UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Dalla αlfa alla βeta

RELATORE

Prof. **Fabio Palomba**

Università degli studi di Salerno

CANDIDATO

Giuseppe Pagano

Matricola: 0512106337

Anno Accademico 2021-2022

INSERIRE QUI UNA DEDICA O UNA CITAZIONE

Sommario

INSERIRE ABSTRACT

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Gli scacchi sono un gioco di strategia deterministico a somma zero e ad informazione completa che si svolge su una tavola quadrata formata da 64 caselle ,di due colori alternati, detta scacchiera sulla quale ogni giocatore contraddistinto da uno di due colori nero o bianco, dispone di 16 pezzi:un re, regina, due alfieri, due cavalli, due torri e otto pedoni. obiettivo del gioco è dare scacco matto, ovvero minacciare la cattura del re avversario mentre esso non ha modo di rimuovere il re dalla sua posizione di pericolo alla sua prossima semimossa. Ulteriori informazioni verranno fornite nei successivi capitoli quando maggiori conoscenze di teoria si riveleranno necessarie per poter procedere allo sviluppo del motore.

1.2 Motivazioni e obiettivi

Gli scacchi,gioco nato in india attorno al 600 d.C,da gioco utilizzato nelle corti aristocratiche per rappresentare rapporti di potere a campo di battaglia tra uomo e macchina in uno dei primi e più famosi tentativi di far superare ad una macchina l'intelletto umano (Kasparov vs Deep Blue 1996-1997) ,gli scacchi , non hanno mai fallito nel saper cattivare l'attenzione del grande pubblico nonostante abbiano ormai più di 1000 anni sulle spalle.
Quello che agli occhi di un profano potrebbe sembrare un fenomeno stranissimo è in realtà di facile spiegazione se ci si concentra su una delle caratteristiche fondamentali del gioco degli

scacchi questa caratteristica è la **complessità**,in una partita di scacchi fin dalla prima semimossa sono possibili 20 scelte per la seconda semimossa il totale di possibili combinazioni sale a 400, dopo 5 semimosse avremo 119,060,324 possibili risposte, le possibili mosse di una partita si stimano attorno alle 2^{155}.
Con uno spazio di ricerca cosi grande non dovrebbe stupire sapere che è da quando esistono i computer che si cerca un modo di sfruttare la loro potenza di calcolo nel mondo degli scacchi.
La nascita degli scacchi computazionali si deve al lavoro di Claude Shannon, famoso per i suoi innumerevoli contributi al campo della teoria dell'informazione,egli, con il suo paper "Programming a Computer for Playing Chess" del 1950 ha gettato le basi per quello che oggi è il campo conosciuto come scacchi computazionali.
Questa tesi nasce dalla volontà di esplorare questo vasto e interessante campo dell'informatica,e dal voler creare un testo in grado di guidare chiunque lo legga nella creazione di un motore scacchistico spiegando tutte le fasi della progettazione ed illustrando le possibili scelte che condizionano l'efficienza di un motore ,dato che la letteratura su questo fronte è non particolarmente florida e soprattutto quasi esclusivamente in lingua inglese.

1.3 Risultati ottenuti

1.4 Struttura della tesi

1.5 Prefazione

Lo sviluppo di un motore scacchistico è fortemente influenzato dalle scelte progettuali, una di queste è il linguaggio di programmazione che si vuole utilizzare, le prestazioni di un motore possono essere fortemente influenzate dalla natura del linguaggio di programmazione, in particolare l'utilizzo di un linguaggio interpretato e non compilato può impattare notevolmente sulla velocità con la quale il nostro motore è in grado di elaborare le milioni di posizioni con le quali dovrà avere a che fare in una singola partita. Tutti gli esempi di codice all'interno di questa tesi saranno scritti nel linguaggio C, si consiglia quindi di avere almeno una minima familiarità con tale linguaggio.Si segnalano comunque diversi tool per il linguaggio python per chi volesse approcciarsi a questo mondo utilizzando un linguaggio più beginner friendly quali:

- **python-chess**: una libreria di python che contiene funzioni di libreria per la rappresentazione di scacchiera e pezzi e per la generazione e validazione delle mosse,utile se ci si

vuole concentrare esclusivamente sulla parte di ricerca e di valutazione di un motore scacchistico.

