Lawn Mowing

Rocco Iuliano, Simone Della Porta

Intelligenza Artificiale A.A. 2023/2024

Contents

1 Introduzione

Il progetto prevede la realizzazione di un agente di Reinforcement Learning in grado di giocare al gioco di Lawn Mowing. In particolare, l'agente dovrà apprendere le corrette azioni da intraprendere all'interno del gioco in modo da accumulare quanti più punti possibili. L'ambiente di gioco è rappresentato da un'interfaccia 2D dotata di un prato con erba a diversa altezze, ostacoli quali rocce ed alberi e dall'agente stesso rappresentato da un tagliaerba. Inoltre l'ambiente è soggetto a variazioni meteorologiche che portano alla crescita dinamica dell'erba. Per la realizzazione dell'agente è previsto l'utilizzo degli algoritmi Q-learnign e SARSA. L'obiettivo del progetto è testare le diverse configurazioni dell'ambiente e dell'agente in modo tale da identificare la configurazione ottimale e i limiti dell'agente.

2 Implementazione

2.1 Interfaccia 2D

Per l'implementazione dell'interfaccia 2D del progetto è stata utilizzata la libreria PyGame. La finestra di gioco è rappresentata da una griglia 8x8 all'interno della quale è presente erba a diverse altezze, ostacoli e l'agente. Per l'erba sono state definite quattro altezze: rasa, bassa, media e alta. Gli ostacoli previsti all'interno della griglia di gioco sono alberi e rocce mentre l'agente è rappresentato da un omino che spinge un tagliaerba. Per quanto riguarda le condizioni meteorologiche sono state definite tre condizioni meteo: soleggiato, nuvoloso e piovoso. Le condizioni meteo vengono aggiornate con cadenza regolare e per esplicitare tali le condizioni meteo è stato inserito all'interno della finestra di gioco una sezione che raffigura il cielo con tre possibili figure che rappresentano rispettivamente le tre condizioni climatiche.

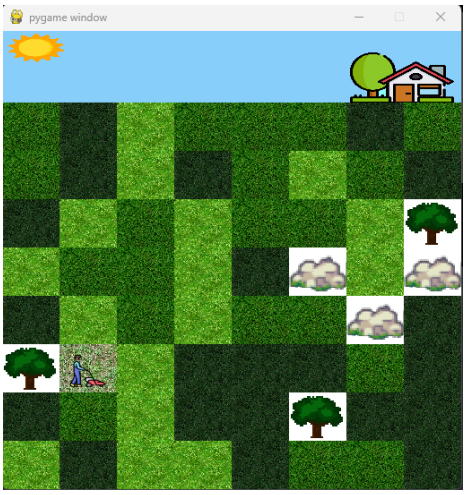

Figure 1: Interfaccia di gioco

2.2 Ambiente

Per l'implementazione dell'ambiente è stata utilizzata la libreria Gymnasium. Dato che la libreria non fornisce di default questo tipo di gioco, è stata creato una classe Python che estende la classe *Env* di Gymnasium in modo da realizzare un ambiente ad hoc. Questa classe si occupa della gestione delle condizioni climatiche, della crescita dinamica dell'erba, di fornire le reward all'agente e di aggiornare lo stato dell'ambiente sulla base dell'azione effettuata dall'agente. In particolare sono stati realizzati due ambienti Gymnasium per poter confrontare le prestazioni del modello con due configurazioni diverse che verranno dettagliate in seguito.
Come precedentemente riportato, l'ambiente prevede la crescita dinamica dell'erba

che dipende dalla simulazione di variabili ambientali che influenzano la sua velocità di crescita. Il valore assunto da quest'ultima è probabilistico e verrà aggiornato nel tempo dopo un certo numero di azioni (10). La crescita dell'erba avviene nel seguente modo:

- Se il meteo è **soleggiato** l'erba crescerà 3 volte prima che il meteo cambi;
- Se il meteo è **nuvoloso** l'erba crescerà 1 volta prima che il meteo cambi;
- Se il meteo è **piovoso** l'erba crescerà 2 volte prima che il meteo cambi.

2.2.1 Ambiente 1

Nel primo ambiente, l'agente può effettuare una delle seguenti azioni:

- spostarsi in avanti;
- spostarsi indietro;
- spostarsi a destra;
- spostarsi a sinistra;
- tagliare l'erba con modalità debole;
- tagliare l'erba con modalità media;
- tagliare l'erba con modalità forte.

