Gabriele Cazzaniga Mat. 859738

Progetto Intelligenza Artificiale

(creato utilizzando l'interprete SWI Prolog)

PECORA SMARRITA

TRAMA

Il progetto è basato sulla storia di Shaun, una pecora che si smarrisce in montagna. Il compito dell'utente è di aiutarla a ritrovare la via di casa seguendo i comandi fornitegli dal programma.

COMANDI

Per iniziare l'avventura bisogna compilare il file principale digitando [pecora_smarrita]. Verrà visualizzata la trama e la lista dei comandi da eseguire:

```
                                PROGETTO PECORA SMARRITA

Sono passati oramai alcuni giorni di cammino, la pecora Shaun delusa ammette quindi di essersi persa in cima al monte Wannahockaloogie
e sta per chiedere indicazioni alla fauna locale, quando riesce a scorgere nel centro della vallata il proprio Recinto.
Resta un unico problema. Disgraziatamente si rende conto di aver terminato le scorte di erbetta soffice e potrebbe non riuscire a tornare
nel suo amato recinto sana e salva.
Aiuta Shaun a salvarsi posizionando nei giusti punti della mappa le caselle di erbetta soffice dove shaun potrà rifocillarsi.
Per fare in modo che Shaun ritrovi la via di casa e riesca a raggiungere il recinto ancora in vita, seguire le seguenti istruzioni:

    - Creare un mondo casuale in cui Shaun possa perdersi liberamente digitando: genera_mondo.

    - Posizionare nei punti più appropriati le caselle di erbetta soffice.

    - Premere 'invio' per ottenere il potere divino di rendere effettive le modifiche al mondo.

    - Caricare la strategia di ricerca della strada di casa digitando: default_strategy.

    - digitare 'solution(Movimenti,Ore_Totali).' per sapere se Shaun è riuscito a tornare nel recinto grazie al tuo aiuto.

    - Rinchiudere Shaun nel lussuoso recinto e buttar via la chiave definitivamente digitando: 'invio'.

    - Per visualizzare tutti gli spostamenti effettuati da Shaun nel mondo digitare: stampa.
```

- Il primo comando che verrà chiesto di eseguire sarà il comando: **genera_mondo.**

```
genera_mondo:-
    retractall(backupterreno(_)),
    retractall(larghezza(_)),
    retractall(altezza(_)),
    crea_mondo(10,10,ListaMondo),
    assert(backupterreno(ListaMondo)),
    assert(larghezza(10)),
    assert(altezza(10)),
% STAMPA TESTO */ %%%%%%%%%%%%%%%%%%%%%%%%
    add_erbettaSoffice(ListaMondo,3),
    starting_state(ST),
    myprint(ST),
    member(vita(V),ST),
    ansi_format([],'Vita: ~w~n',[V]),
% STAMPA TESTO /* %%%%%%%%%%%%%%%%%%%%%%%%
```

genera_mondo, attraverso il predicato crea_mondo, creerà un mondo 10x10 espresso da una lista composta da erba, bosco e roccia (tipi appartenenti alla categoria terreno) associato alla variabile ListaMondo.
Dopo aver fatto un backup del terreno creato, verrà chiesto all'utente di inserire 3 caselle, l'inserimento viene gestito dal predicato add_erbettaSoffice.
genera_mondo provvede anche a stampare i vari pezzi di testo e interfaccia appropriati.

```
crea_mondo(0,10,[]):-!.
crea_mondo(X,0,List):-!,
    NewX is X-1,
    crea_mondo(NewX,10,List).
crea_mondo(X,Y,[E|Coda]):-
    RandNum is random(3),
       (RandNum == 0 -> E=bosco(X,Y);
    RandNum == 1 -> E=roccia(X,Y);
             E=erba(X,Y)),
    NewY is Y-1,
    crea_mondo(X,NewY,Coda).
```

