,0,1}->[$i-1] ne "**"){
        $ostacolo=1;}                            # Rileva ostacoli sul percorso
    }
   }
   else{                                         # Si muove su una riga
    %rimappa_x=('1',"a",'2',"b",'3',"c",'4',"d",
                '5',"e",'6',"f",'7',"g",'8',"h");
```

```
    if($x_start<=$x_end){     # Decide la direzione della scansione ostacoli
     $start=$x_start+1;$end=$x_end;
    }
    else{$start=$x_end;$end=$x_start-1;}
    for($i=$start;$i<=$end;$i++){
     if($ENV_scacchiera{$rimappa_x{$i}}->[(substr $candidato,1,1)-1]ne "**"){
         $ostacolo=1;}             # Rileva ostacoli sul percorso della torre
    }
   }
   if($ostacolo==0){push(@candidati,$candidato);}# Nessun ostacolo sul cammino
   else{push(@non_candidati,$candidato);}        # Ci sono ostacoli sul cammino
   $ostacolo=0;
  }
  else{ push(@non_candidati,$candidato);}      # Il candidato in analisi non è
                                               # quello giusto
 }
 &Move($pezzo,\@candidati,\@non_candidati);
 &Resume();
}


### Gestione regina ###
sub Q {

 &Move($ENV_player."Q",$ENV_scacchi{$ENV_player."Q"},[]);
 &Resume();
}


### Gestione re ###
sub K {

 &Move($ENV_player."K",$ENV_scacchi{$ENV_player."K"},[]);
 &Resume();
}


##############################################################################
# Ricostruisce la partita fino all'n-esimo round specificato (es.: round 11  #
# muove W). Le mosse sono contenute in un array di cui viene passato un      #
# riferimento.Lo stato della scacchiera è contenuto nella variabile globale  #
# %ENV_scacchiera, così come le posizioni dei pezzi sono contenute in        #
# %ENV_scacchi.                                                              #
##############################################################################

sub Parse_Game {
local(@mosse,$n_mossa,$input,$length,$mossa,$i,$length,$item);

 $input=@_[0];
 @mosse=@$input;
 $n_mossa=@_[1]*2;                        # n_mossa=round; ogni round si compone
                                          # di 2 mosse B e W
### Inizializza scacchiera ###
 %ENV_scacchiera=("a",["WR","WP","**","**","**","**","BP","BR"],
                  "b",["WN","WP","**","**","**","**","BP","BN"],
                  "c",["WB","WP","**","**","**","**","BP","BB"],
                  "d",["WQ","WP","**","**","**","**","BP","BQ"],
                  "e",["WK","WP","**","**","**","**","BP","BK"],
                  "f",["WB","WP","**","**","**","**","BP","BB"],
                  "g",["WN","WP","**","**","**","**","BP","BN"],
                  "h",["WR","WP","**","**","**","**","BP","BR"]);

### Inizializzazione posizione scacchi ###
 %ENV_scacchi=("WP",["a2","b2","c2","d2","e2","f2","g2","h2"],
```

```
                "BP",["a7","b7","c7","d7","e7","f7","g7","h7"],
                "WR",["a1","h1"],
                "BR",["a8","h8"],
                "WN",["b1","g1"],
                "BN",["b8","g8"],
                "WB",["c1","f1"],
                "BB",["c8","f8"],
                "WK",["e1"],
                "BK",["e8"],
                "WQ",["d1"],
                "BQ",["d8"]);


 if(@_[2] eq "W"){$n_mossa-=1;}                  # Se l'ultimo player a muovere è
                                                 # W allora la mossa di B
                                                 # successiva viene omessa
 $length=scalar @mosse;
 for($i=0;$i<($length-$n_mossa);$i++){pop(@mosse);}  # Vengono eliminate tutte
                                                     # le mosse successive a
                                                     # n_mossa

 foreach $mossa (@mosse){                        # Analizza mossa per mossa
  if($mossa eq "O-O"||$mossa eq "O-O-O"){&O($mossa);next;}# Mossa di arrocco
  if($mossa=~ m/.+=.+/){&Promote($mossa);next;}      # Mossa di promozione

  @ENV_token=&Token($mossa);                     # Costruisce il token
  $length= scalar @ENV_token;

  while(@ENV_token){                             # Analizza il token
                                                 # carattere per carattere
   $item=pop @ENV_token;
   if($item eq "P"){&P();next;}                  # Muove pedone
   if($item eq "N"){&N();next;}                  # Muove cavallo
   if($item eq "B"){&B();next;}                  # Muove alfiere
   if($item eq "R"){&R();next;}                  # Muove torre
   if($item eq "Q"){&Q();next;}                  # Muove regina
   if($item eq "K"){&K();next;}                  # Muove re
   if($item eq "x"){&x();next;}                  # Mangiata
   if($item ne "P" && $item ne "N" &&
      $item ne "B" && $item ne "R" &&
      $item ne "Q" && $item ne "K" &&
      $item ne "x"){                             # Informazione di partenza
    push(@ENV_partenza,$item);next;
   }
  }
 }
 $ENV_player="W";                                # Ripristina situazione iniziale
}
```

I file di codice descritti in questa appendice sono inclusi nel cd-rom allegato alla tesi, nella cartella Lit.

B – Realizzazione per Microsoft Reader

Main.pl

```
#!/usr/bin/perl

require ("PGN_Parser.pm");
require ("HtmlTools.pm");

%game=&PGN_Parse("Game.pgn");
$mosse=$game{'Mosse'};
$length_partita= int (scalar @$mosse /2);
if($length_partita*2 != scalar @$mosse){$length_partita+=1;}
$player="W";

print"Preparazione intestazione\n";
$intestazione=&Intestazione(\%game);

print"Pubblicazione pagine:\n";
for($i=0;$i<=$length_partita;$i++){
 $contatore=0;
 while($contatore<2){
  &Parse_Game($game{'Mosse'},$i,$player);
  $pagina="Pagina".$i.$player.".htm";
  $id="Pagina".$i.$player;
  $html=&NuovaPagina;

### Costruzione pagina ###
  $contenuto="$intestazione <BR>".&Scacchiera."<BR>".
  &MenuScelte($game{'Mosse'},$length_partita);
  $html=~ s/##contenuto##/$contenuto/;

  if($i==0){                                              # Prima pagina
    $contatore=1;
    $pagina="Index.htm";
    $player="B";
    $id="Index";
  }
  print"Pagina: $pagina \n";
  open(FILE,"> Output/$pagina")
   or die print"Impossibile scrivere pagina $i";
   print FILE $html;                                      # Scrive la pagina
                                                          # inerente alla mossa
  close(FILE);
  $html="";
  if($player eq "W"){$player="B";}else{$player="W";}       # Alterna W e B
  $contatore++;
  $manifest.="<item id=\"$id\" href=\"$pagina\"
              media-type=\"text/x-oeb1-document\" />\n";
  $spine.="<itemref idref=\"$id\" />\n";
 }
}

### Preparazione package file OPF ###
print"Preparazione Package file\n";
open(OPF,"< modello_package-file.opf")
 or die print"Impossibile trovare file opf";              # Legge il modello OPF
 while(!eof(OPF)){$opf_model.=<OPF>;}
close(OPF);
$opf_model=~ s/##manifest##/$manifest/;
$opf_model=~ s/##spine##/$spine/;
```

```
open(OPF,"> Output/package-file.opf")
 or die print"Impossibile scrivere OPF";                    # Scrive package file OPF
 print OPF $opf_model;
close(OPS);

print "Operazione conclusa con successo (Invio per uscire)";
getc;
```

HtmlTools.pm

```
#!/usr/bin/perl

### Inizializza una nuova pagina ###

sub NuovaPagina{
 local($html);

 $html="<HTML><BODY bgcolor=#99ccff>##contenuto##</BODY></HTML>";

 return($html);
}

#################################################################
# Costruisce l'intestazione del documento con i dati passati    #
# derivanti da PGN_Parse.                                       #
#################################################################

sub Intestazione{
 local($input,%game,$html);

 $input=@_[0];%game=%$input;

 $html="<table border=1 cellspacing=0 cellpadding=0 width=100%>
       <tr>
        <td><font face=Verdana, Arial, Helvetica, sans-serif size=1>
         EVENTO: <b>$game{'Evento'}</b></font></td>
        <td><font face=Verdana, Arial, Helvetica, sans-serif size=1>
         LUOGO: <b>$game{'Luogo'}</b></font></td>
        <td><font face=Verdana, Arial, Helvetica, sans-serif size=1>
         DATA: <b>$game{'Data'}</b></font></td>
       </tr>
       <tr>
        <td><font face=Verdana, Arial, Helvetica, sans-serif size=1>
         BIANCO: <b>$game{'Bianco'}</b></font></td>
        <td><font face=Verdana, Arial, Helvetica, sans-serif size=1>
         NERO: <b>$game{'Nero'}</b></font></td>
        <td><font face=Verdana, Arial, Helvetica, sans-serif size=1>
         RISULTATO: <b>$game{'Risultato'}</font></td>
       </tr>
      </table>";

 return($html);
}

### Costruisce la scacchiera basandosi sulle strutture dati di PGN_Parser ###

sub Scacchiera{
 local($html,$colore,$riga,$colonna,@ordine,$pezzo);

 @ordine=("a","b","c","d","e","f","g","h");
```

```
$html="<TABLE border=1 bordercolor=#000000 cellspacing=0 cellpadding=0
       align=center>";
$colore="#99ccff";
for($riga=8;$riga>0;$riga--){
 $html.="<TR><TD bgcolor=93D090 width=10 align=center>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>$riga</B></font>
              </TD>";
 foreach $colonna(@ordine){
  $pezzo=$ENV_scacchiera{$colonna}->[$riga-1];
  $html.="<TD bgcolor=$colore width=40 height=40 valign=middle align=center>
           $pezzo</TD>";
  if($colore eq "#FF0000"){$colore="#99ccff";}else{$colore="#FF0000";};
 }
 $html.="</TR>";
 if($colore eq "#FF0000"){$colore="#99ccff";}else{$colore="#FF0000";};
}
$html.="<TR><TD height=10> </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>A</B></font>
            </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>B</B></font>
            </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>C</B></font>
            </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>D</B></font>
            </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>E</B></font>
            </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>F</B></font>
            </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>G</B></font>
            </TD>
            <TD bgcolor=93D090 align=center valign=middle>
                 <font face=Verdana, Arial, Helvetica, sans-serif size=1>
                 <B>H</B></font>
            </TD>
       </TR></TABLE>";

$html=~ s/WP/<font face="Chess Alpha" size="7">p<\/font>/g;
$html=~ s/BP/<font face="Chess Alpha" size="7">o<\/font>/g;
$html=~ s/WK/<font face="Chess Alpha" size="7">k<\/font>/g;
$html=~ s/BK/<font face="Chess Alpha" size="7">l<\/font>/g;
$html=~ s/WQ/<font face="Chess Alpha" size="7">q<\/font>/g;
$html=~ s/BQ/<font face="Chess Alpha" size="7">w<\/font>/g;
$html=~ s/WR/<font face="Chess Alpha" size="7">r<\/font>/g;
$html=~ s/BR/<font face="Chess Alpha" size="7">t<\/font>/g;
$html=~ s/WB/<font face="Chess Alpha" size="7">b<\/font>/g;
$html=~ s/BB/<font face="Chess Alpha" size="7">n<\/font>/g;
$html=~ s/WN/<font face="Chess Alpha" size="7">h<\/font>/g;
$html=~ s/BN/<font face="Chess Alpha" size="7">j<\/font>/g;
```

