

Intelligenza Artificiale e Laboratorio

Anno Accademico 2019/20

Giuseppe Mazzone - Sante Altamura

Indice

1 Planning in CLIPS

1.1	Descrizione del problema
1.2	Modellazione sistema esperto

2 Implementazioni

2.1	Implementazione in FASI
2.1.1	Modellazione della conoscenza
2.1.2	Modellazione delle regole di expertise
2.2	Implementazione in MODULI
2.2.1	Modellazione della conoscenza
2.2.2	Modellazione delle regole di Expertise

3 Risultati ottenuti: limiti e vantaggi

3.1	Agente implementato con le fasi
3.2	Agente implementato con i moduli previdente
3.3	Agente implementato con i moduli e l'utilizzo di CF
3.4	Agente implementato con i moduli e l'utilizzo di Back-Tracking
3.5	Risultati finali

Capitolo 1

Planning in CLIPS

1.1 Descrizione del problema

Il problema richiesto prevedeva la costruzione di un sistema esperto che giochi ad una versione semplificata della famigerata Battaglia Navale. Il gioco è da intendersi in "solitario", per cui c'è solo un giocatore (ovvero solo il sistema esperto), che data una mappa 10x10 con alcune informazioni note, deve indovinare la distribuzione di più flotte di navi; nello specifico deve individuare:

- 1 nave da 4 caselle posizionata verticalmente o orizzontalmente;
- 2 navi da 3 caselle posizionate verticalmente o orizzontalmente;
- 3 navi da 2 caselle posizionate verticalmente o orizzontalmente;
- 4 navi da 1 casella;

Il problema prevede che tra una nave e un'altra ci siano celle contenute da acqua. Le informazioni note a priori sempre sono il numero di celle occupate da 'nave' per ogni riga e per ogni collona, mentre è possibile rivelare il contenuto di una cella prima di iniziare la risoluzione fino a un massimo di 4 celle.

Le azioni eseguibili dal sistema esperto saranno:

- **fire x y** permette all'agente di svelare il contenuto della cella con coordinate x e y;
- **guess x y** permette all'agente di mettere una bandierina nella cella con coordinate x e y che sta ad indicare una possibile cella con contenuto 'nave';
- **ungess x y** permette di ritrattare su alcune guess precedentemente effettuate;

CAPITOLO 1. PLANNING IN CLIPS

- **solve** viene invocato quando l'agente crede di aver risolto il problema e permette di calcolare uno scoring delle prestazioni dell'agente sulla base delle 3 operazioni descritte;

Inoltre, un ulteriore vincolo del sistema esperto è che si ha a disposizione un numero limitato di operazioni, nello specifico l'agente non potrà eseguire più di 5 fire e non potrà tenere contemporaneamente più di 20 celle con la bandierina.

1.2 Modellazione sistema esperto

Il gioco è già implementato in 3 moduli, di cui uno è l'**AGENT** che è la parte da implementare. Nel creare il sistema esperto per la risoluzione del problema richiesto, si sono validate diverse strategie, quindi diversi sistemi esperti, ognuno con i suoi pro e i suoi contro. Nello specifico si sono implementate 4 diversi sistemi esperti:

- sistema esperto completamente implementato in **FASI**;
- sistemi esperti implementati con i **MODULI**:
 - sistema esperto previdente ;
 - sistema esperto basato sul calcolo di **Certainty Factors**;
 - sistema esperto basato su tecniche di **Backtracking**.

Capitolo 2

Implementazioni

2.1 Implementazione in FASI

In questa implementazione si è deciso di utilizzare un solo modulo AGENT che rappresenta l'agente intelligente nel quale vengono eseguite, dal motore inferenziale, le regole di expertise.

L'inferenza ruota attorno a due componenti principali:

- fasi
 - rappresentate da fatti non ordinati (template)
- salience
 - permette alle regole di expertise che "appartengono" alla stessa fase, di essere eseguite nel momento corretto.

Funzionamento:

- quando l'agente prende una decisione nel modulo AGENT (fire, guess, unguess, solve) restituisce implicitamente il focus al modulo MAIN
- quando viene restituito il focus al modulo AGENT, vengono eseguite una serie di regole di expertise in modo tale da "catturare" i cambiamenti dovuti all'esecuzione di una decisione

2.1.1 Modellazione della conoscenza

Per la creazione del sistema esperto sono stati modellati dei fatti per la creazione della conoscenza. Ecco alcuni dei più importanti:

CAPITOLO 2. IMPLEMENTAZIONI

- **Fatti inziali:**

```
(k-water-per-row (row 0) (num 0))
(k-water-per-row (row 1) (num 0))
.
.
.
(k-water-per-col (col 0) (num 0))
(k-water-per-col (col 1) (num 0))
```

Rappresentano il numero di celle water per ogni riga e colonna. Inizialmente per tutte le righe e colonne il numero di celle water è 0.

- **Fatti non ordinati:**

– **phase:** Gli slot x e y vengono usati per riferirsi ad una precisa cella nella mappa, mentre i multislot row e col vengono usati per riferirsi ad una lista di celle. Una fase non avrà mai tutti gli slot con dei valori.

```
(deftemplate phase
  (slot phase (allowed-values left-to-right
                               right-to-left bot-to-top top-to-bot
                               hor-guesses-middle
                               ver-guesses-middle
                               execute-hor-guesses
                               execute-ver-guesses
                               set-guesses))
  (slot x)
  (slot y)
  (multislot row)
  (multislot col)
)
```

CAPITOLO 2. IMPLEMENTAZIONI

- **water**: Rappresenta una cella di acqua.

```
(deftemplate water
  (slot x)
  (slot y)
  )
```

- **computed-cell**: Per ogni cella della mappa viene tenuto traccia di una valore che determina quanto è possibile che lì ci sia un pezzo di nave

```
(deftemplate computed-cell
  (slot x)
  (slot y)
  (slot prob)
  )
```

- **last-position**: Template utilizzato per memorizzare l'ultima cella verificata da una determinata fase

```
(deftemplate last-position
  (slot x)
  (slot y)
  )
```

- **cell-around-water**: Template utilizzato per rappresentare una cella per la quale è stato asserto l'intorno di acqua

```
(deftemplate cell-around-water
  (slot x)
  (slot y)
  )
```

- **guess-on-middle**: Rappresenta un pezzo di nave middle per il quale è stata effettuata un'analisi

```
(deftemplate guess-on-middle
  (slot x)
  (slot y)
  )
```

- **k-water-per-row - k-water-per-col**: Tiene traccia del numero di celle water per ogni riga - colonna

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

CAPITOLO 2. IMPLEMENTAZIONI

```
)  
  
(deftemplate k-water-per-col  
  (slot col)  
  (slot num)  
)
```

2.1.2 Modellazione delle regole di expertise

Le regole di expertise possono essere suddivise in **macro-categorie** in base alla loro salience o fase alla quale fanno riferimento.

- **Regole con salience 10**

- **unset-around-water**: Rimuove i fatti errati che rappresentano posizioni di acqua;

- **Regole con salience 5**

- **set-around-water**: Asserisce vero l'intorno di acqua di un pezzo di nave noto che è stato già analizzato;

- **Regole con salience 0**

- **modify-k-per-row-col**: Se è presente un pezzo di nave noto, allora diminuisce di uno il numero di pezzi di nave sconosciuti presenti nella riga e nella colonna a cui fa riferimento la boat;
- **set-water-intersection-row**: Asserisce il fatto che ci sia acqua in una cella per la quale si ha un'intersezione con una riga vuota;
- **unset-computed-fired-cell**: rimuove il fatto che rappresenta una cella per la quale è stato calcolato una valore di possibilità se su questa cella è stata effettuata una azione di fire;
- **unset-water-computed-cell**: rimuove il fatto che rappresenta una cella per la quale è stato calcolato una valore di possibilità se questa cella rappresenta sicuramente una cella d'acqua;
- **unset-fired-guessed-cell**: rimuove una guess una cella per sulla quale viene effettuata un'azione di fire;

CAPITOLO 2. IMPLEMENTAZIONI

- **discovered-left**: Regola che fa scattare il meccanismo di left-to-right quando scopro una nuova left boat;

```
(defrule discovered-left
  (k-cell (x ?x) (y ?y) (content left))
  (not (cell-discovered (x ?x) (y ?y)))
  (not (phase (phase ?p)))
=>
  (assert (phase (phase left-to-right)))
  (assert (cell-discovered (x ?x) (y ?y)))
  (assert (last-position (x ?x) (y ?y)))
)
```

