**TetrAIs**

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

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 ……………………….…………………………………**

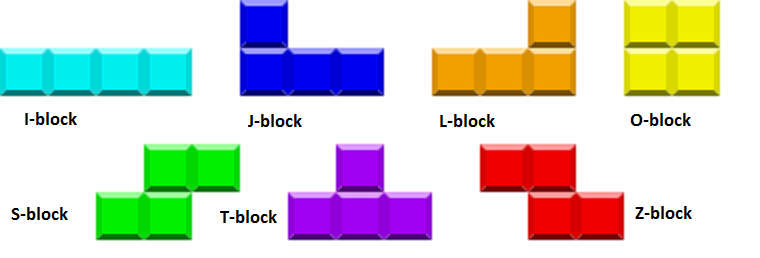
**6 Conclusioni …………………………………….…………………………………**

**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)**

**…**

**3.3 Genetico**

**…**

**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.



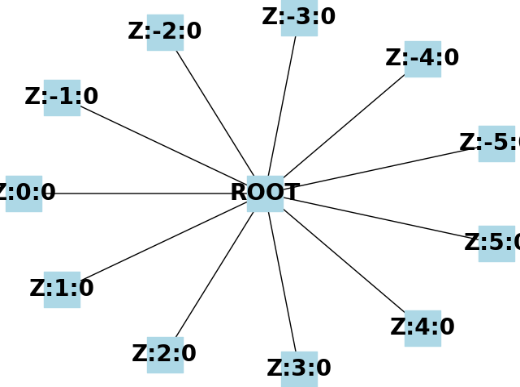
**4. Implementazione**

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**

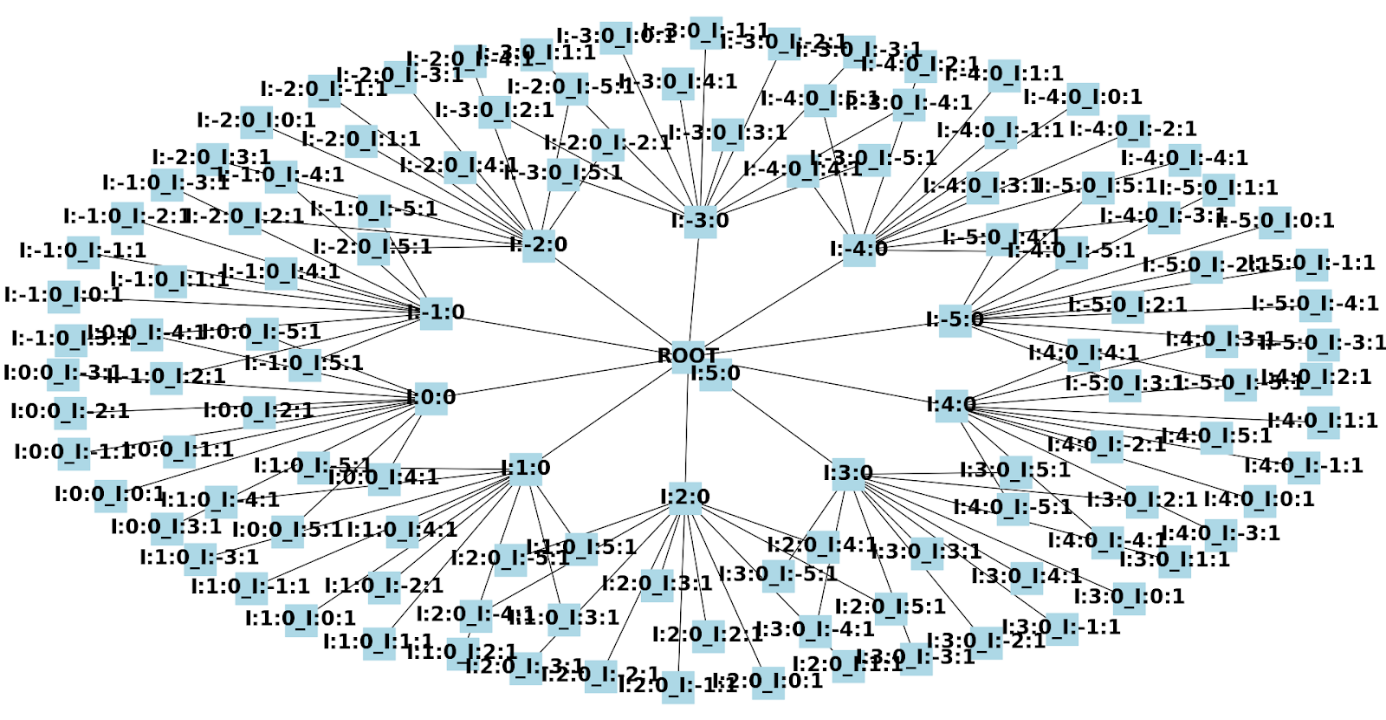
Nel gioco del tetris, non è presente un goal univoco e si cerca, quindi, di ottenere il massimo punteggio possibile. Per raggiungere tale scopo si utilizzano la board corrente, il tetramino corrente e il prossimo tetramino generato (per il DFS\_LV1 si tiene conto solo del tetramino corrente, mentre per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

**…**

**3.3 Genetico**

**…**

**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