```
 $html=~ s/\*\*/ /g;

 return($html);
}
### Costruisce menu mosse ###

sub MenuScelte{
 local($html,@mosse,$input,$W,$B,$contatore,$paginaW,
       $paginaB,$rounds,$conta_righe);

 $input=@_[0];@mosse=@$input;@mosse=reverse @mosse;
 $rounds=@_[1];

 $html.="<TABLE width=100% border=1 cellspacing=3 cellpadding=0>
           <TR><TD align=center colspan=3 bgcolor=00cc00>
                <B><a href=Index.htm><font face=Verdana, Arial, Helvetica,
                    sans-serif size=1 color=000000>Inizio</font>
                  </a></B></TD></TR><TR>";

 while(@mosse){
  $contatore++;$conta_righe++;
  $W=pop(@mosse);$paginaW="Pagina".$contatore."W.htm";
  $B=pop(@mosse);$paginaB="Pagina".$contatore."B.htm";;

  if($conta_righe==7){$html.="</TR><TR>";$conta_righe=1;}
  $html.="<TD align=center valign=middle width=15 bgcolor=93D090><B>
          <font face=Verdana,Arial,Helvetica,sans-serif size=1>$contatore</font>
          </B></TD>
          <TD align=center valign=middle width=40><a href=$paginaW>
            <font face=Verdana, Arial, Helvetica, sans-serif size=1
             color=000000>$W</font></a></TD>
          <TD align=center valign=middle width=40><a href=$paginaB>
            <font face=Verdana, Arial, Helvetica, sans-serif size=1
             color=000000>$B</font></a></TD>";
 }

 $html.="</TABLE>";
 return($html);
}
1
```

Modello di package file

```
<package unique-identifier="isbn">
  <metadata>
     <dc-metadata
       xmlns:dc="http://purl.org/metadata/dublin_core"
       xmlns:oebpackage="http://openebook.org/namespaces/oeb-package/1.0/">
          <dc:Title>Problema scacchistico</dc:Title>
          <dc:Contributor role="aui">Matteo Lancellotti</dc:Contributor>
          <dc:Contributor role="art">Matteo Lancellotti</dc:Contributor>
          <dc:Creator file-as="Lancellotti, Matteo" role="aut">
              Matteo Lancellotti</dc:Creator>
          <dc:Date></dc:Date>
          <dc:Identifier id="isbn">0-451-52671-6</dc:Identifier>
          <dc:Language>it</dc:Language>
          <dc:Publisher>Matteo Lancellotti</dc:Publisher>
    </dc-metadata>
    <x-metadata><meta name="ms-chaptertour" content="chaptertour"/></x-metadata>
   </metadata>
   <manifest>
```

```
      ##manifest##
   </manifest>
   <spine>
     ##spine##
   </spine>
   <guide></guide><tours></tours>
</package>
```

I file di codice descritti in questa appendice sono inclusi nel cd-rom allegato alla tesi, nella cartella Lit.

C – Realizzazione per Microsoft Word

Main.pl

```
#!/usr/bin/perl

require ("PGN_Parser.pm");
use Tk;

### Routine gestione creazione documenti ###

sub Crea{
local($dir,%game,$PGN_file,$mosse,$length_partita,$player,$event_model,
      $i,$contatore,$content,$evento);

 use Cwd;
 $dir = cwd();                              # Identifica directory di lavoro
 $PGN_file=$file->get;                      # Percorso e nome file PGN

 if($PGN_file eq ""){return;}               # Se nessun file è specificato esce

 %game=&PGN_Parse($PGN_file);               # Acquisisce il gioco dal file PGN
 $mosse=$game{'Mosse'};
 $length_partita= int (scalar @$mosse /2);# Individua il numero di turni della
                                           # partita
 if($length_partita*2 != scalar @$mosse){$length_partita+=1;}#
 $player="W";

 open(MOD,"< Modello_evento.txt") or die                  # Legge il modello
 $screen->insert('end',"Impossibile trovare il modello");# degli eventi
  while(!eof(MOD)){$event_model.=<MOD>;}                  #
 close(MOD);                                              #

 $screen->delete(0,'end');
 $screen->insert('end',"Creazione eventi:");
 for($i=0;$i<=$length_partita;$i++){
  $contatore=0;
  while($contatore<2){
   $content=$event_model;                                 # L'evento assume la forma
                                                          # base del modello eventi
   $content=~ s/##Parser Data##/$i,$player/;              # Inserisce dati mancanti
                                                          # nelle righe di comando
                                                          # del modello
   $content=~ s/##Length Partita##/$length_partita/;      #
   $content=~ s/##Work Directory##/"$dir"/;               #
   $content=~ s/##PGN File##/$PGN_file/;                  #
   $content=~ s/##Output Directory##/"$dir\/Output"/;     #
   $evento="Evento".$i.$player.".pl";                     # Nome file fisico

   ### Costruzione eventi ###

   if($i==0){$contatore=1;$evento="Start.pl";$player="B";} # Evento iniziale
   $screen->insert('end',"Evento: $evento");
   open(FILE,"> Output/$evento") or die
   print"Impossibile scrivere file $i";                   # Scrive la pagina inerente
                                                          # alla mossa
    print FILE $content;                                  #
   close(FILE);                                           #

   if($player eq "W"){$player="B";}else{$player="W";}# Alterna W e B
   $contatore++;
```

```
  }
 }

 $screen->insert('end',"Operazione conclusa con successo");
 $word->configure(-state=>'active');
}

### Routine gestione avvio di Word ###

sub Avvia{

 my $path="Word.bat";
 system $path;
}


###################
#   Main Window   #
###################

### Crea finestra principale non ridimensionabile ###

$win= MainWindow->new(-width=>340, -height=>490, -title=>"PGN to WORD v 2.0");
$win->resizable(0,0);

### Sezione scelta file ###

$win->Label(-text=>"Seleziona file PGN:")->place(-x=>7,-y=>10);
$file= $win->Entry(-textvariable => \$file_path,-width=>40)->
       place(-x=>10,-y=>30);                    # Casella input testo (file PGN)
$win->Button(-text => 'Cerca',
             -command => sub {$file_path = $win->getOpenFile(-filetypes=>
                                           [['PGN files',['.pgn']]]) })
    ->place(-x=>275,-y=>25,-width=>50);

### Sezione screen ###

$screen=$win->Scrolled('Listbox', -scrollbars=>'oe',
                                  -height=>20,
                                  -width=>48,
                                  -selectmode=>'single')
              ->place(-x=>10,-y=>150);

### Sezione tasti ###

$win->Button(-text=>"Crea documenti",
             -command=> \&Crea,-width=>15)->place(-x=>10, -y=>90);
$word=$win->Button(-text=>"Avvia Word",
                   -command=> \&Avvia,-width=>15,
                   -state=>'disabled')->place(-x=>220, -y=>90);

### Icona finestra principale ###
$win->update;
my $icon = $win->Photo(-file => 'icon.gif');
$win->iconimage($icon);

MainLoop;
```

Modello degli eventi

```
#!/usr/bin/perl

push(@INC,##Work Directory##);

use Win32::OLE qw(in with);
use Win32::OLE::Const 'Microsoft Word';
use Win32::OLE::Variant;
require("HtmlTools.pm");
require ("PGN_Parser.pm");

my $dir = ##Output Directory##;                 # Individua directory di lavoro

%game=&PGN_Parse("##PGN File##");               # Acquisizione dal file PGN
$mosse=$game{'Mosse'};
&Parse_Game($game{'Mosse'},##Parser Data##);    # Sviluppa il gioco fino al
                                                # livello richiesto

### Costruzione Pagina HTML ###

$html = &NuovaPagina();                         # Carica struttura nuova pagina
$contenuto=&Intestazione(\%game)."<BR>".
           &Scacchiera."<BR>".
           &MenuScelte($game{'Mosse'},##Length Partita##);
$html=~ s/##contenuto##/$contenuto/;

################################
# Apertura della pagina in Word #
################################

### Apre il file temporaneo in Word ###
my $word;my $doc;
$word = Win32::OLE->GetActiveObject('Word.Application');# Cattura eventuali
                                                         # finestre word attive
if(!(defined $word)){                                 # Se non ci sono finestre
                                                      # attive ne apre una
 $word=Win32::OLE->new('Word.Application');
 $word->{Visible}="true";
}
else{                                                 # Se c'è una finestra
 $doc = $word->{'Documents'};                         # attiva ne chiude il
 $doc->Close;                                         # contenuto
}                                                     #

### Scrive file temporaneo contenente codice html ###
open(TMP,"> $dir/Source.tmp") || die "Errore di scrittura nel file\n\n";
 print TMP $html;
close(TMP);

Win32::OLE->Option(Warn => 3);
$doc = $word->{'Documents'}->Open("$dir/Source.tmp");# Apre file temporaneo


undef $doc;
undef $word;
```

I file di codice descritti in questa appendice sono inclusi nel cd-rom allegato alla tesi, nella cartella Word

D – Chess Editor

Ambiente.pm

```
#!/usr/bin/perl

### Colori scacchiera ###

$ENV_color1="light blue";          # Colore caselle chiare
$ENV_color2="#FF0000";             # Colore caselle scure
$ENV_move_col="#ACFFB6";           # Colore visualizzazione pezzo da muovere
$ENV_eat_col="#4CA556";            # Colore visualizzazione mosse possibili

### Mappatura colonne ###

%ENV_colonne=('0',"a",'1',"b",'2',"c",'3',"d",
              '4',"e",'5',"f",'6',"g",'7',"h");

### Inizializzazione scacchiera ###

%ENV_scacchiera=("a",[{Pezzo=>"WR", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BR", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
                 "b",[{Pezzo=>"WN", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BN", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
                 "c",[{Pezzo=>"WB", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BB", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
                  "d",[{Pezzo=>"WQ", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BQ", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
                  "e",[{Pezzo=>"WK", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                      {Pezzo=>"BK", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
```

```
                "f",[{Pezzo=>"WB", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"BB", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
                "g",[{Pezzo=>"WN", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"BN", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
                "h",[{Pezzo=>"WR", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"WP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"**", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"BP", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                     {Pezzo=>"BR", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}]);


### Variabili globali ###

%ENV_let_num=('a',1,'b',2,'c',3,'d',4,
              'e',5,'f',6,'g',7,'h',8);         # Traduce le lettere in numeri

%ENV_num_let=('1','a','2','b','3','c','4','d',
              '5','e','6','f','7','g','8','h');# Traduce i numeri in lettere

$ENV_Click=0;                                   # Indica se c'è stato o meno un click
$ENV_Player="W";                                # Indica il giocatore che deve muovere
$ENV_Start="";                                  # Casella di partenza della mossa
$ENV_Mangiata=0;                                # Segnala una presa nell'ultima mossa
                                                # effettuata
$ENV_Change="";                                 # Segnala un cambiamento nella
                                                # sequenza di mosse
@ENV_candidati=();                              # Lista delle possibili posizioni su
                                                # cui muovere
%ENV_Stack=();                                  # Stack per il salvataggio delle
                                                # configurazioni di scacchiera
$ENV_arrocco_WB="";                             # Specifica la scelta colore
                                                # nell'arrocco (W|B)
$ENV_arrocco_LC="";                             # Specifica la scelta del tipo di
                                                # arrocco (Lungo|Corto)
$ENV_Promozione="";                             # Indica il pezzo scelto per la
                                                # promozione
$ENV_pgn=0;                                     # Specifica la modalità di salvataggio
                                                # nel formato pgn (0|1)
$ENV_pdf=0;                                     # Specifica la modalità di salvataggio
                                                # nel formato pdf (0|1)


