Progetto di Intelligenza Artificiale
Laboratorio

Modulo Planning e Sistemi a Regole
a.a. 2019/2020

Aldo Bushaj 847091
Antonino Bushaj 847013
Silvestro Stefano Frisullo 832813

INTRODUZIONE

Il programma è stato implementato seguendo una logica modulare in cui il file main_agent.clp controlla il passaggio da un modulo all'altro in maniera ciclica.
Per via della formula con cui il punteggio viene calcolato, ovvero il fatto che venga sottratto al punteggio per un'azione di guess nella posizione sbagliata più di quello che viene assegnato per un'azione di guess corretta, abbiamo scelto di implementare il progetto non come una procedura guidata agli obiettivi ma più deterministica.
Dai test è emerso che il programma ottiene un punteggio alto quando ha una background knowledge di alcuni blocchi, mentre ha un punteggio negativo la conoscenza iniziale è di zero blocchi.

MODELLAZIONE DELLA CONOSCENZA

Il seguente fatto

```
(deftemplate closed-boats
(slot one (default 0)(range 0 4))
(slot two (default 0)(range 0 3))
(slot three (default 0)(range 0 2))
(slot four (default 0)(range 0 1))
```

è usato per tenere traccia di quante navi "complete" il sistema crede di aver trovato e nel caso in cui queste siano 4,3,2,1 (quindi se le ha trovate tutte) allora termina.
Dei seguenti template invece

```
(deftemplate to-guess
(slot x) (slot y)
)
(deftemplate guessed (slot x)
(slot y)
)
```

il primo **to-guess** attiva la regola **guesser**

```
(defrule guesser (declare (salience 2))
?f <- (to-guess(x ?x)(y ?y))
(not (guessed(x ?x)(y ?y)))
?r <- (g-per-row (num ?x)(val ?rv))
?c <- (g-per-col (num ?y)(val ?cv))
=>
(if (and (< ?x 10)(< ?y 10)(>= ?y 0)(>= ?x 0))then (assert(guessed (x ?x)(y ?y)))
(modify ?r (val (+ ?rv 1)))
(modify ?c (val (+ ?cv 1)))
)
(retract ?f)
)
```

La logica è che conoscendo come assunzione iniziale che le navi sono distanti almeno una casella le une dalle altre, alla scoperta di una nave asseriamo direttamente come (to-guess(x)(y)) le caselle adiacenti, attivando la regola guesser. Quest'ultima ha il compito di marcare le caselle come guessed controllando però che per ognuna di queste caselle non sia già presente in working memory un fatto guessed e che la loro posizione sia valida.

Per quanto riguarda le caselle contenenti pezzi delle navi su cui è già stata eseguita un'azione di guess viene usato il fatto **guessed**.

I seguenti template

```
(deftemplate g-per-col
(slot num (range 0 9))
(slot val (default 0)(range 0 9))
)
(deftemplate g-per-row (slot num (range 0 9))
(slot val (default 0)(range 0 9))
)
```

memorizzano le caselle marcate come guessed rispettivamente per colonna e per riga.

Il seguente template

```
(deftemplate ausiliar-exec
(slot action) (slot x)
(slot y)
)
```

È usato quando è necessario eseguire più azioni all'interno dello stesso step.
Per esempio, quando viene riconosciuta una nave da 3 k-cell dovremmo fare 2 guess nello stesso passo che quindi non verrebbero conteggiate in *moves* creando inconsistenze ed errori, ma grazie ad ausiliar-exec si pone la prima guess nel fatto exec quella successiva in un fatto ausiliar-exec che quando il focus sarà su AGENT attiverà la regola seguente

```
(defrule to-ausiliar-exec (declare (salience 3))
?f <- (ausiliar-
exec (action ?a) (x ?x)(y ?y)) (status (step ?s)(currently running))
=>
(assert(exec (step ?s) (action guess) (x ?x) (y ?y ))) (retract ?f)
(assert(return))
(pop-focus)
)
```

Questa regola “trasforma” una ausiliar-exec in una exec, garantendo che venga eseguita solamente una azione per ogni passo.
Infine sono stati creati una serie di template che hanno lo scopo di far attivare due regole in sequenza.

STRUTTURA

Per quanto riguarda la struttura del programma il tutto è stato pensato come una serie di fasi eseguite in sequenza ed ordinate in base a specificità ed importanza.
Il file agent.clp contiene i template e le regole di controllo oltre ad avere il compito di alternare l’esecuzione dei file.

- **main_control.clp**
  Innanzi tutto, questo file contiene una regola **end** che servirà al momento della terminazione del programma ossia nel caso in cui esso riesca a trovare tutte le navi prima di esaurire le fire a sua disposizione.
  Esso contiene inoltre le regole to-ausiliar-exec, guesser già citate precedentemente e garbage-w-g che elimina i fatti to-guess fuori dalla mappa o già precedentemente visualizzati e quindi etichettati come **guessed**.

- **after_no_know.clp**
  Qui vi sono le regole che saranno attivate dopo l’esecuzione di una fire da parte di no_know.clp o l’asserzione di una move-x o move-y. Se la fire ha successo verranno cancellati alcuni fatti di controllo dalla WM altrimenti nel caso in cui la fire non abbia avuto successo o siano stati asseriti dei fatti di controllo move-x o move-y poiché il punto in cui si sta lavorando è già stato visitato, allora verrà calcolato il nuovo punto in cui effettuare una fire ed eseguirla.

- **know_double_middle.clp:**
  Le regole qui contenute verranno attivate quando la knowledge base presenta due k-cell contenenti middle adiacenti quindi siamo nel caso della nave con dimensione di quattro caselle.

