Teoria dei giochi e intelligenza artificiale: Ultimate Tic-Tac-Toe

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione a.a. 2025/2026

Luca Belardinelli

Sommario

1 Introduzione

In questa sezione viene fornita una panoramica introduttiva relativa al progetto sviluppato. Si inizia con la definizione degli obiettivi primari del lavoro svolto per poi approfondire i concetti teorici fondamentali che costituiscono la base dell'intelligenza artificiale implementata, con particolare focus sulla teoria dei giochi e sulle tecniche di ricerca avversaria.

1.1 Scopo del progetto

Lo scopo principale di questo progetto è la progettazione e lo sviluppo di un agente intelligente in grado di competere al gioco "Ultimate Tic-Tac-Toe" (o Tris Estremo), una variante complessa del classico Tris che introduce una struttura ricorsiva e un fattore di ramificazione significativamente più elevato. Il sistema è stato realizzato interamente nel linguaggio di programmazione logica Prolog, sfruttandone le capacità di manipolazione simbolica e di ricerca nello spazio degli stati.

L'obiettivo tecnico non si limita alla semplice implementazione delle regole di gioco, ma si focalizza sulla creazione di un avversario artificiale capace di pianificare strategie a lungo termine e di adattarsi alle mosse dell'utente. Per raggiungere tale scopo, il progetto affronta la sfida della complessità computazionale tipica di questo dominio attraverso l'implementazione dell'algoritmo Minimax ottimizzato con la tecnica dell'Alpha-Beta Pruning. Inoltre, il sistema è stato dotato di un'interfaccia grafica interattiva basata sulla libreria XPCE, che permette una fruizione intuitiva del gioco e visualizza in tempo reale le risposte dell'agente intelligente, offrendo all'utente diversi livelli di difficoltà scalabili.

1.2 Prolog e la Programmazione Logica

Il linguaggio scelto per l'implementazione del sistema è Prolog (Programming in Logic), un linguaggio dichiarativo basato sulla logica del primo ordine e sulle clausole di Horn. A differenza dei linguaggi imperativi, che richiedono la descrizione dettagliata dei passaggi necessari per ottenere un risultato, Prolog permette di definire il problema attraverso fatti e regole, delegando al motore inferenziale il compito di trovare le soluzioni tramite il meccanismo di unificazione e backtracking.

Questa caratteristica rende Prolog particolarmente adatto alla modellazione di giochi a turni e problemi di intelligenza artificiale. Nel contesto dell'Ultimate Tic-Tac-Toe, la natura ricorsiva del gioco (dove ogni cella contiene a sua volta una griglia) si sposa perfettamente con la struttura logica del linguaggio, facilitando la rappresentazione dello stato del gioco e la validazione delle mosse legali senza la necessità di complesse strutture dati procedurali.

1.3 Teoria dei Giochi e Algoritmo Minimax

Il cuore decisionale dell'agente sviluppato si fonda sulla Teoria dei Giochi, specificamente applicata a giochi a somma zero e a informazione perfetta. L'approccio algoritmico adottato è il Minimax, una tecnica di ricerca ricorsiva che permette di determinare la mossa ottimale simulando l'evoluzione futura della partita.

Il funzionamento dell'algoritmo si basa sulla contrapposizione di due attori: il giocatore massimizzante (l'intelligenza artificiale), che tenta di ottenere il punteggio più alto possibile, e il giocatore minimizzante (l'avversario umano), che si assume giochi al meglio delle proprie possibilità per ridurre il punteggio dell'IA. L'algoritmo esplora l'albero di gioco generando tutti i possibili scenari futuri fino a una certa profondità o fino al termine della partita. Una volta raggiunti gli stati foglia, i valori di utilità vengono propagati all'indietro verso la radice dell'albero, permettendo all'agente di selezionare la mossa che garantisce il miglior risultato possibile, indipendentemente dalla strategia dell'avversario. Questa metodologia assicura che l'IA non giochi mai casualmente, ma segua sempre un percorso matematicamente giustificato verso la vittoria o il pareggio.

1.4 Ottimizzazione tramite Alpha-Beta Pruning

Una delle sfide principali nell'Ultimate Tic-Tac-Toe è l'esplosione combinatoria delle possibili mosse, che rende impraticabile una ricerca Minimax completa in tempi ragionevoli. Per ovviare a questo problema e garantire tempi di risposta fluidi, è stata integrata la tecnica di ottimizzazione nota come Alpha-Beta Pruning.

Questa tecnica migliora l'efficienza dell'algoritmo Minimax riducendo drasticamente il numero di nodi valutati nell'albero di ricerca, senza però comprometterne l'accuratezza decisionale. Il metodo si basa sul mantenimento di due valori soglia durante l'esplorazione: Alpha, che rappresenta il miglior punteggio già assicurato per il giocatore massimizzante lungo il percorso di ricerca attuale, e Beta, che rappresenta il miglior punteggio assicurato per il giocatore minimizzante. Nel momento in cui l'algoritmo individua una mossa che peggiora la situazione rispetto a un'alternativa già valutata in precedenza, interrompe immediatamente l'analisi di quel ramo specifico, effettuando un "taglio" (pruning). Questo permette all'agente di esplorare l'albero di gioco a una profondità maggiore nello stesso intervallo di tempo, risultando in un comportamento di gioco più intelligente e reattivo.

