SVILUPPO DI UN AGENTE INTELLIGENTE PER ULTIMATE TIC-TAC-TOE

IMPLEMENTAZIONE IN PROLOG CON INTERFACCIA GRAFICA XPCE E ALGORITMO MINIMAX

Autore: Luca Belardinelli
Corso: Intelligenza Artificiale 2025/2026

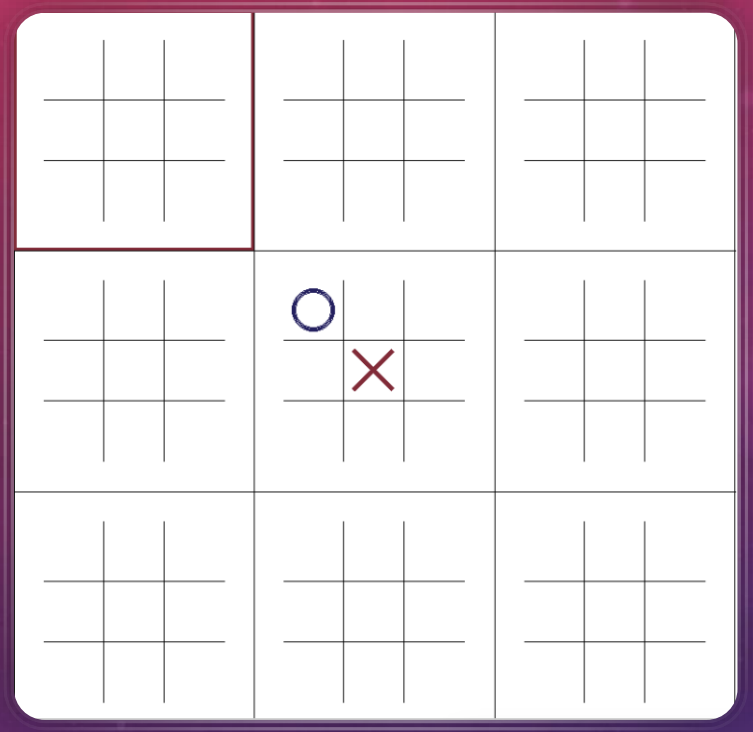
OBBIETTIVI DEL PROGETTO

1. **Sviluppo del Gioco:** Implementazione delle regole complesse di "Ultimate Tic-Tac-Toe" (struttura ricorsiva) utilizzando la logica dichiarativa.
2. **Interfaccia Grafica:** Creazione di un'interfaccia interattiva tramite la libreria **XPCE** per visualizzare lo stato del gioco e gestire l'input utente.
3. **Intelligenza Artificiale:** Implementazione di un agente avversario basato sull'algoritmo **Minimax con Alpha-Beta Pruning**.
4. **Gestione della Difficoltà:** Modulazione della profondità di ricerca dell'albero di gioco per creare livelli di sfida variabili (Facile, Medio, Difficile).

COME FUNZIONA IL GIOCO?

- Ultimate Tic-Tac-Toe espande il classico gioco del fila tre trasformandolo in una sfida di strategia su due livelli. La tabella è costituita da una "Meta-Griglia" principale 3x3 in cui ogni singola casella non è vuota, ma contiene a sua volta un'altra intera partita di tris più piccola, per un totale di 9 scacchiere separate. L'obiettivo finale non è semplicemente allineare tre simboli, ma conquistare tre "micro-scacchiere" consecutive. Quando un giocatore vince su una griglia piccola, quella zona viene marcata col suo simbolo gigante sul tabellone principale, contribuendo alla vittoria globale della partita.

