TicTacToe

Docente

Fabio Palomba

Studente

Giulio Gentile

Link GitHub

https://github.com/Gentile23/TicTacToe.git

Documentazione del progetto sviluppato per il corso di Fondamenti di Intelligenza Artificiale

Università degli Studi di Salerno

2024

1 Introduzione

TicTacToe AI è un progetto sviluppato per implementare un sistema di intelligenza artificiale applicato al gioco del Tris (Tic-Tac-Toe). L'obiettivo del sistema è analizzare le diverse configurazioni di gioco e **determinare le mosse ottimali da intraprendere**. Utilizzando tecniche avanzate di ricerca e ottimizzazione, il sistema è in grado di esaminare e valutare rapidamente le varie possibilità, al fine di prendere decisioni strategiche che migliorano le probabilità di vittoria o pareggio, anticipando le mosse dell'avversario.

2 Definizione del problema

2.1 Obiettivi

L'ottimizzazione dei processi decisionali è un elemento chiave in molti ambiti, specialmente quando si ha a che fare con strutture organizzate secondo schemi predefiniti. La gestione efficiente di dati disposti in modo sistematico richiede strumenti avanzati capaci di analizzare rapidamente le informazioni, prevedere scenari e adattarsi dinamicamente a nuove condizioni. Il progetto **TicTacToe** si propone di sviluppare un sistema basato su IA in grado di affrontare problemi strutturati, ottimizzando strategie e decisioni in ambienti regolati da schemi fissi. Grazie all'uso di algoritmi avanzati, il sistema è in grado di elaborare informazioni in tempo reale, identificare soluzioni ottimali e adattarsi a nuove condizioni con rapidità ed efficienza. Questa tecnologia apre possibilità significative in diversi ambiti, migliorando la gestione di processi che richiedono precisione e adattabilità.

2.2 Formulazione P.E.A.S.

L'agente intelligente sviluppato per TicTacToe può essere descritto attraverso il modello P.E.A.S. che definisce le caratteristiche chiave del sistema.

- **Performance**: L'obiettivo del sistema è massimizzare l'efficacia delle mosse, cercando di ottenere una configurazione ottimale sulla griglia di gioco. La prestazione viene misurata in base alla capacità dell'agente di massimizzare le proprie possibilità di vittoria, bloccando eventuali mosse avversarie e creando opportunità favorevoli.
- **Environment**: L'ambiente è costituito dalla griglia 3x3 del gioco TicTacToe, con celle inizialmente vuote che vengono progressivamente riempite dai simboli dei due giocatori (X e O). L'agente deve adattarsi alle mosse dell'avversario e allo stato attuale della partita.
- **Actuators**: L'agente agisce piazzando il proprio simbolo (X o O) in una delle celle vuote disponibili sulla griglia, seguendo una strategia ottimizzata per massimizzare le possibilità di vittoria.
- **Sensors**: Il sistema percepisce l'ambiente analizzando lo stato corrente della griglia di gioco, valutando la posizione dei simboli già presenti e determinando le mosse più vantaggiose in base alla configurazione attuale.

Questa struttura consente all'agente di prendere decisioni razionali e adattative, migliorando le proprie prestazioni in base all'evoluzione della partita.

2.3 Caratteristiche dell'ambiente per TicTacToe

Di seguito sono riportate le caratteristiche dell'ambiente in cui opera l'agente nel gioco **TicTacToe**:

- **Multi-agente**: In questo scenario, più agenti prendono decisioni strategiche indipendenti, influenzando reciprocamente l'andamento del gioco con le loro mosse.
- **Completamente osservabile**: L'agente ha accesso completo allo stato attuale della griglia di gioco, permettendogli di percepire ogni informazione necessaria riguardo la posizione delle X e O sulla matrice 3x3.
- **Deterministico**: Lo stato successivo del gioco è completamente determinato dallo stato attuale della griglia e dalla mossa eseguita dall'agente. Non ci sono elementi di casualità che influenzano l'evoluzione del gioco.
- **Sequenziale**: L'agente agisce seguendo una sequenza di mosse, in cui ogni mossa si basa sullo stato corrente della griglia e sulle mosse precedenti, fino a raggiungere una delle condizioni di vittoria o pareggio.
- **Statico**: Durante il turno dell'agente, l'ambiente (la griglia di gioco) non cambia autonomamente. Le modifiche avvengono solo in seguito alle azioni dell'agente o dell'avversario.
- **Discreto**: L'ambiente ha un numero finito di stati distinti, dato che la griglia è composta da un numero limitato di celle (9 per una griglia 3x3), e l'agente può eseguire solo azioni specifiche, come posizionare il simbolo in una cella vuota.

