AGENTI & PROBLEM SOLVING

INTELLIGENZA ARTIFICIALE (A.A. 2024-2025)

CLAUDIU DANIEL HROMEI

Università di Roma
Tor Vergata

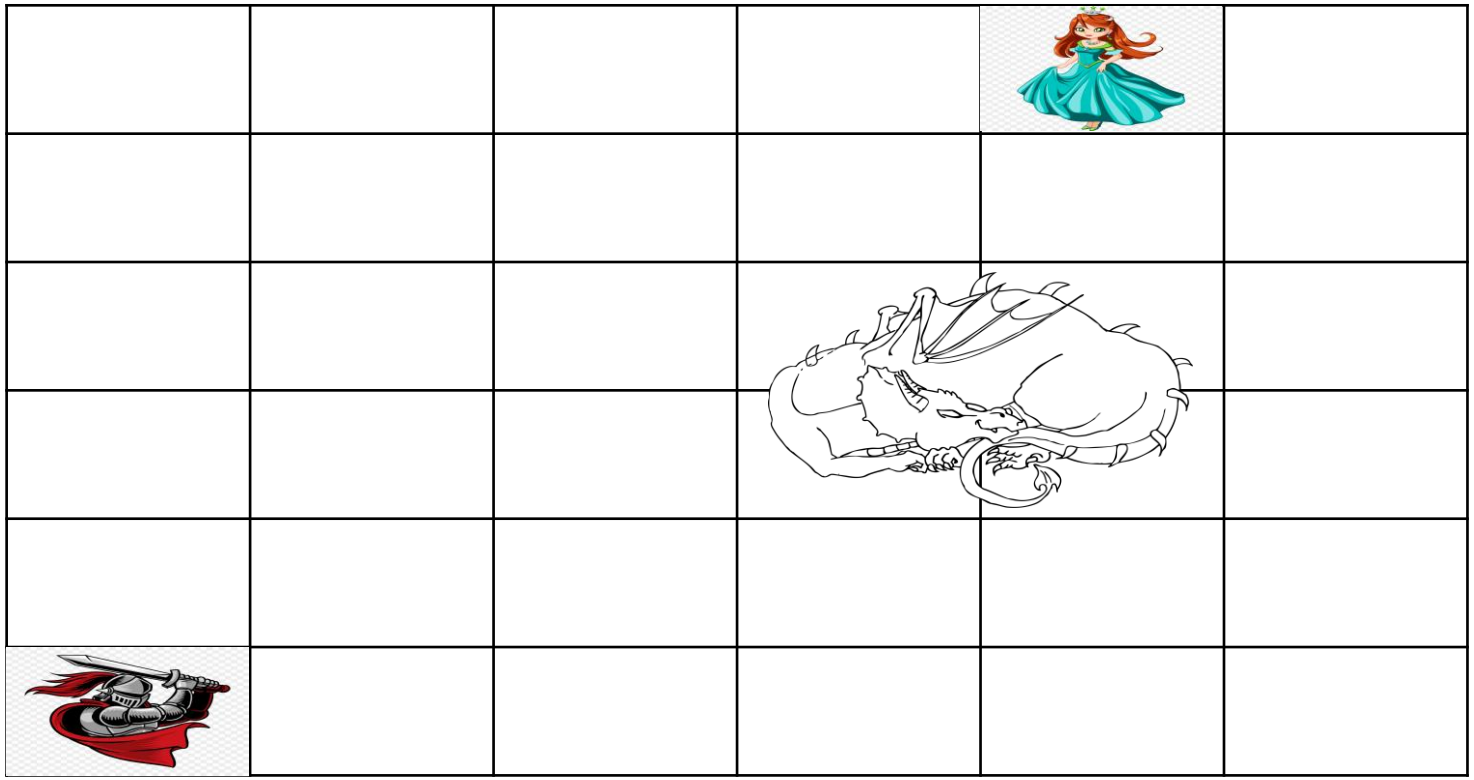
OVERVIEW

- Introduzione ad un problema in un labirinto
- Esempi di risoluzione
- La formalizzazione del problema
- Soluzione in Pseudocodice
- What if ..? Esercizi

INTRODUZIONE AL LABIRINTO

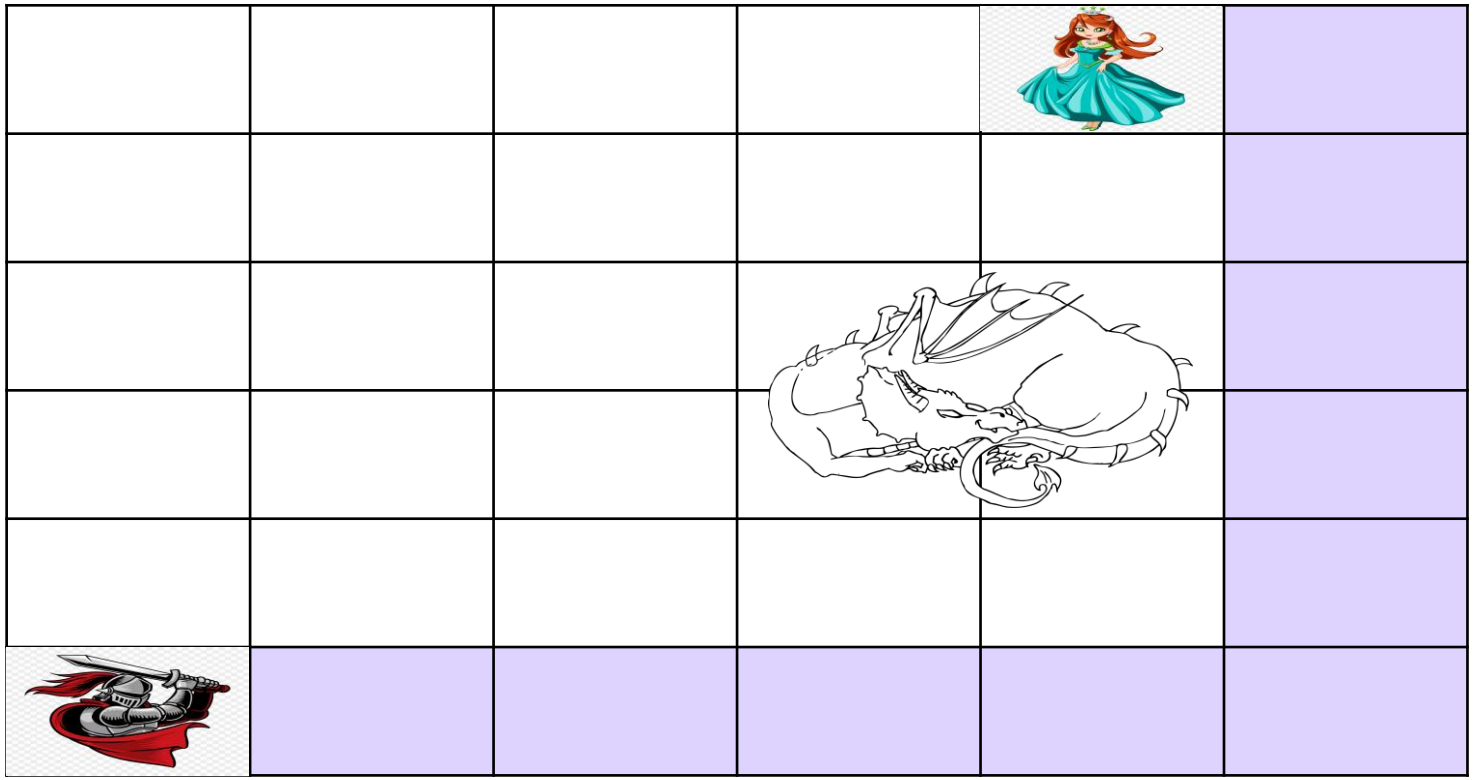
- Una principessa è stata rinchiusa e legata in una grotta insieme ad un drago che le fa da guardia.
- Il cavaliere, che vuole salvarla e riportarla a casa, deve entrare nella grotta, aggirare il drago dormiente, prendere la principessa in braccio e portarla all'uscita.
- Il drago è spietato e non deve essere risvegliato, altrimenti mangerà in un solo boccone il cavaliere.
- Le luci sono accese, c'è una sola entrata/uscita.