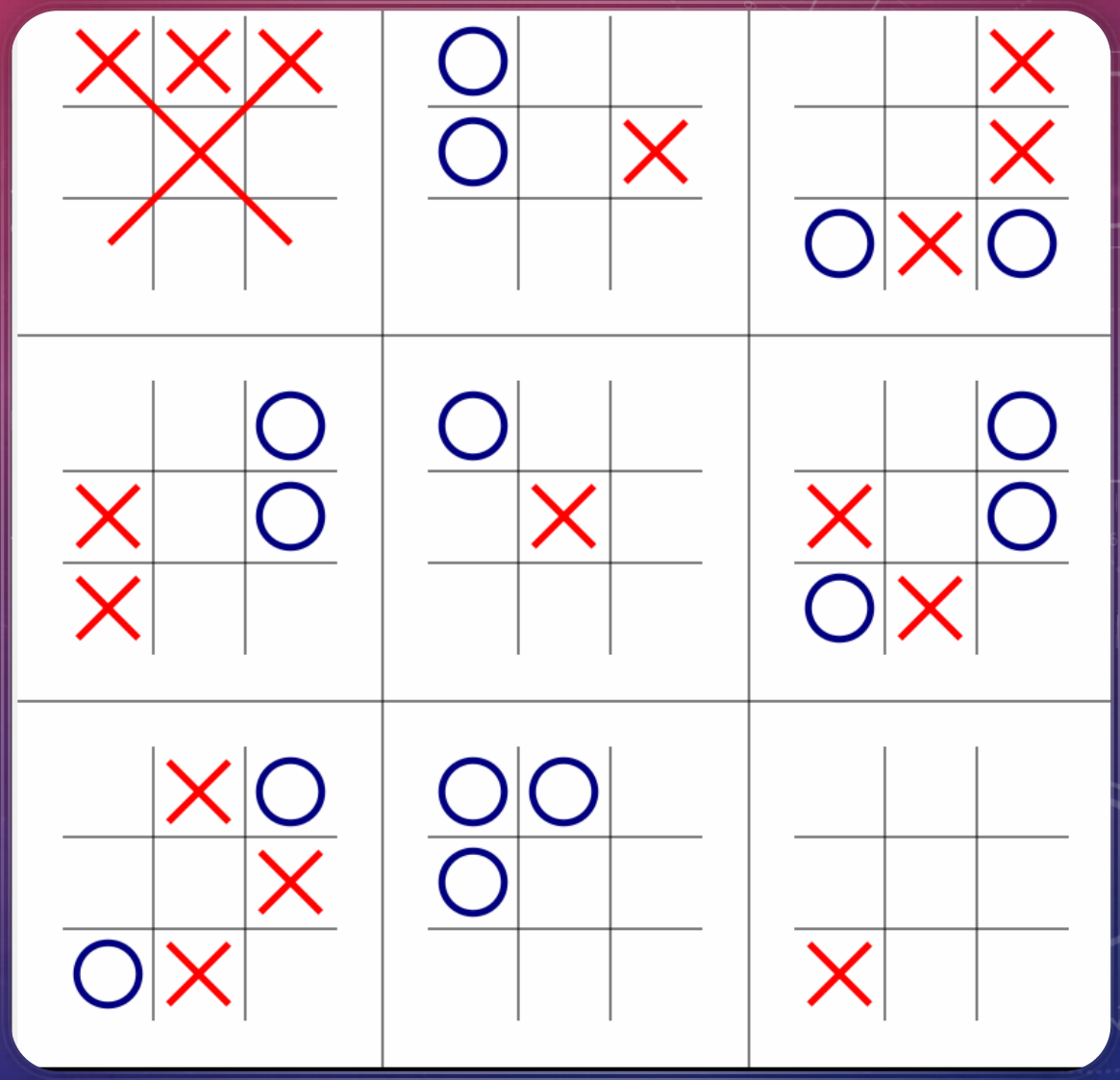
LA STRATEGIA DEL VINCOLO



- Il cuore del gioco risiede in una regola che collega tutte le griglie tra loro: i giocatori non possono scegliere liberamente dove giocare. La mossa effettuata in una cella determina obbligatoriamente la board successiva in cui dovrà giocare l'avversario. Ad esempio, se si inserisce una X nella cella al centro di una qualsiasi griglia piccola, l’avversario sarà costretto a giocare nella griglia situata al centro del tabellone principale. Questo trasforma ogni mossa in una decisione tattica: non si cerca solo di fare punto , ma si cerca anche di inviare l'avversario in una scacchiera dove è svantaggiato o dove non può vincere.