- **Sunfish**: un motore scacchistico per principianti scritto nel linguaggio python che in sole 111 linee di codice illustra ,in maniera semplificata, l'implementazione della gran parte dei concetti chiave di un motore scacchistico.

CAPITOLO 2

Rappresentazione della scacchiera e dei pezzi

In questo capitolo viene mostrato passo passo un procedimento guida alla realizzazione di un motore scacchistico

2.1 Introduzione

Il primo passo dello sviluppo di un motore scacchistico è decidere come si vuole rappresentare la scacchiera , si tratta di una scelta fondamentale,non solo perché in seguito ci permetterà di testare le funzioni che andremo a implementare, ma anche perché è nella scacchiera che ,generalmente, viene conservato lo stato generale della partita.[1] Inoltre il tipo di codifica può influenzare la rapidità e la facilità col quale possiamo accedere alle informazioni sullo stato corrente dei pezzi e come vedremo in seguito è in grado di influenzare funzioni come la generazione delle mosse. Non è raro per motori scacchistici particolarmente complessi l'utilizzo di più tipi di board in base al tipo di informazione da conservare e all'utilizzo che se ne vuole fare. Per la rappresentazione di una scacchiera sono chiaramente possibili moltissime scelte, di seguito verranno illustrate alcune tra le più popolari ed utilizzate.

2.2 Rappresentazione della scacchiera Pezzocentrica

Si definisce rappresentazione pezzocentrica, un qualsiasi tipo di rappresentazione della scacchiera che mantiene liste array o set dei pezzi attualmente presenti sulla scacchiera con annesse le informazioni sulle caselle da essi occupate. Le rappresentazioni più comuni sono:

Piece-Lists

liste o array di ogni pezzo sulla scacchiera,ogni elemento della lista o dell'array associa un pezzo alla casella che esso occupa .le caratteristiche di ogni pezzo (colore,tipo etc) possono essere associate all'indice dell'array in cui si trovano o essere presenti in ulteriori array o liste esterne.

Bitboards

Una Bitboard è una struttura dati specifica per i giochi da tavolo, si tratta in sostanza di una struttura dati in grado di immagazzinare lo stato di ogni casella della scacchiera all'interno di una parola [2] di 64 bit. Vediamo un esempio pratico, immaginiamo di avere una scacchiera che si trova nello stato di default di inizio partita:

[1] per stato di una partita si intendono informazioni come informazioni su chi ha diritto di muovere, i permessi di arrocco,lo stato della regola delle 50 mosse etc

[2] Una parola è un gruppo di bit di una determinata dimensione che sono gestiti come unità dal set di istruzioni o dall'hardware di un processore

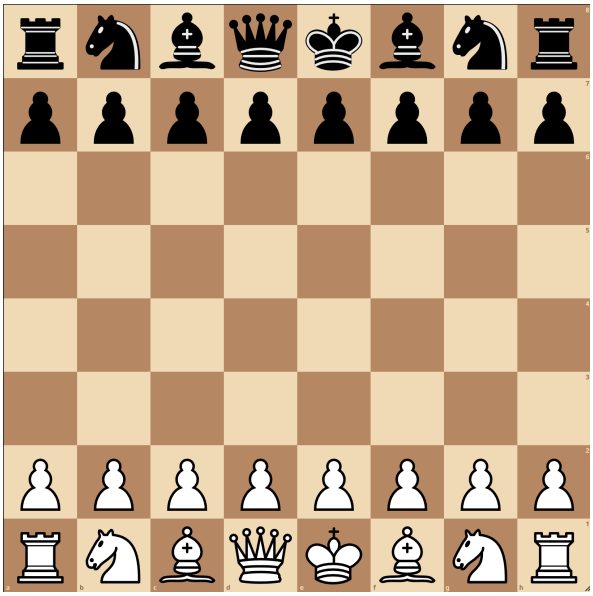

Figura 2.1: Scacchiera nella posizione iniziale di gioco

Una bitboard tipica è quella che ci permette di sapere in quali caselle è presente un pedone nero,per costruirla, operando casella per casella, ci poniamo una domanda "in questa casella è presente un pedone nero?" se si allora quella casella viene marcata con un 1 , altrimenti viene marcata con uno 0, il risultato di questa traduzione diventa in questo caso: La bitboard