### Variabili sezione loader (parser) ###

@ENV_destinazione=();                           # Destinazione del pezzo in esame: x,y
@ENV_partenza=();                               # Informazioni aggiuntive sulla
                                                # locazione di partenza del pezzo
@ENV_token=();                                  # Token in analisi
```

Chess.pm

```
#!/usr/bin/perl

#############################################################
# Funzione per accendere le caselle candidate alla mossa #
#############################################################

sub Accendi{
local($casella,@coords,$candidato);

 foreach $candidato (@ENV_candidati){
  @coords=split("-",$candidato);
  $casella=$ENV_scacchiera{@coords[0]}->
              [@coords[1]-1]->{'Ptr'};         # Prende puntatore a casella
  $casella->configure(-bg=>$ENV_move_col);     # Cambia colore casella
  $ENV_scacchiera{@coords[0]}->
[@coords[1]-1]->{'Sel'}=1;                     # La casella viene etichettata
                                               # come 'Selezionata'
 }
 @ENV_candidati=();

}


#########################################################################
# Funzione di gestione lista caselle candidate alla mossa. Controlla  #
# se la casella di coordinate x,y date può essere un candidato o no . #
# Input = specifica se pedone mangia (1) | pedone non mangia (0) |    #
# pezzo diverso da pedone (2), coordinate x-y casella da controllare. #
#########################################################################

sub Candidato{
local ($mangia,$x,$y,$pezzo,$player);

 ($mangia,$x,$y)=@_;
 $pezzo=$ENV_scacchiera{$ENV_num_let{$x}}->
            [$y-1]->{'Pezzo'};                 # Eventuale pezzo sulla casella
 $player=substr $pezzo,0,1;                    # Controlla colore pezzo (W|B)

 if($mangia==1){                               # Diagonali pedone
  if($pezzo ne "**" && $player ne $ENV_Player){
   push(@ENV_candidati,$ENV_num_let{$x}."-".($y));}
 }
 if($mangia==0){                               # Movimento rettilineo pedone
  if($pezzo eq "**"){push(@ENV_candidati,$ENV_num_let{$x}."-".($y));}
 }
 if($mangia==2){                               # Pezzi diversi dai pedoni
  if($pezzo eq "**" || $player ne $ENV_Player){
   push(@ENV_candidati,$ENV_num_let{$x}."-".($y));}
 }
}


### Gestione pedone ###
sub P {
local(@coords,$mosso,$x_start,$y_start,$pezzo);

 @coords=split("-",$ENV_Start);                # Coordinate casella di partenza
 $mosso=$ENV_scacchiera{@coords[0]}->
            [@coords[1]-1]->{'Mosso'};         # Dice se il pedone ha già mosso
                                               # almeno una volta
```

```
 $x_start=$ENV_let_num{@coords[0]};            # Traduce la lettera in numero
 $y_start=@coords[1];

 if($ENV_Player eq "W"){                       # Pedone bianco, analizza le 4
                                               # mosse possibili

  if($y_start!=8){&Candidato(0,$x_start,$y_start+1);}      # Avanti di 1
  if($mosso==0 && $y_start<7){                             # Avanti di 2
   $pezzo=$ENV_scacchiera{$ENV_num_let{$x_start}}->[@coords[1]]->{'Pezzo'}.
          $ENV_scacchiera{$ENV_num_let{$x_start}}->[@coords[1]+1]->{'Pezzo'};
   if($pezzo eq "****"){                                   # Nessun ostacolo
    push(@ENV_candidati,$ENV_num_let{$x_start}."-".($y_start+2));}
  }
  if($y_start!=8 && $x_start!=8){                          # Mangia a dx
   &Candidato(1,$x_start+1,$y_start+1);}
  if($y_start!=8 && $x_start!=1){                          # Mangia a sx
   &Candidato(1,$x_start-1,$y_start+1);}

 }
 else{                                         # Pedone nero, analizza le 4
                                               # mosse possibili

  if($y_start!=1){&Candidato(0,$x_start,$y_start-1);}# Avanti di 1
  if($mosso==0 && $y_start>2){                             # Avanti di 2
   $pezzo=$ENV_scacchiera{$ENV_num_let{$x_start}}->[@coords[1]-2]->{'Pezzo'}.
          $ENV_scacchiera{$ENV_num_let{$x_start}}->[@coords[1]-3]->{'Pezzo'};
   if($pezzo eq "****"){                                   # Nessun ostacolo
    push(@ENV_candidati,$ENV_num_let{$x_start}."-".($y_start-2));}
  }
  if($y_start!=1 && $x_start!=8){                          # Mangia a dx
   &Candidato(1,$x_start+1,$y_start-1);}
  if($y_start!=1 && $x_start!=1){                          # Mangia a sx
   &Candidato(1,$x_start-1,$y_start-1);}

 }
 &Accendi;
}


### Gestione conflitti pedone ###
sub P_conf{
local($x_start,$y_start,$x_end,$y_end);

 ($x_start,$y_start,$x_end,$y_end)=@_;

 if($ENV_Mangiata==1){                  # I conflitti esistono solo su mangiata

  if($x_end eq "a" || $x_end eq "h"){return("");}    # Se il pedone arriva sul
                                                     # bordo non ha conflitti
  if($ENV_Player eq "W"){                            # Pedone bianco
   $y_end--;
   if($ENV_scacchiera{$ENV_num_let{$ENV_let_num{$x_end}-1}}->
      [$y_end-1]->{'Pezzo'} eq "WP" ||
      $ENV_scacchiera{$ENV_num_let{$ENV_let_num{$x_end}+1}}->
      [$y_end-1]->{'Pezzo'} eq "WP"){return($x_start);}
  }
  else{                                              # Pedone nero
   $y_end++;
   if($ENV_scacchiera{$ENV_num_let{$ENV_let_num{$x_end}-1}}->
      [$y_end-1]->{'Pezzo'} eq "BP" ||
      $ENV_scacchiera{$ENV_num_let{$ENV_let_num{$x_end}+1}}->
      [$y_end-1]->{'Pezzo'} eq "BP"){return($x_start);}
  }
```

```
 }
}



### Gestione cavallo ###
sub N {
local(@coords,$x_start,$y_start);

 @coords=split("-",$ENV_Start);                # Coordinate casella di partenza
 $x_start=$ENV_let_num{@coords[0]};            # Traduce la lettera in numero
 $y_start=@coords[1];

 if($y_start+2 <= 8){                                                  # Muove in alto
  if($x_start-1 >= 1){&Candidato(2,$x_start-1,$y_start+2);}      # Sx
  if($x_start+1 <= 8){&Candidato(2,$x_start+1,$y_start+2);}      # Dx
 }

 if($y_start-2 >= 1){                                                  #Muove in basso
  if($x_start-1 >= 1){&Candidato(2,$x_start-1,$y_start-2);}      # Sx
  if($x_start+1 <= 8){&Candidato(2,$x_start+1,$y_start-2);}      # Dx
 }

 if($x_start+2 <= 8){                                                  # Muove a dx
  if($y_start-1 >= 1){&Candidato(2,$x_start+2,$y_start-1);}      # Giu
  if($y_start+1 <= 8){&Candidato(2,$x_start+2,$y_start+1);}      # Su
 }

 if($x_start-2 >= 1){                                                  # Muove a sx
  if($y_start-1 >= 1){&Candidato(2,$x_start-2,$y_start-1);}      # Giu
  if($y_start+1 <= 8){&Candidato(2,$x_start-2,$y_start+1);}      # Su
 }

 &Accendi;
}


### Gestione conflitti cavallo ###
sub N_conf{
local($x_start,$y_start,$x_end,$y_end,$x,$y,@coords,$output);

 ($x_start,$y_start,$x_end,$y_end)=@_;

 for($x=1;$x<=8;$x++){                                 # Trova l'altro cavallo
  for($y=1;$y<=8;$y++){
   if($ENV_scacchiera{$ENV_num_let{$x}}->[$y-1]->{'Pezzo'} eq
      $ENV_Player."N" && ($y != $y_end || $x != $ENV_let_num{$x_end})){
    $ENV_Start=$ENV_num_let{$x}."-".$y;
    $ENV_scacchiera{$x_end}->[$y_end-1]->
      {'Pezzo'}="**";                                  # Simula casella vuota per
                                                       # controllare conflitti
      &N();                                            # Calcola le mosse
                                                       # possibili dell'altro
                                                       # cavallo
    $ENV_scacchiera{$x_end}->
         [$y_end-1]->{'Pezzo'}=$ENV_Player."N";        # Ripristina il pezzo
   }
  }
 }
 if($ENV_scacchiera{$x_end}->[$y_end-1]->{'Sel'}){     # Esiste un conflitto
  @coords=split("-",$ENV_Start);
  $x=@coords[0]; $y=@coords[1];                        # Coordinate del pezzo in
                                                       # conflitto
 if($x eq $x_start){$output=$y_start;}                 # I due cavalli sono sulla
```

```
                                                        # stessa colonna
  if($y eq $y_start){$output=$x_start;}                 # I due cavalli sono sulla
                                                        # stessa riga
  if($x ne $x_start && $y ne $y_end){$output=$x_start;}# I due cavalli hanno
                                                        # entrambe le
                                                        # coordinate diverse
 }

 return("N".$output);
}


### Gestione alfiere ###
sub B {
local(@coords,$x_start,$y_start,$ostacolo);

 @coords=split("-",$ENV_Start);                 # Coordinate casella di partenza
 $x_start=$ENV_let_num{@coords[0]};             # Traduce la lettera in numero
 $y_start=@coords[1];

 while($x_start<8 && $y_start<8 && $ostacolo==0){# Diagonale alto dx
  $x_start++;
  $y_start++;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}    # Al primo ostacolo si ferma
  &Candidato(2,$x_start,$y_start);
 }
 $x_start=$ENV_let_num{@coords[0]};
 $y_start=@coords[1];
 $ostacolo=0;

 while($x_start>1 && $y_start<8 && $ostacolo==0){# Diagonale alto sx
  $x_start--;
  $y_start++;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}    # Al primo ostacolo si ferma
  &Candidato(2,$x_start,$y_start);
 }
 $x_start=$ENV_let_num{@coords[0]};
 $y_start=@coords[1];
 $ostacolo=0;

 while($x_start<8 && $y_start>1 && $ostacolo==0){# Diagonale basso dx
  $x_start++;
  $y_start--;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}    # Al primo ostacolo si ferma
  &Candidato(2,$x_start,$y_start);
 }
 $x_start=$ENV_let_num{@coords[0]};
 $y_start=@coords[1];
 $ostacolo=0;

 while($x_start>1 && $y_start>1 && $ostacolo==0){# Diagonale basso sx
  $x_start--;
  $y_start--;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}    # Al primo ostacolo si ferma
  &Candidato(2,$x_start,$y_start);
 }

 &Accendi;
}
```

```
### Gestione torre ###
sub R {
local(@coords,$x_start,$y_start,$ostacolo);

