**Università degli Studi di Torino**

Dipartimento di Informatica

Corsi di Laurea Magistrale in Informatica



**Relazione del progetto di Intelligenza**

**Artificiale e Laboratorio**

Salvatore Coluccia

a.a. 2020/2021

Indice

[1. Prolog 3](#_Toc34327020)

[1.1 Iterative Deepening 3](#_Toc34327021)

[1.2 IDA\* 4](#_Toc34327022)

[1.3 A\* 4](#_Toc34327023)

[1.4 Benchmark e considerazioni 5](#_Toc34327024)

[1.4.1 Labirinto 5](#_Toc34327025)

[1.4.2 Considerazioni 7](#_Toc34327026)

[2.ASP 8](#_Toc34327027)

[2.1 Calendario torneo sportivo 8](#_Toc34327028)

[2.2 Trasporto Aereo 9](#_Toc34327029)

# Prolog

Per la parte del progetto relativa al linguaggio Prolog ho implementato 3 algoritmi di ricerca: Iterative Deepening, A\* e IDA\*.

Iterative Deepening appartiene alla famiglia degli algoritmi di ricerca non informata e quindi non utilizza alcun tipo di conoscenza del problema al di là della definizione del problema stesso.

A\* e IDA\* appartengono invece alla famiglia degli algoritmi di ricerca informata che quindi utilizzano una conoscenza aggiuntiva (in questo caso un’euristica) in modo da espandere prima i nodi che soddisfano determinate condizioni.  
Ho utilizzato due funzioni euristiche:

* Distanza di **Manhattan**: *L1(P1,P2) = |x1-x2|+|y1-y2|*
* Distanza di **Chebyshev**: *Linf=max(|x1-x2|+|y1-y2|)*

Entrambe soddisfano le condizioni di **ammissibilità** e **consistenza**.  
Un’euristica è ammissibile se non sbaglia mai per eccesso la stima del costo per arrivare all’obiettivo. Un’euristica è consistente (o monotona) se, per ogni nodo *n* e ogni successore *n0* di *n*, il costo stimato per raggiungere l’obiettivo partendo da *n* non è superiore al costo di passo per arrivare a *n0* sommato al costo stimato per andare da lì all’obiettivo.



***Figura 1.1:*** *La linea verde rappresenta la distanza euclidea mentre le altre indicano alcune possibili rappresentazioni geometriche della distanza di Manhattan (tutte equivalenti).*

Il dominio utilizzato per testare questi algoritmi è quello del ***labirinto***.

## Iterative Deepening

L’algoritmo di Iterative Deepening si basa sull’algoritmo di ricerca in profondità limitata. L’idea sulla quale si basa è quella di eseguire iterativamente l’algoritmo di ricerca in profondità limitata incrementando il limite ad ogni iterazione fino a trovare la soluzione (se esiste).

La ricerca a profondità limitata permette di non espandere rami infiniti che non portano ad alcuna soluzione in quanto al raggiungimento del limite imposto l’algoritmo termina di espandere altri nodi.

L’algoritmo di Iterative deepening è un algoritmo completo ed ottimo in quanto trova sempre una soluzione se esiste e la soluzione trovata è quella ottima.

Nel caso in cui non esista alcuna soluzione non è in grado di accorgersene perché aumenterebbe all’infinito il limite di profondità, non ho volutamente inserito un meccanismo di limitazione della ricerca proprio per verificare la non terminazione dell’algoritmo.

Nell’implementazione tengo traccia dei nodi già espansi per evitare di ripercorrere rami già percorsi e per evitare loop. Questo aumenta lievemente l’ammontare della memoria necessaria durante l’esecuzione dell’algoritmo ma riduce il tempo necessario ad eseguire l’intero algoritmo.

## IDA\*

IDA\* è un algoritmo di ricerca informata che implementa una variante dell’algoritmo di iterative deepening ma utilizzando una funzione euristica per scegliere la profondità massima dell’iterazione corrente:

*f(n) = g(n) + h(n)*

*g(n) = costo effettivo dal nodo iniziale al nodo n*

*h(n) = costo stimato dal nodo n al nodo finale. Ricordo che la stima è stata effettuata utilizzando l’euristica scelta*

si sceglie quindi il min(f(n)) tra tutti i nodi che hanno superato la soglia imposta nell’iterazione precedente.

