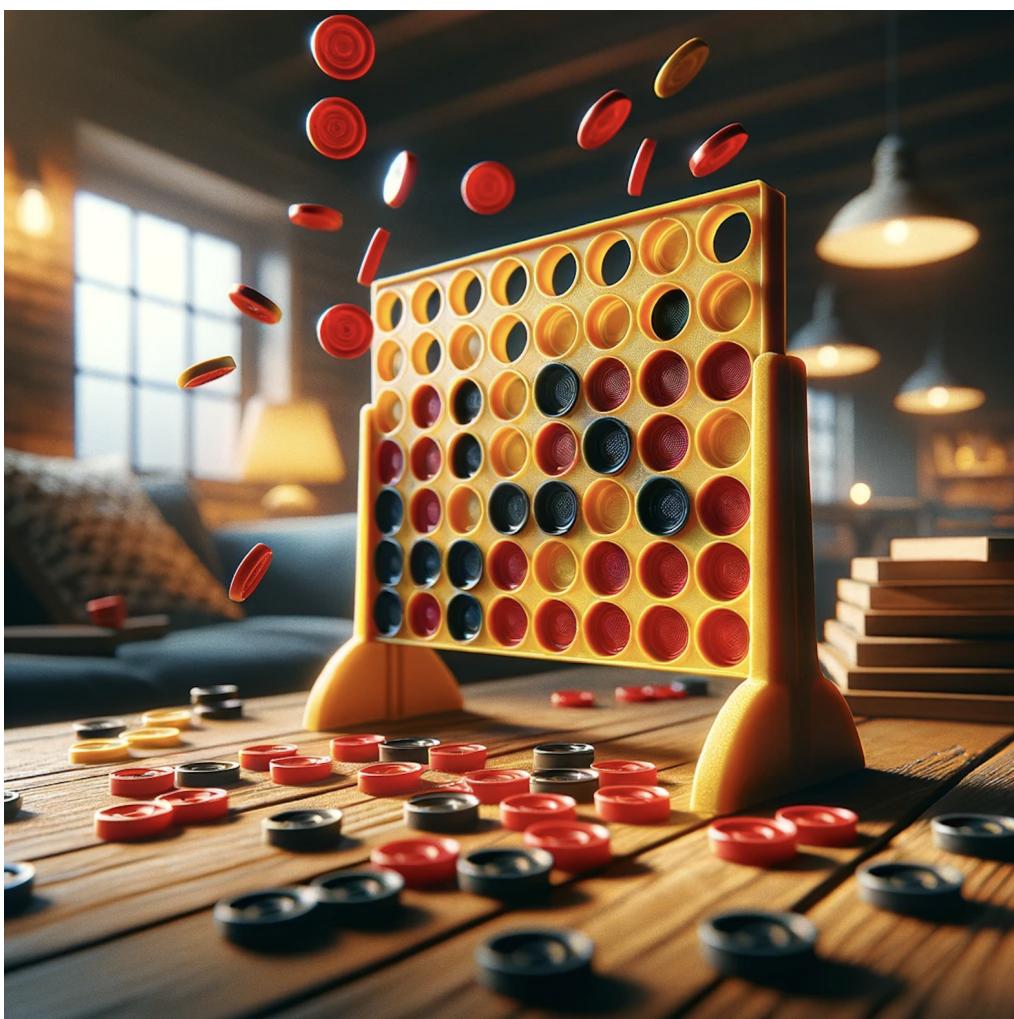


SouthPasadena bot for Connect X - Progetto di Algoritmi e Strutture Dati

Università di Bologna



Michele Garavani
Numero di matricola: 0001079937

Indice

| | | |
|----------|--|----------|
| 1 | Introduzione | 2 |
| 2 | Implementazione | 2 |
| 2.1 | L'algoritmo | 2 |
| 2.1.1 | Minimax | 2 |
| 2.1.2 | Alpha-Beta pruning | 3 |
| 2.1.3 | Iterative Deepening | 4 |
| 2.1.4 | Ordine di esplorazione delle colonne | 5 |
| 2.2 | Tabelle di trasposizione e Zobrist Hashing | 5 |
| 2.2.1 | Zobrist Hashing | 5 |
| 2.2.2 | Tabelle di trasposizione | 6 |
| 2.3 | Heuristic Score | 6 |
| 3 | Cenni sulla complessità computazionale | 7 |
| 4 | Conclusioni | 7 |
| 5 | Bibliografia | 7 |