Sono state definite tre configurazioni di reward - penalty e sono riportate nelle Tabelle 1 - 3.
Environment 1-1

Table 1: tabella reward - penalty environment 1-1

Azione agente	Elemento dell'ambiente	Punteggio
avanti/indietro/destra/sinistra	erba rasa	0
avanti/indietro/destra/sinistra	erba bassa	0
avanti/indietro/destra/sinistra	erba media	0
avanti/indietro/destra/sinistra	erba alta	0
taglio debole/medio/forte	erba rasa	-4
taglio debole	erba bassa	+3
taglio medio/forte	erba bassa	-1
taglio medio	erba media	+6
taglio debole/forte	erba media	-2
taglio forte	erba alta	+9
taglio debole/medio	erba alta	-3
avanti/indietro/destra/sinistra	roccia	0
avanti/indietro/destra/sinistra	albero	0

taglio debole/medio/forte	roccia	-5
taglio debole/medio/forte	albero	-4

N.B.: Quando l'agente esce dalla griglia di gioco ottiene una penalità di -20.

Environment 1-2

Table 2: tabella reward - penalty environment 1-2

Azione agente	Elemento dell'ambiente	Punteggio
avanti/indietro/destra/sinistra	erba rasa	0
avanti/indietro/destra/sinistra	erba bassa	0
avanti/indietro/destra/sinistra	erba media	0
avanti/indietro/destra/sinistra	erba alta	0
taglio debole/medio/forte	erba rasa	-1
taglio debole	erba bassa	+1
taglio medio/forte	erba bassa	-0.15
taglio medio	erba media	+2
taglio debole/forte	erba media	-0.30
taglio forte	erba alta	+3
taglio debole/medio	erba alta	-0.45
avanti/indietro/destra/sinistra	roccia	0
avanti/indietro/destra/sinistra	albero	0
taglio debole/medio/forte	roccia	-2
taglio debole/medio/forte	albero	-1.5

N.B.: Quando l'agente esce dalla griglia di gioco ottiene una penalità di -5.

Environment 1-3

Table 3: tabella reward - penalty environment 1-3

Azione agente	Elemento dell'ambiente	Punteggio
avanti/indietro/destra/sinistra	erba rasa	0
avanti/indietro/destra/sinistra	erba bassa	0
avanti/indietro/destra/sinistra	erba media	0
avanti/indietro/destra/sinistra	erba alta	0
taglio debole/medio/forte	erba rasa	-1
taglio debole	erba bassa	+2
taglio medio/forte	erba bassa	-0.5
taglio medio	erba media	+4
taglio debole/forte	erba media	-1
taglio forte	erba alta	+6
taglio debole/medio	erba alta	-1.5

avanti/indietro/destra/sinistra	roccia	0
avanti/indietro/destra/sinistra	albero	0
taglio debole/medio/forte	roccia	-2.5
taglio debole/medio/forte	albero	-2

N.B.: Quando l'agente esce dalla griglia di gioco ottiene una penalità di -20.

2.2.2 Ambiente 2

Nel secondo ambiente, l'agente può effettuare gli stessi spostamenti consentiti nella prima versione ma prima di ogni spostamento dovrà applicare una delle 3 modalità di taglio. Anche in questo caso, sono state definite tre configurazioni di reward - penalty e sono riportate nelle Tabelle 4 - 6.

Environment 2-1

Table 4: tabella reward - penalty environment 1-1

Azione agente	Elemento dell'ambiente	Punteggio
sposta con taglio debole/medio/forte	erba rasa	-4
sposta con taglio debole	erba bassa	+3
sposta con taglio medio/forte	erba bassa	-1
sposta con taglio medio	erba media	+6
sposta con taglio debole/forte	erba media	-2
sposta con taglio forte	erba alta	+9
sposta con taglio debole/medio	erba alta	-3
sposta con taglio debole/medio/forte	roccia	-5
sposta con taglio debole/medio/forte	albero	-4

N.B.: Quando l'agente esce dalla griglia di gioco ottiene una penalità di -20.

