Laboratorio Metodi di Programmazione

A.A. 2003-2004

Specifiche di Progetto

Stefano Zacchiroli
<zack@cs.unibo.it>

18 aprile 2004

1 Introduzione

Scopo del progetto è la realizzazione di un giocatore artificiale di scacchi, ovvero di un programma che sia in grado di giocare a scacchi contro un avversario (umano o meno).[1]

2 Regole del gioco

In questa sezione non si parla delle regole del gioco degli scacchi[2], ma delle regole di partecipazione al progetto.

Fasi Il progetto è diviso in due fasi: *design* (sez. 3) ed *implementazione* (sez. 4). Nella prima fase dovrete effettuare il design delle classi e delle interfacce, nella seconda dovrete implementare (parte di) tali classi ed interfacce. La consegna è unica. Potete quindi decidere voi quale e quanto tempo dedicare ad ogni fase. Vi consiglio però di svolgere prima la fase di design e successivamente la fase implementativa.

Gruppi Il progetto deve essere svolto in gruppi. Ogni gruppo deve essere composto da 3 persone. In casi eccezionali, richiedenti mia previa autorizzazione, si accettano deroghe a questa regola in difetto (gruppi composti da meno di 3 persone), ma non in eccesso (gruppi composti da più di 3 persone). Data la difficoltà del progetto è comunque fortemente consigliato che i gruppi siano composti da 3 persone.

Devo proprio? Alcune parti (fasi o sottofasi) del progetto sono obbligatorie, altre facoltative.

Valutazione Il progetto verrà complessivamente valutato con un punteggio variabile *da 0 a 30*. Tale punteggio varrà $\frac{1}{3}$ del voto finale dell'esame integrato *Programmazione - Laboratorio Metodi di Programmazione*. Ad un progetto nel quale non sia stata svolta nessuna delle parti facoltative non è attribuibile un punteggio superiore a 26/30. Ad un progetto nel quale non siano state svolte tutte le parti obbligatorie non potrà essere attribuito un punteggio superiore a 17/30.

La valutazione consisterà di due parti: una mia revisione del materiale consegnato ed una discussione tra me ed i membri del gruppo che lo hanno realizzato. La discussione verterà sulla *presentazione del materiale* realizzato, sugli *strumenti di sviluppo* utilizzati e sulle tecniche di implementazione di un *giocatore artificiale*. Non necessariamente la valutazione sarà uniforme tra i componenti di uno stesso gruppo.

A progetti che, a mio insindacabile giudizio, risulteranno *copiati* da altri verrà attribuito un punteggio

[1] La struttura di tali programmi è stata descritta nella lezione "Anatomia comparata di un giocatore artificiale di scacchi" tenutasi Venerdì 2 aprile 2004, non verrà pertanto riportata in questo documento.

[2] per quelle potete dare una occhiata a [3]

pari a 0. Tale informazione verrà inoltre inoltrata al docente titolare del corso che valuterà la possibilità di ulteriori provvedimenti disciplinari. Notate che è molto semplice risalire a quale sia la sorgente della copia e quali siano le destinazioni.

Inizio del progetto e modalità consegna Il progetto ha inizio con la pubblicazione di queste specifiche su web. Tale pubblicazione avverrà presumibilmente in data 5 aprile 2004. Il progetto dovrà essere consegnato *entro e non oltre le 23:59:59 del 6 giugno 2004* (fa fede la data delle macchine del cluster cs). La consegna dovrà essere effettuata *via mail* all'indirizzo `zack@cs.unibo.it`. Ogni mail dovrà avere oggetto `[labprog0304] consegna progetto` e dovrà contenere nelle prime righe i nomi completi dei membri del gruppo, uno per riga. Alla mail dovrà essere allegata una *tarball* contenente un'unica directory, tale directory dovrà contenere (anche ricorsivamente):

