RELAZIONE PROGETTO
ALGORITMI E STRUTTURE DI DATI

Davide Luccioli 0001028403

Player: LoremIpsum

Alma Mater Studiorum – Università di Bologna
Corso di Laurea in Informatica
A.A. 2021-2022

INTRODUZIONE

OBIETTIVI:

Il progetto consiste nella creazione di un giocatore software in grado di giocare in maniera ottimale a tutte le possibili configurazioni di un (M, N, K)-Game: un gioco ad informazione perfetta e a somma zero in cui due giocatori, a turno, marcano con un simbolo (X o O) una casella su una tabella di dimensione MxN al fine di marcare K celle consecutive (verticalmente, orizzontalmente o diagonalmente) con il proprio simbolo.

Il giocatore deve implementare l'interfaccia MNKPlayer, fornita con il pacchetto mnkgame, la quale richiede l'implementazione di metodi per l'inizializzazione del giocatore, e per la selezione della cella da marcare nel turno corrente.

IL PROBLEMA:

La selezione della casella risulta essere la parte centrale del progetto, in quanto il giocatore deve essere in grado di decidere, in maniera intelligente ed entro un limite di tempo (fissato a 10 secondi), quale mossa effettuare.

Tipicamente, la ricerca di strategie ottime su giochi a somma zero viene interpretata come un problema di decisione su alberi: ogni stato della tabella di gioco può infatti essere visto come un nodo di un game tree la cui radice rappresenta lo stato della tabella all'inizio della partita, mentre le foglie sono gli stati finali e gli archi le mosse dei giocatori. Idealmente questo tipo di problema può essere risolto con l'esplorazione dell'intero albero tramite algoritmi quali il MiniMax, che permette di assegnare ad ogni nodo un punteggio attraverso il quale è possibile identificare la mossa migliore; questo non è, però, possibile nella maggior parte dei casi, a causa del grande numero di nodi che compongono l'albero di gioco, il quale risulta, nonostante ottimizzazioni come l'Alpha-Beta Pruning, troppo grande per essere esplorato a fondo.

Bisogna, dunque, sviluppare un algoritmo che permetta al giocatore di:

- Esplorare il maggior numero di nodi possibile entro il tempo limite,
- Fermare l'esplorazione prima dello scadere del tempo,
- Valutare i nodi intermedi dell'albero affidandosi ad una funzione euristica.

SCELTE PROGETTUALI

FUNZIONE DI VALUTAZIONE:

La scelta delle mosse da parte del giocatore si basa sull'algoritmo MiniMax ottimizzato con Alpha-Beta Pruning a profondità limitata: l'esplorazione di un determinato ramo dell'albero si ferma, e lo stato di gioco viene valutato, solo se l'ultimo nodo raggiunto è una foglia, ossia quando il gioco è finito, o se la profondità massima è stata raggiunta; per le foglie, la valutazione avviene assegnando un punteggio di 1000000, in caso di vittoria, -1000000, per le sconfitte, o 0, in caso di pareggio, vengono, inoltre considerate migliori le vittorie (e peggiori le sconfitte) raggiunte a profondità minore, facendo pesare sulle valutazioni la distanza del nodo dalla profondità massima.

La valutazione dei nodi intermedi avviene seguendo un'euristica basata sulla nozione di minaccia: definiamo una minaccia come una particolare configurazione di simboli allineati in modo tale da avvicinare il giocatore a cui appartengono ad una vittoria. Dividiamo le minacce in 3 categorie:

1. Minacce aperte: simboli consecutivi con le entrambe estremità libere,
2. Minacce semi-aperte: simboli consecutivi con una sola estremità libera o simboli separati da un'unica cella libera,
3. Minacce chiuse: simboli consecutive con entrambe le estremità occupate.

