BreakoutAI

Progetto Intelligenza Artificiale

di Bernasconi Davide Guido (matricola 1040844)

Descrizione del gioco e delle regole

Questo progetto implementa il gioco Atari Breakout, in cui lo scopo del giocatore è abbattere un muro di mattoni posto nella parte superiore dello schermo, mentre in quella inferiore c'è solamente una piccola barra che può essere mossa a destra e sinistra: con questa bisogna colpire una palla che rimbalza, in modo che distrugga tutti i mattoni che compongono il muro. Nel gioco classico, ci sono tre vite a disposzione : se il giocatore non riesce a colpire la palla con la propria barra, questa esce dalla schermata ed viene persa una vita. Terminate tutte le vite, la partita termina ("Game over"). Nel gioco implementato, non esistono vite : il gioco termina quando il paddle colpisce tutti i blocchi e l'agente vince. Se non riesce a distruggere tutti i blocchi, il gioco termina raggiunti un massimo di tentativi.
Nella versione classica, i mattoni sono disposti su 8 file ed ogni coppia di file è disegnata con un colore diverso: dal basso in alto, giallo, verde, arancio e rosso. Nella versione implementata, le file di mattono sono soltanto due, anche se inizialmente le file di mattoni erano quattro, ma a causa delle simulazioni troppo lunghe sono state ridotte a due. Ogni mattone colpito assegna un punteggio: 1 punto per i mattoni gialli; 3 punti per i mattoni verdi; 5 punti per i mattoni arancio e 7 punti per quelli rossi. Nella versione implementata, il punteggio rimane costante.
La velocità della barra comandata dal giocatore aumenta dopo 4 colpi, poi dopo altri 12 colpi ed infine quando la pallina raggiunge le file di mattoni arancio e rossi. Quando poi sfonda l'ultima fila di mattoni rossi e colpisce il muro superiore la barra dimezza la sua larghezza. Nel gioco implementato, velocita e dimensioni della barra rimangono costanti.
La pallina parte dal paddle al centro e l'orientamento iniziale della pallina (destra o sinistra) viene generato randomicamente.

Descrizione dell'agente

Il mondo di gioco è un mondo completamente deterministico, di conseguenza abbiamo a disposizione tutte le informazioni sul mondo di gioco (velocità della pallina, velocità del paddle, posizione...) perchè l'ambiente è completamente osservabile.
In questo senso ho deciso di applicare le seguenti restrizioni: l'agente non conosce tutte le informazioni sullo stato del gioco (come posizioni e velocità degli oggetti) ma solo una versione "riassuntiva" di queste informazioni, in modo da creare stati discreti finiti.
Queste informazioni, o stati dell'agente, sono le seguenti:

- CENTER : la distanza relativa tra il centro della pallina e il centro del paddle (lungo l'asse x) è minore della metà della dimensione del paddle (in altre parole, la pallina si trova "sopra" il paddle); successivamente è stato impostato un epsilon di sicurezza in modo tale che l'area centrale non corrispondesse alla dimensione del paddle, ma fosse "un po' meno" di questa dimensione.
- UP_LEFT : la pallina si trova a sinistra (distanza relativa dei centri maggiore della metà della dimensione del paddle)rispetto al paddle e la pallina va verso l'alto.

- UP_RIGHT : a pallina si trova a destra (distanza relativa dei centri maggiore della metà della dimensione del paddle) rispetto al paddle e la pallina va verso l'alto.
- DOWN-LEFT : la pallina si trova a sinistra rispetto al paddle e la pallina va verso il basso.
- DOWN_RIGHT : a pallina si trova a destra rispetto al paddle e la pallina va verso il basso.
- GAME_OVER: la pallina tocca il bordo inferiore e si ricomincia daccapo.

Non sono state considerati stati riguardanti la distruzione dei blocchi in quanto non influenti sulla dinamica del gioco: infatti, lo scopo dell'agente è vincere la partita, o meglio dire non permettere il game over e perdere. Perciò saranno ininfluenti i blocchi e le relative reward.