Nell’implementazione attuata si tiene traccia dei nodi già visitati in modo da evitare di ripercorrere gli stessi path e quindi in modo da ottimizzare i tempi di esecuzione.

Una caratteristica di IDA\* è quella di richiedere meno memoria rispetto all’algoritmo A\* in quanto deve ricordare esclusivamente i nodi del path corrente (oltre a quelli già visitati nella specifica iterazione).

Nel caso del dominio del labirinto si è scelto di assegnare dei costi diversi ai movimenti tra le varie caselle. È stato definito un costo di default pari a 1 per tutti i movimenti che però cambia per i movimenti di specifiche caselle, questo va ad influenzare il valore che assume la funzione g e rende meno vantaggiose le soluzioni che passano da quelle caselle.

## A\*

A\* è un algoritmo di ricerca informata di tipo best-first e cioè che espande sempre il nodo più “vantaggioso” localmente.

Il termine “vantaggioso” è definito in base al valore della funzione euristica f(n) = g(n) + h(n) che è implementata come per l’algoritmo IDA\*.

L’algoritmo A\* è stato implementato utilizzando uno heap dove ad ogni nodo è assegnata la rispettiva priorità f(n). Ad ogni iterazione viene quindi espanso dallo heap il nodo con priorità più bassa fino a quando non viene soddisfatto lo stato goal.

Utilizzando questo tipo di struttura dati si ottiene un incremento prestazionale notevole (rispetto ad utilizzare una coda) in quanto si può utilizzare un operatore per il getMin di complessità O(1) e un operatore di inserimento di complessità O(log(n)), con una coda invece le complessità sarebbero state rispettivamente O(n) e O(1).

A\*, rispetto a IDA\*, richiede molta più memoria in quanto deve ricordare tutti i nodi della coda più quelli già esplorati (che non devono essere riespansi).

A\* è un algoritmo completo per domini finiti in quanto prima o poi la coda diventerà vuota e l’algoritmo terminerà negativamente