GIOCARE SU UNA BOARD COMPLETA

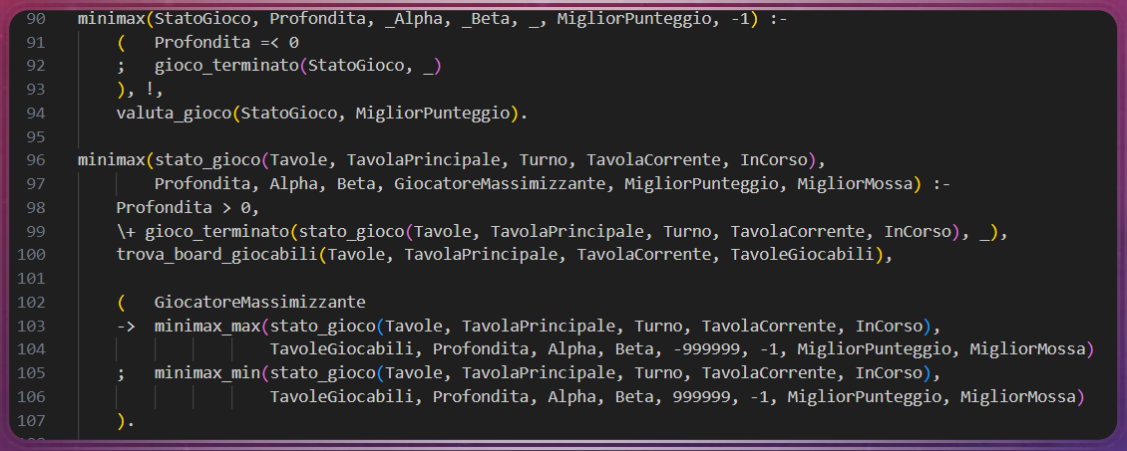
La regola del vincolo prevede un'eccezione critica: qualora un giocatore indirizzi l'avversario verso un board già completata (vinta o piena), il vincolo decade. In questo scenario, l'avversario ottiene un 'Jolly', acquisendo la libertà di giocare in qualsiasi board disponibile. Tale condizione, che conferisce un vantaggio tattico significativo, è segnalata visivamente dall'assenza del reticolo rosso di evidenziazione, indicando che l'intera scacchiera è sbloccata.



ARCHITETTURA SOFTWARE (PROLOG E XPCE)

- Il progetto è sviluppato interamente in SWI-Prolog, sfruttando la programmazione logica per gestire la complessità delle regole ricorsive e dello stato del gioco. L'interfaccia grafica è realizzata nativamente tramite la libreria **XPCE**, che permette l'interazione diretta via mouse e la visualizzazione dinamica delle mosse. Per il giusto funzionamento del gioco, è stata implementata un'Intelligenza Artificiale basata sull'algoritmo **Minimax con Alpha-Beta pruning**; l'algoritmo calcola un albero di possibili futuri scenari valutando non solo le vittorie immediate, ma anche il vantaggio posizionale garantito dalla regola di spostamento dell'avversario.

ALGORITMO MINIMAX



```
90  minimax(StatoGioco, Profondita, _Alpha, _Beta, _, MigliorPunteggio, -1) :-
91      (   Profondita =< 0
92      ;   gioco_terminato(StatoGioco, _)
93      ), !,
94      valuta_gioco(StatoGioco, MigliorPunteggio).
95
96  minimax(stato_gioco(Tavole, TavolaPrincipale, Turno, TavolaCorrente, InCorso),
97          Profondita, Alpha, Beta, GiocatoreMassimizzante, MigliorPunteggio, MigliorMossa) :-
98      Profondita > 0,
99      \+ gioco_terminato(stato_gioco(Tavole, TavolaPrincipale, Turno, TavolaCorrente, InCorso), _),
100     trova_board_giocabili(Tavole, TavolaPrincipale, TavolaCorrente, TavoleGiocabili),
101
102     (   GiocatoreMassimizzante
103     ->  minimax_max(stato_gioco(Tavole, TavolaPrincipale, Turno, TavolaCorrente, InCorso),
104                     TavoleGiocabili, Profondita, Alpha, Beta, -999999, -1, MigliorPunteggio, MigliorMossa)
105     ;   minimax_min(stato_gioco(Tavole, TavolaPrincipale, Turno, TavolaCorrente, InCorso),
106                     TavoleGiocabili, Profondita, Alpha, Beta, 999999, -1, MigliorPunteggio, MigliorMossa)
107     ).
```