- **discovered-sub**: Asserisce come scoperta una cella contenente un sub;
- **discovered-right**: Regola che fa scattare il meccanismo di right-to-left quando scopre una nuova right boat
- **discovered-top**: Regola che fa scattare il meccanismo di top-to-boat quando scopre una nuova top boat;
- **discovered-bot**: Regola che fa scattare il meccanismo di bot-to-top quando scopre una nuova bot boat;

- **Regole FASE left-to-right**: Un insieme di regole per eseguire una serie di fire sicuri, partendo da una boat left nota;

- **start-left-to-right**: Esegue una fire alla destra dell'ultima posizione verificata dalla fase left-to-right, a patto che non ci sia una cella di nave già nota; modifica l'ultima posizione da verificare per la stessa fase;
- **left-to-right**: modifica l'ultima posizione da verificare per la stessa fase, nel caso in cui si passi per una cella già nota; non esegue alcuna fire;
- **end-left-to-right**: se l'ultima posizione da verificare contiene una cella right nota, allora la fase termina;

E' stata adottata una modellazione analoga per le regole appartenenti alle fasi di **right-to-left**, **top-to-bot** e **bot-to-top**;

Tutte le regole che fanno riferimento a queste fasi hanno **salience 5**;

CAPITOLO 2. IMPLEMENTAZIONI

- **Regole con salience -10**

- **discovered-middle:** Quando viene scoperta una nuova boat middle, vengono asserite due fasi per il posizionamento delle guess;

```
(defrule discovered-middle (declare (salience -10))
  (k-cell (x ?x) (y ?y) (content middle))
  (not (cell-discovered (x ?x) (y ?y)))
  =>
  (assert (phase (phase hor-guesses-middle) (x ?x) (y ?y)))
  (assert (phase (phase ver-guesses-middle) (x ?x) (y ?y)))
)
```

- **Regole FASE hor-guesses-middle:** Un insieme di regole che fanno in modo di eseguire due guess (alla sinistra e alla destra) di una boat middle nota;

- **fail-hor-guesses-middle:** Se non è possibile posizionare le guess in orizzontale allora la fase **hor-guesses-middle** viene rimossa;
- **hor-guesses-middle:** si passa alla fase **execute-hor-guesses** se:
 - * il numero di pezzi di nave sconosciuti nella riga è maggiore di quelli nella colonna;
 - oppure**
 - * non è possibile effettuare le guess in verticale;
- **execute-hor-guesses:** vengono eseguite le guess in orizzontale;
- **end-hor-guesses:** termina la fase **execute-hor-guesses** quando non ci sono più guess da eseguire;

E' stata adottata una modellazione analoga per le regole appartenenti alla fase **ver-guesses-middle** e **execute-ver-guesses**.

CAPITOLO 2. IMPLEMENTAZIONI

- **Regole deliberative e di calcolo:** Sono regole che hanno una **salience** molto bassa, in modo tale da essere eseguite quando non c'è più nessuna azione "sicura" da effettuare;
 - **compute-cell (salience -30)** Per ogni cella, che non sia una cella d'acqua certa, calcola un valore dato dalla seguente formula:

$$\frac{NR}{Dif_R} \times \frac{NC}{Dif_C} \quad (2.1)$$

dove:

NR = numero di pezzi di nave sconosciuti nella riga

NC = numero di pezzi di nave sconosciuti nella colonna

$Dif_R = 10 -$ numero di celle water certe nella riga

$Dif_C = 10 -$ numero di celle water certe nella colonna

- **unset-computed-cell (salience -30)** Per ogni cella per la quale è stato calcolato un nuovo valore, asserisce un nuovo punteggio dato dal massimo tra il valore precedente e il nuovo valore, in modo tale da ottenere un punteggio univoco;
- **fire (salience -50)** Esegue un'azione di fire sulla cella con il valore di possibilità più alto;
- **assert-average (salience -55)** Asserisce la media dei valori calcolata prendendo in considerazione tutte le celle per le quali è stato calcolato un valore;
- **guess (salience -60)** Se non ci sono più azioni fire da effettuare, effettua un'azione di guess su tutte quelle celle che hanno il valore di possibilità superiore alla media;

2.2 Implementazione in MODULI

2.2.1 Modellazione della conoscenza

Le tre soluzioni condividono grossomodo la stessa modellazione della conoscenza e alcune delle regole di expertise. In particolare, l'idea generale è quella di tenere una serie di template interni al modulo AGENT che permettono di tenere in memoria una rappresentazione interna (all'AGENT) del problema.

CAPITOLO 2. IMPLEMENTAZIONI

```
(deftemplate occell
  (slot x)
  (slot y)
  (slot check (default FALSE) (allowed-values FALSE TRUE))
  (slot status (default none) (allowed-values none occupied))
  (slot content (allowed-values unknow sure water left right middle top bot sub))
  Only for 2 -> slot certainty (default 0))
)
```

Permette di descrivere la mappa del problema, sono aggiunti tra i deffacts nella fase iniziale e permettono di creare lo stato di ogni cella (inizialmente sconosciuta). L'agente è così in grado di modellare il problema all'interno di tali celle; attraverso gli eventi (dati da una fire o evidenze iniziali) e le azioni (ragionamento e guess) il contenuto di ogni cella può cambiare status, così da permettere nuovi ragionamenti dell'agente. Per il secondo sistema c'è uno slot aggiuntivo per il fattore di certezza e i valori assegnabili allo slot **status** sono anche guess e unguess in quanto il ragionamento viene esteso anche al considerare tali operazioni e di calcolare il fattore di certezza anche in base a questi. Inoltre lo stato **sure**, ovvero uno stato in cui c'è una nave all'interno ma non si conosce quale parte specifica, è contenuto solo nella seconda soluzione.

```
( deftemplate ncol
  (slot col)
  (slot num)  )

( deftemplate nrow
  (slot row)
  (slot num)  )
```

Tali template hanno un duplice scopo: Nel caso del primo e del secondo agente essi permettono di tenere in memoria il numero totale di celle sconosciute per ogni colonna e per ogni riga, numero dato grazie ad una regola che permette di contarle, alle strutture; Nel caso del terzo agente vengono utilizzate per copiare i valori iniziali del numero di celle occupate da 'nave' per ogni riga e colonna che serviranno, insieme ad uno slot id, per tenere traccia di stati precedenti.

Altri tipi di template utilizzati per la modellazione sono:

- **mostguess**: lista di celle su cui fare una guess;

CAPITOLO 2. IMPLEMENTAZIONI

```
( deftemplate mostguess
  (multislot x)
  (multislot y) )
```

- **guessed**: lista di celle che sono in guess;

```
( deftemplate guessed
  (multislot x)
  (multislot y) )
```

- **mostfire**: insieme di celle che vengono scoperte che contengono una nave;

```
( deftemplate mostfire
  (multislot x)
  (multislot y) )
```

- **probfire**: lista di celle che non contengono sicuramente una nave, ma in caso di incertezza vengono colpiti.

```
( deftemplate probfire
  (multislot x)
  (multislot y) )
```

Dati questi template attraverso il comando **deffacts** si sono definiti fatti non ordinati per essi, dove inizialmente i valori sono tutti settati per default. Unico fatto ordinato utilizzato è **(phases ...)** che permette di indicare l'ordine e la ciclicità dei moduli; esso differisce per ogni implementazione.