Al momento della valutazione, per ogni cella sulla tabella vengono controllate, se esistono, le K-1 celle adiacenti sulle linee in alto a destra, a destra, in basso a destra e in basso rispetto a quella iniziale; le linee vengono esplorate salvando gli stati delle estremità (la casella iniziale e l'ultima analizzata), e contando il numero di celle occupate dal primo giocatore di cui si incontra il simbolo, se durante l'esplorazione si trovano celle occupate dall'altro giocatore, il contatore viene azzerato e l'analisi della linea si interrompe. Ad ogni linea viene, quindi, assegnato un punteggio in base al numero di caselle occupate da uno stesso giocatore, e al tipo di minaccia che la linea costituisce; la valutazione dello stato di gioco è, quindi, il risultato della somma dei punteggi di tutte le minacce sulla tabella.

I punteggi sono assegnati ispirandosi all'euristica generica proposta da Abdoulaye-Houndji-Ezin-Aglin [1] secondo cui le minacce più rilevanti sono quelle di lunghezza K-1, sia aperte che semi-aperte, e K-2 di tipo aperto, in quanto, da queste si possono creare minacce K-1 aperte, che risultano sempre in una vittoria del giocatore che le ha create; le minacce semi-aperte di lunghezza K-2 e quelle di dimensioni minori non sono interessanti, per tanto il punteggio ad esse assegnato è pari al numero di celle occupate. Le minacce del giocatore ricevono un punteggio positivo, mentre quelle dell'avversario uno negativo, i punteggi sono sbilanciati a favore dell'avversario, in maniera tale da rendere, per il giocatore, quasi sempre prioritario bloccarlo, piuttosto che cercare di migliorare la propria posizione.

ITERATIVE DEEPENING:

Al fine di massimizzare la profondità esplorata, il giocatore adotta la tecnica dell'iterative deepening: l'albero di gioco viene attraversato più volte, a partire dalla radice, con limiti di profondità crescenti, ad ogni iterazione si otterrà, così, un punteggio sempre più accurato per le mosse disponibili; l'algoritmo si ferma quando la profondità massima dell'albero viene raggiunta o se il tempo disponibile sta per scadere, in tal caso verrà restituita la mossa migliore trovata dall'ultima iterazione completata.

L'iterative deepening permette di esercitare un miglior controllo sul tempo a disposizione del giocatore senza introdurre costi aggiuntivi: nonostante le esplorazioni ripetute, infatti, il costo computazionale risulta, a parità di profondità massima, paragonabile a quello dell'Alpha-Beta Pruning, poiché la maggior parte dei nodi esplorati si trovano sempre ai livelli più bassi dell'albero.

TABELLE DI TRASPOSIZIONE:

È molto comune che, in un albero di gioco, un nodo si ripeta in rami diversi: uno stesso stato di gioco può infatti essere raggiunto con diverse sequenze di mosse, tali ripetizioni sono dette trasposizioni. Per evitare di rivalutare le trasposizioni, il giocatore impiega una tabella di trasposizione: una tabella hash di grandi dimensioni in cui vengono salvate le informazioni relative agli stati di gioco esplorati; grazie alla tabella di trasposizione siamo, dunque, in grado di evitare ricerche ridondanti, riutilizzando il punteggio calcolato in precedenza.

Talvolta il punteggio salvato nella tabella non può essere utilizzato, in quanto questo potrebbe essere il risultato di una ricerca a profondità minore, e quindi meno accurata, inoltre, la valutazione restituita dall'Alpha-Beta Pruning non corrisponde sempre a quella esatta per il nodo esplorato, ma si tratta di un limite superiore o inferiore ad essa; anche in questi casi, la tabella può comunque velocizzare l'operazione di ricalcolo, infatti se il punteggio salvato non dovesse essere esatto, sarà comunque possibile aggiustare i valori di alpha e beta in base ad esso, aumentando le possibilità di un cutoff, inoltre, nella tabella viene anche salvata la mossa migliore trovata, per lo stato di gioco, nell'ultima ricerca, poiché questa avrà buone probabilità di essere una buona mossa anche per ricerche a profondità maggiori e verrà, quindi, sempre esplorata per prima in caso sia necessario ricalcolare il punteggio.

