SVILUPPO DI UN AGENTE PER LA NAVIGAZIONE INDOOR TRAMITE Q-LEARNING

MARGARELLA GRAZIA E SANTORSA NICOLA PIO

Abstract. La navigazione in un ambiente indoor è una problematica che coinvolge diversi robot come quelli sociali o quelli adibiti alla pulizia di un ambiente. Il loro scopo principale è quello di navigare nell'ambiente designato evitando gli ostacoli, quali possono essere persone o oggetti, e raggiungere nel minor tempo possibile l'obiettivo per cui esso è predisposto. Nella seguente relazione viene mostrata la simulazione di uno scenario di navigazione indoor utilizzando algoritmi di reinforcement learning, in particolare il Q-learning e il SARSA, in grado di permettere una navigazione efficiente evitando scontri con gli ostacoli presenti nell'ambiente nel minor tempo possibile.

1. Introduzione al Problema

Lo scopo di un agente in grado di effettuare una navigazione indoor è quello di orientarsi in un ambiente per ottenere un obiettivo. In un robot sociale un potenziale obiettivo potrebbe essere quello di raggiungere un determinato punto ed effettuare delle azioni di interazione con gli utenti. Ma anche altri tipologie di robot sfruttano questo tipo di problematica per ottenere un risultato in seguito alla navigazione dell'ambiente in cui si trova.

In questo progetto andremo a simulare un robot adibito alla pulizia di un ambiente dinamico, in cui dovrà essere in grado di superare degli ostacoli. Essi vengono inseriti in modo graduale ogni volta che l'agente raggiunge un obiettivo, ossia pulisce una macchia, e dovrà farlo nel minor tempo possibile. L'algoritmo di reinforcement learning utilizzato per questo scopo è il Q-Learning.

In particolare il lavoro si è articolato in diverse fasi. Inizialmente si è progettato un ambiente 2D rappresentante una stanza con ostacoli posizionati in modo casuale, dove l'agente deve raggiungere un target evitando gli ostacoli durante il percorso.

Dopo di che sono stati definiti gli stati rappresentativi del problema cercando di fornire una descrizione il più possibile chiara e informativa.

Nello specifico per gli stati dell'ambiente sono stati definiti parametri come la presenza di ostacoli nelle vicinanze rispetto alle quattro direzioni e la direzione in cui è presente il target. Per le azioni disponibili all'agente sono state modellate le quattro direzioni in cui navigare per evitare gli ostacoli.

In seguito è stato utilizzato l'algoritmo di Q-learning per addestrare l'agente a prendere decisioni ottimali per evitare gli ostacoli e completare il percorso nel minor tempo possibile. Questa fase è stata inoltre caratterizzata dall'ottimizzazione dei parametri disponibili nell'algoritmo per migliorare il comportamento dell'agente.

Infine questa soluzione è stata confrontata con la soluzione fornita dall'algoritmo di reinforcement learning SARSA.

La repository con il codice prodotto nell'ambito del progetto è reperibile al seguente link.

2. Metodologia

2.1. **Creazione dell'ambiente 2D.** L'ambiente 2D per il seguente progetto è stato sviluppato utilizzando la libreria *PyGame*, la quale permette di creare delle interfacce grafiche personalizzate adibite allo sviluppo di videogiochi in linguaggio Python. Il caso specifico del mondo reale a cui ci siamo ispirati per costruire quest'ambiente è quello del robot aspirapolvere. Per creare l'interfaccia desiderata quindi si impostano diversi parametri come le dimensioni della finestra desiderata, nel nostro caso 850 x 850, e il frame rate. Successivamente si posizionano componenti come immagini e label di testo utilizzando come riferimento la combinazione di pixel (x, y). Nello specifico i componenti che abbiamo aggiunto all'interfaccia sono principalmente immagini e una label testuale. Le prime per una modifica dello sfondo e come rappresentazione del robot e degli ostacoli nell'ambiente, mentre l'ultima per una rappresentazione di quanti target sono stati acquisiti durante un episodio. Gli ostacoli vengono generati e posizionati in modo casuale nell'ambiente, dopo l'acquisizione di un target. Il loro numero aumenta in modo incrementale relativamente al numero di target acquisiti, anch'essi posizionati in modo casuale, dopo aver effettuato un controllo che esso non sia circondato immediatamente da ostacoli che non ne permetterebbero l'acquisizione da parte dell'agente. Il risultato ottenuto è mostrato nella Figura 1

2.2. **Agente.** L'agente sviluppato dispone di quattro possibili azioni da poter eseguire che corrispondono rispettivamente al movimento in una delle quattro direzioni cardinali. Dagli ipotetici sensori esso preleva sei osservazioni dall'enviroment che sono suddivise in due osservazioni target e quattro osservazioni ostacoli.

