Relazione IALab - CLIPS

Roger Ferrod, Pio Raffaele Fina, Lorenzo Tabasso

Dipartimento di Informatica, Università degli Studi di Torino
roger.ferrod@edu.unito.it,
pio.fina@edu.unito.it,
lorenzo.tabasso@edu.unito.it

1 Introduzione

La seguente relazione illustra la realizzazione del progetto d'esame di CLIPS, all'interno della quale è stato sviluppato un sistema esperto in grado di giocare "in-solitario" ad una versione semplificata di Battaglia Navale.

In questa versione del gioco, il tabellone ha dimensioni 10×10, e al suo interno vi sono le seguenti navi:

- 1 *portaerei* da 4 caselle,
- 2 *incrociatori* da 3 caselle ciascuno,
- 3 *cacciatorpedinieri* da 2 caselle ciascuno,
- 4 *sottomarini* da 1 casella ciascuno.

Le navi possono avere orientamento orizzontale o verticale, e il loro posizionamento non deve eccedere le dimensioni della tabella.

Il sistema ha a disposizione quattro tipi di azioni:

- **fire(x, y)**: equivale a un'azione percettiva, che permette di vedere il contenuto della cella (x, y).
- **guess(x, y)**: indica che il sistema esperto ritiene ci sia una nave in posizione (x, y).
- **unguess(x, y)**: siccome l'azione *guess(x, y)* è considerata un'ipotesi, grazie all'azione *unguess(x, y)* è possibile ritrattare la *guess(x, y)* eseguita sulla cella (x, y).
- **solve**: usata quando il sistema ritiene di aver risolto il gioco.

Di queste azioni, il nostro sistema esperto può usare al massimo **5 fire** e **20 guess**. L'esecuzione di ogni azione contribuisce al punteggio finale del gioco, calcolato come segue:

$$score = (10 \cdot fok + 10 \cdot gok + 15 \cdot sink) - (25 \cdot fko + 15 \cdot gko + 10 \cdot safe) \quad (1)$$

Dove fok, fko rappresentano rispettivamente il numero di celle oggetto di *fire* giuste e sbagliate; gok, gko rappresentano rispettivamente il numero di celle oggetto di *guess* giuste e sbagliate; $sink, safe$ rappresentano rispettivamente il numero di parti di navi affondate e rimaste a galla.

L'obiettivo finale è quello di trovare una sequenza di azioni che **massimizzi** lo score finale.

1.1 Modello BDI

Nel modellare il nostro sistema esperto, ci siamo ispirati al noto modello di agente intelligente BDI (Belief Desire Intention) e abbiamo strutturato l'architettura dell'agente di conseguenza. Definiamo quindi:

- *Beliefs*: rappresentano le credenze dell'agente sul mondo circostante. Differisce dalla conoscenza in quanto i beliefs possono essere falsi. Nello specifico, per conoscenza intendiamo il reale stato del mondo cosi come è stato percepito tramite fire o conoscenza iniziale, mentre per credenza intendiamo le assunzioni fatte dall'agente circa la posizione delle navi.
- *Desires*: rappresentano la motivazione dell'agente, ossia il goal che vuole perseguire. Nel nostro caso i goals rappresentano la volontà di individuare tutte le navi della scacchiera.
- *Intentions*: rappresentano lo stato deliberativo dell'agente, ovvero quale obiettivo l'agente ha scelto di perseguire.

Ad affiancare queste definizioni, troviamo anche i concetti di Azione e Piano (sequenza di azioni) tramite i quali tracciamo e modifichiamo lo stato del mondo.

Sempre seguendo i principi BDI e del Practical Reasoning, possiamo riassumere il comportamento del nostro agente nei seguenti passi:

```
initialize Beliefs;
while true do
    deliberate new Intention;
    make a Plan;
    execute the Plan;
    update Beliefs;
end
```

Ossia, dato lo stato del mondo, l'agente delibera un'intenzione (e.g. affondare una nave in una determinata posizione), costruisce il piano adatto per portare a termine l'intenzione ed esegue, il piano un'azione per volta; al termine dell'esecuzione la conoscenza del mondo verrà aggiornata e il ciclo ricomincia.

1.2 Modellazione del problema

Abbiamo scelto di approcciarci al problema attraverso due rappresentazioni diverse, ma complementari (Figura 1). La prima è basata sulla struttura a celle della scacchiera e delle navi, mentre un secondo livello opera sul precedente, con maggiore astrazione, rappresentando esplicitamente il concetto (e la morfologia) di un'intera nave, senza badare alla sua composizione in celle. In questo modo realizziamo l'agente BDI descritto nel paragrafo precedente e simuliamo il ragionamento umano; un giocatore umano infatti, nel risolvere il rompicapo, terrà a mente la posizione e l'orientamento delle navi e sarà in grado di sfruttare le informazioni delle celle (a maggiore granularità) per completare patterns e rivedere le proprie ipotesi. Allo stesso modo, il nostro agente è in grado di sfruttare entrambe le tipologie di conoscenza. Più in dettaglio, la componente legata ai Desires

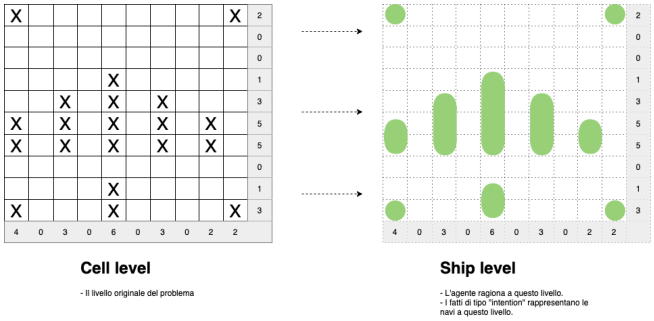


Figura 1. Le due rappresentazioni del problema. A sinistra la rapresentazione basata sul tabellone di gioco e sulle celle, mentre a destra la rappresentazione basata sul concetto di nave.

(numero e tipologia di navi da affondare) e alle Intentions (quale nave posizionare o ritrattare) opera sul livello con maggiore astrazione, mentre le parti che realizzano gli obiettivi preposti (i.e. piano, azioni e osservazioni) utilizzeranno la rappresentazione a celle.

Si presti attenzione al fatto che la rappresentazione esplicita delle navi è l'unica in grado di assicurare il corretto avvicendamento dei goals e delle intenzioni. Risulta infatti possibile procedere gradualmente all'affondamento delle navi, senza forzare il completamento di pattern inesistenti nella realtà. Si pensi ad esempio ad un agente privo di questo livello di rappresentazione, esso sarà solamente in grado di analizzare le celle delle mappa cercando di ricostruire le navi, senza badare a quante e quali ha già affondato in passato, ignorando di fatto tutte le informazioni dettate dal problema (quantità e dimensioni delle navi da affondare) e perseguendo, potenzialmente, obiettivi inverosimili (e.g. l'affondamento, per la terza volta, di una portaerei). Inoltre, un agente cosi costruito avrà una visione locale e limitata del mondo limitando la capacità di inferenza.

