Navigazione autonoma 3D con Reinforcement Learning (PPO) vs A* e Dijkstra

Contesto e obiettivi
La navigazione autonoma prevede che un agente si sposti da un punto iniziale a un obiettivo in un ambiente con ostacoli. Classici algoritmi di path planning come A* o Dijkstra garantiscono di trovare il percorso più breve in mappe note, ma diventano computazionalmente costosi man mano che aumenta lo spazio degli stati (link). Inoltre richiedono una rappresentazione esplicita della mappa. In alternativa, il reinforcement learning (RL) permette di apprendere direttamente la strategia di navigazione dall'esperienza, senza usare una mappa predefinita (link). Studi recenti mostrano che agenti basati su reti neurali (es. DQN) possono imparare compiti di navigazione (evitare ostacoli e raggiungere un target) in ambienti simulati (link).

In questo progetto si propone di creare in Unity un ambiente 3D relativamente semplice (ad esempio un labirinto o la pianta di un edificio) in cui un agente mobile deve raggiungere un bersaglio prefissato evitando gli ostacoli. L'addestramento dell'agente avverrà con PPO (Proximal Policy Optimization), usando il toolkit ML-Agents di Unity (link). Questa scelta è in linea con la letteratura recente: per esempio Li et al. (2024) combinano PPO con Dijkstra in un sistema ibrido per migliorare accuratezza e robustezza della navigazione (link). L'ambiente Unity potrà essere definito su una griglia (facilitando A*/ Dijkstra) oppure come area libera con ostacoli statici. Gli obiettivi principali sono progettare l'ambiente 3D, addestrare l'agente con RL, implementare A* e Dijkstra sullo stesso scenario, e infine confrontare le prestazioni in termini di percorso e tempo.

Stato dell'arte
La letteratura recente offre diversi esempi di navigazione con RL. Ad esempio, Jaramillo-Martínez et al. (2024) propongono una funzione di ricompensa studiata per minimizzare la lunghezza del percorso e il numero di svolte: in test con Deep Q-Learning l'agente apprendeva percorsi più brevi e con meno cambi di direzione rispetto ad A* (link). In quel lavoro si osserva che l'RL può ridurre le svolte del 50% e la distanza totale fino al 36% rispetto ad A*, a fronte però di tempi di calcolo molto superiori (es. oltre 100 secondi di training contro 0.01 secondi di calcolo di A*) (link). In generale queste analisi mostrano che il RL può avvicinarsi o migliorare l'ottimo in termini di lunghezza del percorso, ma richiede un costo computazionale elevato (link)(link).

Altri lavori considerano approcci ibridi. Ad esempio Li et al. (2024) presentano il metodo PPO-Dijkstra (PP-D): l'agente apprende politiche con PPO mentre Dijkstra calcola il percorso globale ottimo. Nei test questo approccio ha migliorato l'accuratezza della navigazione e la robustezza del sistema, trovando percorsi ottimali con meno collisioni in layout complessi (link)(link).

Analogamente, Imhemed & Uzun (2024) hanno usato Unity ML-Agents per simulare agenti di wayfinding in un ambiente urbano 3D, addestrando l'agente a esplorare e localizzare un target (una moschea) (link). Questi studi confermano che Unity ML-Agents è adatto per sperimentare con agenti RL in scenari tridimensionali di navigazione (link)(link).

Proposta di progetto

Il progetto prevede i seguenti passi principali:

- Costruzione dell'ambiente 3D: creare in Unity un layout semplice (ad es. un labirinto di corridoi) con ostacoli statici e un punto di partenza e uno di arrivo. Per semplificare A*/Dijkstra, si può usare una discretizzazione regolare dell'ambiente (griglia o grafo di navigazione).
- Configurazione agente RL: aggiungere un agente in Unity con osservazioni adatte (es. sensori a raggi o stato relativo) e definire l'insieme di azioni (es. avanti, sinistra, destra). Scegliere PPO come algoritmo RL e progettare la funzione di ricompensa: ad esempio penalizzare ogni passo o collisione e fornire ricompensa positiva al raggiungimento dell'obiettivo.
- Implementazione di A e Dijkstra*: sviluppare (o integrare librerie) gli algoritmi A* e Dijkstra sullo stesso ambiente. Se l'ambiente è grigliato, A* con euristica euclidea troverà rapidamente il percorso minimo; Dijkstra fornirà un benchmark alternativo. Assicurarsi di usare la stessa rappresentazione spaziale (coordinate o nodi) impiegata dall'agente.
- Addestramento e simulazione: addestrare l'agente RL con PPO attraverso numerosi episodi di interazione finché impara a raggiungere il bersaglio con buon successo. Parallelamente, eseguire A* e Dijkstra sull'ambiente fisso per ottenere i percorsi ottimali di riferimento.
- Raccolta dati e analisi: per ogni scenario (diverse configurazioni di ostacoli) misurare le prestazioni. In particolare registrare la distanza percorsa dall'agente, il numero di passi effettuati, e i tempi di calcolo dei percorsi (per A*/Dijkstra) o i tempi di addestramento/inferenza dell'agente.

Confronto e metriche

- Distanza percorsa: lunghezza del percorso seguito dall'agente RL confrontata con il percorso ottimo calcolato da A*/Dijkstra. Ciò evidenzia l'efficienza spaziale dell'agente: studi precedenti mostrano che l'RL può ridurre significativamente la distanza e le svolte rispetto ad A* (link).
- Tempo di calcolo: tempo impiegato per trovare il percorso ottimo con A*/Dijkstra rispetto al tempo di addestramento richiesto da PPO. Il training RL richiede decine-centesimi di secondi, mentre A*/Dijkstra risolvono il problema in millisecondi (link). Si può anche misurare il tempo di decisione dell'agente dopo il training (solitamente molto basso).

- Numero di svolte/cambi di direzione: quantificare quanti cambi di direzione effettua l'agente RL. L'ipotesi è che l'agente addestrato privilegi percorsi più "diretti", come osservato negli studi citati (link).

Queste metriche consentono di confrontare quantitativamente i due approcci. In letteratura si osserva spesso che l'RL riduce la lunghezza del percorso e le svolte rispetto agli algoritmi classici, a patto di aver investito abbastanza tempo di training (link)(link).

Conclusioni
In sintesi, il progetto consiste nel realizzare in Unity una simulazione 3D per il wayfinding in cui un agente apprende a navigare tramite RL (PPO) e confrontarne le prestazioni con i metodi classici A* e Dijkstra. L'uso di Unity ML-Agents permette di allenare l'agente con PPO in modo relativamente semplice (link), mentre A*/Dijkstra forniscono il percorso ottimo di riferimento. La complessità proposta è moderata: basti pensare a un labirinto discreto o a un ambiente indoor di dimensioni contenute. I risultati attesi dovrebbero mettere in luce i vantaggi di RL (percorso potenzialmente più breve e adattivo) e i suoi limiti (alto tempo di addestramento) rispetto alle tecniche deterministiche tradizionali. Questo offre un bilanciamento tra semplicità implementativa e spunti di analisi interessanti per un progetto di modelli e simulazione avanzati.

Fonti: Studi recenti su RL e navigazione autonoma (link) (link) (link) (link) (link) (link) (link).