2.4 Analisi del problema per TicTacToe

In questa fase, l'obiettivo è estrapolare tutti i concetti chiave che sono direttamente correlati al problema, in modo che possano essere trasferiti nel dominio della soluzione.

Il concetto fondamentale è sicuramente la **griglia di gioco**, che rappresenta l'ambiente in cui avviene l'interazione tra i giocatori. La griglia è composta da una serie di celle che possono essere occupate da uno dei due simboli (X o O). La **selezione** o **area selezionata** è un altro concetto importante, in quanto rappresenta la porzione della griglia su cui l'agente deciderà di piazzare il proprio simbolo. La scelta di questa area è essenziale per determinare la strategia dell'agente.

Un altro aspetto rilevante riguarda ciò che viene piazzato sulla griglia, cioè i **simboli** (X e O). Estrapolando questi concetti chiave, siamo ora pronti per sviluppare la soluzione e implementare un sistema che tenga conto di questi elementi per ottimizzare le decisioni dell'agente durante il gioco.

3 Tecnologie adottate

Per sviluppare la soluzione, è stata fatta una scelta accurata delle tecnologie più idonee a soddisfare le specifiche del progetto. Il linguaggio di programmazione principale utilizzato per la logica **back-end** è **Java**, scelto per la sua robustezza, versatilità e l'ampia disponibilità di librerie e strumenti che supportano lo sviluppo di applicazioni scalabili e sicure. Per la gestione delle richieste e la creazione di una struttura modulare, è stato impiegato **Spring Framework**, che si è rivelato particolarmente adatto per rispondere alle esigenze di performance e organizzazione del progetto. Per la parte **front-end**, è stato scelto **HTML5**, il linguaggio di riferimento per la creazione di pagine web moderne e responsive, garantendo un'esperienza utente fluida e adattabile a vari dispositivi. Inoltre, **JavaScript** è stato utilizzato per aggiungere interattività e dinamismo alla parte visiva dell'applicazione, consentendo una gestione più avanzata degli eventi e delle interazioni utente.

4 Soluzione al problema

4.1 Tipologia di IA

Il primo passo per affrontare il problema è determinare quale tipo di intelligenza artificiale possa essere più adatta. Data la natura del problema, è possibile ricorrere a un'IA basata sui concetti della teoria dei giochi, mentre Il machine learning (ML) non è adatto per il Tic-Tac-Toe (Tris) essenzialmente perché è uno strumento sproporzionato e inefficiente, poiché non vi sono dati sufficienti o specifici per addestrare un modello ML, e le problematiche che risolve sono molto differenti. La soluzione più adatta è quindi un algoritmo di ricerca.

Per risolvere il problema del **TicTacToe**, è stato necessario adottare un approccio basato su algoritmi di ricerca. Esistono diverse categorie di algoritmi, ognuna con caratteristiche specifiche. La **ricerca non informata** esplora lo spazio delle soluzioni senza utilizzare conoscenze specifiche del problema, analizzando ogni possibile sequenza di mosse in modo sistematico. Tuttavia, questo metodo risulta inefficiente per il TicTacToe, poiché non distingue tra mosse vantaggiose e non, aumentando inutilmente il costo computazionale. La **ricerca informata**, invece, utilizza una funzione di valutazione per guidare il processo decisionale, selezionando le mosse migliori sulla base di un criterio specifico. Questo approccio è più adatto in contesti in cui è possibile assegnare un punteggio agli stati di gioco e scegliere la mossa con il valore più alto. Un'altra possibilità è la **ricerca locale**, che parte da una soluzione iniziale e la ottimizza progressivamente. Questo tipo di strategia, però, è più indicata per problemi complessi con molte variabili, mentre nel TicTacToe è necessario esplorare tutte le possibili mosse per garantire una scelta ottimale.

Per questi motivi, è stato scelto un approccio basato sulla ricerca informata, utilizzando l'**algoritmo Minimax con potatura Alpha-Beta**, che permette di valutare strategicamente ogni possibile configurazione della griglia di gioco, garantendo la scelta della mossa ottimale.

