TetrAIs

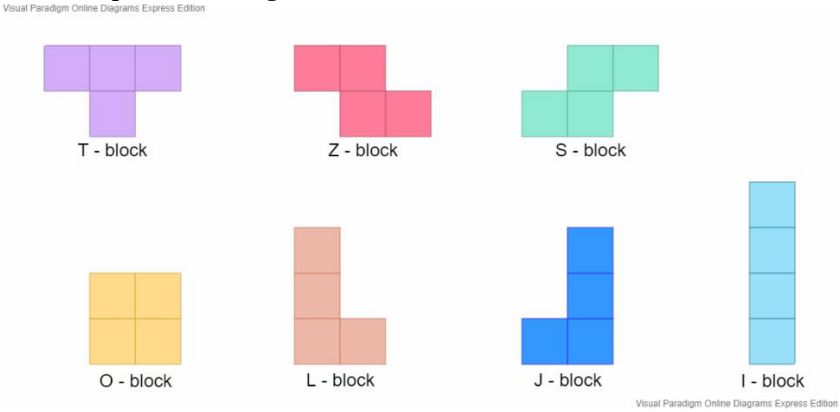
Christian Miccolis	Davide Paduanelli	Mattia Patruno
Matricola: 683313	Matricola: 683127	Matricola: 676401
christian-miccolis@libero.it	davide.paduanelli@gmail.com	m3ttiw@gmail.com
Repository link: https://github.com/Chrism1c/TetrAIs		

Indice

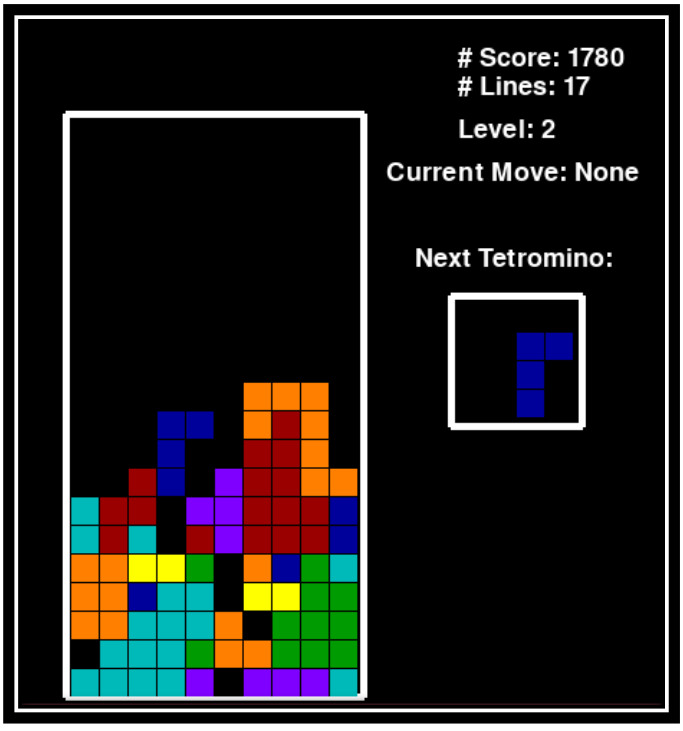
1. Introduzione

Tetris è un videogioco classico pubblicato a metà degli anni '80. Il gioco prevede la disposizione di sette diversi tipi di blocchi (anche noti come "Tetramini") per creare righe lungo l'area di gioco. I blocchi cadono dall'alto verso il basso sullo schema di gioco e possono solo essere ruotati o spostati a sinistra o a destra dal giocatore. Ogni volta che viene posizionato un blocco, uno nuovo blocco casuale inizierà a scendere dalla parte superiore dello schermo. A seconda della variante del gioco, è possibile che lo schema di gioco sia a conoscenza del tipo di blocco successivo oppure no. I blocchi cadono in una griglia alta 20 quadrati e larga 10. Ogni volta che una riga viene occupata interamente, essa scompare lasciando scoperta la riga inferiore. Quando la pila di blocchi raggiunge la cima della griglia, il gioco termina. Poiché i blocchi in arrivo non possono essere previsti, il pianificatore di blocchi deve essere in grado di adattarsi a diversi modelli. Il gioco Tetris risulta essere un gioco invincibile in maniera assoluta, poiché dipende strettamente dalla sequenza di blocchi che vengono generati, di conseguenza un eventuale combinazione infausta costituita da blocchi come la S e la Z condurrebbe la partita ad una conclusione rapida e inevitabile.

- I sette "Tetramini" presenti nel gioco:



- La schermata di gioco di TetrAIs:



2. Funzionalità

TetrAIs dispone di alcune funzionalità aggiuntive che permettono maggiore comprensione riguardo le azioni svolte dalle intelligenze artificiali durante la loro esecuzione.

2.1 Guide Side Panel
2.2 Decisional Tree Plot
2.3 Result Plot
2.4 Real Time Console Prints
2.5 Logger

3. Intelligenze Artificiali

TetrAIs ha a disposizione sei differenti agenti autonomi basati su sei differenti algoritmi di intelligenza artificiale, in grado di operare sullo schema di gioco seguendo diversi approcci per l'ottenimento dello score più alto possibile.

3.1 Deep First Search

Un algoritmo generico di ricerca è indipendente da qualsiasi strategia di ricerca e/o grafo. L'idea è che dato un grafo, si esplorano incrementalmente i percorsi a partire dai nodi di partenza per poi giungere ai nodi-obiettivo.

Si mantiene una frontiera di percorsi già esplorati collegati ad un nodo di partenza, i quali potrebbero costituire segmenti iniziali di percorsi completi verso nodi-obiettivo.

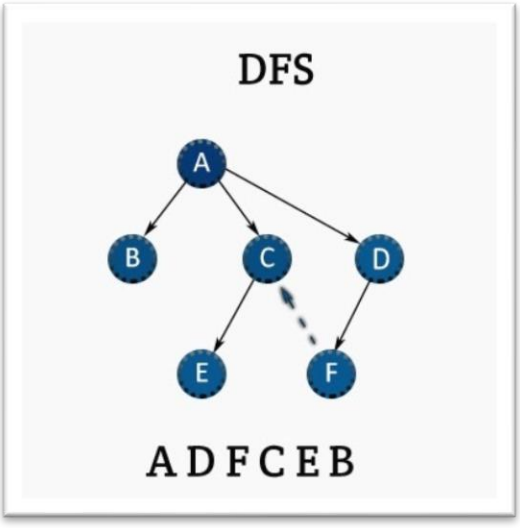
Inizialmente la frontiera è costituita da percorsi semplici che si rivelano essere nodi di partenza. Successivamente vi è un'espansione dei percorsi verso nodi inesplorati fino ad incontrare nodi-obiettivo:

- Si seleziona (e si rimuove dalla frontiera) un percorso
- Si estende il percorso con ogni arco uscente dall'ultimo nodo
- Si aggiungono alla frontiera tali percorsi

Il DFS è un algoritmo di ricerca non informata sui grafi, in cui il sistema ragiona su un modello del mondo fatto di stati, in assenza di incertezza e con finalità da raggiungere:

- Una rappresentazione piatta del dominio,
- Nello spazio degli stati, si cerca un modo per andare dallo stato corrente a un obiettivo.

Nella ricerca in profondità (DFS), la frontiera è organizzata come una pila (LIFO) in cui gli elementi vengono aggiunti uno alla volta e quello selezionato e prelevato sarà l'ultimo aggiunto.



3.2 Stochastic Gradient Descent & Q-Learning

3.2.1 Gradient Descent

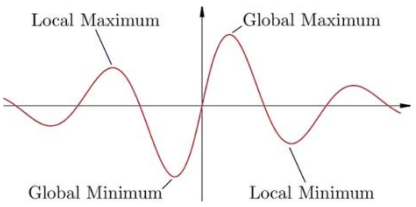
Per comprendere come funziona "'SGD" è necessario comprendere che cos'è la Discesa Gradiente "DG". Gradient Descent "DG" è una tecnica di ottimizzazione molto popolare in Machine Learning e Deep Learning e può essere utilizzata con la maggior parte, se non tutti gli algoritmi di apprendimento. Un gradiente è fondamentalmente la pendenza di una funzione. Matematicamente, può essere descritto come derivate parziali di un insieme di parametri rispetto ai suoi input. La discesa del gradiente può essere descritta come un metodo iterativo che viene utilizzato per trovare i valori dei parametri di una funzione che minimizza il più possibile la funzione di costo. Inizialmente i parametri vengono definiti con un valore particolare e, da quel momento, la Discesa Gradiente viene eseguita in modo iterativo per trovare i valori ottimali dei parametri, usando il calcolo, per trovare il valore minimo possibile della funzione di costo data.

3.2.2 Stochastic Gradient Descent

La discesa stocastica del gradiente è un metodo per trovare la configurazione ottimale dei parametri per un algoritmo di apprendimento automatico. Esso esegue in modo iterativo piccole modifiche alla configurazione di una rete di apprendimento automatico per ridurre l'errore della rete.

Le funzioni di errore sono raramente semplici come una tipica parabola. Molto spesso hanno molte colline e vallate.