2.2.2 Modellazione delle regole di Expertise

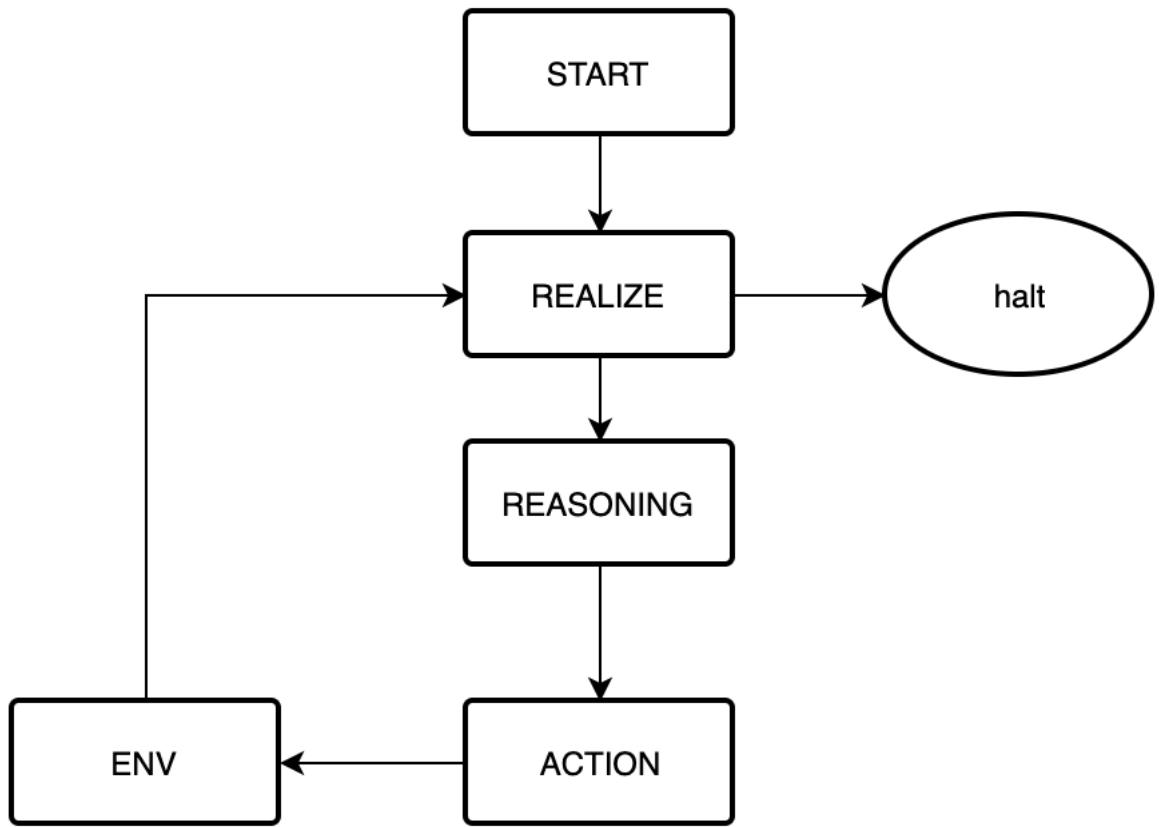
Le regole di expertise definite per il modulo **AGENT** sono quindi ulteriormente suddivise in moduli:

- **Sistema Esperto previdente e con CF**: condividono gran parte del problema, e sono suddivisi in **PERCEPTION**, **REASONING** e **ACTION**;

Per i primi due sistemi esperti si tratta di moduli che lavorano in ricorsione fra di loro; Il metodo **AGENT** permette di definire l'ordine di esecuzione dei moduli attraverso il fatto **(phases ...)** definito nei deffacts. Primo modulo a essere eseguito è quello di **PERCEPTION** ovvero di "percesione" dei cambiamenti avvenuti. Finite le regole applicabili il controllo passa nuovamente al modulo **AGENT** che passa il controllo al modulo di **REASONING**; il

CAPITOLO 2. IMPLEMENTAZIONI

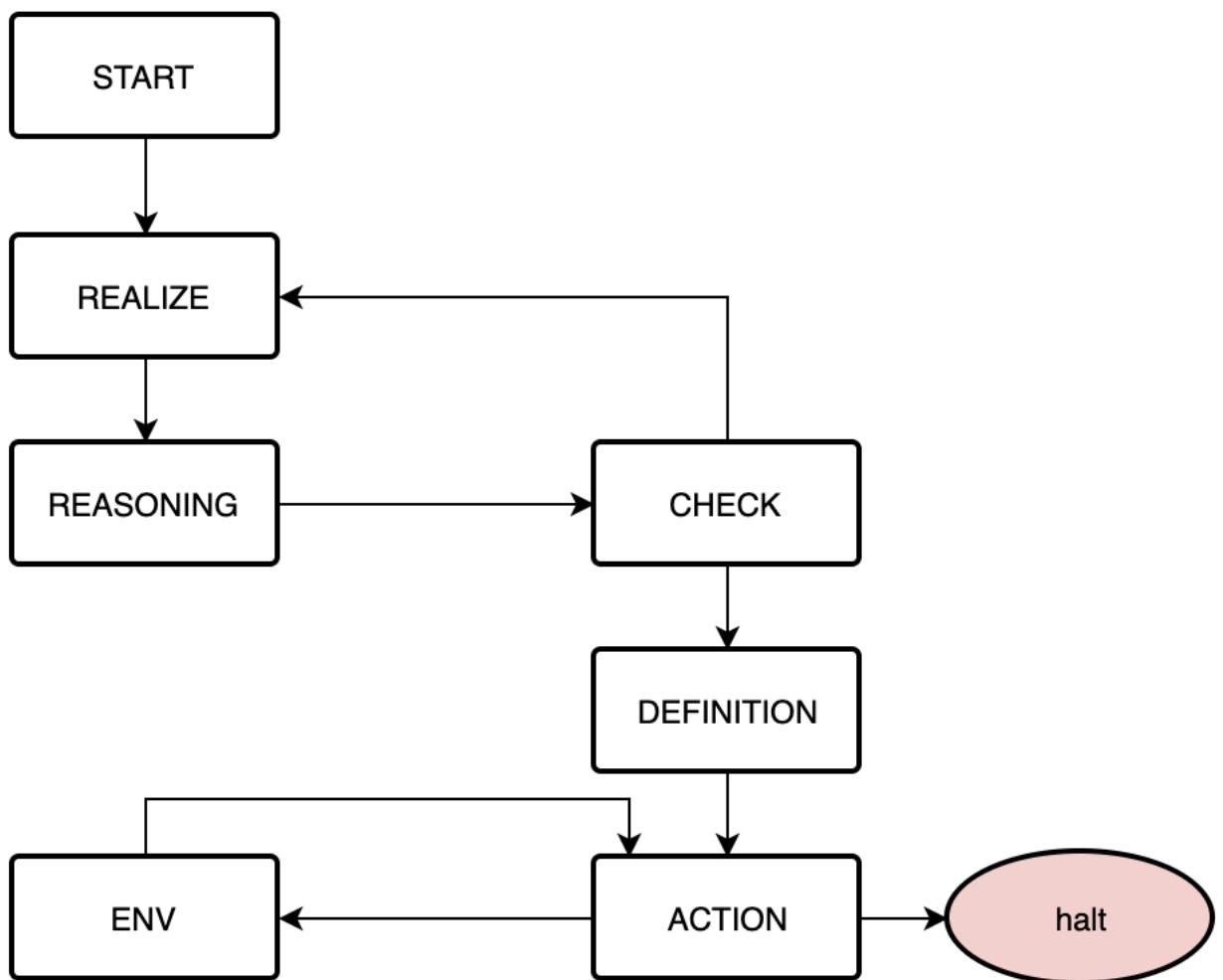
sistema esperto ragiona sulla base dei fatti conosciuti e stabilisce le mosse da eseguire scrivendole nei template. Tali template vengono letti e elaborati dal modulo successivo richiamato da **AGENT** (finite le operazioni di **REASONING**) ovvero il modulo **ACTION** in cui avviene l'effettiva asserzione dell'operazione exec e il passaggio, attraverso una (pop-focus), al modulo **ENV** che, una volta eseguito l'exec, ritornerà il controllo al metodo **AGENT** il quale permetterà un nuovo inizio del ciclo. Tale ricorsione termina non appena l'agente termina le regole applicabili nel modulo **ACTION**, in tal caso, in quanto ha esaurito le operazioni di fire e non sa più quali altre celle considerare per l'operazione di guess o non ne ha più a disposizione, scatta una particolare regola con salience ridotta la quale permette l'esecuzione della exec solve.