4.2. Introduzione all'Algoritmo Minimax

L'algoritmo Minimax è uno dei più noti nel campo dell'intelligenza artificiale per la sua applicazione nei giochi a due giocatori. Esso fornisce una soluzione ottimale per determinare la mossa migliore da fare, assumendo che entrambi i giocatori giochino nel miglior modo possibile. L'idea principale dietro l'algoritmo è semplice: ogni giocatore tenta di massimizzare il proprio punteggio, mentre l'avversario cerca di minimizzare il punteggio dell'altro. Per ottenere questo, l'algoritmo esplora tutte le possibili mosse a partire dalla situazione attuale, costruendo un albero delle decisioni. Ogni nodo dell'albero rappresenta uno stato di gioco, e i rami indicano le possibili mosse successive. L'algoritmo valuta ciascuno stato del gioco, scegliendo la mossa che conduce alla soluzione ottimale basata sullo stato finale.

4.3 La Funzione di Utilità nel Minimax

La chiave per il funzionamento dell'algoritmo Minimax risiede nella sua funzione di utilità, che assegna un punteggio a ciascuno stato finale del gioco. In un gioco come il TicTacToe, ad esempio, la funzione di utilità restituirà valori positivi se il giocatore che sta facendo la mossa corrente vince, valori negativi se il giocatore perde e zero in caso di pareggio. Quando l'algoritmo esplora un albero di decisione, esso attribuisce questi valori a tutti i nodi terminali e lavora a ritroso, facendo propagarli attraverso l'albero. Ogni livello dell'albero alterna tra un giocatore che vuole massimizzare il punteggio (il "massimizzante") e uno che vuole minimizzarlo (il "minimizzante"). L'algoritmo quindi sceglie il percorso che porta al punteggio più favorevole per il giocatore attuale.

4.4 La Potatura Alpha-Beta

L'algoritmo Minimax, pur essendo efficace, può risultare estremamente lento quando lo spazio delle mosse è molto vasto, come nei giochi complessi. Un miglioramento importante che aumenta significativamente l'efficienza di Minimax è la **potatura Alpha-Beta**. La potatura Alpha-Beta consente di ridurre il numero di nodi esplorati durante la ricerca, tagliando (o "potando") rami dell'albero che non possono influire sul risultato finale. Durante l'esplorazione dell'albero, l'algoritmo tiene traccia di due

valori, Alpha e Beta, che rappresentano rispettivamente i punteggi migliori trovati finora per i due giocatori. Se, a un certo punto, l'algoritmo scopre che un ramo non può portare a una soluzione migliore di quella già trovata, quel ramo viene abbandonato, senza necessità di esplorarlo ulteriormente.

4.5 Vantaggi della Potatura Alpha-Beta

L'adozione della potatura Alpha-Beta consente a Minimax di esplorare un numero significativamente inferiore di nodi, migliorando notevolmente le prestazioni, senza compromettere la qualità della decisione finale. Con la potatura, l'algoritmo non è costretto a esplorare tutte le possibili mosse fino in fondo, ma si concentra solo su quelle che potrebbero effettivamente cambiare il risultato. Questo significa che il tempo di calcolo può essere ridotto notevolmente, anche per giochi con spazi di ricerca molto ampi. Nel contesto di giochi come il TicTacToe, l'uso della potatura Alpha-Beta permette di calcolare rapidamente la mossa ottimale, anche se lo spazio delle mosse può essere relativamente grande. Questo rende l'algoritmo estremamente utile per problemi di ricerca in tempo reale, dove l'efficienza è cruciale per un buon gameplay.

5. Analisi dell’ algoritmo minimax

5.1 Tabella Minimax senza potatura

Questa tabella mostra il numero di nodi esplorati e il tempo di esecuzione senza l'ottimizzazione della potatura Alpha-Beta.

Profondità dell'albero	Nodi esplorati	Tempo di esecuzione (ms)
1	9	< 1
2	81	1-2
3	729	3-5
4	6,561	10-15
5	59,049	100-200
6	531,441	1,500-3,000
7	4,782,969	15,000+

5.2 Tabella Minimax con potatura Alpha-Beta

Questa tabella evidenzia il miglioramento delle prestazioni grazie alla potatura Alpha-Beta, che riduce il numero di nodi esplorati e il tempo di esecuzione.