HASING DI ZOBRIST:

Nella tabella di trasposizione, gli stati di gioco sono salvati come chiavi generate tramite la tecnica dell'hashing di Zobrist, che consiste nel generare una tabella di dimensioni MxNx2 in cui vengono salvati numeri pseudo-randomici a 64 bit: ad ogni casella nella tabella di gioco corrispondono due numeri, uno per ogni simbolo da cui la cella può essere occupata; ogni volta che avviene una modifica ad una cella, la chiave dello stato di gioco viene aggiornata tramite uno XOR tra la chiave attuale e il numero corrispondente allo stato della casella marcata.

GESTIONE COLLISIONI:

COLLISIONI DI CHIAVE: Un problema inerente nell'utilizzo dell'hashing di Zobrist si trova nelle collisioni di chiave, è infatti possibile che due stati di gioco diversi producano una stessa chiave, l'utilizzo di chiavi a 64 bit serve a ridurre al minimo la probabilità che ciò avvenga.

COLLISIONI DI INDICE: Per ridurre il numero di collisioni di indice sulla tabella, viene utilizzato un numero primo come capienza della tabella di trasposizione; la tabella impiega, inoltre, una strategia di gestione delle collisioni basata sulla profondità del nodo esplorato: un elemento salvato nella tabella viene rimpiazzato da un altro elemento con lo stesso indice solo se la profondità di esplorazione del nuovo nodo è maggiore o uguale a quella del nodo precedente.

COSTO COMPUTAZIONALE

Il tempo di calcolo dell'algoritmo implementato è determinato dal tempo massimo assegnato al momento dell'inizializzazione, il ciclo dell'iterative deepening si ferma, infatti, quando il tempo a disposizione sta per scadere; la performance del giocatore dipende, quindi, dal costo computazionale del metodo alphaBeta(): minore il costo, maggiore il numero di cicli di iterative deepening, migliore la valutazione.

Un'analisi completa del costo computazionale risulta complessa, poiché questo è determinato dall'ordine in cui le mosse sono esplorate, infatti, sia n il numero di mosse disponibili, nel caso ottimo l'algoritmo esplora sempre la mossa migliore per ogni livello riportando un costo di $O(\sqrt{n!})$, mentre in quello pessimo non viene mai effettuato alcun cutoff poiché le mosse sono esplorate dalla peggiore alla migliore, quindi il costo risulta pari a O(n!); possiamo, però, affermare che, grazie all'implementazione di una tabella di trasposizione insieme all'iterative deepening, siamo in grado di migliorare l'ordinamento dei nodi dell'albero anche quando i punteggi salvati nella tabella non possono essere utilizzati, poiché la prima mossa esplorata è sempre la migliore trovata nell'ultima esplorazione del nodo corrente, il costo medio della ricerca si avvicina, così, a quello ottimo.

POSSIBILI MIGLIORAMENTI:

Esistono molti algoritmi di decisione su alberi, basati su ottimizzazioni all'Alpha-Beta Pruning, come l'algoritmo MTD(f) o il NegaScout, che potrebbero offrire prestazioni migliori rispetto al MiniMax impiegato dal giocatore.

Un' ulteriore ottimizzazione alle operazioni di calcolo potrebbe, inoltre, essere portata dall'individuazione di simmetrie tra gli stati di gioco, in quanto due nodi simmetrici risultano nella stessa valutazione. Riconoscere queste simmetrie permetterebbe, come per le trasposizioni, di evitare la valutazione di nodi il cui punteggio è stato calcolato in precedenza.

CONCLUSIONI

Sebbene sia ancora ottimizzabile, il giocatore implementato risulta in grado di competere in tutte le istanze testate di (M, N, K)-Game effettuando mosse sufficientemente intelligenti per ogni stato di gioco; le tecniche impiegate permettono l'esplorazione di un grande numero di nodi, risultando in valutazioni abbastanza accurate dello stato di gioco e portando alla scelta di buone mosse nella maggior parte delle situazioni.

Bibliografia

[1] Abdoulaye-Houndji-Ezin-Aglin, Generic heuristic for the mnk-games https://www.cari-info.org/Actes-2018/p276-286.pdf.