**TetrAIs**

|  |  |  |
| --- | --- | --- |
| **Christian Miccolis** | **Davide Paduanelli** | **Mattia Patruno** |
| Matricola: 683313 | Matricola: 683127 | Matricola: 676401 |

Indice

**1 Introduzione ……………………………………………………………………. 2**

**2 Funzionalità ……………………………………………………………………….**

**3 Intelligenze Artificiali ……………………………………………………………**

3.1 Deep First Search …………………………………………………………………..

3.2 Stochastic Gradient Descent & Q-Learning…………………………………….

3.3 Genetico …………………………………….…………………………………….….

3.4 Blind Bandit Monte Carlo ………………………………………………………...

3.5 Basato su Regole logiche …………………………………………………………..

3.6 Ricerca Locale …………………………………….…………………………………

**4 Implementazione …………………………….…………………………………**

4.1 Deep First Search …………………………………………………………………..

4.2 Stochastic Gradient Descent & Q-Learning…………………………………….

4.3 Genetico …………………………………….…………………………………….….

4.4 Blind Bandit Monte Carlo ………………………………………………………...

4.5 Basato su Regole logiche …………………………………………………………..

4.6 Ricerca Locale …………………………………….…………………………………

**5 Valutazione e confronto ……………………….…………………………………**

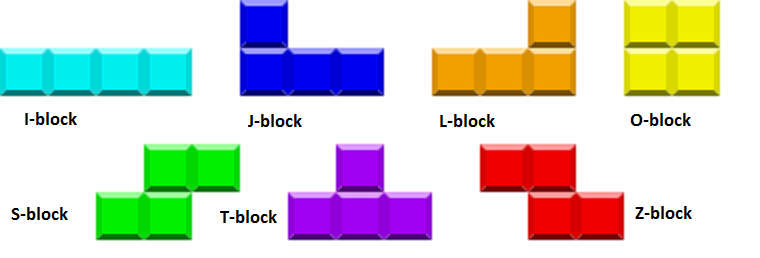
**6 Conclusioni …………………………………….…………………………………**

**7 Bibliografia …………………………………….…………………………………**

**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 spostato 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 alla 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 modo complessivo, 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 fi 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 Tree Plot

2.4 Real Time Console Prints

**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.



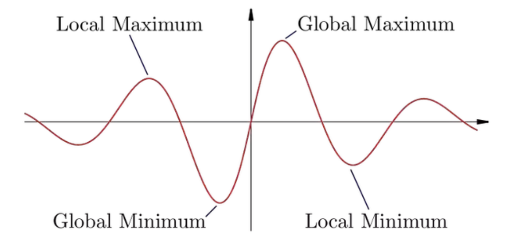
**3.2 Stochastic Gradient Descent** **& Q-Learning**

**3.2.1 Gradient Descent**

Per comprendere come funziona ’”SDG” 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**

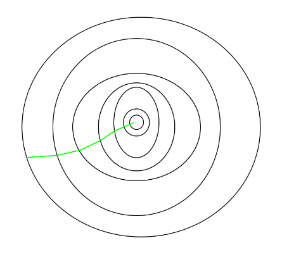
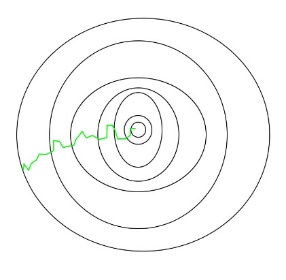
La discesa stocastica del gradiente è un metodo per trovare la configurazione ottimale dei parametri per un algoritmo di apprendimento automatico. Esso esegue in modo iterativo piccole modifiche alla configurazione di una rete di apprendimento automatico per ridurre l'errore della rete.

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 SDG 🡪

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.

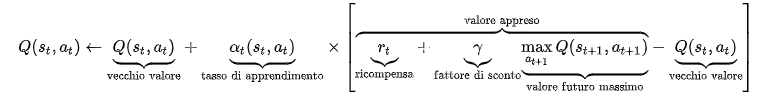
**3.2.4 Q-Learning**

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, osserva 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:



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, anche qui si applica la regola di attualizzazione.

**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 Genetico**

Un algoritmo genetico è un algoritmo euristico 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.

**3.4 Blind Bandit Monte Carlo**

**…**

**3.5 Basato su Regole logiche**

**…**

**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.



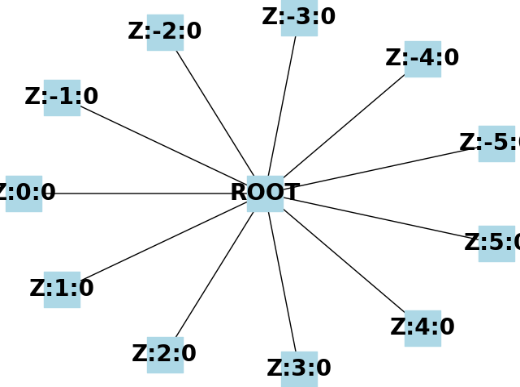
**4. Implementazione**

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)