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figura 2.2: bitboard

che codifica questa informazione sarà quindi la parola di 64 bit 00000000 11111111 00000000 00000000 00000000 00000000 00000000 00000000

Piece-Sets

rappresentazione con set con un bit per ogni pezzo dentro una parola a 32 bit o 2 parole a 16 bit per ogni lato. i Piece-sets hanno delle somiglianze con le bitboards, ma ogni bit del set non è direttamente correlato ad una casella, ma ad un indice dentro ad una piece-list. Spesso la bit-position di un piece-set implica , di che tipo e colore il pezzo è. - mentre le bitboards solitamente mantengono set distinti per pezzi diversi.

2.3 Rappresentazione della scacchiera Casellocentrica

La rappresentazione casellocentrica mantiene un associazione inversa rispetto a quella pezzocentrica, per ogni casella conserviamo in memoria se è vuota o occupata da un pezzo in particolare. La macro-categoria di rappresentazione più comune è la Mailbox:

Mailbox

La rappresentazione Mailbox è una rappresentazione casellocentrica dove la codifica di ogni casella risiede in una struttura dati che permette l'accesso casuale ,solitamente si utilizza un array con l'indice che codifica dal numero della casella in array monodimensionali o dalla coppia traversa/colonna[3] in array bidimensionali. Il nome deriva dall'associazione di ogni indice al concetto di "indirizzo" di una casella postale.Le implementazioni più famose e comuni del concetto di Mailbox sono la 8x8 Board e la 10x12 Board.

8x8 Board

Una board 8x8,figura 2.3 è una rappresentazione pezzocentrica consistente o in un array bidimensionale di bytes o interi, contenenti rappresentazioni codificate per i pezzi e per la casella vuota, con i due indici ricavati dalla coppia traversa/colonna che identifica la casella sulla scacchiera , o più comunemente un array monodimensionale con indici da 0 a 63,uno per ogni casella della scacchiera. Questo tipo di rappresentazione è usata spesso come rappresentazione ridondante all'interno di programmi che utilizzano bitboards per individuare se e quali pezzi sono presenti su una casella in maniera efficiente.

[3] termini scacchistici per indicare le righe e le colonne della scacchiera

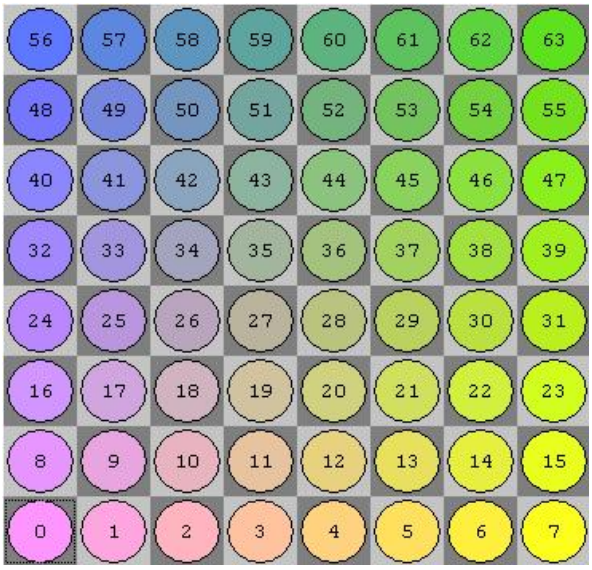

Figura 2.3: board numbering

10x12 Board

Una board 10x12 contorna una board 8x8 con traverse e colonne sentinelle per individuare indici al di fuori della scacchiera durante la generazione delle mosse

