Progetto: Q-Learning con dipendenza dal tempo per il problema di Merton senza consumo

L'obiettivo del progetto è scrivere una classe `OptimalInvestmentWithQLearning` che estenda `TimeDependentQLearning`, disponibile nel progetto zippato caricato sulla pagina Moodle del corso, sezione **Progetto di programmazione**, al fine di approssimare la soluzione numerica del problema di Merton descritto nella Sezione 1.2 dello script, **senza consumo**.

Notate che in questa versione semplificata:

- non è previsto alcun consumo durante l'orizzonte temporale;
- l'agente ottimizza solo la funzione di utilità della ricchezza finale;
- non esistono veri e propri stati assorbenti prima del tempo finale: ogni episodio termina necessariamente al tempo finale.

Implementazione e consigli:

- Come detto, la vostra classe dovrà estendere `TimeDependentQLearning`: questo vuol dire che dovete scrivere il costruttore che richiami quello di `TimeDependentQLearning`, e implementare i tre metodi astratti di `TimeDependentQLearning`;
- L'idea è di vedere lo spazio degli stati come rappresentazione della ricchezza discretizzata, e lo spazio delle azioni come percentuali del capitale investite nell'asset rischioso: potete pensare a un intervallo $[x_{min}, x_{max}]$ per i possibili valori della ricchezza, a uno $[\nu_{min}, \nu_{max}]$ per le possibili quantità degli investimenti nell'asset rischioso, e discretizzarli con step costanti Δ_x, Δ_ν. Ogni valore in questa discretizzazione sarà identificato dal suo indice: per esempio, sapete che l'i-esimo possibile valore per la ricchezza (partendo da $i = 0$) è $x_{min} + i \cdot \Delta_x$, e che il j-esimo possibile valore per l'investimento nel titolo rischioso (partendo da $j = 0$) è $\nu_{min} + j \cdot \Delta_\nu$.
- Come vedete, tra gli altri dovete completare il metodo `generateStateIndex(int oldStateIndex, int actionIndex)`, che genera (in modo stocastico) il prossimo stato, sulla base della dinamica del problema di Merton e dell'investimento scelto. Un possibile approccio consiste nel:
  (a) Calcolare l'azione corrispondente ad `actionIndex`;
  (b) simulare la dinamica stocastica del capitale come si fa nel continuo (ad esempio con un passo di Eulero);
  (c) approssimare la realizzazione ottenuta al livello discreto di ricchezza $x_{min} + i \cdot \Delta_x$ più vicino;
  (d) tornare il corrispondente indice i.
- Per ulteriori dettagli sulla struttura e sul funzionamento della classe `TimeDependentQLearning`, potete consultare direttamente i commenti scritti nel codice. La principale differenza rispetto alla versione che abbiamo visto a lezione è la dipendenza esplicita dal tempo nella Q-table e nella policy ottimale, e il fatto che gli episodi terminino una volta che viene raggiunto il tempo finale e non uno stato assorbente.

Testing:
Potete (e vi consiglio di farlo):

- Scrivere i vostri test, arricchendo la classe di test presente nella cartella `src/test` del progetto: in particolare, potete trovare test significativi per valutare l'efficacia della vostra approssimazione;
- Cambiare i parametri della classe test di cui sopra, vedendo se hanno influenza sulla qualità dei vostri risultati.

Come sottomettere la soluzione
Vi chiedo gentilmente di lavorare in gruppi di 4 persone al massimo chiedo e di mandare una mail a `andrea.mazzon@univr.it` contenente:

- La cartella zippata contenente *l'intero* progetto, quindi anche la parte test;
- Un PDF di due pagine al massimo contenente la descrizione di come è stata implementata la soluzione e una breve descrizione dei risultati ottenuti col test

Deadline: non c'è una vera e propria deadline, dipende da quando volete verbalizzare l'esame.
.