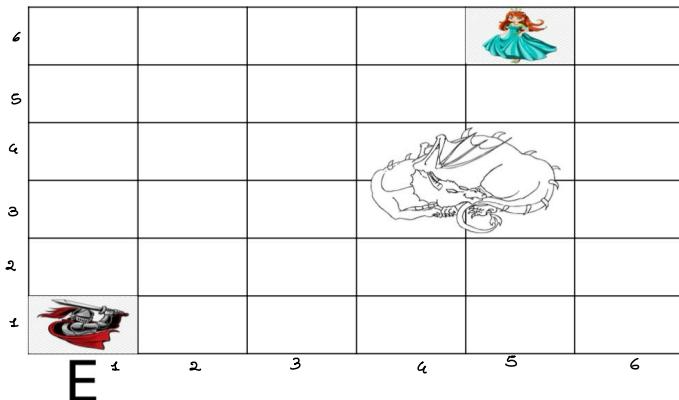


Il problema della principessa



Situazione:

- classificare dove prendere ea principessa, tornare indietro non toccando ie draghi
- Ambiente noto e completamente osservabile

Situazione iniziale: il drago si trova in AgPos(4,4) inizialmente; la principessa si trova in PrPos(5,6);

$$\text{ExPos} = \{(1,1)\}, \\ \text{DrPos} = [(4,3), (4,4), (5,3), (5,4)]$$

Come procedere:

1. Definizione PEAS dell'agente

tipo di agente	prestazioni richieste	tip. ambiente	capacità azione	
	misura di prestazione		attuatori	sensori
guidatore di taxi	sicuro, veloce, leggero alla legge, viaggio confortevole, massimizza i profitti, minimizza l'impatto su altri utenti della strada	strada, altri veicoli nel traffico, polizia, pedoni, clienti, tempo atmosferico	sterzo, acceleratore, freni, frecce, clacson, schermo, voce	telecamere, radar, tachimetro, GPS, sensori del motore, accelerometro, microfoni, touchscreen

Figura 2.4
Descrizione PEAS
dell'ambiente
operativo di un
autista di taxi
automatizzato.

Tipo di agente → Condurre

P = Performance → minimizzare costo impiegato x prendere ea
principessa e portarla all'uscita; Edo
di passare verso ie draghi

E = Environment → Ambiente noto, deterministico e osservabile

A = Actuator → braccia, gambe

S = Sensors → vista

2. Definizione spazio degli Stati

Quale oggetto sono i **caselli**? Nel mondo del labirinto, gli oggetti sono

- **Stato** → il **cavaliere (Agente)**, il **drago** e la **principessa**.

L'ambiente è composto da $6 \times 6 = 36$ caselle, di cui a me occupa il drago, una la principessa. Considerando che, inizialmente, la principessa e il drago sono oggetti fissi, avremo che ci sono 36 stati: il cui il cavaliere si possa trovare.

Analogaamente, quando il cavaliere deve tornare verso l'uscita con la principessa, muovendosi entrambi negli stessi state, avremo 36 stati.

Sommando con i 36 stati iniziali, avremo in totale 72 stati in cui si può trovare l'agente

- **Stato iniziale** → In questo caso, lo stato iniziale è la casella in cui il cavaliere si trova nella **casella (2,1)**; Dopo aver preso la principessa, l'agente avrà un nuovo stato obiettivo, partendo da un nuovo stato iniziale, ovvero **(5,6)**.

- **Stato finale** = Ci sono due casi finali, in base allo stato in cui si trova:

- Caso in cui l'agente si trova in **(2,1)** senza principessa ⇒ Stato finale = **(2,6)** senza principessa;
- Caso in cui l'agente si trova in **(5,6)** con la principessa ⇒ Stato finale = **(2,1)** con la principessa.

- **Azioni** → Il cavaliere può eseguire le azioni di movimento "destra", "sinistra", "su", "giù". Inoltre, può eseguire l'azione "prendi" appena arriva allo stato obiettivo

- **Modello di transizione** → **descrizione delle azioni (cosa comporta)**

- **destra** → Sposta l'agente dalla casella (x,y) nella casella $(x+1,y)$,
 $\forall x=1, \dots, 5$ e $\forall y=1, \dots, 6$

- **sinistra** → Sposta l'agente dalla casella (x,y) alla casella $(x-1,y)$
 $\forall x=2, \dots, 6$ e $\forall y=1, \dots, 6$

- **su** → Sposta l'agente dalla casella (x,y) alla casella $(x,y+1)$
 $\forall x=1, \dots, 6$ e $\forall y=1, \dots, 5$

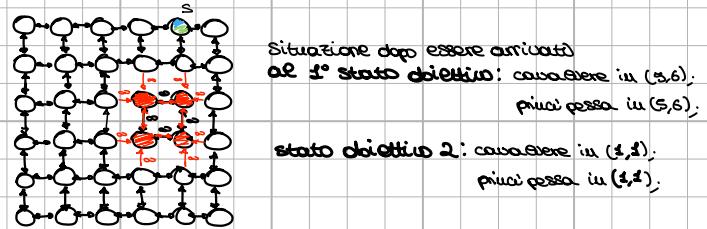
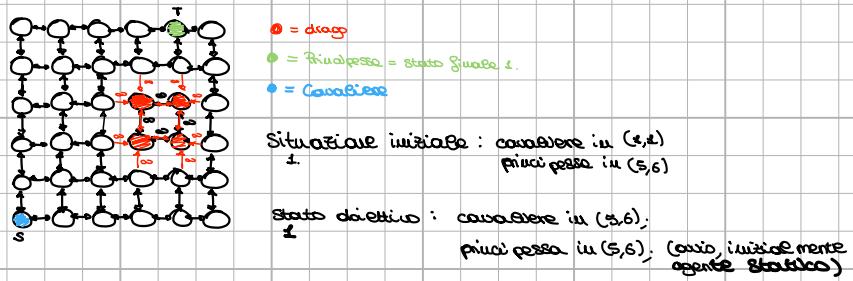
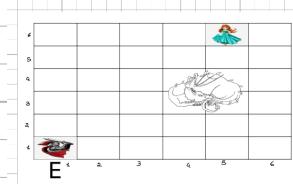
- **giù** → Sposta l'agente dalla casella (x,y) alla casella $(x,y-1)$
 $\forall x=1, \dots, 6$ e $\forall y=2, \dots, 6$

- **prendi** → "prende" la principessa appena arriva al primo stato obiettivo

- **Costo delle azioni** → Ogni azione costa 1,

tranne queste azioni in cui portano l'agente nella casella in cui si trova il drago. Per far sì che non possa andare verso il drago, poniamo il costo a ∞

3. Rappresentazione dello spazio degli Stati tramite grafo orientato pesato G=(V,E)



4. Idea algoritmo

Semplifichiamo l'ambiente nel seguente modo:

Utilizziamo solo "l'idea" degli oggetti "Principessa" e "drago".

Gli agenti "secondari" hanno scopi fondamentali sia, essendo statici, e' inutile descriverli. Pertanto, poniamo una loro "astrazione" nel seguente modo :

- da principessa si trova nello stato obiettivo.
Per il caso iniziale non è necessario fare ulteriori semplificazioni.

Quando il cavaliere raggiunge lo stato obiettivo,
si ha un richiamo alla stessa funzione ma variando
stato iniziale e stato obiettivo: in questo si vorrà
individuare che il cavaliere ha recuperato la
principessa e sta tornando verso l'uscita, inizialmente
stato iniziale.

In questo modo evitiamo di definire una classe Agente
per la principessa.

- Come nel caso descritto precedentemente, possiamo evitare di descrivere l'agente "Drago" in quanto egli non agisce con l'ambiente, ma può essere considerato un "compagno" nel caso in cui l'agente si trovi nelle celle dove c'è presente il drago.

L'obiettivo dell'agente è di evitare di trovarsi faccia a faccia
con il drago. Per individuare (ed evitare) la presenza del drago, poniamo
l'azione a, se da uno stato s, mi sposto in uno stato s', in cui si trova
il drago, di costo a.

Come viene intuito dalla traccia, l'agente conosce l'ambiente in cui si trova, essendo osservabile, noto e deterministico. A ciò, possiamo già concludere che l'agente può evitare una ricerca completa, in quanto l'ambiente non è mai deterministico, ma è parzialmente osservabile, né è inosservabile (quindi, per la risoluzione del problema, non necessitiamo degli algoritmi come Hill Climbing, Local Search e AP).

Gli algoritmi di ricerca non informata possono essere utilizzati per la risoluzione del problema. Alcuni, però, possono darci soluzioni non ottime.

1. Se BFS funziona se il costo delle azioni è $\leq k$ azione. In questo caso, non tutte le azioni hanno costo $\leq k$. Per com'è stato posto lo spazio degli stati, le azioni che portano negli stati in cui c'è presente il drago hanno costo ∞ . Se DFS potrebbe visitare quegli stati e, di conseguenza, l'agente viene mangiato dal drago.

2. Dijkstra è adatto per grafi con costi differenti. In questo caso, se utilizzassimo quest'algoritmo, noi andremmo noi a raggiungere stati in cui c'è presente il drago, ma, per arrivare allo stato obiettivo, impiegherebbe più tempo (\Rightarrow un costo delle azioni superiore rispetto al costo ottimale);

3. Se DFS, in questo caso, funziona, essendo uno spazio degli stati finiti, ma, come nell'algoritmo di Dijkstra, il costo finale non è ottimale;

4. Come per 2 e 3., l'algoritmo Best-First non porta alla soluzione ottimale.

A questo punto, è necessario che l'agente scappa alto, in modo tale che possa arrivare allo stato obiettivo con minor costo possibile. Quindi, definiamo un'euriistica $h(s)$ ottima per la risoluzione:

Utilizziamo come euriistica h la distanza di Manhattan, definita come segue:

Siano $A = (x_1, y_1)$ e $B = (x_2, y_2)$ coordinate. Allora

$$h = |x_2 - x_1| + |y_2 - y_1|$$

Quindi, nel nostro caso, sia $S = (s_1, s_2)$, posizione di un qualunque modo, e $T = (t_1, t_2)$, posizione dello stato obiettivo. Allora

$$h(s) = |5 - s_1| + |6 - s_2|$$

Se dovessimo descrivere graficamente, avremmo:

