

Progetto Sistemi Intelligenti Avanzati

Andrea Delia

A.A. 2019-20

Indice

1	Introduzione	1
1.1	Problema	1
1.2	Azioni	1
1.3	Stati	1
1.4	Reward	3
2	Algoritmo	3
2.1	Parametri	3
2.2	Q-Table	4
2.3	Problema della C	4
	2.3.1 Traccia di Eleggibilità	5
3	Interfaccia Grafica	6
3.1	Schermata principale	6
3.2	Schermata secondaria	7
4	Conclusioni	8
4.1	Risultati	8
4.2	Risultati non raggiunti	10
	4.2.1 Possibile Soluzione	10

1 Introduzione

QSnake è un software Java che consente di analizzare il comportamento di un sistema con rinforzo applicato al celebre gioco Snake. Il gioco consiste nel muovere il serpente all'interno di una griglia, cercando di raggiungere i frutti generati in posizioni casuali della mappa senza collidere con i bordi o con la coda. Ogni qual volta il serpente raggiunge un frutto, questo incrementa la sua dimensione di uno e un nuovo frutto viene generato in un'altra posizione (ovviamente diversa da quella precedente). Nel caso in cui si verifichi una collisione, la partita (e l'episodio) si interrompe ed il serpente riparte dallo stato di partenza con lunghezza pari a tre.

1.1 Problema

Ovviamente, dato che il punto di interesse non è la semplice implementazione del gioco Snake dove il serpente è controllabile da un utente, bensì la realizzazione di un agente in grado di apprendere come muoversi totalizzando il punteggio migliore possibile, è necessario introdurre i concetti di **azione**, **stato** e **ricompensa**.

1.2 Azioni

Le azioni costituiscono ciò che l'agente può fare per affrontare il problema. Nel nostro caso queste coincidono con quelle che potrebbe fare un utente umano che gioca normalmente a snake: le azioni possibili, infatti, sono solo quattro e costituiscono il movimento in uno dei quattro punti cardinali. Ovviamente, in qualsiasi istante di esecuzione, l'agente non potrà mai muoversi nella direzione da cui proviene e, di conseguenza, ad ogni passo potrà scegliere tra tre azioni disponibili.

1.3 Stati

Le azioni dell'agente modificano lo **stato globale** del sistema. Infatti, considerando lo stato globale come configurazione possibile della griglia, le possibilità diventano moltissime e crescono esponenzialmente con l'aumentare delle dimensioni della mappa.

Per evitare il crescere esponenziale della complessità, è stato implementato un sistema differente: i possibili **stati** del sistema sono infatti generati dalla composizione tra ciò che è presente nelle celle adiacenti alla testa del serpente e dalla posizione del frutto rispetto alla testa del serpente. Ogni stato è quindi rappresentabile da un vettore di 8 interi nella forma:

$$[up, right, down, left, fruitUp, fruitRight, fruitDown, fruitLeft]$$

I primi quattro interi rappresentano ciò che l'agente vede rispettivamente sopra di lui, alla sua destra, sotto di lui ed alla sua sinistra. Quando il valore dell'intero corrispondente è pari a 0, significa che il percorso è libero in quella direzione; se è pari ad 1, significa che è occupato da qualcosa (bordo mappa o coda del serpente); infine -1 rappresenta la provenienza del serpente.

Gli ultimi quattro interi, invece, rappresentano la posizione globale del frutto rispetto alla testa del serpente. Di conseguenza, questi valori assumono valore 1 nel caso in cui il frutto si trovi in quella direzione e 0 altrimenti.

Esempio Ad esempio, nel caso in cui l'agente provenga dal basso e si sia spostato nell'angolo in alto a sinistra della mappa, la configurazione della prima metà dello stato sarà

$$[1, 1, -1, 0, *, *, *, *]$$

Supponendo che il frutto si trovi nella seconda riga e nel lato destro della mappa, esso si trova a sud-est rispetto alla testa del serpente (che si trova nell'angolo in alto a sinistra). Per questo motivo, gli ultimi quattro interi avranno valore

$$[*, *, *, *, 0, 0, 1, 1]$$

Complessivamente otteniamo lo stato:

$$[1, 1, -1, 0, 0, 0, 1, 1]$$

Il vantaggio della rappresentazione degli stati in questo modo è principalmente quello di non legarsi alla posizione del serpente nella mappa, ma di concentrarsi solo su ciò che lo circonda. In questo caso, avere un muro alla propria destra mentre si è nella parte alta della mappa o nella parte bassa della mappa è indifferente e, di conseguenza, è trascurabile.