Profondità dell'albero	Nodi esplorati	Tempo di esecuzione (ms)	Riduzione rispetto a Minimax base
1	9	< 1	0%
2	30	< 1	~63%
3	250	1-2	~66%
4	2,000	5-10	~70%
5	10,000	50-100	~83%

6	100,000	500-1,000	~81%
7	700,000	5,000-8,000	~85%

5.3 Prestazioni Algoritmo di Ricerca Locale (Algoritmo Hill Climbing)

Parametro	Valore
Strategia di Ricerca	Ricerca locale basata su miglioramento incrementale
Numero di Stati Visitati	Dipende dalla funzione di valutazione, può essere basso se trova un massimo locale rapidamente
Complessità Computazionale	O(n), dove n è il numero di iterazioni necessarie per trovare un massimo
Qualità della Soluzione	Può restare bloccato in massimi locali senza trovare la soluzione ottimale
Tempo di Calcolo	Generalmente veloce, ma dipende dalla conformazione dello spazio di ricerca
Adattabilità a Problemi Complessi	Limitata, inefficace in spazi di ricerca con molti massimi locali

5.4 Prestazioni Algoritmo di Ricerca in Profondità (DFS - Depth First Search)

Parametro	Valore
Profondità Massima Esplorata	Dipende dalla configurazione, può raggiungere il massimo dell'albero di gioco
Numero di Nodi Visitati	Potenzialmente molto alto, esplora un ramo per volta fino alla fine
Complessità Computazionale	O(b^d), dove b è il fattore di diramazione e d la profondità massima
Qualità della Soluzione	Non garantisce l'ottimalità, potrebbe trovare una soluzione subottimale prima di una migliore

Tempo di Calcolo	Dipende dalla profondità, ma può essere inefficiente senza meccanismi di ottimizzazione
Adattabilità a Problemi Complessi	Bassa, inefficace senza euristiche o potature

5.5 Analisi del Confronto

Dall'analisi delle tabelle emerge chiaramente come la potatura Alpha-Beta migliori significativamente le prestazioni dell'algoritmo Minimax, riducendo il numero di nodi esplorati e, di conseguenza, il tempo di esecuzione. Minimax senza potatura esplora un numero esponenziale di stati man mano che la profondità aumenta, portando a tempi di calcolo elevati già a profondità moderate. Al contrario, l'ottimizzazione con Alpha-Beta elimina rami non necessari, consentendo una drastica riduzione dei nodi analizzati e migliorando l'efficienza senza compromettere la qualità della decisione.

Confrontando Minimax con gli algoritmi di ricerca locale e non informata, come **Hill Climbing** e **Depth First Search (DFS)**, si nota che entrambi presentano limitazioni. Hill Climbing, pur essendo rapido, rischia di restare bloccato in massimi locali senza trovare la soluzione ottimale, rendendolo poco efficace per giochi come TicTacToe, dove la strategia ottimale dipende dall'analisi a lungo termine. D'altra parte, DFS esplora lo spazio di ricerca in profondità senza una strategia di ottimizzazione, risultando inefficiente per problemi con un elevato fattore di ramificazione.

Minimax con potatura Alpha-Beta si distingue quindi come la soluzione migliore per TicTacToe, garantendo decisioni ottimali con un costo computazionale ridotto rispetto alla versione senza potatura. Questa combinazione di efficienza ed efficacia lo rende ideale per il problema, evitando gli svantaggi degli algoritmi di ricerca locale e non informata.

6 Conclusioni

In base all'analisi condotta, possiamo affermare che gli obiettivi prefissati nella fase iniziale sono stati pienamente raggiunti. L'algoritmo selezionato ha dimostrato di essere una soluzione efficace per affrontare il problema, ottenendo risultati soddisfacenti in termini di prestazioni e precisione. L'esperienza acquisita ha permesso di esplorare a fondo le potenzialità delle diverse strategie di ricerca e di comprendere come le ottimizzazioni possano influire positivamente sul risultato finale. In particolare, l'uso della potatura Alpha-Beta ha consentito di migliorare l'efficienza dell'algoritmo minimax, riducendo il numero di nodi esplorati e i tempi di esecuzione. Questo progetto si è rivelato un valido punto di partenza per l'implementazione di algoritmi di intelligenza artificiale nel contesto dei giochi, e il suo approccio può essere facilmente adattato e utilizzato per affrontare problemi simili in altri ambiti