- **know_two.clp:**
  Questo contiene invece tutte le regole in grado di riconoscere una nave note due k-cell ad essa appartenenti.

- **know_one.clp**
  Le regole qui contenute verranno attivate quando siamo a conoscenza di una singola parte di nave (una k-cell) che però è un estremo, quindi un top, bottom, left e right e si può

dedurre facilmente la grandezza della nave e la sua posizione senza dover effettuare la fire

- **know_one_indecision.clp**
  Questo invece è il caso simmetrico al precedente in cui siamo a conoscenza di una singola parte di nave (k-cell) anche in questo caso un estremo della nave ma qui non è possibile dedurre la forma e la posizione della nave, pertanto è necessaria la fire.

- **know_middle.clp**
  Qui sono contenute le regole che si attivano quando siamo a conoscenza di una singola k-cell contenente middle ma non sappiamo né orientamento né grandezza della nave.

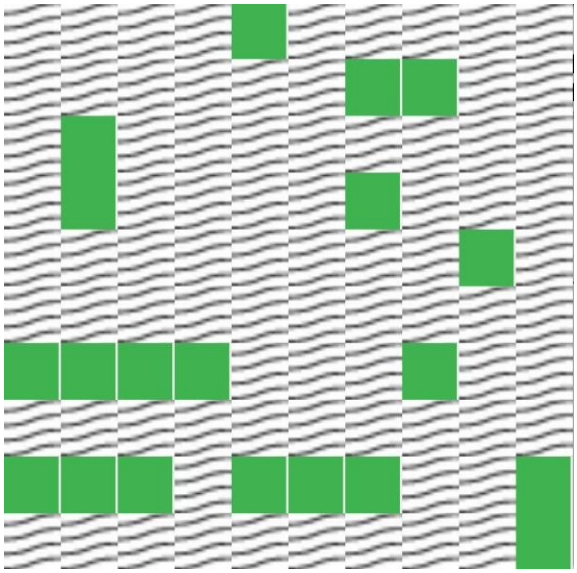
- **no_know.clp**
  Questo è il caso in cui non sappiamo niente della nostra mappa quindi le regole qui presenti verranno attivate nel momento in cui non è nota nessuna k-cell ed il numero delle azioni di fire è maggiore di 0.
  Il programma cerca la riga e la colonna con il massimo numero di k-cell, se la casella che rappresenta la loro intersezione non è ancora stata marcata come guessed viene eseguita una fire, altrimenti controlla se il numero di caselle contenenti k-cell nella riga (k-per-row) è maggiore del numero di caselle contenenti k-cell nella colonna (k-per-row) e asserisce il fatto move-x o move-y a seconda del risultato.

RISCONTRO GRAFICO

È stato implementato un programma java che permette, una volta eseguito il programma clips e la serie di istruzioni successiva, di vedere una mappa identica a quella usata in input contenente caselle verdi rappresentanti le k-cell correttamente trovate e caselle rosse per le mancanti. In questo modo si può avere un riscontro grafico del risultato del programma.

```
(and
(dribble-on /pathname) (facts AGENT)
(dribble-off)
)
```

Si ottiene un risultato di questo tipo:



TEST

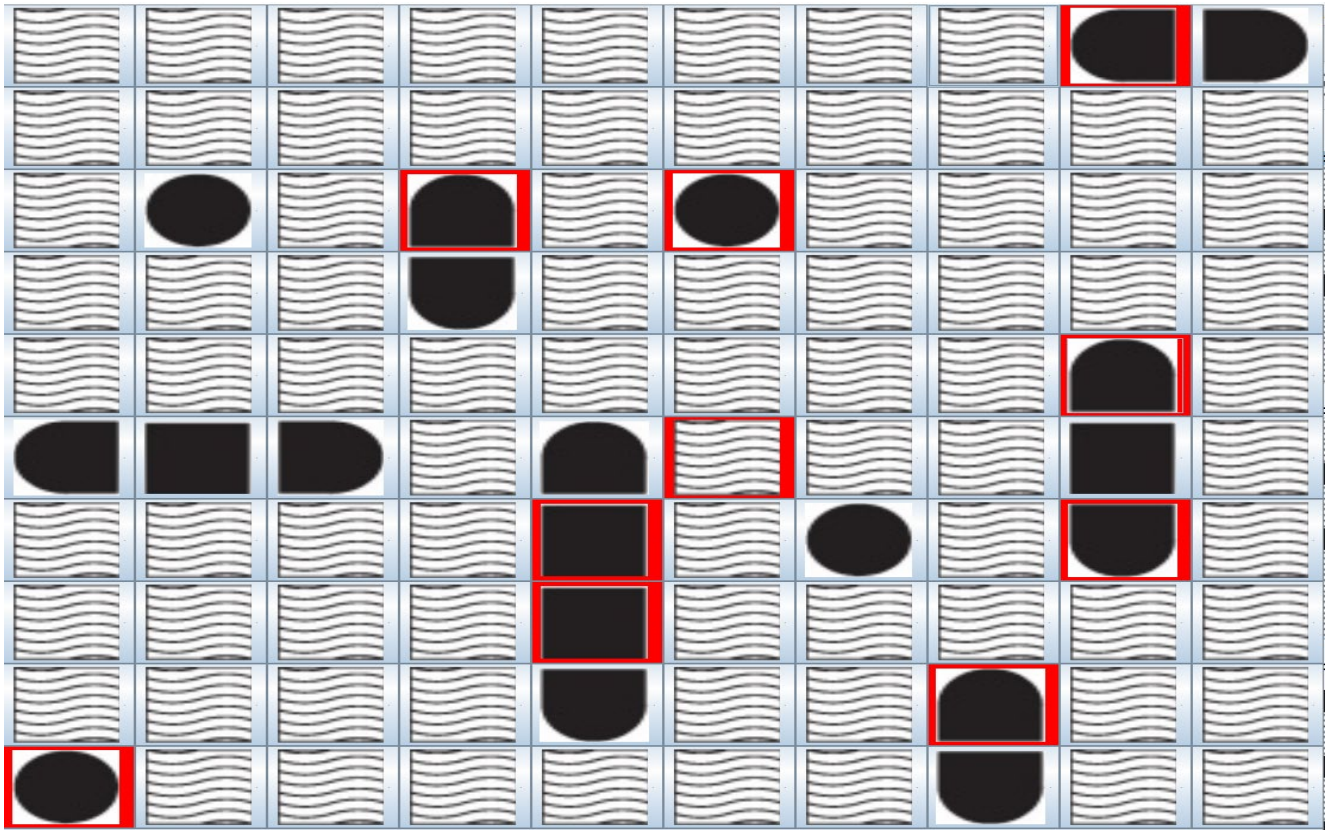
Il programma è stato testato su 5 diverse mappe e per ogni mappa è stato testato con 3 livelli di conoscenza, non conoscendo nulla, conoscendo 5 e 10 caselle e questo è stato il risultato.

