UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

CORSO DI LAUREA IN INGEGNERIA INFORMATICA
CORSO DI INTELLIGENZA ARTIFICIALE PROF. S. MARRONE - A.A. 2023 - 24

TESINA

Introduzione al Mondo della Teoria dei Giochi

What is Game Theory?

Conte Pietro N46006426

Contents

1 Introduzione

1.1 A cosa ci stiamo riferendo?

Lo scopo di questa tesina è quello di andare ad introdurre un nuovo punto di vista, quello riferito alla Teoria dei Giochi.
La Teoria dei Giochi è una delle aree più vecchie in cui è stata applicata l'Intelligenza Artificiale. Risalgono già al 1950, infatti, i primi programmi di scacchi, scritti da Claude Shannon e da Alan Turing. Da allora ci sono stati continui miglioramenti, fino ad arrivare ai giorni nostri, in cui i programmi di gioco possono permettersi di competere con avversari umani molto forti, e anche di batterli. Questi sviluppi sono stati il risultato di un intenso studio e di un grande interesse e possono essere motivati in vario modo.
Verranno studiati problemi che possono essere affrontati a tutti gli effetti come un problema di intelligenza artificiale e, come anche nei "problem solving and search", anche in questo caso si può parlare del concetto di grafo.
Ma perchè la soluzione mediante indicazione a grafo risulta essere cosi conveniente? Perchè in questo modo è possibile mappare la struttura sotto forma tabellare, quindi è possibile andare a visualizzare l'evoluzione degli stati.
Ma, questa volta ci troviamo in un ambiente competitivo (non siamo più da soli), ma ci sarà un altro agente (avversario) competitivo. Mentre il proprio agente cercherà di massimizzare una funzione obiettivo, l'avversario cercherà di minimizzarla.
La soluzione è quella di trovare una successione di stati, vista anche come strategia: si conosce stato iniziale e finale, ma non la soluzione del problema.
Per concludere, diciamo che la particolarità di questo ramo è che, non essendo più gli unici ad operare, i giochi sono considerati a turni.

1.2 Come è strutturato?

Gli avversari rendono l'ambiente **non deterministico**, ma è semplicemente una forzatura imposta in partenza. Difatti, abbiamo visto due tipi di determinismi:

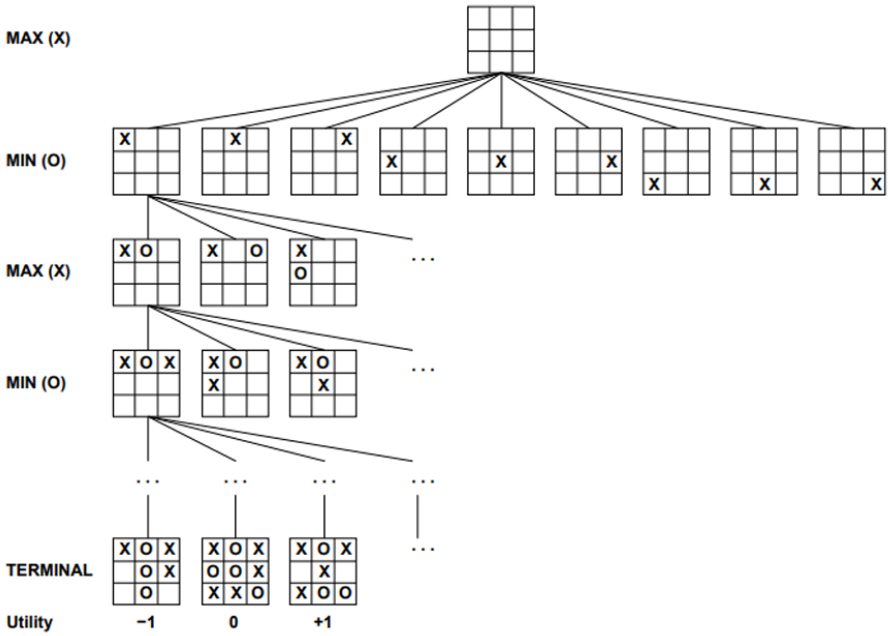
1. Determinismo Intrinseco: associato all'ambiente, l'agente non può farci nulla;

2. Determinismo Causato Dall'Altro Agente: abbiamo noi la possibilità di forzare l'avversario verso situazioni a noi favorevoli.