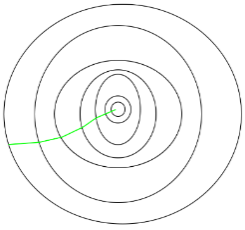
In questo grafico, se la discesa del gradiente dovesse iniziare sul lato sinistro del grafico, si fermerebbe nella valle sinistra perché, indipendentemente da quale direzione si viaggi da questo punto, è necessario spostarsi verso l'alto. Questo punto è noto come minimo locale. Tuttavia, esiste un altro punto nel grafico che è inferiore. Il punto più basso dell'intero grafico è il minimo globale, che è ciò che la discesa del gradiente stocastico tenta di trovare.



La discesa stocastica del gradiente tenta di trovare il minimo globale regolando la configurazione della rete dopo ciascun punto di allenamento. Invece di ridurre l'errore o trovare il gradiente per l'intero set di dati, questo metodo riduce semplicemente l'errore approssimando il gradiente per un batch selezionato casualmente (che può essere piccolo come un singolo campione di addestramento). In pratica, la selezione casuale viene ottenuta mescolando casualmente il set di dati e lavorando in batch in modo graduale.

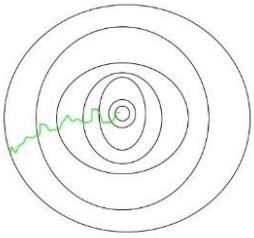
Dal punto di vista euristico, se la rete sbaglia un esempio di addestramento, aggiornerà la configurazione in favore di farlo correttamente in futuro. Tuttavia, l'aggiornamento della configurazione potrebbe comportare errori nel porre altre domande, aumentando così l'errore generale della rete. Pertanto, non tutte le iterazioni di addestramento possono migliorare la rete attraverso l'algoritmo di discesa gradiente stocastico.

D'altro canto, la discesa gradiente stocastica può regolare i parametri di rete in modo da spostare il modello da un minimo locale a un minimo globale. Guardando indietro alla funzione concava nella foto sopra, dopo aver elaborato un esempio di addestramento, l'algoritmo può scegliere di spostarsi a destra sul grafico per uscire dal minimo locale in cui ci trovavamo. Anche se così facendo aumenta l'errore della rete, gli consente di spostarsi sulla collina. Ciò consentirà un ulteriore addestramento per indurre la discesa del gradiente a spostarsi verso il minimo globale.



Un vantaggio della discesa del gradiente stocastico è che richiede molto meno calcolo rispetto alla vera discesa del gradiente (ed è quindi più veloce da calcolare), mentre generalmente converge al minimo (sebbene non necessariamente globale).

← Percorso seguito DG batch | Percorso seguito SGD →



Una cosa da notare è che, poiché la SGD è generalmente più rumorosa della tipica Discesa a gradiente, di solito ci sono voluti un numero maggiore di iterazioni per raggiungere i minimi, a causa della sua casualità nella sua discesa. Anche se richiede un numero maggiore di iterazioni per raggiungere i minimi rispetto alla tipica discesa con gradiente, è comunque computazionalmente molto meno costoso della tipica discesa con gradiente. Pertanto, nella maggior parte degli scenari, SGD è preferito rispetto alla Discendenza gradiente batch per ottimizzare un algoritmo di apprendimento.

3.2.3 Apprendimento per rinforzo

Il Q-learning è un algoritmo basato sull'apprendimento per rinforzo, nonché un'area del Machine Learning. L'apprendimento per rinforzo consiste nell'intraprendere azioni adeguate a massimizzare la ricompensa in una situazione particolare. L'apprendimento per rinforzo differisce dall'apprendimento supervisionato in un modo in cui nell'apprendimento supervisionato i dati di addestramento hanno la chiave di risposta con esso, quindi il modello viene addestrato con la risposta corretta stessa, mentre, nell'apprendimento per rinforzo non c'è risposta ma l'agente di rinforzo decide cosa fare per eseguire l'attività specificata. In assenza di un set di dati di formazione, è tenuto a imparare dalla sua esperienza. Esistono due tipi di rinforzo:

Positivo – se il rinforzo positivo quando è definito nel momento in cui un evento ha un effetto positivo sul comportamento. Troppo rinforzo può portare a un sovraccarico di stati che può ridurre i risultati

Negativo – Il rinforzo negativo è definito come il rafforzamento di un comportamento perché una condizione negativa viene fermata o evitata.

3.2.4 Q-Learning

Il Q-Learning è una forma base di Reinforcement Learning che utilizza i Q-value (chiamati anche valori di azione) per migliorare iterativamente il comportamento dell'agente di apprendimento.

I Q-value o valori azione sono definiti per stati e azioni. Q (S, A) è una stima di quanto sia buono intraprendere l'azione A nello stato S. Questa stima di Q (S, A) sarà calcolata iterativamente usando la regola TD-Update.

Rewards ed Episodi: un agente nel corso della sua vita inizia da uno stato iniziale, effettua una serie di transizioni dal suo stato corrente a uno stato successivo in base alla sua scelta di azione e anche all'ambiente in cui l'agente interagisce. Ad ogni passo di transizione, l'agente da uno stato compie un'azione, ottiene una ricompensa dall'ambiente e quindi passa a un altro stato. Se in qualsiasi momento l'agente finisce in uno degli stati di terminazione, ciò significa che non è possibile alcuna ulteriore transizione. Si dice che questo sia il completamento di un episodio.

Differenza temporale / TD-Update: La regola TD-Update può essere rappresentata come:

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{vecchio valore}} + \underbrace{\alpha_t(s_t,a_t)}_{\text{tasso di apprendimento}} \times \Bigg[\overbrace{\underbrace{r_t}_{\text{ricompensa}} + \underbrace{\gamma}_{\text{fattore di sconto}} \underbrace{\max_{a_{t+1}} Q(s_{t+1},a_{t+1})}_{\text{valore futuro massimo}}}^{\text{valore appreso}} - \underbrace{Q(s_t,a_t)}_{\text{vecchio valore}} \Bigg]$$

Questa regola di aggiornamento per stimare il valore di Q viene applicata in ogni momento dell'interazione degli agenti con l'ambiente. I termini utilizzati sono spiegati di seguito:

s: Stato attuale dell'agente. **s+1** : Next State dove finisce l'agente.
a: Azione corrente scelta in base ad alcune politiche.
a+1: La migliore azione successiva da scegliere usando la stima del Q-value corrente, ovvero selezionare l'azione con il Q-value massimo nello stato successivo.
r: Ricompensa attuale osservata dall'ambiente in risposta all'azione corrente.

γ (0 < γ <= 1): Fattore di sconto per premi futuri. Le future ricompense hanno meno valore delle ricompense attuali, quindi devono essere scontate. Poiché il Q-value è una stima dei premi attesi da uno stato.

α: (0 < γ <= 1): Tasso di apprendimento, rispetto al passo preso per aggiornare la stima di Q (S, A).

Scegliere l'azione da intraprendere usando la politica epsilon-greedy: La politica epsilon-greedy è una politica molto semplice di scelta delle azioni usando le attuali stime del Q-value. Va come segue:

- Con probabilità (1 - epsilon) scegli l'azione che ha il Q-value più alto.
- Con probabilità (epsilon) scegli qualsiasi azione a caso.

3.3 Genetic – Beam

3.3.1 Genetico

Un algoritmo genetico è un algoritmo euristico basato su popolazioni, utilizzato per tentare di risolvere problemi di ottimizzazione per i quali non si conoscono altri algoritmi efficienti di complessità lineare o polinomiale. L'aggettivo "genetico", ispirato al principio della selezione naturale ed evoluzione biologica teorizzato nel 1859 da Charles Darwin, deriva dal fatto che, al pari del modello evolutivo darwiniano che trova spiegazioni nella branca della biologia detta genetica, gli algoritmi genetici attuano dei meccanismi concettualmente simili a quelli dei processi biochimici scoperti da questa scienza.

L'algoritmo genetico prevede una successione di n generazioni composte da un numero fisso o variabile di cromosomi. Ogni cromosoma rappresenta un individuo della popolazione ed è composto da un numero fisso di geni. Ogni gene, proprio come negli esseri viventi, è responsabile di una variazione nelle caratteristiche dell'individuo.

Ogni nuovo elemento di una popolazione è generato combinando le assegnazioni di una coppia di individui genitori. Il crossover consiste nel:

- seleziona coppia di individui
- genera prole prendendo i valori alcune variabili da 1 genitore e resto da altro genitore

Data una popolazione di k individui, si generano nuovi individui fino a trovare una soluzione:

- Selezione casuale di coppie, gli individui con valutazione migliore hanno probabilità maggiore di essere scelti.
- ∀ coppia → crossover;
- si mutano casualmente alcuni, pochi, valori

Tipi di crossover:

- Crossover uniforme: per ogni variabile il valore dei figli dipende da uno dei due genitori
- one-point crossover: data una relazione d'ordine sulle variabile, per ogni coppia di figli da generare si seleziona casualmente un indice i. Per il primo figlio i valori delle variabili fino alla i-esima provengono da un genitore e i restanti dall'altro. Per il secondo figlio si agisce in maniera complementare.

3.3.2 Beam Search

Beam search è un algoritmo di ricerca basato su euristiche che esplora un grafo espandendo il nodo più promettente in un insieme limitato di nodi.

La beam search è un caso particolare di best-first search che mira a ridurre i requisiti di memoria. Best-first search è una ricerca su grafi che ordina tutte le soluzioni parziali (stati) in base ad una certa euristica. Nella beam search è tenuto in considerazione solo un numero predefinito di migliori soluzioni parziali. Di conseguenza, è considerato un algoritmo greedy.