Per creare il mondo in modo casuale si utilizza la funzione random(+IntExpr) che estrae un numero casuale i compreso tra 0 <= i < IntExpr.
A seconda del numero estratto verrà quindi scelto un tipo di terreno da inserire nella lista del mondo fino al completamento della stessa.
L'inserimento nella lista viene eseguito idealmente inserendo tutti gli elementi della prima colonna (utilizzando y- -) e poi spostandosi alla colonna successiva (quando y=0, x=x-1).

```
add_erbettaSoffice(T,0):-!,
    retractall(terreno(_)),
    assert(terreno(T)).
add_erbettaSoffice(T,E):-
    starting_state(ST),
    retractall(terreno(_)),
    assert(terreno(T)),
    myprint(ST),
    member(vita(V),ST),
    ansi_format([],'Vita: ~w~n',[V]),
    writeln("\nDove vuoi posizionare la casella
    writeln("Inserisci la x:"),
    readln(Z),
    nth0(0,Z,X),
    writeln("Inserisci la y:"),
    readln(W),
    writeln(" "),
    nth0(0,W,Y),
    add_and_del(X,Y,T,NT),
    E1 is E-1,
    add_erbettaSoffice(NT,E1).


add_and_del(_X,_Y,[],[]):-!.
add_and_del(X,Y,[Z|Tail],Tail1):-
    Z=..[_P,X,Y],!,
    append([erbetta_soffice(X,Y)],Tail,Tail1).
add_and_del(X,Y,[Z|Tail],[Z|Tail1]):-
    add_and_del(X,Y,Tail,Tail1).
```

Per aggiungere le caselle inserite dall'utente al mondo contenuto nella variabile ListaMondo, il predicato deve dapprima eliminare il vecchio mondo (espresso con il nome “terreno”) per poi asserire quello nuovo contenete la casella aggiunta.
La casella viene aggiunta al mondo attraverso il predicato add_and_del che agisce in modo ricorsivo:
per prima cosa controlla se l'elemento preso in considerazione (indipendentemente dal tipo) è l'elemento che deve essere sostituito con la casella inserita. Se lo è allora interrompe la ricerca (con !) e inserisce la casella al suo posto attraverso la funzione append.
Se invece l'elemento preso in considerazione non ha le stesse coordinate della casella da inserire, lo inserisce nella lista finale e passa all'elemento successivo.

```
?- genera_mondo.

Shaun(P) deve affrontare una discesa non priva di fatiche per tornare al recinto(R).
Tenendo a mente i valori dei terreni, posizionare le 3 caselle di erbetta soffice(E) nella giusta posizione.
        e = arida erba (tempo impiegato per oltrepassare ogni radura erbosa:         1 ora)
        b = fitto bosco (tempo impiegato per oltrepassare ogni sottobosco:           5 ore)
        r = impervia roccia (tempo impiegato per oltrepassare ogni parete rocciosa: 10 ore)

        Punti Vita vengono decrementati di 1 ogni ora
        Punti Vita ottenuti rifocillandosi nelle caselle erbetta soffice(E) = 8.

  10 9 8 7 6 5 4 3 2 1
10 P r b r e e r b r b
 9 b e e b e e e e e r
 8 e e r b r b e r e b
 7 r r b b b b b b e b
 6 r b r e r b e b r e
 5 e b r r e e b e e r
 4 b e b r b r e b b e
 3 b e b e e b b b r r
 2 e e e e e b r e b e r
 1 e b b r r b r e b R
Vita: 30

Dove vuoi posizionare la casella erbetta soffice?
Inserisci la x:
|:
```

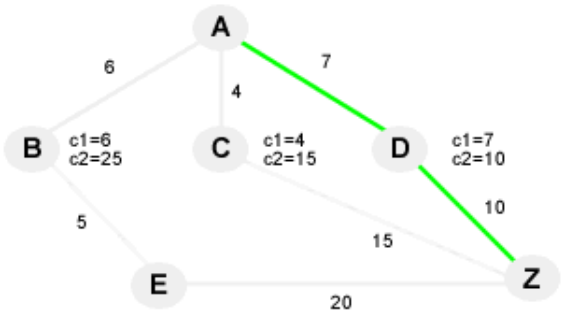
Ciò che verrà mostrato dal predicato genera_mondo, come si evince dalla foto qui sopra, tratterà nel dettaglio cosa bisogna tener conto per posizionare nei giusti punti le caselle di erbetta soffice che saranno di fondamentale importanza per il raggiungimento del traguardo.

● Il secondo comando che verrà chiesto di eseguire (dopo aver premuto 'invio' che serve solo per poterlo inserire) sarà il comando **default_strategy.**

Questo si trova nel file di supporto action.pl, il quale utilizza il file mr.pl per caricare le strategie di ricerca A* e Pota Chiusi(Introduce un'ottimizzazione rispetto alla normale ricerca troncando l'albero di ricerca ogni qual volta s'incontra un nodo già visitato. Oltre ad un miglioramento delle prestazioni permette di non finire intrappolati in possibili cicli presenti nel grafo).