1.3 Tipologia di gioco

Sebbene il problema si riconduca al gioco della battaglia navale, il numero limitato di fire (ossia le mosse con le quali si percepisce il mondo) e la presenza delle informazioni sulle righe e sulle colonne (quante celle sono presenti in una data riga/colonna) mettono subito in risalto le differenze con classico gioco da tavolo. In particolare, l'esiguo numero di mosse possibili non ci permette di seguire una strategia "colpisci e osserva" tipica del gioco. D'altra parte, i suggerimenti posti sulle righe e colonne mostrano forti analogie con altri giochi matematici, in particolare con il Sudoku.

Sulla base di queste osservazioni, abbiamo scelto di concentrare la nostra attenzione principalmente sulle informazioni delle scacchiera, tralasciando ogni analogia con il gioco della battaglia navale, con l'eccezione delle prime mosse. Sfruttando infatti ogni possibile fire nei momenti iniziali, è possibile giocare

secondo le classiche regole della battaglia navale solamente per i primi 5 passi, dopodiché il gioco prosegue con il completamente delle informazioni mancanti, in maniera analoga a quanto accade in un Sudoku o cruciverba.

2 Metodologia

Di seguito verranno presentate, a livello concettuale, le principali caratteristiche e meccanismi implementati nell'agente.

Heatmap Al fine di selezionare le *intentions* che massimizzano lo score finale, abbiamo dotato l'agente di un'ulteriore livello di rappresentazione dell'ambiente. Questo livello, graficamente, può essere rappresentato da una heatmap (Figura 2). Ad ogni cella dell'ambiente di gioco, ne corrisponde una della heatmap a cui è associato uno score numerico che indica la **verosimiglianza** di trovare un pezzo di nave in quella cella.

Lo score è calcolato da una semplice equazione:

$$h(x, y) = R_x + C_y \qquad \forall\ x, y \in \{0, .., 9\}$$

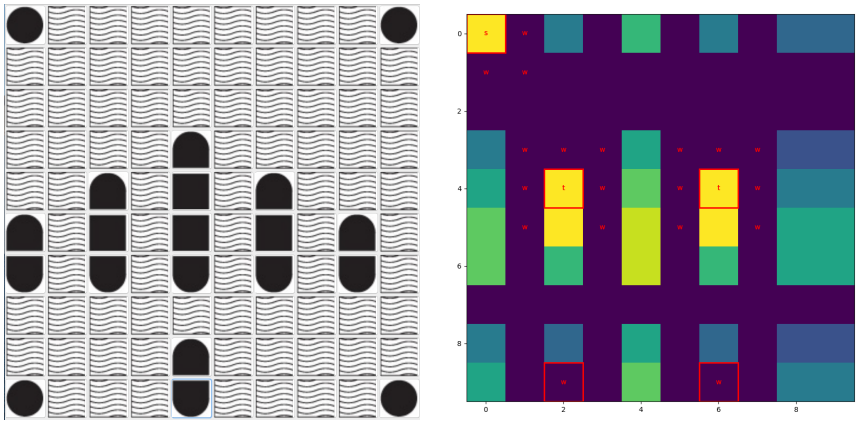
dove $h(x, y)$ è il valore di heat per la cella in posizione (x, y) e R_x, C_y sono rispettivamente il numero di celle occupate da pezzi di navi nella riga x e colonna y. Ricordiamo che R_x, C_y costituiscono parte della conoscenza iniziale disponibile all'agente. La heatmap viene calcolata nella fase iniziale ed aggiornata in base all'ambiente percepito durante la fase di fire. Essa fornisce all'agente una rappresentazione probabilistica dell'ambiente, utile nell'informare le strategie di gioco.

Fase di Fire L'ambiente in cui l'agente agisce è caratterizzato dalla presenza di conoscenza parziale ed incerta. Le uniche azioni a disposizione dell'agente per percepire lo stato del mondo, ed eliminare l'incertezza sul contenuto delle celle, sono le azioni di *fire*. Per questo motivo si è scelto di utilizzare tutte le azioni fire a disposizione dell'agente (5 nella versione del gioco considerata). Vogliamo sottolineare come questa strategia, pur permettendo all'agente di ridurre al minimo l'incertezza sull'ambiente, rappresenta un'operazione "rischiosa" in termini di score finale. La nella formula di scoring (1) si può osservare come alla quantità *fko* sia associato un peso che penalizza maggiormente le fire non andate a segno.

La strategia naïve, per la scelta delle celle su cui effettuare le *fire*, prevedeva di campionare la posizione da una distribuzione uniforme. Sebbene la dimensione della mappa sia ridotta (10×10) la probabilità [1] di avere 5 fire a segno è $\approx 3.2 \times 10e^{-4}$. Per massimizzare l'utilità delle fire si è scelto di sfruttare la heatmap descritta precedentemente (Figura 2b). Nello specifico, le azioni di $fire(x, y)$ vengono eseguite sulle celle che hanno come valore di score la **mediana** dei valori presenti nella heatmap (2).

$$\{fire(x, y) |\ x, y \wedge median(heat(x, y))\} \qquad \forall\ x, y \in \{0, \ldots, 9\} \tag{2}$$

[1] $P(X = 5)$ con $X \sim Binomial(5, \frac{20}{100})$, dove 20 è il numero totale di pezzi di nave, 100 è il numero totale di celle



(a) (b)

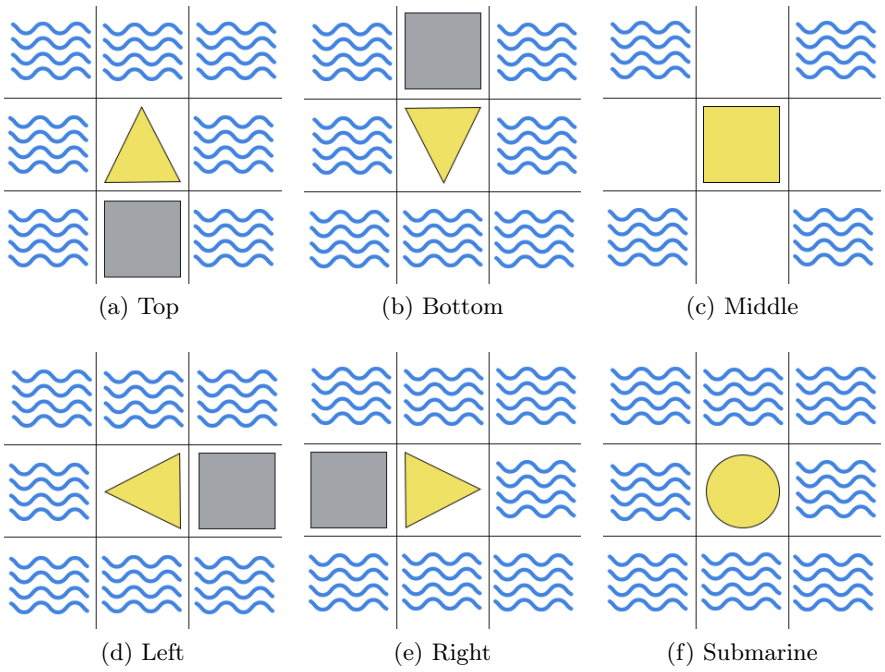
Figura 2. Esempio di mappa e heatmap associata. Le celle in giallo rappresentano la massima verosimiglianza di trovare un pezzo di nave. Le celle con il contorno di colore rosso indicano dove sono state eseguite le azioni di fire. Le celle con la label "w", "t", "s" indicano rispettivamente quelle contenenti acqua, un pezzo top e un sottomarino.

