

Corso di Laurea Magistrale in

Ingegneria Informatica

Progetto di Intelligenza Artificiale

Gruppo CrAIzy

Di Cello Danny 224527

Miriello Raffaele 227564

Palumbo Francesco 235805

Seminara Giuseppe 227540

Anno Accademico 2021/2022

***Descrizione sintetica dell’idea progettuale***

Il progetto in questione consiste nello sviluppo di un giocatore automatico per il gioco Fission. Si è deciso di sviluppare il seguente giocatore utilizzando il linguaggio JAVA, in particolare la logica applicativa è implementata in due classi: BitBoard, Minimax.

**Rappresentazione della scacchiera**

La scacchiera di gioco viene rappresentata attraverso due Bitset, ovvero un vettore binario, dove la presenza di un 1 indica che la posizione corrispondente è occupata da una pedina, mentre la presenza di uno 0 indica che la posizione è libera.

La classe Bitboard implementa tutta la logica relativa alla rappresentazione della scacchiera, possiamo dunque individuare due oggetti Bitset, che rappresentano rispettivamente le pedine del giocatore Bianco e del giocatore Nero. In questo modo è possibile rappresentare tutte le pedine di un colore tramite un intero a 64 bit riducendo notevolmente la complessità spaziale.

All’interno della classe Bitboard troviamo inoltre due numeri interi che permettono di tenere traccia del numero di pedine a disposizione di ogni giocatore, in questo modo si evita di dover calcolare tale valore ad ogni mossa, ma si procede semplicemente aggiornandoli, ottenendo dunque una maggiore efficienza computazionale.

All’interno di tale classe, particolare attenzione va riposta ai metodi isPossible e muovi, i quali si occupano rispettivamente di verificare se una data mossa è lecita, secondo le regole del gioco ed eventualmente effettuarla, aggiornando quindi lo stato della scacchiera in modo coerente. Entrambi i metodi appena citati necessitano dello stato complessivo della scacchiera che tiene conto sia delle pedine bianche, sia delle nere e per tale motivo, viene calcolato l'OR tra i due bitset (operazione computazionalmente trascurabile); i metodi lavoreranno sul risultato di tale operazione.

Troviamo anche alcuni metodi di utilità all’interno della classe Bitboard come ad esempio:

* clone, che si occupa di effettuare una copia profonda di un bitset (utilizzato nei metodi isPossible e Muovi);
* esplodi ed elimina, che si occupano di aggiornare lo stato della scacchiera e il numero di pedine dopo uno scontro tra quest’ultime.

Merita anche attenzione il metodo mossePossibili, il quale viene utilizzato per generare tutti i possibili stati raggiungibili partendo dallo stato attuale della scacchiera. Esso ricopre un ruolo fondamentale nel processo di creazione e selezione della mossa migliore.

***Euristica***

Sempre all’interno della classe Bitboard troviamo i metodi diff e somma, che permettono di calcolare il valore euristico associato ad una data configurazione della scacchiera. Ad esempio, per il giocatore bianco, restituisce semplicemente la differenza tra il numero di pedine bianche e il numero di pedine nere, pesata rispetto al totale delle pedine.

Il discorso è analogo per il giocatore nero.

Abbiamo deciso quindi, di adottare un’euristica che fosse il giusto compromesso tra una aggressiva, disposta a sacrificare qualcosa per avere del vantaggio futuro, ed una conservativa, che non è disposta in nessun caso a perdere pedine. Così facendo le mosse che eliminano il maggior numero di pedine avranno un valore euristico maggiore in valore assoluto e dunque saranno più o meno favorite. Questo perché quando si è in vantaggio è preferibile puntare a chiudere la partita (vengono ad esempio favorite le mosse che eliminano un egual numero di pedine bianche e nere), al contrario in una situazione di svantaggio si vuole cercare di prendere più tempo possibile (in particolare alla fine della partita se si è in una situazione del genere 1 vs. 2 si cerca di pattare, e quindi bisogna sfavorire le mosse che portano a 0 vs. 1 (euristica = -1) e favorire quelle che mantengono 1 vs. 2 (euristica = -1/3).

***Algoritmi di ricerca***

La classe principale è Minimax che contiene la logica del giocatore artificiale e il main dell’applicazione. Nel corso dello sviluppo del progetto sono stati considerati diversi algoritmi di ricerca via via più complessi ed efficienti, con lo scopo di individuare la strategia migliore in vista del torneo.

Possiamo individuare inoltre la presenza di un inner class chiamata Nodo, tale classe contiene al suo interno una Bitboard per mantenere lo stato della scacchiera, un float che indica il valore dell’euristica associato a tale configurazione, una priority queue di nodi, che rappresenta tutte le possibili configurazioni raggiungibili dal nodo stesso ordinate secondo il valore euristico della configurazione (in questo modo si ottiene un miglioramento delle prestazioni negli algoritmi che utilizzano il pruning dato che solitamente le mosse iniziali migliori portano a configurazioni finali migliori) e l’ultima mossa eseguita per raggiungere tale configurazione. Tale classe rappresenta la base, utilizzata dai diversi algoritmi di ricerca per costruisce l’albero delle mosse ed individuare la scelta migliore.

Sono stati implementati inoltre, i seguenti metodi che corrispondono a diverse strategie di ricerca:

* minMax: algoritmo base utilizzato come punto di partenza.
* alfabeta: aggiunta della potatura alfa-beta all’algoritmo minimax.
* negamaxAlphaBeta: algoritmo negamax con potatura alfa-beta.
* negascout: implementazione dell’algoritmo negascout.

Tra gli altri metodi presenti in tale classe troviamo i metodi:

* scegli: che si occupa di invocare la strategia di ricerca scelta con i rispettivi parametri, e restituisce la prima mossa tra quelle che massimizzano il valore dell’euristica per il nostro giocatore.
* generaFigli: si occupa di generare, attraverso l’invocazione del metodo mossePossibili, tutte le possibili configurazioni raggiungibili partendo da quella attuale.
* mossaCrAIzy: in cui si sceglie la prima mossa da effettuare tra le configurazioni che, dopo varie analisi, risultano essere le migliori.

Infine, troviamo il main, che si occupa di inizializzare tutti i dati necessari e gestisce la comunicazione col server.

Valutando i diversi algoritmi di ricerca descritti in precedenza, si è concluso che alfabeta è l’algoritmo che porta i maggiori benefici. In particolare, l’algoritmo base è stato modificato introducendo dei pesi, calcolati sulla base del valore euristico del nodo in questione e della sua profondità, che si vanno a sommare al valore euristico propagato dai livelli inferiori dell’albero. In questo modo, tra tutte le mosse migliori restituite dall’algoritmo, andiamo a preferire quella che effettua prima le mosse migliori (prioritizzando comunque il valore euristico del nodo foglia). Si noti come l’algoritmo modificato sia presente solo nel codice del minimizzatore, in questo modo l’avversario non gioca prioritizzando il valore euristico della mossa immediatamente successivo; infatti, si è supposto che gli algoritmi avversari non implementino una logica del genere.

***Classi di supporto***

Sono presenti, inoltre, tre classi di supporto:

* Mossa: è una classe che codifica una tripla formata da Direzione, riga e colonna della scacchiera, e implementa la conversione tra la notazione adottata dal server e quella adottata dal nostro giocatore per rappresentare la scacchiera e le mosse.
* Direzione: è un’enumerazione che si occupa di convertire i punti cardinali(direzione), in una coppia di versori, per semplificare la logica applicativa utilizzata all’interno della classe BitBoard.
* ServerComunication: è una semplice classe che contiene i metodi utilizzati per comunicare col server.