Le prime sono quelle che guidano l'agente verso il target e sono, come detto in precedenza, due: una per il movimento in orizzontale e l'altra per il movimento in verticale. Il loro valore varia da 0 a 2 a seconda se l'agente sia allineato al target, disallineato positivamente o disallineato negativamente.

Il secondo tipo di osservazioni riguardano gli ostacoli e sono 4. Esse rappresentano la presenza di un ostacolo nelle immediate vicinanze dell'agente nelle 4 direzioni cardinali, il valore è di tipo booleano e quindi varia tra True e False. Qui di seguito un esempio su uno stato tipico dell'agente:

$$(Vertical, Horizontal, ObsUp, ObsLeft, ObsRight, ObsDown)$$
$$(1, 0, True, False, False, False)$$

2.3. **Reinforcement Learning.** Il reinforcement learning, detto anche apprendimento per rinforzo, è una tecnica che viene utilizzata per addestrare un agente a svolgere un attività utilizzando una strategia di reward.

Questa strategia si basa sul concetto di premiare l'agente con un reward positivo quando l'azione che esso intraprende è positiva, al contrario punire l'agente con un reward negativo quando effettua un'azione da evitare.

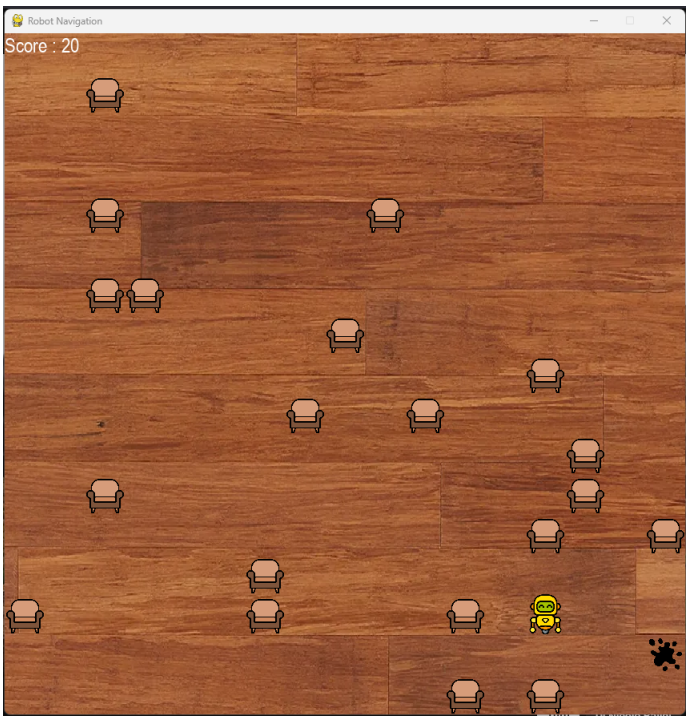

FIGURE 1. Visualizzazione environment dopo che l'agente ha raccolto 20 target

Utilizzando questo approccio l'agente apprenderà automaticamente il comportamento da tenere in ogni situazione utilizzando una funzione di massimizzazione delle ricompense, detta *value function*.

Gli algoritmi possono essere suddivisi in due categorie in base al tipo di policy che utilizzano e sono:

- **On Policy**: L'agente impara la value function tramite un azione derivata dalla policy corrente;
- **Off Policy**: L'agente impara la value function tramite un azione proveniente da un altra policy.