1.4 Reward

Per quanto riguarda il sistema di ricompense, l'idea iniziale di base era quella di fornire un reward positivo nel caso in cui l'agente si fosse spostato in una cella libera ed uno negativo nel caso in cui abbia deciso di spostarsi in una cella occupata. Ovviamente questo non gli permetteva di sfruttare la parte finale dello stato e, di conseguenza, di andare in direzione del frutto. Per questo motivo sono stati modificati i reward ed è stato deciso di attribuire un reward positivo nel caso in cui l'agente vada in direzione del frutto, poco negativo nel caso in cui vada in una cella libera non in direzione del frutto ed uno molto negativo nel caso di una collisione.

In particolare, abbiamo che:

- **Collisione:** -5
- **Spostamento verso frutto:** +2
- **Spostamento non verso frutto:** -0.5

2 Algoritmo

L'algoritmo che viene utilizzato per implementare il sistema con rinforzo è il **Q-Learning**, in particolare con l'utilizzo della traccia di eleggibilità. Il funzionamento di questo algoritmo è regolato mediante l'utilizzo di alcuni **parametri** e di una tabella, detta **Q-Table**, che associa ad ogni coppia (stato, azione) un valore (reward cumulativo).

2.1 Parametri

La scelta dei parametri è stata affrontata mediante l'analisi di una serie di tentativi, variandoli di volta in volta. Procedendo in questo modo, sono stati individuati i seguenti valori:

- **Tasso di apprendimento α :** 0.8
- **Fattore di sconto γ :** 0.2

Attraverso l'utilizzo di questi parametri si ottiene un agente lungimirante, che quindi non si limita ad essere soddisfatto di una buona ricompensa istantanea, e che si interessa ciò che accade nel breve periodo.

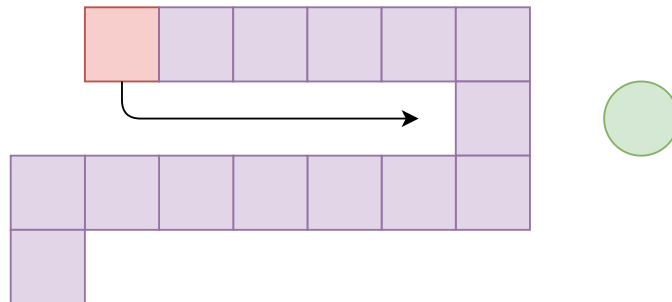
2.2 Q-Table

La Q-Table è una tabella che associa ad ogni coppia (stato, azione) un valore che consente di capire quanto quell'azione sia buona da applicare mentre si è in quello stato. Questi valori vengono aggiornati nel tempo e costituiscono l'apprendimento vero e proprio dell'agente. Inoltre, l'algoritmo è accompagnato dalla traccia di eleggibilità che consente di guardare nel passato, aggiornando anche i valori delle coppie percorse nel breve periodo.

In particolare, per quanto riguarda l'implementazione di QSnake, la Q-Table è rappresentata da una HashMap che viene aggiornata ad ogni passo. Inoltre, quando un nuovo stato viene scoperto, la Q-Table aumenta di dimensioni e le righe aggiunte (corrispondenti al nuovo stato) vengono inizializzate con i valori dei reward istantanei associati; in questo modo, quando l'agente scopre un nuovo stato, non ne viene sorpreso ed evita di agire in modo poco intelligente (anche se non sempre con successo).

2.3 Problema della C

Uno dei problemi relativi al software è quello della mancata gestione di ciò che l'agente stesso non può vedere. Il suo campo visivo, infatti, è ristretto alle sole quattro celle adiacenti alla sua testa. Per questo motivo, l'agente non riesce a rendersi conto di azioni sbagliate che non coinvolgono la collisione diretta contro un ostacolo.