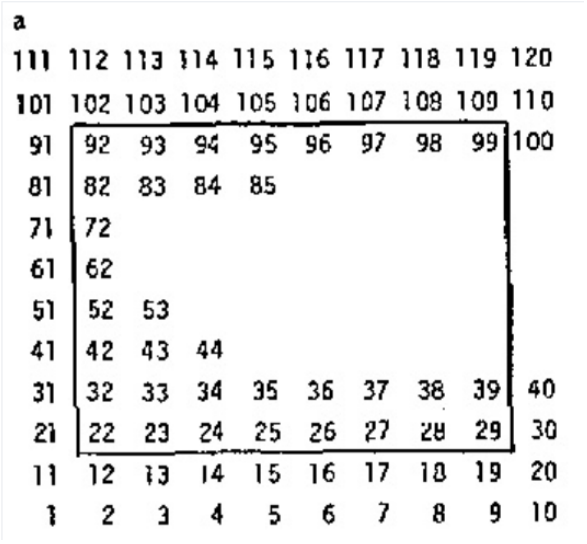

Figura 2.4: Rappresentazione di una board 10x12,notare come le 2 traverse extra sopra e sotto siano necessarie per evitare accessi out of bounds in caso di cavalli posizionati sulla prima o ottava fila

2.4 Rappresentazione dei pezzi

una volta scelto il tipo di rappresentazione della scacchiera si può iniziare a pensare alla rappresentazione dei pezzi, anche se può sembrare controintuitivo i pezzi non hanno bisogno di una struttura elaborata,la generazione delle possibili mosse verrà gestita nella move generation ed il loro spostamento all'interno della scacchiera verrà gestito dalla funzione MakeMove (approfondimenti riguardo questi due argomenti sono presenti nel capitolo 3),per i pezzi quindi abbiamo bisogno di una rappresentazione semplice, di facile interpretazione e che occupi poco spazio.
Rappresentazioni molto comuni sono quella tramite interi, dove ad ogni tipo di pezzo viene assegnato un numero che funge da identificativo univoco e quella tramite caratteri, dove ad ogni tipo di pezzo viene assegnato un carattere che lo identifica.

CAPITOLO 3

Move Generation

3.1 Introduzione

Una volta stabilito il tipo di rappresentazione il passo successivo è quello della generazione delle mosse,per generazione delle mosse si intende la generazione di tutte le mosse legali eseguibili data una posizione di partenza , la generazione delle mosse è un processo fondamentale in quanto identifica i rami che il nostro algoritmo potrà esplorare nella successiva fase, quella appunto di ricerca, trattata nel capitolo 4.

3.2 approcci alla generazione delle mosse

Una volta stabilito il tipo di rappresentazione il passo successivo è quello della generazione delle mosse,per generazione delle mosse si intende la generazione di tutte le mosse legali eseguibili data una posizione di partenza , la generazione delle mosse è un processo fondamentale in quanto identifica i rami che il nostro algoritmo potrà esplorare nella successiva fase, quella appunto di ricerca, trattata nel capitolo 4. La generazione delle mosse viene affrontata con 3 approcci radicalmente diversi a seconda del tipo di pezzo che stiamo trattando,in particolare la distinzione si effettua tra pedoni,pezzi scorrevoli (alfieri,torri,regine) e non scorrevoli(re e cavalli).

3.2.1 pedoni

Il pedone muove solo in avanti, mai indietro o di lato. Un pedone può avanzare di due caselle se è la prima volta che viene mosso,altrimenti può avanzare solo di una casella. Il un pedone mangia un pezzo avversario spostandosi diagonalmente di una casella,sempre soltanto in avanti, o a destra o a sinistra, se un pedone raggiunge la traversa finale della scacchiera rispetto alla sua direzione di movimento allora viene promosso, diventa quindi,a scelta del giocatore che possiede la pedina, uno qualsiasi degli altri pezzi (ad eccezione del re). In un approccio basato su board 8x8 o 12x10 possiamo calcolare le possibili mosse del pedone affidandoci a un flag che rappresentano la possibilità di effettuare catture en passant e controllare la possibilità di muovere di 2 caselle e di promozione basandoci sulla posizione di partenza del pedone,le caselle di arrivo possibili sono a quel punto calcolabili sommando alla casella di partenza degli offset prefissati.

3.2.2 pezzi non scorrevoli

Table-driven Move Generation

I pezzi non scorrevoli sono i pezzi che possono spostarsi solo di un numero finito di caselle,questo vuol dire che,conoscendo il tipo e la posizione di un pezzo non scorrevole,possiamo calcolare i suoi movimenti sommando alla sua posizione dei valori prefissati,a quel punto dovremo solo assicurarci che la casella di arrivo sia nei limiti della scacchiera e che la mossa risultante sia legale[1].