ESEMPIO



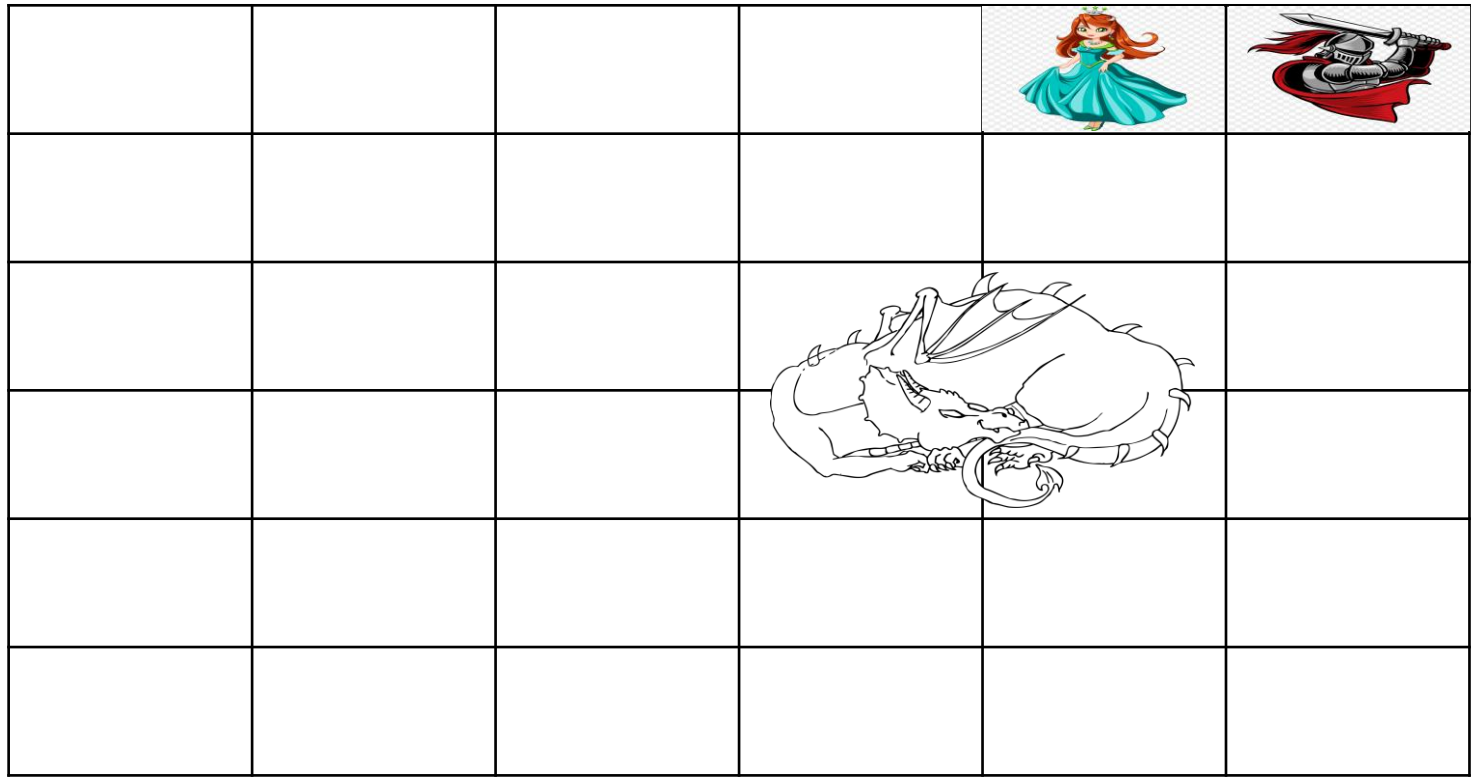
E

ESEMPIO

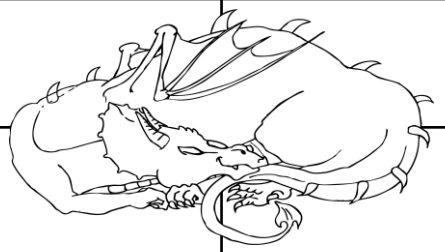


E

ESEMPIO

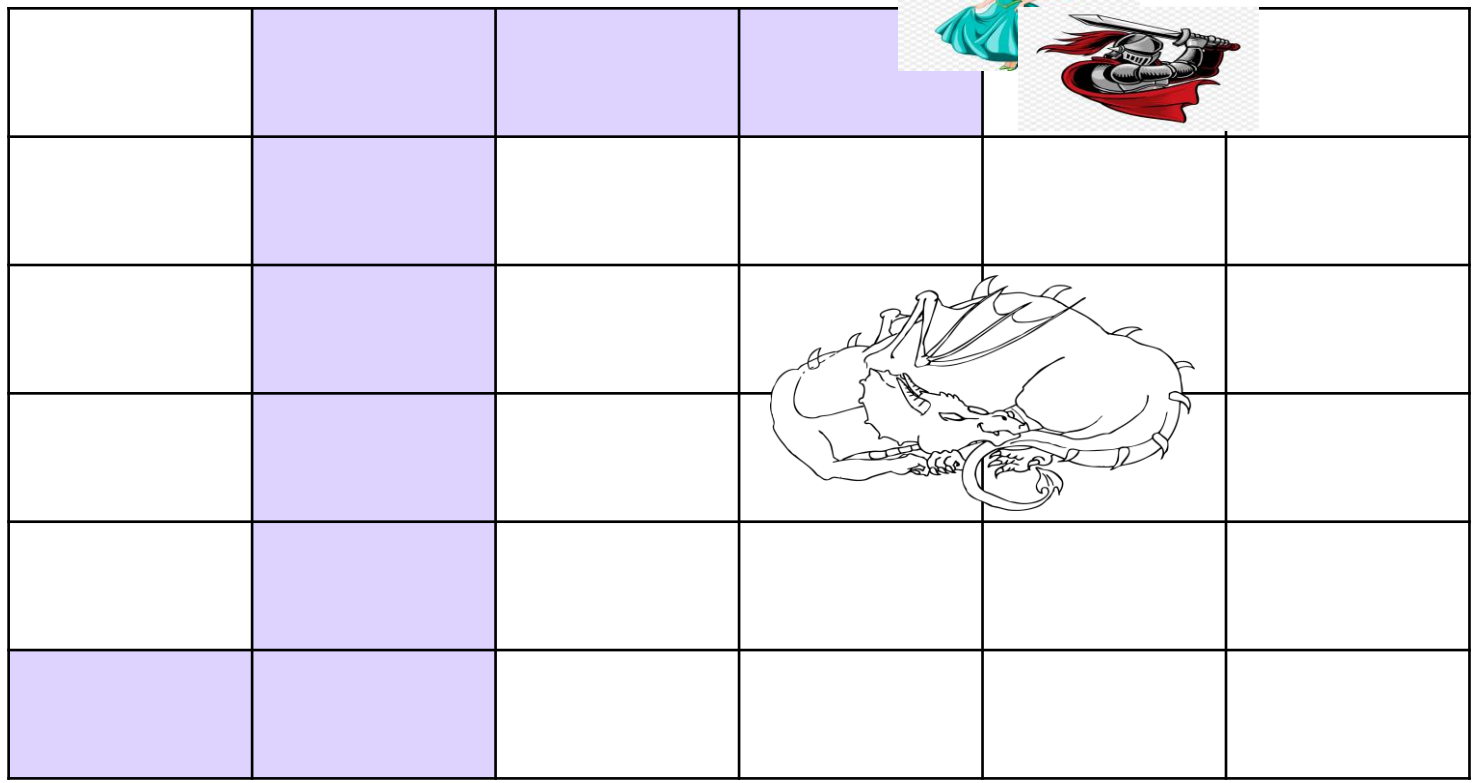