1.5 Funzioni di Valutazione Euristica

Poiché non è sempre possibile esplorare l'albero di gioco fino alla conclusione della partita a causa dei limiti computazionali, il sistema necessita di un metodo per stimare la bontà di una configurazione intermedia. A tale scopo è stata progettata e implementata una funzione di valutazione euristica che traduce lo stato visivo della tavola in un valore numerico.

Questa funzione non si limita a un conteggio materiale dei simboli, ma applica una somma pesata basata su criteri strategici posizionali e tattici. Il sistema assegna pesi differenziati alle celle in base alla loro importanza topologica, privilegiando il controllo del centro e degli angoli

rispetto ai lati. Inoltre, viene applicato un sistema di moltiplicatori per i nove board locali, riconoscendo che la conquista della griglia centrale offre un vantaggio strategico decisivo per il controllo globale della partita. Infine, l'euristica include un meccanismo di rilevamento delle minacce che assegna bonus o penalità significative in presenza di configurazioni critiche, come due simboli allineati con una cella vuota. Questo approccio sofisticato permette all'agente di "intuire" la direzione della partita e di pianificare mosse che massimizzano le probabilità di vittoria a lungo termine.

1.6 Analisi della Complessità Computazionale

Per comprendere appieno la necessità di utilizzare tecniche di ottimizzazione come l'Alpha-Beta Pruning, è doveroso analizzare la complessità dello spazio degli stati dell'Ultimate Tic-Tac-Toe rispetto alla sua variante classica.

Nel Tic-Tac-Toe tradizionale (3x3), il numero massimo di mosse possibili è 9!, e lo spazio degli stati è relativamente piccolo (circa 5.478 stati legali), rendendolo risolvibile banalmente anche con una ricerca esaustiva.

Nell'Ultimate Tic-Tac-Toe, invece, la scacchiera è composta da 9 board locali, per un totale di 81 celle.

- **Fattore di Ramificazione (Branching Factor):** Alla prima mossa, il giocatore ha 81 opzioni. Anche nelle mosse successive, se il giocatore viene inviato in un board vuoto, ha 9 opzioni, ma se ottiene una "mossa libera" (Jolly) perché il board target è pieno, le opzioni risalgono drasticamente (fino a circa 60-70 celle libere a metà partita).
- **Profondità dell'Albero:** Una partita può durare molte più mosse rispetto al tris classico.

Senza l'ottimizzazione Alpha-Beta, un algoritmo Minimax che dovesse esplorare 6 mosse in avanti con un fattore di ramificazione medio prudenziale di 10 (basso per questo gioco), dovrebbe valutare 10^6 nodi. Tuttavia, nei casi di "mossa libera", il fattore esplode. Questo giustifica l'implementazione della logica adattiva descritta nel capitolo successivo.

2 Implementazione in Prolog

Questa sezione descrive l'architettura tecnica del sistema, analizzando come la logica del gioco, l'interfaccia grafica e l'intelligenza artificiale siano state tradotte in codice Prolog. Il progetto è stato suddiviso in moduli distinti per garantire una chiara separazione delle responsabilità e facilitare la manutenzione del codice.

2.1 Rappresentazione dello Stato di Gioco

La base dell'intera implementazione risiede nella struttura dati scelta per rappresentare il complesso stato dell'Ultimate Tic-Tac-Toe. A differenza del Tris classico, che richiede una semplice matrice 3x3, questo gioco necessita di una struttura gerarchica. Nel modulo regole_gioco.pl, lo stato è stato modellato attraverso un termine complesso che incapsula tutte le informazioni necessarie per definire un istante della partita: la lista delle nove griglie locali, la griglia principale che traccia le vittorie dei singoli board, il turno corrente e l'indice del board attivo che vincola la mossa successiva.

```
88
89  inizializza_gioco(stato_gioco(Tavole, TavolaPrincipale, Turno, TavolaCorrente, true)) :-
90      lunghezza(TavolaVuota, 9),
91      mappa_lista(=(0), TavolaVuota),
92      lunghezza(Tavole, 9),
93      mappa_lista(=(TavolaVuota), Tavole),
94      mappa_lista(=(0), TavolaPrincipale),
95      Turno = 1,
96      TavolaCorrente = -1.
```

Le griglie sono rappresentate come liste lineari di nove elementi, dove lo zero indica una cella vuota, mentre 1 e -1 rappresentano rispettivamente i giocatori X e O. Questa scelta permette di sfruttare le potenti capacità di manipolazione delle liste di Prolog. Inoltre, per gestire le impostazioni volatili della partita, come il livello di difficoltà o la scelta del simbolo iniziale, sono stati utilizzati predicati dinamici. Questi permettono di modificare il comportamento del programma a tempo di esecuzione senza alterare la logica statica delle regole, memorizzando le preferenze dell'utente tra una partita e l'altra.