- **Sistema Esperto con Backtracking:** suddiviso in **START**, **REALIZE**, **REASONING**, **CHECK**, **DEFINITION** e **ACTION**. Nel terzo sistema esperto invece il funzionamento è differente; in questo caso l'agente inizia dal metodo **START**, utilizzato solo nella fase iniziale, che inizializza le variabili per il backtracking e passa il controllo al metodo **REALIZE** che, analogamente al metodo **PERCEPTION** del sistema precedente, rende sui template di rappresentazione le modifiche delle

CAPITOLO 2. IMPLEMENTAZIONI

osservazioni iniziali e del ragionamento dell'agente (importante notare che tra i template ci sono anche le navi da posizionare). Dopo aver terminato le regole applicabili il controllo passa al metodo di **REASONING** che effettua un planning nello stato successivo e passa il controllo al metodo **CHECK** che verifica se si è trovato un piano che porta alla soluzione, in caso negativo ritorna al **REALIZE** e continua ciclando su tutti i possibili punti di scelta. Quando viene trovato il planning che risolve il problema, si passa al modulo di **DEFINITION** che permette di codificare il piano in una lista e di passare al metodo **ACTION** che eseguirà il piano alternandosi con il modulo **ENV**.



CAPITOLO 2. IMPLEMENTAZIONI

Moduli del primo e del secondo sistema esperto

PERCEPTION

Nel modulo Perception ci sono regole di expertise che permettono al sistema esperto di aggiornare la conoscenza sulla mappa, sia inizialmente quando verranno resi noti i numeri di celle occupate per ogni riga e colonna, ed eventualmente le celle note a priori, successivamente invece, in base all'evoluzione dei moduli di Reasoning e Action. Questo modulo è condiviso dal primo e dal secondo sistema esperto ed è composto dalle seguenti regole, ad eccezione del fatto che il secondo presenta alcune regole in più e condizioni aggiuntive per tutte le regole per i nuovi status introdotti:

- **stopped**: scatta quando lo status dell'esecuzione è **stopped** e permette di terminare l'esecuzione con un halt;
- **reset**: scatta per tutte le celle sconosciute e permette di settare a 0 il contatore di celle e righe sconosciute (corrispondenti alle coordinate della cella) e permette di portare il campo **status** di ognuna a none, per essere nuovamente contatto dal prossimo modulo.
- **reveal**: scatta quando ci sono delle **k-cell** da rivelare alla mappa interna. Infatti permette di memorizzare nel template interno **occell** tutte le celle rese note dal modulo ENV;
- **revealn**: scatta quando c'è una cella che non è nello stato **water** o **unknow**, ovvero quando il contenuto della cella è noto essere un pezzo di nave. Permette di decrementare il numero di caselle occupate per riga e per colonna corrispondenti alle coordinate della cella.
- **revealFireSure**: scatta quando c'è una occorrenza di coordinate nel template **mostfire**. Permette di portare la cella nello stato **sure** ed eliminarla dai **mostfire**. Questa regola indica una differenza sostanziale tra il primo sistema esperto e il secondo; mentre nel primo tale regola scatta solo quando non ci sono più mosse fire disponibili, nel secondo agente serve proprio per il ragionamento opposto, ovvero le celle contenute in **mostfire** contengono un pezzo di nave, allora le identifica con lo stato **sure** sin da subito, e la inserisce nel template **mostguess** così da parsimoniare sulle operazioni di fire.
- **paddingFire**: scatta quando c'è una cella con stato **sure** e permette di portare allo stato **water** le celle in diagonale;
- **paddingSub**: scatta quando c'è una cella con stato **sub** e permette di portare al contenuto **water** le celle che la circondano;

CAPITOLO 2. IMPLEMENTAZIONI

- `paddingMiddleVert`: scatta quando c'è una cella con stato `middle` e con almeno una cella tra destra e sinistra contenute da acqua. Essa permette di portare al contenuto `water` le celle al lato destro e sinistro di tale cella `middle`;
- `paddingMiddleOrizz`: scatta quando c'è una cella con stato `middle` e con almeno una cella, subito sotto o sopra, contenute da acqua. Essa permette di portare al contenuto `water` le celle al lato inferiore e superiore di tale cella `middle`;
- `paddingTop`: scatta quando c'è una cella una con stato `top` e permette di portare al contenuto `water` le celle superiori e laterali di tale cella;
- `paddingBot`: scatta quando c'è una cella una con stato `bot` e permette di portare al contenuto `water` le celle inferiori e laterali di tale cella.
- `paddingRight`: scatta quando c'è una cella una con stato `right` e permette di portare al contenuto `water` le celle a destra e inferiori e superiori di tale cella.
- `paddingLeft`: scatta quando c'è una cella una con stato `left` e permette di portare al contenuto `water` le celle a sinistra e inferiori e superiori di tale cella.
- `findWater`: scatta quando il numero di navi da posizionare `k-per-row` o `k-per-col` è pari a 0. Essa permette di portare allo stato `water` le celle di tale riga/colonna.

REASONING

Nel modulo di Reasoning ci sono regole di expertise che permettono al sistema di individuare pezzi di navi e di fare supposizioni su alcune celle andando così a popolare i template `mostguess` e `mostfire`. Le principali regole condivise dai suoi sistemi sono:

- `firevert`: scatta quando c'è una cella con stato `middle` e con almeno una cella tra destra e sinistra contenute da acqua. Essa permette di inserire nel template `mostfire` le celle subito sopra e subito sotto della cella `middle`;
- `fireorizz`: scatta quando c'è una cella con stato `middle` e con almeno una cella, subito sotto o sopra, contenute da acqua. Essa permette di inserire nel template `mostfire` le celle subito a destra e subito a sinistra della cella `middle`;
- `topfire`: scatta quando c'è una cella con stato `top` e permette di inserire nel template `mostfire` la cella subito inferiore;
- `botfire`: scatta quando c'è una cella con stato `bot` e permette di inserire nel template `mostfire` la cella subito superiore;

CAPITOLO 2. IMPLEMENTAZIONI

- **rightfire**: scatta quando c'è una cella con stato **right** e permette di inserire nel template **mostfire** la cella subito a sinistra;
- **leftfire**: scatta quando c'è una cella con stato **left** e permette di inserire nel template **mostfire** la cella subito a destra;
- **conrig**: scatta per ogni cella con stato **unknow** nella mappa e permette di aggiornare i template **nrow** e **ncol**, ovvero del numero di celle sconosciute per ogni riga e per ogni colonna, incrementandoli (da notare l'utilizzo della salience per permettere il calcolo su tutte le celle prima di proseguire con le operazioni successive);
- **mostfirecol** e **mostfirerow**: scatta quando il numero di navi da posizionare **k-per-row** o **k-per-col** è pari al numero **nrow** o **ncol** rispettivamente. Essa permette di inserire nel template **mostfire** le celle di tale riga/colonna;
- **guessmiddle**: scatta quando c'è una cella con stato **middle** e tutte le celle nel suo intorno sono sconosciute. Permette di inserire nel template **mostguess** le celle al lato inferiore, superiore e laterali della cella **middle** così da assicurare il 50% di scelta giusta
- **mostguessrow** e **mostguesscol**: Tale regola rappresenta una differenza per i due sistemi. Nel primo sistema esperto tale regola scatta quando una riga/colonna ha numero di celle sconosciute, **nrow** e **ncol**, superiore di al massimo 2 rispetto al numero di celle che contengono effettivamente una nave per tale riga/colonna. Essa quindi permette di inserire nel template **mostguess** le celle di tale riga/colonna considerando sempre un errore di 1 o 2 celle; nel secondo sistema esperto invece tale regola scatta solo un numero predefinito di volte (attraverso il fatto **count**) e per diverse righe/colonne. Per ognuna di esse selezionerà la cella con CF maggiore e verrà inserita nel template **mostguess**.

Inoltre per il secondo sistema si è aggiunto una ulteriore e fondamentale regola:

- **computeCertainly**: permette di calcolare per ogni cella un fattore di certezza calcolato in base alle evidenze note sulle righe e le evidenze note sulle colonne. Il calcolo di certezza per una cella, dato il numero di celle sconosciute nella corrispondente riga/colonna e il numero di celle effettivamente occupate da nave in tale riga/colonna, è il seguente:

$$1/(numeroCelleSconosciute - numeroCelleConNave)$$

Tale calcolo viene effettuato sia per le righe che per le colonne. In questo modo avremo un valore inversamente proporzionale; minore è la distanza maggiore

CAPITOLO 2. IMPLEMENTAZIONI

sarà il risultato, maggiore sarà la differenza minore sarà il risultato. Essendo due i fattori di certezza per una singola cella, uno per la colonna ed uno per la riga si sono combinati i due valori ottenuti con il metodo di combinazione di due CF ovvero:

$$(CF1 + CF2) - (CF1 * CF2)$$

Il calcolo dei CF viene sempre sommato al calcolo dei CF precedente.

- **AVGCertainty:** Permette di calcolare una soglia minima per il CF, calcolando la somma totale e dividendo per il numero di celle sconosciute.

ACTION

Nel modulo Action ci sono regole di expertise che permettono al sistema di eseguire le 4 operazioni principali e soprattutto di effettuare le operazioni utilizzando i template `mostguess`, `mostfire` e `probfire`, per il secondo anche `unguess`. Le regole in comune ai due sistemi sono:

- **notguess:** scatta se nel template `mostguess` c'è una cella che non risulta essere né `unknow` e né `sure`, oppure se è già stata eseguita una `guess` su di essa. Permette di eliminare dal template tale cella;
- **guess:** scatta se nel template `mostguess` c'è una cella e se su tale cella non è stato precedentemente effettuata una `guess`. Essa asserisce una `exec guess` di tale cella e la elimina dal template precedente inserendola nel template `guessed` effettuando in fine un `pop-focus` per rilasciare il funzionamento del modulo e lasciare l'esecuzione all'ENV;
- **unguess:** Tale regola scatta quando una cella nello stato `guess` cambia il suo contenuto da `unknow` a `water` per via di ragionamenti dei moduli precedenti. Nel sistema esperto con i CF tale regola scatta quando sono presenti delle coordinate nel template `unguess`. Asserisce una `exec unguess` di tale cella presa dal template `guessed` dalla quale verrà eliminata. Anche quest regola rilascia il modulo con un `pop-focus`.
- **finish:** scatta quando il sistema esperto pensa di aver risolto il problema o di aver fatto tutti i possibili ragionamenti per scoprire il contenuto delle celle. Infatti permette di asserire una `solve` e di rilasciare il modulo con una `pop-focus`.

Importante notare che tali regole ad eccezione della `finish`, hanno una salince maggiore rispetto alle altre, in quanto le operazioni di `guess` e soprattutto di `unguess` devono prevalere sul resto. Alcune regole presenti solo per il primo sistema:

CAPITOLO 2. IMPLEMENTAZIONI

- **randomUnguessToFire**: scatta quando il sistema non sa più su quali celle effettuare una guess e non ha certezza su un cella da sparare ma ha ancora delle fire a disposizione, in tal caso seleziona una cella contenuta in `guessed` casualmente. Asserisce una `exec unguess` di tale cella e la inserisce nel template `probfire` e infine permette di rilasciare il modulo con pop-focus;
- **fireProbably**: scatta quando non ci sono fire sicure, o guess da eseguire, e se all'interno del template `probfire` c'è una coordinata. Essa permette di asserire una `exec fire` di tale cella e di rilasciare il modulo con pop-focus;
- **firesure**: scatta quando non ci sono guess o unguess da fare e nel template `mostfire` c'è una cella da sparare. Essa permette di asserire una `exec fire` di tale cella e di rilasciare il modulo con pop-focus;
- **firerisk**: scatta quando non si hanno celle in `guess` e non si hanno celle sicure ovvero in casi di massima incertezza, seleziona una cella casuale e permette di asserire una `exec fire` di essa rilasciando il modulo con pop-focus;

Alcune regole presenti solo per il secondo sistema:

- **findUnguess** e **findUnguessAVG**: queste due regole scattano quando una cella nello stato `guessed` o è diventata water certa per via di ragionamenti precedenti o il suo fattore di certezza non supera la soglia minima consentita. Tali coordinate vengono inserite nel template `unguess`.
- **unguessToFire**: questa regola scatta quando non ci sono più guess da eseguire o non se ne possono eseguire più ma ci sono ancora delle fire a disposizione. In tal caso prende una cella contenuta in `guessed`, scelta in base al maggior numero di certezza fra le celle in `guessed`, e permette di asserire una `exec unguess` di tale cella, inserendola nel template `probfire`. Anche in questo caso viene rilasciato il modulo con una pop-focus;
- **fireProbably**: scatta quando non ci sono guess o unguess da fare e nel template `probfire` c'è una cella. Permette di asserire una `exec fire` di tale cella e di eliminarla dal template precedente rilasciando in fine il modulo con una pop-focus;
- **finallyFire**: scatta quando l'agente pensa di essere arrivato alla soluzione ancor prima di terminare tutte le fire a disposizione. In tal caso seleziona una cella con stato `sure` ed asserisce una `exec fire` rilasciando il controllo del modulo con una pop-focus.

In tutti i moduli, si è utilizzata la salience per permettere una corretta esecuzione ordinata delle regole.

CAPITOLO 2. IMPLEMENTAZIONI

La differenza sostanziale tra il primo e il secondo sistema esperto sta nel fatto che mentre il primo esegue subito una **fire** non appena sa che è sicura, il secondo invece, per le celle sicure non effettua una fire, ma la simula andando ad aggiornare i valori come se fosse una fire andata a buon fine. Mentre il primo sistema cerca di agire con le fire solo data assoluta certezza e di cercare di capire al meglio le guess restanti, il secondo è più conservatore per le fire e prima di agire prova ad allocare delle guess in base ai valori di certezza e solo dopo per rendersi conto della realtà, effettua la fire della cella più certa, ma sconosciuta.

Moduli del terzo sistema esperto

Prima di procedere con la descrizione dei moduli, è importante precisare che in CLIPS una soluzione che fa affidamento a tecniche di backtracking per la ricerca di una soluzione è una soluzione forzata che va più verso dei problemi CSP-like. Nonostante ciò, grazie al metodo di funzionamento di Clips, implementare una soluzione con backtracking diventa quasi naturale. Tale meccanismo è dato dall'utilizzo di uno slot aggiuntivo per tutti i fatti del problema chiamato **id** che permette di identificare tutti i fatti che appartengono ad uno stato specifico. Con l'avanzare dei ragionamenti del sistema e quindi con l'esplorazione in profondità delle possibili risoluzioni, verranno man mano prodotti delle evoluzioni degli stati, il sistema effettua delle scelte, segnando sia di aver fatto quella determinata scelta in quello specifico punto (o **id**) e sia incrementando il valore di **id** ad ogni nuova scelta. Se il sistema dovesse arrivare ad un vicolo cieco, ovvero una soluzione parziale che non può portare alla soluzione finale richiesta (rappresentata da una regola contenuta nel modulo **CHECK**), torneranno attivi in WM i fatti e le regole attivabili nei precedenti punti di scelta, lasciando quindi la possibilità di scartare quello precedentemente elaborato. Viene assicurato che le stesse scelte non vengano ri-effettuate e il sistema procede a tentativi provando tutte le possibili strade per la ricerca della soluzione. Questa rappresenta un tecnica di Planning Goal-Driven. In virtù di tale funzionamento si è modificata la modellazione della conoscenza andando ad aggiungere per ogni template anche lo slot **id**, per il campo, ovvero la mappa interna dell'agente inoltre, si è aggiunto uno slot chiamato **pid** che permette di tenere traccia del precedente punto di scelta dal quale continua l'**id** attuale. Inoltre si sono aggiunti due nuovi template:

- **count**: permette tenere il conto dell'**id** in modo tale che ogni volta sia unico.

```
( deftemplate count
  (slot c) )
```

CAPITOLO 2. IMPLEMENTAZIONI

- **ship**: permette di descrivere le navi, verranno infatti definiti con **deffacts** tutte le possibili tipologie di navi da posizionare con relativo nome univoco, **id**, e se è stato assegnato o meno nel percorso considerato:

```
( deftemplate ship
  (slot id (default 0))
  (slot name)
  (multislot x)
  (multislot y)
  (slot dim)
  (slot assegned (default FALSE)) )
```

- **chosen**: permette di tenere traccia delle scelte di posizionamento di una specifica nave in un determinato punto di scelta:

```
( deftemplate chosen
  (slot id)
  (slot dim)
  (multislot x)
  (multislot y) )
```

è creato anche un template chiamato **chosen** che permette appunto di tenere traccia delle scelte precedentemente già effettuate.

START

Il metodo START permette inizializzare la mappa interna dell'agente, più nel dettaglio permette di rivelare eventuali celle già conosciute e di rivelare il numero di celle occupate per ogni riga ed ogni colonna. Inoltre tale modulo permette di passare il controllo a realize modificando il fatto (**phases...**) eliminando il modulo START permettendo così l'inizio del ciclo di ricerca della soluzione.