- un **file** `README` che descriva chi sono i membri del gruppo che ha svolto il progetto, quali parti del progetto sono state svolte e quale sia la classe principale da invocare per interfacciare il progetto ad XBoard[3] (vedi sez. 4.9)
- il **diagramma delle classi** risultante dalla fase di design in un qualche formato grafico *finale* (es. `pdf`, `png`, `gif`, `ps` vanno bene; corel draw, open office, powerpoint non vanno bene)
- i **sorgenti Java** del progetto risultanti dalla fase di implementazione (una collezione di file `.java`)
- un **Makefile** che presenti almeno il target `all`. L'invocazione di tale target (`make all`) dovrà compilare i sorgenti Java in corrispondenti file `.class` (per chi volesse usare Apache ANT[4] il Makefile deve richiamare ant quando invocato con il target `all`)

Una volta compilato, il progetto dovrà essere eseguibile invocando il comando `java NomeClasse.java` dove *NomeClasse* è il nome della classe principale dell'implementazione come riportato nel file `README`.

Saranno penalizzati i progetti che non verranno consegnati secondo le modalità sopra descritte.

3 Fase α – Design

In questa sezione viene descritta la prima fase del progetto: il *design*.

L'implementazione di un programma utilizzando un linguaggio di programmazione ad oggetti è solitamente preceduta da una fase di sviluppo chiamata *object oriented design*. In tale fase si delineano quali siano le classi necessarie per raggiungere lo scopo che il programma si prefigge, quali siano i loro "contratti" e quali relazioni esistano tra di esse. Nel caso particolare di Java è necessario stabilire come minimo:

- quali classi e interfacce si vogliano realizzare
- quali classi siano astratte
- quali metodi e attributi siano presenti per ogni classe/interfaccia e quali siano i loro "contratti" (tipo dei parametri, tipo del valore di ritorno, altre assunzioni)
- quali relazioni di ereditarietà (`extends`) esistano
- quali relazioni di implementazione (`implements`) esistano

La maggior parte di tali caratteristiche sono rappresentabili mediante un *diagramma delle classi*.

Tanto meglio svolgerete questa fase, tanto più semplice sarà la fase di implementazione e tanto più facilmente riuscirete ad adattare il codice che scriverete nella fase di implementazione ai problemi che via via incontrerete.

In questa fase del progetto vi viene chiesto di effettuare tale design. Il risultato finale fase deve essere un diagramma delle classi che rappresenti le classi e le interfacce che ritenete necessarie per implementare un giocatore artificiale di scacchi. Per ognuna di esse devono essere rappresentati metodi e attributi. Per ogni metodo devono essere riportate la caratteristiche di accessibilità (`public`, `protected`, `private`,

[3] nel caso vogliate racchiudere tutte le classi in un unico *package* specificate anche il nome di tale package

[4] `http://ant.apache.org`

`package`), nome e tipo degli argomenti, altre caratteristiche del metodo (ad esempio `static` o `final`) e tipo di ritorno. Informazioni analoghe devono essere rappresentate per gli attributi. Potete omettere di riportare metodi e attributi ereditati.

Dovete inoltre rappresentare nel diagramma le relazioni di ereditarietà tra classi o interfacce (tipicamente con una freccia che va da una classe/interfaccia figlio ad una classe/interfaccia padre) e di implementazione (tipicamente con una freccia che va da una classe verso le interfacce che essa implementa).

Non è necessario che nella successiva fase implementativa (fase β) implementiate tutto ciò che avete riportato nel diagramma delle classi. È consigliabile quindi effettuare in questa fase il design delle parti facoltative: in questo modo se avrete tempo di implementarle potrete farlo senza dovere rimettere mano al design iniziale.

Per effettuare il design potete usare gli strumenti che preferite (anche matita e gomma, digitalizzando il disegno risultante). Alcuni software vi possono però aiutare in questo compito, ad esempio `dia`[5] (utilizzando la palette "UML") e `argouml`.[6]

Questa fase è **obbligatoria** e sarà oggetto di valutazione.

In data 21 aprile 2004 si terrà una **lezione di discussione** su questa fase del progetto.

4 Fase β – Implementazione

In questa sezione viene descritta la seconda fase del progetto: l'*implementazione*.