Mappa numero	Celle conosciute	Navi trovate (segnate per dimensione)	Score
1	0	(2/4)4-3	-110
1	5	4-3-3-2-2-1-1	175
1	10	4-3-3-2-2-2-1-1-1-1	260
2	0	4-3-(2/3)3	10
2	5	4-3-3-2-1	105
2	10	4-3-3-2-2-2-1-1-1-1	270
3	0	4-2-1	-40
3	5	4-3-3-2-1	70
3	10	4-3-3-2-2-2-1-1-1-1	245
4	0	4-2-2	-20
4	5	4-3-2-2-2-1	115
4	10	4-3-3-2-2-2-1-1-1-1	270
5	0	4-3	-30
5	5	4-3-3-2-2-1-1	175
5	10	4-3-3-2-2-2-1-1-1-1	260

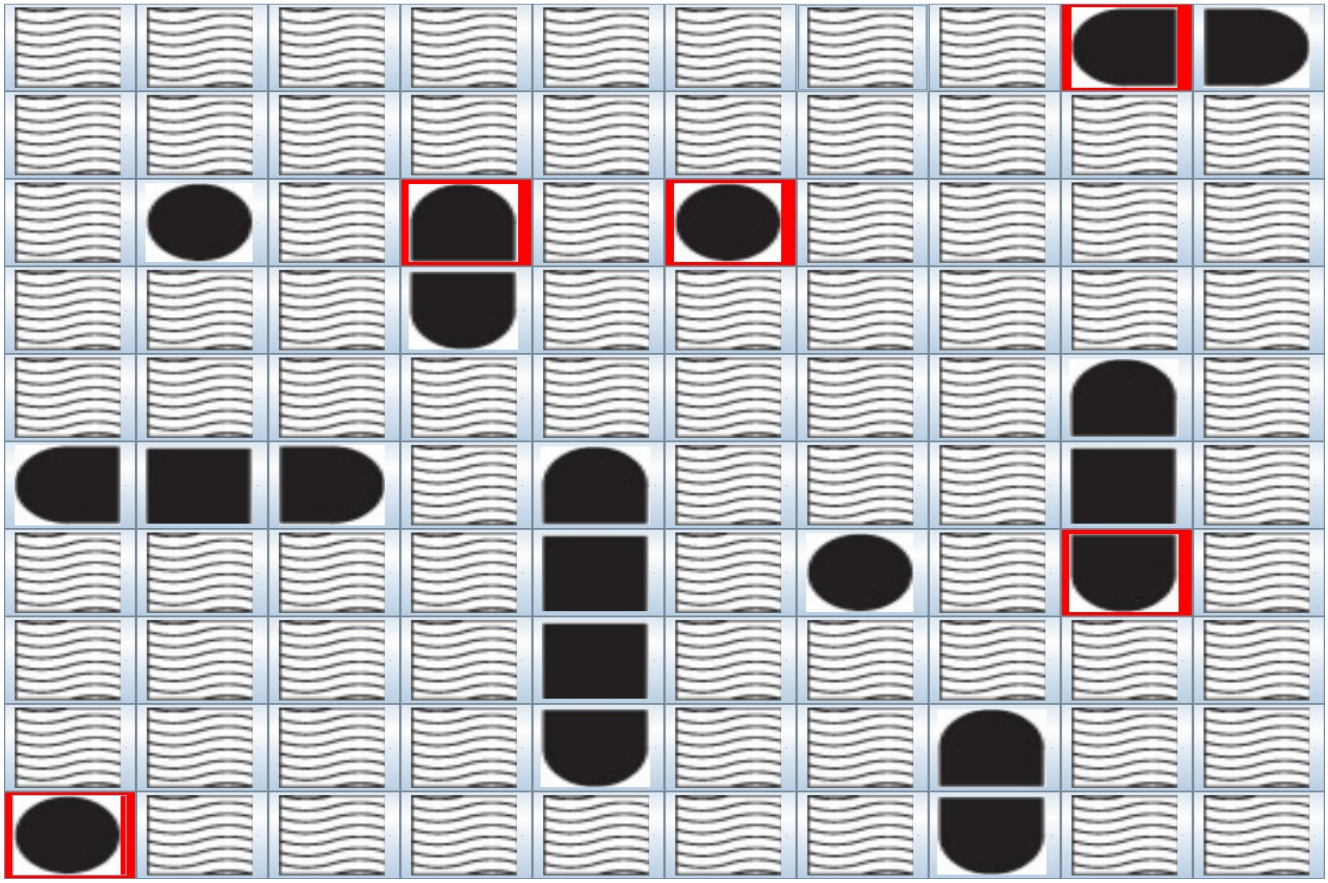
Le celle conosciute comprendono anche l'acqua che non da un vantaggio al programma paragonabile alla conoscenza di una k-cell.