1 Introduzione

Connect Four è un gioco da tavolo che vede affrontarsi due giocatori, ognuno rappresentato da un colore. I due giocatori, a turno, lasciano cadere gettoni del loro colore in una griglia a sei file e sette colonne sospesa verticalmente. I pezzi cadono direttamente secondo la gravità, occupando lo spazio più basso disponibile all'interno della colonna. L'obiettivo del gioco di è formare per primi una linea orizzontale, verticale o diagonale di quattro pedine. Si tratta quindi di un tipo di gioco M, N, X (7, 6, 4) con posizionamento limitato dei pezzi.

Connect X è semplicemente la versione generalizzata di tale gioco, ovvero le dimensioni della tabella ed il numero di gettoni da connettere sono arbitrarie.

Il progetto consiste nell'implementare un giocatore software che sia in grado di giocare mosse ottimali in Connect X, ovvero limitare il più possibile le possibilità di vincita dell'avversario, e massimizzare le proprie. Il giocatore deve essere in grado di scegliere la mossa migliore nella posizione corrente senza superare un limite fissato di tempo.

L'interfaccia per il progetto è stata fornita dai docenti (*CXPlayer*), e basandosi su essa lo scopo è di costruire una classe che implementa tale interfaccia e che restituisce la migliore colonna su cui effettuare la propria mossa nella posizione corrente di gioco.

La mia classe, *SouthPasadena*, compie esattamente quanto detto, ovvero implementa l'interfaccia *CXPlayer*, e restituisce la mossa che reputa sia la migliore entro il limite prestabilito di tempo. La classe *SouthPasadena* fa uso di algoritmi nell'ambito della game theory. Per sfruttare al meglio il vincolo temporale, *SouthPasadena* usufruisce di strategie di ottimizzazione di risorse. *SouthPasadena* sfrutta anche numerose strutture dati, in particolar modo gli arrays.

2 Implementazione

2.1 L'algoritmo

2.1.1 Minimax

L'algoritmo alla base della logica di SouthPasadena è il Minimax. Tale algoritmo cerca di minimizzare la possibile perdita per un giocatore, assumendo che l'avversario stia massimizzando il proprio guadagno. Attraverso un'analisi ricorsiva delle mosse possibili, l'algoritmo sceglie la mossa che porta al risultato migliore possibile, considerando che l'avversario non commetterà errori.

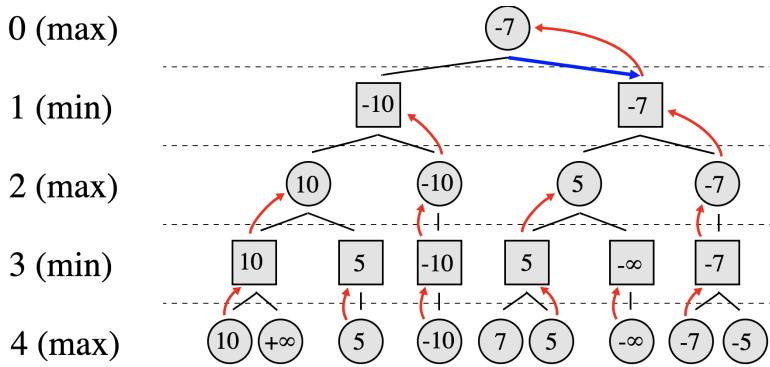


Figura 1: Un semplice esempio di un albero con Minimax dove ogni giocatore ha al massimo due mosse.