Abbiamo visto a lezione che un giocatore artificiale di scacchi è diviso in componenti relativamente indipendenti tra loro. La fase di implementazione del progetto segue tale divisione. In questa sezione sono descritte singolarmente le componenti di un giocatore artificiale di scacchi che vi è chiesto di implementare. Per ognuna di esse è indicato se si tratti di una parte obbligatoria o facoltativa.

Notate che non è assunto un mapping 1-1 tra componenti e classi: una componente può essere implementate da più classi ed una classe può implementare più componenti, spetta a voi la decisione.

Analogamente, nel caso in cui siano riportate quali operazioni devono essere effettuabili con una data componente, non è assunto un mapping 1-1 tra operazioni e metodi. Una operazione può essere implementata da più metodi e un singolo metodo può implementare più operazioni, spetta a voi la decisione.

Durante la fase di implementazione ricordate che il nostro obiettivo *non* è quello di realizzare un giocatore artificiale di scacchi competitivo. Non sarete quindi premiati nella valutazione per implementazioni particolarmente efficienti, ma piuttosto per implementazioni "eleganti" che seguano fedelmente il design della fase precedente. Parimenti non sarete premiati per la bravura del vostro giocatore artificiale.

In data 3 maggio 2004 si terrà una **lezione di discussione** su questa fase del progetto.

4.1 Rappresentazione della scacchiera

Questa componente è il cuore di ogni giocatore artificiale di scacchi, il suo scopo è quello di mantenere lo stato della scacchiera ovvero la posizione di tutti i pezzi ancora in gioco. La rappresentazione scelta deve permettere di effettuare come minimo le seguenti operazioni:

- accesso ad una cella della scacchiera in lettura/scrittura (i.e. "dammi pezzo alla data cella", "metti pezzo alla data cella")
- movimento (i.e. "muovi pezzo da ... a ...")
- setup iniziale (i.e. "(ri)porta la scacchiera alla posizione iniziale prevista dal gioco degli scacchi")

La rappresentazione della scacchiera è una **componente obbligatoria**.

4.2 Tabella delle trasposizioni

La tabella della trasposizioni è una tabella che può essere utilizzata per memorizzare informazioni relative

[5] `http://www.lysator.liu.se/~alla/dia/`
[6] `http://argouml.tigris.org/`

ad una data configurazione della scacchiera. Per realizzarla è sufficiente implementare una funzione hash sulla rappresentazione della scacchiera ed utilizzare alcune classi già disponibili nella libreria standard di Java.

La tabella delle trasposizioni è una **componente facoltativa**.

4.3 Generatore di mosse

Il generatore di mosse è la componente che, data una particolare configurazione della scacchiera ed il turno corrente (i.e. a chi tocca muovere), restituisce un insieme di *mosse valide*. Esistono molti criteri per valutare quali mosse siano valide e quali non lo siano, riporto alcuni di essi:

1. una mossa è valida se rispetta le regole di movimento del pezzo che si sta muovendo
2. una mossa è valida se il pezzo mosso non viene mosso fuori dalla scacchiera
3. una mossa è valida se la casella di destinazione non contiene un pezzo amico
4. una mossa è valida se la posizione risultante non permette la cattura del re di chi ha effettuato la mossa

È **obbligatorio** che implementiate i criteri da 1 a 4, è invece **facoltativa** l'implementazione di altri criteri non riportati nella lista di cui sopra.

4.4 Regole particolari

L'implementazione delle seguenti regole del gioco degli scacchi è **facoltativa**:

- arrocco
- promozione
- presa *en passant*

4.5 Funzione di valutazione

La funzione di valutazione associa ad una particolare configurazione della scacchiera un valore positivo nel caso in cui si ritenga che il bianco sia in vantaggio, negativo in caso contrario e pari a 0 in caso si ritenga di trovarsi in una situazione di pareggio.