Da un punto di vista della teoria dell'informazione, il valore mediano della heatmap rappresenta la **massima incertezza** sulla presenza o meno in una determinata cella di una parte di nave. Scegliendo il massimo della heatmap, l'agente confermerebbe quello che conosce con maggiore certezza e, in modo completamente opposto, scegliendo il minimo aumenterebbe il rischio di effettuare una *fire* su una cella contente acqua. L'utilizzo della mediana rappresenta un trade-off tra le 2 strategie opposte.

Inferenze contestuali Ogni qualvolta venga eseguita un' azione *fire*, l'agente percepisce esplicitamente il contenuto della cella. In base alle regole del gioco, l'agente può implicitamente effettuare inferenze sul **neighborhood** della cella oggetto di fire (Figura 3). Anche le celle scoperte attraverso le inferenze contestuali partecipano ad aggiornare i valori nella heatmap (se il contenuto scoperto è acqua o un pezzo di nave il valore della score viene posto rispettivamente a 0 o 100).

La fase di fire couadiuvata dal meccanismo di inferenze contestuali permette di ridurre drasticamente l'incertezza nell'ambiente. Si consideri ad esempio il caso in cui 4 delle 5 fire a disposizione scoprano 4 sottomarini, grazie alle inferenze implicite (Figura 3f) le celle totali scoperte sono $4 \cdot 9 = 36$ sulle 100 totali.

Fase di Filtering Segue, alle fasi di percezione del mondo, una ricerca per il migliore candidato al posizionamento di una nave. Osservando attentamente le heatmap generate nei passi precedenti, è possibile individuare una forte correlazione tra la rappresentazione probabilistica e la composizione originale della mappa. In modo del tutto analogo a quanto farebbe l'occhio umano, abbiamo scelto di cercare pattern chiari e sovrapponibili, massimizzando il punteggio della



(a) Top (b) Bottom (c) Middle

(d) Left (e) Right (f) Submarine

Figura 3. Tutti i possibili casi di inferenze contestuali del neighborhood. Le celle in giallo rappresentano le celle oggetto di fire. Su queste celle l'agente percepisce direttamente il contenuto. Nelle restanti 8 celle il contenuto può essere inferito implicitamente in base alle regole del gioco e dal tipo di cella scoperta. Si presti attenzione, come nel casi (a), (b), (d), (e); l'agente è certo di trovare un pezzo di nave (quadrato di colore grigio), ma non può asserire di quale tipologia si tratti.

heatmap. L'agente prosegue quindi con un operazione di filtraggio spaziale (ispirato ai filtri convolutivi utilizzati nelle elaborazioni di immagini) sull'intera mappa. Per semplicità ci riferiremo a questa fase con il termine di "**convoluzione**" in quando basata sulla convoluzione discreta bidirezionale:

$$(f * g)(x, y) = \sum_{i=0}^{M} \sum_{j=0}^{N} f(x, y) g(x - i, y - j) \tag{3}$$

dove f è l'immagine, g la maschera del filtro e MxN la dimensione dell'immagine. La maschera, nel nostro caso specifico, rappresenta il possibile posizionamento di una nave e ne condivide dimensioni e orientamento; i suoi valori (i.e. funzione g) sono posti tutti a 1, riducendo quindi l'operazione ad una semplice somma sui valori della heatmap sottostante.

Per via dei numerosi vincoli che governano il gioco, l'operazione di convoluzione non è realizzabile in ogni coordinata, in particolare deve sottostare ai seguenti vincoli logici:

1. non deve superare i bordi della mappa
2. le celle sottostanti non devono contenere acqua
3. non vi devono essere incompatibilità con le guess precedenti
4. devono essere rispettati i vincoli posti su righe e colonne

Se tutte le pre-condizioni sono rispettate, è possibile calcolare il punteggio (dato dall'operazione stessa di convoluzione) in ogni coordinata e orientamento possibile, massimizzandone il valore. In questo modo la risoluzione del gioco prosegue in maniera "**greedy**", ossia selezionando, ad ogni passo, la migliore posizione e orientamento possibile della nave. Tale scelta rappresenta quindi l'intenzione dell'agente e, in riferimento al paradigma BDI descritto in precedenza, la fase di filtraggio convolutivo è la fase di deliberazione dell'agente. L'operazione viene ripetuta, fintantoché possibile, per ogni nave.

Backtracking Qualora non sia più possibile proseguire con il posizionamento delle navi, si rende necessaria un'operazione di backtracking per eliminare le ipotesi precedenti. A tal fine è importante, in fase di planning, tenere traccia di ogni azione in modo da poter disfare un piano. In particolare, questo viene implementato costruendo uno stack dei piani contenente, in ordine LIFO (Last In First Out), tutti i piani eseguiti in passato. Più in dettaglio, l'operazione di backtracking ricava dalla cima dello stack l'ultimo piano eseguito (i.e. l'ultima nave posizionata) e tutte le azioni corrispondenti, quindi annulla tutti i cambiamenti apportati ed elimina il piano dallo stack.

Una volta ripristinato lo stato precedente del mondo però, in assenza di ulteriori informazioni, l'agente tornerà a commettere gli stessi errori, ossia deliberando le medesime intenzioni e generando, di conseguenza, un ciclo infinito. Onde evitare questa situazione, la fase deliberativa dell'agente deve tenere in considerazione i tentativi già percorsi in passato, evitando di eseguire l'operazione di filtering su tali posizioni. Per svolgere questo compito, manteniamo i vecchi piani in memoria. Eseguendo semplicemente il pop sullo stack infatti, il piano oggetto di backtracking risulta comunque in working memory e, ad esso, sono associate tutte le informazioni utili ad evitare il ripetersi dell'errore (coordinate, tipo di nave e orientamento).

Tuttavia, l'ipotesi che si possa tentare un'assegnazione solamente una volta è molto forte e, pertanto, abbiamo preferito implementare un meccanismo di *aging*, simile a quanto avviene nello scheduling dei processi nei moderni sistemi operativi. Ogni piano viene inizializzato con un'età pari a zero e il suo valore viene incrementato ad ogni nuovo tentativo. In tal senso, in fase di planning, se è già presente un piano associabile all'intenzione deliberata, non ne viene instanziato uno nuovo, bensì si recupera quello già presente riposizionandolo in cima allo stack. Quando l'età di un piano supera una determinata soglia (iperparametro da configurare), risulta possibile eseguire nuovamente quella mossa. In questo modo viene concessa una seconda, ed ultima, possibilità.

Maccanismo di resa Per come ci siamo posti dinanzi al problema, il nostro agente continuerà a perseguire i propri obiettivi finché non sarà riuscito ad individuare tutte le navi. Purtroppo però, a causa dei vincoli imposti dal backtracking

(aging) e dal numero limitato di mosse possibili, può accadere che l'agente sia costretto ad arrendersi senza aver trovato una soluzione completa. Questa situazione occorre quando, avendo esaurito tutti i backtrack consentiti, l'operazione di filtering non riesce ad individuare una posizione per la nave a meno di dover richiedere un ulteriore backtracking. Ne consegue una situazione di stallo che l'agente interpreta come esaurimento delle mosse disponibili e richiede, in modo esplicito, la terminazione del gioco.

3 Architettura

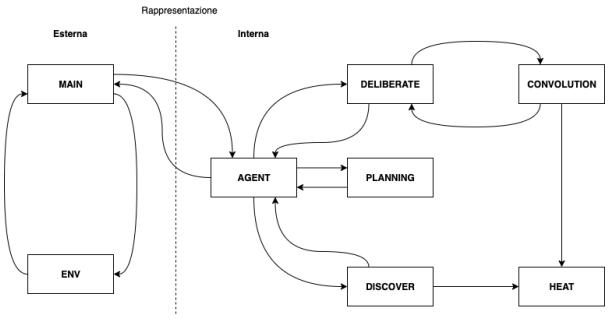


Figura 4. l'architettura del sistema esperto realizzato, sviluppato secondo il modello BDI.

A livello architetturale l'agente è stato implementato utilizzando il costrutto `defmodule`. L'architettura dell' agente (Figura 4) si compone in 3 moduli principali: `AGENT, DELIBERATE, PLANNING` (che richiamano il framework BDI) e altri 3 moduli di supporto: `HEAT, DISCOVER, CONVOLUTION` utili per semplificare il flusso di esecuzione.

- `AGENT`: il ruolo principale di questo modulo è **coordinare** la comunicazione ed il flusso di esecuzione tra gli altri moduli. Inoltre data la struttura dello *stack dei piani* e la presenza di un *metalivello* di rappresentazione, ha l'ulteriore compito di **selezionare** una singola azione dallo stack e "**tradurre**" la rappresentazione delle azioni interna all'agente nella rappresentazione esterna utilizzata nel modulo `MAIN` ed `ENV`.
- `DELIBERATE`: il ruolo principale di questo modulo è analizzare lo stato del mondo in un determinato step e decidere quale *intention* asserire. Per quanto rigurda le `intention-sink`, il modulo in questione è strettamente legato al modulo `CONVOLUTION`, in quanto quest'ultimo, gli fornisce la migliore posizione su cui asserire l'intenzione per affondare una barca. In maniera complementare, se il modulo `CONVOLUTION` non riuscirà a calcolare lo score, questo indicherà al modulo `DELIBERATE` che non è possibile procedere con

l'attuale piano, dunque verrà asserito un fatto `intention-abort` che indica l'intenzione di distruggere il piano precedente ed innescare il meccanismo di backtracking.
- `CONVOLUTION`: questo modulo implementa la fase di filtering intrdotta nella sezione Metodologia. Tra le molteplici regole presenti, di fondamentale importanza sono le regole `check-conv-cell-*` che verificano se i vincoli logici per effettuare l'operazione di convoluzione in una determinata area della mappa siano rispettati.
- `HEAT`: questo modulo implementa la heatmap e il meccanismo di calcolo della mediana presentati nella sezione Metodologia. Nello specifico, il `deftemplate heatmap` è la principale struttura dati utilizzata nella rappresentazione.
- `DISCOVER`: questo modulo implementa il meccanismo di inferenze contestuali presentato in Metodologia. I casi mostrati in Figura3 sono codificati nelle regole `defrule discover-neighborhood-*`. In base al risultato dell'azione fire eseguita, le regole di questo modulo hanno due funzioni: asserire le `deftemplate k-cell` associate al neighborhood e aggiornare i valori presenti in `deftemplate heatmap`.
- `PLANNING`: questo modulo contiene una serie di regole e funzioni utili a generare e manipolare lo *stack dei piani*, i piani e le singole azioni all'interno di essi. Per ridurre al minimo il numero di regole necessarie alle funzionalità del modulo, la maggior parte della conoscenza è stata codificata in modo procedurale sfruttando la natura del problema (vedasi `deffunction generate-ship-guess` e `deffunction generate-ship-*-water`). Inoltre attraverso la `defrule plan-backtracking` viene inizializzato la fase di backtracking, effettuando l'operazione di *pop* del piano corrente dallo stack.

3.1 Flusso di Esecuzione

L'utilizzo del costrutto `defmodule` ha permesso sia di attuare il principio di *Separation of Concerns* ma sopratutto di semplificare il flusso di esecuzione dell'interprete. Per la strategia di gioco implementata nell'agente, l'esecuzione può essere divisa in 4 macro fasi lineari (Figura 5).

Figura 5. Principali fasi del flusso di esecuzione.

Nella fase di *init* (Figura 7), viene principalmente inizializzato l'ambiente di gioco e calcolati gli score del livello di heatmap. Nella fase di *fire* (Figura 8)

vengono utilizzate le 5 azioni di fire a disposizione, scoperti i relativi neighbors attraverso il meccanismo di inferenze contestuali e aggiornata la heatmap.

La fase di *guess* (Figura 9) rappresenta il *game loop* principale, dove avviene la delibera delle intenzioni per posizionare le 20 guess a disposizione, la creazione dei piani e l'eventuale meccanismo di backtracking. Nell'ultima fase di *solve* (Figura 9) viene proposta la soluzione finale del gioco se l'agente è riuscito a trovare la soluzione completa, oppure, attraverso il meccanismo di resa. Segue il calcolo del punteggio finale.

