ColorShapeAiLinks Agent

Dario Battaglia

Yamiro De Rosa

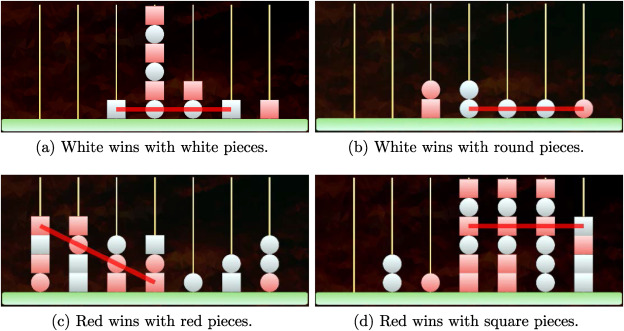
Ingegneria Informatica Magistrale LM

Attività Progettuale Fondamenti di Intelligenza Artificiale LM

# Introduzione

In questa relazione presentiamo un agente IA creato per la competizione di gioco da tavolo ColorShapeLinks, traccia base (board 6x7, time limit per AI: 0.2 secondi) In questo gioco, la scacchiera è una griglia disposta verticalmente e le pedine cadono con la gravità. In modo simile a Connect-4, il primo giocatore a posizionare 4 pedine dello stesso tipo in una sequenza (sia verticalmente che orizzontalmente o diagonalmente), vince. Le pedine sono definite per colore, bianco o rosso, e forma, rotonde o quadrate. Il primo giocatore vince con pedine rotonde o bianche, mentre il secondo giocatore vince con pedine quadrate o rosse. La forma ha la priorità sul colore come condizione di vittoria e, mentre i giocatori giocano solo con pedine del loro colore, possono giocare con pedine di entrambe le forme, rendendo il gioco più interessante e complesso (vedi Figura 1). Il nostro agente cerca le migliori soluzioni utilizzando un approccio di ricerca con un certo numero di ottimizzazioni rispetto ad un Minimax standard per cercare il più a fondo e ampiamente possibile la mossa. La ricerca è guidata da un’euristica il quale peso è stato ottimizzato con un metodo “try&error”.

Figura : Esempi di vittoria nel gioco ColorShapeAiLinks



# Analisi del problema

Il problema di creare un agente AI in grado di essere competitivo con un limite di pensiero di 0.2 secondi è complesso perché richiede una combinazione di abilità, tra cui la capacità di valutare efficacemente le miglior mosse possibili ed impiegare poco tempo per ottenere queste informazioni. Si è deciso quindi di utilizzare varie tecniche (che verranno spiegate più in dettaglio successivamente) che permettono rispetto ad un’implementazione basica del MinMax di ottenere risultati migliori (scendendo più in profondità nell’albero di gioco) in tempi minori.

# Scelte implementative

## Negamax

Il Negamax è una variante dell'algoritmo Minimax che si basa sulle proprietà dei giochi a somma zero con due giocatori. Negamax utilizza la stessa logica di Minimax, ma utilizzando solo una variabile per rappresentare i valori massimi e minimi delle mosse, invece di due. Negamax si basa sulla seguente proprietà: max(a,b) = -min(-a,-b). Ciò permette di snellire l’algoritmo in quanto non vi sono più due rami if (uno per il maximizing player e l’altro per il minimizing player) ma il tutto può essere racchiuso in un’unica linea di codice.

## Potatura Alpha Beta

La potatura alfa-beta è una miglioria che riduce drasticamente il numero di nodi da valutare nell'albero di ricerca dell'algoritmo Negamax. L'algoritmo Negamax con potatura alpha beta introduce due nuove variabili, alpha e beta per limitare le ricerche inutili e ottenere una maggiore efficienza in termini di tempo di esecuzione. La Potatura Alpha-Beta utilizza due valori, chiamati "alpha" e "beta", per tenere traccia del miglior punteggio attuale per il giocatore attuale (alpha) e del valore negativo del peggiore punteggio attuale per il giocatore successivo (beta). Durante l'esplorazione delle possibili mosse, se una mossa porta ad un punteggio peggiore del valore negativo del punteggio già trovato per il giocatore successivo (beta), la potatura alpha-beta interrompe la ricerca in quella direzione poiché sa che quella mossa non può essere la migliore per il giocatore attuale. Allo stesso modo, se una mossa porta ad un punteggio migliore del punteggio già trovato per il giocatore attuale (alpha), la potatura alpha-beta interrompe la ricerca in quella direzione poiché sa che quella mossa non può essere la peggiore per il giocatore successivo.