ESEMPIO



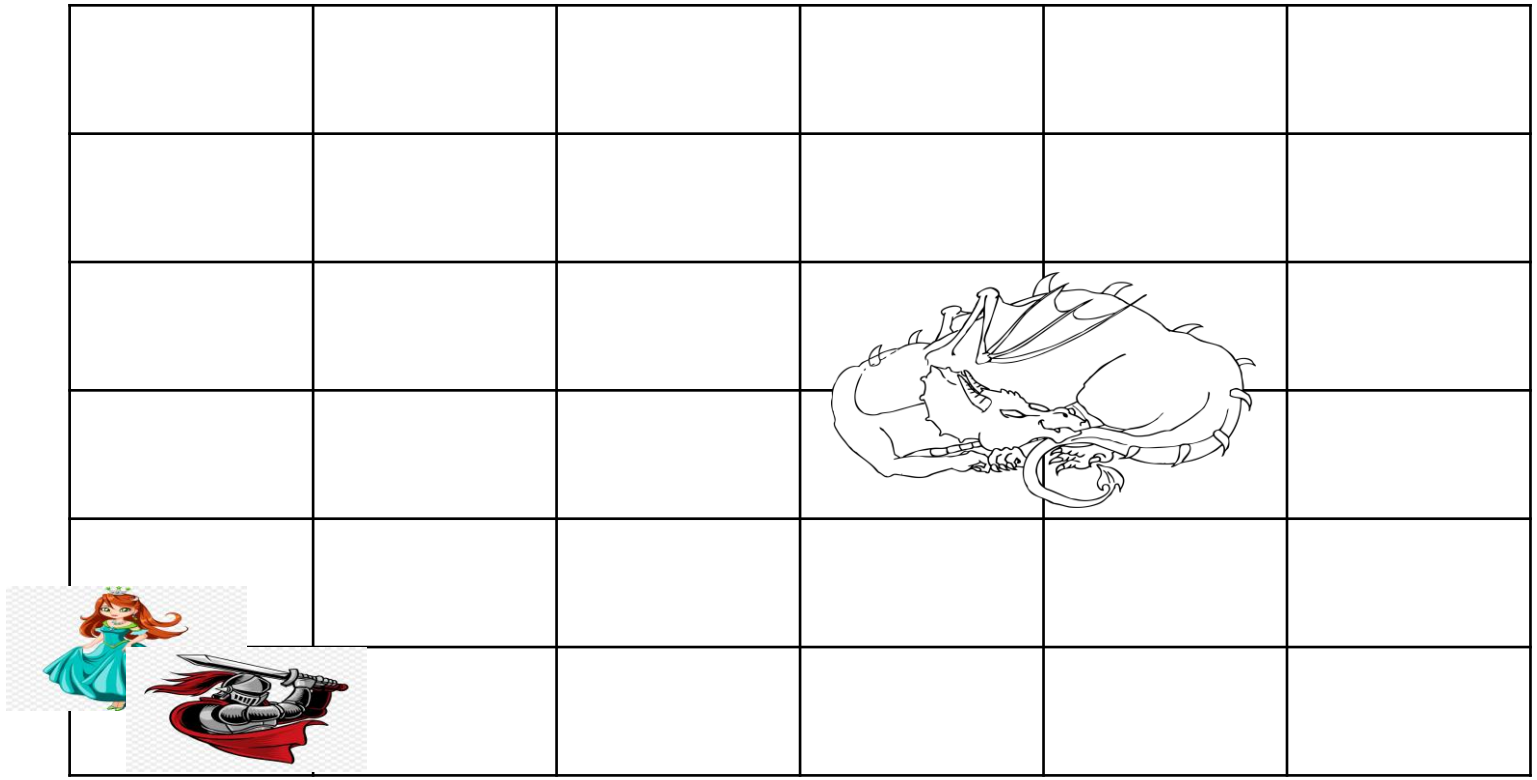
E

ESEMPIO



E

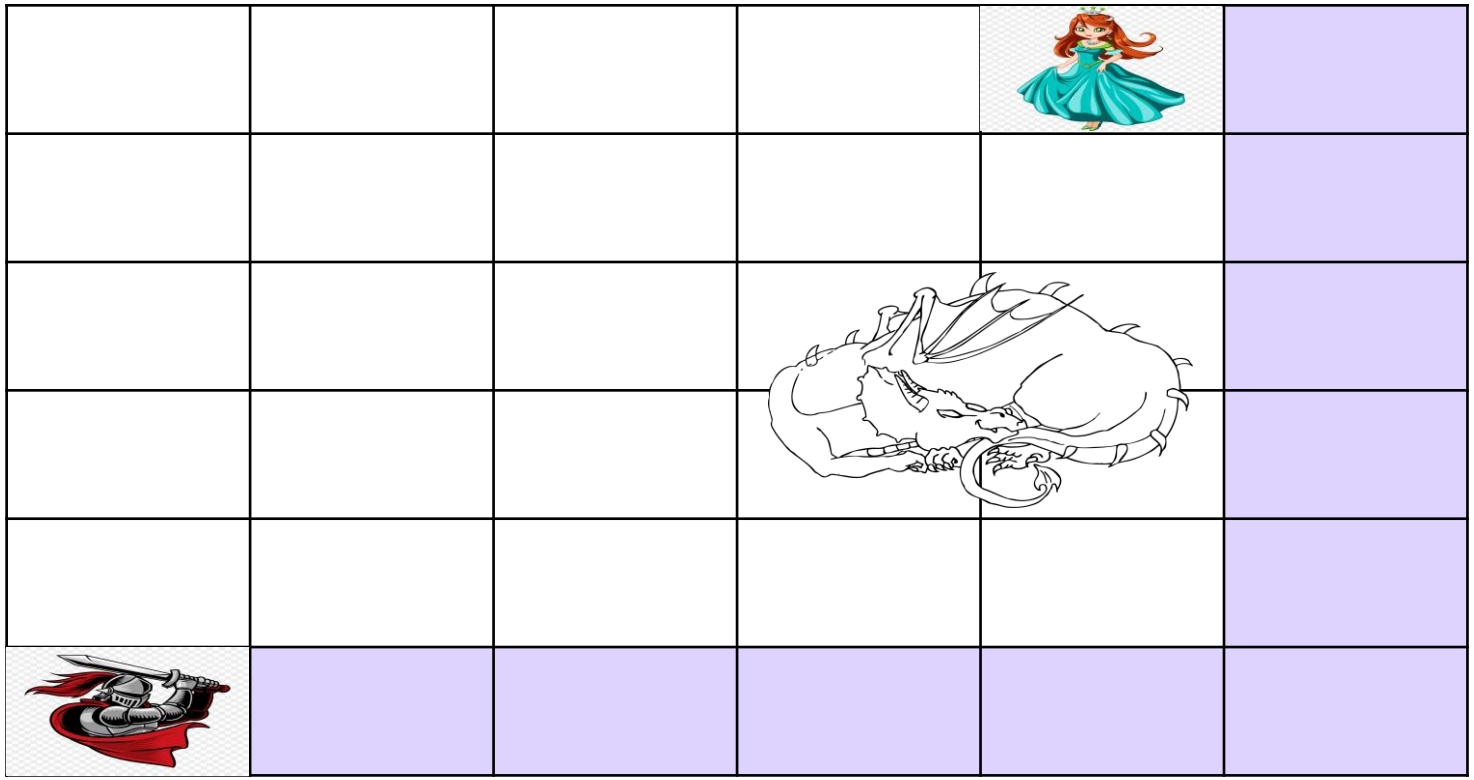
ESEMPIO



E

ERA LA MIGLIORE SOLUZIONE?