REALIZE

Il metodo REALIZE funziona pressoché nello stesso modo di Perception dei primi due sistemi esperti. Infatti esso trova tutte le possibili celle contenenti acqua attraverso le regole di **padding** e di **findWater** con l'unica differenza che in questo caso si prende prima di tutto la cella più recente (in WM) con corrispondente **Id** e **Pid** e per tutte le celle con id pari a **Id** (di cui quindi si conosce il contenuto preciso), prende la corrispondente cella nel piano precedente (quindi con id pari a **Pid**) e la si riscrive con **Id** nuovo e contenuto aggiornato. Ciò che invece non è presente in questo metodo rispetto al metodo precedentemente descritto sta nell'eliminazione delle regole per il reset e di reveal e nell'aggiunta invece delle seguenti regole:

CAPITOLO 2. IMPLEMENTAZIONI

- **realizer**: scatta, dopo aver effettuato le operazioni di ritrovamento di water, sulla cella con **Id** e **Pid** più recente e permette di riportare tutta la conoscenza delle celle con id equivalente a **Pid**, nel nuovo stato con id appunto **Id**.
- **reaizerShip**: scatta sulla cella con **Id** e **Pid** più recente e permette di riportare tutta la conoscenza delle navi e dei loro eventuali assegnamenti passati, quindi con id equivalente a **Pid**, nel nuovo stato con id appunto **Id**.
- **cleanMemory**: scatta quando, presa la cella con **Id** e **Pid** più recente, la differenza tra tale questi due id è diversa da 1. In tal caso vuol dire che il ramo precedente è stato scartato e la produzione del piano procederà incrementando l'id. Permette di eliminare (retract) tutti i template popolati per gli stati precedentemente elaborati, ovvero tutti quei fatti in memoria che hanno id maggiore di **Pid** e con id minore di **Id**.

REASONING

Il metodo REASONING ha un funzionamento completamente differente rispetto al suo omonimo descritto prima. Infatti, essendo questa soluzione goal-driven, in questo modulo abbiamo delle regole che permettono di trovare delle possibili configurazioni di assegnazione per ogni nave da assegnare. Ad ogni assegnamento però viene effettuato un (**pop-focus**) che permette di passare al modulo CHECK. Le regole che permettono di fare questi assegnamenti sono:

- **find_shipx4orizz**: permette di trovare una configurazione per il posizionamento della nave da 4 caselle in verticale se non precedentemente assegnata in quella soluzione parziale, prima di passare il controllo al prossimo modulo asserisce la scelta effettuata (così da non ripeterla), incrementa il contatore e asserisce nuovi fatti per le celle occupate dalla nave con Id aggiornato;
- **find_shipx4vert**: funzionamento analogo al precedente ma trova una configurazione in orizzontale;
- **find_shipx3vert**: funzionamento analogo al precedente ma trova una configurazione in verticale per una nave da 3 caselle;
- **find_shipx3orizz**: funzionamento analogo al precedente ma trova una configurazione in orizzontale per una nave da 3 caselle;
- **find_shipx2vert**: funzionamento analogo al precedente ma trova una configurazione in orizzontale per una nave da 2 caselle;
- **find_shipx2orizz**: funzionamento analogo al precedente ma trova una configurazione in orizzontale per una nave da 2 caselle;

CAPITOLO 2. IMPLEMENTAZIONI

- `find_sub`: funzionamento analogo al precedente ma trova una configurazione per una nave da 1 caselle;

Importante precisare che le regole definite in questo modulo possono scattare solo se tutte le navi della dimensione superiore sono già state assegnate. Il ragionamento parte dall'assegnamento delle navi di 4 caselle per poi procedere con le navi da 3 caselle, alle quali seguiranno quelle da 2 e solo dopo aver trovato l'allocazione per tutte le navi da 2, si cerca di allocare i sottomarini. Tale metodologia di risoluzione rispetta un po' l'euristica chiamata FAF (Fewer Alternatives First), risolviamo prima i "difetti" con meno resolvers, ovvero posizioniamo le navi con meno configurazioni possibili.

CHECK

Il metodo CHECK permette di verificare se l'id attuale è una soluzione al problema, in particolare ha la regola:

- `are_we_done`: permette di verificare se si è giunti ad una soluzione o meno, se tutte le navi sono state posizionate e il numero di celle da occupare per ogni riga e per ogni colonna è pari a 0 allora passa il controllo al metodo DEFINITION asserendo il numero di Id della soluzione che ha attivato la regola con un fatto non ordinato `last`. Se invece non è ancora giunta la soluzione, trasferisce il comando al metodo REALIZE. Il ciclo così creato permette di effettuare backtraking e di giungere infine ad una soluzione.

DEFINITION

Il metodo DEFINITION permette definire le operazioni da effettuare per poter eseguire il piano sviluppato dai metodi precedenti e con id identificato dal fatto `last`:

- `deffire`: permette di inserire nel template `mostfire` tutte le coordinate delle navi descritte dalla soluzione con id `last` ;
- `done`: permette passare al metodo ACTION.

ACTION

Il metodo ACTION ha la stessa funzionalità di quello omonimo descritto precedentemente, differisce solo nella semplicità in quanto in questo contesto dovrà solo eseguire le operazioni una per volta di fire e guess fino a svuotare la lista `mostfire` e subito dopo di asserire la `solve` per ottenere il puneggio e terminare il gioco.

Capitolo 3

Risultati ottenuti: limiti e vantaggi

Tutti i sistemi esperti presentati sono stati testati con le stesse mappe, così da poter effettuare una sperimentazione dei risultati. Le mappe create variano in base alla semplicità/complessità e in base al numero di osservazioni iniziali date al sistema.

Inoltre è stato creato un tool implementato con le librerie JavaFX per la visualizzazione grafica dei risultati. Nella rappresentazione dei risultati le celle con i bordi rossi rappresentano le informazioni iniziali.

Ancor prima di valutare i risultati ottenuti sulle mappe è importante precisare quanto sia sembrata conveniente la suddivisione in moduli dell’agente in quanto, con tale suddivisione si permette un controllo più stabile delle operazioni da eseguire, cosa che invece nella organizzazione a fasi è più difficile ottenere. I risultati però mostrano pro e contro tra i vari sistemi.

3.1 Agente implementato con le fasi

I punteggi ottenuti variano, a seconda della posizione delle navi sconosciute. Questo agente non sembra risentire in maniera estremamente significativa del numero e della tipologia di celle note all’inizio, questo perchè in assenza di informazioni calcola un valore matematico per ogni cella ed effettua l’azione di fire su quella più sicura. La natura della agente fa sì che le azioni di fire vadano quasi sempre a buon fine, mentre le azioni di guess risultino meno precise, ma comunque nella maggior parte efficaci; in conclusione questo agente sbaglia poche volte le fire e nella maggior parte delle volte le guess nelle posizioni giuste sono in numero maggiore di quelle nelle posizioni sbagliate.

CAPITOLO 3. *RISULTATI OTTENUTI: LIMITI E VANTAGGI*

3.2 Agente implementato con i moduli previdente

I punteggi ottenuti variano soprattutto in base alla complessità della mappa poichè una mappa molto complessa risulterà con molte meno possibilità di ragionamento dell'agente. Un ulteriore problema riscontrato è dato dalla mancata evidenza di celle di navi nella fase iniziale. Infatti senza di esse il sistema a ben poco su cui ragionare; nella maggior parte dei casi effettua delle fire casuali che fanno scendere il punteggio drasticamente. Nei casi invece di conoscenze fornite, l'agente, effettua subito le fire delle celle conosciute e quindi, cercando di rimandare l'incertezza delle fire (con una strategia più conservativa), riesce ad ottenere un punteggio discreto, senza sbagliare le guess ma lasciando molti pezzi di navi liberi (sink). Tende più a rivelare e a eseguire ciò che sà per certo, senza osare.

3.3 Agente implementato con i moduli e l'utilizzo di CF

I punteggi ottenuti con l'utilizzo dei CF in alcuni casi a dimostrato un netto vantaggio rispetto all'agente precedente, soprattutto nelle situazioni in cui la conoscenza era minore. Anche nelle situazioni di alta osservabilità risulta comportarsi meglio in quanto riesce a ragionare su più punti i vista, considerando appunto il CF, ma anche rischiando di più, in quanto lascia che le fire sicure vengano markate come guess e le simula per i conteggi come fire. In questo modo il ragionamento dell'agente riesce ad andare oltre e a continuare il ragionamento anche senza più fire disponibili. È importante notare come in alcuni casi il valore dei CF risulta troppo valorizzante per una determinata area della mappa a discapito di zone più povere che però vengono scartate.