3.2.3 pezzi scorrevoli

Si definiscono pezzi scorrevoli i pezzi che possono spostarsi di un numero non prefissato di caselle lungo l'asse orizzontale,verticale o diagonale,fino a raggiungere il bordo della scacchiera o un altro pezzo. I pezzi scorrevoli consistono di alfiere,torre e regina,la generazione delle mosse di questo tipo di pezzi è più complessa rispetto a quella degli altri pezzi, quanto bisogna controllare la presenza di pezzi,propri o avversari che siano, in grado di fermare il movimento del pezzo e bisogna assicurarsi che il pezzo non superi il bordo della scacchiera. nel corso dei decenni si è arrivati a stabilire due approcci alla progettazione ottimali per la generazione delle mosse dei pezzi scorrevoli, da preferire in base all'architettura della macchina che esegue il motore

Bitboard magiche

L'approccio con le bitboard machine consiste nell'assegnare ad ogni pezzo scorrevole un numero "magico" pre-calcolato per ogni casella della scacchiera, rappresentata da una bitboard,quel numero verrà poi moltiplicato per il valore individuato dalla posizione dei pezzi bloccanti sulla scacchiera sulla bitboard, questa moltiplicazione darà luogo ad un indice con il quale accedere ad un archivio di bitboard di attacco[2] pre-calcolate,Si tratta di una soluzione meno efficiente rispetto alle bitboard che si basano sull'utilizzo di pext e va utilizzata solo in caso di architetture che non supportano nativamente istruzioni pext

[1]Nel gioco degli scacchi una mossa illegale è una mossa che pone il re sotto scacco o non libera il re da uno scacco subito alla mossa precedente

[2]bitboard che segnala con dei bit ad 1 quali caselle il pezzo può attaccare

PEXT Bitboards

Una bitboard PEXT funziona in maniera simile ad una bitboard magica,utilizza il valore individuato dalla posizione dei pezzi bloccanti sulla scacchiera sulla bitboard per generare un indice da usare per accedere ad una bitboard di attacco,la differenza sostanziale però è nell'approccio, una pext bitboard, come intuibile dal nome, utilizza l' istruzione pext introdotta nel set di istruzioni BMI2 da intel sui processori haswell,e con la serie 5000 dei processori ryzen da AMD,si tratta di un'istruzione che ,dato un input ed una maschera,trasferisce i bit dell'input individuati dalla maschera,in ordine contiguo e da destra a sinistra. Con l'istruzione pext, usando una bitboard che rappresenta la scacchiera come input e la posizione dei pezzi bloccanti come maschera, si può ottenere un indice per la hash table delle bitboard di attacco,il tutto viene eseguito in un solo ciclo di cpu e risulta quindi molto più efficiente di un approccio con bitboard magiche.

3.3 Perft

Perft è una funzione di debugging che attraversa l'albero delle mosse legali generate e conta tutti i nodi foglia fino ad una data profondità n.Il numero viene poi comparato con dei valori noti per controllare la presenza di bug. I nodi vengono contati alla fine della generazione, dopo l'ultimo makemove,non vengono quindi contati i nodi foglia che si trovano a profondità superiori di n.Perft inoltre ignora i pareggi per ripetizione, per la regola delle 50 mosse, e per materiale insufficiente.Utilizzando la stessa versione di perft, o In alternativa una versione estremamente simile di perft,è possibile comparare il tempo impiegato da diversi generatori di mosse.

```
typedef unsigned long long u64;

u64 Perft(int depth)
{
  MOVE move_list[256];
  int n_moves, i;
  u64 nodes = 0;

  if (depth == 0)
    return 1ULL;

  n_moves = GenerateLegalMoves(move_list);
  for (i = 0; i < n_moves; i++) {
    MakeMove(move_list[i]);
    nodes += Perft(depth - 1);
    UndoMove(move_list[i]);
  }
  return nodes;
}
```

Figura 3.1: codice di una funzione basica di perft

CAPITOLO 4

Ricerca

In questo capitolo viene mostrato passo passo un procedimento guida alla realizzazione di un motore scacchistico

4.1 Esempio di implementazione

CAPITOLO 5

Valutazione

Questo capitolo illustra lo stato dell'arte e i lavori presenti in letteratura sugli aspetti di ricerca trattati nel nostro studio. ECC ECC...