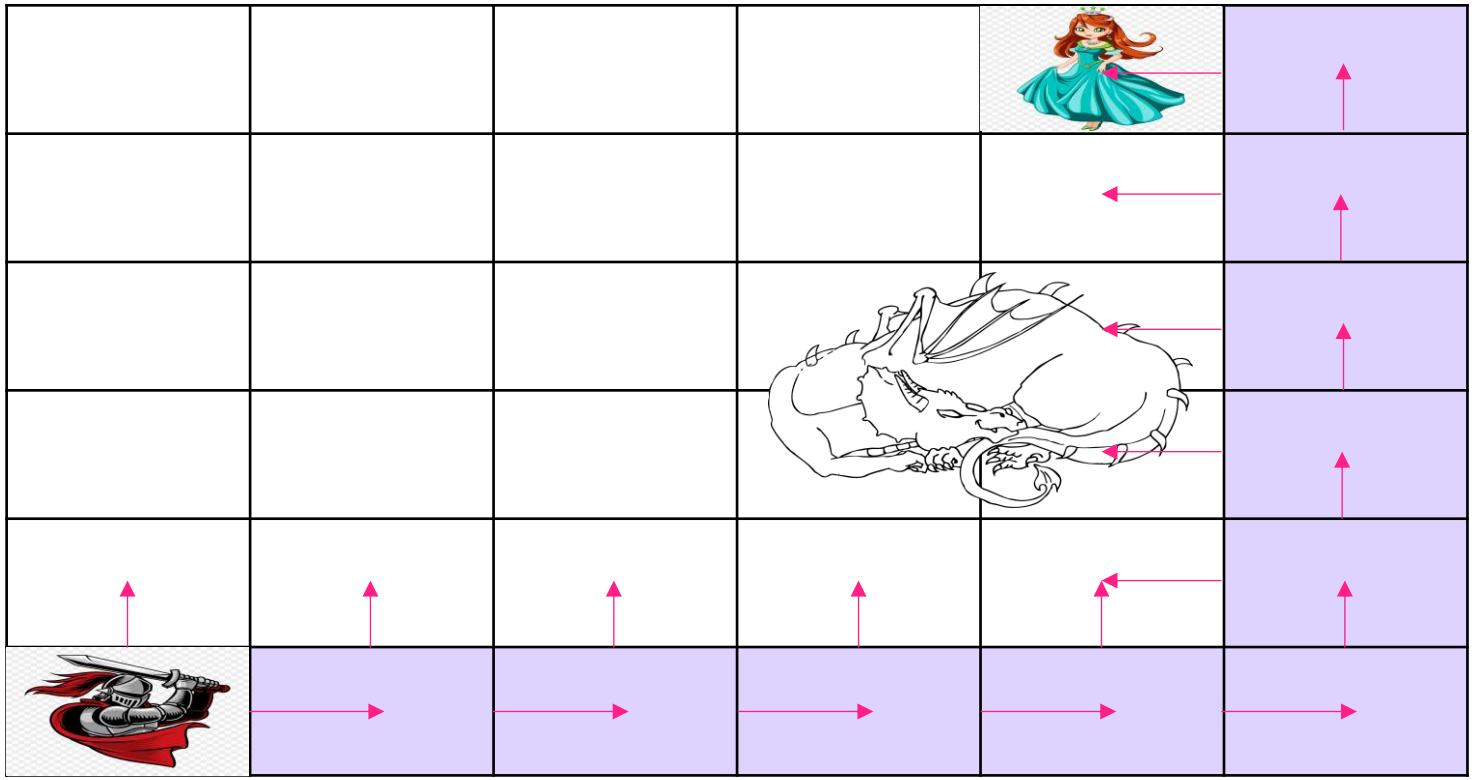
DFS=10



E

COSA ESPANDE IL DFS?

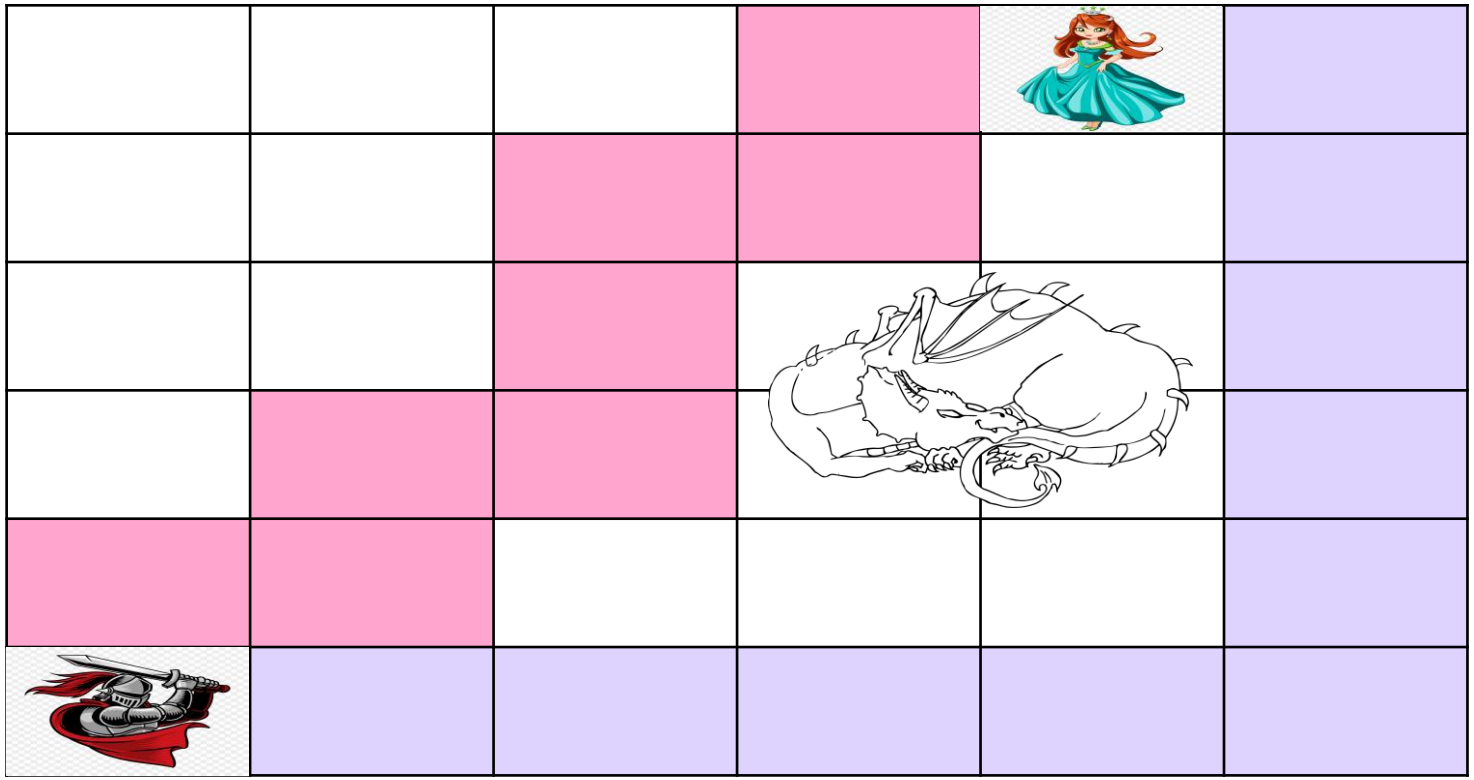
DFS=10



E

UN PATH MIGLIORE?