Environment 2-2

Table 5: tabella reward - penalty environment 1-2

Azione agente	Elemento dell'ambiente	Punteggio
sposta con taglio debole/medio/forte	erba rasa	-1
sposta con taglio debole	erba bassa	+1
sposta con taglio medio/forte	erba bassa	-0.15
sposta con taglio medio	erba media	+2
sposta con taglio debole/forte	erba media	-0.30
sposta con taglio forte	erba alta	+3
sposta con taglio debole/medio	erba alta	-0.45
sposta con taglio debole/medio/forte	roccia	-2

sposta con taglio debole/medio/forte	albero	-1.5

N.B.: Quando l'agente esce dalla griglia di gioco ottiene una penalità di -5.

Environment 2-3

Table 6: tabella reward - penalty environment 1-3

Azione agente	Elemento dell'ambiente	Punteggio
sposta con taglio debole/medio/forte	erba rasa	-1
sposta con taglio debole	erba bassa	+2
sposta con taglio medio/forte	erba bassa	-0.5
sposta con taglio medio	erba media	+4
sposta con taglio debole/forte	erba media	-1
sposta con taglio forte	erba alta	+6
sposta con taglio debole/medio	erba alta	-1.5
sposta con taglio debole/medio/forte	roccia	-2.5
sposta con taglio debole/medio/forte	albero	-2

N.B.: Quando l'agente esce dalla griglia di gioco ottiene una penalità di -20.

2.2.3 Criteri di arresto

L'agente terminerà la partita quando si verificheranno una delle seguenti condizioni:

1. L'agente ha raggiunto un certo punteggio p con $p = 200$;
2. L'agente ha raggiunto un certo grado di penalità p', dove p' coincide con la penalty assegnata quando l'agente esce dalla griglia di gioco.

2.3 Stati dell'ambiente

2.3.1 Analisi degli stati dell'ambiente

Da un'attenta analisi dell'ambiente di gioco e dalla definizione degli stati, si evince che:

- Per la **prima versione dell'ambiente** abbiamo 64^7 possibili stati. Questo perchè la griglia di gioco è una matrice 8x8 quindi composta da 64 celle in cui l'agente può posizionarsi e in ogni cella si possono effettuare 7 possibili azioni;
- Per la **seconda versione dell'ambiente** abbiamo 64^{12} possibili stati. Questo perchè la griglia di gioco è una matrice 8x8 quindi composta da 64 celle in cui l'agente può posizionarsi e in ogni cella si possono effettuare 12 possibili azioni.

Per questo motivo, l'algoritmo *SARSA* e di *Q-learning classico* non possono essere applicati in quanto si ha un'eccessiva dimensione della *lookup table*. Di conseguenza verrà implementato l'algoritmo di Deep Q-Learning che consente di approssimare la tabella tramite modelli di Deep Learning.

2.3.2 Rappresentazione degli stati di gioco

Ogni cella della griglia di gioco può assumere uno dei seguenti valori:

- 0 - erba rasa
- 1 - erba bassa
- 2 - erba media
- 3 - erba alta
- 4 - albero
- 5 - roccia

Quando l'agente si sposta un una determinata posizione della griglia, per tener traccia di dove si trova, il valore di quella cella viene incrementato di 6.

2.4 Agente

Sono state definite due classi, ovvero *agent* e *agent2* che rappresentano le due implementazioni dell'agente di Reinforcement Learning. Le due classi sono responsabili di fornire l'azione scelta dall'agente, addestrare l'agente e memorizzare le transizioni stato-azione-reward-nuovo stato per poterlo addestrare. Entrambe le classi prendo in input i seguenti iperparametri:

- γ che rappresenta il fattore di sconto, ovvero quanto considerare le ricompense future. Questo parametro è stato impostato a 0.8;
- ϵ che rappresenta il trade-off tra exploration ed exploitation. Questo parametro parte da un valore pari a 1 e viene decrementato durante l'addestramento fino a raggiungere la soglia minima pari a 0.01;
- α che rappresenta il tasso di apprendimento, ovvero di quanto aggiornare i pesi del modello. Questo parametro è stato impostato a 0.001

Le azioni dell'agente sono fornite da una rete neurale implementata dalla classe *DeepQNetwork* presente nel file *model* per l'agente implementato dal file *agent* e dalla classe *QNN* per l'agente implementato dal file *agent2*. Entrambe le reti neurali prendono in input la griglia di gioco 8x8, ovvero lo stato corrente e forniscono in output l'azione da intraprendere. La differenza tra i due agenti è che il secondo (agent2) utilizza due reti neurali per far sì che i pesi del modello non cambiano troppo frequentemente e velocemente.