## Benchmark e considerazioni

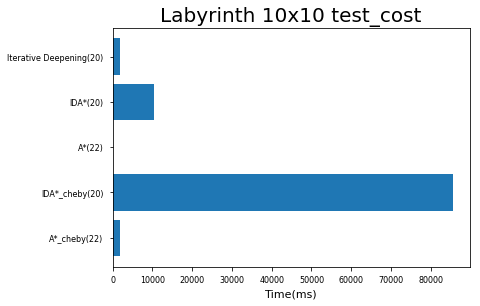
### 1.4.1 Labirinto

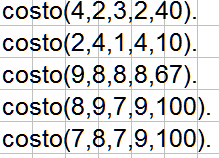








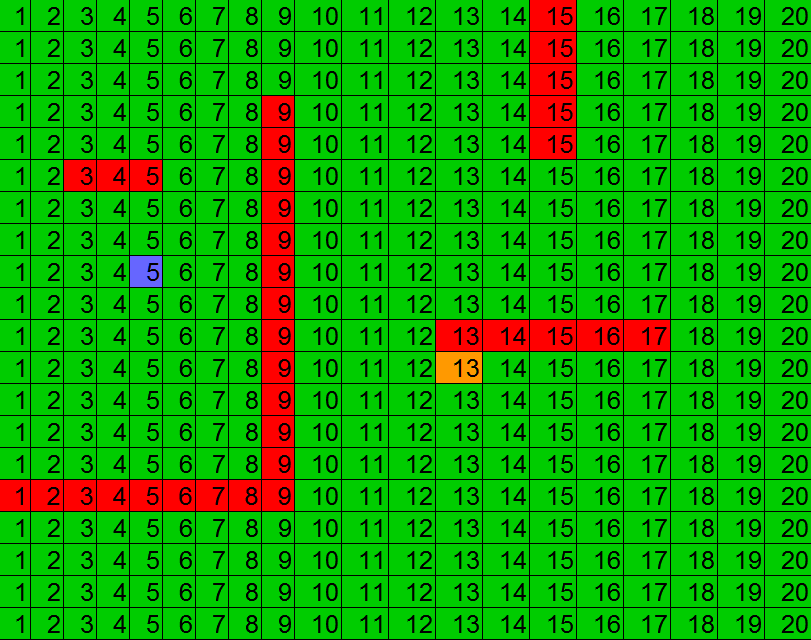
















### 1.4.2 Considerazioni

Dai test condotti possiamo subito osservare come gli algoritmi testati abbiano performance nettamente differenti in termini di tempo ma molto simili in termini di soluzione trovata.

Possiamo constatare come l’algoritmo di ricerca A\*, nonostante sia meno performante in termini di costo di memoria, risulta essere il più veloce in termini di tempo impiegato.

Si vede anche come sia quasi sempre l’algoritmo IDA\* quello con i tempi peggiori in quanto ogni volta deve rieffettuare la ricerca ripartendo dalla radice (ma con un taglio più profondo).

Possiamo anche notare che l’unico algoritmo che riesce sempre a terminare anche nella casistica senza soluzioni sia A\*.

Tra le due euristiche utilizzate quella che porta a risultati migliori è la distanza di Manhattan mentre con quella di Chebyshev i tempi si dilatano notevolmente.  
Questo era prevedibile poiché per IDA\* questa euristica porta ad un incremento del taglio più “lento” e quindi ad una dilatazione dei tempi totali, per A\*questo è spiegabile dal fatto ad esempio che le celle [0,0]-[10,10] e [0,0]-[1,10] hanno la stessa distanza anche se la cella [1,10] in realtà è più vicina alla [0,0] (considerando il numero minimo di celle da percorrere per arrivare alla cella [0,0]).

Notiamo anche come nel caso in cui ci siano costi differenti per le azioni effettuabili l’unico algoritmo di ricerca che ne “beneficia” è la ricerca A\* in quanto questo cambia la priorità nella coda delle azioni possibili. Nell’algoritmo IDA\* invece questo sostanzialmente non cambia nulla in quanto l’euristica serve per scegliere il nuovo taglio ma che è sempre scelto come il minimo tra quelli che superano il taglio attuale.

# 2.ASP

Per il progetto relativo alla parte sull’Answer Set Programming ho implementato una mia soluzione relativa al problema a vincoli per la programmazione del *“Master in Progettazione e Management del Multimedia per la Comunicazione”* .

Elenco qui le scelte progettuali che secondo me sono state più importanti:

1. Ho modellato i concetti con i seguenti termini:

|  |  |
| --- | --- |
| CONCETTO | IMPLEMENTAZIONE ASP |
| Insegnate | insegnante(X). |
| Insegnamento | insegnamento(Nome,Prof,OreTot). |
| Vincolo di propedeuticità | propedeutica(X,Y).  *Indica che X è propedeutico per Y* |
| Tempo | giornata(Num,Ore).  *Num indica il giorno; Ore può essere 8 o 5* |
| Lezione | lezione(Giornata,Professore,Insegnamento,Ora).  *La lezione di Insegnamento tenuta da Professore si svolgerà nella giornata numero Giornata nella Ora ora* |

1. Ho strutturato il problema cercando di seguire l’approccio **Generate and Test** caratterizzante del linguaggio ASP:
   1. Ho prima definito delle regole che generassero i termini *lezione(G,P,I,O)* vincolandole nella cardinalità in base al fatto che fosse un giorno da 8 ore oppure un giorno di al più 5 ore
   2. Successivamente ho definito tutti i constraint (regole senza testa) necessari per scartare i modelli inconsistenti con i requisiti imposti dal progetto.
2. Ho risolto i vincoli sulla cardinalità facendo largo uso della notazione *Min{<termine>:<dominio>}Max* per indicare il numero minimo e massimo di quei termini che potevano comparire nell’answer set in output.
3. Per implementare requisiti più complessi come quelli relativi al fatto che la prima lezione di una materia deve iniziare dopo le prime 4 ore di un’altra, ho utilizzato gli operatori *#min* e *#max* di Clingo. Grazie a questi sono riuscito a recuperare le giornate relative alla “prima lezione” e “ultima lezione” di uno specifico insegnamento e quindi ad utilizzarle per specificare i relativi vincoli.
4. **TEST**: Per convincermi del fatto che l’answer set generato in output fosse quello corretto (oltre che analizzare a mano quello che veniva stampato in console) ho implementato un tool in Python che tramite l’utilizzo della libreria Pandas controlla uno per uno tutti i vincoli rigidi richiesti nei requisiti. Questo mi ha permesso anche di individuare inconsistenze non evidenti ma che evidenziavano un errore di programmazione nella definizione dei vincoli.