 @coords=split("-",$ENV_Start);                 # Coordinate casella di partenza
 $x_start=$ENV_let_num{@coords[0]};             # Traduce la lettera in numero
 $y_start=@coords[1];

 while($x_start<8 && $ostacolo==0){             # Muove verso dx
  $x_start++;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}    # Al primo ostacolo si ferma
  &Candidato(2,$x_start,$y_start);
 }
 $x_start=$ENV_let_num{@coords[0]};
 $y_start=@coords[1];
 $ostacolo=0;

 while($x_start>1 && $ostacolo==0){             # Muove verso sx
  $x_start--;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}
  &Candidato(2,$x_start,$y_start);
 }
 $x_start=$ENV_let_num{@coords[0]};
 $y_start=@coords[1];
 $ostacolo=0;

 while($y_start<8 && $ostacolo==0){             # Muove verso l'alto
  $y_start++;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}
  &Candidato(2,$x_start,$y_start);
 }
 $x_start=$ENV_let_num{@coords[0]};
 $y_start=@coords[1];
 $ostacolo=0;

 while($y_start>1 && $ostacolo==0){             # Muove verso il basso
  $y_start--;
  if($ENV_scacchiera{$ENV_num_let{$x_start}}->[$y_start-1]->
            {'Pezzo'} ne "**"){$ostacolo=1;}
  &Candidato(2,$x_start,$y_start);
 }

 &Accendi;
}


### Gestione conflitti torre ###
sub R_conf{
local($x_start,$y_start,$x_end,$y_end,$x,$y,@coords,$output);

 ($x_start,$y_start,$x_end,$y_end)=@_;

 for($x=1;$x<=8;$x++){                          # Trova l'altra torre
  for($y=1;$y<=8;$y++){
   if($ENV_scacchiera{$ENV_num_let{$x}}->[$y-1]->{'Pezzo'} eq
      $ENV_Player."R" && ($y != $y_end || $x != $ENV_let_num{$x_end})){
      $ENV_Start=$ENV_num_let{$x}."-".$y;
      $ENV_scacchiera{$x_end}->[$y_end-1]->
         {'Pezzo'}="**";      # Simula casella vuota per controllare conflitti
      &R();                   # Calcola le mosse possibili dell'altra torre
```

```
      $ENV_scacchiera{$x_end}->[$y_end-1]->
            {'Pezzo'}=$ENV_Player."R";          # Ripristina il pezzo
    }
   }
  }

 if($ENV_scacchiera{$x_end}->[$y_end-1]->{'Sel'}){# Esiste un conflitto
  @coords=split("-",$ENV_Start);
  $x=@coords[0]; $y=@coords[1];           # Coordinate del pezzo in conflitto
  if($x eq $x_start){$output=$y_start;}   # Torri sulla stessa colonna
  if($y eq $y_start){$output=$x_start;}   # Torri sulla stessa riga
  if($x ne $x_start && $y ne $y_end){$output=$x_start;}    # Le due torri hanno
                                                           # entrambe le
                                                           # coordinate diverse
 }

 return("R".$output);
}

### Gestione regina ###
sub Q {

 &R();                                          # Calcola movimenti rettilinei
 &B();                                          # Calcola movimenti diagonali
}


### Gestione re ###
sub K {
local(@coords,$x_start,$y_start);

 @coords=split("-",$ENV_Start);                 # Coordinate casella di partenza
 $x_start=$ENV_let_num{@coords[0]};             # Traduce la lettera in numero
 $y_start=@coords[1];

 if($y_start+1 <= 8){                                           # Su
  &Candidato(2,$x_start,$y_start+1);
  if($x_start+1 <= 8){&Candidato(2,$x_start+1,$y_start+1);}     # Su-Dx
  if($x_start-1 >= 1){&Candidato(2,$x_start-1,$y_start+1);}     # Su-Sx
 }

 if($y_start-1 >= 1){                                           # Giu
  &Candidato(2,$x_start,$y_start-1);
  if($x_start+1 <= 8){&Candidato(2,$x_start+1,$y_start-1);}     # Giu-Dx
  if($x_start-1 >= 1){&Candidato(2,$x_start-1,$y_start-1);}     # Giu-Dx
 }

 if($x_start+1 <= 8){&Candidato(2,$x_start+1,$y_start);}        # Dx

 if($x_start-1 >= 1){&Candidato(2,$x_start-1,$y_start);}        # Sx

 &Accendi;
}
```

Eventi.pm

```
#!/usr/bin/perl

require("Chess.pm");
require("Salvataggi.pm");
require("Loader.pm");

#######################################################
# Funzione di congelamento configurazione scacchiera #
#######################################################

sub Freeze{
local(%tmp_conf,$colonna,$i);

%tmp_conf=("a",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
           "b",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
           "c",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
           "d",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
           "e",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
           "f",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                  {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
```

```
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
          "g",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}],
          "h",[{Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0},
                {Pezzo=>"", Ptr=>'', Bg=>'', Sel=>0, Mosso=>0}]);

 foreach $colonna (keys %ENV_scacchiera){
  for($i=0;$i<8;$i++){
   $tmp_conf{$colonna}->[$i]->{'Pezzo'}=
      $ENV_scacchiera{$colonna}->[$i]->{'Pezzo'};
   $tmp_conf{$colonna}->[$i]->{'Ptr'}=$ENV_scacchiera{$colonna}->[$i]->{'Ptr'};
   $tmp_conf{$colonna}->[$i]->{'Bg'}=$ENV_scacchiera{$colonna}->[$i]->{'Bg'};
   $tmp_conf{$colonna}->[$i]->{'Sel'}=$ENV_scacchiera{$colonna}->[$i]->{'Sel'};
   $tmp_conf{$colonna}->[$i]->{'Mosso'}=
      $ENV_scacchiera{$colonna}->[$i]->{'Mosso'};
  }
 }

 return(%tmp_conf);
}


#####################################################
# Funzione di ripristino valori default dopo mossa #
#####################################################

sub Resume{
local($colonna,$casella,$ptr,$color);

 $ENV_Click=0;
 $ENV_Mangiata=0;
 foreach $colonna (keys %ENV_scacchiera){            # Controlla tutte le
                                                     # colonne della scacchiera
  for($casella=0;$casella<8;$casella++){             # Controlla ogni casella
                                                     # della colonna
   $ENV_scacchiera{$colonna}->[$casella]->{'Sel'}=0; # Casella deselezionata
   $ptr=$ENV_scacchiera{$colonna}->[$casella]->{'Ptr'};# Puntatore alla casella
   $color=$ENV_scacchiera{$colonna}->[$casella]->{'Bg'};# Ripristina aspetto e
                                                     # colore originali
   $ptr->configure(-bg=>$color,-relief=>'solid');    #
  }
 }
}


### Finestra salvataggio dei file ###
sub Salva {

 # Crea finestra di dialogo
 $s_window=$win->Toplevel(-title=>"Salva con nome",-width=>320,-height=>100);
 $s_window->resizable(0,0);
 $s_window->grab;                                # La finestra diviene un popup
```

```
 $s_window->Label(-text=>"Nome file:")->place(-x=>7,-y=>10);
 $s_window->Checkbutton(-text=>"PGN",-variable=>\$ENV_pgn)->
           place(-x=>5,-y=>50);               # Scelta PGN
 $s_window->Checkbutton(-text=>"PDF",-variable=>\$ENV_pdf)->
           place(-x=>5,-y=>70);               # Scelta PDF

 my $path;
 $s_window->Button(-text => 'Cerca',
                   -command => sub {$path = $s_window->getSaveFile})->
           place(-x=>265,-y=>25,-width=>50);
 $s_filename=$s_window->Entry(-textvariable => \$path,-width=>40)->
                  place(-x=>10,-y=>30);
 $s_window->Button(-text=>"OK",-command=>\&Salva_OK)->
                  place(-x=>265,-y=>65,-width=>50);

 $s_window->update;
 my $icon = $s_window->Photo(-file => 'immagini/icon.gif');
 $s_window->iconimage($icon);
}


### Pulsante salvataggio dati ###
sub Salva_OK{

 if($ENV_pgn==1){&PGN($s_filename->get);}
 if($ENV_pdf==1){&PDF($s_filename->get);}

 $s_window->grabRelease;                 # Distrugge la finestra di dialogo
 $s_window->destroy;                     #
 undef $s_window;
}


### Finestra caricamento dati da file ###
sub Load {

 $s_window=$win->Toplevel(-title=>"Carica partita",-width=>320,-height=>100);
      # Crea finestra di dialogo
 $s_window->resizable(0,0);
 $s_window->grab;                        # La finestra diviene un popup
 $s_window->Label(-text=>"Nome file:")->place(-x=>7,-y=>10);

 my $path;
 $s_window->Button(-text => 'Cerca',
                   -command => sub {$path = $s_window->getOpenFile(
                   -filetypes=>[['PGN files',['.pgn']]]) })->
           place(-x=>265,-y=>25,-width=>50);
 $s_filename=$s_window->Entry(-textvariable => \$path,-width=>40)->
             place(-x=>10,-y=>30);
 $s_window->Button(-text=>"OK",-command=>\&Load_OK)->
             place(-x=>265,-y=>65,-width=>50);

 $s_window->update;
 my $icon = $s_window->Photo(-file => 'immagini/icon.gif');
 $s_window->iconimage($icon);
}

### Pulsante caricamento dati ###
sub Load_OK{

 &Loader($s_filename->get);
 $s_window->grabRelease;                 # Distrugge la finestra di dialogo
 $s_window->destroy;                     #
 undef $s_window;
```

```
}

###  Gestisce finestra promozione ###
sub Promote {
local($pezzo,$img,%lista_pezzi,$casella,$x,$y,
      $player,$item,$img_name);

 if($ENV_Promozione ne ""){return(0);}    # L'operazione di promozione è stata
                                          # invocata dal loader

 $s_window=$win->Toplevel(-title=>"Promozione pedone",
                          -width=>220,
                          -height=>100); # Crea finestra di dialogo
 $s_window->grab;                        # La finestra diviene un popup
 $s_window->Label(-text=>"Scegli un pezzo per la promozione:")
           ->place(-x=>7,-y=>10);

 %lista_pezzi=('Q',1,'R',2,'N',2,'B',2);

 foreach $pezzo (keys %lista_pezzi){
  my $nome=$pezzo;
  $img_name=$pezzo."_no";
  $casella=$s_window->Canvas(-width=>40, -height=>40)->place(-x=>7+$x,-y=>35);
  if($lista_pezzi{$pezzo}>0){
      ### Mappa l'azione di click sulla casella ###
      $casella->Tk::bind('<Button-1>' =>sub{$ENV_Promozione=$nome;
                                            $s_window->grabRelease;
                                            $s_window->destroy;
                                            undef $s_window;});
    $img_name=$ENV_Player.$pezzo;
  }
  $img=$s_window->Photo(-file=>"immagini/$img_name.gif");# Carica l'immagine
  $casella->createImage(23,23, -image=>$img,-tag=>"img");# Posiziona l'immagine
                                                         # nella casella
  $x+=42;
 }
 $s_window->update;
 my $icon = $s_window->Photo(-file => 'immagini/icon.gif');
 $s_window->iconimage($icon);
 $s_window->waitWindow;             # La finestra principale si mette in
                                    # attesa del completamento dell'operazione
}