Nelle sezioni successive approfondiremo due degli algoritmi di reinforcement learning più utilizzati: ***Q-Learning e SARSA***

2.3.1. *Q-Learning.* Il Q-Learning è una tecnica di reinforcement learning di tipo *Off Policy* che utilizza i cosiddetti valori Q, anche chiamati *action values*. Questi vengono definiti per tutti gli stati S e le azioni A, e sono una stima di quanto ottima sia l'azione A quando l'agente si trova nello stato S. Questa stima può essere indicata con *Q(S,A)*. L'equazione che aggiorna il valore di ogni action value è la seguente:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

In questa equazione il valore $Q(s_t, a_t)$ indica l'action value per lo stato s_t e l'azione a_t, mentre il valore α indica il learning rate, il quale rappresenta con

quale velocità il modello impara il comportamento da adottare. Questo va modificato adeguatamente per evitare che il modello impari comportamenti erronei, ma che generalizzi correttamente. Infine gli ultimi valori presenti nell'equazione sono γ che rappresenta il fattore di sconto, detto anche *Discount Rate*, che determina quanto lo stato nell'istante t-esimo sia promettente o meno in termini di reward futuri, simboleggiato con r_{t+1}, ed $max_a Q(s_{t+1}, a)$ che restituisce il massimo action value per lo stato nell'istante di tempo t-esimo. Gli action values così calcolati vengono memorizzati in una *Q-Table* così da essere facilmente accessibili e modificabili.

2.3.2. *SARSA.* Il SARSA è una tecnica di reinforcement learning con alcune variazioni rispetto all'algoritmo Q-Learning, infatti mentre il Q è un algoritmo Off Policy, il SARSA è invece un algoritmo *On Policy*. Il nome dell'algoritmo è un abbreviazione e sta per **State Action Reward State Action**, il che simbolizza la tupla *(s,a,r,s',a')*. L'equazione che aggiorna il valore di ogni action value è la seguente:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

2.4. **Implementazione Q-Table.** Per implementare la struttura della Q-Table, abbiamo utilizzato una struttura dati di tipo Set nella quale il vettore dello stato, sotto forma di tupla[1] funge da chiave per ottenere il vettore delle action values per le quattro possibili azioni disponibili per ogni stato.

2.5. **Gestione dei Reward.** Alla base degli algoritmi di reinforcement learning ci sono i valori di reward che vengono assegnati all'agente quando si trova in uno specifico stato e da questi impara il comportamento. Nello specifico i reward che sono stati impostati per l'agente sono:

- Scontro con Ostacolo o Bordo: -10
- Passo: -0.5
- Acquisizione target: 10000

3. Risultati

3.1. **Metriche di Valutazione.** Per valutare le prestazioni dell'agente addestrato con i due algoritmi di reinforcement learning utilizzeremo principalmente due metriche: il **numero di target acquisiti** in un singolo episode e la **media aritmetica degli score** durante l'addestramento. Per quanto riguarda il comportamento dell'agente nello spazio è stato possibile valutarlo in modo empirico tramite l'osservazione della simulazione. Fattori positivi del training sono ad esempio la navigazione in linea retta o il superare gli ostacoli in modo veloce ed efficiente. Fattori negativi il ripetersi delle stesse azioni in modo consecutivo, non collezionare target per tante iterazioni o il conservare troppa memoria della configurazione dell'episodio precedente.

[1] Su un oggetto di tipo Tupla è possibile applicare una funzione hash mentre su una lista non è possibile

3.2. **Ottimizzazione dei parametri.** L'addestramento dell'agente ha avuto una forte fase esplorativa sulla ricerca della giusta combinazione dei parametri per il miglioramento del comportamento del robot. Ricordando che i parametri per il training sono:

- **Punteggi di premi e penalizzazioni**: essi sono definiti nell'ambiente di simulazione e vanno a premiare nel caso in cui l'agente ottenga un risultato, penalizzare in modo più cospicuo nel caso in cui si vada a colpire un ostacolo o un muro e penalizzare in maniera più lieve ad ogni passo eseguito per indirizzare l'agente a fare il suo percorso nel minor tempo possibile;
- **Numero di episodi di training**;
- **Numero di iterazioni massime per episodio**;
- **Probabilità di esplorazione**, ossia probabilità dell'introduzione di azioni casuali per permettere all'agente di esplorare lo spazio degli stati;
- **Tasso di decrescenza della probabilità di esplorazione**;
- **Minimo valore della probabilità di esplorazione**;
- **Tasso di apprendimento**, che rappresenta quanto l'agente deve essere influenzato dalle scorse osservazioni e quanto da quelle attuali;
- **Parametro** γ, ossia il fattore di sconto che permette di determinare quanto le ricompense future rendono uno stato promettente nella computazione della prossima azione.

L'ottimizzazione dei parametri citati è stata guidata dalle metriche di valutazione descritte in precedenza.