2.2 Logica e Gestione delle Regole

Il modulo regole_gioco.pl contiene il nucleo inferenziale del sistema, definendo cosa costituisce una mossa valida e come si evolve la partita. La complessità principale dell'Ultimate Tic-Tac-Toe risiede nel vincolo che lega la mossa attuale alla posizione della mossa successiva. Questa logica è stata implementata nel predicato di esecuzione della mossa, il quale non solo aggiorna la tavola, ma calcola dinamicamente quale sarà il prossimo "board attivo". Il sistema gestisce automaticamente i casi limite, come quando il giocatore

viene inviato su un board già completo o vinto: in tale scenario, la logica Prolog sblocca l'intero tabellone, permettendo al giocatore di muovere liberamente ovunque (regola del "Jolly").

```
122
123  esegui_mossa(stato_gioco(Tavole, TavolaPrincipale, Turno, _, InCorso),
124               IndiceTavola, IndiceCella,
125               stato_gioco(NuoveTavole, NuovaTavolaPrincipale, NuovoTurno, ProssimaTavola, InCorso)) :-
126      elemento_n(IndiceTavola, Tavole, Tavola),
127      sostituisci_nth0(IndiceCella, Tavola, Turno, NuovaTavola),
128      sostituisci_nth0(IndiceTavola, Tavole, NuovaTavola, NuoveTavole),
129      controlla_vittoria(NuovaTavola, Vincitore),
130      sostituisci_nth0(IndiceTavola, TavolaPrincipale, Vincitore, NuovaTavolaPrincipale),
131      NuovoTurno is -Turno,
132      (   elemento_n(IndiceCella, NuovaTavolaPrincipale, 0),
133          elemento_n(IndiceCella, NuoveTavole, TavolaTarget),
134          membro(0, TavolaTarget)
135      ->  ProssimaTavola = IndiceCella
136      ;   ProssimaTavola = -1
137      ).
```

Parallelamente, il controllo della vittoria sfrutta la ricorsione e il pattern matching per verificare allineamenti vincenti su righe, colonne e diagonali. Il predicato di controllo viene applicato su due livelli: prima verifica se una mossa ha decretato la vittoria di un board locale e, in caso affermativo, aggiorna il board principale per verificare se tale conquista ha portato alla vittoria globale della partita. Questa struttura a due livelli specchia fedelmente la natura frattale del gioco.

2.3 Interfaccia Grafica con XPCE

Per rendere il gioco accessibile e interattivo, è stata sviluppata un'interfaccia grafica utilizzando la libreria XPCE, nativa in SWI-Prolog. Il modulo interfaccia_grafica.pl definisce una classe che eredita le proprietà di una finestra di sistema, gestendo il ciclo di vita dell'applicazione e gli eventi di input.La sfida principale nell'integrazione tra un linguaggio logico e una GUI orientata agli eventi è stata la gestione del flusso temporale. Poiché il calcolo dell'IA può richiedere diversi secondi ai livelli di difficoltà più elevati, l'esecuzione diretta avrebbe "congelato" l'interfaccia, rendendola non responsiva. Per ovviare a questo problema, è stato implementato un sistema basato su timer asincroni. Quando è il turno del computer, l'interfaccia non invoca direttamente l'algoritmo di ricerca, ma avvia un timer che, allo scadere, innesca il processo decisionale dell'IA. Questo approccio permette alla finestra di completare il rendering grafico dell'ultima mossa dell'utente prima che il sistema inizi a elaborare la contromossa, garantendo un'esperienza di gioco fluida. Il disegno della tavola avviene tramite primitive grafiche vettoriali che ricalcano dinamicamente lo stato del gioco, evidenziando con colori specifici il board attivo e le mosse effettuate.

```
26    initialise(W) :->
27        send_super(W, initialise, 'Ultimate Tic-Tac-Toe'),
28        larghezza_finestra(Larghezza),
29        altezza_finestra(Altezza),
30        send(W, append, new(Canvas, picture)),
31        send(Canvas, size, size(Larghezza, Altezza)),
32        send(W, slot, canvas, Canvas),
33        send(W, slot, timer_ia, new(_, timer(0.1, message(W, turno_ia)))),
```

2.4 Intelligenza Artificiale e Algoritmo Minimax

Il componente più sofisticato del sistema è contenuto nel modulo intelligenza_artificiale.pl, che implementa il "cervello" dell'agente. L'approccio adottato è quello della ricerca avversaria tramite l'algoritmo Minimax con potatura Alpha-Beta. Questo algoritmo esplora l'albero delle possibili mosse future cercando di massimizzare il vantaggio dell'IA e minimizzare quello dell'avversario.

```
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).
```

Data l'enorme vastità dello spazio degli stati dell'Ultimate Tic-Tac-Toe, una ricerca completa è computazionalmente impossibile. L'implementazione, pertanto, limita la ricerca a una profondità variabile in base al livello di difficoltà selezionato. A livello "Difficile", l'algoritmo

adotta una strategia adattiva: tenta di esplorare fino a sei mosse in profondità, ma è programmato per ridurre autonomamente questo orizzonte se il numero di board aperti è troppo elevato, prevenendo così tempi di attesa eccessivi.