Questo processo crea un albero di chiamate ricorsive, iniziando dalla posizione attuale come radice. Ogni ramo dell’albero rappresenta una mossa legale del giocatore, seguita dai possibili contraccolpi dell’avversario, e così via, fino a raggiungere, idealmente, un risultato finale per ogni raggiungibile sequenza di mosse. Se nella posizione finale vince il giocatore, la valutazione è ∞ . Se vince l’avversario, la valutazione è $-\infty$. Se invece la posizione è un pareggio allora la valutazione è 0. Successivamente, i risultati vengono riportati indietro al nodo di partenza, considerando che l’avversario giochi in modo perfetto. In figura 1, un esempio di un game tree dove il giocatore cerca di massimizzare il punteggio.

Per questioni di costo computazionale di tempo, non sempre tutti i nodi di chiamate ricorsive dell’algoritmo possono essere visitati, dato che il loro incremento è esponenziale. Per tale ragione, l’algoritmo Minimax include anche una profondità massima di visita. Ovvero, se ci sono altre mosse disponibili a partire da una posizione alla profondità massima prestabilita dell’algoritmo, verrà assegnato a tale posizione un valore euristico che non è per forza ∞ , $-\infty$ o 0. Nella sezione 2.3, la funzione euristica verrà spiegata in dettaglio.

2.1.2 Alpha-Beta pruning

Un miglioramento significativo dell’algoritmo Minimax è rappresentato dalla tecnica di alpha-beta pruning. Questo metodo riduce il numero di nodi valutati nell’albero di gioco senza influire sul risultato finale. La potatura alfa-beta si basa su due valori, alpha e beta, che rappresentano rispettivamente il minimo punteggio che il giocatore massimizzante è assicurato di ottenere e il massimo punteggio che il giocatore minimizzante è assicurato di ottenere. Essi sono inizializzati rispettivamente a $-\infty$ e ∞ . Durante la traversata dell’albero, se viene identificato un nodo il cui valore è peggiore del miglior risultato già trovato lungo il percorso per il giocatore corrente

(maggiore di beta per il minimizzante, minore di alfa per il massimizzante), quel ramo può essere "potato", ovvero ignorato, poiché non influenzerà il risultato finale. Questo processo permette di escludere grandi porzioni dell'albero di ricerca, rendendo l'algoritmo molto più efficiente in termini di tempo di calcolo, specialmente in giochi con un grande spazio di ricerca. L'algoritmo 1 mostra il Minimax con alpha-beta pruning.

Algorithm 1 Minimax with Alpha-Beta Pruning

```

1: function MINIMAX(currentPos, depth, maximizingPlayer,  $\alpha$ ,  $\beta$ )
2:   if depth = 0 or someone won or the game is a draw then
3:     return heuristic value of currentPos
4:   end if
5:   if maximizingPlayer then
6:     value  $\leftarrow -\infty$ 
7:     for each possible move from currentPos do
8:       value  $\leftarrow \max(\text{value}, \text{MINIMAX}(\text{currentPos.child}, \text{depth} -$ 
      1, FALSE,  $\alpha, \beta))$ 
9:       if value  $\geq \beta$  then
10:        break                                 $\triangleright$  Beta cut-off
11:       end if
12:        $\alpha \leftarrow \max(\alpha, \text{value})$ 
13:     end for
14:     return value
15:   else
16:     value  $\leftarrow +\infty$ 
17:     for each possible move from currentPos do
18:       value  $\leftarrow \min(\text{value}, \text{MINIMAX}(\text{currentPos.child}, \text{depth} -$ 
      1, TRUE,  $\alpha, \beta))$ 
19:       if value  $\leq \alpha$  then
20:         break                                 $\triangleright$  Alpha cut-off
21:       end if
22:        $\beta \leftarrow \min(\beta, \text{value})$ 
23:     end for
24:     return value
25:   end if
26: end function