Come possiamo notare nella tabella sottostante, misurando la CPU Time alterando solamente il numero di vincoli auspicabili attivi nella definizione del problema, il tempo totale di esecuzione varia discretamente solo nel caso in cui ci sono tutti i vincoli attivi (circa +2 secondi). Questo delta è riconducibile al fatto che il solver scarterà molti più modelli di quanti non ne scarta nella versione senza vincoli auspicabili.

|  |  |  |
| --- | --- | --- |
| Tutti i vincoli attivi | Vincoli auspicabili non attivi | Solo i primi 3 vincoli auspicabili |
| 4.375s | 2.797s | 2.766s |

# 3.Clips

Il progetto implementato tramite CLIPS è relativo alla definizione di un agente intelligente che sappia giocare al gioco della battaglia navale (secondo le regole e i vincoli specificati nel relativo documento).

## 3.1 Descrizione soluzione

L’approccio adottato è stato quello di cercare di definire delle regole che ricalcassero quanto più fedelmente possibile il ragionamento di un giocatore umano.

È stato quindi definito un nuovo template *deduced-cell* avente la stessa struttura del template *k-cell* e che è servito ad identificare le celle il cui contenuto è stato dedotto dall’agente tramite una GUESS.

Le mosse di tipo GUESS sono state utilizzate per tutti i casi in cui l’agente poteva dedurre con certezza il contenuto di una determinata cella.  
Ad esempio: se l’agente è a conoscenza che il contenuto della cella [0,0] è LEFT e quello della cella [0,2] è RIGHT, può effettuare una operazione di GUESS nella cella [0,1].

Le mosse di tipo FIRE sono invece state utilizzate non solo per casistiche certe, ma anche per casi in cui era solo probabile che ci fosse una casella piena.  
Nello specifico, le mosse di tipo FIRE sono state utilizzate anche nel caso in cui l’agente non abbia, in un determinato momento, una conoscenza tale da permettergli di prendere una decisione più sensata di una casuale: in tal caso (ad esempio nei test senza conoscenza iniziale) l’agente sceglie una casella casuale facendosi “influenzare” esclusivamente dai fatti di tipo *k-per-row* e *k-per-col* (regola: *random-fire-smart*)

Usando le mosse di tipo FIRE nei casi di incertezza, come quello più estremo relativo a nessuna conoscenza iniziale, permette eventualmente di scoprire nuovi fatti che possono portare l’agente a poter effettuare altre azioni di GUESS o FIRE più intelligenti.