Una funzione di valutazione viene solitamente implementata combinando tra loro i risultati di più *funzioni di valutazione elementari*. Alcune funzioni di valutazione elementari sono le seguenti:

1. *valutazione del materiale*. Per la valutazione del materiale viene assegnato un punteggio ad ogni pezzo ad eccezione del re. Valori tipici (ma non vincolanti per voi) sono i seguenti:

pedone	100
cavallo	300
alfiere	325
torre	500
regina	900

   La valutazione consiste nel sommare i valori associato ai pezzi bianchi ancora in gioco e sottrarre al valore così ottenuto i valori associati ai pezzi neri ancora in gioco.

2. *fattore di mobilità*. Questa valutazione consiste nel sottrarre al numero di mosse legali del bianco il numero di mosse legali del nero.

3. *controllo dello spazio statico*. Per effettuare questa valutazione viene associata ad ogni cella della scacchiera un valore, tanto più è alto tale valore, tanto più quella cella è importante per uno dei due giocatori. Alcuni valori di esempio (ma non vincolanti per voi) dal punto di vista del bianco sono i seguenti:

9	10	11	12	12	11	10	9
9	12	13	14	14	13	12	9
9	12	14	15	15	14	12	9
9	12	14	15	15	14	12	9
8	11	13	14	14	13	11	9
6	9	11	12	12	11	9	6
4	7	8	9	9	8	7	4
2	3	4	5	5	4	3	2

La valutazione si ottiene sommando i valori delle celle attualmente tenute sotto scacco da pezzi bianchi e sottraendo al valore così ottenuto l'analogo calcolo effettuato per le celle tenute sotto scacco da pezzi neri (utilizzando una tabella simmetrica sull'asse orizzontale rispetto a quella riportato).

4. *controllo dello spazio dinamico.* È una valutazione simile alla precedente, ma associa valori alle celle della scacchiera "dinamicamente" in base alla posizione del re nemico, trovate valori di esempio su [1].

5. *valore di sicurezza.* Vedi [1] o [2]

6. *sviluppo dei pezzi.* Vedi [1] o [2]

7. *struttura pedonale.* Vedi [1] o [2]

È **obbligatoria** l'implementazione di una funzione di valutazione che consideri almeno le funzioni elementari 1 e 2 (materiale e mobilità). È lasciata a voi la scelta dei pesi da assegnare ai valori restituiti dalle funzioni di valutazione elementari. È **facoltativa** l'implementazione di una funzione di valutazione che consideri anche altre funzioni elementari di valutazione.

4.6 Algoritmo di ricerca: α-β pruning

È **obbligatoria** l'implementazione dell'algoritmo di ricerca α-β pruning[4] visto a lezione. La scelta della profondità è a vostra discrezione, ma non deve essere inferiore a 2.

4.7 Libro di aperture

Per risparmiare tempo di calcolo nelle posizioni iniziali del gioco, è prassi comune mantenere una tabella delle trasposizioni di suggerimenti associati a tali posizioni. Tali suggerimenti sono solitamente raccolti in libri di aperture, ne potete trovare moltissimi su web, ad esempio [6].

Per implementare il libro delle aperture è sufficiente popolare all'avvio del vostro giocatore artificiale la tabella dei suggerimenti e modificare l'algoritmo di ricerca in modo che, prima di procedere nella ricerca, controlli se esiste un suggerimento per la posizione corrente. Nel caso esista viene immediatamente restituita la mossa consigliata senza procedere nella ricerca.

Il libro di aperture è una **componente facoltativa** e richiede l'implementazione della tabella delle trasposizioni.

4.8 Iterative deepening

Abbiamo visto a lezione che l'efficienza di α-β dipende dall'ordine nel quale le mosse vengono considerate. Abbiamo visto inoltre che l'algoritmo chiamato *Iterative deepening* fornisce una buona euristica su quale sia l'ordinamento ottimale sulle mosse da considerare.

L'implementazione di iterative deepening è una **componente facoltativa**.

4.9 Interfaccia "utente"

Per quanto riguarda l'interfaccia utente, non vi sarà richiesto di implementarne una da zero. Dovrete piuttosto interfacciarvi ad XBoard[7], una delle interfacce grafiche utilizzabili per giocare contro il programma GNU Chess.[8].