## Transposition Table e Zobrist Hashing

Le transposition tables sono una tecnica utilizzata in molti algoritmi di ricerca di giochi per ridurre il numero di nodi che devono essere esplorati. Durante l’esplorazione vengono memorizzati i risultati della valutazione dei nodi della ricerca in una tabella di trasposizione, in modo che se un nodo viene visitato nuovamente in seguito, il risultato può essere recuperato dalla tabella invece di dover essere ricalcolato. Nel nostro caso di studio essa è stata implementata come un dizionario in cui la chiave è il valore di hash della configurazione di gioco e dove il valore consiste in un array di oggetti nel quale vengono salvati la profondità (per evitare di prendere in considerazione risultati obsoleti), il valore ottenuto tramite l’euristica ed un flag (Upper, Lower, Exact). Utilizzare questi flag per indicare il tipo di informazione immagazzinata nella transposition table è importante poiché permette di utilizzare in modo sicuro i risultati precedenti. Il flag "exact" indica che il valore di gioco immagazzinato per la configurazione di gioco è esatto e può quindi essere utilizzato per prendere una decisione finale su come giocare la configurazione. I flag “lower” and “upper” servono invece ad indicare che il valore recuperato non è un valore reale. In questo caso lo score recuperato può essere utilizzato per aggiornare i valori alpha (se flag lower) o beta (se flag upper).

Lo Zobrist hashing è una tecnica utilizzata per generare un valore di hash per una configurazione di gioco, e viene utilizzato per memorizzare le configurazioni in una tabella di trasposizione. Il valore di hash viene calcolato utilizzando una serie di numeri casuali, noti come valori di Zobrist, che sono assegnati a ogni posizione della tabella del gioco e ad ogni tipo di pedina. Quando si genera un valore di hash per una configurazione di gioco, si esegue l'operazione XOR sui valori di Zobrist assegnati a ogni pedina nella tabella di gioco. Ciò consente di ottenere un valore di hash univoco per ogni configurazione di gioco, che può quindi essere utilizzato per individuare rapidamente la configurazione in una tabella di trasposizione. Ci sono alcuni problemi comuni che possono verificarsi quando si utilizzano le transposition tables e lo Zobrist hashing:

- Collisioni di hash: a volte, due configurazioni di gioco possono avere lo stesso valore di hash, il che significa che verranno memorizzate nella stessa posizione della tabella di trasposizione. Ciò può causare problemi se si cerca di recuperare il valore per una delle due configurazioni, poiché è possibile che si ottenga il valore dell'altra configurazione e non di quella ricercata. È possibile che avvengano collisioni con lo Zobrist hashing quindi per ridurre la probabilità di collisioni si è deciso di utilizzare valori di hash a 64 bit e di utilizzare 4 valori di Zobrist (una per ogni tipo di pedina) per ognuna delle 42 posizioni disponibili, per un totale di 168 valori hash differenti.

- Risultati obsoleti: nel momento in cui vengono salvati i risultati nella transposition table viene salvato anche la profondità sulla quale si basa lo score di quella particolare configurazione di gioco. Una profondità di '0' significa che abbiamo memorizzato la posizione in base a una valutazione su un nodo foglia (un nodo orizzonte). Una profondità di '1' significa che abbiamo memorizzato la posizione su un nodo frontiera e così via. Quindi, se incontriamo una trasposizione con una profondità memorizzata di '5', sappiamo che la valutazione ad essa collegata si basa su altri 5 livelli di ricerca. Questo è molto importante, perché se stiamo per cercare una posizione con una profondità di 8 e a profondità 3 ci imbattiamo in una trasposizione che abbiamo trovato in una ricerca di profondità 4, non possiamo utilizzare i valori che abbiamo per essa. Questo perché tali valori si basano su una ricerca a 4 livelli, che è molto più debole di quella prevista a 8. In breve, nel momento in cui si ha una hit sulla transposition table se la profondità memorizzata è maggiore o uguale alla profondità attuale della ricerca, allora il valore memorizzato può essere utilizzato, altrimenti non può essere utilizzato.

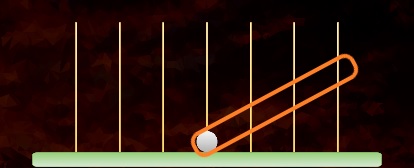
## Iterative Deepening Search

L'idea alla base di IDS è quella di eseguire più passate di ricerca in profondità (nel nostro caso Negamax) aumentando progressivamente la profondità massima di ricerca ad ogni passata, essenzialmente essa performa una ricerca depth-first in una maniera breadth-first. Aumentare gradualmente la profondità di ricerca con l'IDS può aiutare a trovare soluzioni migliori in alcuni casi perché permette di esplorare lo spazio di ricerca. Ad ogni iterazione del Negamax prima della valutazione della board, viene eseguito il calcolo hash della board attuale, estrazione dei valori precedentemente descritti ed aggiornamento di Alpha e Beta se la profondità salvata in memoria è maggiore di quella attuale. Ad ogni iterazione alla fine dell’esecuzione del Negamax, se l’hash della board non è presente nella transposition table, vengono aggiunte le informazioni riguardanti questo stato di gioco.