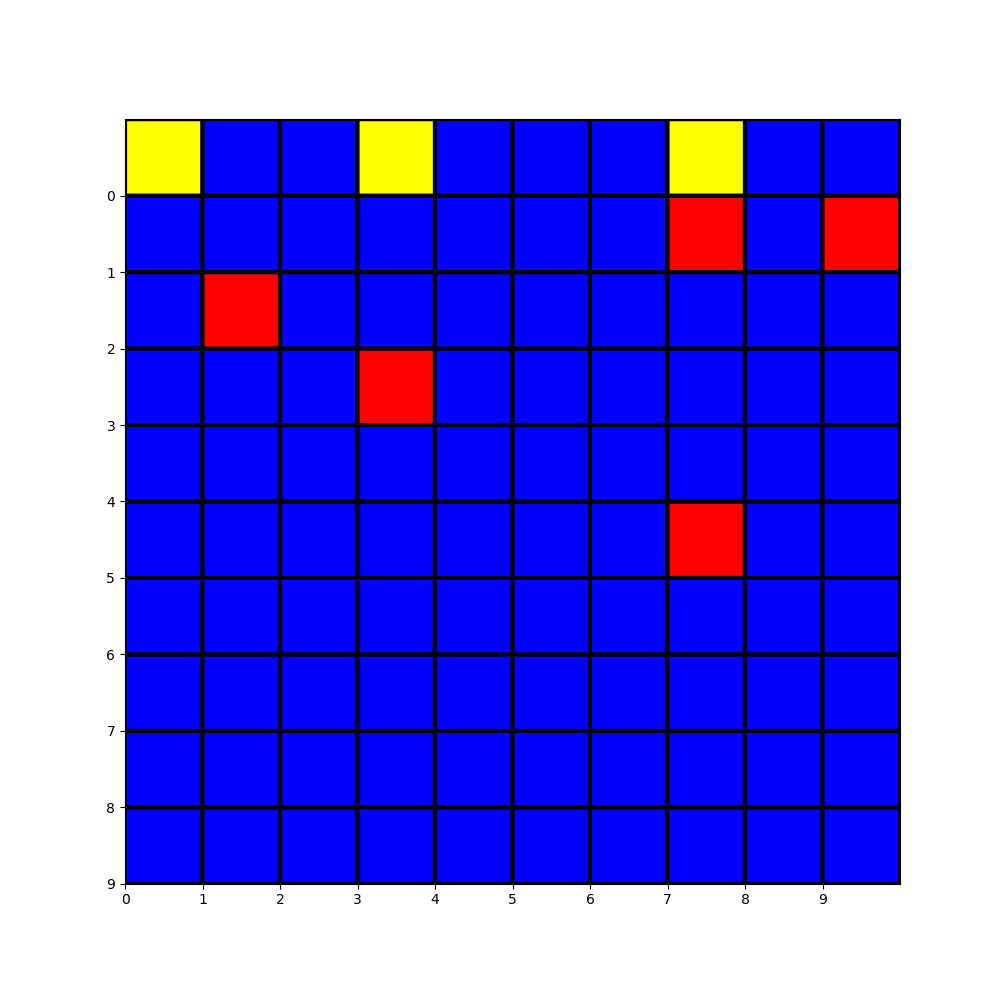
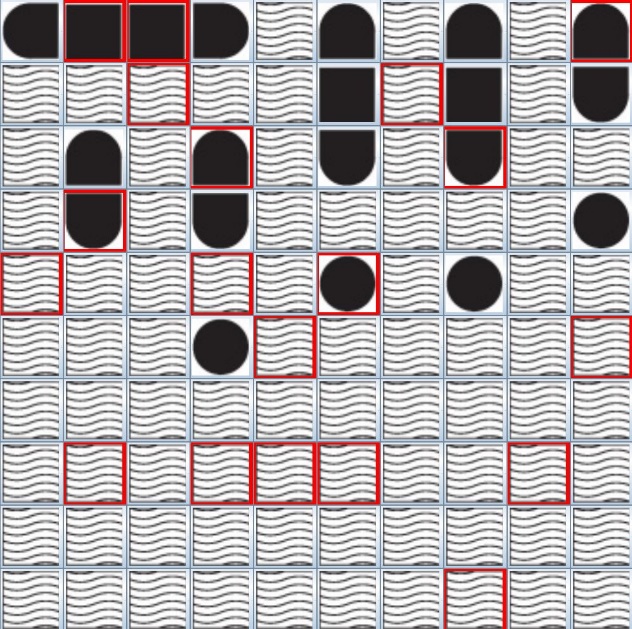
Una critica che si potrebbe avanzare è che una fire-ko incide molto di più di una guess-ko e quindi magari avrebbe più senso usare le fire in caso di certezza e usare le guess in caso di incertezza. Questa sarebbe un’osservazione perfettamente sensata anche se bisognerebbe valutare quanto incide il fatto che una guess non ci permette effettivamente di vedere cosa c’è su una casella e quindi non aggiunge una conoscenza reale del mondo.