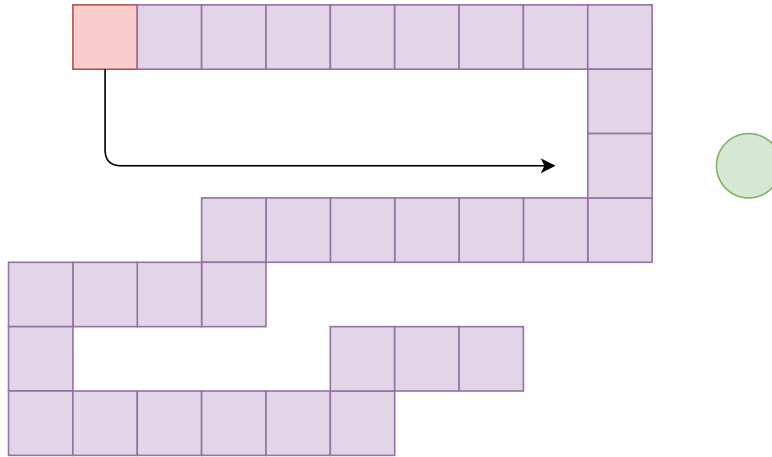


Considerando una situazione simile a quella mostrata in figura, l'agente, vedendo il frutto (cerchio verde) in quella posizione, non è in grado di rendersi conto che, arrivato al termine della freccia, non potrà fare niente per sopravvivere. Definiamo questo problema come “Problema della C”, dato che la forma della posizione del serpente in questo caso è simile ad una C.

2.3.1 Traccia di Eleggibilità

Per risolvere il problema della C, è stato implementato un sistema di traccia di eleggibilità con parametro $\lambda = 0.8$. In questo modo, l'agente potrà considerare l'azione di “entrare nella C” come svantaggiosa poiché porta sempre ad una collisione. Ovviamente, questo sistema ha un riscontro positivo solamente quando, a seguito dell'esecuzione di più episodi, la lunghezza del serpente diventa di dimensioni elevate.

Tuttavia, il problema della C non è risolto completamente, ancora una volta a causa dello scarso campo visivo dell'agente.



In questo caso l'agente sta cadendo nuovamente nel problema della C, senza possibilità di rendersene conto. Il sistema della traccia riesce quindi a garantire un miglioramento per delle “C strette”, ovvero quelle che portano alla sconfitta con più frequenza, ma non riesce a vedere le “C larghe”, le quali però portano alla sconfitta solo in caso di lunghezze del serpente molto elevate (maggiori a 50).

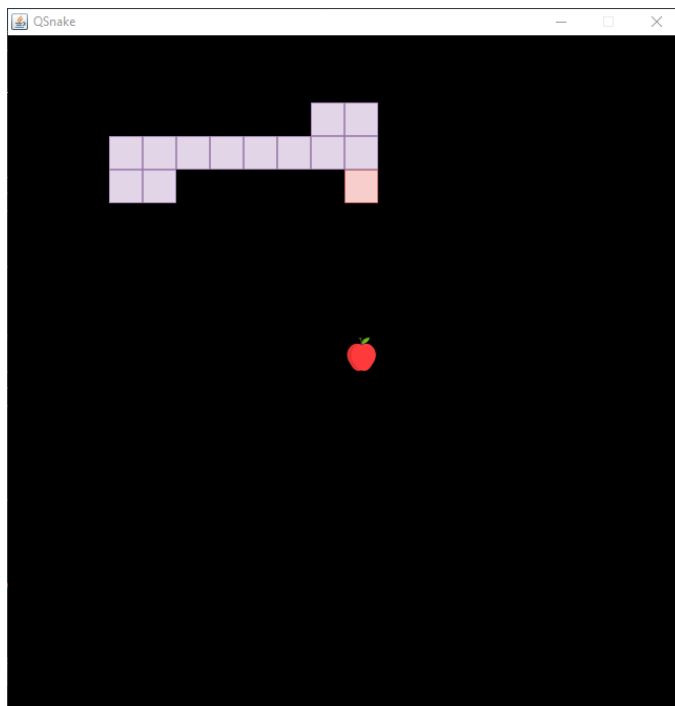
Il risultato ottenuto dalla traccia è quindi soddisfacente nel caso di lunghezze medie.

3 Interfaccia Grafica

Il software si presenta con due finestre: quella principale, in cui viene eseguito il gioco, ed una secondaria, in cui vengono visualizzati alcuni dettagli dell'esecuzione.

