

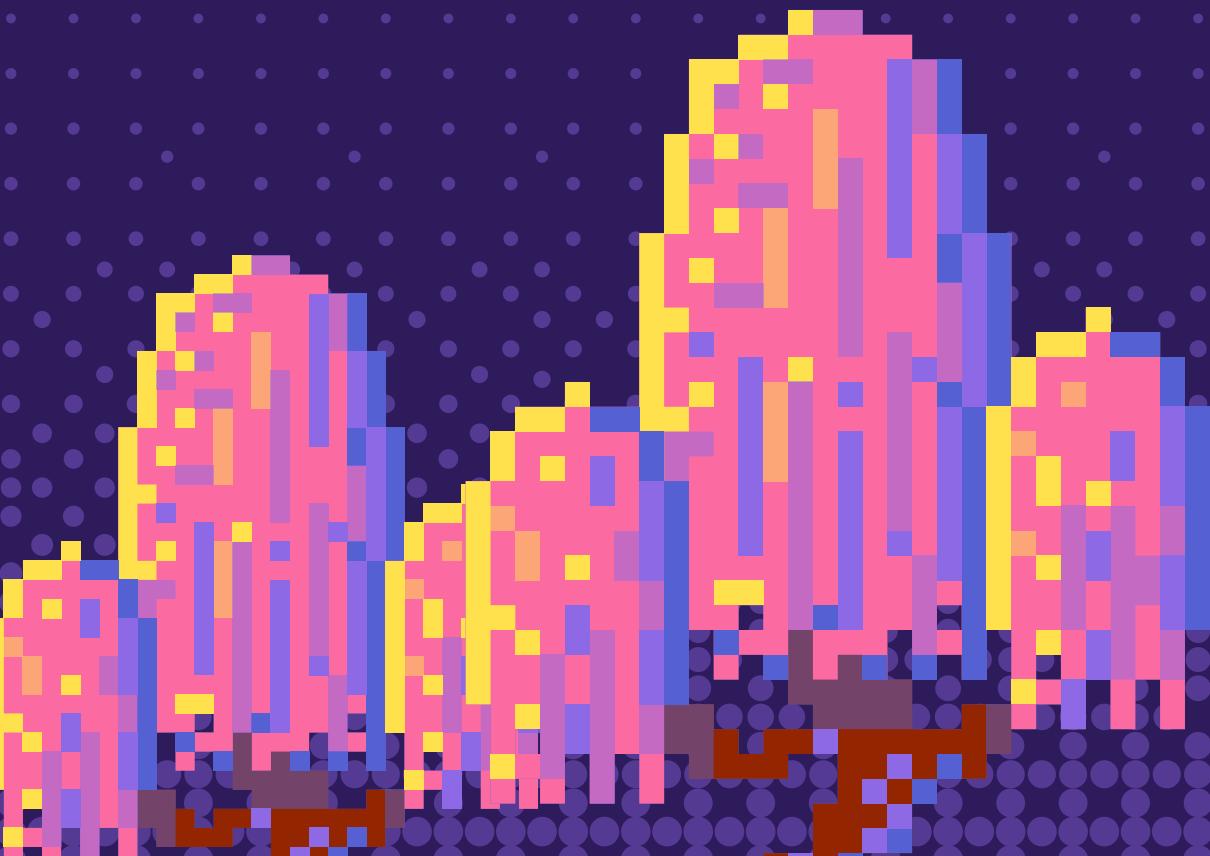


# TRIS, UN PÒ DI STORIA

Chiunque, almeno una volta nella vita, ha giocato a Tris, magari scarabocchiando una griglia su un foglio di carta durante una lezione noiosa o sfidando un amico su un angolo di un quaderno. Questo semplice ma avvincente gioco ha origini antichissime: forme simili al Tris risalgono all'Impero Romano, dove venivano incise su pietre o legni. Nel tempo, il gioco si è evoluto fino a diventare la versione moderna che tutti conosciamo oggi, diffusa in tutto il mondo grazie alla sua immediatezza e alla sua profondità strategica nascosta dietro poche e semplici regole.

## REGOLE:

- Due giocatori (X e O) giocano su una griglia 3x3.
- Vince chi allinea tre simboli in fila, colonna o diagonale.
- Se tutte le caselle sono piene senza un vincitore, la partita finisce in pareggio.





# PERCHÉ IL TRIS È INTERESSANTE PER L'IA?

Il Tris è uno dei giochi più semplici nel panorama dei giochi strategici a due giocatori, ma proprio questa semplicità lo rende perfetto per sperimentare con l'intelligenza artificiale. Essendo un gioco a informazione perfetta, dove entrambi i giocatori hanno piena visibilità dello stato della partita, è possibile prevedere tutte le mosse future e analizzarle matematicamente.