```

2.1.3 Iterative Deepening

Per migliorare l'efficienza della ricerca, SouthPasadena utilizza anche una semplice istanza di Iterative Deepening, che permette di esplorare l'albero di gioco a profondità (parametro *depth* nell'algoritmo Minimax) crescenti. Invece di eseguire una singola ricerca fino a una profondità massima fissata

dall'inizio, l'approfondimento iterativo inizia con una profondità di 1 e la incrementa progressivamente con ogni iterazione del ciclo di ricerca, finché c'è tempo. Questo metodo offre il vantaggio di assicurare che venga trovata la migliore mossa possibile nel tempo disponibile, dato che fornisce soluzioni intermedie che possono essere utilizzate se il tempo si esaurisce prima di completare l'esplorazione a una profondità maggiore.

2.1.4 Ordine di esplorazione delle colonne

L'algoritmo Minimax visita tutte le mosse possibili nella posizione corrente. Per rendere tale processo il più efficiente possibile, ed avere maggiore probabilità di "potare" una parte dell'albero il prima possibile, SouthPasadena usufruisce di un metodo *orderColumns()* che ordina le colonne in base all'ordine in cui saranno esplorate. Tale ordine prioritizza le mosse in colonne centrali e poi si allontana sempre di più dal centro fino ad arrivare alle colonne alle estremità, che saranno le ultime ad essere esplorate. Il motivo per tale ordine è che le colonne centrali possono includere un maggior numero di combinazioni vincenti e quindi c'è una maggiore probabilità che una mossa in una colonna centrale sia considerabile. Ovviamente, solo le colonne non piene, ovvero quelle dove è effettivamente possibile fare una mossa, saranno considerate.

2.2 Tabelle di trasposizione e Zobrist Hashing

Le tabelle di trasposizione e lo Zobrist Hashing sono due tecniche avanzate utilizzate per ottimizzare gli algoritmi di ricerca negli spazi di gioco. Queste tecniche migliorano significativamente l'efficienza della ricerca senza sacrificare l'accuratezza del risultato.

2.2.1 Zobrist Hashing

Lo Zobrist Hashing è un metodo per generare un identificatore univoco (hash) per ogni configurazione possibile del gioco, utilizzando un approccio basato su operazioni XOR di valori pre-generati casualmente. Prima di tutto, il metodo *initZobrist()* crea un numero random associato ad ogni casella della tabella di Connect X per ogni possibile occupazione di tale casella (ovvero del giocatore 1, del giocatore 2 o libera). Lo Zobrist hash di una configurazione del gioco è calcolato tramite il metodo *computeZobristHash()*, che attraversa lo stato corrente del gioco e applica operazioni XOR tra i valori nella tabella Zobrist e lo stato delle celle sulla tabella. Questa tecnica garantisce che configurazioni diverse producano hash differenti con alta probabilità, consentendo un accesso efficiente e affidabile alle valutazioni memorizzate nelle tabelle di trasposizione.

2.2.2 Tabelle di trasposizione

Le tabelle di trasposizione sono utilizzate per memorizzare le valutazioni di stati del gioco precedentemente analizzati. Ogni entry nella tabella di trasposizione, rappresentata dalla classe *TranspositionEntry*, include il valore Zobrist hash dello stato del gioco, la valutazione dell’analisi e la profondità dell’analisi. Durante l’esecuzione dell’algoritmo Minimax si controlla quindi se la posizione da calcolare è già stata calcolata e se la sua profondità è maggiore o uguale a quella della posizione da calcolare. In caso positivo, viene direttamente ritornato il valore euristico di tale posizione. In caso contrario, dopo che Minimax ha calcolato il valore, tale posizione con il suo valore viene salvata nella tabella per usi futuri. L’uso di tabelle di trasposizione nel contesto dell’algoritmo Minimax, come implementato in SouthPasadena, accelera notevolmente la ricerca, consentendo di esplorare l’albero di gioco più in profondità in minor tempo.