**3.1 Deep First Search**

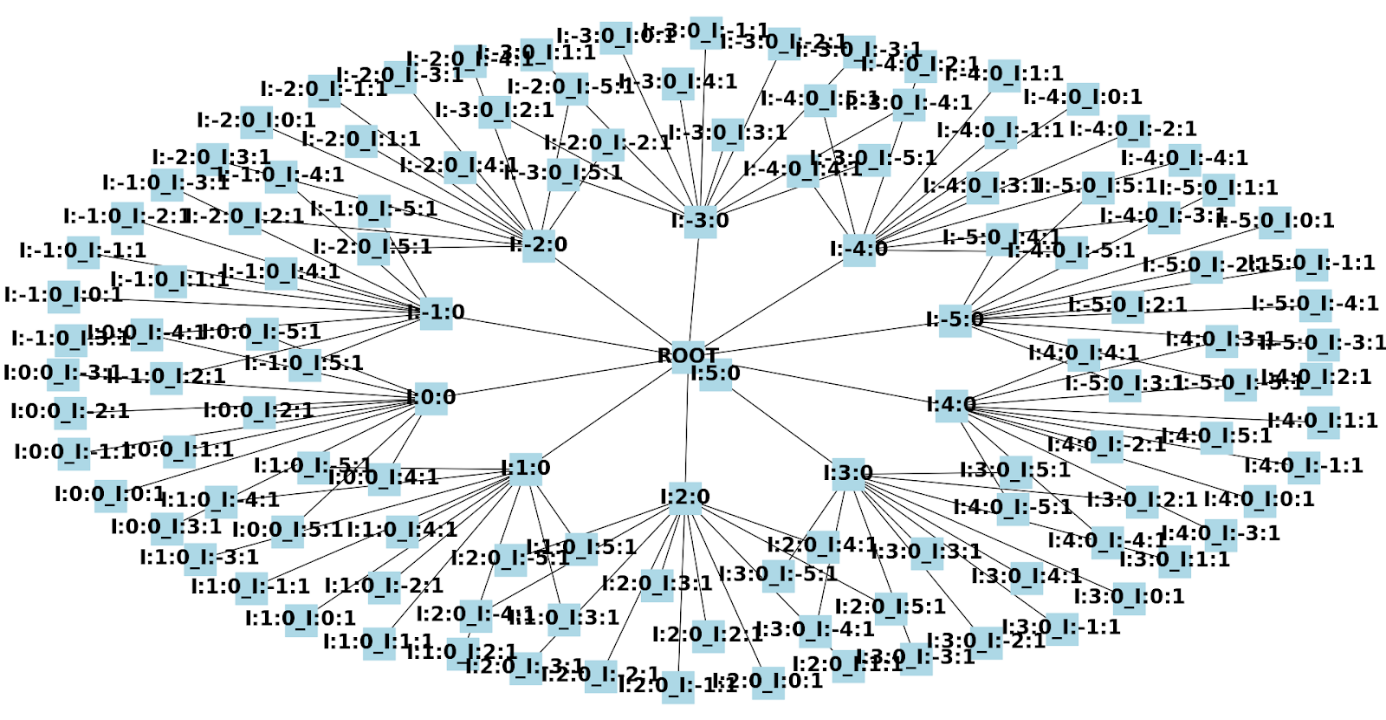
Nel gioco del tetris, non è presente un goal univoco e si cerca, quindi, di ottenere il massimo punteggio possibile. Per raggiungere tale scopo si utilizzano la board corrente, il tetramino corrente e il prossimo tetramino generato (per il DFS\_LV1 si tiene conto solo del tetramino corrente, mentre per 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, poichè 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.



**3.2 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 i pesi 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 [102 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”.

**3.3 Genetico**

Nella nostra implementazione ogni gene è un peso che determina quanto una delle seguenti caratteristiche è presa in considerazione per singola mossa:

* numero di linee completate
* numero di buchi creati
* numero di blocchi presenti nella board
* altezza massima
* deviazione standard delle altezze
* valore assoluto della differenza fra le colonne della board
* massima differenza fra le colonne della board

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”
* ¼ crossing tra i migliori cromosomi della generazione precedente, per creare nuovi cromosomi mescolando i geni che risultano vincenti
* ¼ nuovi cromosomi, per “rimescolare le carte in tavola”

In questo modo si mantiene stabile il numero di individui per generazione.

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.

Dell’ultima generazione salviamo il miglior cromosoma, che sarà possibile testare nella “Perfect Run”.

Vantaggi

Con un numero elevato di generazioni, questa tecnica converge sempre verso il miglior cromosoma possibile.

Dalla curva di apprendimento possiamo notare che i punti di discontinuità corrispondono a nuovi cromosomi o crossing fallimentari.

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).

**3.4 Blind Bandit Monte Carlo**

**…**

**3.5 Basato su Regole logiche**

**…**

**3.6 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**

… sezione “core” dedicata ai grafici e i risultati ottenuti durante i test

**6. Conclusioni**

… resoconto sulle AI

**7. Bibliografia**

… riferimenti e link a risorse utili