Inoltre, il numero di possibili partite è relativamente limitato rispetto a giochi più complessi come gli Scacchi o il Go. Il numero totale di partite uniche nel Tris è di circa 255.168, mentre il numero di configurazioni possibili della griglia è 19.683. Questo rende il gioco risolvibile attraverso algoritmi di ricerca

# ANALISI DEL PROBLEMA CON PEAS

Per descrivere l'interazione tra l'IA e l'ambiente di gioco, possiamo utilizzare il modello PEAS (Performance measure, Environment, Actuators, Sensors):

## • PERFORMANCE MEASURE (MISURA DELLE PRESTAZIONI):

l'IA deve cercare di vincere la partita o almeno di evitare la sconfitta. Un buon algoritmo non dovrebbe mai perdere contro un avversario umano o contro un altro giocatore che gioca in modo casuale.

## • ENVIRONMENT (AMBIENTE):

è rappresentato dalla griglia 3x3 su cui si svolge il gioco. Questo ambiente è completamente osservabile, statico e deterministico, il che significa che ogni stato è visibile e non cambia se non a seguito di una mossa.

## • ACTUATORS (ATTUATORI):

l'IA interagisce con l'ambiente posizionando un simbolo "X" o "O" in una delle caselle disponibili. Questo è l'unico modo in cui può influenzare lo stato del gioco.

## • SENSORS (SENSORI):

l'IA è in grado di leggere lo stato della griglia in ogni momento, analizzando la disposizione dei simboli per decidere la mossa migliore in base alla configurazione attuale.

# CARATTERISTICHE DELL'AMBIENTE DI GIOCO

01 Completamente osservabile

02 Deterministico

03 Sequentiale

02 Multi-agente

05 Statico

02 Discreto



# ALGORITMO SCELTO: MINIMAX CON POTATURA ALPHA-BETA

Per realizzare un'intelligenza artificiale capace di giocare in modo perfetto, ho deciso di utilizzare l'algoritmo Minimax, una tecnica di ricerca decisionale ampiamente utilizzata nei giochi a due giocatori a somma zero.

Il principio alla base di Minimax è piuttosto semplice: ogni giocatore cerca di massimizzare il proprio punteggio e minimizzare quello dell'avversario. Nel contesto del Tris, l'IA analizza tutte le possibili mosse future, costruendo un albero decisionale in cui ogni nodo rappresenta una configurazione della griglia. Durante la ricerca, l'algoritmo assume che l'avversario giocherà sempre in modo ottimale e selezionerà la mossa che minimizza il punteggio dell'IA.

Il funzionamento di Minimax segue una logica ricorsiva. Partendo dallo stato attuale della partita, l'IA genera tutte le possibili mosse e valuta ogni scenario futuro in termini di vittoria, sconfitta o pareggio. Se in un determinato nodo della ricerca si raggiunge una configurazione finale (ovvero una vittoria per X, una vittoria per O o un pareggio), viene assegnato un punteggio:

- **+1 se l'IA vince la partita.**
- **-1 se l'avversario vince la partita.**
- **0 se la partita termina in pareggio.**

Procedendo a ritroso, Minimax propaga questi punteggi fino alla radice dell'albero decisionale, permettendo all'IA di scegliere la mossa iniziale che garantisce il miglior risultato possibile.

Tuttavia, nonostante la relativa semplicità del Tris, la generazione completa dell'albero delle mosse può essere computazionalmente costosa. In scenari più complessi, come nel caso degli Scacchi, il numero di stati possibili cresce in modo esponenziale, rendendo proibitivo un approccio basato sulla pura esplorazione di tutte le possibilità. Per questo motivo, ho introdotto un'ottimizzazione fondamentale: **la potatura Alpha-Beta**.