2.3 Heuristic Score

Nell’uso dell’algoritmo Minimax, nonostante le ottimizzazioni effettuate, non è possibile esplorare l’intero albero di gioco fino ai suoi stati finali rimanendo nei limiti di tempo. Qui entra in gioco il concetto di punteggio euristico: una valutazione approssimativa dello stato del gioco che aiuta a stimare le probabilità di vittoria da una data configurazione senza necessità di analizzare ogni possibile esito futuro. Questo punteggio euristico viene assegnato ai nodi intermedi dell’albero, permettendo all’algoritmo di prendere decisioni intelligenti anche quando l’analisi completa dell’albero non è finalizzabile.

La mia funzione euristica considera prima di tutto gli stati finali, che restituiscono una valutazione infinitamente positiva o negativa a dipendenza di chi ha vinto nella posizione corrente. Nel caso di un pareggio la valutazione è 0.

Negli stati non terminali, SouthPasadena considera tutte le potenziali vittorie per ogni giocatore, dando **esponenzialmente** più peso a quelle che hanno un numero maggiore di gettoni che potrebbero risultare in una vittoria. Per fare ciò, vengono prima valutate tutte le combinazioni verticali che potrebbero essere ultimate. In modo simile, si guarda quante combinazioni in orizzontale sono disponibili. Infine, si controllano le combinazioni per le diagonali ascendenti e discendenti. Ogni potenziale combinazione diagonale che potrebbe portare ad una vittoria vale il doppio, in quanto le combinazioni diagonali tendono ad essere più strategiche. Ogni combinazione plausibile per SouthPasadena viene sommata alla valutazione, mentre invece ogni combinazione plausibile per l’avversario viene sottratta alla valutazione. Il valore finale considerando il bilancio di tutte le combinazioni di entrambi i giocatori è la valutazione della posizione corrente di gioco.

Questo approccio euristico permette all'algoritmo di prendere decisioni intelligenti in scenari di gioco complessi dove un'esplorazione esaustiva di tutte le mosse possibili è impraticabile.

3 Cenni sulla complessità computazionale

Il costo dell'algoritmo Minimax è influenzato da diversi fattori, dato che la sua implementazione include diverse ottimizzazioni che possono influenzare in modo significativo le prestazioni dell'algoritmo. Ad esempio: alpha-beta pruning e le tabelle di trasposizione. Il costo dell'algoritmo Minimax senza ottimizzazioni e senza limite di tempo (quindi con profondità infinita -cercherebbe tutto l'albero finché non si arriva solo a stati terminali-) sarebbe, nel caso pessimo $O(N^{M \cdot N})$, con una matrice di gioco con M righe e N colonne. Con alpha-beta pruning, il costo diventerebbe nel best-case scenario $O(\sqrt{N^{M \cdot N}})$. Aggiungendo anche le tabelle di trasposizione, sebbene esse non modifichino direttamente la complessità computazionale teorica dell'algoritmo Minimax con alpha-beta pruning, esse migliorano significativamente le prestazioni pratiche. Inoltre, considerando i limiti di tempo, l'algoritmo non sarà mai in grado di analizzare l'intero albero di ricorsione di Minimax (tranne per matrici di gioco molto piccole), e per tale ragione la profondità massima toccata giocherà anche un ruolo nella considerazione della complessità computazionale.

4 Conclusioni

SouthPasadena risulta essere un giocatore adeguato e competente nel trovare la mossa migliore in Connect X. Ciò è possibile grazie ad algoritmi strategici e tecniche di ottimizzazione.

Personalmente, mi sono molto divertito a creare questo progetto in quanto mi ha per la prima volta davvero immerso nel programmare.

5 Bibliografia

- blog.gamesolver.org
- chessprogramming.org
- È anche stata usata un'intelligenza artificiale generativa per guidarmi nelle scelte concettuali