3.2 Codifica CLIPS

Per codificare il problema nel linguaggio CLIPS, occorre introdurre numerosi fatti di supporto in grado di rappresentare l'intero meccanismo e gestire l'attivazione delle regole. In questa sezione riportiamo i fatti più significativi.

b-cell Parallelamente ai fatti k-cell, che definiscono la conoscenza certa del mondo, occorre sviluppare una fatto b-cell rappresentante la credenza dell'agente, ovvero l'ipotesi sul mondo a seguito di un'azione guess o unguess.

```
(deftemplate b-cell
    (slot x)
    (slot y)
    (slot content (allowed-values water boat hint))
)
```

Come si evince dal template del fatto, il contenuto di una b-cell può assumere i valori *water* (l'agente crede che vi sia acqua), *boat* (l'agente crede che vi sia una parte di nave) e *hint* (l'agente ha inferito, ed è certo, che in quella posizione vi sia una parte di nave). La b-cell di tipo *hint* differisce da *boat* per il fatto che rappresenta una conoscenza non ritrattabile, ovvero non soggetta ad aventuali backtrack. Le b-cell *hint* vengono infatti asserite durante le inferenze sui neighborhoods e mai più modificate.

update-k-per-* Le informazioni sulle righe e sulle colonne, codificate con i rispettivi fatti *k-per-row* e *k-per-col*, derivano unicamente dalla situazione iniziale della mappa e rivestono un ruolo essenziale per il calcolo delle heatmap. Tuttavia occorre aggiornare questi valori tenendo in considerazione gli sviluppi del gioco. Onde evitare di sovrascrivere le preziose informazioni contenute in *k-per-row* e *k-per-col*, abbiamo scelto di sviluppare i fatti *update-k-per-row* e *update-k-per-col*, i quali vengono mantenuti costantemente aggiorati.

```
(deftemplate updated-k-per-row
    (slot row)
    (slot num)
)

(deftemplate updated-k-per-col
```

```
    (slot col)
    (slot num)
)
```

Il loro valore è determinato dal valore iniziale, presente nella corrispettiva k-per-*, e dal numero di *b-cell* presenti in quella riga/colonna. Si presti attenzione al fatto che *k-cell* e *b-cell* di tipo hint non partecipano al calcolo. Se cosi fosse, l'agente non riuscirebbe a posizionare correttamente le navi in quanto il vincolo sulla riga/colonna imporrebbe una condizione che non rispecchia la realtà. Infatti, alle *k-cell* o *b-cell* hint non corrisponde una guess e, agli occhi dell'agente, quelle celle sono perfettamente sovrascrivibili. Questa particolarità nasce dal doppio sistema di rappresentazione del mondo: a celle e a navi. Una *k-cell*, *b-cell* hint, cosi come un valore di heat posto a 100, suggeriscono all'agente che in quelle coordinate vi è la certezza di trovare una parte di nave, tuttavia l'agente deve poter sovrapporre a quelle cella un'intera nave e, pertanto, i vincoli sulle righe e sulle colonne devono permette quest'operazione.

intention-sink Intention-sink codifica l'intenzione, da parte dell'agente, di affondare una specifica tipologia di nave, con un determinato orientamento e posizione. Cosi come avviene nella convoluzione e, in generale in tutto il progetto, identifichiamo una nave tramite le coordinate della poppa.

```
(deftemplate intention-sink
    (slot x-stern)
    (slot y-stern)
    (slot orientation (allowed-values ver hor))
    (slot type (allowed-values air-carrier cruiser destroyer submarine))
)
```

intention-fire L'intention-fire codifica l'intenzione di eseguire una fire in determinate coordinate.

```
(deftemplate intention-fire
    (slot x)
    (slot y)
)
```

intention-abort L'intention-abort traccia la necessità di eseguire backtracking. In questo caso non occorre specificare alcuna coordinata in quanto il meccanismo di backtracking lavorerà unicamente a partire dall'ultimo piano presente nello stack. Ne consegue quindi che il fatto può essere modellato come fatto ordinato e non necessità di un template.

intention-solve Intention-solve è l'intenzione, deliberata al termine del gioco, che indica la volontà di terminare l'esecuzione e calcolare il punteggio finale. Può sopraggiungere in soli due casi: l'agente crede d'aver trovato una soluzione

completa e richiede il calcolo del punteggio, oppure non è più possibile continuare con nuove ricerche e si richiede il calcolo del punteggio sulla soluzione parzialmente elaborata. Anche in questo caso il fatto è ordinato.

action La singola azione è codificata con il fatto *action* che può assumere i valori *guess*, *unguess*, *water* e *fire* a seconda di quale sia l'esecuzione richiesta. La generazione di un id univoco è fondamentale per permettere la racconta delle azioni all'intero dei piani.

```
(deftemplate action
    (slot id (default-dynamic (gensym*))) ; genX
    (slot x)
    (slot y)
    (slot type (allowed-values guess unguess water fire))
)
```

plan Il piano, codificato con il fatto *plan*, contiene la sequenza di azioni da eseguire (rappresentata come un multislot contenente gli id delle azioni) oltre a tutte le informazioni utili all'indentificazione della nave coinvolta. Troviamo quindi le coordinate della poppa, l'orientamento e la tipologia di nave. Inoltre, come descritto nella Sezione 2 (par.*backtracking*), è presente lo slot *age* che codifica l'età (i.e. priorità) del piano. Infine, poiché sarebbe rischioso operare direttamente sulla lista di azioni, manipolandole man mano che vengono eseguite, teniamo traccia, tramite un indice (*counter*), della prossima azione da eseguire. Il valore dell'indice è inizializzato a 1 al momento della creazione del piano e raggiungerà il valore massimo (corrispondente alla lunghezza della sequenza di azioni) al termine dell'esecuzione dell'intero piano. In caso di backtracking, si provvederà ad eseguire le azioni a ritroso, ossia partendo dall'ultima posizione fino alla prima, riportando il valore a 1.

```
(deftemplate plan
(slot id (default-dynamic (gensym*)))
    (slot counter)
    (slot ship)
    (multislot action-sequence (type SYMBOL))
    (slot x)
    (slot y)
    (slot orientation)
    (slot age)
)
```

convolution-area e **conv-cell** Per modellare il meccanismo di convoluzione, ci affidiamo a due tipologie di fatti: *convolution-area* e *conv-cell*. In maniera analoga a quanto avviene con piani ed azioni (dove il piano opera a livello di navi mentre le azioni sulle singole celle), la convoluzione è gestita dal fatto *convolution-area*, che codifica l'intera superficie coinvolta (i.e. filtro convolutivo), e *conv-cell* che rappresenta la singola cella appartenente all'area sottoposta

a convoluzione. Ne consegue quindi che i fatti *conv-cell* sono contenuti, tramite un multislot di id, all'intero di *convolution-area*. Più in dettaglio, *convolution-area* contiene anche la tipologia, la dimensione, le coordinate e l'orientamento della nave soggetta a convoluzione, oltre allo score corrispondente, computato sulla base dei valori heatmap sottostanti. Contiene inoltre altri slot di supporto, come *computed*, *visited* e *collected*, necessari alla corretta gestione del flusso operativo.

Il fatto *conv-cell* contiene invece l'informazione parziale presente in una determinata cella del filtro. Contiene inoltre un riferimento al fatto *convolution-area* di cui fa parte.

```
(deftemplate convolution-area
    (slot id)
    (slot type)
    (slot size)
    (slot x)
    (slot y)
    (slot orientation (allowed-values ver hor))
    (slot score (default 0))
    (multislot area)
    (slot computed (default FALSE))
    (slot visited (default 0))
    (slot collected (default FALSE))
)

(deftemplate conv-cell
    (slot id)
    (slot area-id)
    (slot x)
    (slot y)
    (slot counted (default FALSE))
)
```

Per ulteriori dettagli sugli altri fatti modellati nel progetto, si rimanda al codice completo.