3.3. **Configurazione ottimale.** Un parametro lasciato immutato tra le varie iterazioni è stato il fattore di sconto γ uguale a 0.99, dal momento che il contesto dinamico rende lo stato in cui l'agente si trova nel momento della valutazione possibilmente svantaggioso nell'immediato, ma permettere il raggiungimento dell'obiettivo in un secondo momento.

Per quanto riguarda il numero di episodi, esso è stato fissato a 1100, dal momento che l'aumentare del numero di episodi non ha portato ad un effettivo miglioramento del comportamento dell'agente, come si può anche notare da 2. In questo numero di iterazioni, l'agente termina di esplorare i diversi stati e utilizza per 100 episodi la sola policy che ha determinato.

In ogni episodio il numero massimo di iterazioni è pari a 5000, dal momento che rappresenta un compromesso tra un numero di iterazioni valido per permettere episodi in cui vengono raccolti molti target ed episodi in cui l'agente va ad esplorare lo spazio, soprattutto nelle fasi iniziali del training.

I due fattori che hanno modificato di più il comportamento dell'agente sono stati il learning rate e il fattore di esplorazione. Per quanto riguarda il learning rate, il valore ottimale determinato è pari a 0.001. Con un learning rate più alto, come ad esempio pari a 0.01, sono state evidenziate problematiche nella navigazione. Nel breve periodo l'agente tende a schivare ostacoli presenti nell'ambiente precedente e non in quello attuale. Sul lungo periodo è stato evidenziato un decrescere vistoso delle prestazioni. Per risolvere queste criticità è stato decrementato questo valore fino a 0.001 e non ulteriormente dal momento

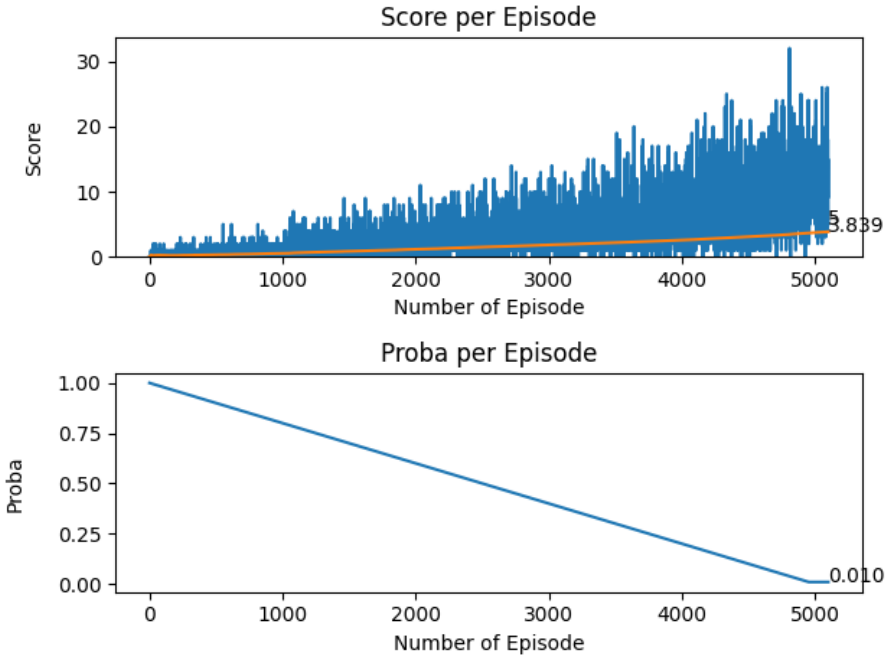

FIGURE 2. Prestazioni con 5100 episode e 0.0002 tasso decrescenza

che valori più bassi non hanno portato ad altri miglioramenti. Le prestazioni ottenute con un learning rate alto sono mostrate nella Figura 3.