Per valutare la bontà delle posizioni che non sono terminali (né vittoria né sconfitta), il sistema utilizza una funzione di valutazione euristica descritta nel predicato valuta_gioco.pl Questa funzione assegna un punteggio numerico alla tavola sommando diversi fattori strategici ponderati: il controllo del centro, il possesso degli angoli e la creazione di minacce (due simboli allineati). L'euristica è progettata per riconoscere che non tutti i board hanno lo stesso valore; conquistare il board centrale della griglia globale, ad esempio, garantisce un bonus di punteggio significativamente superiore rispetto ai board laterali, indirizzando l'IA verso strategie di gioco posizionalmente dominanti.

```
136  minimax_max_celle(StatoGioco, IndiceTavola, [IndiceCella|RestoCelle], Profondita, Alpha, Beta,
137                    MigliorCorrente, MossaCorrente, MigliorPunteggio, MigliorMossa, AlphaFinale) :-
138      StatoGioco = stato_gioco(_Tavole, _TavolaPrincipale, _Turno, _, _InCorso),
139      esegui_mossa(StatoGioco, IndiceTavola, IndiceCella, NuovoStatoGioco),
140      Profondita1 is Profondita - 1,
141      minimax(NuovoStatoGioco, Profondita1, Alpha, Beta, false, Punteggio, _),
142      (   Punteggio > MigliorCorrente
143      ->  NuovoMigliore = Punteggio, NuovaMossa = IndiceTavola, NuovoAlpha is max(Alpha, Punteggio)
144      ;   NuovoMigliore = MigliorCorrente, NuovaMossa = MossaCorrente, NuovoAlpha = Alpha
145      ),
146      (   NuovoAlpha >= Beta
147      ->  MigliorPunteggio = NuovoMigliore, MigliorMossa = NuovaMossa, AlphaFinale = NuovoAlpha
148      ;   minimax_max_celle(StatoGioco, IndiceTavola, RestoCelle, Profondita, NuovoAlpha, Beta,
149                            NuovoMigliore, NuovaMossa, MigliorPunteggio, MigliorMossa, AlphaFinale)
150      ).
```

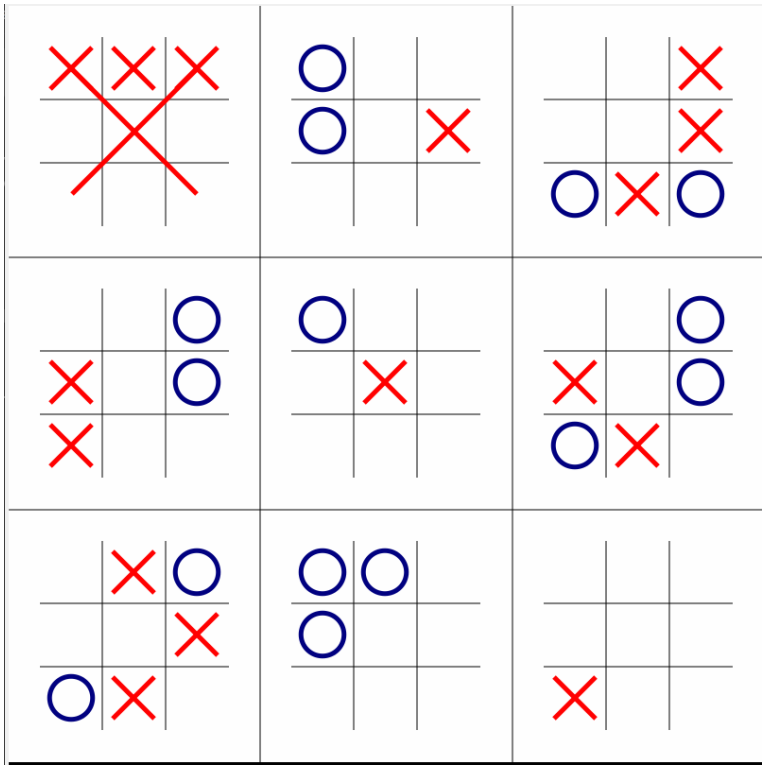
3 Metodologia di Test e Validazione

Lo sviluppo di un sistema basato su intelligenza artificiale richiede una fase di testing rigorosa per verificare non solo la correttezza delle regole ("il gioco funziona?"), ma anche la qualità delle decisioni prese dall'agente ("l'IA è intelligente?").

3.1 Testing Funzionale

Il testing funzionale si è concentrato sulla verifica della corretta implementazione delle regole complesse dell'Ultimate Tic-Tac-Toe. Sono stati simulati scenari limite per validare la robustezza del codice:

- **Regola del Vincolo:** È stato verificato che l'interfaccia impedisca all'utente di cliccare su board diversi da quello attivo (evidenziato in rosso).
- **Gestione del Jolly:** Sono state create partite ad hoc per forzare il giocatore su un board già vinto. Il sistema ha risposto correttamente rimuovendo il vincolo grafico e logico, permettendo la selezione di qualsiasi cella libera (ProssimaTavola = -1 nel codice).



- **Riconoscimento Vittoria:** È stato testato il corretto aggiornamento della Tavola Principale quando un giocatore allinea tre simboli in un board locale, e la conseguente vittoria globale.

3.2 Testing Prestazionale e Log di Debug

Per valutare le performance dell'algoritmo Minimax, è stato utilizzato il sistema di logging integrato nella console di SWI-Prolog.