## Euristica

L’euristica di default implementata nel Minimax è un’euristica che favorisce semplicemente le zone più centrali della griglia di gioco. Nell’euristica da noi implementata, invece, vengono esaminati tutti i possibili insiemi di 4 posizioni consecutive della board (in verticale, in orizzontale ed in obliquo) e viene assegnato un punteggio progressivo in base al numero di sequenze da 1,2 e 3 pedine che possono portare alla vittoria del giocatore. Per ognuna di queste sequenze, nel caso in cui non ci siano pedine che tra loro si “annullino” e che quindi renderebbero la sequenza inutile ai fini della vittoria (e.g. pedina rossa e pedina bianca presenti sulla stessa linea da 4), vengono calcolati il numero di pedine dello stesso colore o della stessa forma presenti ed in base alla numerosità si assegna un valore incrementale che viene sommato (o sottratto nel caso di pedine avversarie) al corrente valore euristico. Oltre i tre valori di base da sommare (o sottrarre) in base della numerosità delle pedine dell’agente (o dell’avversario) nelle varie sequenze da quattro, si è deciso di utilizzare degli altri accorgimenti di modo che determinate sequenze vengano valorizzate (o penalizzate) maggiormente. Data la predominanza della forma della pedina rispetto al colore per la vittoria, nel momento in cui durante il calcolo dell’euristica si effettuano i controlli riguardanti la forma viene assegnato un valore bonus pari a ¼ del valore iniziale (E.g. avere tre pedine dello stesso colore in una sequenza ha un valore di 200, avere tre pedine della stessa forma in una sequenza ha un valore pari a 250). Inoltre, dato che l’algoritmo euristico prende in considerazione tutte le possibili sequenze da quattro, anche quelle che nel momento in cui viene esaminata la board non potrebbero essere utilizzate per la vittoria, viene utilizzato un determinato fattore moltiplicativo per dare più importanza a quelle sequenze che invece potrebbero essere realizzate nel breve periodo (vedi esempio nella figura 2).

Figura : la sequenza da 4 diagonale mostrata in figura, non è di immediata realizzazione in quanto per poter posizionare le pedine in quelle posizioni, vi è la necessità che le posizioni sottostanti siano tutte occupate.



# Risultati

# Confronto performance: tempi di calcolo delle mosse

In questa sezione verranno confrontati i tempi necessari alla scelta della mossa, il confronto è stato effettuato tra due agenti Iterative Deepening Search che utilizzano l’euristica di default e l’euristica implementata da noi e il Minimax presente di default nel progetto. Il log completo delle partite si può trovare al seguente link: https://github.com/dbattaglia97/AI-Agent-ColorShapeAiLinks/blob/main/Report%20partite/Tempi-Torneo-depth3.txt.

### Profondità massima 3

|  |  |  |
| --- | --- | --- |
|  | N. MOSSE | TEMPO MEDIO MOSSA |
| MINIMAXD3 - Default | 318 | 0.2289 |
| IDS\_D3- Euristica Default | 318 | 0.0789 |
| IDS\_D3 – Nostra Euristica | 528 | 0.1107 |

### Profondità massima 5

In questo ultimo confronto, è stata impostata la profondità massima di ricerca a 5 e si può notare maggiormente i vantaggi nell’utilizzo dell’agente creato in confronto all’utilizzo di un Minimax. Il log completo delle partite si può trovare al seguente link: <https://github.com/dbattaglia97/AI-Agent-ColorShapeAiLinks/blob/main/Report%20partite/Tempi%20Torneo-depth5.txt> .

|  |  |  |
| --- | --- | --- |
|  | N. MOSSE | TEMPO MEDIO MOSSA |
| MINIMAXD5 - Default | 266 | 32.2794 |
| IDS\_D5- Euristica Default | 318 | 1.9310 |
| IDS\_D5 – Nostra Euristica | 528 | 3.2374 |

# Confronto performance: scelta delle mosse migliori

Facendo sfidare l’IDS da noi implementato, sia con l’euristica di default che con quella appositamente creata per il caso di studio, è possibile notare come l’euristica creata sembra essere più consistente rispetto a quella di default.

### Limite di tempo 0.2 secondi e no upper bound su profondità massima

In questo caso avendo un limite di 0.2 secondi, il Minmax di default perderebbe a tavolino in quanto non rispetterebbe i limiti di tempo poiché non effettua alcun controllo sui limiti di tempo. Il confronto è stato fatto quindi implementando in IDS\_D10000T3 l’euristica del minmax di default.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | VITTORE  NOSTRA EURISTICA | VITTORIE EURISTICA DEFAULT | NUMERO PARTITE GIOCATE | Win Rate Nostra Euristica |
| Time Limit 0.2 | 43 | 7 | 50 | 86% |
| Depth 3- NO time limit | 50 | 0 | 50 | 100% |
| Depth 5- NO time limit | 50 | 0 | 50 | 100% |

Il log completo delle partite con Time Limit 0.2 si può trovare al seguente link: <https://github.com/dbattaglia97/AI-Agent-ColorShapeAiLinks/blob/main/Report%20partite/TimeLimit0_2.txt>.  
Il log completo delle partite con depth 3 si può trovare al seguente link: <https://github.com/dbattaglia97/AI-Agent-ColorShapeAiLinks/blob/main/Report%20partite/Depth3-Minimax-IDS3_1.txt>.

Il log completo delle partite con depth 3 si può trovare al seguente link: <https://github.com/dbattaglia97/AI-Agent-ColorShapeAiLinks/blob/main/Report%20partite/Depth5-IDS5_1-IDS5_3.txt> .