L'avversario va a cambiare completamente lo scenario di chi lo sfida ma non lo fa in maniera casuale o con delle regole, bensì con dei ragionamenti.
Nel caso di due giocatori, si parla di **gioco a somma zero** o **a somma costante** quando sono deterministici, a turni e la somma è zero, ossia si dice che vince uno solo e quindi non si divide il "bottino" (vincita di uno = perdita dell'altro).

1.2.1 Nozioni sull'albero di gioco

Il gioco presentato è quello del Tris. Man mano che si gioca, il grafo degli stati evolve e si passa da uno stato all'altro.
Ci sono due giocatori, MIN e MAX, ed in generale inizia sempre MAX.
Sorge spontanea una domanda:
ma se ci stiamo focalizzando su sistemi offline, ovvero sistemi in cui io eseguo l'algoritmo ancor prima della partita, come faccio a capire che mossa farà MIN? Questa cosa fa sì che ad ogni livello si abbiano mondi diversi paralleli che, in base alla scelta fatta, distruggono tutte le realtà non utilizzate e l'albero continui in base alle scelte prese in cosiderazione da entrambi i giocatori.
Un'esempio di albero di gioco risulta essere questo: (altro non è che lo spazio degli stati).



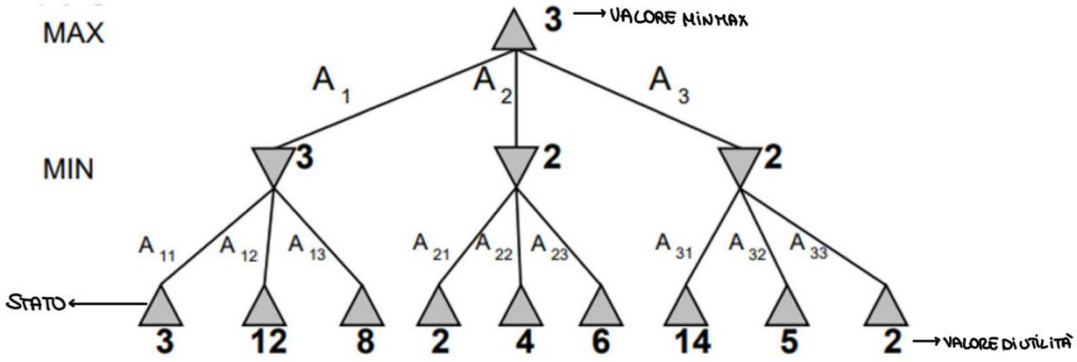
Come notiamo, gli stati soluzione presentano tutti un particolare valore, chiamato **valore di utilità** che è valore della funzione obiettivo, e poichè tutto fa riferimento a MAX, questa è la funzione percepita da quest'ultimo.
Alla funzione di utilià verrà associato il valore di +1 quando ci troviamo nel caso di vittoria da parte di MAX, valore pari a 0 nel caso di pareggio, -1 altrimenti.

2 Algoritmi

Presentiamo ora gli algoritmi studiati durante il corso.
L'algoritmo che verrà illustrato adesso prende il nome di **MINIMAX**.

2.1 MINIMAX

Esso è visto come l'algoritmo perfetto, quello che fa fare la miglior partita possibile.
Qualunque giocatore utilizzi MINIMAX, per definizione non può perdere, al più pareggia, quindi chi parte con MINIMAX ha il vantaggio.
Per spiegare come funziona nel dettaglio, inseriamo un'immagine illustrativa per facilitare l'analisi:



Il piano è: in ogni momento, giocando dal punto di vista di MAX, la scelta da fare viene fatta in base al più alto valore minimax. (MAX dà per scontato che dall'altro lato vi sia un giocatore che gioca perfettamente). MAX, per essere contento, vuole terminare nello stato con funzione di utilità pari a 14, se non può, pari a 12 e così via.

NOTA:

1. Mi è possibile conoscere le funzioni di utilità;
2. I due valori hanno un preciso modo per essere letti, ossia:
   (a) Valore MINIMAX: MAX terminerà la partita avente valore di utilità ALMENO pari a 3;
   (b) Valore di Utilità: Questo nodo rappresenta uno stato terminale, ossia un nodo dove la partita è terminata e MAX ha un valore di utilità pari a 2.

Questo valore minimax è il miglior valore raggiungibile da parte di MAX o il più piccolo valore raggiungibile da parte di MIN e, nei nodi foglia, il valore minimax sarà identico al valore di utilità.
Minimax decide la strategia facendo un'ipotesi: MIN fa la miglior giocata possibile, e pensando ciò, MAX può seguire la giusta sequenza delle mosse perchè sa cosa aspettarsi da MIN ad ogni contromossa.