STRATEGIA A* (CON DISTANZA IN LINEA D'ARIA)

Si basa sia sul costo effettivo C1 necessario per raggiungere un nodo intermedio n dal nodo di origine sia sulla stima del valore euristico di C2. $\mathbf{F(n) = C_1(n) + C_2(n)}$



In tal modo permette di analizzare dapprima i cammini migliori, ossia quelli in grado di minimizzare la somma del costo C1 e dell'euristica C2.
L'ottimalità della strategia di ricerca A* è determinata dall'euristica utilizzata per stimare il costo C2 dal nodo intermedio al nodo finale, la quale deve sempre essere un'euristica ammissibile (l'errore di previsione non è mai in eccesso).
La stima del costo di cammino "in linea d'aria" è sempre inferiore rispetto al costo del cammino effettivo. L'errore assoluto, ossia la differenza tra il costo effettivo del cammino (distanza reale) e il costo stimato (distanza in linea d'aria) è sempre positivo.

EURISTICA UTILIZZATA

```
mr:h(ListNodo,H):-
    member(pos_pecora(X,Y),ListNodo),
    member(pos_recinto(XR,YR),ListNodo),
    member(vita(Vita),ListNodo),
    H is 5*sqrt((XR-X)*(XR-X)+(Y-YR)*(Y-YR))+10/Vita.
```

L'euristica utilizzata dalla strategia A* (in questo caso) è la distanza in “linea d'aria” **H = √((ΔxAB)² + (ΔyAB)²)** moltiplicata **x5** a cui viene sommata **10/Vita** della pecora. Questo viene effettuato per pesare in modo equilibrato le forze che devono agire sulla scelta del cammino da intraprendere, in quanto se la pecora avrà poca vita, quest'ultima inciderà di più nel totale portando la pecora a scegliere una direzione più centrata al recupero della stessa. La distanza viene invece moltiplicata per far sì che più la distanza che separa la pecora dal recinto è elevata, più la scelta verrà effettuata con il criterio in linea d'aria.

●Il terzo comando che verrà chiesto di eseguire è **solution(Movimenti,Ore_Totali).**
Questo si trova nel file di supporto action.pl, il quale richiama il file search.pl e frontiera_ord.pl per utilizzare l'algoritmo generico di ricerca di una soluzione.

```
solution(Sol, Cost) :-
    starting_state(S0),
    (
        solve(S0,nc(S, Path, Cost)) ->
        (
            reverse([S|Path], SL),
            states_to_actions(SL, Sol),
            retractall(sol(_)),
            assert(sol(Sol))
        )
        ;
        (
            writeln("\n\n\n"),
            ansi_format([bold, fg(red)],'             GAME OVER~n',[]),
            writeln("Shaun purtroppo è morto di fame, le caselle erbetta soffice non sono state disposte nei punti giusti."),
            ansi_format([],'Digitare 1 per riprovare, 0 per terminare~n',[]),
            readln(Z),
            nth0(0,Z,Risp),
            writeln("\n"),
            Risp = 1 -> (backupterreno(T),add_erbettaSoffice(T,3),writeln(" "),solution(Sol,Cost));true
        )
    ).
```

Nel caso in cui si troverà una soluzione si procederà alla stampa di essa espressa attraverso la lista dei movimenti effettuati e il tempo impiegato per effettuarli

esempio:

```
?- solution(Movimenti,Ore_Totali).
Movimenti = [sotto(10, 9, 29), destra(9, 9, 28), sotto(9, 8, 23), destra(8, 8, 18), destra(7, 8, 26), destra(6, 8, 25), destra(5, 8, 24), sotto(5, 7,
23), sotto(..., ..., ...)|...],
Ore_Totali = 45 .
```

Nel caso invece non venga trovata una soluzione verrà stampata la schermata di **game over** nella quale verrà chiesto all'utente se desidera riprovare, in caso affermativo verrà recuperato il mondo generato inizialmente privo di caselle erbetta soffice e l'utente potrà così riprovare a cimentarsi nell'impresa.

```
            GAME OVER
Shaun purtroppo è morto di fame, le caselle erbetta soffice non sono state disposte nei punti giusti.
Digitare 1 per riprovare, 0 per terminare
```