3.4 Blind Bandit Monte Carlo

L'MCTS (Monte Carlo Tree Search) è una strategia di ricerca euristica adottata in alcuni tipi di processi decisionali, come ad esempio quei processi decisionali che tipicamente si adottano nei giochi.

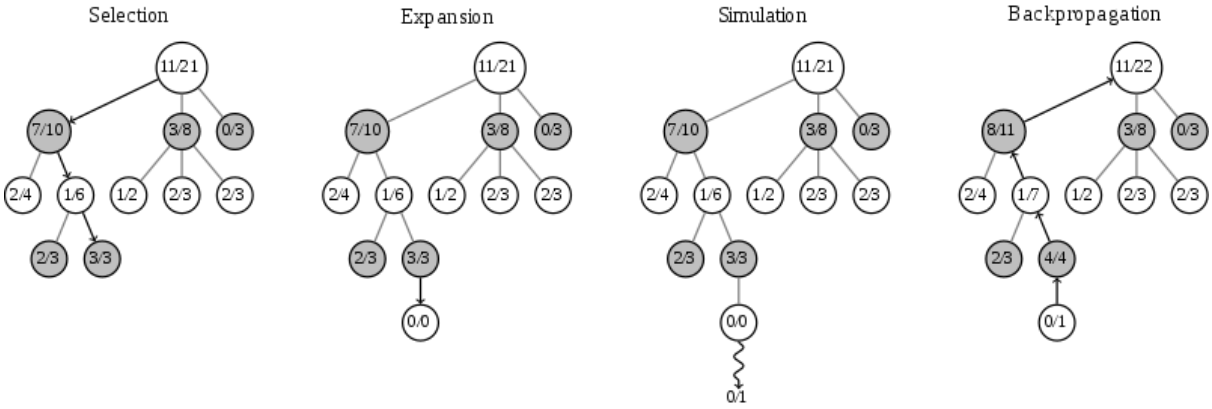
Vengono applicati metodi Markov Chain Monte Carlo (MCMC) per generare campioni:

- Viene costruita una catena di campioni con la distribuzione-obiettivo come sua sola distribuzione stazionaria e quindi campionare dalla catena;
    - Campioni distribuiti secondo la distribuzione-obiettivo;
- Riscaldamento/burn-in: è possibile scartare i primi campioni, quelli considerati più lontani dalla distribuzione stazionaria.

L'obiettivo dell'MCTS è analizzare i playout più promettenti, espandendo l'albero di ricerca, il quale è basato su un random sampling dello spazio di ricerca. Il risultato finale di ogni playout, viene utilizzato per pesare i nodi dell'albero di ricerca, cosicché successivamente i nodi migliori abbiano più possibilità di essere scelti per i playout futuri.

Ogni round della scelta della mossa nell'MCTS è composto da quattro passi:

- **Selection**: si parte da una radice R e si selezionano tutti i successivi nodi figli, finché un nodo foglia L viene raggiunto. Il root è lo stato corrente di gioco e la foglia è qualsiasi nodo oltre il quale non si sia fatta alcuna simulazione di playout.
- **Expansion**: a meno che L non termini il gioco (gameover), vengono creati uno o più nodi figli e si sceglie un nodo C da essi. I nodi figli sono l'applicazione di qualsiasi mossa valida allo stato di gioco salvato nel nodo L.
- **Simulation**: completa un playout random dal nodo C. Un playout può essere semplice come decidere mosse random uniformi che cambino lo stato del gioco in una maniera predefinita (ad esempio che ottenga il punteggio massimo o che rispetti determinate caratteristiche).
- **Backpropagation**: viene utilizzato il risultato del playout per aggiornare le informazioni nei nodi presenti nel cammino dal nodo C al nodo R.



In entrambe le varianti la difficoltà principale nella selezione dei figli è mantenere un equilibrio tra l'utilizzo di varianti più profonde e quindi più complesse, ma con un punteggio più alto e l'esplorazione di mosse più semplici con uno sviluppo in profondità inferiore (e quindi un numero di simulazioni inferiore). Per ovviare a tale problema si è soliti utilizzare una formula che equilibri tali parametri, tale formula è chiamata UCT (Upper-Bound Confidence applied to Trees). Tuttavia, nella nostra implementazione abbiamo deciso di non applicare alla lettera la formula classica dell'UCT, ma di utilizzare il punteggio della simulazione come parametro di riferimento.

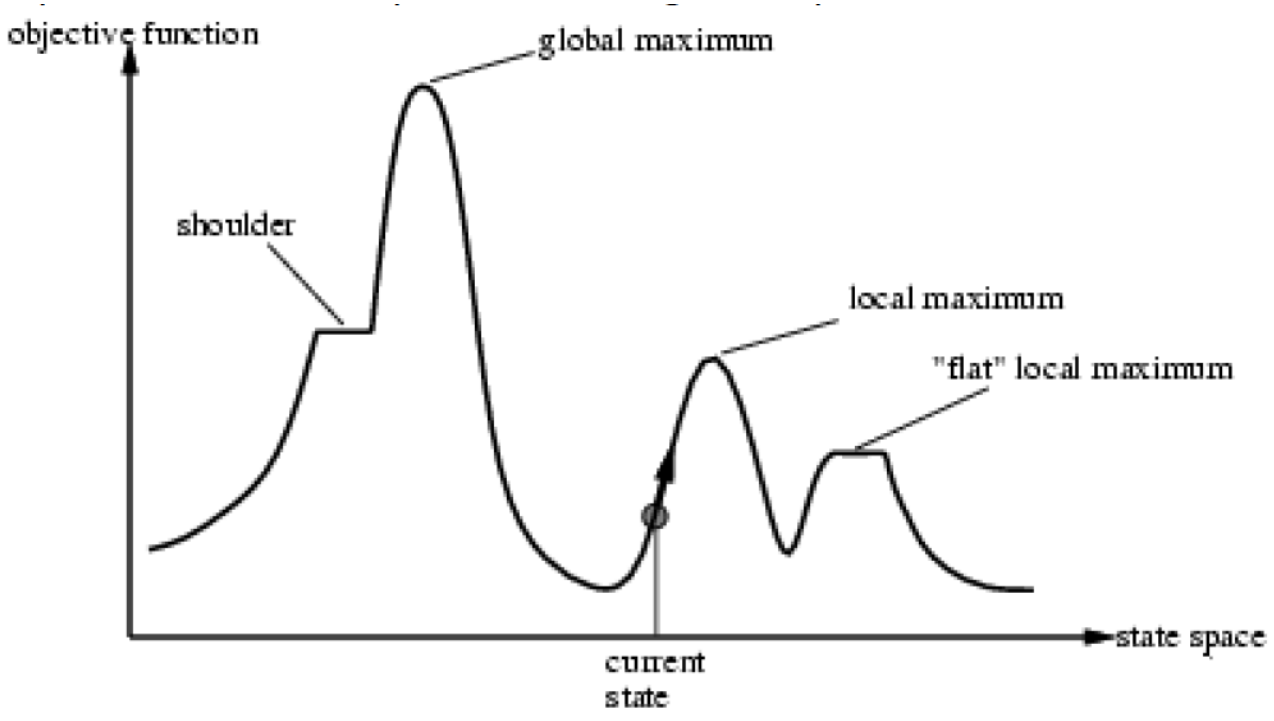
3.5 Basato su Regole logiche

Per poter rendere efficace l'agente basato su regole assumiamo conoscenza completa: ad ogni passo, la base di conoscenza ha una visione completa del mondo analizzato (closed-world assumption). Gli assiomi che descrivono il mondo sono composti da atomi la cui veridicità è confermata dal completamento di Clark: se almeno uno degli atomi che compongono una clausola è vero, allora otteniamo una relazione di equivalenza tra la testa della clausola e il suo corpo. se la clausola non ha corpo, allora si assume che ci sia un equivalenza tra la clausola e false.

Le conseguenze logiche dedotte dalla base di conoscenza seguono il ragionamento non monotono: le conclusioni sono invalidate con aggiunta di altra conoscenza.

3.6 Ricerca Locale

La ricerca locale greedy è un tipo di algoritmo di ricerca locale basato sul Miglioramento Iterativo della situazione corrente tramite l'esplorazione dei nodi vicini. Essendo una ricerca locale, l'algoritmo di ricerca greedy limita l'esplorazione in uno spazio di ricerca limitato, quello dei nodi vicini al nodo corrente. Questo tipo di ricerca locale viene detto "greedy" (goloso) in quanto seleziona soltanto la via migliore tra tutte quelle immediatamente disponibili senza considerare le conseguenze della scelta nei passi successivi.
Esso dimostra la sua utilità quando gli spazi sono molto grandi o infiniti poiché non effettua una ricerca sistematica. L'algoritmo di ricerca locale greedy seleziona il miglior successore dell'assegnazione corrente in termini di una funzione di valutazione (es. costo). Ne esistono due differenti varianti:
1) Greedy Descent: funzione da minimizzare 2) Greedy Ascent: funzione da massimizzare
La funzione di valutazione considera il numero di conflitti, ossia di vincoli violati; essa si può raffinare pesando tali vincoli in maniera diversa. L'ottimo locale è un'assegnazione tale da non essere migliorabile da alcun successore (minimo/massimo locale nel greedy descent / ascent). L'ottimo globale è il miglior valore tra tutte assegnazioni. L'Ottimo globale è sempre un ottimo locale, se la ricerca trova un minimo locale, non si può sapere se esso sia un minimo globale. L'algoritmo è completo poiché considera il miglior successore anche quando esso non migliora la valutazione rispetto all'assegnazione corrente.