## 3.2 Test e risultati

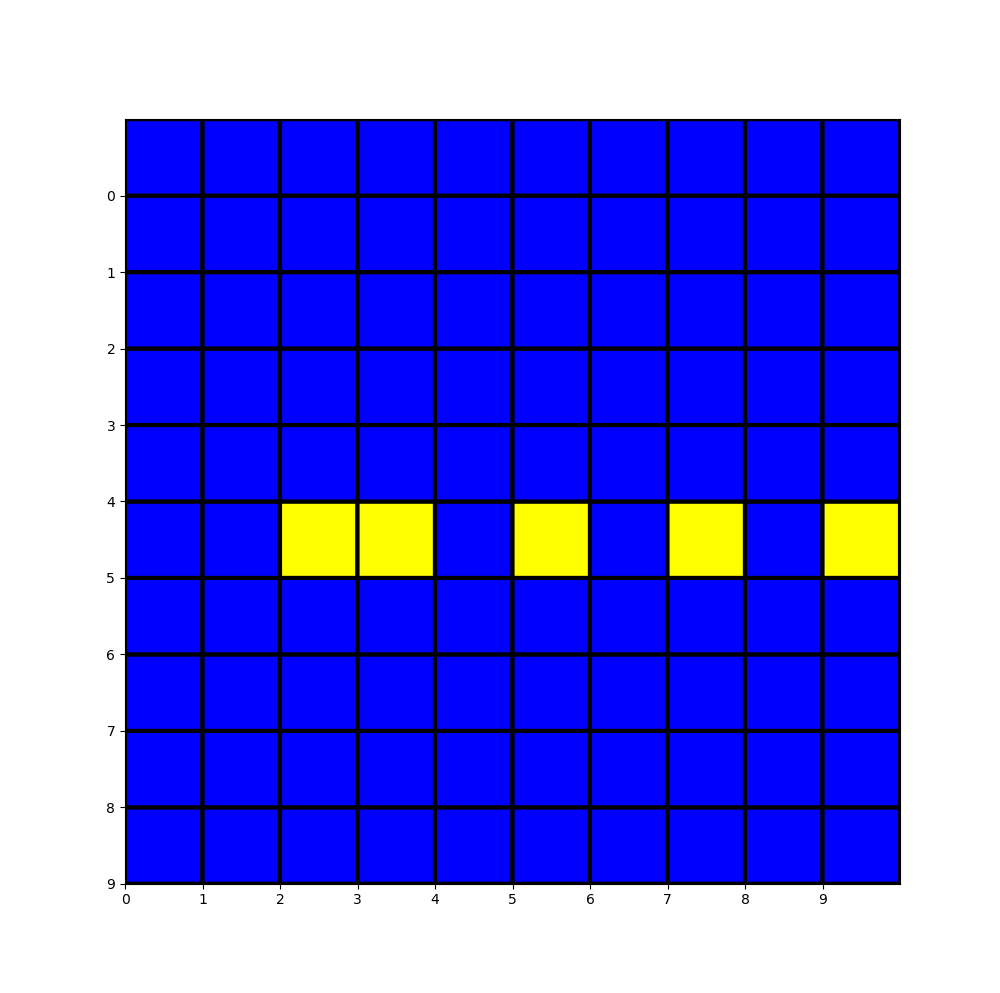
Sono stati effettuati dei test su due mappe diverse e con versioni aventi della conoscenza iniziale o meno.  
Per ogni caso vengono riportate le statistiche finali.

Le celle di colore giallo rappresentano le guess, quelle di colore rosso rappresentano le fire.