4 Esperimenti e Risultati

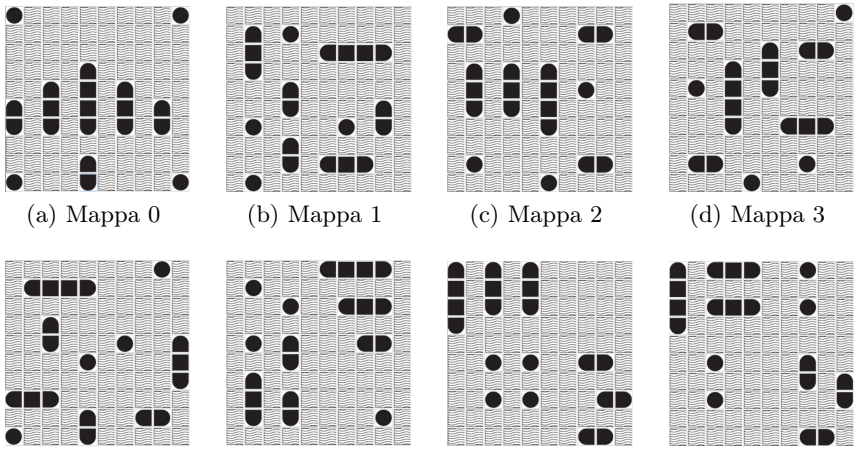
Per valutare il comportamento dell'agente al variare dell'ambiente e delle strategie utilizzate abbiamo creato una suite di esperimenti. Nello specifico gli esperimenti sono rivolti a rispondere a due principali domande:

- *"Quale stategia di gioco implementata nell'agente massimizza il punteggio finale?"* Esperimento 1 e 2
- *"Qual'è l'impatto del livello di conoscenza iniziale?"* Esperimento 3.

Nei risultati sono riportate 2 metriche: il **numero di steps** necessari per proporre la soluzione finale e lo **score** finale. Inoltre riportiamo se la soluzione finale proposta è completa oppure ottenuta attraverso il meccanismo di resa, indicata con il simbolo ⚐.

4.1 Mappe

Per stabilire la bontà delle decisioni prese in fase di progettazione, si è deciso di valutare il comportamento dell'agente su differenti ambienti di gioco. Come osservabile in Figura 6, sono state create 8 mappe con differente morfologia che rappresentano differenti livelli di difficoltà.



(a) Mappa 0 (b) Mappa 1 (c) Mappa 2 (d) Mappa 3

(e) Mappa 4 (f) Mappa 5 (g) Mappa 6 (h) Mappa 7

Figura 6. Configurazione delle mappe utilizzate negli esperimenti.

Durante le sperimentazioni è stata osservata una forte correlazione tra "complessità" della mappa e score finale. Sebbene non esista una definizione unica e formale del livello di complessità della mappa, si è provato a quantificare questo fenomeno calcolando la varianza dei valori della heatmap associata. Intuitivamente, mappe con cluster di navi "ben definiti" e spazialmente distanti tra di loro (Figura 10b, 10l, 10p), portano a delle heatmap con alta varianza. Dal punto di vista dell'agente questo si traduce nell' avere aree in cui i beliefs espressi dalla heatmap sono ben rappresentativi della vera configurazione della mappa sottostante. Vogliamo sottolineare che la varianza non sempre è indicativa e supporta la nostra ipotesi (Figura 10n), in questi casi servirebbe un'analisi più approfondita.

4.2 Esperimento 1: Aging

La prima tranche di esperimenti mira ad individuare la migliore configurazione per il parametro "Max age", ossia per quanto tempo l'agente che necessità di backtracking non può eseguire nuovamente piani già tentati in precedenza. In Tabella 1 sono riportati i risultati degli esperimenti con valori di aging pari a 5, 10 e 20.

Mappa	Steps	Score	Steps	Score	Steps	Score
age	5		10		20	
0	25	115	25	115	25	115
1	29 ⚑	70	34 ⚑	50	61 ⚑	50
2	28 ⚑	55	34 ⚑	55	45 ⚑	-15
3	25	-70	25	-70	25	-70
4	28 ⚑	-210	34 ⚑	-210	42 ⚑	-210
5	28 ⚑	60	33 ⚑	40	83 ⚑	0
6	25	275	25	275	25	275
7	24	150	24	150	24	150

Tabella 1. Risultati dell'esperimento 1 al variare dell'aging.

E' possibile osservare un netto peggioramento delle prestazioni all'aumentare del valore di aging. Si passa infatti da un valore medio di score pari a 55.6 con age=5, ad uno score medio di 36.9 con age=20. Tale risultato conferma l'ipotesi iniziale, secondo la quale occorre fornire all'agente la possibilità di ripetere una strada già percorsa in passato. Infatti, al crescere del valore di age tale capacità e nel caso limite in cui age valga infinito si ottiene un agente impossibilitato a ritornare sui propri passi.

4.3 Esperimento 2: Fire

Sebbene il meccanismo delle fire sia stato ideato a partire dal concetto di mediana, abbiamo provata a cambiare strategia per valutarne meglio l'efficacia. Se consideriamo le inferenze contestuali descritte nella sezione Metodologia, possiamo notare come sia molto più informativa una fire eseguita su una nave, piuttosto che nell'acqua; inoltre, una fire andata a segno non penalizzerà lo score finale. Abbiamo pertanto sperimentato una nuova strategia che sposta l'attenzione verso i valori più alti di heatmap. In particolare l'obiettivo è trovare il massimo valore che occorre più volte ed eseguire le fire su tutte le celle corrispondenti; per questo motivo faremo riferimento a questa strategia con il nome "max-max". In tabella Tabella 2 sono riportati i risultati.

Sebbene vi sia un aumento medio delle prestazioni (+55%), vi sono diverse criticità. La nuova strategia rischia infatti di minare gli ottimi risultati già ottenuti con il calcolo della mediana, aumentando in certi casi il numero di errori; inoltre la strategia max-max è più complessa da implementare. Per questi motivi

Mappa	h	Median		h	Max-max	
0	6	25	115	8	25	140
1	6	29 ⚐	70	8	34 ⚐	85
2	5	28 ⚐	55	8	34 ⚐	120
3	4	25	-70	7	25 ⚐	-105
4	4	28 ⚐	-210	7	34	100
5	5	28 ⚐	60	8	33 ⚐	95
6	5	25	275	8	25	200
7	5	24	150	7	24 ⚐	55

Tabella 2. Risultati dell'esperimento 2 al variare della strategia di selezione.

riteniamo che i relativi benefici portati dalla nuova strategia non ne giustifichino l'adozione, pertanto d'ora in avanti continueremo ad usare la strategia con l'utilizzo della mediana, descritta nella Sezione 2.

4.4 Esperimento 3: Conoscenza iniziale

Infine, con l'ultima tranche di esperimenti, verifichiamo il comportamento dell'agente all'aumentare della conoscenza iniziale. Siccome la conoscenza iniziale può fornire informazioni aggiuntive sia sulla presenza di navi o sulla presenza di acqua, abbiamo scelto di aumentare entrambe le tipologie in ugual misura. Di conseguenza, nella mappa con 2 celle conosciute a priori avremo la conoscenza certa su una parte di nave e una cella occupata da acqua. Allo stesso modo la mappa con 4 celle conosciute avrà 2 parti di navi (per convenzione apparterranno sempre a navi diverse) e 2 celle con acqua. I risultati sono riportati in Tabella 3.

Mappa	0		2		4		8		12	
0	25	115	25	105	25	95	25	75	25	240
1	29 ⚐	70	29 ⚐	155	29 ⚐	120	28 ⚐	155	25	240
2	28 ⚐	55	28 ⚐	45	29 ⚐	85	25	110	24 ⚐	115
3	25	-70	29 ⚐	-260	28 ⚐	-310	27 ⚐	-355	32 ⚐	-15
4	28 ⚐	-210	28 ⚐	-200	32 ⚐	-55	29 ⚐	25	25	240
5	28 ⚐	60	28 ⚐	50	28 ⚐	40	25	210	23	240
6	25 ⚐	275	27 ⚐	-55	25	255	23	260	22	240
7	24 ⚐	150	27 ⚐	65	24	95	23	285	21	290

Tabella 3. Risultati dell'esperimento 3 al variare del livello iniziale di conoscenza.

In questo caso si nota un notevole e controintuitivo peggioramento delle prestazioni in tutte le mappe. Il motivo però è legato al calcolo dello score (descritto in Sezione 1) piuttosto che alle inferenze dell'agente. Infatti, ad una cella conosciuta sin dall'inizio sarà associato un fatto *k-cell* che andrà ad influire il calcolo

delle statistiche finali del gioco. In particolare, anche in presenza di una guess corretta, la cella conosciuta verrà conteggiata come "fired" anziché "guessed". Questo provoca una diminuzione dello score totale che può essere contrastato, nelle mappe con maggiore conoscenza, con la diminuzione degli errori. Per questo motivo occorre aggiungere 8 o 12 celle conosciute per osservare un miglioramento dello score. Tuttavia ricordiamo che questo errore non è imputabile all'agente in quanto, osservando le guess inferite, si può verificare la correttezza dell'operato.

5 Appendice A

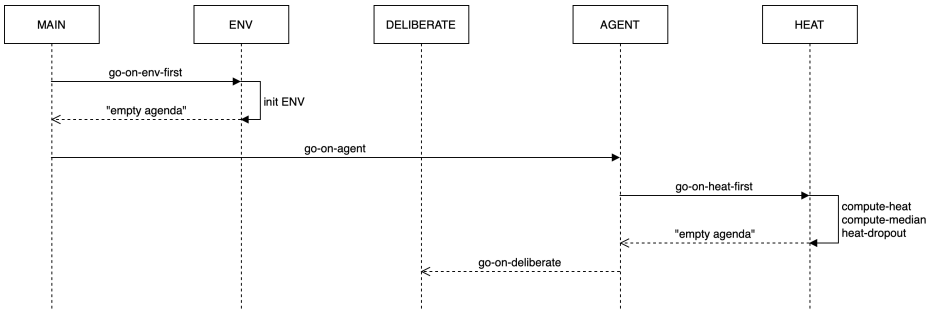


Figura 7. Flusso di attivazione delle regole e fatti in fase iniziale.

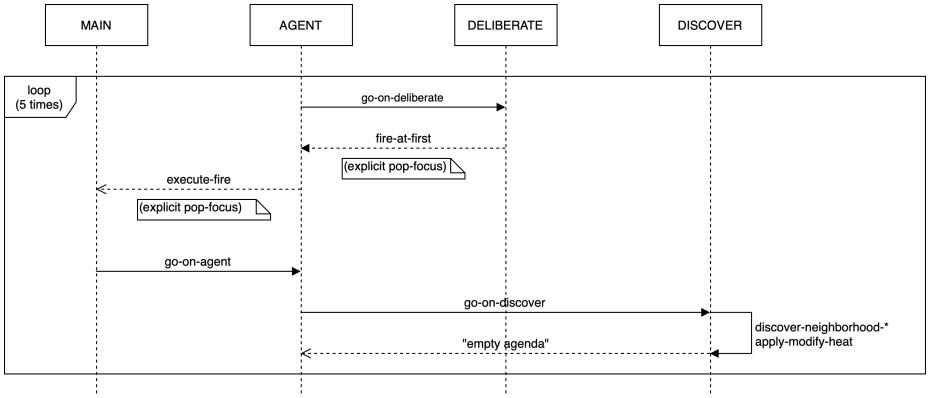


Figura 8. Flusso di attivazione delle regole e fatti in fase di fire.

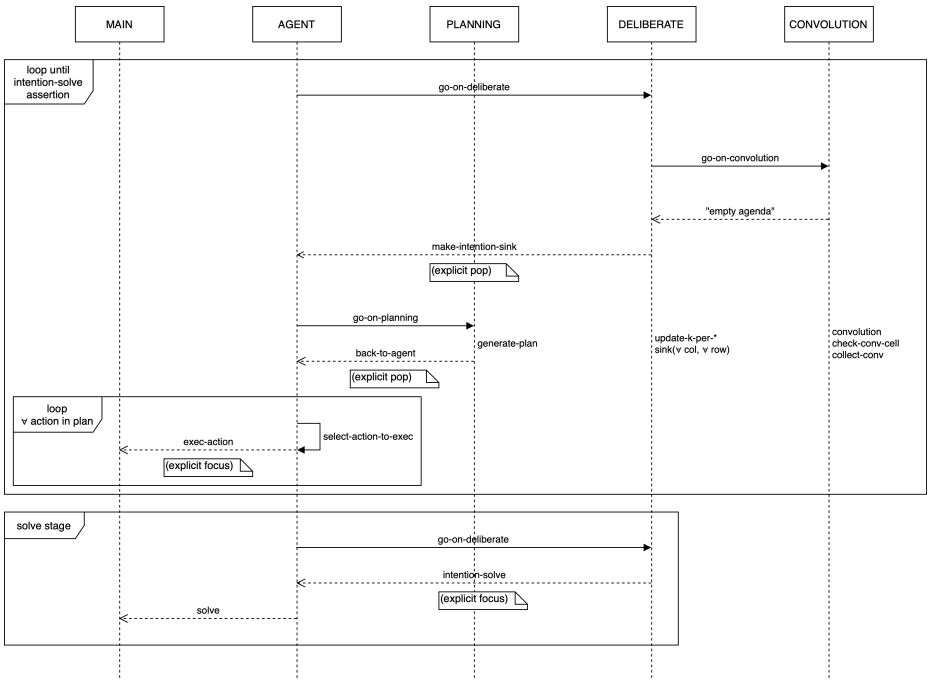
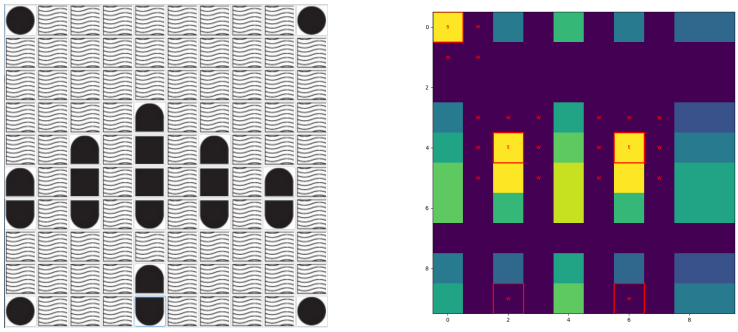
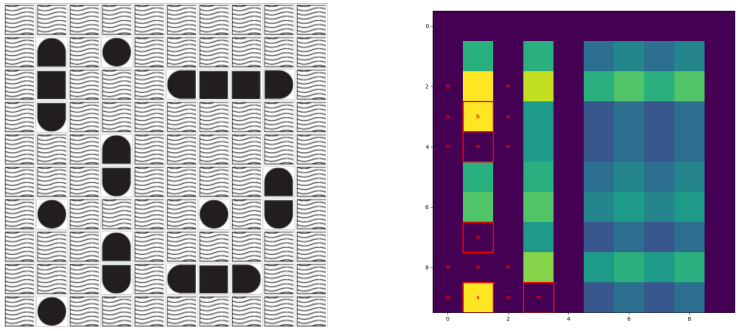