XBoard fornisce solamente una interfaccia grafica per il gioco degli scacchi ed utilizza come motore di gioco un programma ausiliario. Tale programma è di default GNU Chess, ma può essere specificato a linea di comando con il parametro `-fcp` (i.e. First Chess Program).

Il vostro obiettivo è quello di scrivere un giocatore artificiale in modo che per giocare contro di lui sia sufficiente invocare:

```
xboard -fcp 'java NomeClasse'
```

dove *NomeClasse* è il nome della classe principale da voi implementata.

Affinché il vostro giocatore possa "dialogare" con XBoard è necessario che voi implementiate un protocollo chiamato *Chess Engine Communication Protocol*[7] .Per il vostro semplice gioco di scacchi non

[7] `http://www.tim-mann.org/xboard.html`
[8] `http://www.tim-mann.org/gnuchess.html`

è necessario implementarlo interamente, è sufficiente che implementiate alcuni comandi fondamentali.

Il protocollo prevede che la comunicazione tra XBoard e il motore di gioco avvenga via *standard input* e *standard output*: XBoard invierà al vostro giocatore artificiale comandi che potrete leggere da standard input; il vostro giocatore artificiale dovrà rispondere a XBoard scrivendo su standard output. Nello specifico di Java, per leggere da standard input dovrete utilizzare lo stream di input `System.in`, per scrivere su standard output dovrete utilizzare lo stream di stampa `System.out`.

Il ciclo principale del vostro giocatore artificiale sarà quindi simile allo pseudo codice che segue:

```
while (true) {
  comando1 = leggi_comando_da_System_in();
  comando2 = processa_comando(comando1);
  scrivi_comando_su_System_out(comando2);
}
```

Ogni comando è su una riga a se stante e termina con il carattere di fine riga.

I comandi *interessanti* che potete ricevere da XBoard e i relativi comandi di risposta che dovrete inviare sono i seguenti:

- `protover 2`

  È il primo comando che XBoard vi invierà.

  XBoard invia questo comando per verificare che il motore di gioco conosca la versione 2 del protocollo di comunicazione. Dovrete rispondere a questo comando con il *comando di inizializzazione*, indicando quali caratteristiche del protocollo di comunicazione volete utilizzare.

  Il comando di inizializzazione che vi consiglio è il seguente: `feature done=0 ping=0 usermove=1 time=0 draw=0 reuse=0 analyze=0 myname=Nome variants=normal colors=0 sigint=0 sigterm=0 done=1` (tutto su una unica riga). Al posto di *Nome* potete riportare il nome del vostro giocatore artificiale.

  Ho scelto questo comando di inizializzazione in modo da semplificare al massimo la comunicazione con XBoard. Siete liberi di cambiarlo, ma questo potrebbe richiedere che voi implementiate parti aggiuntive del protocollo. Per maggiori dettagli vi rimando a [7].

- `usermove <mossa>`

  È il comando che XBoard invia per segnalarvi che l'utente (vostro avversario) ha effettuato una mossa. `<mossa>` è una stringa che descrive la mossa effettuata in formato *notazione algebrica coordinate*. In tale formato le mosse sono descritte come segue:

  mosse "normali" : una lettera per la colonna di partenza (da 'a' colonna di sinistra, a 'h' colonna di destra), una cifra per la riga di partenza (da 1 riga più in basso, a 8 riga più in alto), una lettera per la colonna di arrivo, una lettera per la riga di arrivo.
  Esempio: e2e4

  promozione : come sopra, ma con una lettera aggiuntiva che indica a quale pezzo si vuole promuovere il pedone.
  Esempio: e7e8q

  arrocco : come per le mosse normali, indicando la mossa del re.
  Esempio: e1g1

  Dovete rispondere ad XBoard indicando quale sia la vostra mossa utilizzando il comando `move <mossa>` (notate la differenza tra "move" e "usermove"). Dove `<mossa>` è la mossa scelta dal vostro giocatore artificiale, espressa nel formato precedente. Nel caso in cui il vostro giocatore artificiale voglia arrendersi, potete rispondere al comando `usermove` con il comando `resign`.