3.1 Schermata principale

La schermata principale si presenta come un rettangolo nero, ovvero la mappa sul quale scorre un serpente composto da semplici quadrati. La sua testa è rappresentata da un quadrato rosso, come quello mostrato negli esempi precedenti, mentre la coda (ovvero il resto del suo corpo), da quadratini violacei. Il frutto è semplicemente rappresentato da una mela rossa. Ogni elemento dell'interfaccia è composto da icone di dimensione 32×32 e la griglia stessa è di dimensione 640×640 . Di conseguenza, la mappa percorribile dal serpente è costituita da una griglia di 20×20 quadretti.



Chiudendo l'interfaccia principale il processo termina la sua esecuzione.

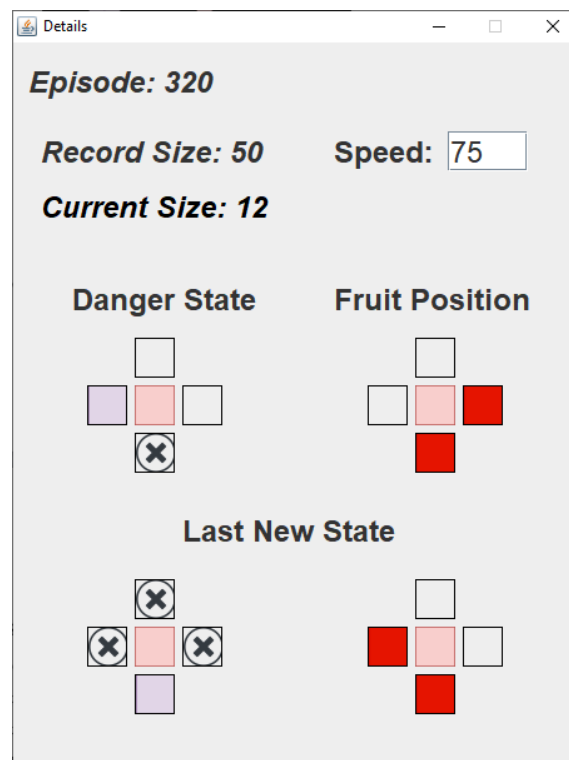
3.2 Schermata secondaria

L'interfaccia secondaria è quella più interessante dal punto di vista dell'algoritmo. Questo perché essa contiene ciò che l'agente, ovvero il serpente, **vede** in ogni istante. Il Danger State ed il Fruit Position, considerati insieme, consentono di avere una rappresentazione grafica dello stato analizzato precedentemente in forma numerica.

Oltre alla rappresentazione dello stato corrente, è presente la rappresentazione dell'ultimo stato scoperto, ovvero l'ultimo stato aggiunto alla Q-Table (considerando tutte le possibili azioni eseguibili in quello stato).

In entrambi i casi precedenti, la croce indica il valore 1 nella relativa posizione, ovvero la presenza di un oggetto che genera collisione (bordo o coda).

Oltre agli elementi più interessanti, sono anche presenti i tre contatori che rappresentano la quantità di episodi eseguiti, il record di lunghezza e la lunghezza corrente.



Per quanto riguarda il campo Speed, è stato implementato tramite una JTextField ed il valore al suo interno costituisce la quantità di millisecondi da attendere tra un passo dell'agente e l'altro.

Impostando questo valore a 0 (valore minimo), si ottiene un'esecuzione veloce che può mostrare rapidamente i risultati dell'apprendimento; impostando invece il valore a 1000 (valore massimo), si ottiene un'esecuzione lenta che aiuta la visione delle informazioni presenti nella finestra secondaria. Il valore di default per questo parametro è 75, ovvero una via di mezzo tra un'esecuzione veloce ma che faccia capire qualcosa di ciò che sta accadendo all'agente.

Ovviamente, inserendo valori negativi oppure superiori a 1000, il software gestirà il parametro errato sostituendolo con uno corretto.

Selezionando un valore di attesa molto alto si può riscontrare un leggero lag, dato che queste finestre non sono implementate tramite Thread e, di conseguenza, l'attesa di una causa un'attesa anche dell'altra.