Le azioni sono i tre possibili movimenti del paddle e sono LEFT (L), STOP (S) e RIGHT (R).

Per utilizzare gli algoritmi di reinforcement, bisogna che venga soddifatta la Markov assumption, che afferma che la probabilità di passare allo stato successivo è data solo dallo stato corrente e non dagli stati passati. Di conseguenza, bisogna che gli stati siano indipendenti dal passato. Chiaramente nel caso reale ogni stato presente non è indipendente dal passato: ad esempio, se il gioco si trova nello stato DOWN_RIGHT di un determinato frame del gioco, è ovviamente molto probabile che nel frame precedente lo stato sia ancora DOWN_RIGHT (infatti lo spostamento percorso dalla pallina o dal paddle tra un frame e l'altro è molto piccolo). Per questo bisogna teorizzare che gli stati di gioco cambiano in modo totalmente casuale, o perlomeno, visto che non rappresenta la situazione reale, come può essere osservato e percepito dall'agente. Ovviamente questo rappresenta un limite per l'agente, un'approssimazione del comportamento dell'ambiente.

Per quanto riguarda le reward si usano questi tipi:

- L'idea iniziale consisteva nel premiare l'agente dando come reward positivo il punteggio del blocco distrutto durante quel frame. Il problema consiste nel fatto che c'è un grande divario temporale tra l'esecuzione di un'azione che vale la pena premiare (la palla che rimbalza sul paddle) e il momento in cui la palla rompe un mattone. Per questo motivo, ho deciso di premiare l'agente quando fa rimbalzare con successo la palla sul paddle e non quando rompe un mattone. (idea presa dal paper citato alla fine della relazione).
- Per velocizzare l'apprendimento, viene data una piccola reward negativa proporzionale alla distanza tra la posizione x del centro del paddle e la posizione x della palla.
- Quando la palla va verso il basso, la reward negativa appena citata raddoppia rispetto a quando va verso l'alto o si trova al centro del paddle.

Algoritmo SARSA

Ecco di seguito lo pseudocodice dell'algorimo:

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state},\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Ogni episode è rappresentato da una nuova sessione di gioco, ogni time step è un frame. L'algoritmo esegue la policy ε-greedy basandosi sulla action-value function Q(s, a). La scelta dell'azione da intraprendere, basandosi su questa policy, risulta

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

Come si evince il parametro che gestisce la policy è ε, che rappresenta il rapporto tra esploration e exploitation dell'agente. Maggiore è ε, maggiore è la probabilità di intraprendere azioni casuali e quindi favorisce l'esplorazione dell'ambiente, viceversa minore è ε, maggiore la probabilità di prendere l'azione che massimizza l'action-value function Q(s, a) e quindi favorire lo sfruttamento dell'ambiente.

Esperimenti

Gli esperimenti si basano sulla variazione del parametro ε e quindi del rapporto exploitation/exploration. Nel file statistics.exe vengono raccolti i dati dell'esperimento.
Alcune considerazioni:

- In generale, dopo un numero considerevole di tentativi, non è possibile distinguere se l'agente sta apprendendo molto lentamente, e quindi vincerà dopo molti tentativi, o non sta apprendendo affatto, ovvero non vinverà mai e il gioco continuerà all'infinito. Di conseguenza, è stata impostata un soglia massima di 30 tentativi in modo tale che, una volta raggiunta questa soglia, si può considerare l'agente incapace di raggiungere l'obiettivo e quindi perde il gioco.

I dati sono presentati nella seguente forma:

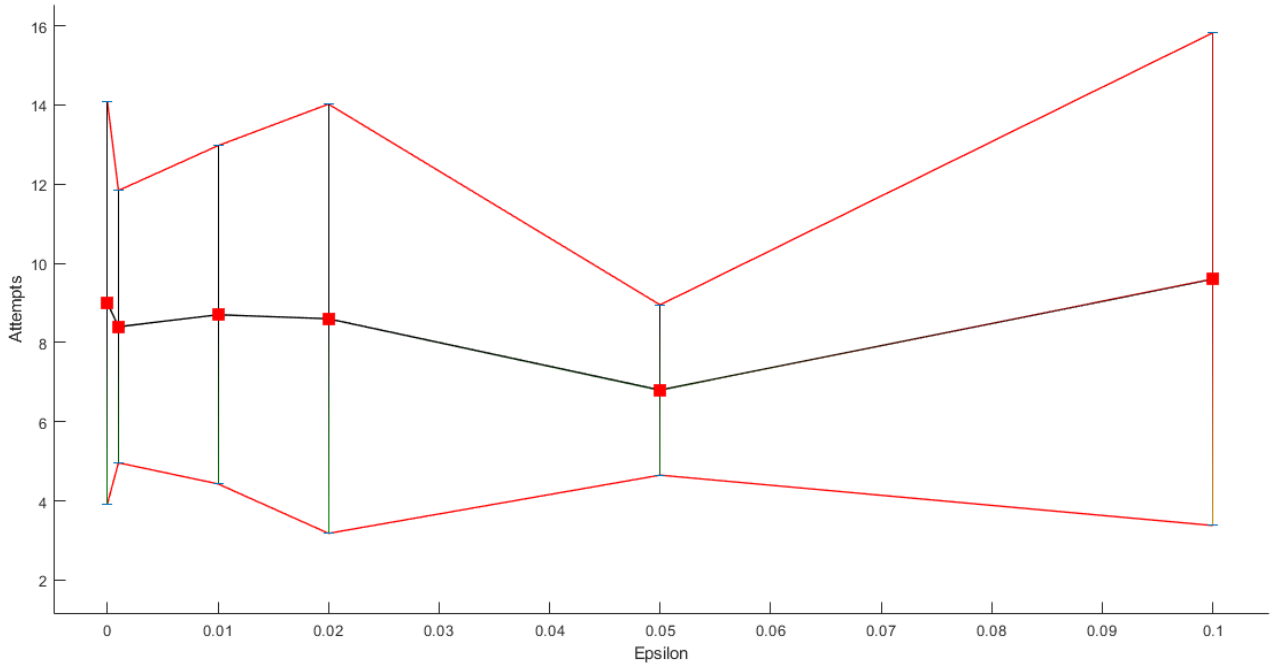
- epsilon: valore di ε usato in quell'esperimento
- number of blocks : rappresenta i numeri dei blocchi distrutti durante una sessione di gioco.
- attempts: numero di sessioni di gioco che l'agente ha giocato prima di riuscire a vincere il gioco.

Come si evince dal dati il number of blocks non è un buon modo per valutare la policy, in quanto i dati sono molto simili tra loro e variano poco tra ogni esperimento (a parte certi outliers non significativi -dovuti ad errori, ad esempio, di compenetrazione del paddle e della palla che porta a farla cadere-) . Un buono metodo per verificare la bontà di una

policy è attempts, in particolare prenderemo la media delle misurazioni, in quanto l'incertezza è molto alta.
Sono stati esclusi a priori il punteggio della sessione (che coincide con i blocchi distrutti, quindi vale lo stesso ragionamento di number of blocks), e il numero di time step, che non dipende esclusivamente dallabilità dell'agente, ma ha dipende in modo "casuale" per chè l'agente non ha il controllo sulla direzione della pallina.
Ho fatto dieci eperimenti per ogni valore di ε (che si possono trovare su statistics.txt), prendendo in considerazione media campionaria e deviazione standard:

- ε = 0.2 (1 / 5 azioni è casuale)
  μ[attempts] = INFINITY
  σ[attempts] = Nan
- ε = 0.1 (1 / 10 azioni è casuale)
  μ[attempts] = 9.6
  σ[attempts] = 6.221825384170719
- ε = 0.05 (1 / 20 azioni è casuale)
  μ[attempts] = 6.8
  σ[attempts] = 2.149935399546280
- ε = 0.02 (1 / 50 azioni è casuale)
  μ[attempts] = 8.6
  σ[attempts] = 5.420127099780759
- ε = 0.01 (1 / 100 azioni è casuale)
  μ[attempts] = 8.7
  σ[attempts] = 4.270050741306634
- ε = 0.001 (1 / 1000 azioni è casuale)
  μ[attempts] = 8.4
  σ[attempts] = 3.438345855527367
- ε = 0 (nessuna azione casuale – caso particolare–)
  μ[attempts] = 9
  σ[attempts] = 5.077182070575939

Ecco il grafico che mostra il risultatom, mostrando in nero la media e in rosso l'intervallo di confidenza (media +/- standard deviation)



4.Considerazioni finali e interpretazioni

Il caso $\varepsilon = 0$, ε-greedy diventa un metodo greedy, in quanto prende sempre l'azione che massimizza Q(s, a). Di conseguenza l'algoritmo risulta simile a Q-Learning, ma a differenza di quest'ultimo rimane un metodo on-policy che segue la policy greedy.

Il caso $\varepsilon = 0.2$ può essere considerato un caso limite, un valore massimo per ε, in quanto l'agente non riesce ad apprendere come risolvere il gioco (o perlomeno riesce ma in tempi non brevi).

Dal grafico degli esperimenti si evince che il valore che in media porta a una policy ottimale vale $\varepsilon = 0.05$, in quanto ha la media dei tentativi inferiore rispetto agli altri, oltre ad avere un incertezza (deviazione standard) inferiore.

Per gli altri valori, si ha la tendenza ad aumentare il valore del numero medio dei tentativi al diminuire dei valori di ε inferiori a 0.05 e all'aumentare dei valori di ε superiori a 0.05 (per $\varepsilon \geq 0.2$ la media tende ad infinito perchè l'agente non vince il gioco).

Una possibile interpretazione di questo comportamento consiste nel fatto che, in quanto ε rappresenta la percentuale media delle azioni scelte casualmente nel gruppo di quelle azioni che **non** massimizzano l'action-value function Q(s, a) su tutte le azioni intraprese dall'agente (le altre azioni selezionate sono quelle che massimizzano l'action value Q(s, a)), queste azioni casuali possano influenzare l'andamento dell'agente. Per $\varepsilon \to 0$, queste azioni casuali vengono prese più raramente e quindi influenzano meno il comportamento dell'agente, mentre le azioni che lo influenzano maggiormente sono le azioni che massimizzano Q(s,a); questo comportamento tende a non favorire l'esplorazione di nuove alternative che potrebbero potenzialmente portare a soluzioni migliori per vincere il gioco (nel nostro caso, vale per $\varepsilon < 0.05$). Per $\varepsilon \to 1$ la frequenza delle azioni casuali aumenta e e di conseguenza influenzano in modo cnsiderevle il comportamento edll'agente, tendendo a farlo sempre più casuale e imprevedibile, andando a compromettere l'apprendimento dell'agente man mano che ci si avvicina a $\varepsilon = 1$ (infatti per $\varepsilon \geq 0.2$, il numero di azioni casuali (maggiori di 1 su 4)

è tale per cui l'agente non riesce più a vincere il gioco).
Il trade-off per bilanciare questi due aspetti vale, secondo i risultati dell'esperimento, proprio per $\varepsilon = 0.05$.
Ovviamente questi risultati si basano sugli esperimenti da me condotti, con i dovuti limiti nel numero di simulazioni effettuate. Per valutare la bontà di una policy in un esperimento reale, è necessario aumentare il numero di simulazioni in modo da ridurre l'incertezza e avere una idea precisa sulla qualità della policy.

Riferimenti bibliografici

- https://it.wikipedia.org/wiki/Breakout_(videogioco)
- Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto
- Reinforcement Learning for Atari Breakout, Stanford University CS 221 Project Paper, Vincent-Pierre Berges, Priyanka Rao, Reid Pryzant