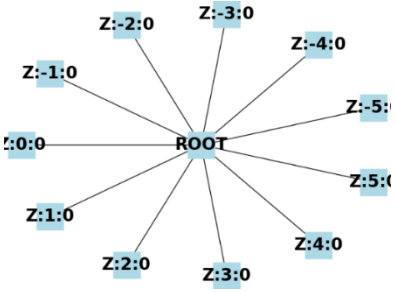
4. Implementazione

L' implementazione adottata per TetrAIs è basata interamente sull'utilizzo del linguaggio python 3, sia per il lato (front -end) grafico sia per la parte (back-end) dedicata al ragionamento, controllo e scelta delle mosse da svolgere, l'unica eccezione è rappresentata dall'AI Basata su regole logiche che presenta un bridge per la comunicazione con una base di conoscenza scritta nel linguaggio prolog (Swi-prolog)

4.1 Deep First Search

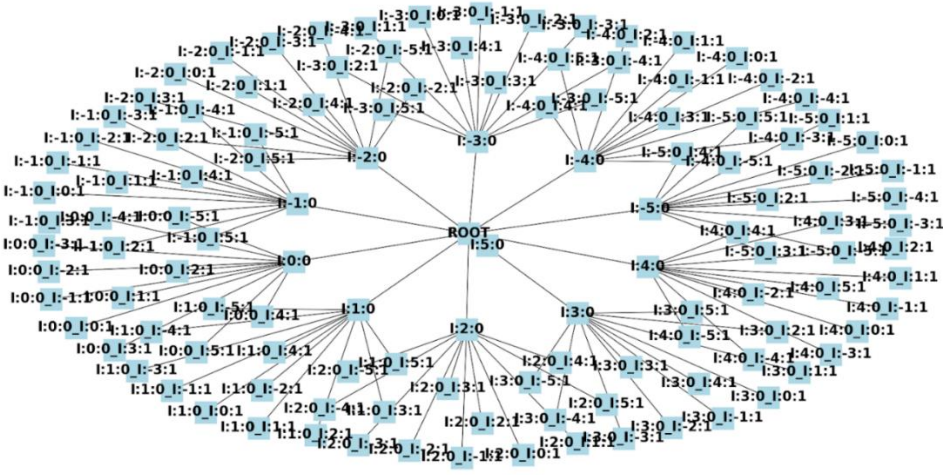
Nel gioco del Tetris, non è presente un goal univoco e si cerca, quindi, di ottenere il massimo punteggio possibile. Per raggiungere tale scopo si utilizzano la board corrente, il tetramino corrente e il prossimo tetramino generato (per il DFS_LV1 si tiene conto solo del tetramino corrente, mentre per il DFS_FULL si tiene conto di entrambi i tetramini, sia quello corrente che il prossimo disponibile).

Nel DFS_LV1 viene passata la board corrente e il "falling_piece". Lo stato iniziale è costituito dalla board vuota e dal primo tetramino, il quale viene sempre scelto in maniera randomica a ogni inizio partita. I successivi nodi sono tutte le possibili posizioni e rotazioni in cui il tetramino può essere piazzato, verrà selezionato lo stato in cui il posizionamento e la rotazione del tetramino produce lo score più alto. Successivamente si procede ricorsivamente sui tetramini che vengono generati man mano.



Lo score viene calcolato in base ad alcune metriche ottenibili dalla "board", quali ad esempio: il numero di "fori" generati dal posizionamento dei tetramini, il punto più alto raggiunto dai tetramini, ecc…

Il DFS_FULL si differenzia dal DFS_LV1, poiché tiene conto sia del tetramino corrente, che del successivo, in modo da trovare una combinazione, in posizione e rotazione dei due tetramini, che generi lo score più alto possibile.



X:Y:Z = Pezzo:Sideway:Profondità X:Y:Z_W:K:J = PezzoPadre:SidewayPadre:ProfonditàPadre_Pezzo:Sideway:Profondità

4.2 Stochastic Gradient Descent & Q-Learning

L'algoritmo SDG_QL è basato sull'utilizzo dell'algoritmo di Acesa del Gradiente Stocastica come ottimizzazione dell'algoritmo di reinforcement learning noto come Q-Learning

L'implementazione che fa uso di tale algoritmo per giocare a Tetris utilizza un "vettore di pesi" rappresentanti l'importanza che ogni metrica possiede all'interno della funzione di calcolo dello score. Per poter scegliere la mossa migliore da svolgere dato un tetramino e uno schema di gioco (State), l'algoritmo confronta le possibili mosse (Action) (costituite da rotazione e sideway) riguardanti solo il tetramino corrente, mediante delle apposite simulazioni di "drop" del tetramino sulla board corrente.

Ogni volta che in una simulazione il tetramino è inserito in una posizione lecita, l'AI effettua il calcolo della reward relativa all'esecuzione dell'Action sullo State corrente. Il valore di reward è calcolato mediante la funzione: Reward = 5 * (lines_removed * lines_removed) - (height_sum - reference_height)

Nonché, la differenza tra il quintuplo del quadrato delle linee rimosse dalla singola azione e la differenza tra la somma delle altezze dopo l'Action e la somma delle altezze prima dell'Action. Tale funzione di reward è in grado di premiare maggiormente le mosse che eliminano più linee contemporaneamente (lines_removed) accumulando di conseguenza un valore di score più elevato attraverso i moltiplicatori.

In questo modo si fornisce un feedback diretto relativo all'azione svolta dalla AI basata sul "vettore di pesi" corrente. A questo punto, viene eseguita la regola di TD-Update dell'algoritmo Q-learning che restituisce il "Q-value" nonché il nuovo peso nel vettore ("gene" e "Cromosoma" per analogia al Genetico)

wx[i] = wx[i] + alpha * wx[i] * (one_step_reward - old_params[i] + gamma * new_params[i])

Dove alpha è il learning rate pari a 0.01, gamma è il fattore di sconto pari a 0.9, "new_parmas" e "old_parms" sono rispettivamente le metriche ottenute sulla board (State) dopo la mossa (Action+1) e le metriche precedenti alla mossa (Action). In questo modo il "vettore di pesi" viene costantemente aggiornato mossa per mossa andando ad influire di conseguenza sulla Action successiva. In fine, prima di procedere, il "Cromosoma" viene normalizzato in modo tale che la somma dei "geni" sia sempre pari a 100 (100%) e che ogni "gene" sia nell'intervallo dell'ordine $[10^2$ e $10^{-4}]$.

La policy di scelta dell'Action corrente è determinata da un valore di probabilità "epsilon" di estrarre una Action differente da quella con Q-value più elevato. Il valore di epsilon dopo ogni esecuzione viene ridotto moltiplicandolo per 0.99. In questo modo facendo convergere tale valore verso zero siamo in grado di ottimizzare l'apprendimento del Q-Learning permettendo ad esso di esplorare mosse non convenzionali durante la fase iniziale di learning e poter ottenere feedback utili all'ottimizzazione dei pesi.

L'algoritmo proposto è in grado di aggiornare i propri pesi e convergere verso dei valori sufficienti all'ottenimento di score molto elevati in relativamente poche run di "ottimizzazione / riscaldamento".

4.3 Genetic – Beam

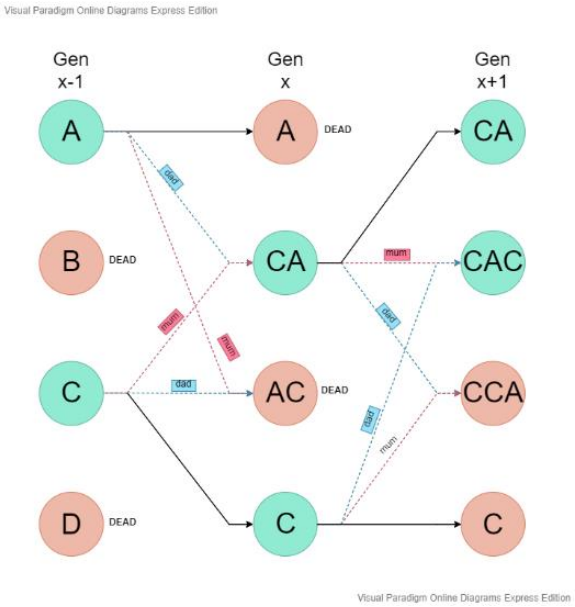
Per la nostra implementazione di questo agente abbiamo deciso di unire la caratteristica riproduttiva dell'algoritmo genetico all'efficienza dell'algoritmo beam search, poiché nel genetico puro l'individuo migliore genera una prole meno performante del genitore.

Ogni gene rappresenta il peso di una delle euristiche utilizzate dal sistema. Una variazione, anche minima di una delle caratteristiche modifica di molto il carattere decisionale dell'agente.

Nella fase di Training, ogni cromosoma effettua più run (3 o 5) e consideriamo la media degli score di ogni partita come punteggio da assegnare al cromosoma.