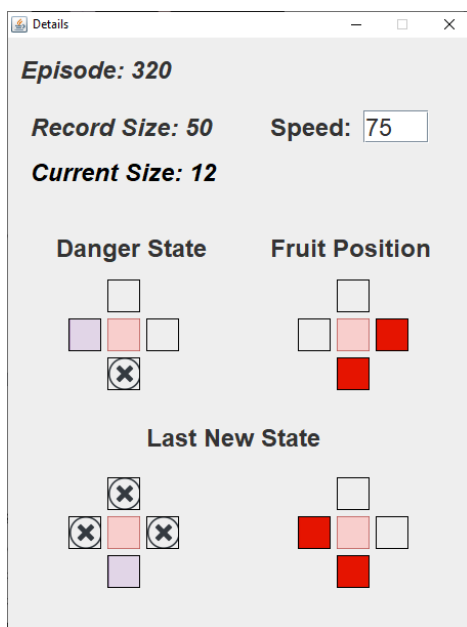
Chiudendo l'interfaccia principale il processo non termina la sua esecuzione, ma non sarà più possibile analizzare le statistiche in essa presenti.

4 Conclusioni

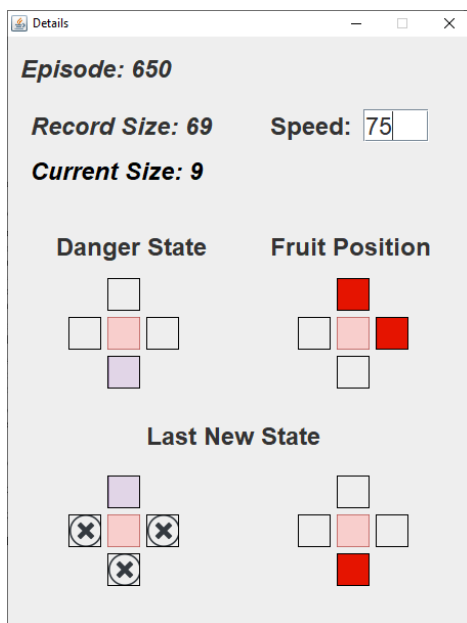
In conclusione, i risultati ottenuti sono soddisfacenti ma non perfetti. In termini di complessità, il software riesce a *risparmiare* grazie all'utilizzo degli stati descritto precedentemente e all'utilizzo di un campo visivo di raggio 1. Tuttavia, dato il campo visivo ristretto, si verifica il problema della C, che genera dei risultati un po' peggiori per lunghezze elevate.

4.1 Risultati

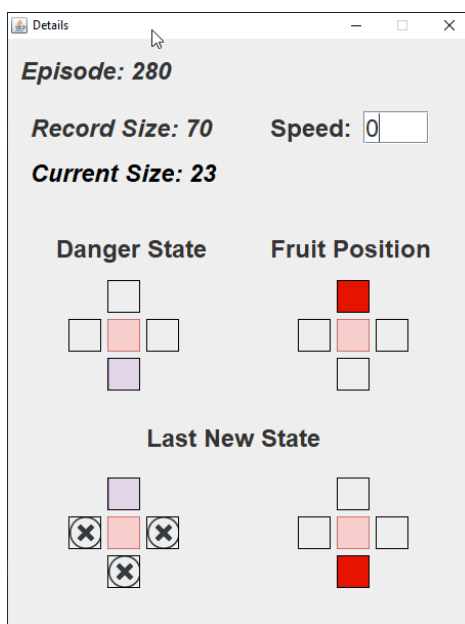
In media, dopo circa 250 episodi si raggiunge un risultato di circa 50 sulla lunghezza del serpente.



Andando avanti con l'esecuzione, raggiungendo più di 500 episodi, è comune vedere lunghezze comprese tra 60 e 70.



A volte l'esecuzione fornisce dei risultati migliori alla media; il risultato migliore riscontrato è una lunghezza di 70 in 280 episodi; tuttavia, è probabile che questo sia dovuto alla casualità dell'esecuzione che, probabilmente, in quella situazione non ha fatto verificare frequenti problemi della C larga.



4.2 Risultati non raggiunti

Il risultato che ovviamente non è stato raggiunto è quello del completamento perfetto della griglia (e quindi del gioco). Questo obiettivo è irraggiungibile per via dei problemi legati al campo visivo (che dovrebbe quindi essere globale, non semplicemente di raggio 2 o superiore).

4.2.1 Possibile Soluzione

Una possibile soluzione parziale potrebbe essere sicuramente quella dell'incremento del campo visivo, aumentando la dimensione di ogni stato da 8 interi a 12 o 16 interi; questo però causerebbe un **notevole** incremento di complessità del sistema. Un'altra scelta potrebbe essere quella di implementare il sistema con algoritmi diversi (specialmente scegliendo di ampliare il campo visivo) come, ad esempio, tramite una rete neurale.