MAPPE

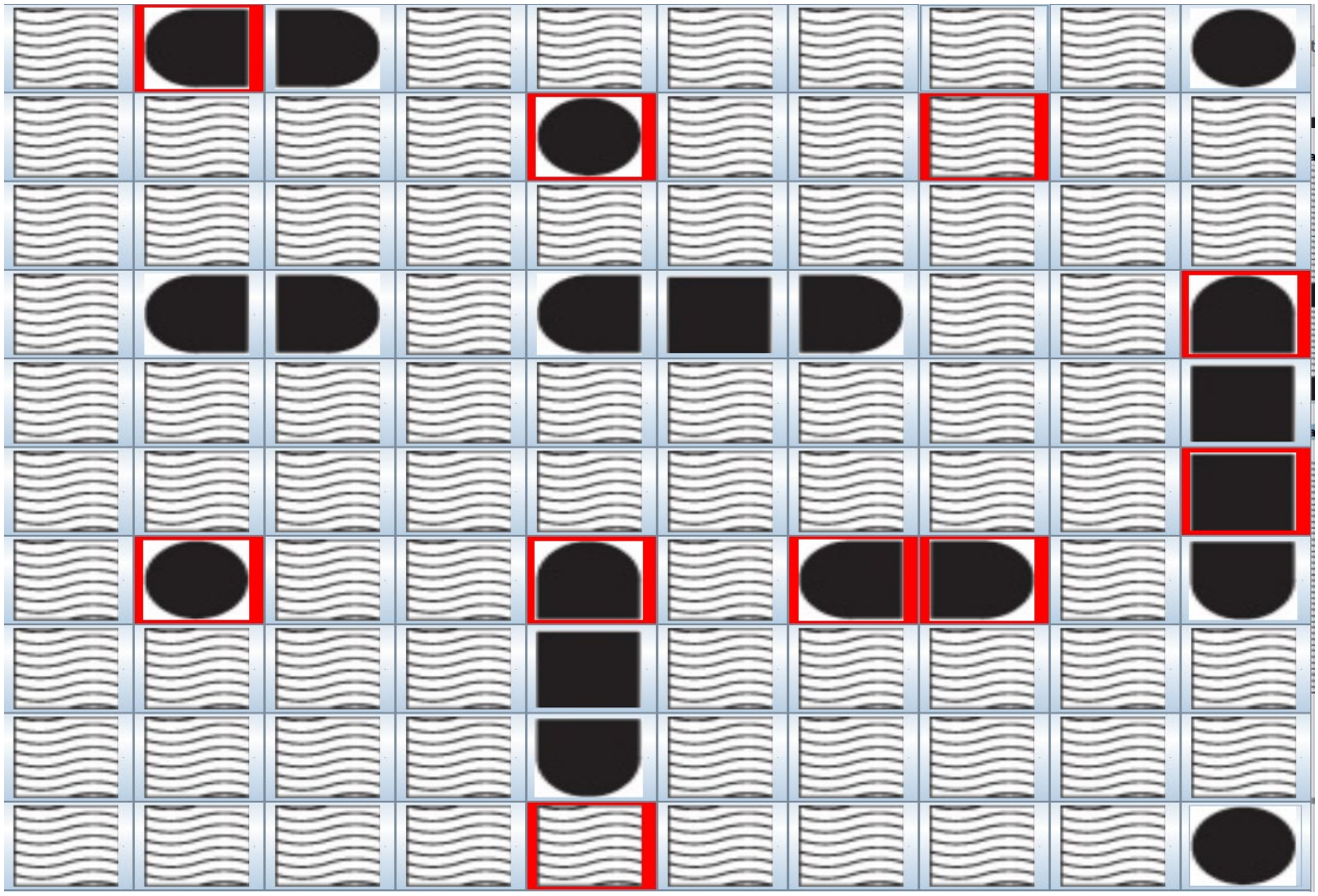
mappa 1 con conoscenza di 10 blocchi:



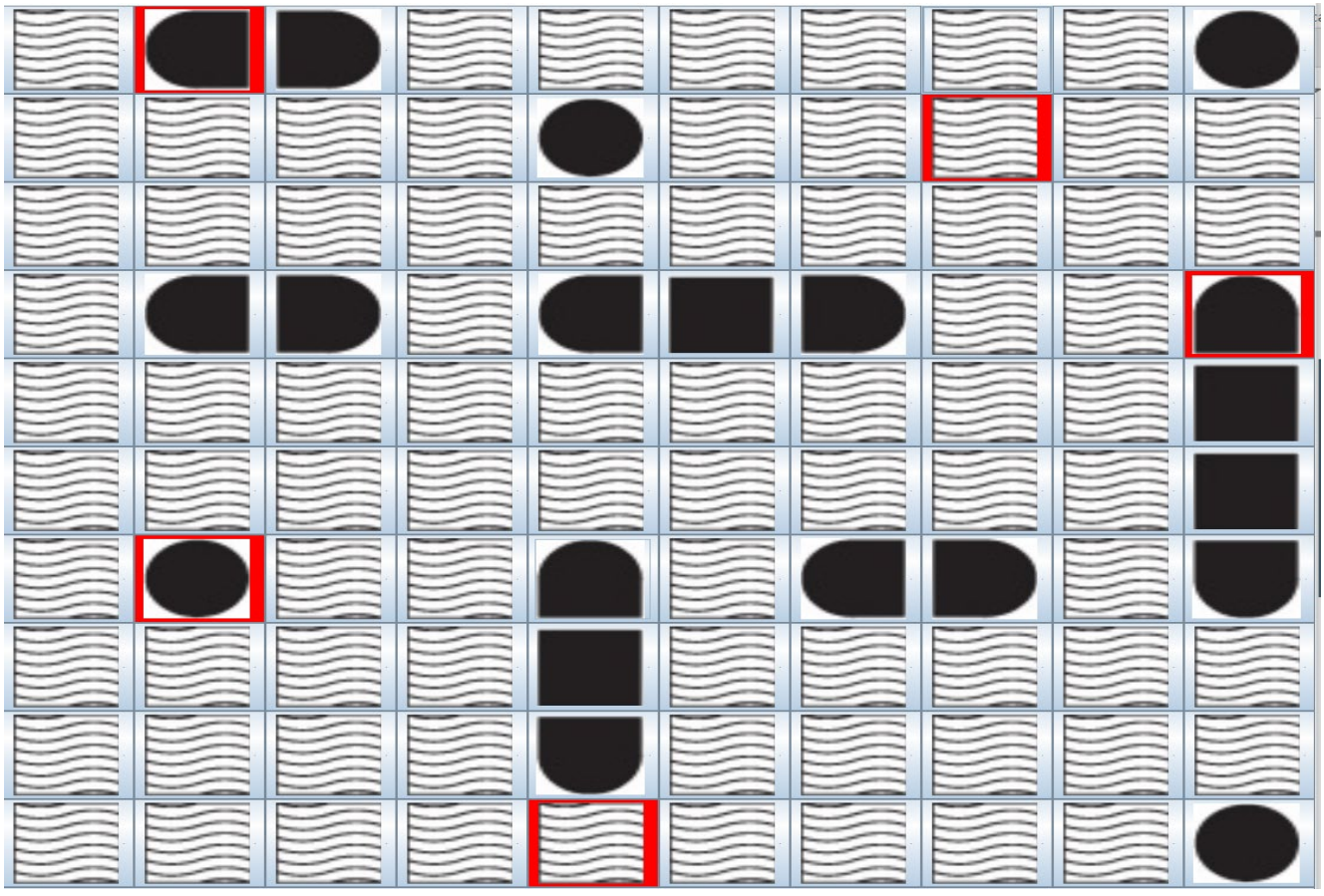
mappa 1 con conoscenza di 5 blocchi:



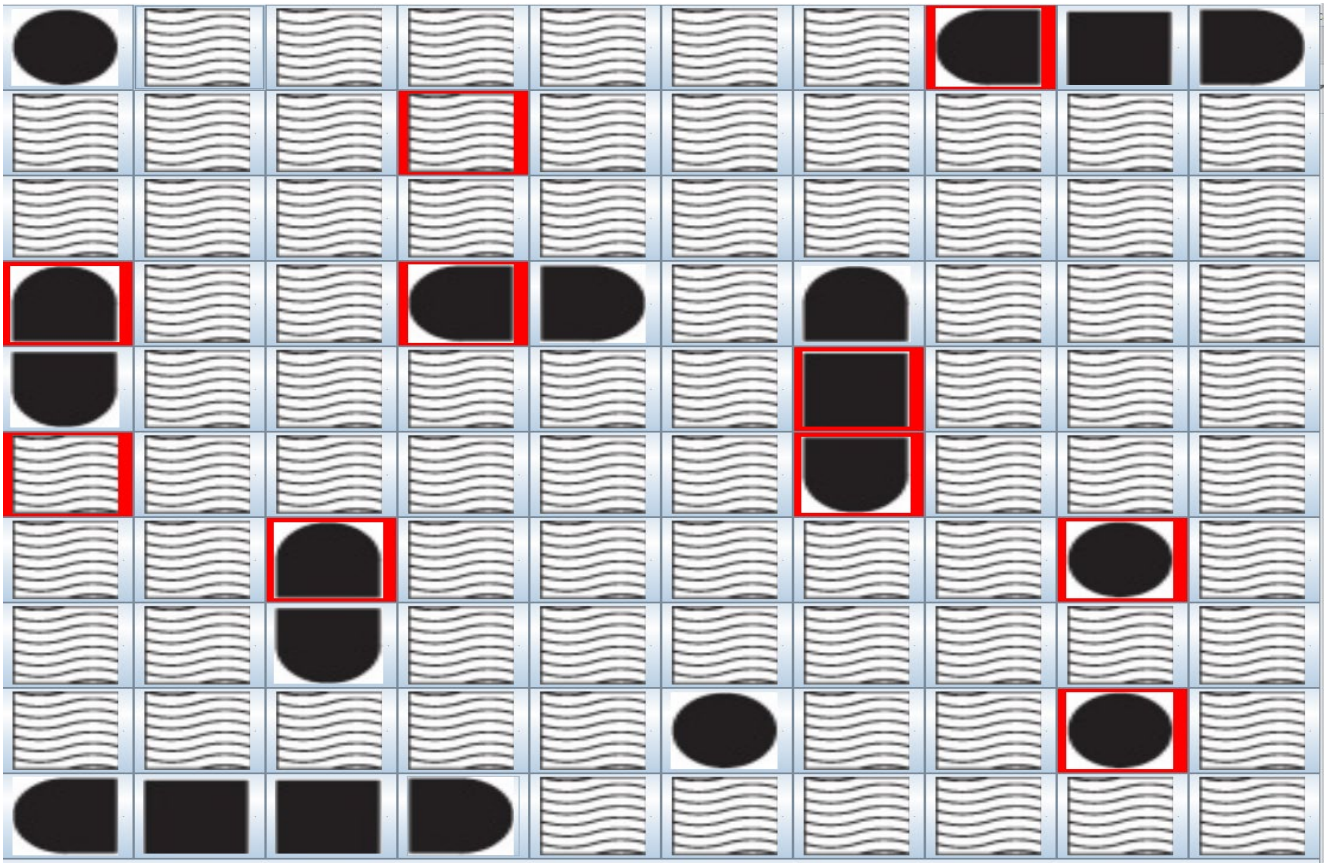
mappa 2 con conoscenza di 10 blocchi:



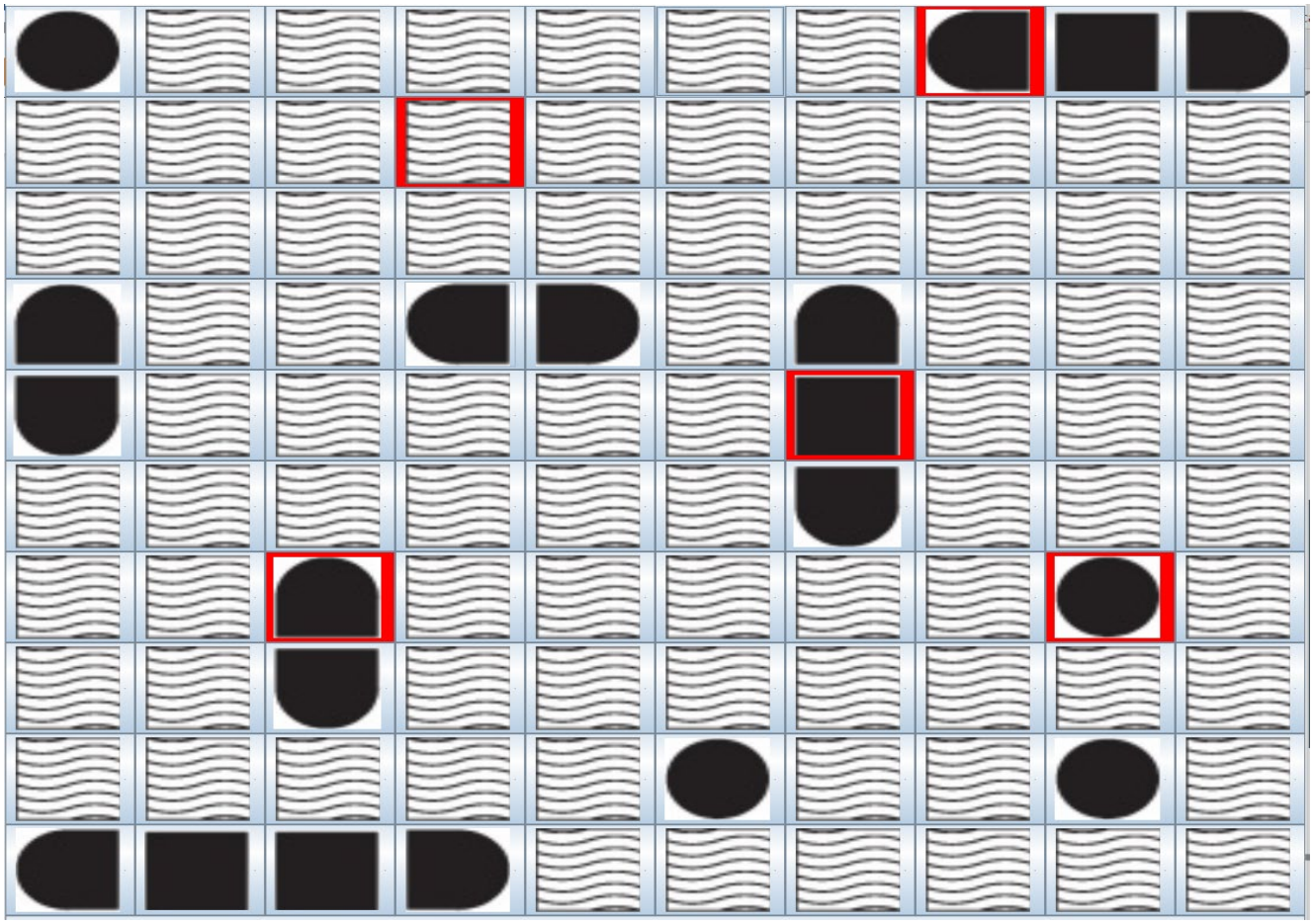
mappa 2 con conoscenza di 5 blocchi:



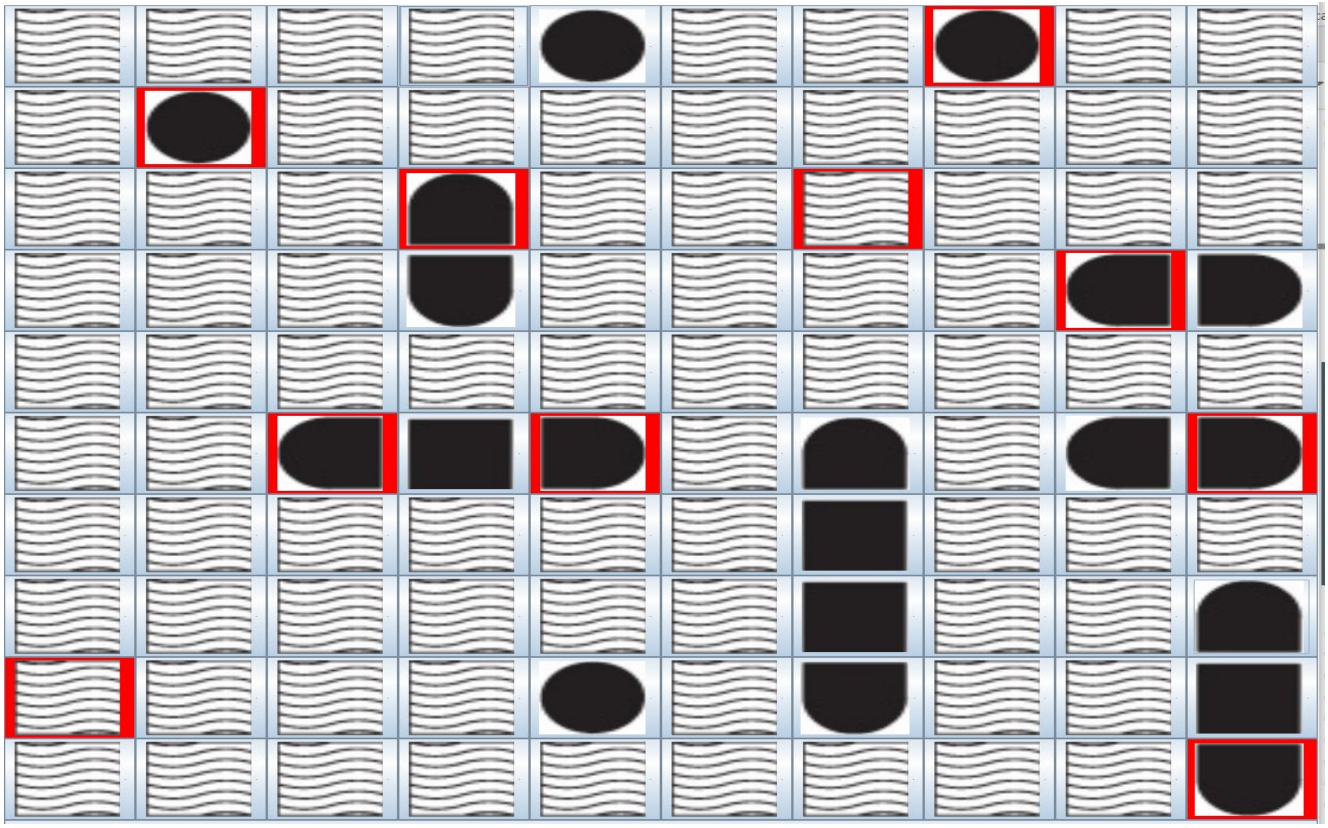
mappa 3 con conoscenza di 10 blocchi:



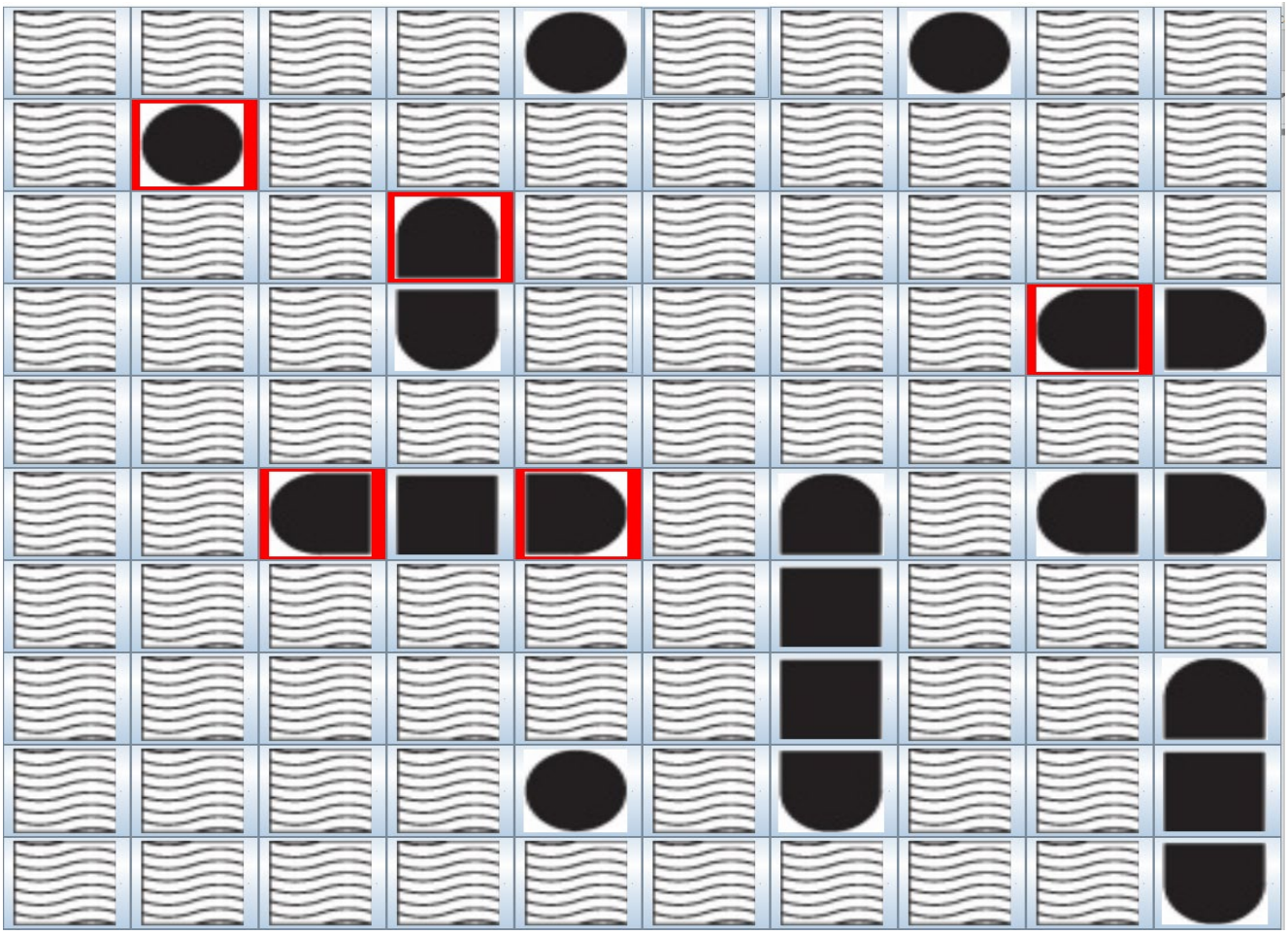
mappa 3 con conoscenza di 5 blocchi:



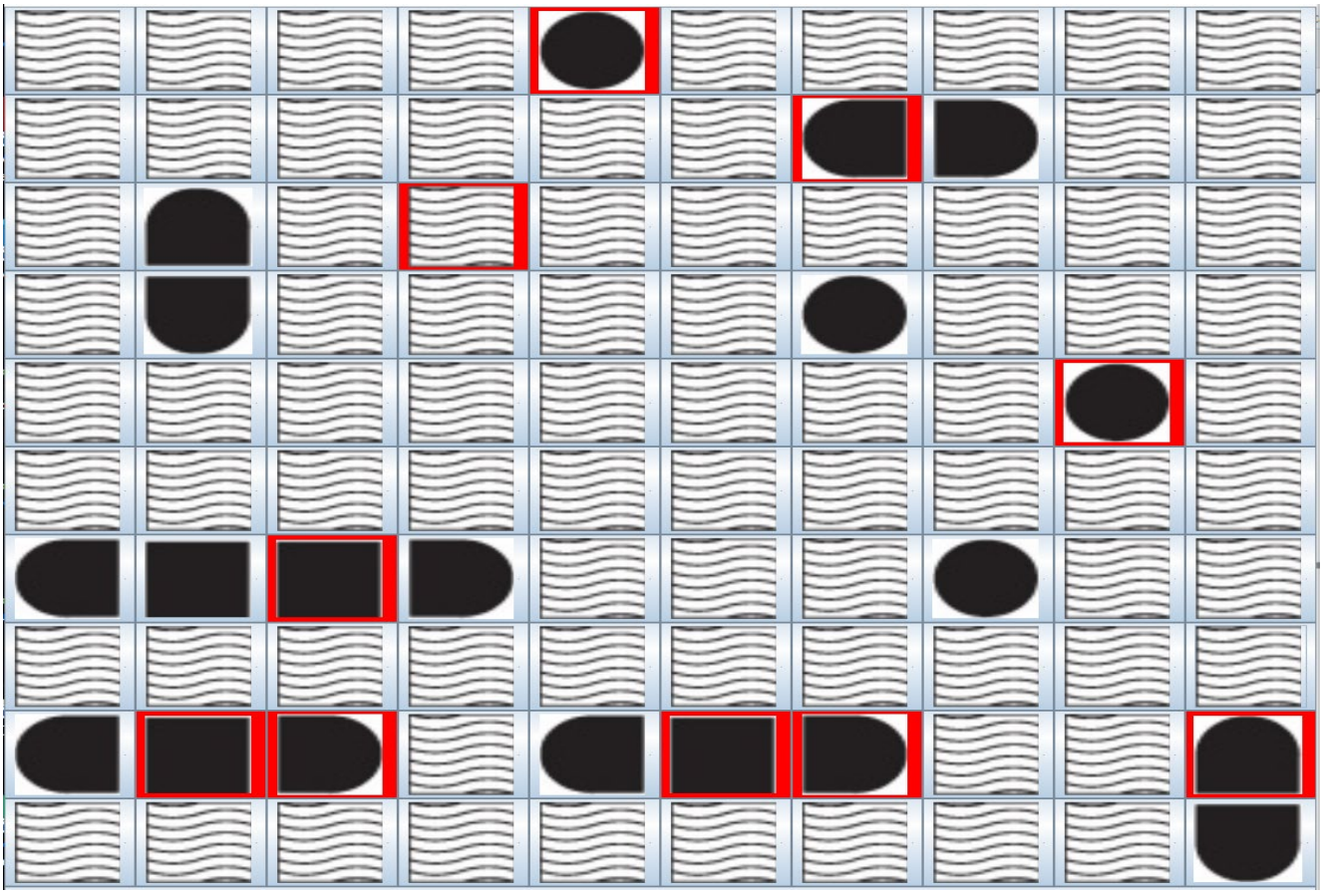
mappa 4 con conoscenza di 10 blocchi:



mappa 4 con conoscenza di 5 blocchi:



mappa 5 con conoscenza di 10 blocchi:



mappa 5 con conoscenza di 5 blocchi: