Costruiamo un giocatore artificiale di Othello

Come si crea un programma di Othello? E come "ragiona" questo giocatore artificiale?
Ce lo racconta Michele Diodati, Candidato Maestro romano e unico giocatore a vincere due Campionati Italiani di Categoria.

Un saluto a tutti gli amici giocatori di othello da parte di Diodati Michele.
Una breve prefazione per spiegare perché un nuovo programma di othello e neanche il migliore, ma per fortuna neanche il peggiore.
Per prima cosa il nome, "torazzo", derivato dalla famosa mucca pazza: torazzo programma pazzo…
Vorrei mostrare cosa c'è in comune tra vari giochi come l'othello, la dama o gli scacchi, e anche qualche novità (per quanto ne so io) per la programmazione di Intelligenza Artificiale (I.A.).

Gioco, quindi posso migliorare
Giocare è qualcosa di indispensabile per gli esseri viventi, l'unico modo in cui tutti possono migliorarsi a vicenda senza avere nessun danno e tutti possono apportare un contributo, che gli altri possono fare proprio, e nello stesso tempo capire quale sia la propria preparazione o stato di forma; questo noi othellisti lo abbiamo sperimentato tutti, specie nei tornei.
Cosa c'è in comune tra gli scacchi, la dama o l'othello?
Per quanto riguarda la I.A. molto, in quanto le routine minimax, alpha-beta, tabelle Hash e libro di aperture sono pressoché uguali, mentre per gli esseri umani direi (tranne la dama internazionale) la grandezza della scacchiera, otto per otto, che a mio parere va ad influenzare le sequenze tattiche, che nella maggioranza dei casi non superano le otto mosse consecutive, e come strategia il controllo del centro che impedisce all'avversario di sviluppare il proprio gioco e (come è evidente fin da i tempi di Turing e Shannon e Michail Botvinnik, fisico e matematico russo che cercò di sviluppare un programma di scacchi di nome Pioner, e nel criterio di valutazione utilizzò due parametri che venivano sommati l'uno all'altro), una valutazione materiale più una valutazione di posizione.
Il parametro materiale è evidente negli scacchi e dama, ed è rappresentato dal numero dei pezzi presenti sulla scacchiera, ma è meno evidente in othello in quanto il numero dei pezzi varia in continuazione, almeno fino a quando non si ottengono pedine stabili, che a quel punto diventano parte di una valutazione materiale ma che non ha nessun valore se nell'ultima mossa non sono superiori a quelle avversarie ed inoltre nel conteggio finale vengono considerate anche pedine tecnicamente non stabili; in questo l'othello si distingue molto da altri giochi che hanno valori materiali più facilmente valutabili, anche dal Go, che più gli somiglia.
Il primo passo è descrivere un generatore di mosse, generalmente per questioni di efficienza vengono generate mosse valide e non valide per poi scartare quelle non valide, il tutto in linguaggi di basso o medio livello per ottenere il maggior numero possibile di mosse, vengono poi usate maschere di bit per ottenere queste mosse in modo quasi automatico usando le funzioni or e and.
Io invece avendo usato il basic compilato, quindi lento, ho preferito generare subito soltanto mosse legali, cercando di ottenere il maggior numero di informazioni possibili in questa operazione.
Sono partito creando una matrice di 10 per 10 celle che rappresenta la scacchiera, poi ho memorizzato due 1 che rappresentano le pedine nere e due 2 le bianche, dei 9 che rappresentano il bordo esterno e servono ad impedire degli errori di fuori matrice nel funzionamento, mentre dei 5 riempiono il resto della scacchiera.
La ricerca viene effettuata su tutte le caselle vuote, e per ogni casella vengono eseguiti due cicli FOR annidati da -1 a 1 che rappresentano le coordinate x ed y, che a loro volta eseguono un controllo su le 8 caselle circostanti la casella considerata con una condizione che elimina i valori 0 per x e 0 per y, che andrebbero ad individuare la casella considerata, o meglio ancora questi valori possono essere memorizzati a priori in due

matrici; dopo di questo si controlla se vi sia una pedina avversaria: se questo è vero si memorizzano due variabili che rappresentano la lunghezza della mossa e una matrice che chiameremo *memmosse*, dove verranno memorizzati la lunghezza totale della mossa più i valori di posizione x ed y, dopo con un ciclo DO-LOOP si prosegue se vi sono pedine avversarie, contemporaneamente due valori di lunghezza vengono aumentati di una unità, uno definitivo e uno provvisorio solo per la direzione considerata, poi se viene confermata la direzione con una casella con pedina del giocatore, la lunghezza definitiva rimane quella, e quella provvisoria viene azzerata, in caso contrario quella provvisoria viene sottratta a quella definitiva e poi azzerata e i valori x ed y memorizzati nella matrice vengono sovrascritti dalla nuova direzione.
Al termine del controllo di una casella, se esiste una mossa valida verrà incrementata una variabile che rappresenta il numero di posizione e che incrementerà la posizione di *memmosse*.
Alla fine del controllo della scacchiera avremo tutte le mosse memorizzate con le relative lunghezze e numero di posizioni possibili, quindi anche le mobilità possibili.
Io, poi ho inserito anche una valutazione di valori memorizzati a priori in una matrice 8 per 8, che mi serve per capire quanto può valere la mossa possibile, ed ottenuta tramite un ragionamento che ho chiamato "succede quello che potrebbe succedere".
Può sembrare banale come frase, ma non sono banali le conseguenze, quindi vale la pena spiegarlo.
In pratica tutti sapete che le caselle d'angolo valgono di più delle altre e le x-square sono pericolose, ma se vi chiedessi il valore matematico non so quanti saprebbero rispondermi.
Sono partito proprio dalle caselle d'angolo e ho contato quante pedine potessero essere prese al massimo con le proprie già piazzate, per l'angolo esattamente 19, per le caselle a fianco 17, 15 per le x-square, e così via. Una volta fatto questo si dividono i valori per i valori delle caselle che potrebbero prendere quella considerata, mentre quelle stabili non vengono divise, quindi solo le 4 d'angolo, e si avrà una matrice di valori che si avvicinano molto ai valori che diamo alle caselle sulla scacchiera. Questo darà al generatore di mosse la possibilità di capire quale sia il valore delle caselle dove è possibile muovere.
È poi possibile memorizzare le pedine che generano le mosse, infatti 3 mosse generate da una pedina sarà diverso da 3 mosse generate da 2 pedine.
Avranno un valore anche le caselle vuote, in quanto dividendo le pedine presenti per le caselle vuote si otterrà una cubicità delle pedine stesse, questo perché il rapporto sarà più alto, e moltiplicando le pedine presenti per questo rapporto il programma cercherà di compattare le pedine.
Quindi alla fine avremo:

- numero di pedine,
- numero di pedine prendibili (mobilità) che saranno di più di quelle realmente presenti, in quanto nel conteggio si ripasserà su pedine presenti già contate,
- numero di mosse possibili e il loro valore,
- numero di caselle vuote lungo i bordi delle pedine presenti.

Si può migliorare questo generatore facendogli memorizzare sia le mosse possibili che quelle avversarie nello stesso tempo, e non in 2 chiamate con i valori di bianco e nero invertiti come ho fatto io.

Mettiamo un po' di ordine

Questo è il generatore, ora però andranno valutate le mosse e messe per ordine decrescente.
Per fare questo ho scritto una routine che chiamerò *mossefuture*, che fa la mossa memorizzata nella matrice, ne attribuisce un valore attraverso una funzione statica di valutazione e tramite una routine di ordinamento restituisce la matrice con le stesse mosse ma ordinate e valutate, pronta per essere passata alla routine che farà la valutazione minimax con tagli alpha-beta.
Inizialmente questa routine faceva la valutazione su 2 mosse consecutive con una valutazione statica più leggera dal punto di vista computazionale, ma ho poi ripiegato su una mossa valutata più pesantemente, ma

nulla vieta di fargli fare una analisi in profondità con tagli alpha-beta per le pedine stabili, questo darebbe la valutazione su una mossa priva di difetti da effetto orizzonte.
Quello che scarti oggi può essere la tua rovina di domani...
Il resto del programma fa una ricerca minimax con tagli ed estensioni nel caso di pedine stabili, quindi su questo è già stato scritto molto nella programmazione di giochi, tranne una sperimentazione fatta da me che può interessare, cioè l'uso dei tagli alpha-beta per includere e non solo per tagliare, questo si può fare richiamando nella routine che effettua il minimax subito dopo che ha scritto la mossa sulla scacchiera la routine *mossefuture* ma con valori di pedine invertite, e quindi facendo un taglio, per esempio, nel caso di un taglio alpha attuale un taglio beta futuro, visto che le condizioni sono invertite, che ponendo due condizioni potrebbe invece di tagliare, includere.
Ma l'ho tralasciato dopo aver notato che non c'è un reale guadagno di tempo, e poi andrebbe considerato per l'othello l'eventualità che non vi siano mosse possibili successive, perché nel caso di un taglio alpha attuale ci sarebbe un altro taglio alpha futuro e non beta, però c'è un reale taglio con inclusione che andrebbe approfondito.

Stabilità, una bella cosa

Nell'othello le pedine stabili sono la parte materiale del gioco, ma nelle partite tra buoni giocatori intervengono solo verso il finale: come si fa ad individuarle?
Il sistema usato è partire dagli angoli, che una volta presi diventano stabili, e trasmettere questa stabilità alle pedine adiacenti; questo sistema può essere computerizzato, anche se ha la pecca nel finale di non essere preciso, però si può anche tralasciare se si usa per il proprio programma un finale perfetto, cioè il conteggio di tutte le pedine e le possibilità di gioco alcune mosse prima del termine della partita; questo si può fare facilmente con il minimax e applicando alle sequenze finali la funzione più semplice: pedineproprie-pedineavversarie. Con un'avvertenza nel caso di uso del minimax di invertire i termini nella mossa avversaria, mentre nel negamax viene invertita la funzione.
Questo è un modo di calcolare le pedine stabili, io ne ho comunque individuati altri due.
Il primo consiste per ogni pedina sulla scacchiera, per poi guardare in tutte le direzioni e proseguire verso i bordi della scacchiera fino a che vi siano pedine dello stesso colore, e per ogni volta che si raggiunge il bordo si aumenta di una unità una variabile, se al termine del controllo di ogni pedina questa variabile è uguale o maggiore di quattro la pedina può definirsi stabile, a patto che l'incremento della variabile sia stato consecutivo; infatti, se per esempio questa variabile raggiunge il valore 3, poi non si incrementa nella direzione successiva ma in quella dopo, raggiunge ugualmente il valore 4, ma non è stabile, infatti per ottenere il giusto risultato bisogna azzerare la variabile ogni volta che non si raggiunge il bordo.
L'altro modo è forse il più interessante, anche se non ho ancora sviluppato tutte le implicazioni: consiste nel copiare la situazione che si vuole vedere in una matrice e poi applicare la stessa funzione dove si ricercano le mosse, con la differenza che le pedine trovate vengono cancellate ed inoltre bisogna eseguire anche mosse non legali in quanto partendo da una casella vuota si prosegue nella direzione dove si trovano pedine avversarie ma anche se al termine invece di una propria pedina si trova una casella vuota si devono cancellare comunque, poi questo va ripetuto anche dal punto di vista avversario, al termine si guarda se è rimasto qualcosa sulla matrice provvisoria, e quello rimasto sono pedine stabili, in più potrebbero rimanere pedine non prendibili subito, ma magari dopo la mossa avversaria, quindi si potrebbe risolvere con una valutazione di due mosse in profondità più la funzione con le mosse illegali. Questo diverrebbe pesante dal punto di vista del calcolo, ma in compenso fornirebbe tutti i valori che ci servono (mosse possibili, mobilità, numero pedine, sia stabili che non ecc.) dal punto di vista nostro e dell'avversario, e in più introdurrebbe un concetto di pedina semistabile, cioè quelle non prendibili immediatamente, ma in questo non ho ancora ben capito che valore dare a queste pedine, e in più non so ancora come trarre in modo efficiente tutto quello che è possibile, visto

che avrebbe implicazioni anche sull'effetto orizzonte e sui tagli alpha-beta, quindi nel programma ho applicato il primo metodo.

Valutare sembra semplice

Una funzione di valutazione può sembrare semplice, la valutazione delle mie mosse meno quelle avversarie, ma non è così.

Un piccolo esempio: se io ho 10 mosse e l'avversario 5, il risultato è 5, ma anche 15 meno 10 fa 5 eppure nella partita reale la mia valutazione umana darebbe una stima migliore per la prima situazione perché fa avvicinare allo 0 l'avversario.

Allora come si ottiene matematicamente la stessa stima?

Bisognerà dividere la sottrazione per il valore avversario, questo farà seguire la strada del valore più piccolo dell'avversario a parità di differenza: (10-5)/5 = 1:(15-10)/10 = 0.5

Il valore più grande dell'avversario a parità di differenza: (10-5)*10 = 50:(15-10)*15 = 75.

Purtroppo la sperimentazione con il programma non dà il risultato desiderato, perché cerca di perseguire questo valore più piccolo dell'avversario in modo troppo precipitoso, e questo porta ad una situazione molto scoordinata che alla fine sfavorisce il programma, mentre un umano lo farebbe in modo più bilanciato e coerente, quindi per il programma è meglio seguire la strada del valore più grande a parità di differenza. Credo però che sia una mancanza di conoscenza che porta a queste diversità tra il ragionamento umano e quello matematico, e forse c'è una via di mezzo.

Comunque si può anche normalizzare il risultato: (10*5)/(10+5) che porta al maggior valore a parità di differenza, con lo svantaggio che questo valore non può essere visto sia dalla propria parte, o anche a parti invertite, che nel minimax serve, mentre non occorre nel caso del negamax, oppure si può semplicemente moltiplicare il valore per se stesso e sottrarlo all'altro moltiplicato allo stesso modo: (10*10)-(5*5) = 75.

Un'altra verifica che ho potuto fare per la valutazione posizionale è che bisogna scegliere un tipo di valore, io ho scelto le pedine non stabili negativizzandole, cioè sottraendole ad un numero fisso e sommando invece quelle avversarie, e moltiplicandole per un fattore che le renda più importanti dei fattori che interverranno in seguito, ma si può per esempio scegliere anche il numero di posizioni possibili con la stessa modalità o altri ancora.

(100-pedinenonstabiliproprie+pedinenonstabiliavversarie)*100

Per ottenere questo risultato, più che moltiplicarlo per le proprie possibilità conviene dividerlo per le possibilità avversarie o anche per le proprie carenze (come le caselle vuote intorno alle proprie pedine), questo perché la moltiplicazione porta ad un gioco non coerente, mentre la divisione crea un gioco più armonico.

Meglio ancora, conviene applicare queste divisioni al proprio risultato a cui sottrarre poi la stessa operazione ma invertendo i valori, cioè visti dall'avversario. Detto in linguaggio naturale: non conviene aprire nuove strade, con la speranza che l'avversario le percorra, ma ridurre quelle che si hanno a disposizione eliminando quelle favorevoli all'avversario e, tagliando tutto quello che sembra meglio per l'altro, alla fine forse siamo proprio sopra alla strada cercata.

Quindi una buona funzione posizionale potrebbe essere:

1000
più
((100-
pedinenonstabiliproprie)*100)/(numeromosseavversarie)/(casellevuoteproprie)/(valorenumeromosseavversarie)
meno
((100-

pedinenonstabiliavversarie)*100)/(numeromosseproprie)/(casellevuoteavversarie)/(valorenumeromossepr
oprie)
e così via, dividendo per tutto quello che riusciamo a vedere di negativo per la nostra posizione.
Le pedine non stabili si ottengono per sottrazione tra tutte le pedine presenti meno quelle stabili e sono da considerare negative, perché una forte loro presenza mette in difficolta il nostro gioco, ma d'altro canto una loro minima presenza è indispensabile per avere numerose opzioni di gioco, per questo vanno soppesate al fine di ottenere quelle migliori per noi.
Che ne pensate, è facile valutare?

Un'altra piccola attenzione è da porre nel fatto che quando si moltiplica un numero intero per una frazione di unità in realtà lo si divide, o viceversa se si intende dividere lo si moltiplica.
Attenzione anche alle possibili negativizzazioni dei valori:10-5 = 5:5-10 = -5; è quindi opportuno far rimanere i numeri sempre in ambito positivo sommando un numero positivo opportuno.
Un altro possibile errore nel programma potrebbe essere la somma tra la valutazione materiale e quella posizionale, in quanto la parte materiale se ben calcolata porta alla vittoria.
Vediamo questo esempio: 10 pedine stabili con 10 mosse possibili contro 10 pedine stabili avversarie e 5 mosse possibili. Anche se io attribuisco un valore 100 per ogni stabile, non riesco a far emergere la grandezza in proporzione delle pedine stabili contro i valori di posizione, infatti ((10-10)+(10-5)) dà lo stesso risultato di ((1000-1000)+(10-5)); sarà quindi meglio moltiplicare le stabili per il valore 100 dopo la sottrazione e dopo che si è portato questa sottrazione in ambito positivo sommando un numero opportuno.
Quindi una piccola disattenzione di calcolo nella funzione di valutazione potrebbe portare a credere che si sia sbagliato nel valutare.

Giocare per perdere
Ho inserito nel programma una funzione con cui molti othellisti si sono cimentati: giocare per perdere.
Ma come si gioca per perdere?
D'istinto si direbbe cedere gli angoli, ma è la stessa sensazione che si ha quando si inizia a giocare ad othello, cioè che per vincere bisogna prendere quante più pedine possibile.
In realtà con il mio programma risulta molto facile, dato che la funzione materiale è divisa dalla posizionale, quindi se ammettiamo che si vince con le pedine stabili basta invertire questo dato, mentre è importante che la funzione posizionale sia la stessa di quando si gioca per vincere, infatti è importante padroneggiare la scacchiera per costringere l'avversario a vincere (cioè perdere), provare per credere.

Ho scritto questo per invogliare altre persone alla scrittura di un programma che giochi ad othello, ma scritto in un linguaggio che sia più rapido del basic e magari per portarlo su altre piattaforme, come telefonini o tablet.
Ed inoltre aggiungere parti mancanti, come la memorizzazione di risultati già calcolati nelle mosse precedenti, per il riuso con chiavi ad accesso casuale e l'autoapprendimento tramite la memorizzazione di una libreria con le partite giocate e una libreria di aperture.

Othello un minuto per imparare...

Un salutone a tutti