3.4 Agente implementato con i moduli e l'utilizzo di Back-Tracking

I punteggi ottenuti con l'utilizzo della tecnica di Back-Tracking risulta essere quella con i punteggi più alti. Questo è sicuramente dovuto al fatto che le navi da posizionare sono fornite come fatti e su una mappa 10x10, con determinati numeri per le righe e per le colonne, il numero di possibili configurazioni di tali navi non è un numero troppo elevato e in alcuni casi è anche unico. Molto importante quindi notare che anche qui le osservazioni giocano un ruolo importante, in quanto senza osservazioni il numero di possibili configurazioni potrebbe aumentare, mentre con determinati posizionamenti, la ricerca potrebbe essere ristretta. Un contro molto influente però di tale soluzione è il tempo di esecuzione in quanto il Back-Tracking

CAPITOLO 3. RISULTATI OTTENUTI: LIMITI E VANTAGGI

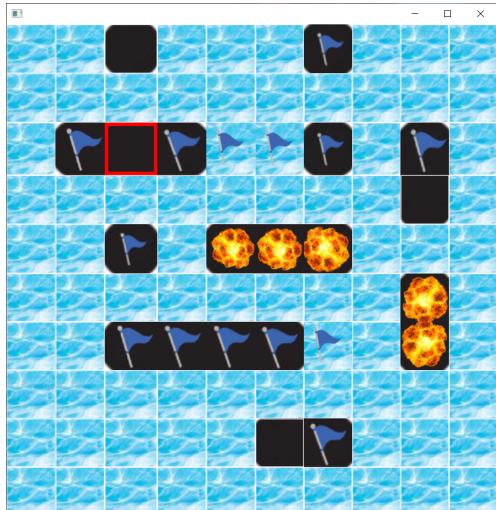
cronologico che ci permette di sfruttare CLIPS è non informato in nessun modo ed effettua le scelte in base all'ordine in WM. Utilizzando l'euristica FAF considerando come vincoli più semplici (con meno alternative) le navi più grandi, si è riuscito a diminuire di molto il tempo di esecuzione (che rimane comunque un tempo elevato). Un ulteriore problema riscontrato è quello di occupazione di memoria in quanto ogni sotto-ramo inizialmente non veniva cancellato. Si è risolto tale problema con la regola cleanMemory descritta nell'Implementazione.

3.5 Risultati finali

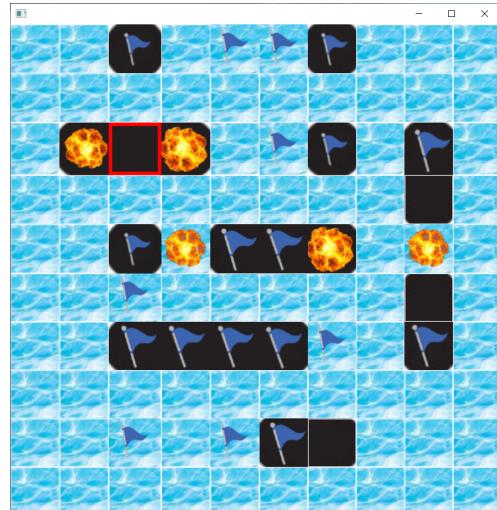
Esempio 1:

Numero osservazioni iniziali: 1

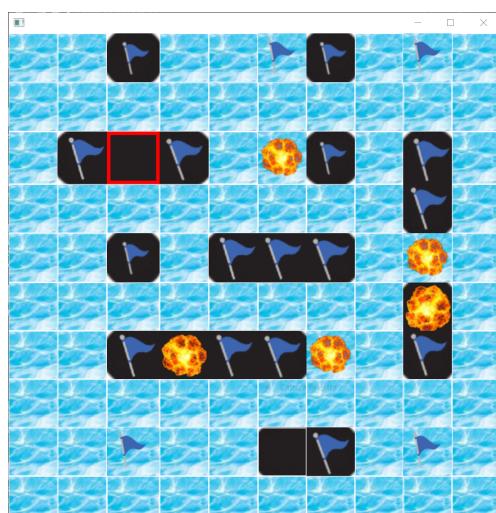
Complessità: facile



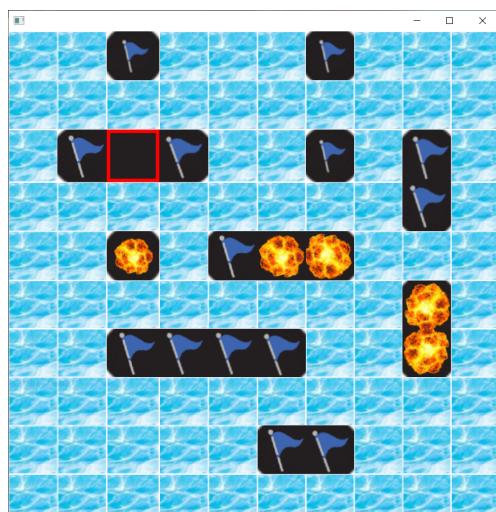
(a) Agente con le fasi
Punteggio: 190



(b) Agente previdente
Punteggio: 80



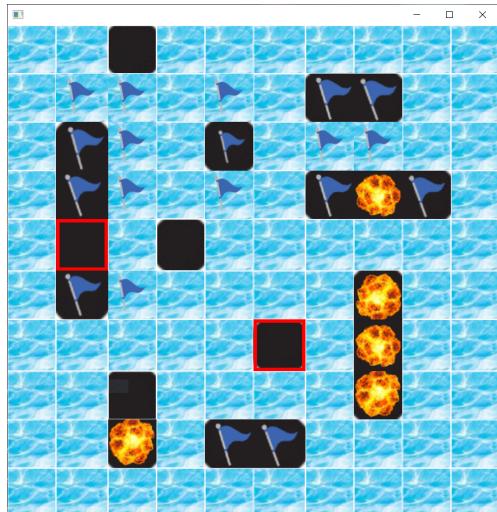
(c) Agente con CF
Punteggio: 170



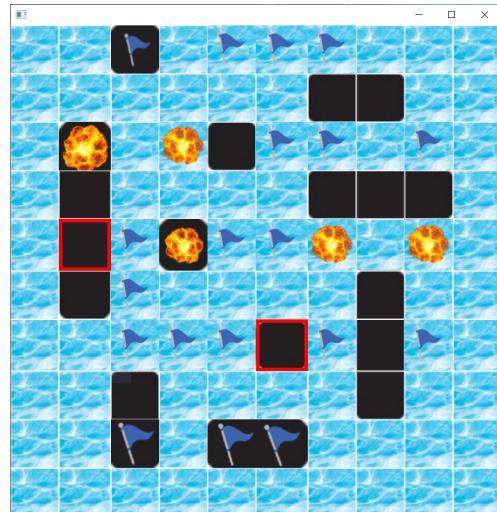
(d)
Agente con Back-Tracking
Punteggio: 340

CAPITOLO 3. RISULTATI OTTENUTI: LIMITI E VANTAGGI

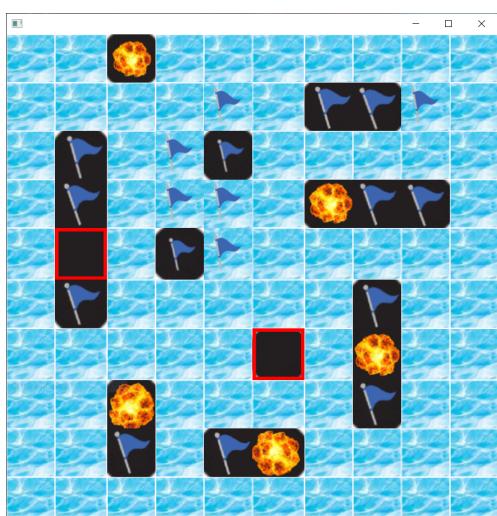
Esempio 2:
Numero osservazioni iniziali: 2
Complessità: Media



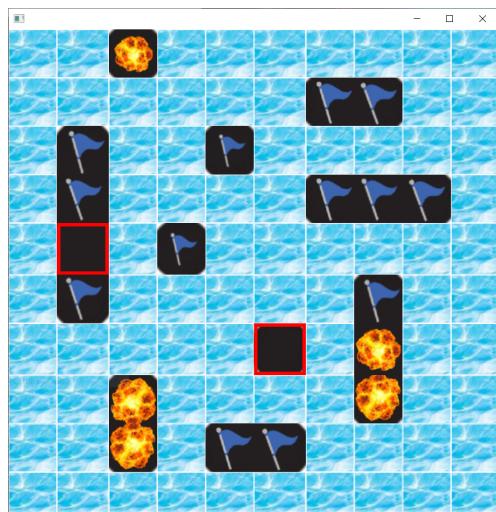
(a) Agente con le fasi
Punteggio: 75



(b) Agente previdente
Punteggio: 15



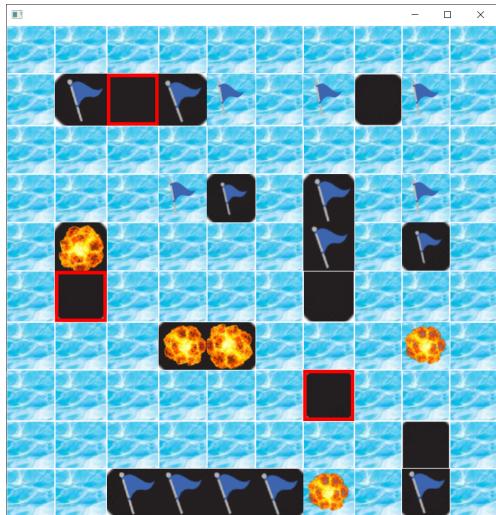
(c) Agente con CF
Punteggio: 240



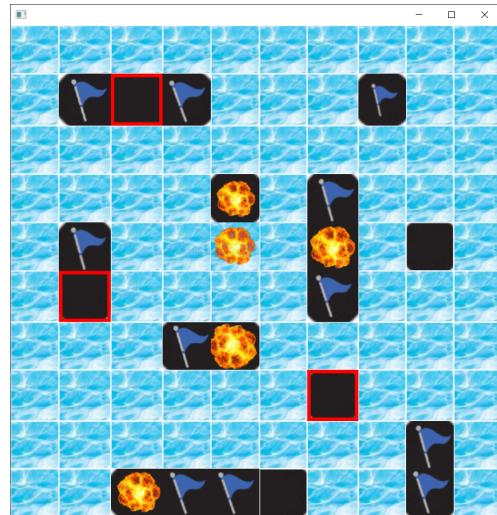
(d)
Agente con Back-Tracking
Punteggio: 330

CAPITOLO 3. RISULTATI OTTENUTI: LIMITI E VANTAGGI

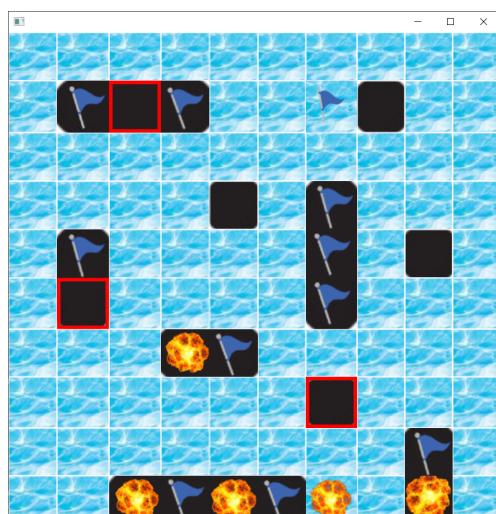
Esempio 3:
Numero osservazioni iniziali: 3
Complessità: Media-Difficile



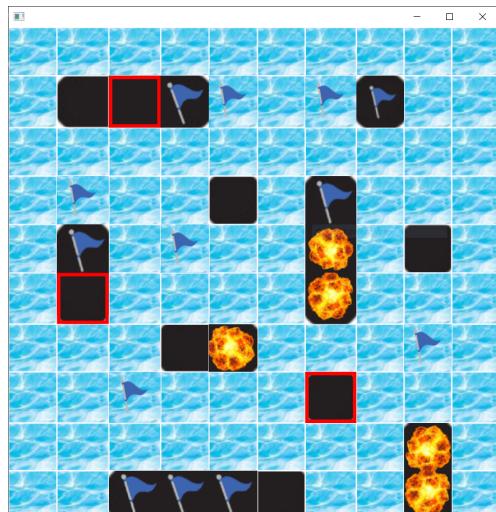
(a) Agente con le fasi
Punteggio: 90



(b) Agente previdente
Punteggio: 140



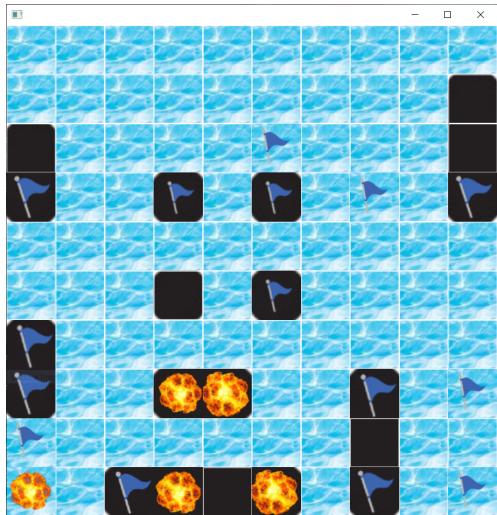
(c) Agente con CF
Punteggio: 175



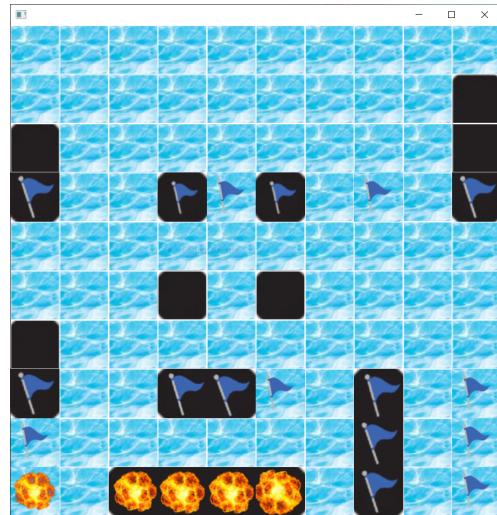
(d)
Agente con Back-Tracking
Punteggio: 75

CAPITOLO 3. RISULTATI OTTENUTI: LIMITI E VANTAGGI

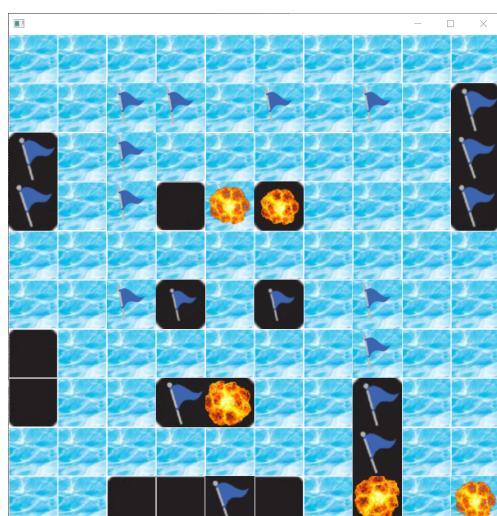
Esempio 4:
Numero osservazioni iniziali: 0
Complessità: Difficile



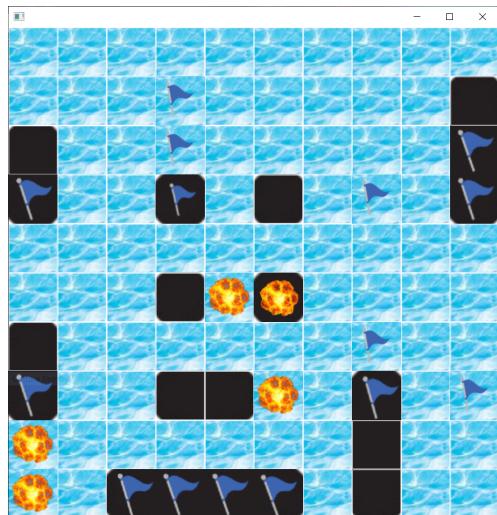
(a) Agente con le fasi
Punteggio: 195



(b) Agente previdente
Punteggio: 25



(c) Agente con CF
Punteggio: -60



(d)
Agente con Back-Tracking
Punteggio: -110