Quando tutti i cromosomi di una generazione hanno terminato il training, si passa alla fase di selezione della successiva generazione. La "next generation" è composta da:

- ½ migliori cromosomi della generazione precedente, come avviene nella "selezione naturale" (beam search);
- ½ one-point crossing tra i migliori cromosomi della generazione precedente, per creare nuovi cromosomi mescolando i geni che risultano vincenti (genetico), ogni coppia di genitori genera due figli;



Nella fase di crossing selezioniamo due cromosomi fra i migliori e li accoppiamo. Per ogni gene del cromosoma figlio di due cromosomi genitori:

- 20% di possibilità che il gene provenga da uno dei due genitori
- 80% di possibilità che il gene sia una media dei rispettivi geni dei due genitori

In questa fase c'è anche il 10% di possibilità di una lieve mutazione del gene.

Nell'ultima generazione salviamo il miglior cromosoma, che sarà possibile testare nella "Perfect Run".

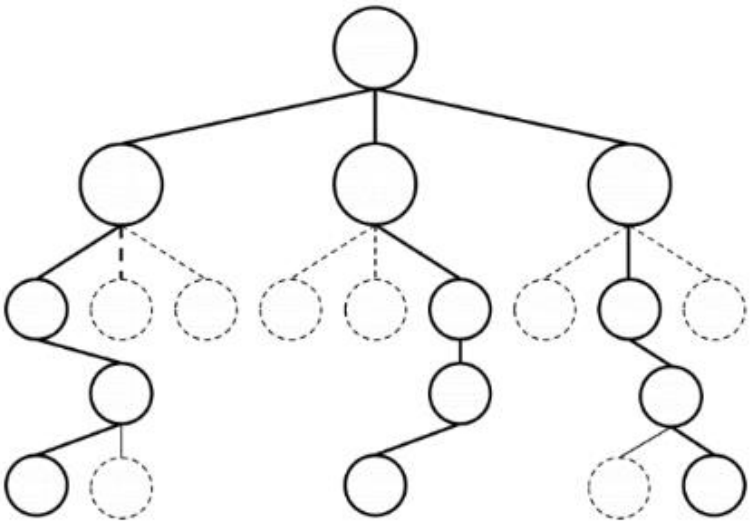
Vantaggi: Anche con un numero non troppo elevato di generazioni, questa tecnica converge sempre verso il miglior cromosoma possibile, mentre il beam search puro necessita mediamente di molte più generazioni per ottenere lo stesso risultato. Il genetico puro, con lo stesso numero di generazioni porta a un individuo che ottiene un punteggio mediamente scarso.

Svantaggi: Il training è molto dispendioso poiché avviene in "real time". Con una popolazione per generazione di 16 individui e un numero di generazioni maggiore di 20, il training può superare le 48 ore. Per questo abbiamo deciso di dare la possibilità di "uccidere" la run dopo un tot di minuti (si consiglia 20).

4.4 Blind Bandit Monte Carlo

Si è deciso di utilizzare due varianti dell'MCTS, la prima è quella in cui viene effettuato un playout di tutte le mosse possibili (più simile ad un algoritmo di ricerca, la generazione dell'albero di ricerca è osservabile in figura 1), mentre la seconda variante è quella del Blind-Bandit in cui viene effettuato un playout basato su un numero randomico di possibili mosse (la generazione dell'albero di ricerca è osservabile in figura 2).

Il Blind-Bandit MCTS ha buone performance anche se deve scegliere playout in periodi di tempo limitati. A differenza degli algoritmi di ricerca (come ad esempio il BFS), non calcola tutti i possibili stati generabili da tutte le possibili mosse valide, bensì calcola una stima della mossa migliore esplorando un numero random di possibili cammini. È probabile che questa variante non restituirà la mossa migliore in assoluto per quel playout, ma restituirà comunque una buona mossa.



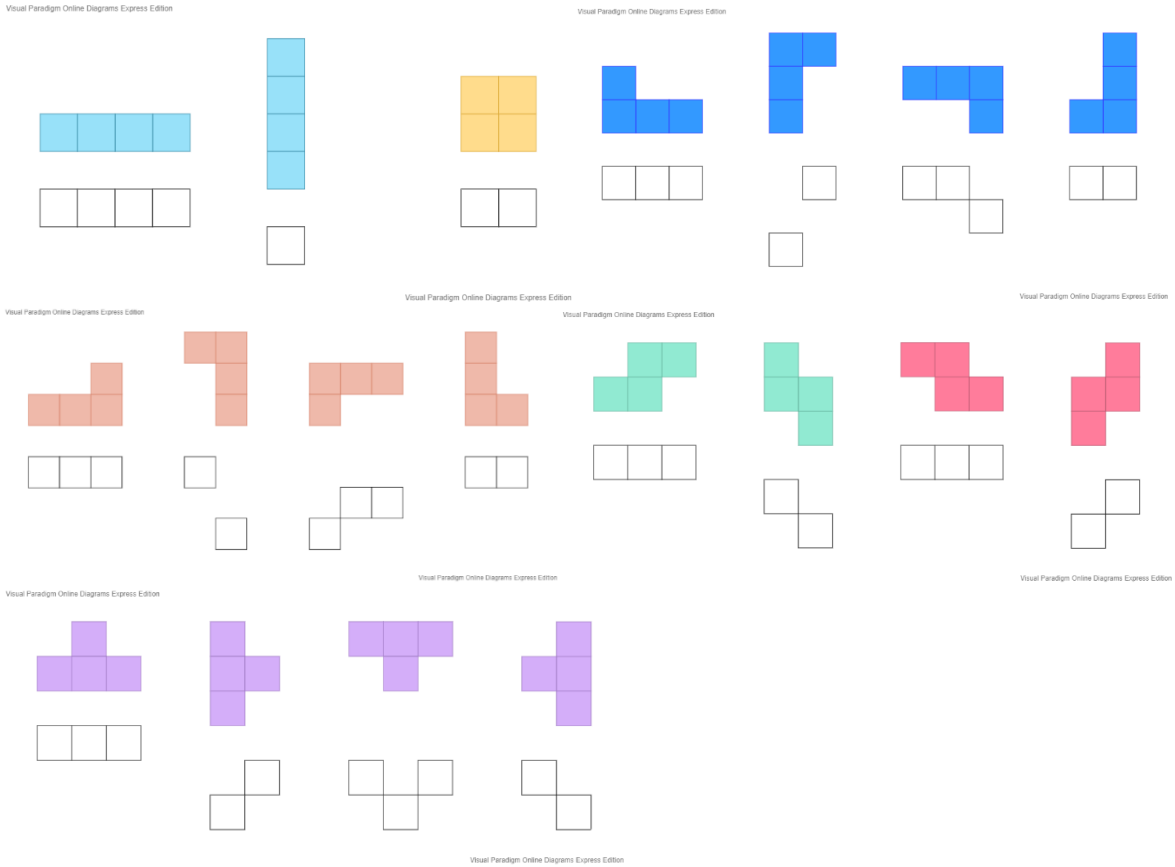
La fase di **Simulation** avviene mediante la funzione **"simulate_board()"** a cui viene passata una "test_board", cioè una copia dello stato attuale della "board", il tetramino e la mossa che si vuole simulare; successivamente avviene la fase di **Selection** in cui viene salvato in un nodo lo stato della "board" modificata dalla mossa. La fase di **Backpropagation** avviene al termine delle due fasi precedenti Il metodo **"get_expected_score()"** è utile per calcolare la nostra implementazione di UCT. La fase di ***Expansion*** avviene ogni qual volta le fasi precedenti terminano.

4.5 Logic Rule Based

Variante Per "Rule Based" Agent, intendiamo un Ai che utilizza una base di conoscenza per trovare la miglior mossa con un tetramino, in una istanza della board.

Codificare tutte le possibili istanze della board, oltre che essere oneroso, lo avrebbe reso un agente non intelligente, poiché quando si parla di AI bisogna solo descrivere la soluzione al problema, non come arrivarci.

Così abbiamo preso in considerazione le "ombre" dei tetramini: ogni tetramino, in ogni sua rotazione, proietta sulla cresta della board una diversa ombra.



In questo modo otteniamo un problema di ottimizzazione della posizione con un "fit", cioè quella parte di cresta (window) che ha la forma dell'ombra del tetramino in una rotazione. Possiamo quindi rappresentare il problema come una rotazione del tetramino e la soluzione del problema come l'ombra della rotazione. La base di conoscenza, scritta in Prolog, codifica:

- un insieme di clausole "statiche": la raccolta "shadow" delle ombre per ogni rotazione di tetramino
- un insieme di clausole "dinamiche": grazie alla keyword "dynamic" la clausola inCrest può essere aggiornata dall'agente prima di interrogare la base di conoscenza
- una regola bestFit: se possibile, associa ad ogni rotazione di tetramino ('shape') una posizione nella cresta ('X')

Interrogando la base di conoscenza con tutte le possibili rotazioni del pezzo in esame otterremo una lista di possibili posizioni. L'agente sceglie la posizione con lo score più alto. Nel caso la query ritorni "false", l'agente eseguirà una mossa casuale, in modo da "rimescolare le carte" per la successiva mossa.

4.6 Ricerca Locale

La variante dell'algoritmo di ricerca locale da noi adottata è quella basata sul miglioramento iterativo greedy Ascent. Essa scansiona solamente il primo livello dell'albero di ricerca, selezionando il nodo rappresentante la mossa che durante la simulazione avrebbe restituito lo score maggiore ottenendo uno stato di ottimo locale. In seguito, l'algoritmo effettua la ricerca dello stato successivo utilizzando il "next tetramino" nei nodi-stato ottenuti dallo stato precedente. In questo modo, l'algoritmo riduce notevolmente il numero di simulazioni totali e quindi raggiunge un ottimo locale.