# OTTIMIZZAZIONE CON POTATURA ALPHA-BETA

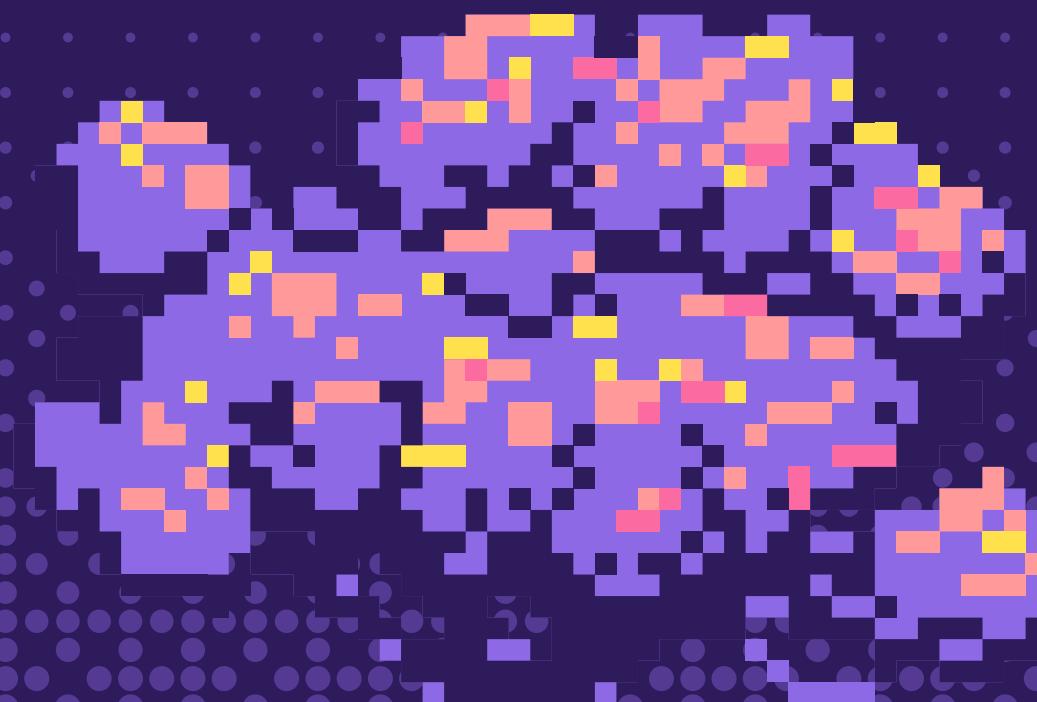
La potatura Alpha-Beta è una tecnica che riduce il numero di nodi esplorati senza alterare il risultato finale dell'algoritmo Minimax. Il suo obiettivo è quello di eliminare dalla ricerca le mosse che, in base all'analisi precedente, risultano irrilevanti ai fini della decisione ottimale.

Il principio di funzionamento è basato sull'idea di stabilire due valori limite durante la ricerca:

- Alpha rappresenta il miglior punteggio che il giocatore massimizzante (l'IA) può garantire fino a quel momento.
- Beta rappresenta il miglior punteggio che il giocatore minimizzante (l'avversario) può ottenere.

Se durante l'esplorazione Minimax si raggiunge un nodo in cui si scopre che un avversario razionale non sceglierrebbe mai quella mossa, allora è inutile continuare a valutare gli stati successivi: il ramo viene "potato", ovvero escluso dalla ricerca.

Questa ottimizzazione permette di ridurre drasticamente il numero di stati esplorati, migliorando l'efficienza dell'algoritmo e rendendo l'IA molto più veloce nelle decisioni. Nel caso del Tris, la potatura Alpha-Beta riduce il numero di nodi da circa 255.000 a poche decine di migliaia, garantendo una risposta in tempo reale anche su hardware modesti.



# CONFRONTO CON ALTRE STRATEGIE DI RICERCA

Per sviluppare l'IA avrei potuto utilizzare altre strategie di ricerca, ma ciascuna presentava delle limitazioni rispetto a Minimax (ricerca informata):

- **RICERCA LOCALE**

analizza solo configurazioni vicine per trovare una soluzione ottimale, ma può bloccarsi in ottimi locali senza considerare l'intera partita.

- **RICERCA NON INFORMATA:**

esplora tutti gli stati possibili senza un criterio strategico, risultando inefficiente e troppo lenta per il problema.

# IMPLEMENTAZIONE

Per l'implementazione dell'IA ho utilizzato il framework **Spring Boot** per sviluppare il backend in **Java**, garantendo un'architettura scalabile e ben strutturata. Il backend gestisce la logica di gioco, elaborando le mosse e restituendo la risposta in tempi rapidi.

Per il frontend ho scelto **HTML e CSS**, affiancandoli a JavaScript per rendere l'interfaccia più dinamica e interattiva. Gli script in **JavaScript** si occupano dell'aggiornamento della griglia di gioco, della gestione degli eventi utente e della comunicazione con il backend.

# RISULTATI

Dopo aver completato l'implementazione, ho condotto una serie di test per valutare le prestazioni dell'IA. L'agente è stato messo alla prova in numerose partite contro giocatori umani e avversari con strategie casuali. I risultati hanno confermato l'efficacia dell'approccio adottato: l'IA ha dimostrato di essere estremamente solida, evitando sconfitte e garantendo risposte rapide in ogni situazione di gioco.

Ad oggi quest'AI ha un percentuale di sconfitta pari allo 0%.



A pixel art scene featuring a dark purple background with a dotted pattern. At the top, there are two layers of stylized clouds in shades of pink, yellow, and purple. Small yellow star-like sparkles are scattered throughout the scene. The bottom layer features a city skyline silhouette in yellow and purple pixels.

GRAZIE PER  
L'ATTENZIONE