5.1 Esempio di implementazione

CAPITOLO 6

Libro delle aperture,Tablebases

Questo capitolo illustra lo stato dell'arte e i lavori presenti in letteratura sugli aspetti di ricerca trattati nel nostro studio. ECC ECC...

6.1 Esempio di implementazione

CAPITOLO 7

Validazione preliminare

BREVE SPIEGAZIONE CONTENUTO CAPITOLO

Introduzione al concetto di ELO,rappresentazione ELO stimata del motore,metriche prestazionali varie

CAPITOLO 8

Stato dell'arte per Algoritmi di intelligenza artificiale e motori scacchistici per il gioco degli scacchi

Questo capitolo illustra lo stato dell'arte e i lavori presenti in letteratura sugli aspetti di ricerca trattati nel nostro studio. ECC ECC...

8.1 Pre Neural Network Stockfish

8.1.1 Board Representation

8x8 Board Bitboards with Little-Endian Rank-File Mapping (LERF) Magic Bitboards BMI2 - PEXT Bitboards (not recommend for AMD Ryzen [28] prior to Zen 3)

8.1.2 Search

Iterative Deepening Aspiration Windows Parallel Search using Threads YBWC prior to Stockfish 7 Lazy SMP since Stockfish 7, January 2016 Principal Variation Search Transposition Table Shared Hash Table 10 Bytes per Entry, 3 Entries per Cluster Depth-preferred Replacement Strategy No PV-Node probing Prefetch Move Ordering Countermove Heuristic Counter Moves History since Stockfish 7, January 2016 [32] History Heuristic Internal Iterative Deepening Killer Heuristic MVV/LVA SEE Selectivity Extensions Check Extensions if SEE >= 0 Restricted Singular Extensions Pruning Futility Pruning Move Count Based Pruning Null Move Pruning Dynamic Depth Reduction based on depth and value Static Null Move Pruning Verification search at high depths ProbCut SEE Pruning Reductions Late Move Reductions Razoring Quiescence Search

8.1.3 Evaluation

8.1.3.1 Material

Bishop Pair Imbalance Tables Material Hash Table

8.1.3.2 Mobility

Trapped Pieces Rooks on (Semi) Open Files

8.1.3.3 Pawn Structure

Pawn Hash Table

Backward Pawn

Doubled Pawn

Isolated Pawn

Phalanx

Connected Pawns

Passed Pawn

King Safety

Attacking King Zone Pawn Shelter Pawn Storm Square Control

Tapered Eval

Evaluation Patterns

Piece-Square Tables

Space

Outposts

8.2 Google AlphaZero

8.3 Post Neural Networ Stockfish-NNUE e LCZero

Nonostante le critiche ed i dubbi sulla performance di AlphaZero, gli appassionati di scacchi computazionali non rimasero impassibili davanti ai meriti di un approccio orientato alle reti neurali,il 6 agosto del 2020, un anno e mezzo dopo la pubblicazione definitiva dell'articolo su AlphaZero da parte di DeepMind e dopo un anno di lavoro, viene ufficialmente introdotta ,all'interno della repo di Stockfish,NNUE. NNUE, acronimo di "Effeciently Upgradable Neural Network" scritto da sinistra a destra, è una rete neurale per la valutazione di posizioni shogi, alle quali assegna un punteggio utilizzato poi in fase di potatura,adattata per operare sugli scacchi ed essere integrata in Stockfish. In questa versione Stockfish mantiene le caratteristiche principali che contraddistiguono la sua versione precedente, e la valutazione NNUE viene utilizzata solo in posizioni materialmente bilanciate

CAPITOLO 9

Conclusioni e Sviluppi Futuri

BREVE SPIEGAZIONE CONTENUTO CAPITOLO

Ringraziamenti

INSERIRE RINGRAZIAMENTI QUI