In questo particolare esempio sorge una domanda: MAX sicuramente raggiungerà come funzione obiettivo la foglia 3? NO. Secondo la strategia, nella peggiori delle ipotesi avrà questo valore, altrimenti superiore. La cosa che è certa, è che non avrà mai un valore inferiore a 3.
Per non confonderci con il termine sovracitato, ossia quello di mossa, si introduce un nuovo termine,

ossia quello di **ply** (play), ossia una giocata di un singolo giocatore. Invece con il termine **mossa**, in alcuni giochi si fa riferimento alla coppia mossa+risposta.

Concludendo, possiamo allora confermare che si tratta di un algoritmo su grafo dove, MIN ogni volta che viene interrogato scende fino alla fine e restitusice il valore più piccolo, MAX restituirà il miglior valore possibile tra i peggiori.
Se MAX e MIN prendono la scelta di approfondire un nodo alla volta, l'albero di gioco verrà esplorato in profondità. Esso non risulterà essere altro che un'algoritmo di ricerca in profondità, solo che la scelta in un dato momento non è più basata solo sui valori locali, ma anche su quelli minimax.

2.1.1 Proprietà

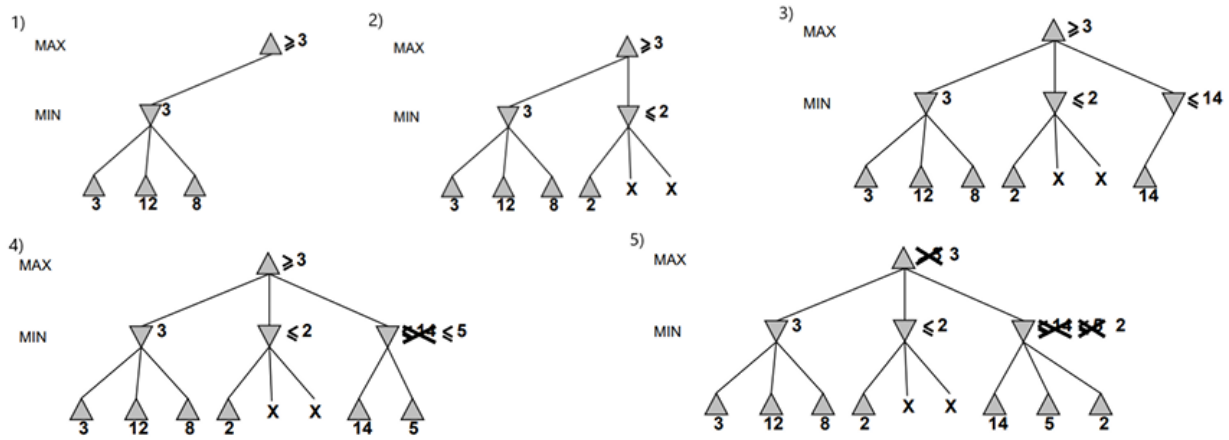
Esistono delle metriche di performance che possono essere utilizzate per misurare la bontà delle varie strategie di ricerca. Tra le varie metriche ci sono i concetti di:

- Completezza: è completo? Esso viene visto come un algoritmo in profondità, il quale ricordiamo essere competo solo nel caso in cui la profondità non è infinita. Quindi se l'albero è finito, MINIMAX è completo.
- Ottimalità: Si, è stato costruito proprio per questo motivo.
- Complessità temporale: Ereditando dal depth-first, è esponenziale rispetto al brenching factor e alla profondità media della soluzione.
- Complessità spaziale: Ereditando dal depth-first, è lineare.

Problema: Nel caso di giochi con più mosse, quest'algoritmo, calcolando la profondità e il branching factor, diventerebbe infattibile per qualsiasi calcolatore.
Ma, è davvero necessario esplorare tutti i rami? No, ma capiamo il motivo.
Non mi interessa associare i valori MINIMAX a tutti gli stati, ma riuscire ad avere una strategia e costruire i valori minimax per il percorso migliore dalla foglia alla radice. Se l'avversario fa la partita perfetta, la soluzione, per forza di cose, sarà unica e vi è solo un percorso. Quindi, vogliamo fare in modo di ricostruite i valori minimax solo degli stati presenti nella soluzione.