### Gestisce pulsante arrocco ###
sub Arrocco{
local($x_re_start,$y_re_start,$x_re_end,$y_re_end,
      $x_torre_start,$y_torre_start,$x_torre_end,$y_end,
      $re,$torre,$img,$img_name_re,$img_name_torre,%tmp_conf,
      @stack,$n_mossa,$riga);

 if($ENV_arrocco_WB eq "" || $ENV_arrocco_LC eq "" ||
    $ENV_arrocco_WB ne $ENV_Player){return(0);}      # Dati mancanti o giocatore
                                                     # errato
 $x_re_start="e";
 if($ENV_arrocco_WB eq "W" &&
    $ENV_scacchiera{'e'}->[0]->{'Pezzo'} eq "WK"){   # Giocatore bianco e
                                                     # posizione re corretta
  $y_re_start=0;
  $img_name_re="WK";$img_name_torre="WR";            # Nomi immagini dei pezzi
  if($ENV_arrocco_LC eq "L" &&
     $ENV_scacchiera{'a'}->[0]->{'Pezzo'} eq "WR" &&
     $ENV_scacchiera{'b'}->[0]->{'Pezzo'} eq "**" &&
     $ENV_scacchiera{'c'}->[0]->{'Pezzo'} eq "**" &&
     $ENV_scacchiera{'d'}->[0]->{'Pezzo'} eq "**"){  # Torre in posizione e
```

```
                                                    # cammino libero per
                                                    # arrocco lungo
  $x_torre_start="a";$y_torre_start=0;
  $x_torre_end="d";$y_torre_end=0;
  $x_re_end="c";$y_re_end=0;
  $riga="O-O-O";}

 if($ENV_arrocco_LC eq "C" &&
    $ENV_scacchiera{'h'}->[0]->{'Pezzo'} eq "WR" &&
    $ENV_scacchiera{'f'}->[0]->{'Pezzo'} eq "**" &&
    $ENV_scacchiera{'g'}->[0]->{'Pezzo'} eq "**"){  # Torre in posizione e
                                                    # cammino libero per
                                                    # arrocco corto
  $x_torre_start="h";$y_torre_start=0;
  $x_torre_end="f";$y_torre_end=0;
  $x_re_end="g";$y_re_end=0;
  $riga="O-O";}
}
if($ENV_arrocco_WB eq "B" &&
   $ENV_scacchiera{'e'}->[7]->{'Pezzo'} eq "BK"){   # Giocatore nero e
                                                    # posizione re corretta
 $y_re_start=7;
 $img_name_re="BK";$img_name_torre="BR";
 if($ENV_arrocco_LC eq "L" &&
    $ENV_scacchiera{'a'}->[7]->{'Pezzo'} eq "BR" &&
    $ENV_scacchiera{'b'}->[7]->{'Pezzo'} eq "**" &&
    $ENV_scacchiera{'c'}->[7]->{'Pezzo'} eq "**" &&
    $ENV_scacchiera{'d'}->[7]->{'Pezzo'} eq "**"){  # Torre in posizione e
                                                    # cammino libero per
                                                    # arrocco lungo
  $x_torre_start="a";$y_torre_start=7;
  $x_torre_end="d";$y_torre_end=7;
  $x_re_end="c";$y_re_end=7;
  $riga="O-O-O";}

 if($ENV_arrocco_LC eq "C" &&
    $ENV_scacchiera{'h'}->[7]->{'Pezzo'} eq "BR" &&
    $ENV_scacchiera{'f'}->[7]->{'Pezzo'} eq "**" &&
    $ENV_scacchiera{'g'}->[7]->{'Pezzo'} eq "**"){  # Torre in posizione e
                                                    # cammino libero per
                                                    # arrocco corto
  $x_torre_start="h";$y_torre_start=7;
  $x_torre_end="f";$y_torre_end=7;
  $x_re_end="g";$y_re_end=7;
  $riga="O-O";}
}

if($x_torre_start eq ""){return(0);}                # Arrocco non possibile

$re=$ENV_scacchiera{$x_re_start}->[$y_re_start]->
    {'Ptr'};                                        # Casella partenza re
$torre=$ENV_scacchiera{$x_torre_start}->[$y_torre_start]->
       {'Ptr'};                                     # Casella partenza torre
$ENV_scacchiera{$x_re_start}->[$y_re_start]->
     {'Pezzo'}="**";                                # Cancella il re
$re->delete("img");                                 #
$ENV_scacchiera{$x_torre_start}->[$y_torre_start]->
     {'Pezzo'}="**";                                # Cancella la torre
$torre->delete("img");                              #

$re=$ENV_scacchiera{$x_re_end}->[$y_re_end]->{'Ptr'};# Casella arrivo re
$torre=$ENV_scacchiera{$x_torre_end}->[$y_torre_end]->
           {'Ptr'};                                 # Casella arrivo torre
$ENV_scacchiera{$x_re_end}->[$y_re_end]->
```

```
           {'Pezzo'}=$img_name_re;                    # Aggiorna il re
 $img=$win->Photo(-file=>"immagini/$img_name_re.gif");# Ridisegna il re
 $re->createImage(23,23, -image=>$img,-tag=>"img");    #
 $ENV_scacchiera{$x_torre_end}->[$y_torre_end]->
      {'Pezzo'}=$img_name_torre;                    # Aggiorna la torre
 $img=$win->Photo(-file=>"immagini/$img_name_torre.gif");# Ridisegna la torre
 $torre->createImage(23,23, -image=>$img,-tag=>"img");   #

 %tmp_conf=&Freeze;    # Crea una copia dell'attuale configurazione,congelandola
 @stack=%ENV_Stack;                                  # Stato attuale dello stack
 if($ENV_Player eq "W"){                       # Sceglie in quale lista scrivere
  $n_mossa=$mosse_W->index('end')+1;
  push(@stack,$n_mossa."W",\%tmp_conf);
  $mosse_W->insert('end',"$n_mossa $riga");
 }
 else{
  $n_mossa=$mosse_B->index('end')+1;
  push(@stack,$n_mossa."B",\%tmp_conf);
  $mosse_B->insert('end',"$n_mossa $riga");
 }
 %ENV_Stack=@stack;           # Registra l'attuale configurazione nello stack

 &Resume;                     # Elimina eventuali click precedenti
 if($ENV_Player eq "W"){$ENV_Player="B";}# Cambia il turno giocatore
 else{$ENV_Player="W";}                  #
}

############################################################
# Gestisce  le  operazioni  relative alla  casella Mosse  #
# Input = Scrivere una  mossa ('write') | leggere  una    #
# mossa ('read'), riferimento  coordinate  della  casella #
# target, lista mosse a cui si riferisce l'azione (B|W).  #
############################################################
sub Mosse {
local($azione,@coords,$coords,$scelta,$pezzo,$n_mossa,$riga,
      @start_coords,$start_coords,@stack,%tmp_conf,$colonna,
      $casella,%casella,$y,$pezzo,$img,$indice,$colonna,$tmp_mangiata);

 if(@_[0] ne "write" && @_[0] ne "read"){shift @_;}
 ($azione,$start_coords,$coords,$scelta)=@_;

 @coords=@$coords;@start_coords=@$start_coords;

 if($azione eq "write"){           # Scrive mossa eseguita ed aggiorna lo stack
  $pezzo=substr $ENV_scacchiera{@coords[0]}->[@coords[1]-1]->
           {'Pezzo'},1,1;          # Stabilisce il pezzo che ha mosso
  if($pezzo eq "P"){$pezzo="";}    # P non viene scritto
  if($ENV_Mangiata==1){$riga="x";} # Contrassegna una mossa di presa
  $riga.="@coords[0]@coords[1]";

  $tmp_mangiata=$ENV_Mangiata;     # Salva la variabile per i pedoni
  &Resume;                         # Resetta la scacchiera per permettere
                                   # l'analisi dei conflitti
  $ENV_Mangiata=$tmp_mangiata;     # Ripristina variabile per i pedoni

  if($pezzo eq ""){
   $riga=&P_conf(@start_coords[0],@start_coords[1],
   @coords[0],@coords[1]).$riga;}  # Risolve conflitti pedone
  if($pezzo eq "N"){
   $riga=&N_conf(@start_coords[0],@start_coords[1],
   @coords[0],@coords[1]).$riga;   # Risolve conflitti cavallo
  if($pezzo eq "R"){
   $riga=&R_conf(@start_coords[0],@start_coords[1],
   @coords[0],@coords[1]).$riga;}  # Risolve conflitti torre
```

```
 if($pezzo eq "Q" || $pezzo eq "K" || $pezzo eq "B"){
  $riga=$pezzo.$riga;}            # Q, K e B non possono avere conflitti

 if($pezzo eq "" &&(@coords[1]==1||@coords[1]==8)){# Gestione promozioni pedoni
  &Promote;
  if($ENV_Promozione ne ""){      # Promozione ha avuto buon esito
   $ENV_scacchiera{@coords[0]}->[@coords[1]-1]->
     {'Pezzo'}=$ENV_Player.$ENV_Promozione;
   $casella=$ENV_scacchiera{@coords[0]}->[@coords[1]-1]->
     {'Ptr'};                     # Puntatore alla casella
   $casella->delete("img");       # Cancella immagine casella vecchia
   $img=$win->Photo(-file=>
          "immagini/$ENV_Player$ENV_Promozione.gif");# Carica l'immagine nuova
   $casella->createImage(23,23,
          -image=>$img,-tag=>"img");# Posiziona l'immagine nuova nella casella
   $riga.="=$ENV_Promozione";
   $ENV_Promozione="";
  }
 }

 &Resume;                                    # Resetta la scacchiera per un
                                             # corretto salvataggio di
                                             # configurazione
 %tmp_conf=&Freeze;                          # Crea una copia dell'attuale
                                             # configurazione, congelandola
 @stack=%ENV_Stack;                          # Stato attuale dello stack

 if($scelta eq "W"){                         # Sceglie in quale lista scrivere
  $n_mossa=$mosse_W->index('end')+1;
  push(@stack,$n_mossa."W",\%tmp_conf);
  $mosse_W->insert('end',"$n_mossa $riga");
 }
 else{
  $n_mossa=$mosse_B->index('end')+1;
  push(@stack,$n_mossa."B",\%tmp_conf);
  $mosse_B->insert('end',"$n_mossa $riga");
 }
 %ENV_Stack=@stack;                          # Registra l'attuale
                                             # configurazione nello stack
}

if($azione eq "read"){                       # Legge mossa dalla lista e
                                             # aggiorna la grafica con dati
                                             # stack
 &Resume;
 if($scelta eq "W"){                         # Trova l'indice dello stack
                                             # corrispondente alla selezione
  $indice=($mosse_W -> index('active') +1)."-W";
  $ENV_Player="B";                           # Imposta il giocatore che deve
                                             # effettuare la mossa succesiva
 }
 else{
  $indice=($mosse_B -> index('active') +1)."-B";
  $ENV_Player="W";
 }
 $ENV_Change=$indice;                        # Indica un possibile cambiamento
                                             # nella sequenza di mosse
 $indice=~ s/-//;
 $indice=$ENV_Stack{$indice};                # Puntatore alla config. cercata

 %ENV_scacchiera=%$indice;                   # La scacchiera viene caricata
                                             # con la giusta configurazione
 %tmp_conf=&Freeze;                          # Svincola i puntatori
 %ENV_scacchiera=%tmp_conf;                  #
```

```
  foreach $colonna (keys %ENV_scacchiera){      # Controlla tutte le colonne
                                                # della scacchiera
   for($y=0;$y<8;$y++){                         # Controlla caselle della colonna
    $casella=$ENV_scacchiera{$colonna}->[$y]->{'Ptr'};# Puntatore alla casella
    $pezzo=$ENV_scacchiera{$colonna}->[$y]->{'Pezzo'};# Pezzo nella casella
    $casella->delete("img");                    # Cancella immagine vecchia
    if($pezzo ne "**"){                         # Controlla se la casella è vuota
     $img=$win->Photo(-file=>"immagini/$pezzo.gif"); # Carica l'immagine nuova
     $casella->createImage(23,23,
      -image=>$img,-tag=>"img");    # Posiziona l'immagine nuova nella casella
    }
   }
  }
 }
}