ALGORITMO GENERICO DI RICERCA (con A* e pota chiusi)

```
pred cerca(frontiera(TN), nodo(TN)).
%  cerca(+F, -Sol) nondet
%  Spec:   Sol contiene una soluzione raggiunta da un nodo di F

cerca(Frontiera, Soluzione) :-
     scelto(nc(N,Path,Cost),Frontiera,F1),      % scelto dalla strategia
     ( trovato(N),
       Soluzione=nc(N,Path,Cost)                % restituita la soluzione
       ;
       vicini(N,Vicini),                        % dati dal problema
       trasforma(Vicini,nc(N,Path,Cost),FrontieraVicini),
       aggiunta(FrontieraVicini,F1,NuovaFrontiera),
                                                % in base alla strategia
       cerca(NuovaFrontiera,Soluzione)          % ricorsione coda
     ).
```

Si seleziona un nodo N dalla frontiera attraverso pred scelto,
in F1 sarà contenuto il resto dei nodi della frontiera.
Se N è un nodo finale (trovato(N)) allora N è la soluzione e si restituisce come Soluzione;
altrimenti si trovano i vicini di N potando quelli già incontrati tramite pred trasforma e si aggiungono alla frontiera rimanente F1 ordinandoli secondo la strategia A*.
Si prosegue poi la ricerca utilizzando la nuova frontiera ottenuta procedendo per **ricorsione** fino al raggiungimento di una soluzione.

●L'ultimo comando che verrà chiesto di eseguire (dopo aver premuto 'invio' che serve solo per poter inserire questo ultimo comando) è il comando **stampa.**

STAMPA

```
stampa:-
    writeln("\n"),
    starting_state(ST),
    myprint(ST),
    member(vita(V),ST),
    ansi_format([],'Vita: ~w~n',[V]),
    writeln("\npremi 'invio' per passa
    readln(_Z),
    member(pos_recinto(X,Y),ST),
    sol(S),
    print_story(S,pos_recinto(X,Y)).
```

stampa stato iniziale con **myprint.** Aspetta risposta utente (readln(_Z)). Dopo aver trovato le coordinate del Recinto e aver richiamato la lista contenente il percorso risolutivo (**sol(S)**), chiama **print_story** che stampa tutta la storia memorizzata uno stato alla volta.

```
print_story([St], X):-!,
    myprint([St,X]),
    /*TESTO*/
print_story([St1|St2], X):-
    myprint([St1, X]),
    readln(_Z),
    print_story(St2, X).
```

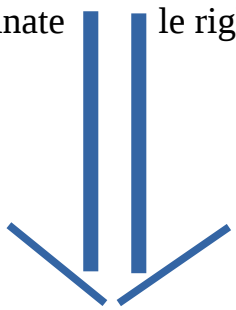
Stampa lo stato corrente con **myprint.** st1 è l'azione corrente, st2 contiene le rimanenti azioni contenute in **sol(S)**, X è la posizione del Recinto, la quale viene aggiunta all'azione estratta da sol(S) creando una lista che verrà utilizzata in **myprint.** Una volta stampato lo stato corrente si aspetta la risposta dell'utente. Per stampare la mappa completa di ogni stato viene utilizzata la ricorsione che richiama **print_story.** La ricorsione termina quando nella lista delle azioni risolutive (**sol(S)**) rimane un'unica azione.
Quando ciò accade viene eseguito il pred **print_story** che fino a quel momento aveva fallito (in quanto la lista conteneva più di un'azione) che termina stampando l'ultimo stato attraverso **myprint**.

```
myprint(ST):-
    larghezza(Larg),
    altezza(Alt),
    print_raw(ST, Larg, Alt).
```

larghezza fissa,
altezza fissa.
Stampa tutta la griglia dello stato richiamando **print_row.**

```
print_raw(_St, _Larg, 0):-!.
print_raw(ST, Larg, Alt):-
    print_column(ST,Larg, Alt),
    NewAlt is Alt - 1,
    print_raw(ST, Larg, NewAlt).
```

viene utilizzato il pred **print_column** per stampare tutti i caratteri della prima riga. Terminata la prima riga si passa alla seconda decrementando Alt (Y=Y-1) e richiamando **print_raw** attivando la ricorsione e quindi richiamando più volte **print_column**. La ricorsione termina quando Alt == 0 ovvero quando saranno terminate le righe da stampare.



```
print_column(_ST, 0, _Y):-!, write("\n").
print_column(ST, X, Y):-
    (
        (member(Pred, [pos_pecora,destra, sinistra, sopra, sotto]),(Obj1 =..[Pred,X,Y,_V];Obj1 =..[Pred,X,Y]),member(Obj1, ST)
                -> (!,ansi_format([bold, fg(white)],'P',[]))
                ;
                (member(pos_recinto(X,Y), ST)-> (!,ansi_format([bold, fg(red)],'R',[])))
        )
        ;
        (terreno(T),
         member(E, T),
         E =.. [Tipo, X,Y],!,
         (Tipo == bosco -> (ansi_format([bold, fg(blue)],'b',[]),!)
         ;
         (Tipo == roccia -> (!,ansi_format([bold, fg(black)],'r',[]))
         ;
         (Tipo == erbetta_soffice -> (!,ansi_format([bold, fg(green)],'E',[]))
         ;
         (!,ansi_format([bold, fg(green)],'e',[])))))
        )
    ),
    NewX is X-1,
    ansi_format([],' ',[]),
    print_column(ST,NewX,Y).
```

Ci si chiede se nello stato preso in considerazione c'è un "**pred**" che ha nome = [pos_pecora, destra, sinistra, sopra, sotto] e se come coordinate ha le stesse di quelle prese in considerazione:

Se c'è viene stampato il carattere 'P'.

Se non è presente, ci si chiede se in (X,Y) è quindi presente il Recinto:

Se c'è viene stampato il carattere 'R'.

Se non è presente, si spezza l'elemento **E** terreno in [Tipo, X, Y],
si verifica se le coordinate prese in considerazioni sono uguali alle coordinate dell'elemento **E**:

Se sono uguali si stampa o il carattere 'b' o 'e' o 'E' o 'r' a seconda del tipo associato a E.

Se non sono uguali si passa ad un altro **E**lemento del terreno
(si parte dalla prima posizione e si controllano tutte le posizioni fino al raggiungimento delle coordinate di un elemento **E**).
Una volta selezionato l'elemento giusto e stampato il carattere corrispondente,
si passa alla prima posizione della colonna successiva decrementando X (X = X-1).
Vine infine stampato uno spazio come separatore tra i caratteri.
Una volta creata tutta la riga (quando X = 0) la ricorsione termina stampando l'ultimo carattere, ovvero '\n'.

LE AZIONI

Le azioni che possono essere effettuate dalla pecora sono sinistra,destra,sopra,sotto.

esempio (sinistra):

```
add_del(sinistra(GX,GY,NX),ST,Fluenti,[pos_pecora(GPX,GPY),vita(X)],Costo):-
        member(pos_pecora(GPX,GPY),ST),
        member(vita(X),ST),
        GX is GPX+1,
        GY is GPY,
        calcolacosto(GX,GY,Costo,E),
        NX is X - Costo + E ,
        NX>0,
        ((E>0 , member(visitata_erbetta(GX,GY),ST)) -> fail;true),
        (E>0 -> Fluenti = [pos_pecora(GX,GY),vita(NX),visitata_erbetta(GX,GY)];Fluenti = [pos_pecora(GX,GY),vita(NX)]).
```

In queste viene calcolato:

- lo spostamento,
- il costo del terreno visitato tenendo conto della vita che deve essere aggiunta nel caso sia un terreno di tipo erbetta soffice,
- viene controllato che ad ogni movimento Shaun disponga ancora di vita
- infine viene controllato che se Shaun ha già usufruito di una casella specifica di erbetta soffice non potrà usufruirne una seconda volta.

Quest'ultimo punto viene effettuato trattando la casella erbetta soffice come fluente e agendo come segue:

- Nel caso non sia ancora stata visitata, la pecora sarà libera di muoversi sulla casella e rigenerarsi, la casella visitata verrà quindi aggiunta ai fluenti come visitata_erbetta(X,Y) e quindi allo stato.
- Nel caso sia già stata visitata, la pecora non potrà rivisitarla e ciò viene effettuato controllando che nello stato ci sia visitata_erbetta(X,Y), se è presente,l'azione nella direzione dell'erbetta soffice non potrà essere effettuata.