2.2 Alfa-Beta Pruning

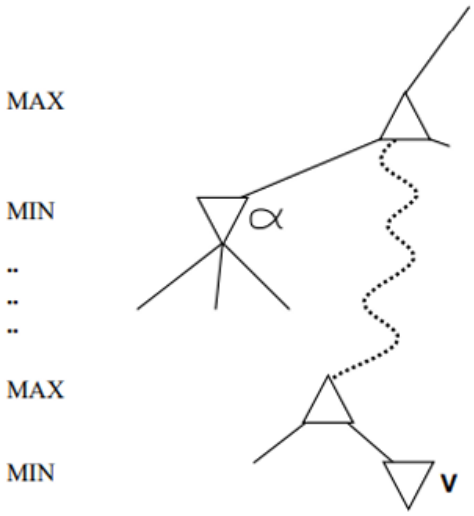
Per fare ciò che abbiamo spiegato sopra, si utilizza l'alfa-beta pruning.
Semplicemente, si vanno a tagliare alcuni rami dell'albero che si sta costruendo, ossia si vanno a tagliare tutti i rami "non promettenti".
Visualizziamo un esempio:



[NOTA –> 2) Il valore di MIN è al più 2 e ci possiamo fermare perchè già sappiamo che MAX non entrerà mai lì].

Perchè si chiama alfa-beta pruning? Perchè ai valori di minimax in ogni stato, si associano due valori, alfa e beta. Essi sono il minor e il maggior valore raggiungibile a quel nodo. In pratica, alfa per MAX rappresenta il miglior valore raggiungibile, beta è il contrario per MIN.

Graficamente, troviamo un elemento di questo tipo:



Le proprietà non variano rispetto a MINIMAX e non impattano sul risultato finale, ma ci possono essere dei problemi.

Il primo fa riferimento all'**ordinamento**, ossia il modo in cui si esplorano i nodi.
Essenzialmente, nel caso peggiore, alfa-beta non ha nessun impatto, ma si può decidere di usarlo sempre. Perchè allora non viene scelto di ordinarlo? Perchè si è visto che il tempo che si impiega per ordinarlo risulta essere maggiore di MINIMAX, quindi a quel punto, si sceglie di utilizzare quest'ultimo.

Ma è possibile trovare strategie per migliorare tale approccio. Si è visto che, invece di scegliere il primo nodo nella struttura dati, se ne sceglie uno randomico tra quelli disponibili: mediamente si migliora tantissimo, soprattutto su alberi enormi.
Ovviamente, invece di fare una scelta randomica, una scelta intelligente è quella di tenere traccia, man mano che si compiono le scelte, di quelle migliori in passato: in questo caso parliamo di Killer Moves. Ma nonostante tutti questi accorgimenti, minimax con alfa-beta pruning continua ad essere computazionalmente molto oneroso.

Altro problema fa riferimento al caso del **non avversario perfetto**: in questo caso quest'algoritmo può diventare particolarmente oneroso. Perchè? Perchè devo rieseguirlo ogni volta che l'avversario fa una mossa non perfetta.
Ma questo concetto può tornarci di aiuto. Cosa intendo?
Facciamo riferimento a un gioco quando ho un limite di tempo sulla mossa: se io sto giocando contro un'avversario minimax, per forza di cose, mi serve una giocata perfetta; tuttavia, avendo poco tempo a disposizione, si va a limitare il tempo che dedico agli errori. Allora posso pensare di vincere bloccando l'algoritmo avversario, ma in che modo? Sbagliando sempre e obbligandolo a rieseguire minimax in tempo reale.

2.3 Strategie Approssimate

Ora, cosa succede se non ho abbastanza memoria per memorizzare l'albero man mano che cresce? (senza memoria rischio di fermarmi a metà, e questo è un problema grave perchè non posso far risalire il valore di utilità).
In questo caso devo perdere la proprietà di garantire il valore e mettere su un'euristica, ossia una variante minimax che si chiama Euristic Minimax.

L'idea è questa:
continuo a scegliere finchè posso, dopodichè, nel momento in cui non posso andare avanti, mi chiedo: è un nodo terminale? Bene, il valore minimax associato a quel nodo è proprio il valore di utilità; mi sono invece fermato in un nodo che non è terminale, che faccio? Lancio una funzione euristica (chiamata **EVAL**) che mi dà una stima di quanto è buono quello stato, che mi vada a dire: se mi fermo qui, potenzialmente con che valore finirà la partita? Ovviamente essa è una stima, non un valore vero. Quindi una volta lanciata l'euristica assegno il valore minimax pari a questo valore e faccio procedere l'algoritmo così come solito. Così facendo, sostanzialmente, sto tagliando l'albero, facendo un taglio orizzontale in base alla potenza computazionale che ho a disposizione. Quando taglio? Dipende. In generale posso tagliare quando non si ha più potenza computazionale, o ancora, si ha una certa funzione di **CUTOFF** che determina quando tagliare e, sulla base di alcune caratteristiche e scelte si ripete il procedimemto di prima domandandosi se è un nodo terminale o bisogna utilizzare EVAL.

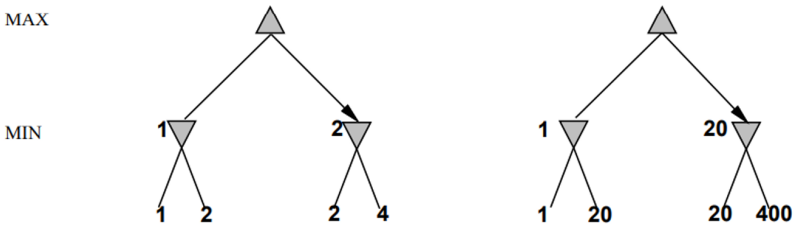
Noto che:

1. Nell'euristic minimax perdo la garanzia sul riferimento al valore vero con il quale minimax potrebbe "circa" finire la partita. E quindi? Come faccio a fidarmi? Non posso, ma, se riesco a garantire che il valore sia affidabile e comunque con valori che sono compresi tra la **sconfitta** e la **vittoria** vera, allora è ragionevole che il valore alla radice sia verosimile;

2. Come stimo il valore di EVAL? La risposta è dipende e, come in A*, mi viene data da un'esperto del dominio. Ci sarà qualcuno che permette di avere una funzione che vada a quantificare la bontà di uno stato rispetto ad un'altro (avrò una certa funzione di **valutazione**);

3. Non è possibile avere una guida che dica che un'euristica sia meglio dell'altra, quindi, o ci affidiamo all'esperienza dell esperto o si vanno a mettere su altre strategie, come quella del ML.

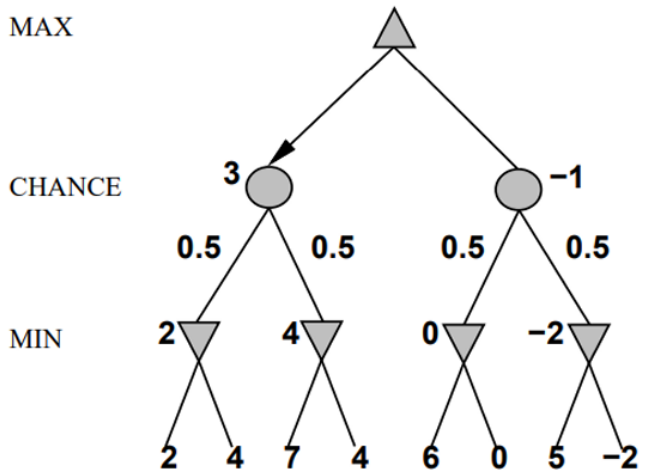
Abbiamo detto che non si ha nessun modo di stimare l'euristica rispetto al valore vero, quindi come ci si può fidare se non si sa di quanto si sta sbagliando? Si è visto che i valori precisi **non contano**.



Non importa se una è migliore o peggiore dell'altra. Non importa il vero valore, l'importante e che la stima segue un'andamento monotomico, la stessa trasformazione dappertutto. Questo porterà un'impatto sul valore minimax della radice, ed è per questo che quel valore non è affidabile.

3 Giochi non Deterministici

Aggiungiamo adesso al discorso il fattore di non deterministico. In questo caso MAX non può fare la scelta basata sulla miglior scelta che farebbe MIN, ha quindi bisogno di una certa probabilità che qualcosa succeda al di là di eventuali controlli.
Bisogna quindi trovare un modo per modellare l'elemento di randomicità. Si va ad aggiungere quindi una nuova tipologia di nodi: i nodi di **possibilità** o di **chance**.
In questo caso, il valore minimax associato ai nuovi nodi, viene sostituito dall'**expected minimax**.
Vediamo un'esempio per capire meglio:

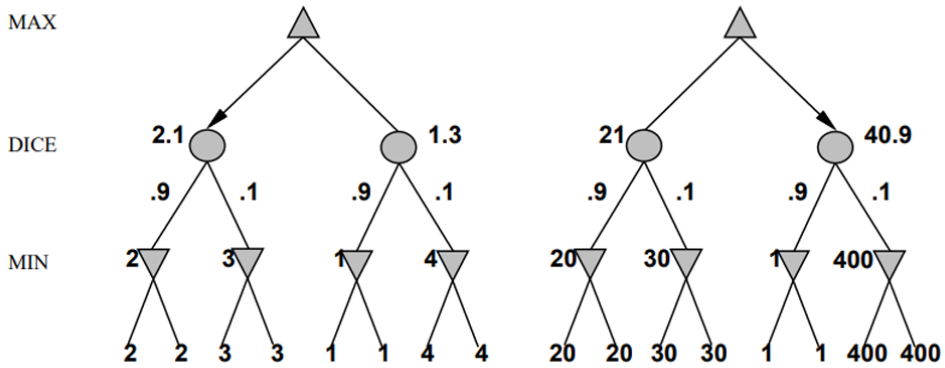


Quanti nodi chance? Dipende dal particolare gioco.

Problema:

1. Ogni volta che si ha un nodo probabilità, si aggiunge almeno una play e si aumenta la profondità dell'albero. Ma sappiamo bene che minimax è molto sensibile a quest'ultimo.
   Quindi, il fatto di aver reso il gioco non deterministico, impatta molto su MINIMAX.
   Notiamo che si può ancora utilizzare alfa-beta, poichè si è semplicemente aggiunto un nodo il cui comportamento è noto.

2. I **Valori esatti sono importanti**: Come notiamo dall'immagine sottostante, il valore effettivo della funzione di EVAL è cruciale. La monotonia è rispettata, ma la seconda trasformazione è non lineare. Come abbiamo visto prima, questa cosa nel caso di euristic minimax non ha nessun impatto, invece qui può avere un'impatto perchè questi valori vanno pesati per la realizzazione della variabile aleatoria. Si passa dallo scegliere un nodo a sinistra ad uno a detra.



Solo per scopo illustrativo, bisogna poi fare una differenza tra i giochi a informazione imperfetta e giochi parzialmente osservabili. Essenzialmente:

1. Ai primi è legato il fatto che l'informazione c'è ma non risulta accessibile.
2. Nei secondi, l'informazione si può averla, ma i sensori a disposizione non permettono di arrivarci: quindi è un fattore legato all'osservabilità.

4 Esempio di Gioco

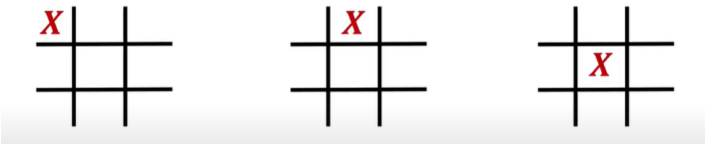
Per concludere il discorso e per capire mediante un esempio come funziona tutto questo interessantissimo ramo, andiamo a visionare delle partite con un gioco chiamato **Gioco del TRIS**.

Vedremo in questo capitolo come giocare la partita perfetta, ossia il metodo per non perdere mai e, se l'avversario commette anche solo un errore (non più perfetto), si vincerà la partita; se l'avversario invece giocherà la partita perfetta, finirà in parità.

Il nostro compito è risolvere il gioco e con risolverlo si intende la procedura di ricerca di un'algoritmo migliore in ogni eventualità.
Questo perchè? Perchè è possibile agire in un modo razionale e studiare il gioco a tavolino e capire qual è la strategia migliore.
Nota: Limiteremo l'analisi al caso di inizio da parte di MAX.

Verranno inserite mano mano delle foto per illustrare l'andamento della partita.

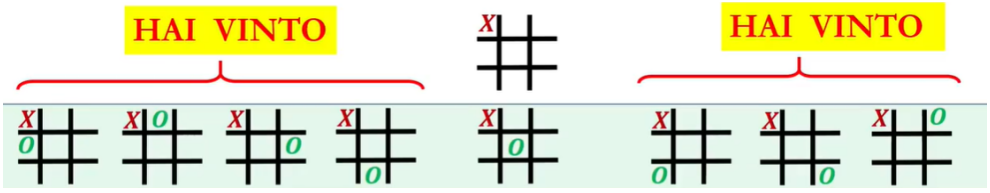
INIZIAMO A GIOCARE:



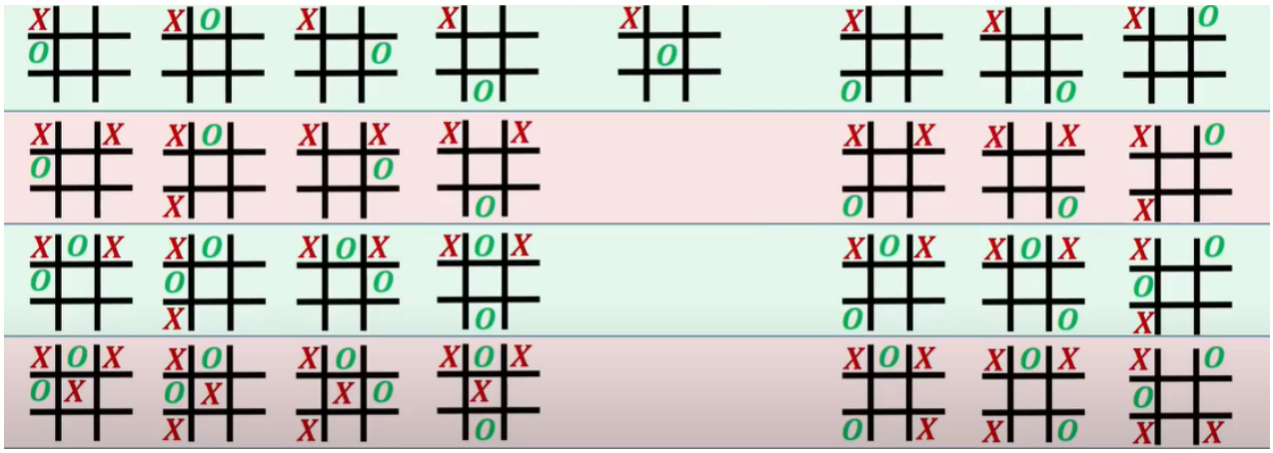
In questo momento notiamo che ci sono 9 caselle libere, ma solo effettivamente 3 possibili scelte, ossia quella ad angolo, quella sulla linea esterna centrale e quella al centro. Le altre sono essenzialmente simmetriche, ecco perchè si parla di 3 e non di 9. La **scelta migliore** è quella di giocare nell'angolo, infatti tramite una ricerca effettuata, si può scopire che questa scelta invece delle altre due è quella che porta a maggior successo.

Verrà illustrato soltanto questa casistica.

Giocando in un angolo, l'avversario ha ancora 8 caselle libere e quindi può rispondere in 8 modi diversi. Notiamo che già da questa scelta dell'avversario si può capire che la partita "già potrebbe essere decisa". Con 7 mosse su 8, la partita può già considerarsi vinta e ne viene illustrato il motivo.

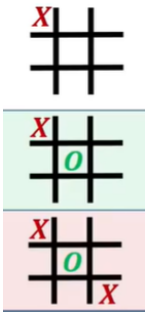


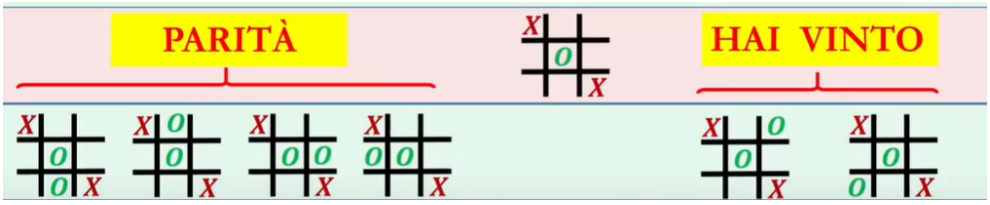
Si evince che è sufficiente inserire la **X** ad angolo evitando la **O** dell'avversario. In questo momento l'avversario sarà obbligato a mettere la sua O tra le due X inserite da noi (avversario non farà un errore banale), ed ora sarà sufficiente inserire una X in modo da aprire due tris, e il gioco è fatto.