Poiché la ricerca locale non garantisce l'ottenimento di un ottimo globale e il suo utilizzo è spesso adoperato in situazioni in cui i singoli percorsi risultano essere molto lunghi, nel nostro utilizzo si dimostra particolarmente inefficiente e poco produttivo. Spesso le esecuzioni del gioco che lo utilizzano, terminano con un punteggio pari a zero, questo si verifica poiché i percorsi del grafo sono molto brevi e l'ottimo locale raggiunto non garantisce la permanenza in gioco e l'ottenimento di risultati anche solo paragonabili con quelli degli altri algoritmi implementati.

5. Valutazione e confronto

Ogni agente implementato si approccia al gioco in modo unico, sfruttando tecniche e algoritmi molto differenti. Quindi è utile effettuare un confronto prestazionale per poter determinare i punti di forza e di debolezza di ognuno di essi. Il genetico e l'SDG Q-Learning sono in grado di migliorare le proprie performance nel tempo, mentre DFS, Logic Rule Based, Monte Carlo e Local Search non presentano nessuna forma di apprendimento ma rimangono statici nella loro interazione con il gioco. Sfruttando il Result Plot è possibile visualizzare al termine del training del genetico o al termine delle run per l'SDG_QL come sia variato il punteggio nel tempo e come si siano evoluti / variati i pesi rappresentanti le metriche di score.

Per poter rendere la valutazione il più consistente possibile abbiamo pensato di inserire una modalità di gioco applicabile per ogni AI che fornisca una sequenza deterministica di tetramini; in questo modo è possibile rimuovere l'incertezza dovuta all'estrazione casuale che renderebbe i risultati poco confrontabili.

Per fare ciò si è pensato di utilizzare la caratteristica del "PI greco" di essere un numero aperiodico, infinito e ricalcolabile, come sequenza di indici da associare ai 7 tetramini disponibili nel gioco del Tetris. Così facendo, il circuito su cui le AI dovranno mettersi alla prova risulterà deterministico e l'esecuzione potrà essere replicata in futuro.

Al fine della valutazione e classificazione delle AI su ogni modalità disponibile, di seguito sono riportati i risultati e i grafici ottenuti dai tutti e 3 i membri del gruppo TetrAIs.

5.1 Deep First Search

W1l50n2208:

DFS LV1 Random → 3656 DFS LV2 Random → 28460

m3ttiw:

DFS LV1 Random → 1468 DFS LV2 Random → 17750

Chrism1c:

DFS LV1 Random → 3420 DFS LV2 Random→ 8320

5.1.1 Test deterministico sul "circuito PI Greco":

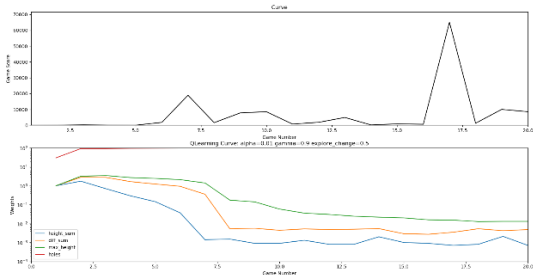
W1l50n2208:

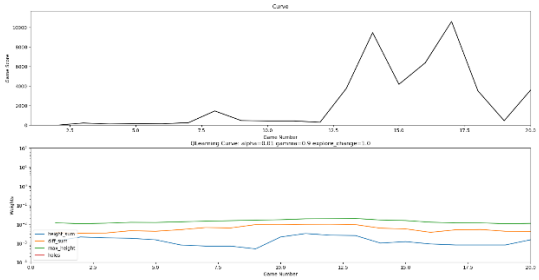
DFS LV1 Random → 800 DFS LV2 Random → 7040

5.2 Stochastic Gradient Descent & Q-Learning

Chrism1c:

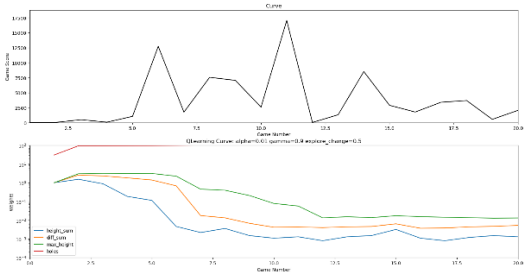
DSG Q-Learning P0.5 20Run Random → **8560** DSG Q-Learning P1.0 20Run Random → **3580**

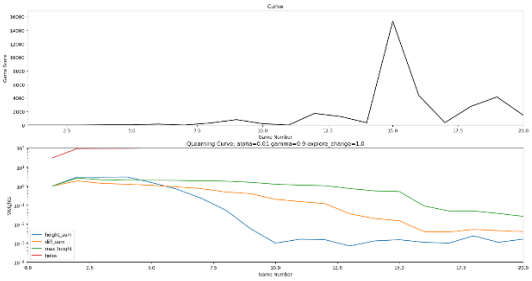




m3ttiw:

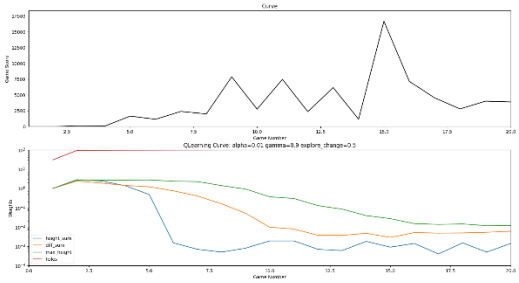
DSG Q-Learning P0.5 20Run Random → **2060** DSG Q-Learning P1.0 20Run Random → **1480**

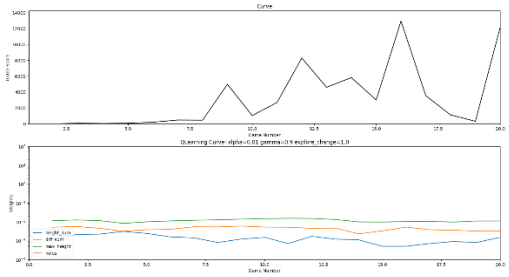




W1l50n2208:

DSG Q-Learning P0.5 20Run Random → **3900** DSG Q-Learning P1.0 20Run Random→ **12140**

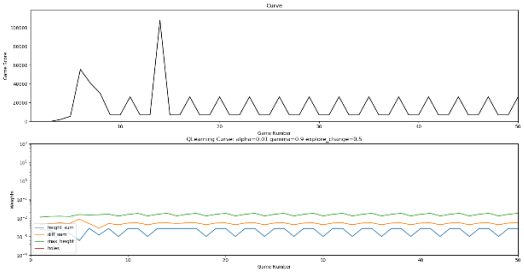


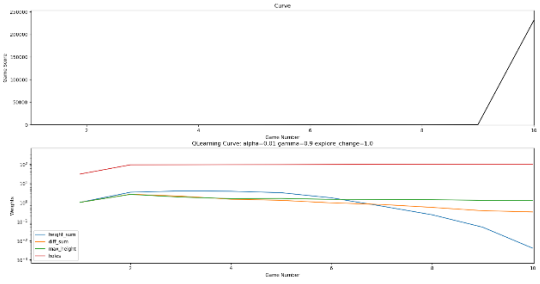


5.2.1 Test deterministico sul "circuito PI Greco":

Chrism1c:

DSG Q-Learning P0.5 50Run PI → **26280** DSG Q-Learning P1.0 10Run PI → **250000++**