### Gestisce le operazioni sulla scacchiera ###
sub Scacchiera {
local($casella,$pezzo_target,$start,$color,$selected,$player,
      $tipo_pezzo_target,@coords,@start_coords,$pezzo_start,
      $casella_start,$img,$indice,@stack,$item,@indice);

 $casella=@_[0];                    # Puntatore all'oggetto casella cliccata

 @coords=split("-",@_[1]);          # Ottiene coordinate separte della casella
 $pezzo_target=$ENV_scacchiera{@coords[0]}->[@coords[1]-1]->
      {'Pezzo'};                    # Cattura l'eventuale pezzo nella casella
 $selected=$ENV_scacchiera{@coords[0]}->[@coords[1]-1]->
      {'Sel'};                      # Valore che indica se la casella era tra le
                                    # selezionate o meno
 $player=substr $pezzo_target,0,1;  # Giocatore legato al pezzo selezionato(W|B)
 $tipo_pezzo_target=substr $pezzo_target,1,1;  # Tipo di pezzo selezionato

 if($ENV_Click==0 && $pezzo_target ne "**" &&
      $player eq $ENV_Player){      # Nessun click precedente,casella non
                                    # vuota,player giusto
  $ENV_Start=@_[1];                         # Memorizza posizione di partenza
  $casella->configure(-relief=>'groove',
                      -bg=>$ENV_eat_col); # Evidenzia la casella selezionata
  if($tipo_pezzo_target eq "P"){&P();}    # Muove pedone
  if($tipo_pezzo_target eq "N"){&N();}    # Muove cavallo
  if($tipo_pezzo_target eq "B"){&B();}    # Muove alfiere
  if($tipo_pezzo_target eq "R"){&R();}    # Muove torre
  if($tipo_pezzo_target eq "Q"){&Q();}    # Muove regina
  if($tipo_pezzo_target eq "K"){&K();}    # Muove re
  $ENV_Click=1;
 }
 else{                                    # Esiste un click precedente

  if($ENV_Click==1 && $pezzo_target ne "**" &&
      $player eq $ENV_Player){            # Cambio di scelta del pezzo da muovere
   @start_coords=split("-",$ENV_Start);   # Ottiene coordinate di partenza
   $start=$ENV_scacchiera{@start_coords[0]}->[@start_coords[1]-1]->
            {'Ptr'};                      # Puntatore casella di partenza
   $color=$ENV_scacchiera{@start_coords[0]}->[@start_coords[1]-1]->
            {'Bg'};                       # Ripristina casella
   $start->configure(-relief=>'solid', -bg=>$color); #
   &Resume;                               # Resetta scacchiera
   &Scacchiera($casella,@_[1]);           # Si richiama a gestire il nuovo pezzo
                                          # selezionato
  }

  else{                                   # Effettua la mossa
```

```
   if($selected){                          # La mossa è effettuata solo sulle
                                           # caselle abilitate
    @start_coords=split("-",$ENV_Start); # Coordinate della casella di partenza
    $pezzo_start=$ENV_scacchiera{@start_coords[0]}->[@start_coords[1]-1]->
      {'Pezzo'};                           # Pezzo che sta muovendo
    $casella_start=$ENV_scacchiera{@start_coords[0]}->[@start_coords[1]-1]->
      {'Ptr'};                             # Puntatore casella di partenza
    $color=$ENV_scacchiera{@start_coords[0]}->[@start_coords[1]-1]->{'Bg'};
    $casella_start->configure(-bg=>$color,
                              -relief=>'solid');# Ripristina l'aspetto iniziale
    $ENV_scacchiera{@start_coords[0]}->[@start_coords[1]-1]->
      {'Pezzo'}="**";                      # Il pezzo che muove viene cancellato
                                           # dalla posizione precedente
    $casella_start->delete("img");         #

    if($pezzo_target ne "**"){             # Il pezzo muove mangiando
     $ENV_Mangiata=1;
     $casella->delete("img");
    }
    $ENV_scacchiera{@coords[0]}->[@coords[1]-1]->
      {'Pezzo'}=$pezzo_start;              # Il pezzo viene messo nella nuova
                                           # casella
    $ENV_scacchiera{@coords[0]}->[@coords[1]-1]->
      {'Mosso'}=1;                         # Segnala che il pezzo ha gia compiuto
                                           # almeno una mossa
    $img=$win->Photo(-file=>
            "immagini/$pezzo_start.gif");# Ridisegna il pezzo spostato
    $casella->createImage(23,23, -image=>$img,-tag=>"img");#

    if($ENV_Change ne ""){                 # La mossa può cambiare la sequenza
                                           # preesistente
     @indice=split("-",$ENV_Change);
     $indice=@indice[0];                   # Indice richiedente
     $player=@indice[1];                   # Listbox richiedente
     $mosse_W->delete($indice,'end');
     if($player eq "W"){
      $mosse_B->delete($indice-1,'end');}# Cancella mosse indesiderate dalle
                                           # listbox
     else{$mosse_B->delete($indice,'end');}

     foreach $item ($mosse_W->get(0,'end')){# Pulisce lo stack
      @indice=split(" ",$item);
      $item=@indice[0]."W";
      push(@stack,$item,$ENV_Stack{$item});
     }
     foreach $item ($mosse_B->get(0,'end')){# Pulisce lo stack
      @indice=split(" ",$item);
      $item=@indice[0]."B";
      push(@stack,$item,$ENV_Stack{$item});
     }
     %ENV_Stack=@stack;
    }
    $ENV_Change="";

    &Mosse("write",\@start_coords,\@coords,$ENV_Player);#Scrive mossa effettuata
    if($ENV_Player eq "W"){$ENV_Player="B";}
    else{$ENV_Player="W";}                 # Cambia il turno giocatore
    &Resume;                               # Resetta scacchiera
   }
  }
 }
}
```

Loader.pm

```
#!/usr/bin/perl

##############################################
# Funzione di inizializzazione strutture dati #
##############################################

sub Init{
local($pgn,@game,$colonna,$y,@riga,$pezzo,$casella,
      $img,%game);

use Chess::PGN::Parse;

### Resetta scacchiera ###
 @riga=("R","N","B","Q","K","B","N","R");
 foreach $colonna (keys %ENV_scacchiera){            # Analizza tutte le caselle
                                                     # per resettarle
  $pezzo=@riga[$ENV_let_num{$colonna}-1];
  $ENV_scacchiera{$colonna}->[0]->{'Pezzo'}="W$pezzo";# Riga pezzi bianchi
  $ENV_scacchiera{$colonna}->[7]->{'Pezzo'}="B$pezzo";# Riga pezzi neri

  for($y=0;$y<8;$y++){
   $ENV_scacchiera{$colonna}->[$y]->{'Sel'}=0;
   $ENV_scacchiera{$colonna}->[$y]->{'Mosso'}=0;
   if($y==2 || $y==3 || $y==4 || $y==5){$ENV_scacchiera{$colonna}->[$y]->
      {'Pezzo'}="**";}                               # Caselle vuote
   if($y==1){$ENV_scacchiera{$colonna}->[$y]->{'Pezzo'}="WP";}# Pedoni bianchi
   if($y==6){$ENV_scacchiera{$colonna}->[$y]->{'Pezzo'}="BP";}# Pedoni neri

   $casella=$ENV_scacchiera{$colonna}->[$y]->{'Ptr'};# Puntatore alla casella
   if($casella->find('withtag',"img")){               # Se la casella ha già
      $casella->delete("img");}                       # un'immagine la cancella
   $pezzo=$ENV_scacchiera{$colonna}->[$y]->{'Pezzo'};# Acquisisce il nuovo pezzo
   if($pezzo ne "**"){                               # Se la casella non è vuota
                                                     # inserisce nuova immagine
    $img=$win->Photo(-file=>"immagini/$pezzo.gif");
    $casella->createImage(23,23, -image=>$img,-tag=>"img");
   }
  }
 }

### Resetta liste mosse e dati partita###
 $mosse_W->delete(0,'end');
 $mosse_B->delete(0,'end');
 $Wplayer->delete(0,'end');
 $Bplayer->delete(0,'end');
 $risultato->delete(0,'end');
 $evento->delete(0,'end');
 $luogo->delete(0,'end');
 $data->delete(0,'end');

### Resetta variabili ambiente Editor e Parser ###
 $ENV_Click=0;
 $ENV_Player="W";
 $ENV_Start="";
 $ENV_Mangiata=0;
 $ENV_Change="";
 %ENV_Stack=();
 $ENV_arrocco_WB="";
 $ENV_arrocco_LC="";
```

```
### Acquisizione file partita ###
 $pgn = new Chess::PGN::Parse @_[0];
 while ($pgn->read_game()) {
  push(@game, "Bianco",$pgn->white,"Nero",$pgn->black,
              "Data",$pgn->date,"Evento",$pgn->event,
              "Round",$pgn->round,"Luogo",$pgn->site,
              "Risultato",$pgn->result,"Mosse",$pgn->smart_parse_game());
 }
 %game=@game;

### Aggiorna dati partita ###
 $Wplayer->insert('end',$game{'Bianco'});
 $Bplayer->insert('end',$game{'Nero'});
 $risultato->insert('end',$game{'Risultato'});
 $evento->insert('end',$game{'Evento'});
 $luogo->insert('end',$game{'Luogo'});
 $data->insert('end',$game{'Data'});

 return(%game);
}

##############################################################################
# Data una mossa di n caratteri la trasforma in un array di n celle. Il token  #
# ottenuto viene invertito in modo da permetterne il parsing bottom-up. In     #
# questa fase vengono trattati i caratteri speciali "+" e "#" e viene definita #
# la destinazione della mossa.                                                 #
##############################################################################


sub Token {
local($mossa,$i,@token,$length,$x,$y);

 $mossa=@_[0];

 push(@token,"P");                          # Preparazione a eventuali mosse pedone
 for($i=0;$i<length($mossa);$i++){
  push(@token,substr $mossa,$i,1);
 }
 $length=scalar @token -1;

 if(@token[$length] eq "#" || @token[$length] eq "+"){
  pop @token;}                              # Elimina simboli scacco e scacco matto
 push (@ENV_destinazione,pop @token,pop @token);# Identifica destinazione pezzo
 @ENV_destinazione=reverse @ENV_destinazione;

 return(@token);
}


#################################################################
# Data una tipologia di pezzo restituisce la lista di tutte   #
# le posizioni che quel tipo di pezzo occupa sulla scacchiera #
#################################################################

sub Find{
local($pezzo,@lista,$colonna,$y);

 $pezzo=@_[0];

 foreach $colonna (keys %ENV_scacchiera){
  for($y=0;$y<8;$y++){
   if($ENV_scacchiera{$colonna}->[$y]->{'Pezzo'} eq $pezzo){
      push(@lista,$colonna.($y+1));}
  }
```

```
 }

 return(@lista);
}


#################
# Funzione main #
#################

sub Loader{
local($filename,%game,@mosse,$input,$mossa,$item,@mossa);

 if(@_[0] eq ""){return(0);}             # Il nome del file deve essere presente

 %game=&Init(@_[0]);                     # Inizializza l'ambiente di lavoro
 $input=$game{'Mosse'};@mosse=@$input;    # Cattura array delle mosse

 foreach $mossa (@mosse){                # Si analizza mossa per mossa
  if($mossa eq "O-O" || $mossa eq "O-O-O"){&O_L($mossa);next;}# Mossa di arrocco

  if($mossa=~ m/.+=.+/){                 # Mossa di promozione
   $mossa=~ s/#|\+//;                    # Elimina caratteri + e #
   @mossa=split("=",$mossa);
   $mossa=@mossa[0];
   $ENV_Promozione=@mossa[1];
  }