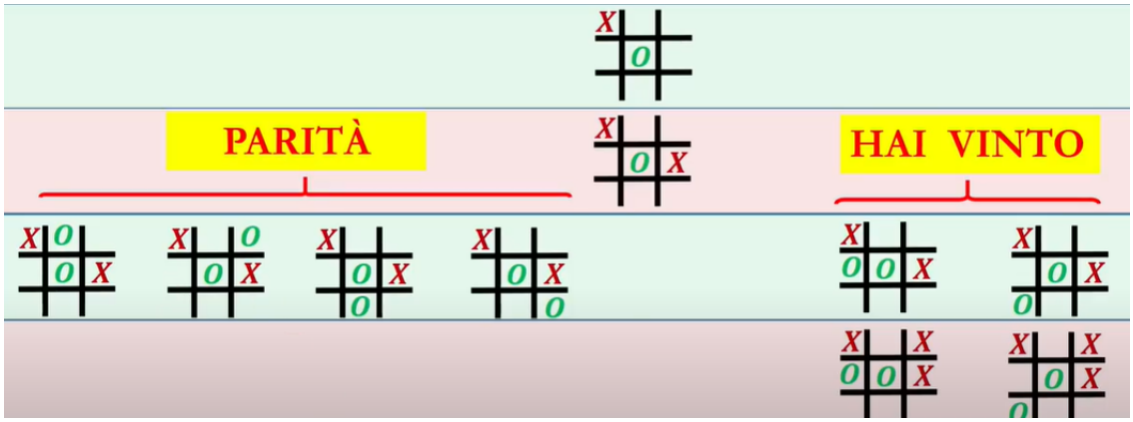
Illustriamo adesso come condurre la partita nell'ultimo caso rimasto, ossia quando l'avversario effettua una giocata perfetta.
Si risponderà, da parte nostra, con una X sulla stessa diagonale e all'avversario resteranno 6 disponibilità.

Se gioca ad angolo, la partita verrà definita **vinta**, se gioca non ad angolo, la partita verrà definita in **parità** (avversario perfetto, con noi perfetti, al più andremo a pareggiare). Come faccio a vincere? Semplice, anche in questo caso si metterà una X in modo da aprire due tris e l'avversario non potrà difendersi.



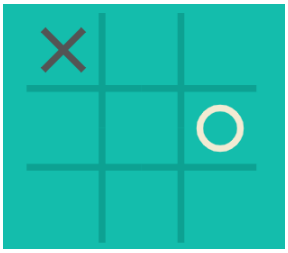


Se sappiamo che l'avversario conosca già le eventuali contromosse, si può applicare una **variante**, cioè giocare la X non in un'angolo ma accanto alla O dell'avversario, vediamo come:

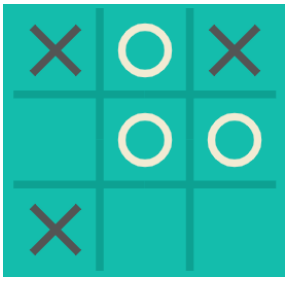


4.1 TEST

Per testare se davvero tutte queste analisi e ricerche sono veritiere, è stato fatto un **TEST** sia nel caso di avversario perfetto che non, per vedere, che nel caso di avversario perfetto, e noi perfetti, si finirà in parità.
Iniziamo con l'avversario non perfetto. (Iniziamo noi con la X).

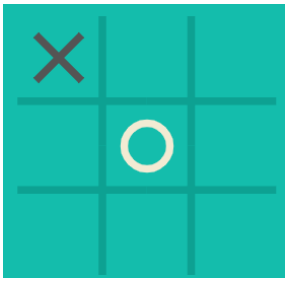


Come notiamo, l'avversario ha fatto la scelta di posizionare la O al centro su un lato e, tornando allo studio fatto prima sappiamo che, se non facciamo errori, sicuramente vinceremo la partita.

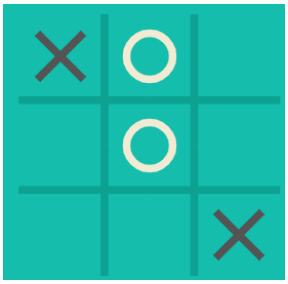


Esempio **verificato**.

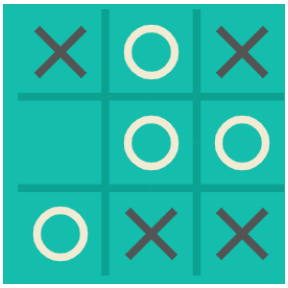
Vediamo adesso il caso di avversario perfetto.



Come ci aspettavamo, l'avversario ha fatto la scelta giusta.



Possiamo notare che, se si continua a giocare nel modo perfetto, arriveremo al caso di parità.



Esempio **verificato**.

5 Saluti Finali

Siamo giunti alla fine di questa introduzione al mondo della teoria dei giochi, dove sono state trattate nozioni principali, algoritmi con le relative proprietà, il caso delle strategie approssimate e il come influisce il fattore del Non Determinismo.
Inoltre, nell'ultimo capitolo, è stato portato un'esempio di gioco con i relativi test.

Grazie per l'attenzione.

Pietro Conte