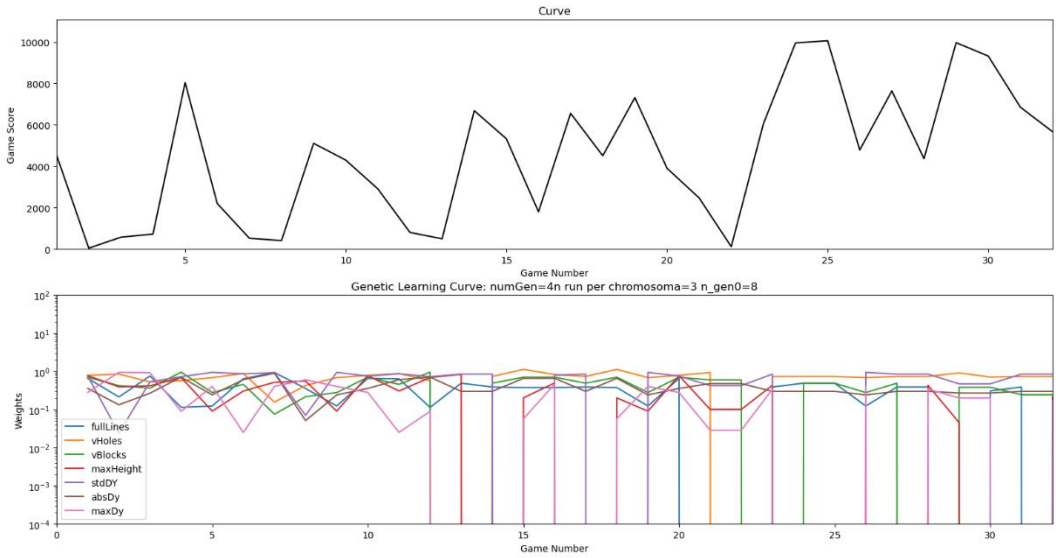




5.3 Genetico

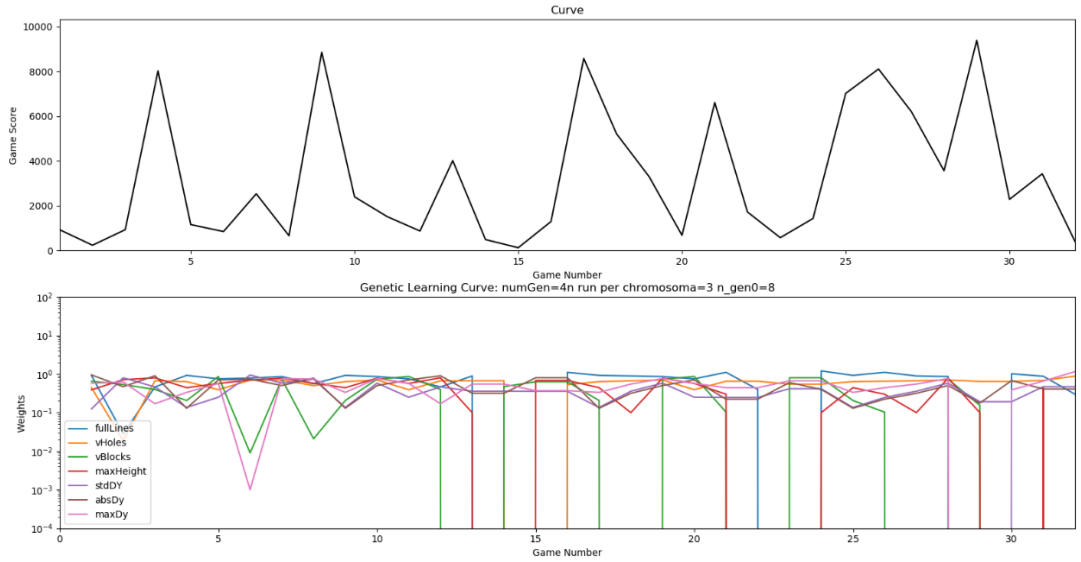
m3ttiw:

4 Gen 8 Cromosomi 3 Run Random→ Risultato Cromosoma perfetto: **90700**



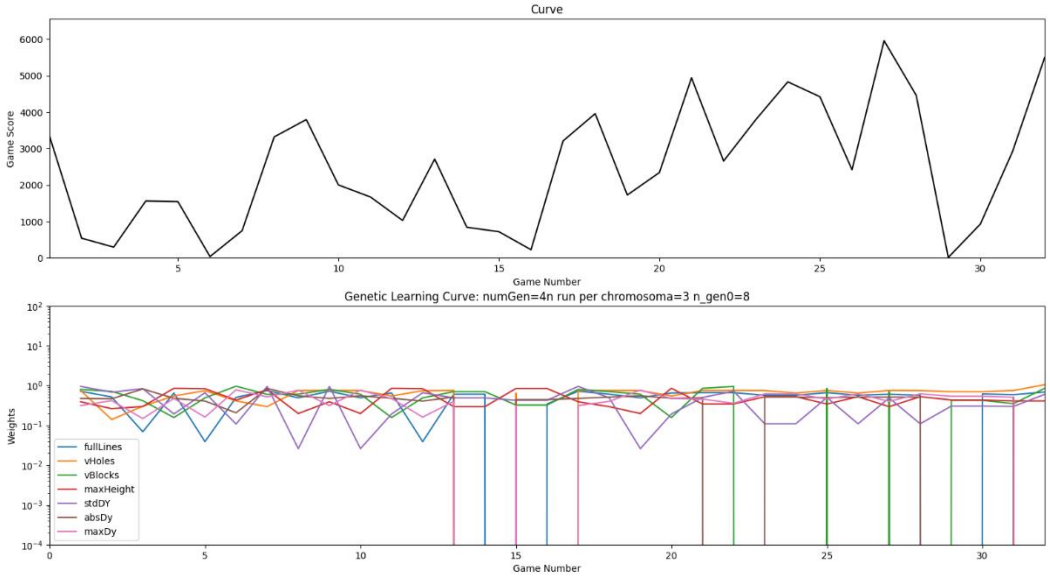
Chrism1c:

4 Gen 8 Cromosomi 3 Run Random→ Risultato Cromosoma perfetto: **71640**



W1l50n2208:

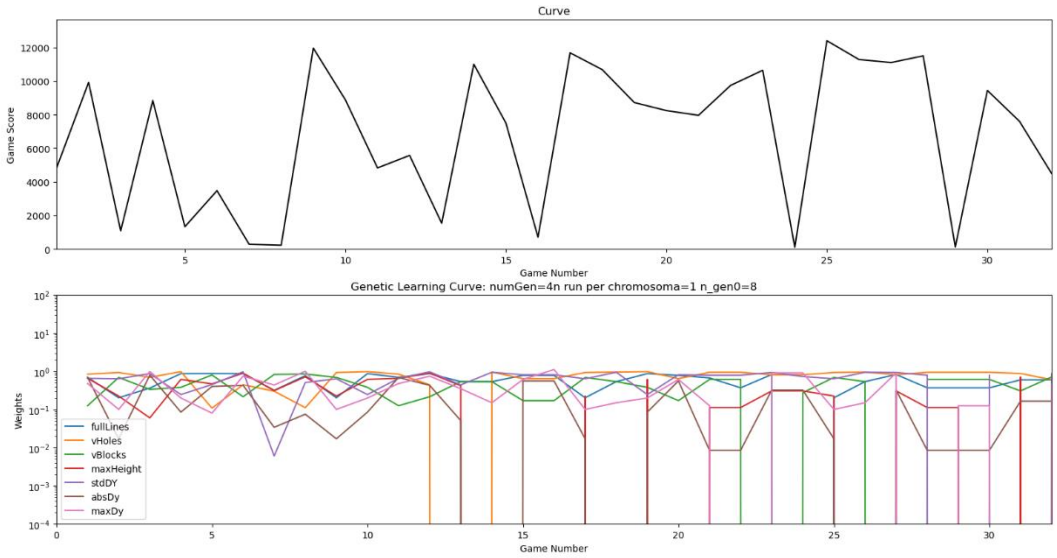
4 Gen 8 Cromosomi 3 Run Random→ Risultato Cromosoma perfetto: **38600**



5.3.1 Test deterministico sul "circuito PI Greco":

m3ttiw:

4 Gen 8 Cromosomi 3 Run PI → Risultato Cromosoma perfetto: **45840**



5.4 Blind Bandit Monte Carlo

W1l50n2208:
MonteCarlo fullScan Random → **1216** MonteCarlo randScan Random → **1120**
m3ttiw:
MonteCarlo fullScan Random → **660** MonteCarlo randScan Random → **1180**
Chrism1c:
MonteCarlo fullScan Random → **1620** MonteCarlo randScan Random → **720**

5.4.1 Test deterministico sul "circuito PI Greco":

W1l50n2208:
MonteCarlo fullScan PI → **2440** MonteCarlo randScan PI → **4340**

5.5 Basato su Regole logiche

m3ttiw:
Logic RuleBased Random → **76**
Chrism1c:
Logic RuleBased Random → **72**
W1l50n2208:
Logic RuleBased Random → **68**

5.5.1 Test deterministico sul "circuito PI Greco":

m3ttiw:
Logic RuleBased Random → **300**

5.6 Ricerca Locale

Chrism1c:
LocalSearch Random → **8**
W1l50n2208:
LocalSearch Random → **8**
m3ttiw:
LocalSearch Random → **8**

5.6.1 Test deterministico sul "circuito PI Greco":

Chrism1c:
LocalSearch Random → **0**

5.5 Tabella riassuntiva dei risultati ottenuti su tutte le intelligenze artificiali

RANDOM						
Artificial Intelligence	# of Runs	Chrism1c	W1l50n2208	m3ttiw	AVG	Ranking
Deep First Search [LV1]	5 Runs	3420	3656	1468	2848,00	5
Deep First Search [LV2]	2 Run	8320	28460	17750	18176,67	2
SDG Q-Learning [P0.5]	20 Runs	8560	3900	2060	4840,00	4
SDG Q-Learning [P1.0]	20 Runs	3580	12140	1480	5733,33	3
Genetic-Beam Training	4 Gen 8 Chromosome 3 Runs	71640	38600	90700	66980,00	1
Local Search Greedy Ascent	5 Runs	8	8	8	8,00	9
Monte Carlo [fullScan]	1 Run	1620	1216	660	1165,33	6
Monte Carlo [randScan]	1 Run	720	1120	1180	1006,67	7
Logic Rule Based	5 Run	72	68	76	72,00	8
PI GREEK (π)						
Artificial Intelligence	# of Runs	Chrism1c	W1l50n2208	m3ttiw	Results	Ranking
Deep First Search [LV1]	1 Run		800		800	7
Deep First Search [LV2]	1 Run		7040		7040	4
SDG Q-Learning [P0.5]	50 Runs	26280			26280	3
SDG Q-Learning [P1.0]	10 Runs	250000			250000	1
Genetic-Beam Training	4 Gen 8 Chromosome 1 Run			45840	45840	2
Local Search Greedy Ascent	1 Run	0			0	9
Monte Carlo [fullScan]	1 Run		2440		2440	6
Monte Carlo [randScan]	1 Run		4340		4340	5
Logic Rule Based	1 Run			300	300	8