### 3.2.1 Test A



**SCORE: 45 FIRE\_OK:4 FIRE\_KO:1 GUESS\_OK:3 GUESS\_KO:0 SAFE:6 SINK:4**

Versione senza conoscenza iniziale

**SCORE: -265**

**FIRE\_OK: 1**

**FIRE\_KO: 4**

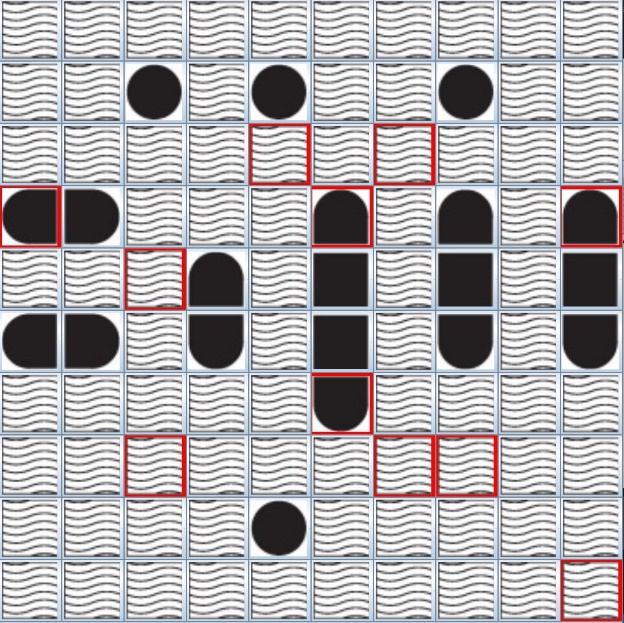
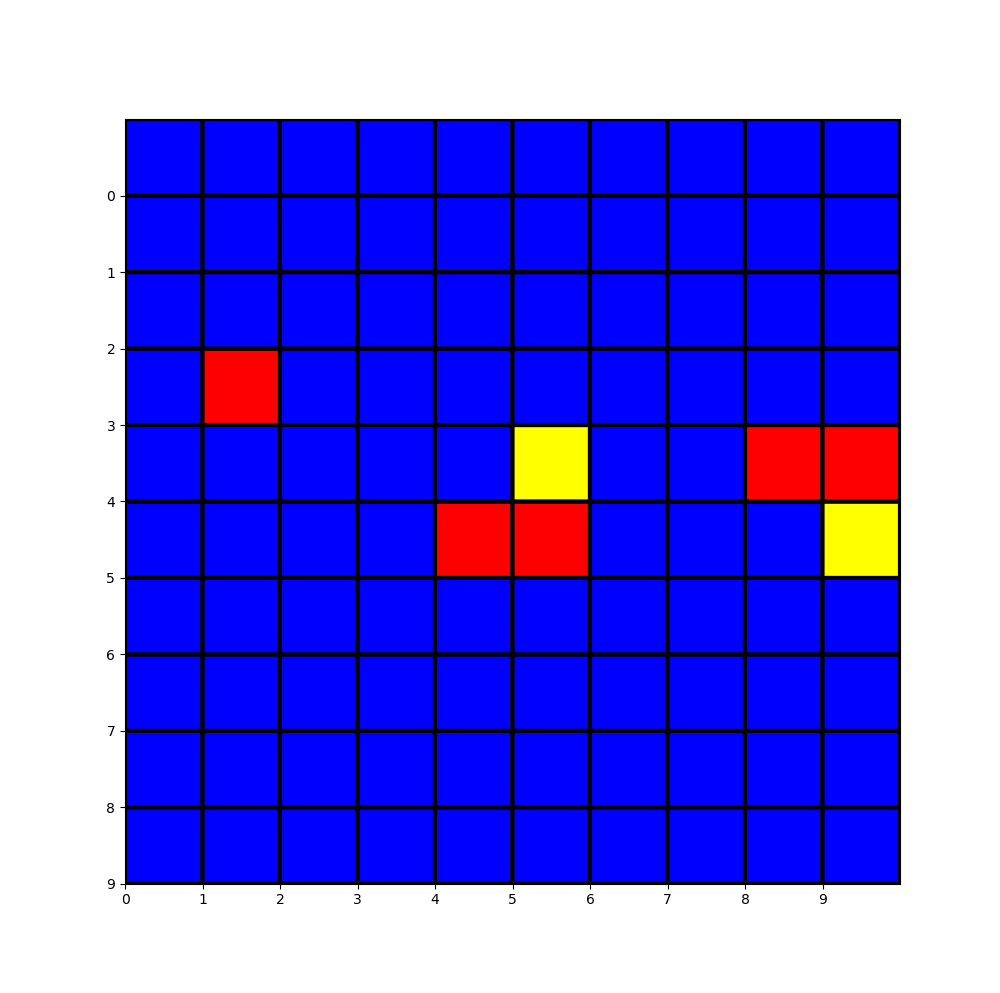
**GUESS\_OK: 0**

**GUESS\_KO: 0**

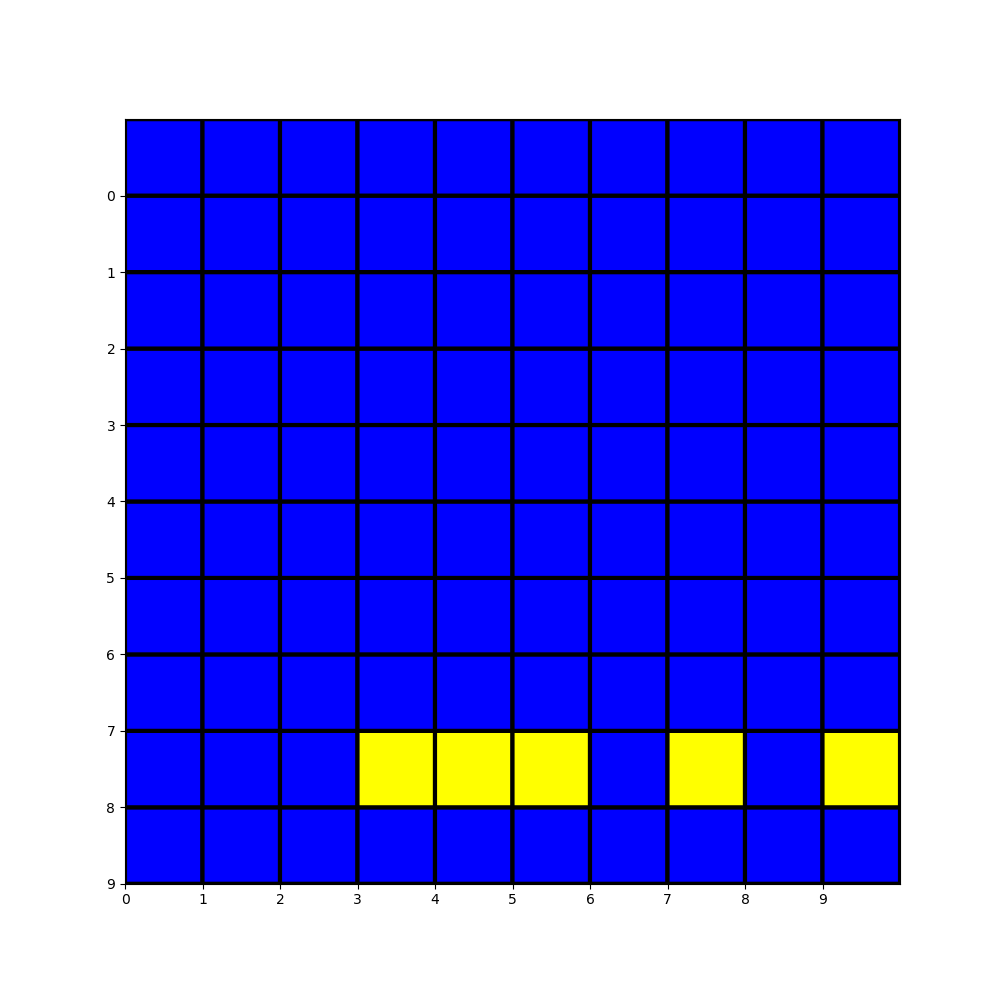
**SAFE: 19**

**SINK: 1**

### 3.2.2 Test B

**SCORE: -95 FIRE\_OK:3 FIRE\_KO:2 GUESS\_OK:2 GUESS\_KO:0 SAFE:11 SINK:1**



Versione senza conoscenza iniziale

**SCORE: -265**

**FIRE\_OK: 1**

**FIRE\_KO: 4**

**GUESS\_OK: 0**

**GUESS\_KO: 0**

**SAFE: 19**

**SINK: 1**