  @ENV_token=&Token($mossa);             # Si costruisce il token
  while(@ENV_token){                     #Analizza token carattere per carattere
   $item=pop @ENV_token;
   if($item eq "P"){&Move("P");next;}              # Muove pedone
   if($item eq "N"){&Move("N");next;}              # Muove cavallo
   if($item eq "B"){&Move("B");next;}              # Muove alfiere
   if($item eq "R"){&Move("R");next;}              # Muove torre
   if($item eq "Q"){&Move("Q");next;}              # Muove regina
   if($item eq "K"){&Move("K");next;}              # Muove re
   if($item eq "x"){$ENV_Mangiata=1;next;}         # Mossa di presa
   if($item ne "P" && $item ne "N" &&
      $item ne "B" && $item ne "R" &&
      $item ne "Q" && $item ne "K" &&
      $item ne "x"){                               # Informazione di partenza
    push(@ENV_partenza,$item);
    next;
   }
  }
  @ENV_partenza=();
  @ENV_destinazione=();
  $ENV_Mangiata=0;
 }
}


### Movimentazione dei pezzi ###
sub Move {
local($pezzo,@posizioni,$posizione,$casella,$x,$y,
      $tmp_Mangiata,$item);

 $pezzo=$ENV_Player.@_[0];
 @posizioni=&Find($pezzo);                    # Mappa le posizioni di tutti i
                                              # pezzi del tipo da muovere
 $tmp_Mangiata=$ENV_Mangiata;                 # Salva il valore di Mangiata

 foreach $posizione (@posizioni){             # Cerca il pezzo che deve muovere
```

```
  $x=substr $posizione,0,1; $y=substr $posizione,1,1;
  $casella=$ENV_scacchiera{$x}->[$y-1]->{'Ptr'};# Puntatore alla casella del
                                                 # candidato alla mossa
  $posizione="$x-$y";                           # Posizione viene scritto nel
                                                # formato accettato da Scacchiera
  &Scacchiera($casella,$posizione);

  if($ENV_scacchiera{@ENV_destinazione[0]}->[@ENV_destinazione[1]-1]->
      {'Sel'}){                                 # Il pezzo giusto viene trovato
   if(@ENV_partenza){                           # Esiste un conflitto, controlla
                                                # informazioni aggiuntive
    $item=@ENV_partenza[0];
    if($item=~ m/[a-h]/){                       # L'informazione è una colonna
     if($x eq $item){$ENV_Start="$posizione";last;}# Il candidato risponde alle
                                                # caratteristiche
     else{&Resume;$ENV_Mangiata=$tmp_Mangiata;}
    }
    else{                                       # L'informazione è una riga
     if($y eq $item){$ENV_Start="$posizione";last;}# Il candidato risponde alle
                                                # caratteristiche
     else{&Resume;$ENV_Mangiata=$tmp_Mangiata;}
    }
   }
   else{$ENV_Start="$posizione";last;}          # Nessun conflitto
  }
  else{&Resume;$ENV_Mangiata=$tmp_Mangiata;}    # Deseleziona caselle errate e
                                                # ripristina $ENV_Mangiata
 }

 $ENV_Click=1;
 $posizione="@ENV_destinazione[0]-@ENV_destinazione[1]";
 $casella=$ENV_scacchiera{@ENV_destinazione[0]}->[@ENV_destinazione[1]-1]->
           {'Ptr'};
 &Scacchiera($casella,$posizione);              # Simula un click di spostamento
                                                # sulla scacchiera
 @ENV_token=();                                 # Una volta mosso il pezzo, token
                                                # non serve più
}

### Gestione arrocco ###
sub O_L{

 $ENV_arrocco_WB=$ENV_Player;
 if(@_[0] eq "O-O"){$ENV_arrocco_LC="C";}       # Arrocco corto
 else{$ENV_arrocco_LC="L";}                     # Arrocco lungo
 $arrocco->invoke;

}
```

Salvataggi.pm

```
#!/usr/bin/perl

##############################################
# Funzione salvataggio partita in formato PGN #
##############################################

sub PGN{
local($filename,$dir,$modello,$e,$d,$W,$B,$l,$r,
      @W_list,@B_list,$mossa,$mosse,$contatore,$contamosse);

 use Cwd;
 $filename=@_[0];
```

```
 $dir=cwd();
 ($e,$d,$W,$B,$l,$r)=($evento->get,$data->get,$Wplayer->get,
                      $Bplayer->get,$luogo->get,$risultato->get);

 if($filename eq ""){return(0);}          # Il nome del file non può essere vuoto
 $filename=~ s/(.+)\.(.+)/\1/;            # Elimina estensioni indesiderate

 open(MOD,"< Modello.pgn") or print"Impossibile trovare Modello.pgn";
  while(!eof(MOD)){$modello.=<MOD>;}
 close(MOD);
 $modello=~ s/##evento##/$e/;
 $modello=~ s/##date##/$d/;
 $modello=~ s/##white##/$W/;
 $modello=~ s/##black##/$B/;
 $modello=~ s/##site##/$l/;
 $modello=~ s/##result##/$r/;

 @W_list=$mosse_W->get(0,'end');
 @B_list=$mosse_B->get(0,'end');
 foreach $mossa (@W_list){
  $contatore++;$contamosse++;
  $mossa=~ s/^([1-9] |[1-9][0-9] )(.+)/\2/;     # Elimina indici di indesiderati
  @B_list[$contatore-1]=~ s/^([1-9] |[1-9][0-9] )(.+)/\2/; #
  $mosse.="$contatore. $mossa @B_list[$contatore-1] ";
  if($contamosse==7){$contamosse=0;$mosse.="\n";}    # Massimo di mosse per riga
 }
 $modello=~ s/##mosse##/$mosse/;
 $modello.=$r;

 open(MOD,"> $filename.pgn") or print"Impossibile salvare";# Scrive il file PGN
  print MOD $modello;
 close(MOD);
}


################################################
# Funzione salvataggio partita in formato PDF #
################################################

sub PDF{
local(%pezzi_clear,%pezzi_sfondo,$scacchiera,$contatore,$player,
      $ptr,%scacchiera,$colonna,$riga,$pezzo,$n_mosse);

 if(@_[0] eq ""){return(0);}              # Il nome del file non può essere vuoto
 @_[0]=~ s/(.+)\.(.+)/\1/;                # Elimina estensioni indesiderate

 use PDF::Reuse;

 ($e,$d,$W,$B,$l,$r)=($evento->get,$data->get,$Wplayer->get,
                  $Bplayer->get,$luogo->get,$risultato->get);    # Dati partita

 %pezzi_clear= ('WP'=>"p",'WR'=>"r",'WN'=>"h",'WK'=>"k",
            'WB'=>"b",'WQ'=>"q",'BP'=>"o",'BR'=>"t",
            'BN'=>"j",'BK'=>"l",'BB'=>"n",'BQ'=>"w",
            '**'=>"d");             # Mappa i pezzi senza sfondo sui caratteri
 %pezzi_sfondo=('WP'=>"P",'WR'=>"R",'WN'=>"H",'WK'=>"K",
            'WB'=>"B",'WQ'=>"Q",'BP'=>"O",'BR'=>"T",
            'BN'=>"J",'BK'=>"L",'BB'=>"N",'BQ'=>"W",
            '**'=>"+");             # Mappa i pezzi con sfondo sui caratteri

 ### Corrispondenza numeri-lettere per colonne ###
 %colonne=(0,"a",1,"b",2,"c",3,"d",4,"e",5,"f",6,"g",7,"h ");

 ### Caricamento dati scacchiera ###
```

```
 $scacchiera="[\"tJnWlNjT OoOoOoOo d+d+d+d+ +d+d+d+d d+d+d+d+ +d+d+d+d pPpPpPpP
RhBqKbHr\",";# Configurazione iniziale

 $contatore=1;
 $player="W";
 $sfondo=0;                                  # Decide caselle chiare o con sfondo
 while($contatore){
  $ptr=$ENV_Stack{$contatore.$player};%scacchiera=%$ptr;    # Acquisisce immagine
                                                           # scacchiera
  $scacchiera.="\"";
  for($riga=7;$riga>=0;$riga--){# Viene analizzata ogni casella della scacchiera
   for($colonna=0;$colonna<8;$colonna++){
    $pezzo=$scacchiera{$colonne{$colonna}}->[$riga]->{'Pezzo'};  # Pezzo sulla
                                                                 # casella
    if($sfondo){$scacchiera.="$pezzi_sfondo{$pezzo}";}     # Pezzo con sfondo
    else{$scacchiera.="$pezzi_clear{$pezzo}";}             # Pezzo senza sfondo
    if($sfondo){$sfondo--;}else{$sfondo++;}                # Alterna caselle con
                                                           # sfondo e non
   }
   if($sfondo){$sfondo--;}else{$sfondo++;}
   $scacchiera.=" ";
  }
  chop($scacchiera);
  $scacchiera.="\",";
  $n_mosse++;
  if($player eq "W"){$player="B";}else{$player="W";$contatore++;}
  if(!(defined $ENV_Stack{$contatore.$player})){last;}     # Il ciclo continua
                                                           # su tutto lo stack
 }
 chop($scacchiera);
 $scacchiera.="]";

 prFile(@_[0].".pdf");                          # File di output

 ### Codice JavaScript embedded ###

 open(JS,"< JavaScript.js");
  while(!eof(JS)){$jsCode.=<JS>;}
 close(JS);
 $jsCode=~ s/##scacchiera##/$scacchiera/;
 prJs($jsCode);

 ### Codice JavaScript inizializzazione ###

 my $jsCode = "Compile(\"$B\",\"$W\",\"$r\",\"$e\",\"$l\",\"$d\",$n_mosse);
             Slide('indietro');";

 prInit($jsCode);

 prDocForm('Modello.pdf');                      # Usa il form creato in Acrobat
 prEnd();
}
```

JavaScript.js

```
// Funzione azione pulsante

function Slide (target)
{
 var Scacchiera=##scacchiera##;
```

```
 // Mossa successiva
 if(target == "avanti" &&
    this.getField("indice").value < this.getField("MAX").value)
   {this.getField("indice").value = this.getField("indice").value + 1;}

 // Mossa precedente
 if(target == "indietro" && this.getField("indice").value != 0)
   {this.getField("indice").value = this.getField("indice").value-1;}

 // Mossa di apertura
 if(target == "primo")
   {this.getField("indice").value=0;}

 // Ultima mossa
 if(target == "ultimo")
   {this.getField("indice").value=this.getField("MAX").value;}

 this.getField("scacchiera").value = Scacchiera[this.getField("indice").value];
}

// Funzione di inizializzazione form

function Compile (nome_nero,nome_bianco,risultato,evento,luogo,data,MAX)
{
 this.getField("nero").value = nome_nero;
 this.getField("bianco").value = nome_bianco;
 this.getField("risultato").value = risultato;
 this.getField("evento").value = evento;
 this.getField("luogo").value = luogo;
 this.getField("data").value = data;
 this.getField("indice").value=0;
 this.getField("MAX").value=MAX;
}
```

Chess_Editor.pl

```
#!/usr/bin/perl

require("Eventi.pm");
require("Ambiente.pm");
use Tk;