DFS=10

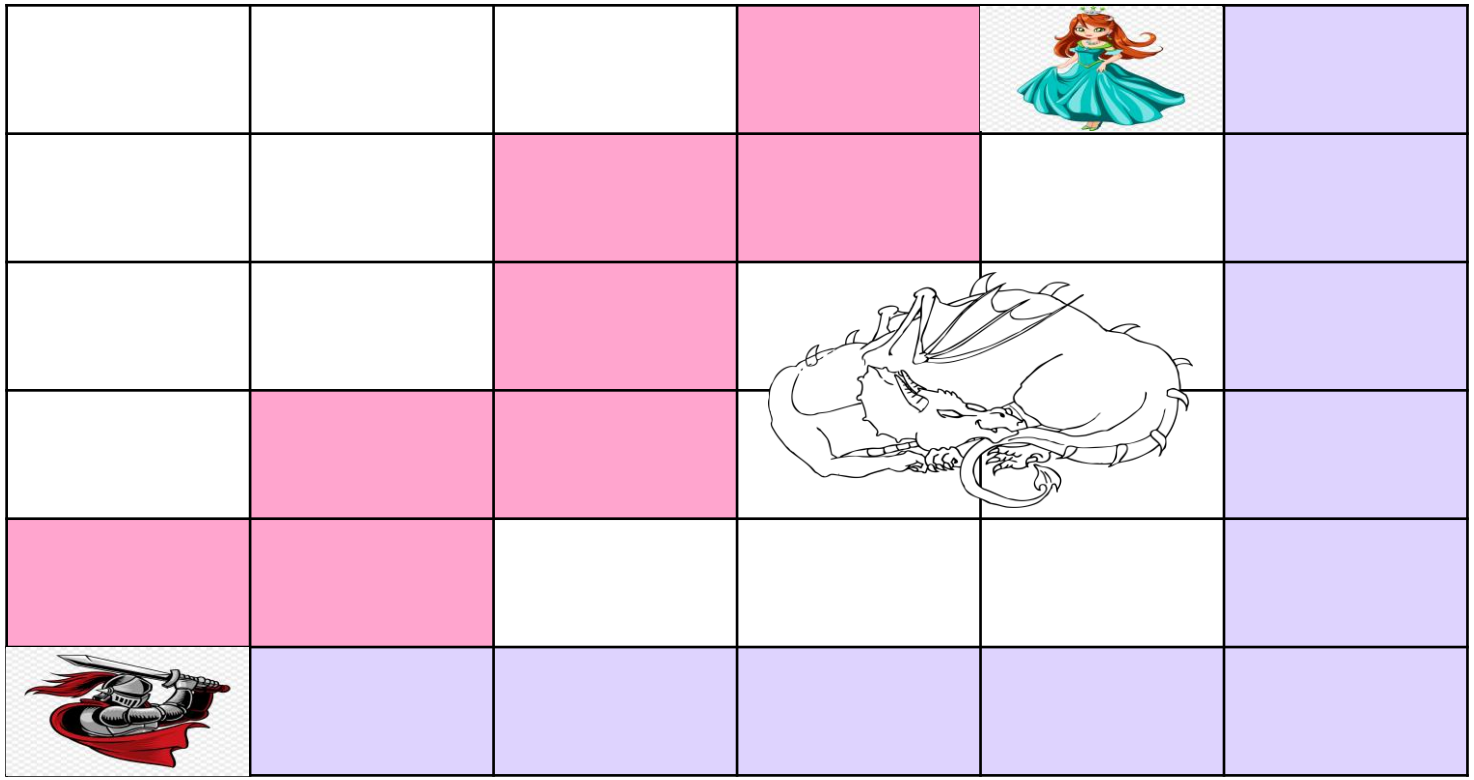


E

PERCHÉ A* È MIGLIORE

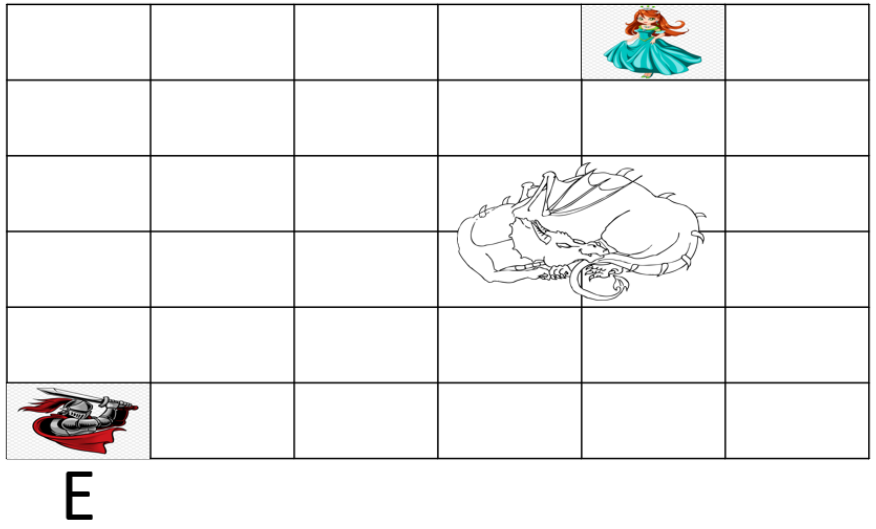
A*= 8

DFS=10



E

OBIETTIVO



- Modellare il problema:
  - Definizione *PEAS* dell'agente.
  - Definizione delle proprietà dell'ambiente.
- Dare una soluzione in Pseudocodice, definendo le procedure importanti `save(AgPos, DrPos, ExPos, PrPos) -> Path`, dove:
  - `AgPos` è la posizione (X_a, Y_a) iniziale dell'agente
  - `DrPos` è la posizione $((X_1, Y_1), (X_2, Y_2), (X_3, Y_3), (X_4, Y_4))$ del drago
  - `ExPos` è la posizione di entrata/uscita
  - `PrPos` è la posizione (X_p,Y_p) della principessa
  - `Path` è la sequenza di azioni (N,S,E,W,P) che il cavaliere esegue per salvare la principessa

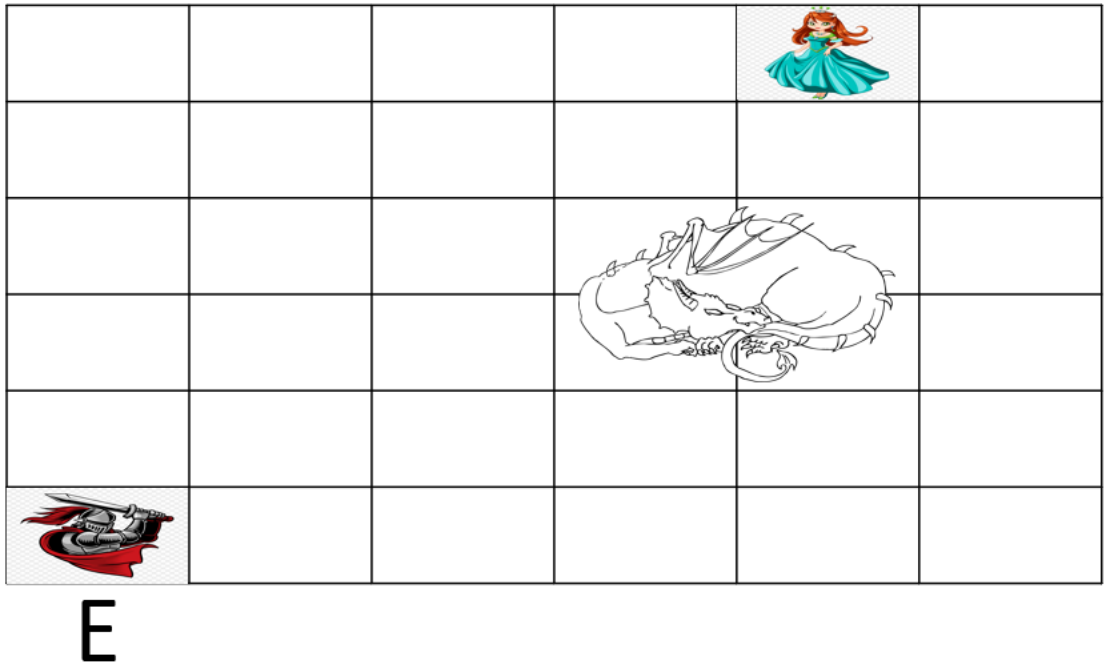
SOLUZIONE

Applicare A* da `AgPos` per raggiungere `PrPos`, in cui `DrPos` hanno valore ∞, in modo da non raggiungere mai il drago.
Algoritmo ad alto livello:

1. Calcolare percorso P_1 per `PrPos` con A*;
2. Aggiungere a `Path` le mosse ed eseguire il percorso P_1;
3. Afferrare la principessa;
4. Calcolare percorso P_2 per `Exit` con A*;
5. Aggiungere a `Path` le mosse ed eseguire il percorso P_2;
6. Arrivato in `Exit`, uscire e ricevere la gloria eterna del Re.

MODELLAZIONE DELL'AMBIENTE

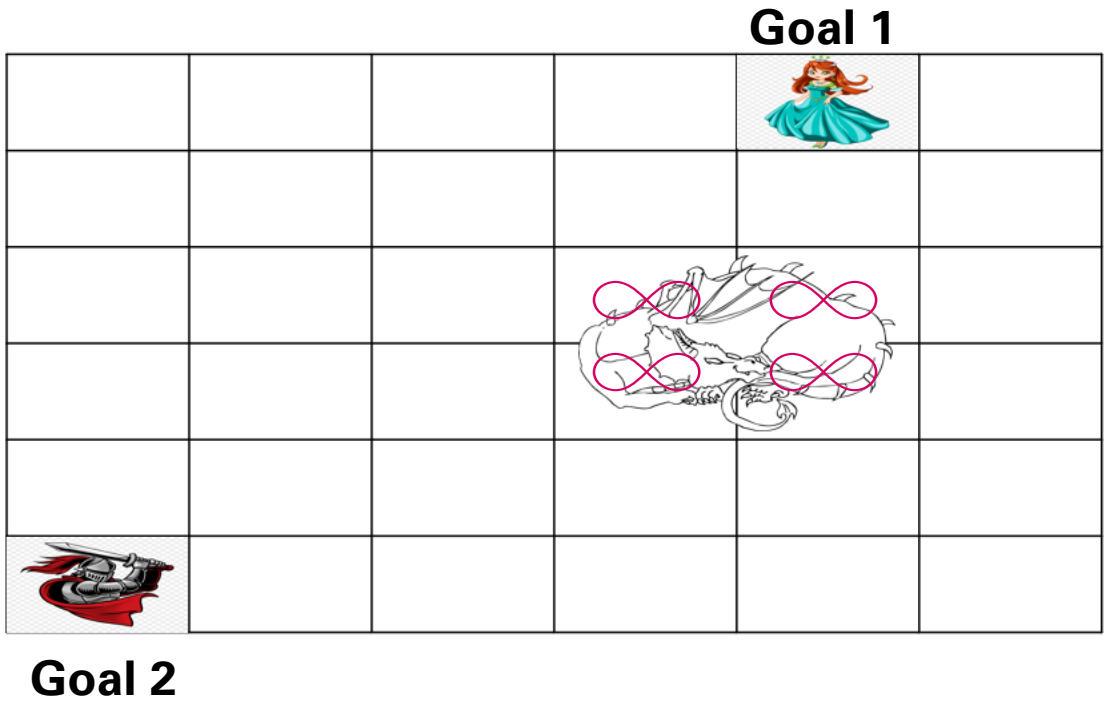
```
%%%%%%%%%%%%%%%%%%%% MAZE 6x6 %%%%%%%%%%%%%%%%%%%%
%    [(5,0), (5,1), (5,2), (5,3), (5,4), (5,5)]
%    [(4,0), (4,1), (4,2), (4,3), (4,4), (4,5)]
%    [(3,0), (3,1), (3,2), (3,3), (3,4), (3,5)]
%    [(2,0), (2,1), (2,2), (2,3), (2,4), (2,5)]
%    [(1,0), (1,1), (1,2), (1,3), (1,4), (1,5)]
%    [(0,0), (0,1), (0,2), (0,3), (0,4), (0,5)]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

edge((0,0),(0,1), 1).
edge((0,0),(1,0), 1).
edge((0,1),(0,2), 1).
edge((0,1),(1,1), 1).
edge((0,2),(0,3), 1).
edge((0,2),(1,2), 1).
edge((0,3),(0,4), 1).
edge((0,3),(1,3), 1).
edge((0,4),(0,5), 1).
edge((0,4),(1,4), 1).
edge((0,5),(1,5), 1).
```