Durante l'esecuzione al livello "Difficile", il sistema stampa a video la profondità di ricerca utilizzata per ogni mossa. I test hanno mostrato che:

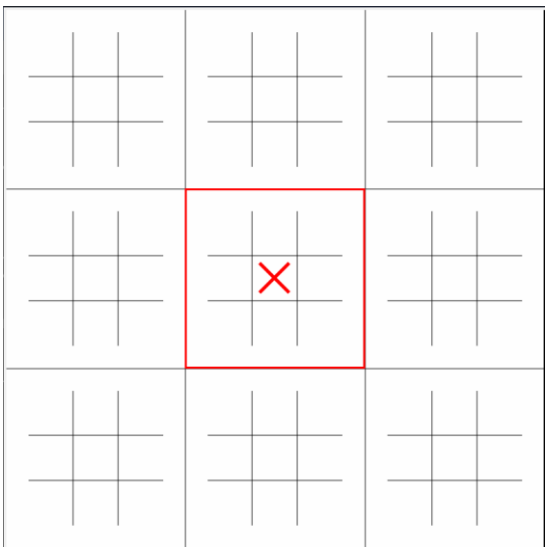
1. Nelle fasi iniziali e finali, l'IA mantiene la profondità massima impostata (5).
2. Nelle fasi centrali della partita (mosse 10-30), quando il numero di board aperti è elevato, il meccanismo di controllo implementato in interfaccia_grafica.pl riduce automaticamente la profondità a 3.

Questo monitoraggio ha confermato che il tempo di risposta dell'IA rimane sempre in un range accettabile, garantendo un'esperienza utente fluida senza blocchi eccessivi dell'interfaccia grafica.

3.3 Validazione della Strategia

La qualità strategica è stata validata empiricamente giocando numerose partite contro l'agente. Si è osservato che:

- **Apertura:** L'IA occupa sempre il centro del board centrale.



- **Mid-game:** L'IA dimostra di "capire" il pericolo, bloccando sistematicamente le coppie avversarie quando necessario (funzione valuta_coppie).

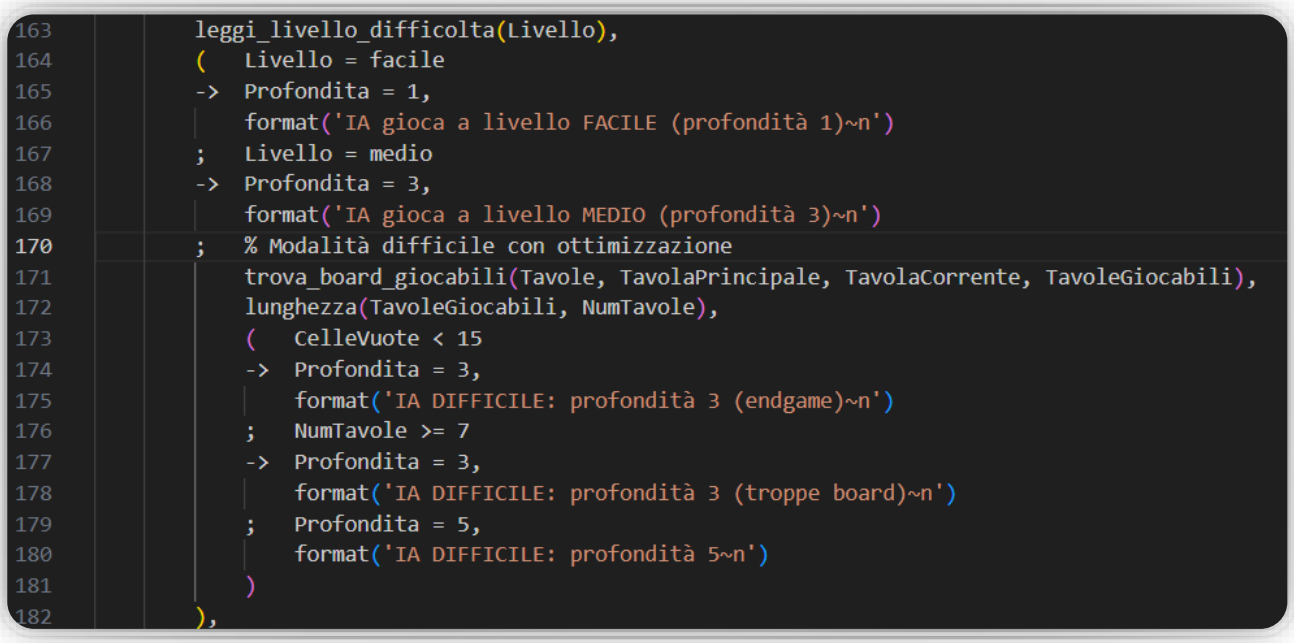
- **End-game:** L'agente è in grado di sacrificare un board locale se questo gli permette di vincere la partita globale, dimostrando che la propagazione dei punteggi nel Minimax funziona correttamente rispetto alla visione "greedy" locale.

4 Conclusioni

In questa sezione conclusiva vengono analizzati i risultati ottenuti dallo sviluppo dell'agente intelligente per il gioco Ultimate Tic-Tac-Toe, discutendo l'efficacia delle strategie implementate e le performance del sistema. Vengono inoltre delineate le osservazioni finali sul progetto e proposti possibili scenari per l'evoluzione futura del software.

4.1 Analisi delle Prestazioni dell'IA

La valutazione principale del progetto riguarda la capacità dell'agente di competere efficacemente contro un giocatore umano. I test condotti hanno evidenziato come l'integrazione dell'algoritmo Minimax con la potatura Alpha-Beta sia stata determinante per rendere il gioco fruibile. Senza l'applicazione del pruning, la ricerca a profondità elevate (superiori a 3) risultava proibitiva in termini di tempo di calcolo, a causa dell'alto fattore di ramificazione del gioco.



```
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         ),
```

Grazie all'ottimizzazione Alpha-Beta, l'agente a livello "Difficile" riesce a esplorare l'albero di gioco fino a una profondità di 5 semi-mosse mantenendo tempi di risposta accettabili. Particolarmente efficace si è rivelata la logica adattiva implementata nell'interfaccia, la quale riduce dinamicamente la profondità di ricerca (da 5 a 3) qualora il numero di board aperti sia elevato. Questo meccanismo ha permesso di evitare il blocco dell'applicazione nelle fasi intermedie della partita, dove le possibilità di mossa sono massime, garantendo un equilibrio ottimale tra intelligenza di gioco e fluidità dell'esperienza utente.

4.2 Efficacia dell'Euristica Posizionale

L'analisi qualitativa delle partite giocate ha confermato la validità della funzione di valutazione euristica progettata. Il sistema di pesi posizionali ha conferito all'IA uno stile di gioco che emula il ragionamento strategico umano. Si osserva che l'agente tende sistematicamente a occupare le celle centrali dei board locali, sfruttando il peso maggiore (0.22) assegnato a tali posizioni rispetto agli angoli e ai lati.

```
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      ).
```

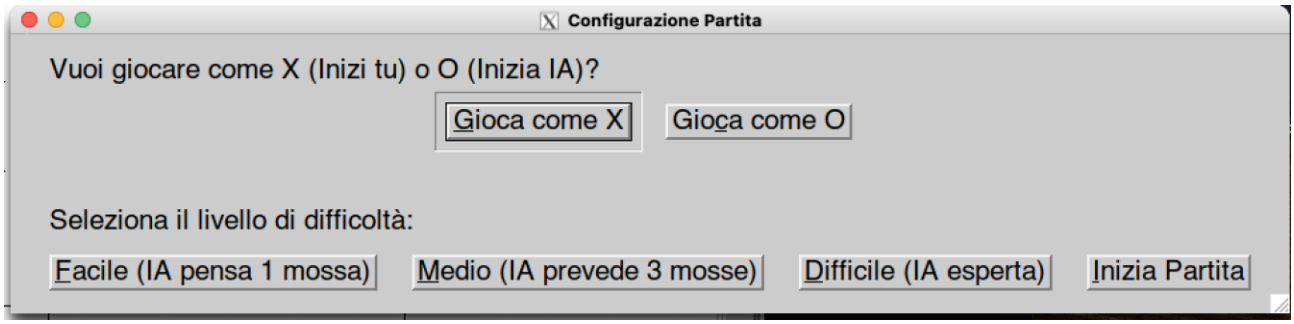
Ancora più rilevante è l'impatto dei moltiplicatori assegnati ai board globali. L'intelligenza artificiale dimostra una chiara priorità nella conquista del board centrale della griglia principale, riconoscendone il valore strategico superiore (moltiplicatore 2.5) per il controllo del flusso di gioco. Inoltre, il meccanismo di rilevamento delle minacce, che assegna un bonus significativo alle configurazioni di "due in fila", permette all'agente di passare efficacemente da una strategia posizionale a una tattica aggressiva o difensiva non appena si presenta l'occasione di chiudere un punto o la necessità di bloccare l'avversario.

4.3 Confronto tra i Livelli di Difficoltà

Confrontando il comportamento dell'agente ai diversi livelli di difficoltà, emerge chiaramente il valore della profondità di ricerca. Al livello "Facile" (profondità 1), l'IA gioca in modo quasi puramente reattivo, limitandosi a valutare la situazione corrente senza prevedere le

conseguenze delle proprie azioni; questo la rende vulnerabile a trappole e strategie a lungo termine impostate dall'utente.

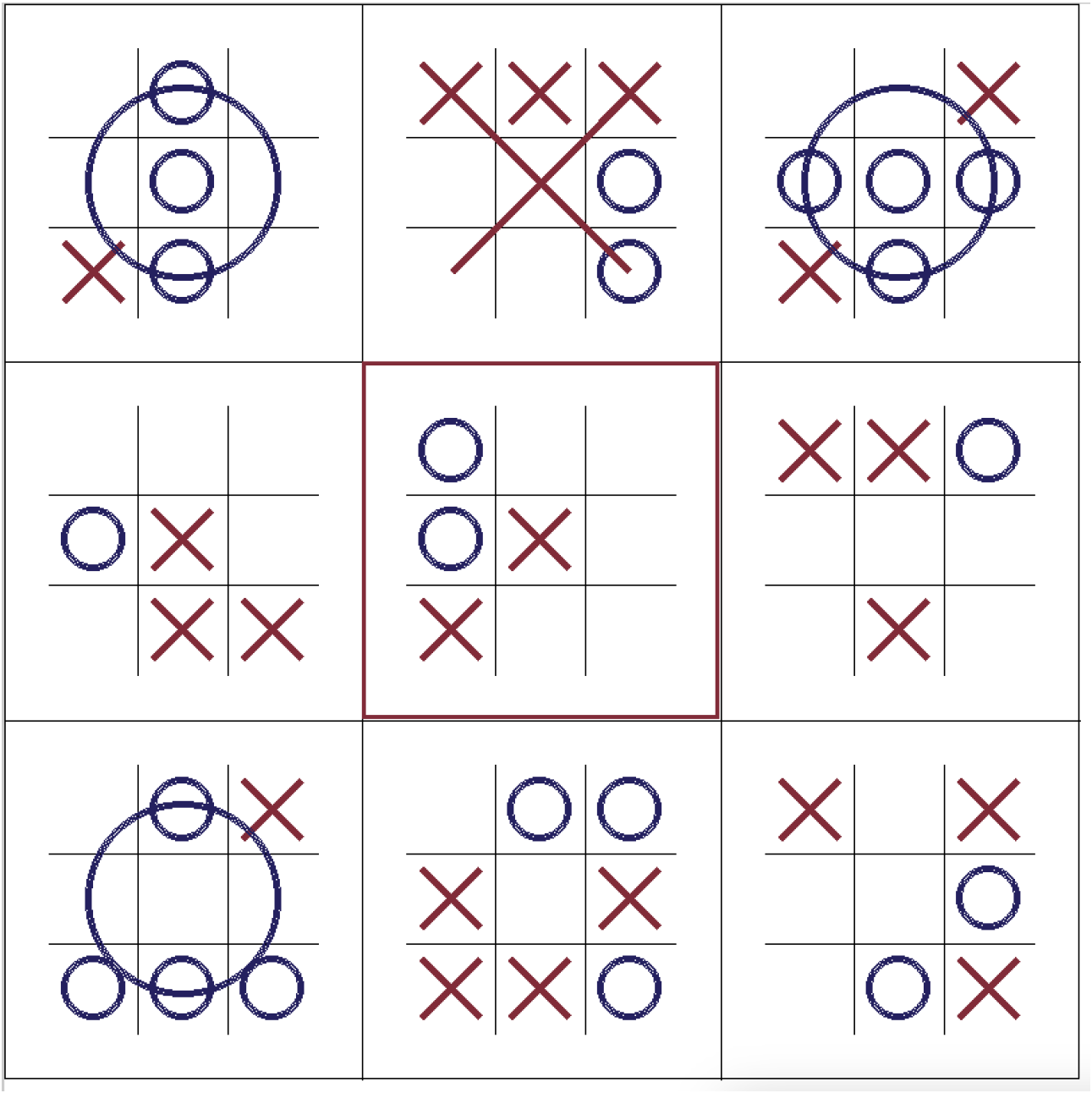
Al contrario, al livello "Difficile", la capacità di previsione a lungo raggio emerge con forza. L'agente è in grado di sacrificare un vantaggio immediato in un board locale per costringere l'avversario a giocare in un board dove l'IA ha un vantaggio decisivo. Questa capacità di manipolare il vincolo della mossa successiva rappresenta il vero salto qualitativo rispetto a un'implementazione banale e dimostra il successo dell'approccio algoritmico adottato.



4.4 Esempio di Comportamento Strategico

Un esempio concreto del funzionamento del sistema si osserva quando l'IA si trova a dover scegliere tra una mossa che porta alla vittoria di un board laterale e una che blocca l'avversario nel board centrale. In virtù dei pesi euristici implementati, l'agente spesso predilige la difesa del centro, calcolando che la perdita di quella posizione strategica comprometterebbe l'intera partita più di quanto la conquista di un board periferico possa beneficiarla.

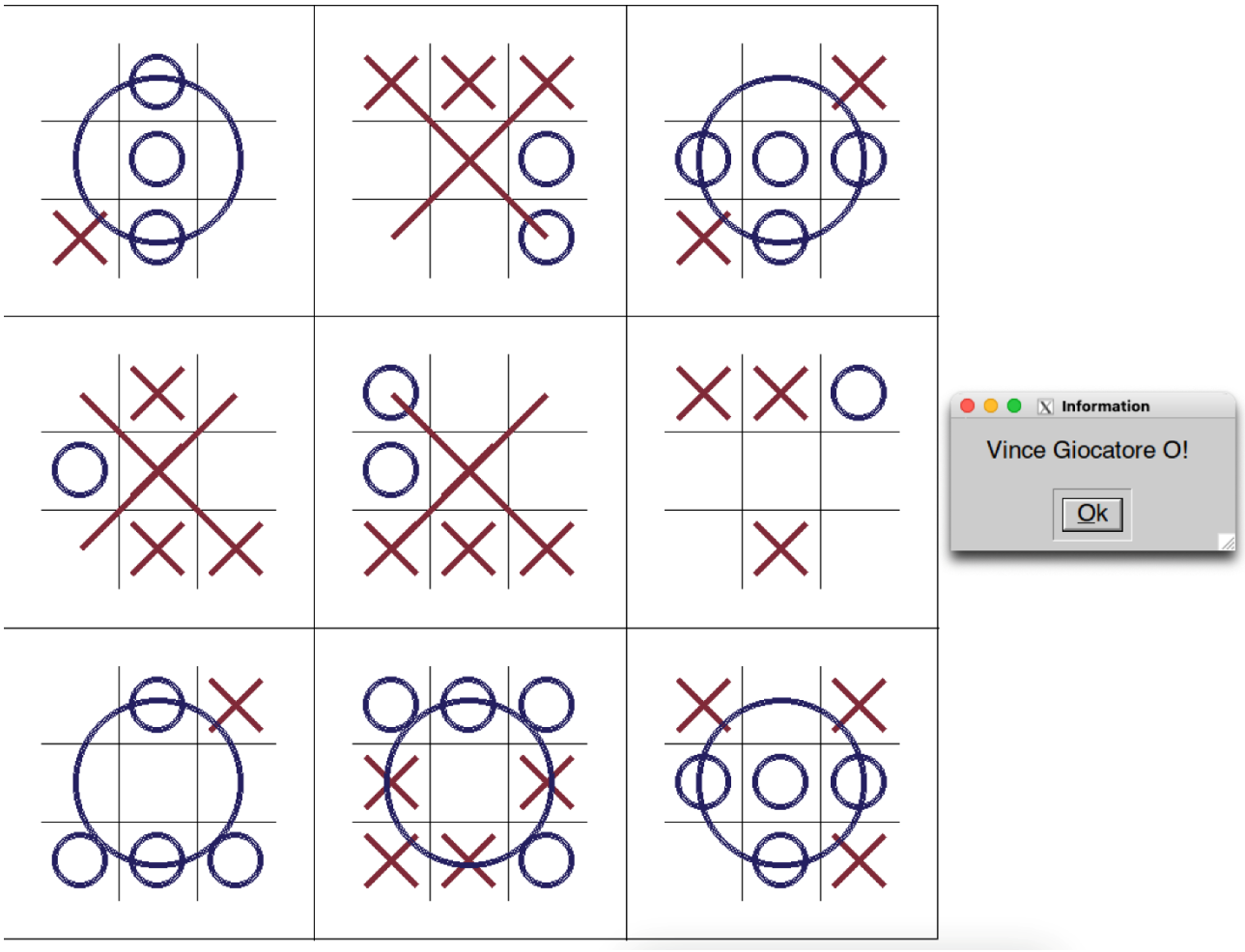
Similmente, grazie alla ricerca in profondità, l'IA evita mosse che, pur sembrando vantaggiose nell'immediato, invierebbero l'avversario su un board già vinto (concedendogli quindi una mossa libera o "Jolly"). Il sistema riconosce tale scenario come estremamente pericoloso attraverso la propagazione dei valori Minimax e sceglie percorsi alternativi che mantengono il controllo del gioco, dimostrando una "comprensione" implicita delle regole avanzate dell'Ultimate Tic-Tac-Toe.



4.5 Osservazioni Finali

In sintesi, il progetto ha raggiunto pienamente gli obiettivi prefissati, dimostrando come il paradigma della programmazione logica offerto da Prolog sia eccezionalmente adatto alla modellazione di giochi complessi a informazione perfetta. La struttura ricorsiva del codice rispecchia elegantemente la natura frattale del gioco, mentre l'uso di XPCE ha permesso di creare un'interfaccia funzionale senza dover ricorrere a linguaggi esterni.

Il sistema risultante è robusto, gestisce correttamente tutti i casi limite previsti dal regolamento e offre un livello di sfida scalabile che va dal principiante all'esperto. La separazione modulare tra logica, interfaccia e intelligenza artificiale ha inoltre garantito un codice pulito ed estensibile, facilitando eventuali interventi futuri di manutenzione o miglioramento.



4.6 Possibili Sviluppi Futuri

Nonostante i risultati soddisfacenti, esistono diverse direzioni di ricerca per evolvere ulteriormente il sistema. Un primo sviluppo fondamentale riguarderebbe l'implementazione dell'ordinamento delle mosse (Move Ordering) all'interno dell'algoritmo Minimax. Valutare per prime le mosse più promettenti (ad esempio quelle che portano alla conquista di una cella centrale o alla vittoria di un board) permetterebbe di massimizzare l'efficacia del taglio Alpha-Beta, incrementando ulteriormente la profondità di ricerca raggiungibile o riducendo i tempi di calcolo.

Un secondo ambito di miglioramento concerne la calibrazione dell'euristica. Attualmente i pesi posizionali sono statici e definiti a priori; l'introduzione di tecniche di apprendimento automatico, come algoritmi genetici o apprendimento per rinforzo, consentirebbe all'agente di ottimizzare autonomamente questi parametri giocando migliaia di partite contro se stesso. Infine, l'interfaccia grafica potrebbe essere arricchita con animazioni per le vittorie e indicatori visivi che mostrino all'utente su quali board l'IA sta concentrando la sua analisi, rendendo il processo decisionale della macchina più trasparente e didattico.