  XBoard controlla per voi che l'utente non generi mosse illegali, pertanto non riceverete mai un comando `usermove` corrispondente ad una mossa illegale. Tuttavia alcune mosse che per XBoard sono legali, potrebbero non esserlo per il vostro giocatore artificiale. In particolare ciò può accadere se decidete di non implementare una o più delle regole particolari descritte in sez. 4.4. In questo caso dovete segnalare ad XBoard che la mossa ricevuta non è valida. Per effettuare tale

segnalazione è sufficiente rispondere con il comando `Illegal move (<motivo>): <mossa>`. Al posto di `<motivo>` dovete indicare la ragione per la quale la mossa non è valida (una stringa di vostra scelta che verrà comunicata all'utente) e al posto di `<mossa>` dovete riportare la mossa illegale ricevuta.

- `quit`

  XBoard vi invierà questo comando quando l'utente chiede di terminare la sessione di gioco. Non dovete rispondere a questo comando, dovete semplicemente terminare l'esecuzione del vostro giocatore artificiale

Nel caso abbiate problemi di comunicazione con XBoard vi consiglio di eseguirlo aggiungendo il parametro a riga di comando `-debug`. Aggiungendo tale parametro XBoard riporta durante l'esecuzione tutti i comandi scambiati tra lui ed il motore di gioco.

5 Consigli conclusivi

Il progetto di quest'anno non è semplice, vi consiglio quindi di:

- *collaborare tra gruppi*, sia di persona che discutendo sul newsgroup `unibo.cs.labprogrammazione`. Io seguirò il newsgroup e parteciperò alle discussioni per quanto mi sarà possibile.
  N.B. collaborare $\neq$ copiare

- *partecipare alle lezioni di discussione*. Tali lezioni non saranno preorganizzate da me, ma saranno occasioni di discussione tra voi su scelte implementative e di design con la mia supervisione. Partecipate quindi *preparati* avendo già una idea delle problematiche delle quali si discuterà, altrimenti saranno lezioni inutili

- *non reinventare la ruota*. La libreria standard di Java[8] è uno strumento molto potente in quanto contiene molte classi che possono esservi utili nell'implementazione del progetto. Ogni volta che vi si pone un problema implementativo quindi, prima di cominciare ad implementare una soluzione a testa bassa, controllate che non esistano già una o più classi che risolvano problemi simili nella libreria standard di Java. In particolare vi consiglio di dare un'occhiata ai seguenti argomenti:

  – *iteratori*
    interfaccia `java.util.Iterator`
  – *metodi della classe* `java.lang.Object`, in particolare: `equals`, `hashCode`, `toString`. Ridefinire tali metodi vi potrebbe essere molto utile
  – *collezioni*
    interfacce `java.util.Collection`, `java.util.Set`, `java.util.List` e implementazioni corrispondenti
  – *reader bufferizzati*
    classi `java.io.BufferedReader` e `java.io.InputStreamReader`, vi potrebbero tornare molto utili per l'implementazione del protocollo di comunicazione

In bocca al lupo!

—*il lupo*

Riferimenti bibliografici

[1] Paolo Ciancarini, I giocatori artificiali, Mursia, Milano 1992

[2] Francois Dominic Laramee, Chess Programming,
http://www.gamedev.net/reference/programming/features/chess1/

[3] Piccolo manuale degli scacchi,
http://scacchi.qnet.it/manuale/scacchi.htm

[4] WikipediA: Alpha-beta pruning
http://en.wikipedia.org/wiki/Alpha-beta_pruning

[5] Laboratorio Metodi di Programmazione A.A. 2003-2004
http://www.cs.unibo.it/~zacchiro/courses/labprog0304/

[6] Piccolo catalogo di aperture
http://scacchi.qnet.it/manuale/aperture.htm

[7] Tim Mann, Chess Engine Communication Protocol,
http://www.tim-mann.org/xboard/engine-intf.html

[8] Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification
http://java.sun.com/j2se/1.4.2/docs/api/index.html