### Main Window ###

$win= MainWindow->new(-width=>615, -height=>530,
                      -title=>"CHESS EDITOR v 1.0");  # Crea finestra principale
$win->resizable(0,0);

### Sezione TAG principali ###

$win->Label(-text=>"Giocatore bianco")->
      place(-x=>7,-y=>10);               # Testo esplicativo casella input testo
$Wplayer= $win->Entry()->place(-x=>10,-y=>30); # Casella input giocatore bianco
$Wplayer->insert('end','?');

$win->Label(-text=>"Giocatore nero")->place(-x=>157,-y=>10);
$Bplayer= $win->Entry()->place(-x=>160,-y=>30);
$Bplayer->insert('end','?');

$win->Label(-text=>"Risultato")->place(-x=>307,-y=>10);
$risultato= $win->Entry()->place(-x=>310,-y=>30);
```

```
$risultato->insert('end','?');

$win->Label(-text=>"Evento")->place(-x=>7,-y=>50);
$evento= $win->Entry()->place(-x=>10,-y=>70);
$evento->insert('end','?');

$win->Label(-text=>"Luogo")->place(-x=>157,-y=>50);
$luogo= $win->Entry()->place(-x=>160,-y=>70);
$luogo->insert('end','?');

$win->Label(-text=>"Data")->place(-x=>307,-y=>50);
$data= $win->Entry()->place(-x=>310,-y=>70);
$data->insert('end','?');

$win->Label(-text=>"Bianco")->place(-x=>457,-y=>10); # Casella di riepilogo
                                                     # mosse bianco
$mosse_W=$win->Scrolled('Listbox', -scrollbars=>'oe',
                        -height=>27,-width=>8,
                        -selectmode=>'single')->place(-x=>460,-y=>30);
$mosse_W->bind('<Double-Button-1>' => [\&Mosse,"read",\@null,\@null,"W"]);

$win->Label(-text=>"Nero")->place(-x=>538,-y=>10);   # Casella di riepilogo
                                                     # mosse nero
$mosse_B=$win->Scrolled('Listbox', -scrollbars=>'oe',
                        -height=>27,-width=>8,
                        -selectmode=>'single')->place(-x=>540,-y=>30);
$mosse_B->bind('<Double-Button-1>' => [\&Mosse,"read",\@null,\@null,"B"]);

### Scacchiera ###

my $color=$ENV_color1;

for($i=0;$i<8;$i++){
 for($j=7;$j>=0;$j--){

  my $tag=$ENV_colonne{$i}."-".($j+1);               # Etichetta ogni casella
                                                     # con le coordinate
  my $casella=$win->Canvas(-width=>40, -height=>40,  # Crea una singola
                           -borderwidth=>'1',        # casella della scacchiera
                           -relief=>'solid', -bg=>$color)->      #
                   place(-x=>50+(44*$i),-y=>108+(308-(44*$j))); #
  $ENV_scacchiera{$ENV_colonne{$i}}->[$j]->
      {'Bg'}=$color;                                 # Salva colore di fondo
  $ENV_scacchiera{$ENV_colonne{$i}}->[$j]->
      {'Ptr'}=$casella;                              # Salva puntatore a casella
  $casella->Tk::bind('<Button-1>' =>
            [\&Scacchiera,$tag,$casella]);           # Mappa l'azione di click
                                                     # sulla casella
  my $img_name=$ENV_scacchiera{$ENV_colonne{$i}}->[$j]->
                 {'Pezzo'};                          # Trova il pezzo da
                                                     # inserire sulla scacchiera
  if($img_name ne "**"){                             # Verifica se casella vuota
   my $img=$win->Photo(-file=>"immagini/$img_name.gif");   # Carica l'immagine
   $casella->createImage(23,23,
             -image=>$img,-tag=>"img");              # Posiziona l'immagine
                                                     # nella casella
  }

  if($color eq $ENV_color2){$color=$ENV_color1;}
  else{$color=$ENV_color2;}                    # Alterna i colori delle caselle
  $win->Label(-text=>$j+1)->place(-x=>35,-y=>123+(308-(44*$j)));
 }
 if($color eq $ENV_color2){$color=$ENV_color1;}else{$color=$ENV_color2;}
 $win->Label(-text=>$ENV_num_let{$i+1})->place(-x=>66+(44*$i),-y=>460);
```

```
}

### Sezione tasti ###

$win->Button(-text=>"Resetta", -command=> \&Init,
             -width=>5)->place(-x=>462, -y=>490);
$win->Button(-text=>"Salva", -command=> \&Salva,
             -width=>5)->place(-x=>514, -y=>490);
$win->Button(-text=>"Carica", -command=> \&Load,
             -width=>5)->place(-x=>566, -y=>490);
$arrocco=$win->Button(-text=>"Arrocco",
                      -command=> \&Arrocco)->place(-x=>50, -y=>490);
$arrocco_W=$win->Radiobutton(-variable=>\$ENV_arrocco_WB,-value=>"W",
                             -text=>"Bianco")->place(-x=>120, -y=>480);
$arrocco_B=$win->Radiobutton(-variable=>\$ENV_arrocco_WB, -value=>"B",
                             -text=>"Nero")->place(-x=>120, -y=>500);
$arrocco_L=$win->Radiobutton(-variable=>\$ENV_arrocco_LC, -value=>"L",
                             -text=>"Lungo (O-O-O)")->place(-x=>190, -y=>480);
$arrocco_C=$win->Radiobutton(-variable=>\$ENV_arrocco_LC, -value=>"C",
                             -text=>"Corto (O-O)")->place(-x=>190, -y=>500);

### Icona finestra principale ###
$win->update;
my $icon = $win->Photo(-file => 'immagini/icon.gif');
$win->iconimage($icon);

MainLoop;
```

I file di codice descritti in questa appendice sono inclusi nel cd-rom allegato alla tesi, nella cartella Pdf.

Bibliografia

- Ordine degli ingegneri di Bergamo, “Storia dell’ipertesto”, 2001, http://www.ordineingegneri.bergamo.it/Commissioni/2001/14/Documenti/moduli/Lezione04/ipert.htm.

- [XAN] Nelson T., “Project Xanadu”, 2003, http://www.xanadu.com.

- H'obbes' Z. R., “Hobbes' Internet Timeline v7.0”, 2004, http://www.zakon.org/robert/internet/timeline.

- Ciancarini P., “La produzione di documenti digitali”, 2003, http://www.cs.unibo.it/~cianca/wwwpages/dd/ps-pdf.pdf.

- Magnik J., “Typography & Page Layout”, 1997, http://www.typography-1st.com/typo/typeface.shtml.

- Staff Microsoft, “ClearType information”, 2004, http://www.microsoft.com/typography/ClearTypeInfo.mspx.

- Leurs L., “The history of PDF”, 2003, http://www.prepressure.com/pdf/history/history01.htm.

- Autore sconosciuto, “Digital Asset Server”, 2004, http://www.evolutionbook.com/Mod_02.php?data_dir=Tech&data_file=speciale-tecnologia&data=2004-07-12&numero=00.

- Autore sconosciuto, “Un po’ di storia di HTML e CSS”, 2004, http://www.extrowebsite.com/lezioniCSS.asp?ID=1.

- Wilson B., “HTML Overview”, 2003, http://www.eskimo.com/~bloo/indexdot/history/html.htm.

- Staff Adobe, “What is SVG?”, 2004, http://www.adobe.com/svg/overview/svg.html.

- [XSL] W3C, “The Extensible Stylesheet Language Family (XSL)”, 2004, http://www.w3.org/style/xsl.

- Watkins P., “Validating moves and commentary using XSLT ”, 2004, http://www.interfootball.co.uk/chess/live/chess.htm.

- Melander A., Bulsink B., Pivovarov J., “PGNtoJS”, 2002, http://www.mailchess.de/pgntojse.html.

- [PGN1] Autori vari, “Portable Game Notation Specification and Implementation Guide”, 1994, http://www.very-best.de/pgn-spec.htm.

- [PGN2] Maxia G., “Chess-PGN-Parse-0.18”, 2002, http://search.cpan.org/~gmax/Chess-Pgn-Parse-0.18/Parse.pm.

- Autori vari, “Esempi di file PGN”, 2004, http://www.deltafarms.com/chess/pgn.php3.

- [LIT] Staff Microsoft Reader, “Microsoft Reader homepage”, 2001, http://www.microsoft.com/reader.

- [OeB] Autori vari, “Required Rights Features for Digital Books”, 2003, http://oebf.org/specifications/coodinatedfiles/OeBF%20Rights%20Grammar%20Requirements.htm.

- [ORW] Staff Overdrive, “ReaderWorks homepage”, 2004, http://www.overdrive.com/readerworks.

- [DOB] Walsh N., “DocBook.org”, 2004, http://www.docbook.org.

- Galassi M., “DocBook intro”, 1998, http://nis-www.lanl.gov/~rosalia/mydocs/docbook-intro/docbook-intro.html.

- [MSW] Dubois J., “Win32-OLE-0.17”, 2003, http://search.cpan.org/~jdb/Win32-OLE-0.17/lib/OLE.pm.

- [PTK] Siever E., Spainhour S., Patwardan N., “Perl/Tk”, in: *PERL Guida di riferimento*, O’Reilly, 2000, capitolo 18.

- Autori vari, “Argomenti CPAN relativi a Perl/Tk”, http://search.cpan.org/search?query=Tk&mode=all.

- Autori vari, “Forum di discussione sull’utilizzo di Perl/Tk”, http://www.perltk.org.

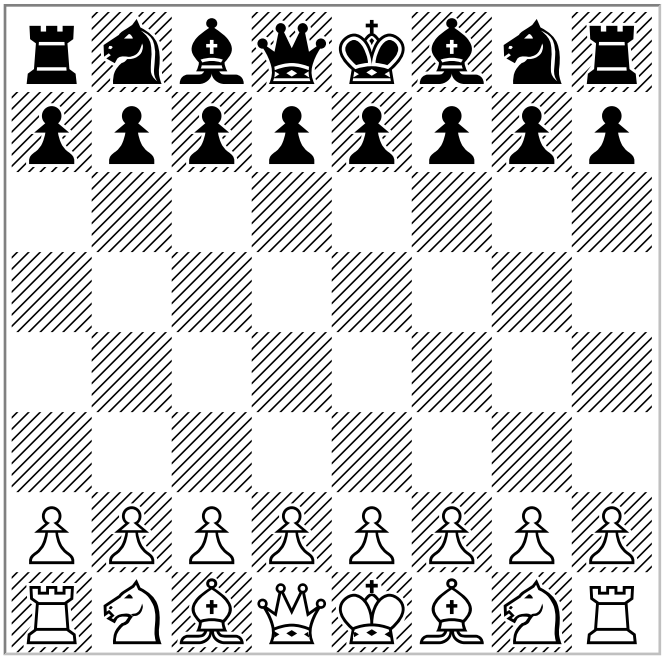
- [PDF1] Heinz S., “Home page di FreePDF v2.02”, 2004, http://shbox.de.

- Lundberg L., “PDF-Reuse-0.29”, 2003, http://search.cpan.org/~larslund/ PDF-Reuse-0.29/Reuse.pm.

- [PDF2] Lundberg L., “PDF-Reuse-0.29”, 2004, http://search.cpan.org/~larslund/ PDF-Reuse-0.29/Reuse.pm.

- Autori vari, “PlanetPDF”, 2004, http://www.planetpdf.com.

Bianco:	Nero:	Risultato:
Grigor Minchev	Nejad	1-0
Evento:	Luogo:	Data:
Open	Modena	1996.??.??



|< < > >|