L'algoritmo Minimax rappresenta il cuore decisionale dell'agente. Si tratta di un metodo ricorsivo utilizzato nella teoria dei giochi per determinarne la mossa ottimale.

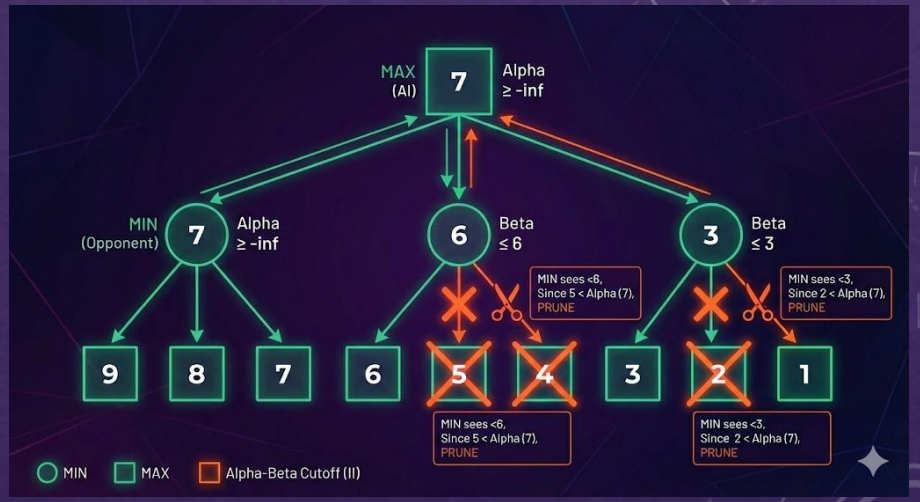
Il funzionamento si basa sulla simulazione dell'intero albero delle possibili mosse future. L'agente (definito come giocatore "Massimizzante") ipotizza di eseguire una mossa e simula la risposta dell'avversario (giocatore "Minimizzante"), assumendo che quest'ultimo giochi sempre al meglio delle sue possibilità. Questo processo di simulazione prosegue fino a raggiungere una configurazione terminale di vittoria o, come nel nostro caso specifico, un limite di profondità prefissato.

Una volta raggiunta la fine della simulazione, i valori di utilità vengono "propagati all'indietro" (back-propagation) verso la radice dell'albero: l'agente sceglierà quindi il ramo che conduce allo stato con il valore più alto garantito, minimizzando al contempo il rischio di sconfitta.

ALPHA-BETA PRUNING

Poiché l'Ultimate Tic-Tac-Toe presenta un fattore di ramificazione molto elevato, l'esplorazione completa dell'albero di gioco risulta computazionalmente impossibile in tempi ragionevoli. Per questo motivo, all'algoritmo Minimax è stata applicata l'ottimizzazione Alpha-Beta Pruning.

Questa tecnica non altera il risultato finale della decisione, ma riduce drasticamente il numero di nodi valutati "tagliando" i rami dell'albero che si dimostrano irrilevanti. Il sistema mantiene due valori soglia: Alpha (il miglior punteggio che l'IA ha già assicurato a se stessa) e Beta (il miglior punteggio che l'avversario ha già assicurato a se stesso). Nel momento in cui l'algoritmo scopre una mossa che peggiora la situazione rispetto a un'alternativa già trovata in precedenza, interrompe immediatamente l'analisi di quel ramo specifico.



GESTIONE DELLA DIFFICOLTÀ E PROFONDITÀ DI RICERCA

Il livello di competenza dell'agente è determinato dalla profondità dell'albero di ricerca esplorato dall'algoritmo Minimax. Maggiore è la profondità, maggiore è la capacità dell'IA di anticipare le conseguenze delle proprie azioni. Nella prossima slide sono riportati i livelli di profondità del progetto.

```
163         leggi_livello_difficolta(Livello),
164         (   Livello = facile
165         ->  Profondita = 1,
166             format('IA gioca a livello FACILE (profondità 1)~n')
167         ;   Livello = medio
168         ->  Profondita = 3,
169             format('IA gioca a livello MEDIO (profondità 3)~n')
170         ;   % Modalità difficile con ottimizzazione
171             trova_board_giocabili(Tavole, TavolaPrincipale, TavolaCorrente, TavoleGiocabili),
172             lunghezza(TavoleGiocabili, NumTavole),
173             (   CelleVuote < 15
174             ->  Profondita = 3,
175                 format('IA DIFFICILE: profondità 3 (endgame)~n')
176             ;   NumTavole >= 7
177             ->  Profondita = 3,
178                 format('IA DIFFICILE: profondità 3 (troppe board)~n')
179             ;   Profondita = 5,
180                 format('IA DIFFICILE: profondità 5~n')
181             )
182         ),
```

GESTIONE DELLA DIFFICOLTÀ E PROFONDITÀ DI RICERCA

🟢 Livello Facile (Profondità 1)

- Comportamento Reattivo: L'IA valuta esclusivamente lo stato attuale della tabella.
- Strategia: Gioca la mossa che massimizza il punteggio immediato (approccio *Greedy*), senza considerare le possibili contromosse dell'avversario.

🟠 Livello Medio (Profondità 3)

- Pianificazione Tattica: L'IA prevede fino a 3 mezzi-turni nel futuro (Mossa IA → Risposta Utente → Controrisposta IA).
- Capacità: Riesce a individuare trappole semplici, bloccare tentativi di vittoria diretti dell'avversario e costruire piccole sequenze vincenti.

🔴 Livello Difficile (Profondità Adattiva 5-3)

- Strategia Avanzata: L'IA punta a una profondità di 5 mosse, elaborando strategie complesse a lungo termine.
- Ottimizzazione Dinamica: Per evitare rallentamenti (freeze) quando il *Branching Factor* è troppo alto (molti board aperti), l'algoritmo riduce automaticamente la profondità a 3, bilanciando perfettamente intelligenza e velocità di esecuzione.

LA FUNZIONE DI VALUTAZIONE EURISTICA

Dato il limite di profondità dell'albero di ricerca, l'algoritmo utilizza una funzione euristica per valutare gli stati intermedi di gioco. Il sistema attribuisce punteggi specifici a configurazioni vantaggiose come il controllo del centro, la conquista degli angoli o la formazione di coppie. La somma di questi valori genera un punteggio complessivo: confrontando i punteggi delle diverse alternative, l'IA identifica la mossa che le garantisce il vantaggio maggiore.

```
46  valuta_coppie(Tavola, Bonus) :-
47      findall(B, (
48          (   controlla_coppia(Tavola, [0,1,2], B)
49          ;   controlla_coppia(Tavola, [3,4,5], B)
50          ;   controlla_coppia(Tavola, [6,7,8], B)
51          ;   controlla_coppia(Tavola, [0,3,6], B)
52          ;   controlla_coppia(Tavola, [1,4,7], B)
53          ;   controlla_coppia(Tavola, [2,5,8], B)
54          ;   controlla_coppia(Tavola, [0,4,8], B)
55          ;   controlla_coppia(Tavola, [2,4,6], B)
56          )
57      ), BonusLista),
58      sumlist(BonusLista, Bonus).
59
60  controlla_coppia(Tavola, [I1, I2, I3], Bonus) :-
61      elemento_n(I1, Tavola, V1),
62      elemento_n(I2, Tavola, V2),
63      elemento_n(I3, Tavola, V3),
64      (   V1 =:= 1, V2 =:= 1, V3 =:= 0 -> Bonus = -30
65      ;   V1 =:= -1, V2 =:= -1, V3 =:= 0 -> Bonus = 30
66      ;   Bonus = 0
67      ).
```

```
10
11    peso_posizione(0, 0.2).
12    peso_posizione(1, 0.17).
13    peso_posizione(2, 0.2).
14    peso_posizione(3, 0.17).
15    peso_posizione(4, 0.22).
16    peso_posizione(5, 0.17).
17    peso_posizione(6, 0.2).
18    peso_posizione(7, 0.17).
19    peso_posizione(8, 0.2).
20
21    moltiplicatore_board(0, 2.0).
22    moltiplicatore_board(1, 1.2).
23    moltiplicatore_board(2, 2.0).
24    moltiplicatore_board(3, 1.2).
25    moltiplicatore_board(4, 2.5).
26    moltiplicatore_board(5, 1.2).
27    moltiplicatore_board(6, 2.0).
28    moltiplicatore_board(7, 1.2).
29    moltiplicatore_board(8, 2.0).
```

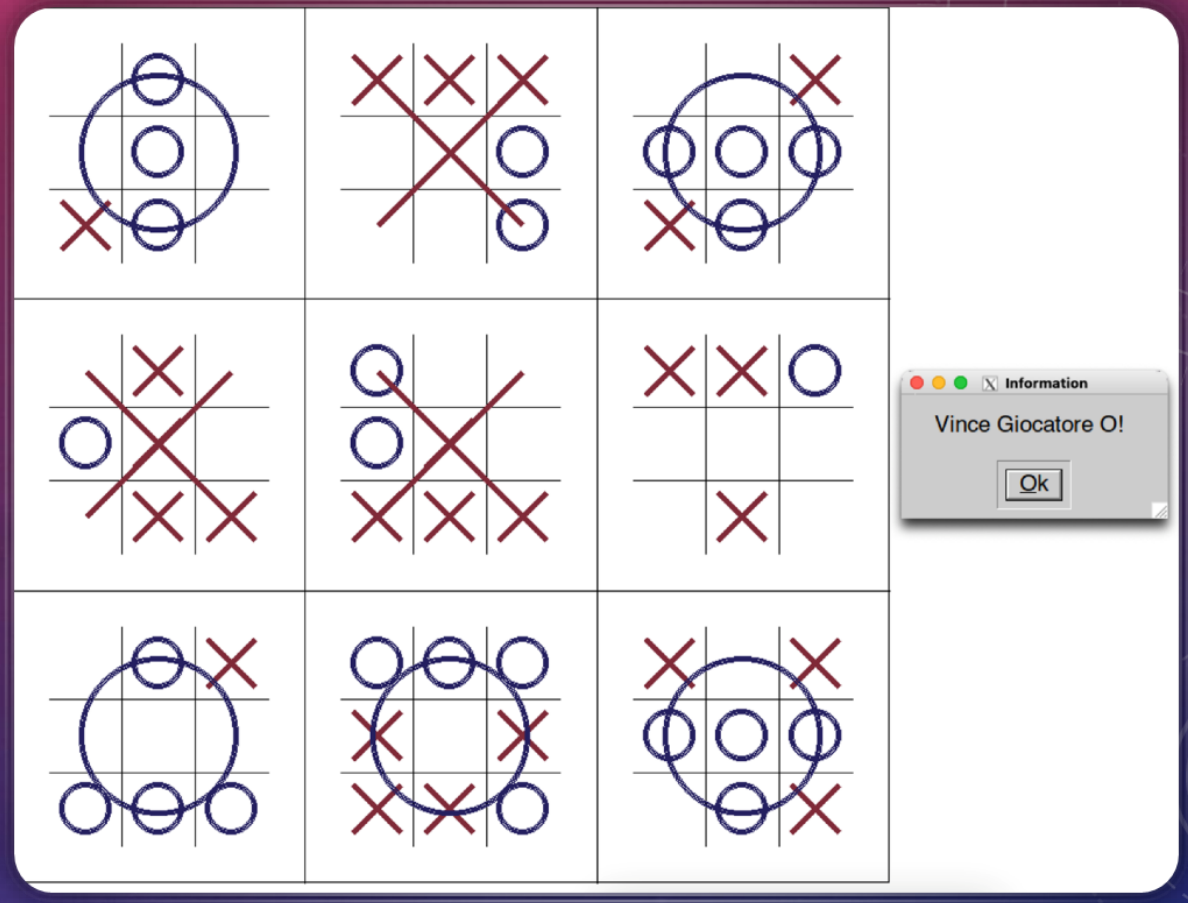
STRATEGIA E PESI NEL PROGETTO

Per quanto riguarda la strategia e i pesi applicati al gioco nel progetto, l'IA valuta lo stato della tabella basandosi su tre criteri fondamentali:

- Valore Posizionale (peso posizione): Il sistema riconosce che le celle non hanno tutte lo stesso valore strategico. Viene data massima priorità alla casella centrale con un peso di 0.22, poiché offre più opportunità di connessione, seguita dagli angoli (0.20) e infine dai lati (0.17), che offrono meno opzioni di vittoria.
- Importanza dei Board (moltiplicatore board): L'algoritmo distingue il valore della conquista delle diverse griglie locali. Vincere il board centrale è ritenuto cruciale per collegare le linee e vale 2.5 volte il punteggio base; anche i board angolari hanno un valore elevato (2.0x), mentre quelli laterali hanno un impatto minore (1.2x).
- Minacce e Opportunità (valuta coppie): Infine, l'IA rileva le situazioni critiche assegnando un bonus o penalità significativa (±30 punti) quando individua due simboli già allineati con una terza cella vuota, interpretandoli correttamente come una minaccia immediata di vittoria o sconfitta.

ANALISI DEI RISULTATI

- I test effettuati hanno confermato l'efficacia dell'architettura proposta sotto diversi aspetti. Sul piano computazionale, l'applicazione dell'ottimizzazione Alpha-Beta Pruning ha permesso all'agente di gestire l'elevata complessità del gioco (81 celle totali), raggiungendo una profondità di analisi fino a 5 mosse in tempi ragionevoli. Questo si traduce in una strategia di gioco che emula il comportamento umano esperto, con una predilezione intelligente per il controllo del centro e degli angoli, dettata dall'euristica posizionale.



SVILUPPI FUTURI

Il progetto pone le basi per diverse evoluzioni mirate a potenziare l'intelligenza e l'usabilità del sistema:

- Ottimizzazione della Ricerca (Move Ordering): L'efficacia dell'Alpha-Beta Pruning dipende fortemente dall'ordine in cui vengono visitati i nodi. Implementare un sistema di ordinamento delle mosse (che valuti prima quelle più promettenti) permetterebbe di massimizzare i "tagli" dell'albero, consentendo all'IA di calcolare più in profondità nello stesso tempo di esecuzione.
- Machine Learning e Pesi Adattivi: Attualmente i pesi euristici sono statici (fissati manualmente). Un passo successivo prevede l'uso di algoritmi genetici o reti neurali per far apprendere all'agente i pesi ottimali giocando migliaia di partite contro se stesso, superando i limiti della taratura manuale.
- Evoluzione dell'Interfaccia (GUI): Migliorare l'esperienza utente arricchendo l'interfaccia grafica XPCE con animazioni fluide per le vittorie e un feedback visivo in tempo reale che mostri le aree di gioco su cui l'IA sta "ragionando", rendendo il processo decisionale più trasparente e coinvolgente.

CONCLUSIONI

L'implementazione in Prolog ha dimostrato come la programmazione logica sia uno strumento potente per la gestione di spazi di stati complessi, permettendo di tradurre regole di gioco articolate in una struttura dichiarativa efficiente. L'agente sviluppato riesce così a esibire un comportamento intelligente e lungimirante, mantenendo una perfetta coerenza logica durante tutta la partita.

Grazie alla sinergia tra Minimax, l'ottimizzazione Alpha-Beta Pruning e un'euristica raffinata sui pesi posizionali, il sistema offre un livello di sfida scalabile. L'IA si dimostra capace non solo di reagire alle mosse dell'utente, ma di anticipare le minacce e pianificare strategie complesse su più board contemporaneamente.

GRAZIE PER L’ATTENZIONE