6. Conclusioni

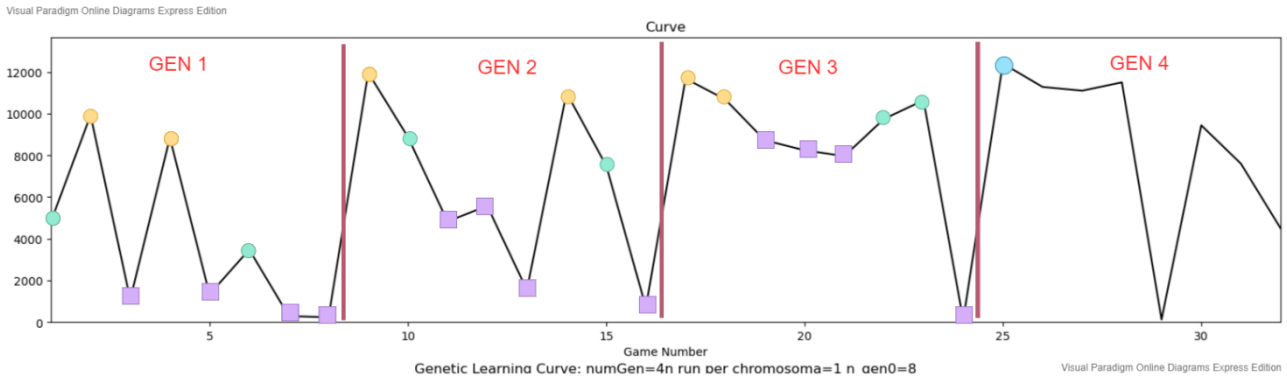
6.1 Considerazioni generali

Con questo esperimento si vuole evidenziare che :

- non tutti gli approcci riguardanti l'intelligenza artificiale sono performanti in qualunque dominio applicativo;
- la tecnica del QLearning e i metodi di ricerca basati su popolazione (Genetico Beam-Search) sono notevolmente più performanti degli altri vista la loro natura di "apprendimento di esperienza passata" a scapito di un tempo di Training considerevolmente superiore;

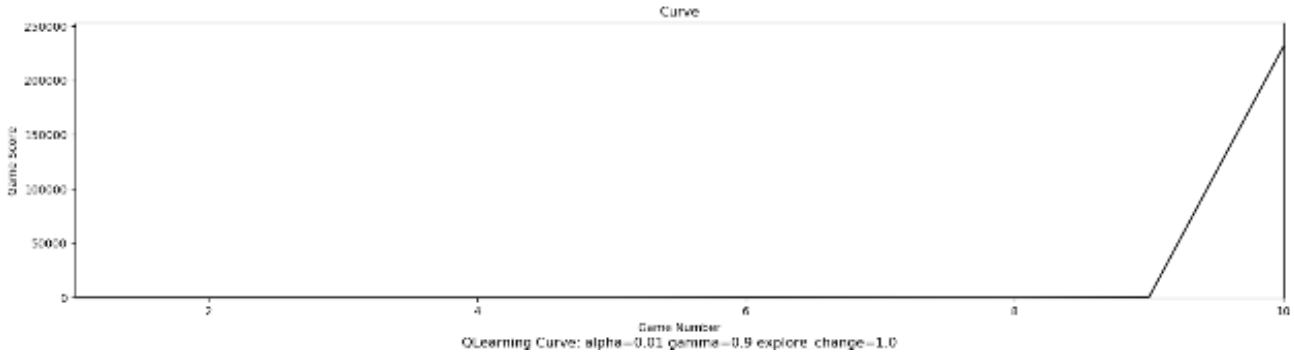
Analisi del training del Genetic – Beam:

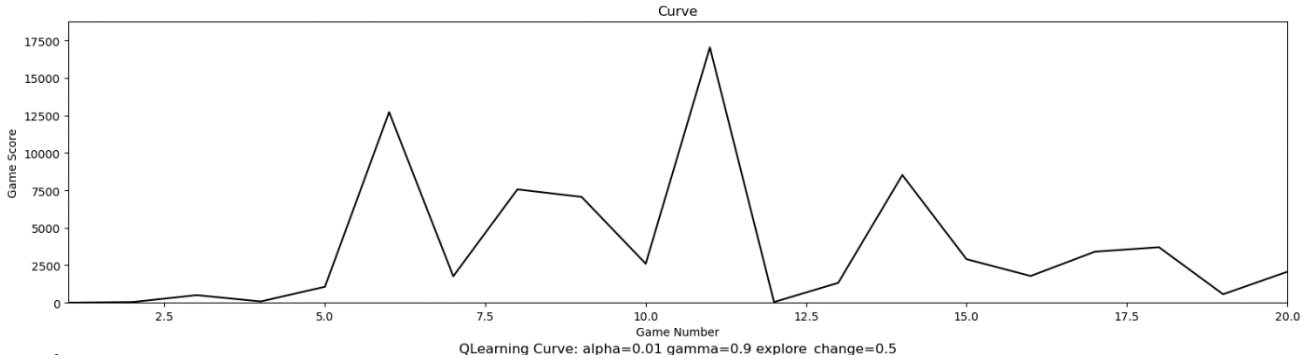
- il pallino blu rappresenta il campione (perfect chromosome)
- i pallini gialli e i pallini verdi sono le coppie di individui che generano due figli per la generazione successiva e sopravvivono alla selezione dei migliori;
- i quadratini rappresentano gli individui che vengono "uccisi"



Analisi del training del SdgQLearning:

L'algoritmo ha prestazioni notevolmente migliori quando l'algoritmo apprende su un percorso deterministico es labirinto (figura 1) piuttosto che su un percorso casuale, cioè diverso per ogni iterazione.





Analisi del Blind Bandit Monte Carlo:

L'algoritmo si dimostrerebbe molto più efficiente in altri domini dove le mosse possono essere retrocesse in caso di errore, mentre nel Tetris una mossa errata determina un effetto domino che porta nella maggior parte dei casi ad una rapida conclusione. Nel nostro caso rimane "attratto" da mosse immaginarie, difficilmente concretizzate.

Analisi del Deep First Search:

Statisticamente risulta essere equilibrato sia dal punto di vista temporale che dei risultati raggiunti. Potrebbe essere migliorato con delle analisi più in profondità, ad esempio implementando l'altezza dell'albero di ricerca, tenendo d'occhio le prestazioni.

Analisi del Local Search Greedy Ascent:

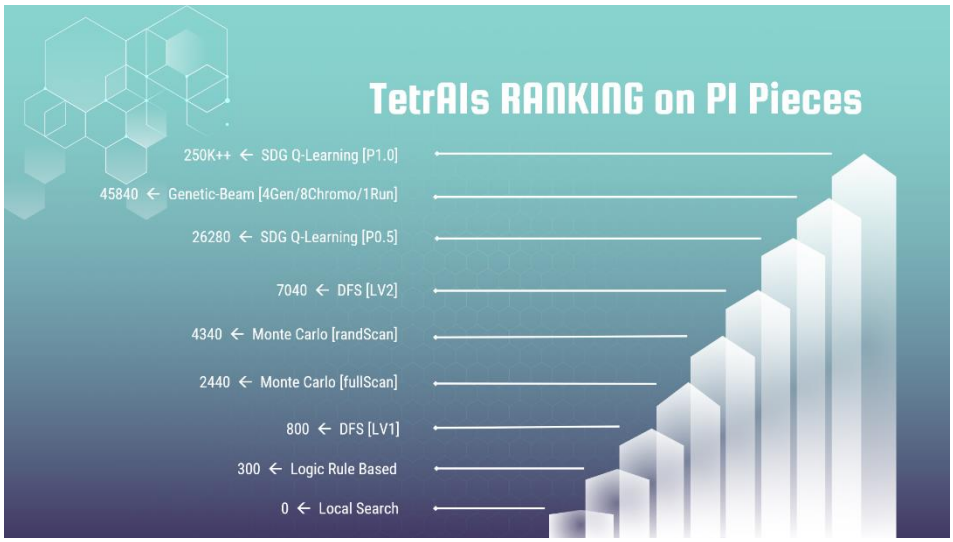
Visti gli scarsi risultati, si potrebbere migliorare l'algoritmo integrando metodi alternativi come il "random restart" per la scansione dell'albero di ricerca al fine di raggiungere l'ottimo globale anziché l'ottimo locale.

Analisi del Logic Rule Based:

Per poter migliorare le prestazioni di questo algoritmo sarebbe utile considerare clausole e regole di diversa natura, per valutare più aspetti del terreno di gioco, piuttosto che valutare la successiva mossa basandosi solo sull'ombra dei tetramini. In questo modo si estenderebbe la conoscenza infieribile dalla base di conoscenza.

6.2 Classifica AI sul circuito Random

6.3 Classifica AI sul circuito PI



7. Bibliografia

Libro: "Artificial Intelligence: Foundations of Computational Agents"

"https://www.geeksforgeeks.org/what-is-reinforcement-learning/"
"Deeplizard Reinforcement Learning - Goal Oriented Intelligence"
"https://www.geeksforgeeks.org/q-learning-in-python/"
"https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e"
"https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31"
"https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/"
"https://www.okpedia.it/ricerca_locale_greedy"
"https://medium.com/cracking-the-data-science-interview/an-introduction-to-optimization-in-intelligent-systems-c1aa408d3ac2"