E

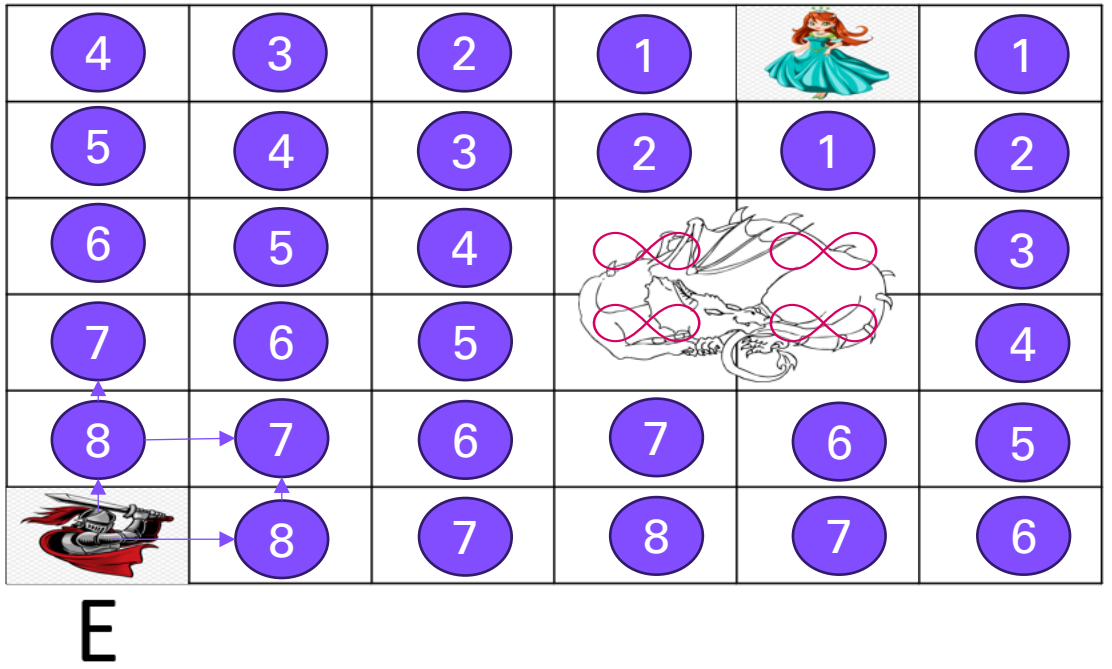
PROCEDURA PRINCIPALE

```
save(AgPos, DrPos, ExitPos, PrPos) -> Path {
    // make dragon unreachable => set edge cost to infinite
    DrPos is ((X1, Y1), (X2, Y2), (X3, Y3), (X4, Y4)),
    set_cost(edge((_,_), (X1, Y1), infinite)),
    set_cost(edge((_,_), (X2, Y2), infinite)),
    set_cost(edge((_,_), (X3, Y3), infinite)),
    set_cost(edge((_,_), (X4, Y4), infinite)),

    // set the first goal, i.e. go to the Princess
    Path_to_Princess, Cost is astar(AgPos, PrPos, Manhattan_H),

    write("Agent goes to the Princess: " + Path_to_Princess),
    write("With cost: " + Cost),
    write("Princess picked up!"),

    // now go to the exit
    Path_to_Exit, Cost2 is astar(AgPos, ExitPos, Manhattan_H),

    write("Agent and Princess go to exit: " + Path_to_Exit),
    write("With cost: " + Cost2),

    Path is Path_to_Princess + Path_to_Exit,
    write("Agent and Princess are out!"),
    return Path.
}
```



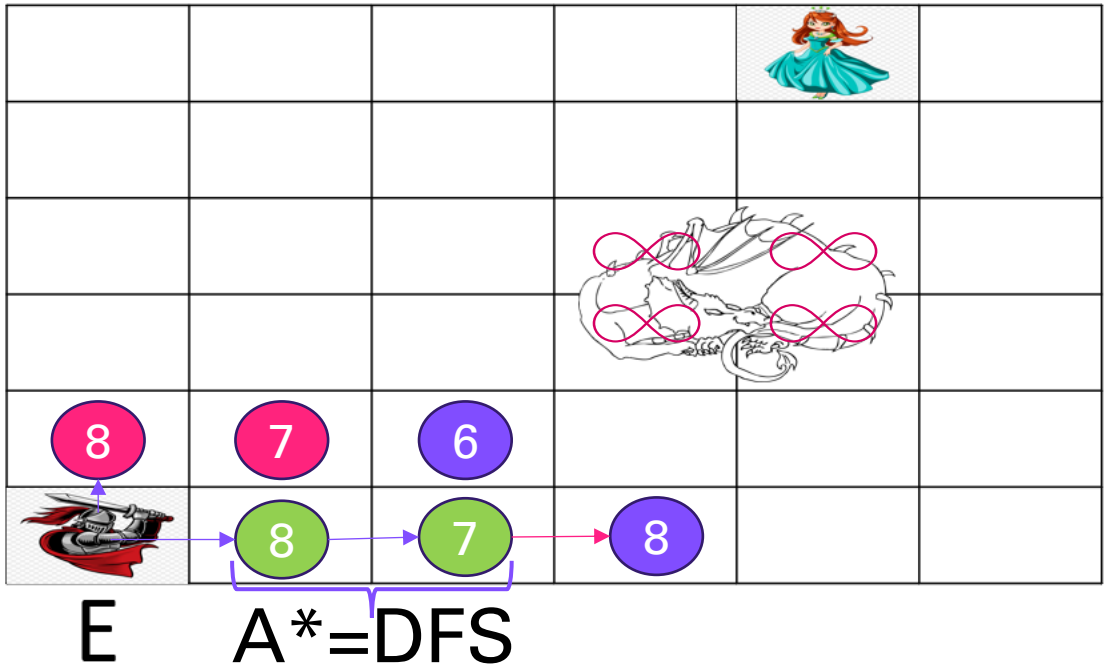
L'ALGORITMO DI A*

```
astar(Start, End, H_Function) -> Path, Cost {
    // Heuristic function definition
    g(Start) is 0,
    h(Start) is H_Function(Start, End),
    f(Start) is g(Start) + h(Start),
    /* Search over active successors/states
        One successor is a 5-pil:
        - current node
        - current depth (in the exploration tree)
        - current incurrend cost, g()
        - current estimated cost f()
        - current solution (in reverse order)
    Starting from (Start, 0, 0, f, []), we search until
    we reach the End state (or the Frontier is empty) */
    S, Cost is search([(Start, 0, 0, f, [])], End),
    Path is reverse(S),
    return Path, Cost.
}
```