Inoltre è stato addestrato anche un terzo agente fornito dalla libreria Gymnasium, ovvero **stable_baseline3** che utilizza la policy chiamata *MlpPolicy* o detta anche *DQNPolicy*. Gli iperparametri di quest'ultimo sono gli stessi degli altri agenti.
Infine, sono state implementate due varianti degli agenti in cui non è consentito effettuare azioni non valide, ovvero uscire dalla griglia di gioco. In tal caso, i criteri di arresto sono rimasti invariati.

2.5 Struttura del progetto

Il progetto è così strutturato:

- **.venv** che rappresenta il virtual environment di Python;
- **gym_game** che contiene l'ambiente custom di Gymnasium, gli agenti di reinforcement learning e le reti neurali;
- **models** contenente i risultati di train e di test dei tre agenti sulle varie configurazioni dell'ambiente di gioco;
- **__main__.py** per poter addestrare e/o testare la prima versione dell'agente;
- **__main_baseline__.py** per poter addestrare e/o testare il modello di reinforcement learning fornito da Gymnasium;
- **__main_DDQN__.py** per poter addestrare e/o testare la seconda versione dell'agente;
- **requirements.txt** è un file contenente tutte le librerie necessarie per creare lo stesso virtual environment richiesto per il progetto.

3 Risultati

Per individuare la configurazione migliore dell'ambiente e dell'agente, abbiamo addestrato e testato gli agenti su tutte le configurazioni definite in precedenza. Nell'effettuare queste varie prove, abbiamo tracciato durante la fase di train e test l'andamento delle reward cumulative delle varie partite, la partita migliore e il punteggio massimo ottenuto. Tutti questi dati sono stati memorizzati all'interno di un file *csv* presente nella cartella dedicata all'agente in questione. Inoltre, durante la fase di train, è stata salvata la configurazione dei pesi del modello in esecuzione ma abbiamo memorizzato solo quelle per le quali si otteneva un punteggio soddisfacente in fase di test. Anche le configurazioni sono state memorizzate nelle rispettive cartelle degli agenti di reinforcement learning. Dopo aver testato tutti gli agenti su tutte le configurazioni dell'ambiente, gli esiti migliori ottenuti sono riportati nella Tabella 7.

Table 7: Migliori esiti ottenuti

Modello	Ambiente	Max Train Score	Max Test Score
mode1	2-1	70	49
mode2	2-3	200	80
mode3	2-1	38.4 (media ultimi 100 time step)	208

Per visualizzare i risultati degli agenti sulle altre configurazioni si rimanda ai rispettivi file csv.
A valle di ciò, possiamo affermare che l'agente con migliori prestazioni è l'agente fornito da Gymnasium, ovvero *stable_baseline3* con 50200 timesteps di addestramento sulla configurazione dell'ambiente riportato dalla Tabella 4. La Figura 2 mostra l'andamento delle prestazioni dell'agente in termini di reward in fase di test. Le Figure 3 - 6, invece, mostrano l'andamento delle prestazioni in fase di train e test delle configurazioni migliori degli altri due agenti.
Nonostante l'agente fornito da Gymnasium ha ottenuto ottimi risultati, quando si ritestano i tre agenti sui rispettivi ambienti senza effettuare la fase di train, esso non ottiene un alto punteggio mentre gli altri due ottengono reward migliori.

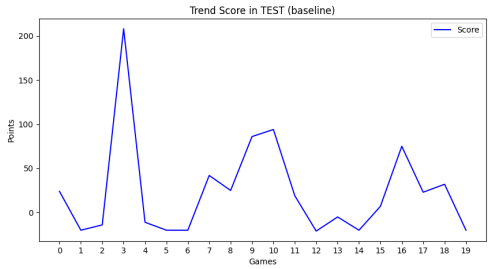

Figure 2: Andamento di stable_baseline3 in test

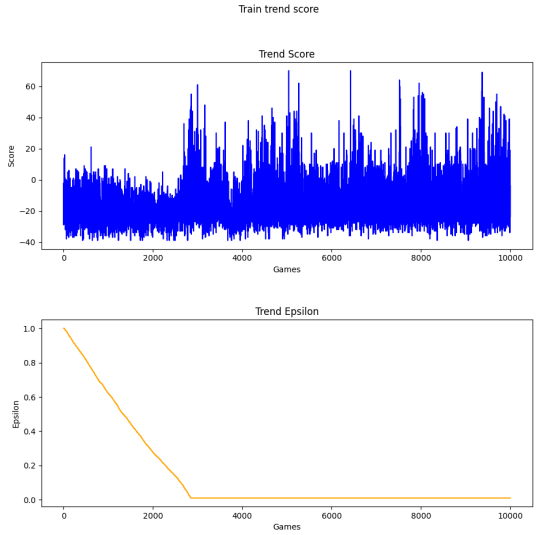

Figure 3: Andamento del 1° agente in train

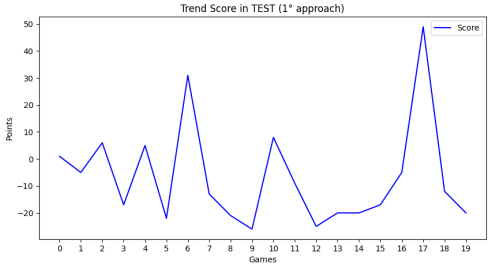

Figure 4: Andamento del 1° agente in test

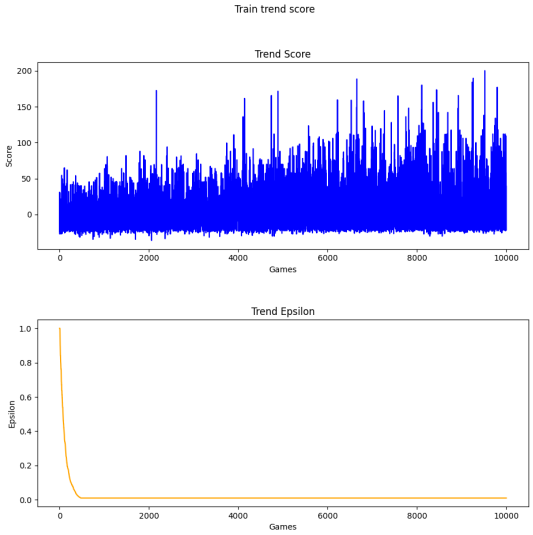

Figure 5: Andamento del 2° in train

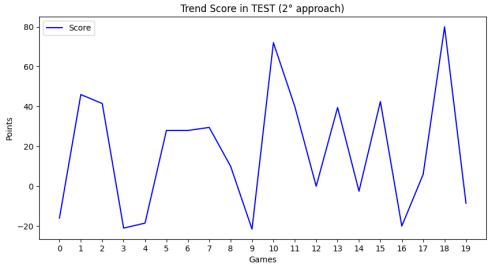

Figure 6: Andamento del 2° agente in test

4 Conclusioni

In conclusione possiamo affermare che gli algoritmi SARSA e Q-Learning classico non possono essere applicati in questo contesto a causa dell'elevato numero di stati di gioco. Nonostante ciò, la realizzazione di agenti di Deep Q-Learning hanno reso possibile la realizzazione di una soluzione di reinforcement learning andando ad approssimare la *lookup table* tramite modelli di deep learning ottenendo discreti risultati. Nella repository del progetto sono disponibili gli andamenti e le configurazioni dei tre agenti citati nelle sezioni precedenti. Per poter replicare i risultati ottenuti, bisogna eseguire i seguenti file:

- *__main__.py* per l'addestramento e/o il test del 1° agente.
- *__mainDDQN__.py* per l'addestramento e/o il test del 2° agente;
- *__main_baseline__.py* per l'addestramento e/o il test del 3° agente

Durante l'esecuzione di uno dei 3 file, se si sceglie di eseguire la fase di train è possibile selezionare il tipo di ambiente su cui si vuole provare l'agente, scegliere i valori degli iperparametri γ, learning rate e la dimensione dei batch e definire il numero di partite di train. Se si vuole condurre soltanto una fase di test, invece, verrà chiesto di scegliere l'ambiente da utilizzare e il numero di partite di test da eseguire e verrà caricata la migliore configurazione dell'agente in questione.