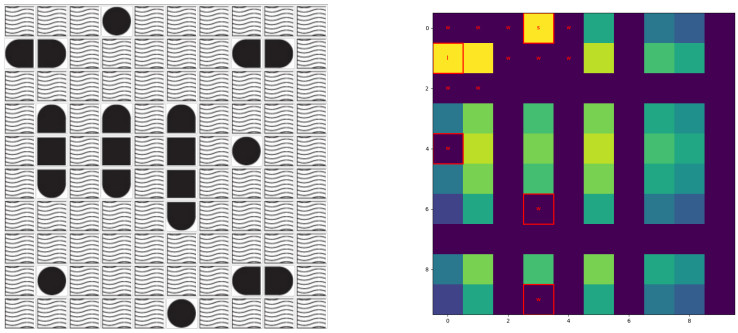
Figura 9. Flusso di attivazione delle regole e fatti in fase di guess e durante la fase finale di solve (in basso).



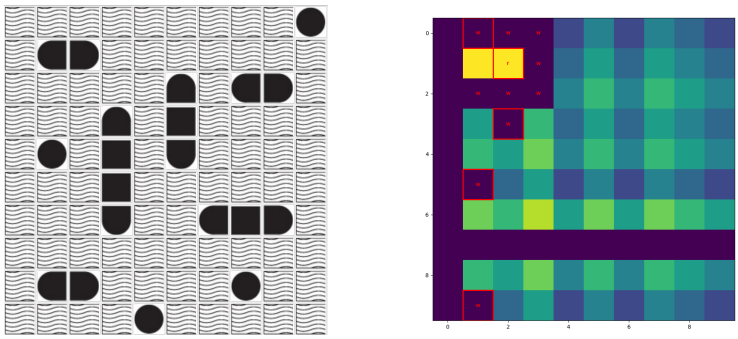
(a) Mappa 0 (b) $\sigma^2 = 4.40$



(c) Mappa 1 (d) $\sigma^2 = 3.57$

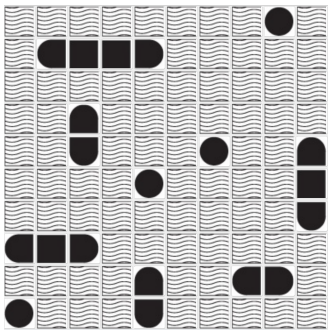


(e) Mappa 2 (f) $\sigma^2 = 3.80$

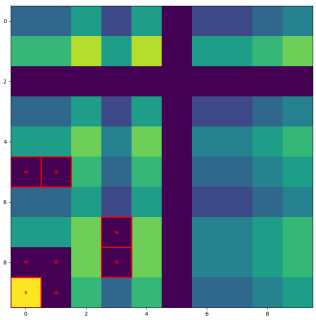


(g) Mappa 3 (h) $\sigma^2 = 2.15$

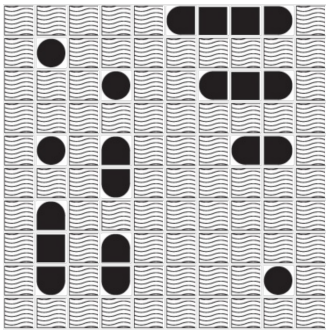
Figura 10. Configurazione delle mappe utilizzate negli esperimenti e heatmap associate (prime 4).



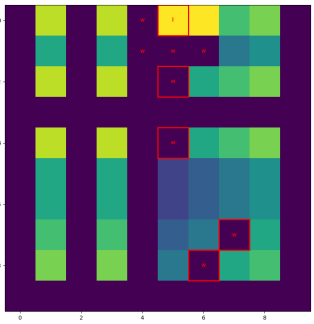
(i) Mappa 4



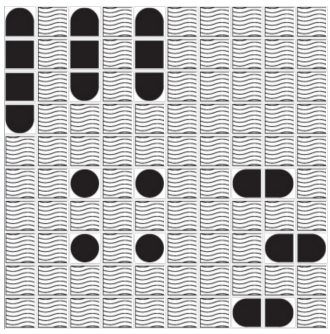
(j) $\sigma^2 = 2.07$



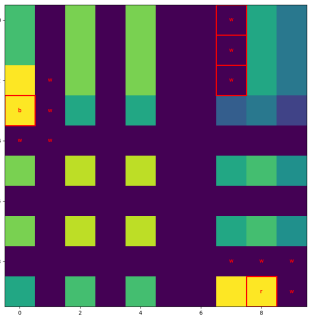
(k) Mappa 5



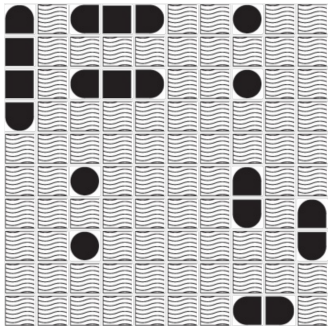
(l) $\sigma^2 = 4.06$



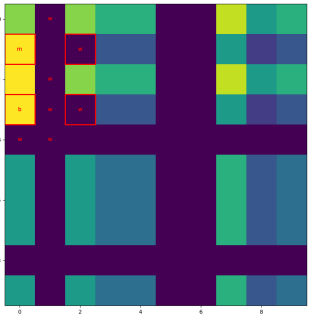
(m) Mappa 6



(n) $\sigma^2 = 3.28$



(o) Mappa 7



(p) $\sigma^2 = 4.16$

Figura 10. Configurazione delle mappe utilizzate negli esperimenti e heatmap associate (ultime 4).