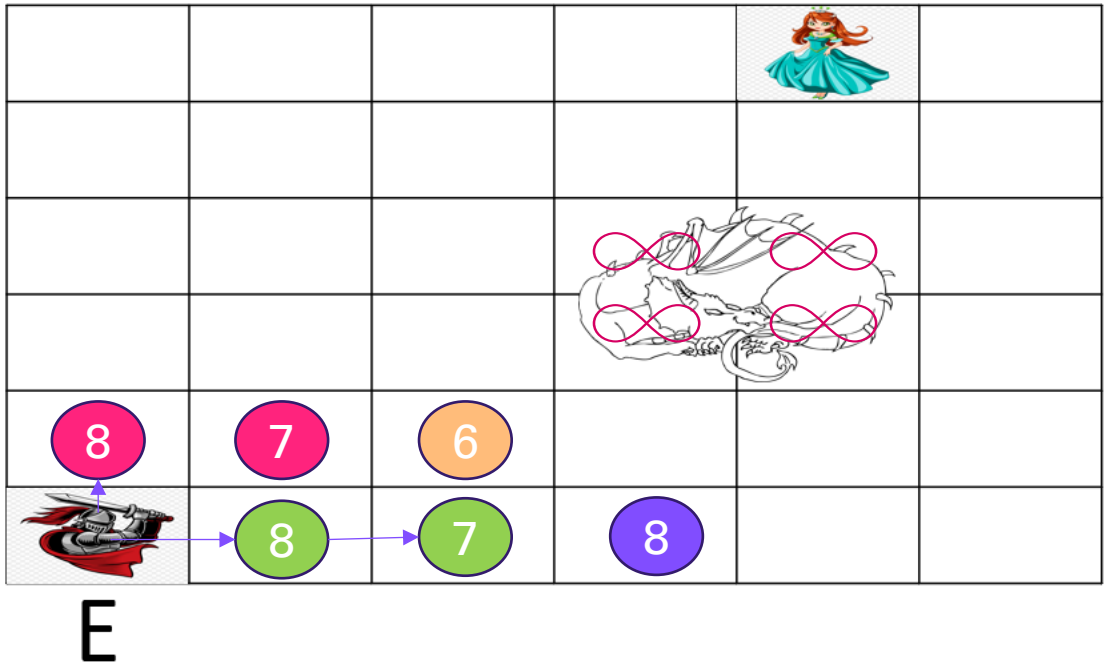
L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



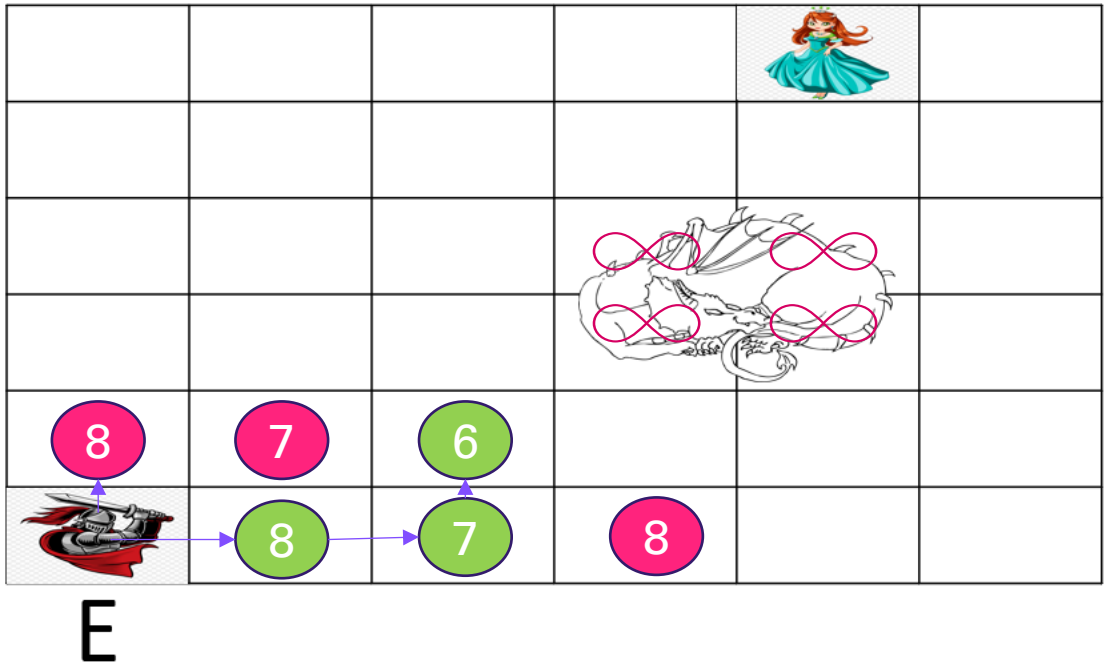
L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



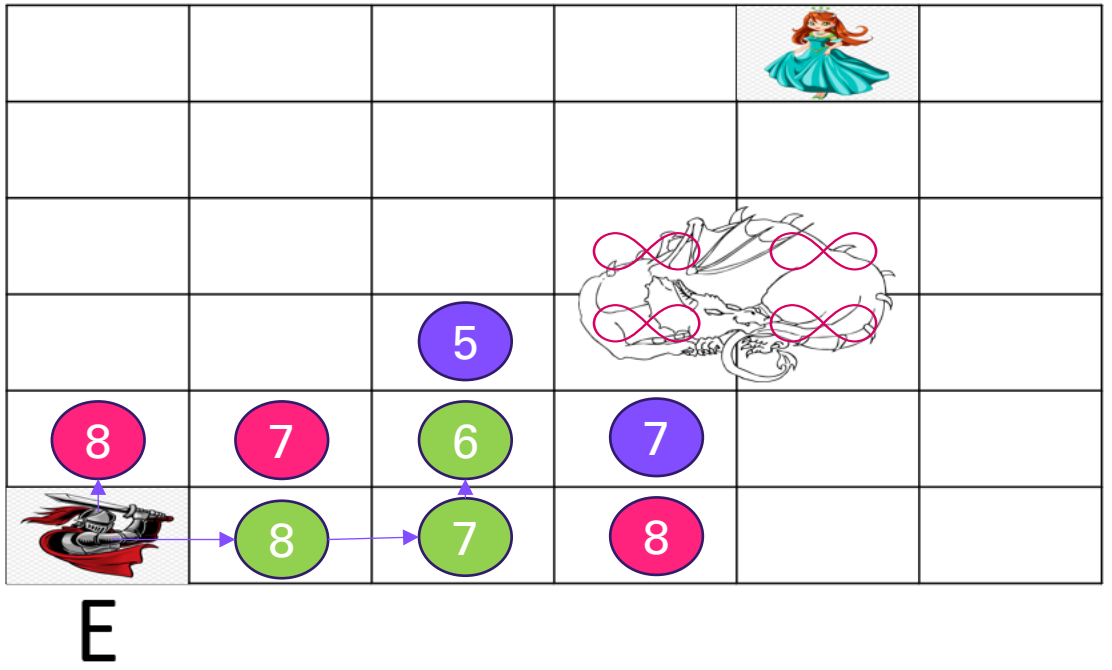
L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



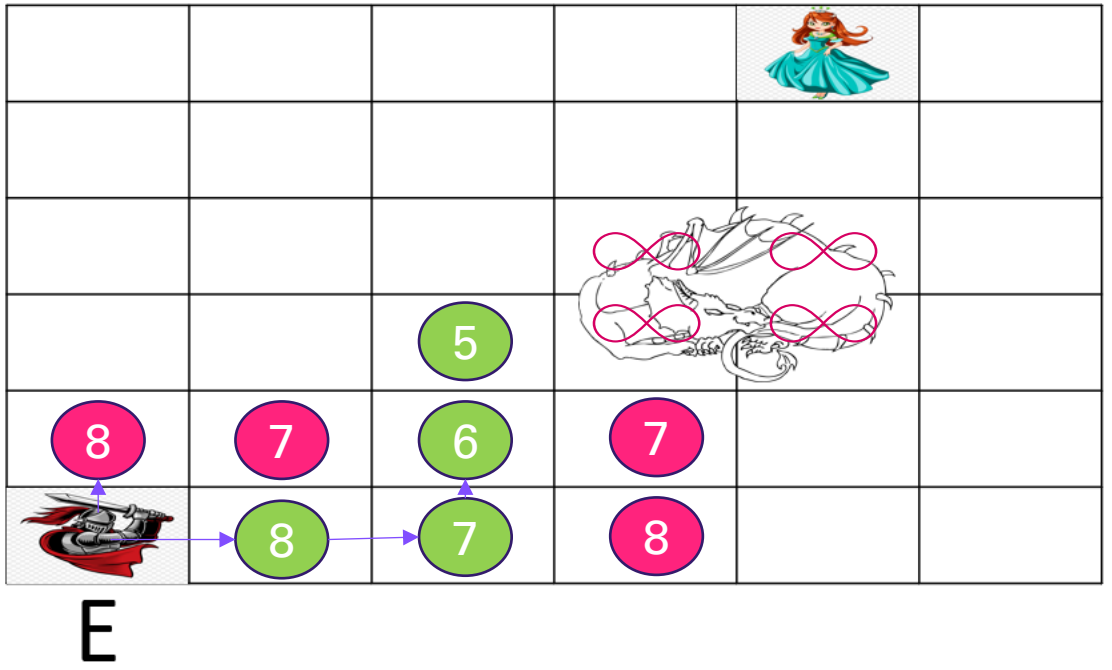
L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



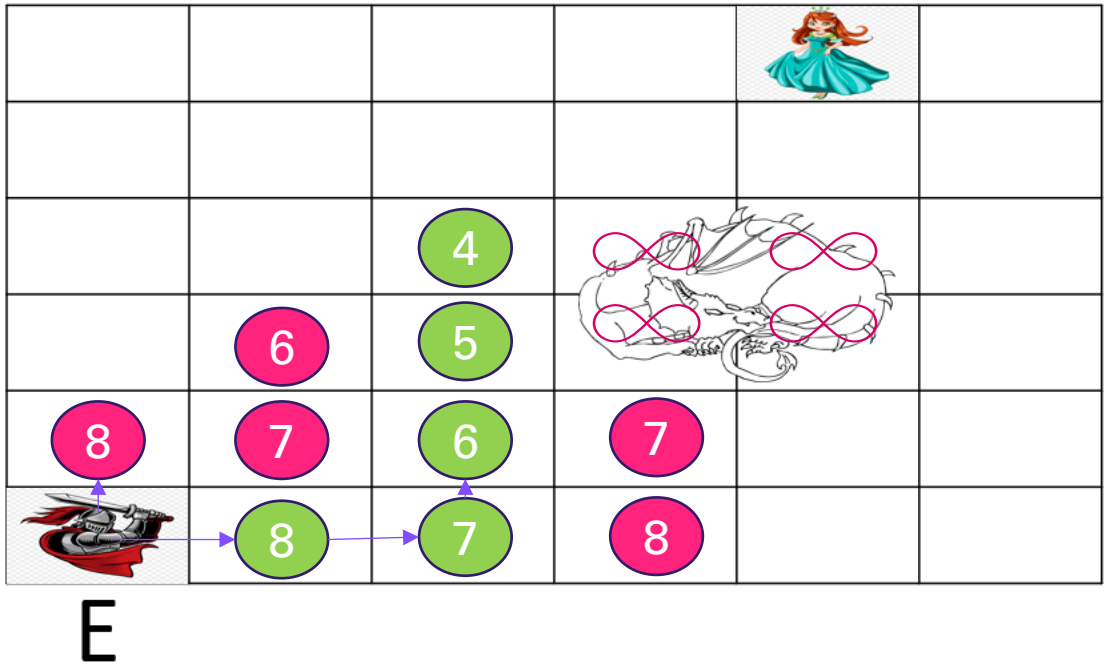
L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



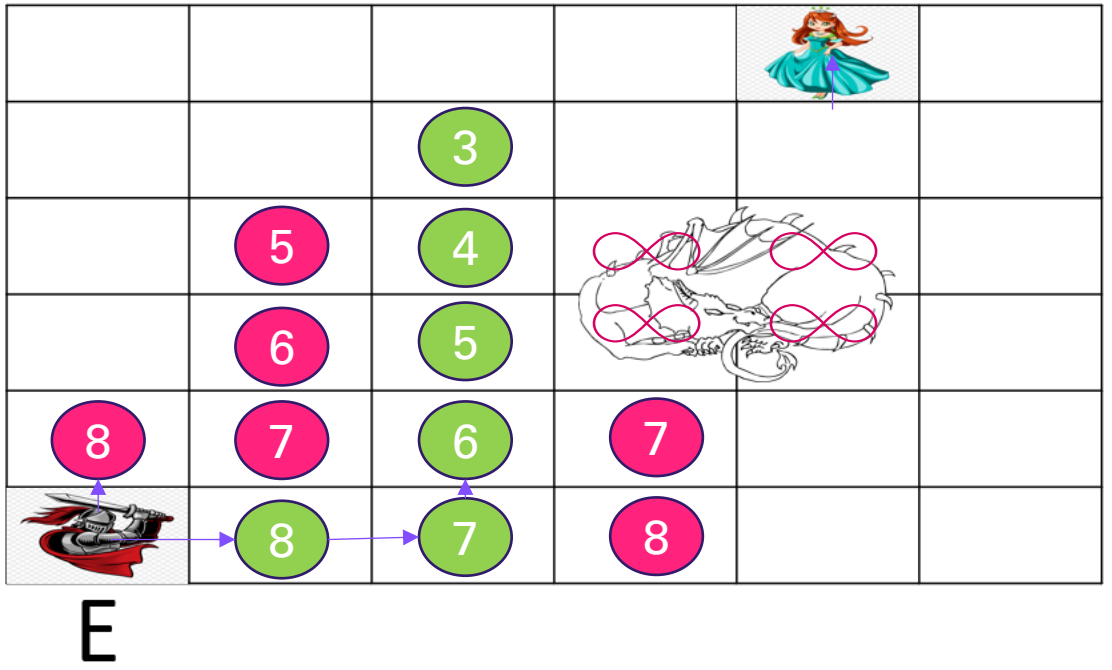
L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



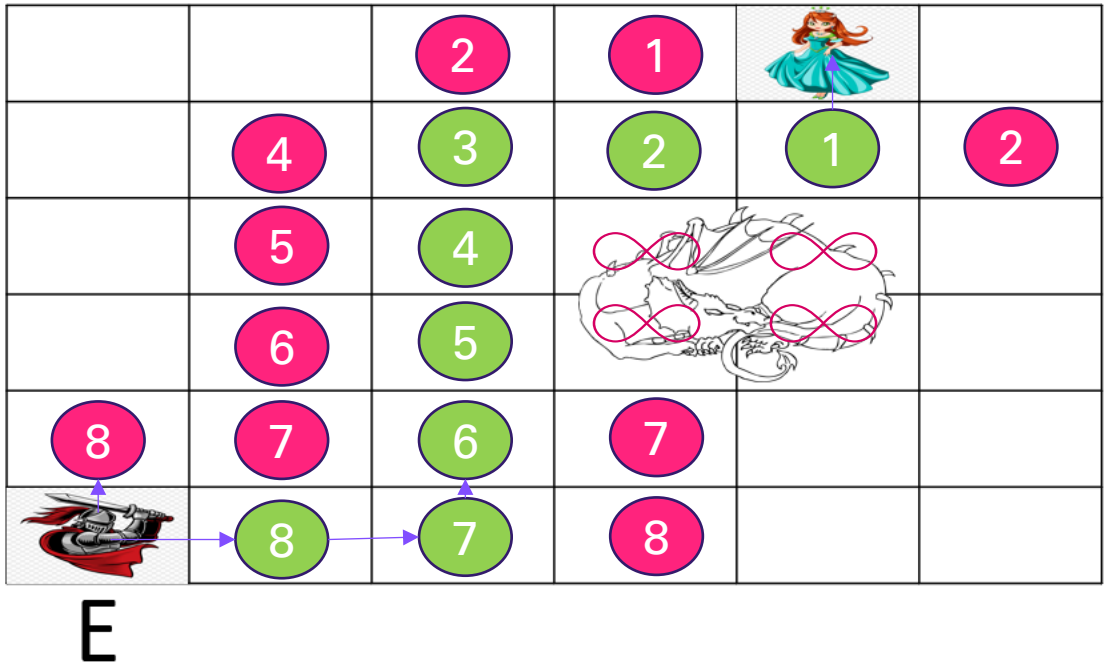
L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



L'IMPLEMENTAZIONE DELLA RICERCA

```
search([State | Frontier], End) -> Solution, Cost {
    Nodes_From_State is expand(State),
    Candidates is estimate(State, Nodes_From_State),
    New_Frontier is insert_ordered(Candidates, Frontier),
    Solution, Cost is search(New_Frontier, End),
    return Solution, Cost.
}
```



ESERCIZIO

Implementare il gioco del Cavaliere in Prolog, linguaggio logico sottoinsieme della FOL. In questo modo possiamo trovare la modellazione logica del problema!

What if ..?:

1. Le luci fossero spente e il cavaliere avesse una torcia che illumina le caselle adiacenti?
2. Il drago non fosse dormiente, ma cieco? E si muovesse casualmente di 1 casella per volta?
3. Cadessero casualmente dei massi dal soffitto della grotta con una probabilità non nulla ad ogni movimento?
4. La grotta fosse su due livelli e ci fossero pochi punti di sola andata da un livello all'altro?
5. Ci fossero delle trappole su alcune caselle che si attivano a pressione e il cavaliere avesse un potere speciale? Questo potere potrebbe dirgli se ci sono trappole adiacenti ma non dove!

Da consegnare via email (codice e pdf) all'indirizzo **hromei@ing.uniroma2.it**