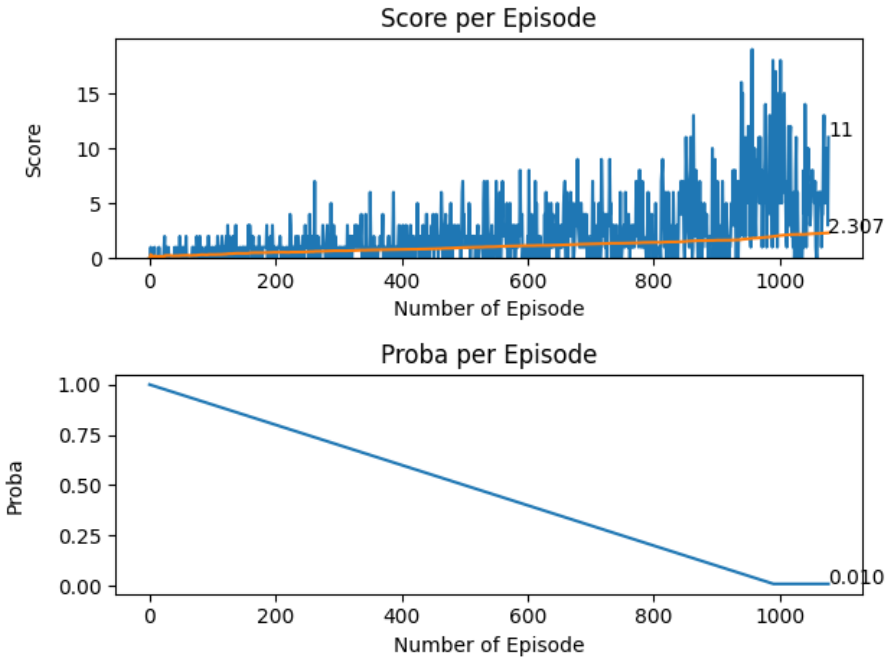

FIGURE 3. Prestazioni con learning rate pari a 0.01

Per quanto riguarda invece il fattore di esplorazione, inizializzato ad 1 e portato fino a 0.01, è stato variato il tasso di decrescenza. Il valore finale selezionato è pari a 0.001 dal momento che permette all'agente di esplorare per un numero di episodi sufficiente lo spazio e le diverse combinazioni di ostacoli che deve superare senza però perdere di generalizzazione.

3.4. **Discussione dei risultati del Q-learning.** Per quanto riguarda i risultati ottenuti con questi parametri, durante il corso delle esecuzioni è stato ottenuto un agente in grado di raggiungere picchi di 30 target raggiunti, con una media di 4 target per 1100 esecuzioni, come mostrato dal grafico in Figura 4.

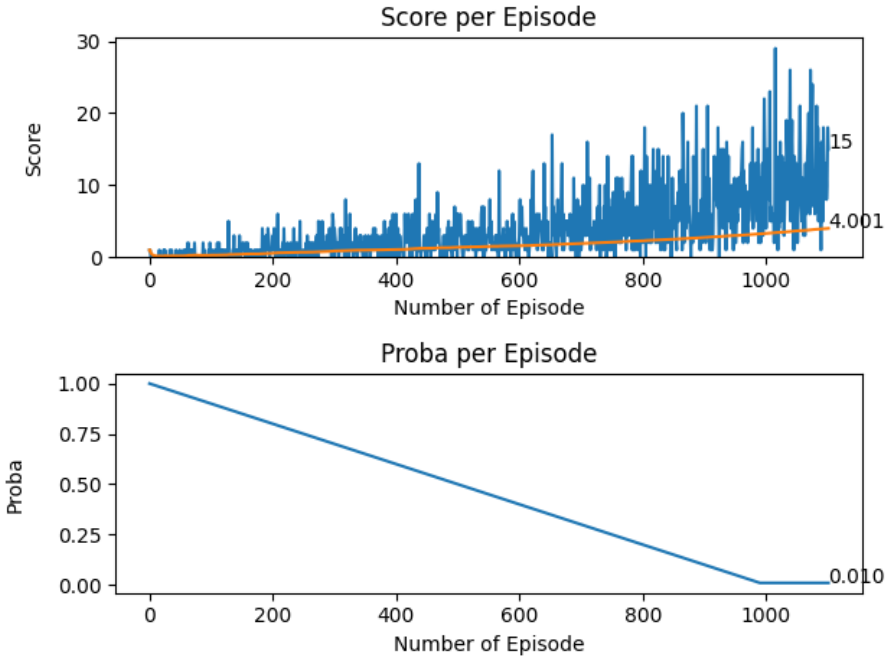

FIGURE 4. Prestazioni con i valori ottimali

Nonostante i risultati, l'agente non risulta essere sempre performante nel superare gli ostacoli. Molto spesso si ritrova a tornare nella posizione precedente piuttosto che arginare l'ostacolo, quando il target da raggiungere si trova dopo di esso. In altri casi preferisce superare tutti gli ostacoli rispetto al raggiungere l'obiettivo. Queste problematiche sono state analizzate sia con il variare delle ricompense sia con diversi learning rate e altre combinazioni dei parametri.

3.5. **Confronto con i risultati di SARSA.** Una volta ottenuti i risultati esposti nella sezione precedente, si è proceduto nell'implementazione dell'agente utilizzando l'algoritmo SARSA. In questo caso i parametri selezionati sono gli stessi descritti per il Q-learning, in modo tale da permettere un confronto tra essi. Per quanto riguarda il comportamento dell'agente durante la navigazione possiamo notare ad esempio come, rispetto al Q-learning, sia più veloce nell'apprendere la necessità di navigare in modo regolare l'ambiente. Nonostante le prestazioni restino inferiori rispetto all'altro approccio, infatti il picco massimo di target

acquisiti è 19 e la media 3.40, comunque il SARSA presenta una maggiore regolarità nell'apprendimento, quindi non presenta troppe variazioni tra un episodio e l'altro in termini di target raggiunti. Nonostante qualche vantaggio del SARSA rispetto al Q-learning, anche in questo caso si sono presentate le stesse problematiche nel superamento degli ostacoli quando il target è posto dopo un ostacolo. Le statistiche dei risultati ottenuti sono descritti nella Figura 5.

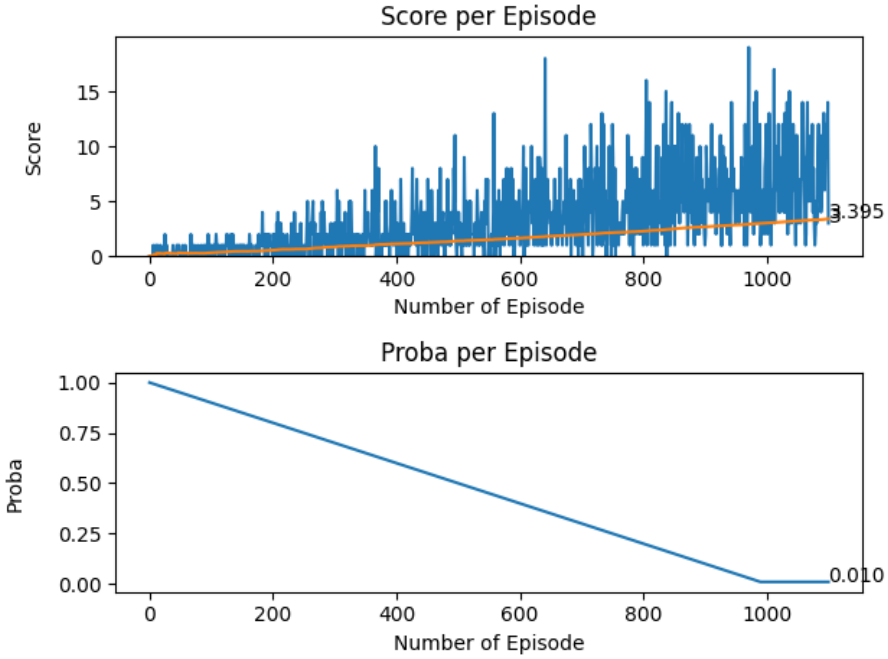

FIGURE 5. Prestazioni SARSA

4. CONCLUSIONI

In conclusione, abbiamo potuto osservare che il comportamento dell'agente, con l'utilizzo di entrambi gli algoritmi, non risulta essere ottimo, soprattutto nella circumnavigazione di un ostacolo posto in linea retta prima di un target. Un'ipotesi che abbiamo formulato trova come possibile causa la dinamicità dell'ambiente, la quale porta ad una non completa esplorazione di tutti i possibili stati dell'agente e ad una conseguente inesperienza su di essi. Esplorando eventuali soluzioni a questo problema, osservando ulteriori progetti con ambienti dinamici sul web, siamo arrivati alla conclusione che una possibile risoluzione sia quella di creare un campo di osservazione locale dell'agente abbastanza ampio da poter prevedere ampiamente l'ostacolo e agire d'anticipo, ma questo porta ad una problematica di uno spazio degli stati troppo grande e quindi con una conseguente esplorazione inefficiente degli stati. Per questa motivazione possiamo ipotizzare come un futuro sviluppo del progetto la ricerca di un trade-off tra dimensione dello stato ed efficienza di navigazione, oppure un'implementazione considerando algoritmi più complessi, come il Deep Q-Learning, con una struttura degli stati dell'agente differente.