

# Intelligenza Artificiale e Laboratorio - Planning e Sistemi a Regole

---

## Indice:

---

- Progetto d'esame: Battaglia Navale singolo giocatore in linguaggio CLIPS
- Pre-requisiti
- Descrizione del sistema esperto sviluppato
  - Descrizione moduli
  - Descrizione stati
  - Saliency
- Implementazione
  - Agent
  - Init
  - Update Knowledge Control
  - Deliberation
    - Sistema esperto dump
    - Sistema esperto intelligente
  - Action
- Formule del progetto
  - Formula probabilità su singola cella
  - Formula ultima discesa
  - Formula original score (probabilità a priori)

## Progetto d'esame: Battaglia Navale singolo giocatore in linguaggio CLIPS

---

(docente: Roberto Micalizio)

L'obiettivo del progetto è quello di sviluppare un sistema esperto che giochi ad una versione semplificata della famigerata *Battaglia Navale*.

Come di consueto le navi da individuare sono le seguenti:

- **1 corazzata** da **4 caselle**
- **2 incrociatori** da **3 caselle** ciascuno
- **3 cacciatorpedinieri** da **2 caselle** ciascuno
- **4 sottomarinieri** da **1 casella** ciascuno

Nel documento [progetto\\_CLIPS-2022-2023.pdf](#) è possibile reperire la traccia completa.

## Pre-requisiti

---

E' necessaria l'installazione di **CLIPS** presso il [sito ufficiale](#). Per questo progetto è stata utilizzata la versione **6.4.1**.

**NOTA:** I risultati potrebbero variare nella versione precedente a causa di implementazioni diverse. Principalmente è dovuta alla gestione della rifrazione.

## Descrizione del sistema esperto sviluppato

---

**NOTA:** I link in questa sezione fanno riferimento al [6\\_Deliberation\\_dump.cl](#) solo come esempio. Il funzionamento rimane invariato anche in [6\\_Deliberation\\_intelligent.cl](#).

## Descrizione Moduli

I sistemi esperti implementati hanno i seguenti MODULI:

- **Moduli di sistema pre-implementati:**
  - **MAIN:** gestisce lo **SCHEDULING** tra i moduli di **AGENT** e i moduli del sistema pre-implementati(**MAIN** e **ENV**);
  - **ENV:** gestisce la **CONOSCENZA DI DOMINIO** in caso di eventi provenienti dagli agenti.
- **Moduli degli agenti:**
  - **AGENT:** come il Main per l'ambiente, gestisce lo scheduling tra i moduli all'interno degli agenti

- **INIT**: inizializza la **CONOSCENZA DI CONTROLLO** travasando la **CONOSCENZA DI DOMINIO** all'interno di essa. Viene utilizzato solo al primo avvio dell'agente.
- **UPDATE KNOWLEDGE CONTROL**: ha il compito di aggiornare la **CONOSCENZA DI CONTROLLO** ogni qualvolta che viene eseguita un'azione sull'ambiente. Gestisce il **FRAME PROBLEM** all'interno di questo dominio.
- **DELIBERATION (DUMP)**: gestisce la strategia di deliberazione **dump**, ossia senza particolare intelligenza. Quando arriva al fondo di una ricerca senza aver impostato tutte le *guess*, fa backtracking fino all'ultima azione GREEDY e ramifica la ricerca verso un altro ramo.
- **DELIBERATION (INTELLIGENT)**: gestisce la strategia di deliberazione **intelligente**, ossia effettuando il backtracking informato. Questo consiste nella ricerca della casella con il punteggio maggiore iniziale all'interno delle caselle in stato **EXPLORE**.

## Descrizione Stati

I sistemi esperti implementati hanno degli STATI, composti da **fatti ordinati**. Sono stati definiti all'interno dei moduli di **DELIBERATION** sono stati definiti i seguenti stati:

- **GREEDY**: vengono eseguite le *guess* proveniente da una conoscenza pregressa avuta input, oppure viene effettuata una RICERCA INFORMATATA posizionando da subito delle *fire* e restringere lo SPAZIO DI RICERCA MENO INFORMATO;
- **EXPLORE**: entriamo in uno SPAZIO DI RICERCA MENO INFORMATO, in cui posizioniamo le *guess* al meglio che possiamo, sfruttando le regole di risoluzione definite nei MODULI di DELIBERAZIONE;
- **BACKTRACKING**: vengono effettuate le *unguess* sulla base delle scelte prese nella fase di ricerca **EXPLORE**. Le regole di attuazione del backtracking variano nei due moduli di DELIBERAZIONE:
  - **BACKTRACKING INFORMATO**: viene trovata, nella regolare una RADICE come limite di backtracking. Viene scelta sulla base della **minore probabilità iniziale** che hanno tutte le celle su cui è stata eseguita la *guess* in stato di **EXPLORE**. Verranno di seguito effettuate le *unguess* fino alla radice trovata.

**NOTA:** viene mantenuto questo dato in ogni singola cella della **CONOSCENZA DI CONTROLLO**.

- **BACKTRACKING NON INFORMATO**: viene effettuato il backtracking fino alla radice che corrisponde alla prima *guess* posizionata in fase **EXPLORE**.
- **SOLVE**: ultimo stato in cui viene mandato all'ambiente il comando di *solve* per terminare la ricerca.

## Salience

Non è stato implementato un vero e proprio criterio univoco per tutti i moduli in merito alla scelta delle salience. Si è cercato di rispettare l'offset di 5/10 come punteggio di salience tra parti di esecuzioni eseguite (i.e. **RICERCA CORAZZATA** con regole di salience 100 e **RICERCA INCROCIATORI** con regole di salience 95). Anche se sono state fissate in un certo ordine numerico, il passaggio di stato nella ricerca è guidato da moduli e fatti ordinati (trattato in [descrizione degli stati](#)).

## Implementazione

---

### Agent

Il modulo **AGENT** è responsabile dello scheduling di esecuzione tra i differenti moduli dell'agente.

#### FATTI NON ORDINATI:

- **cell-agent**: copia del template **cell** del module **ENV** con l'aggiunta dello score (calcolato con la [formula della probabilità](#)) e dell'**original-score**, mantenendo lo score calcolato all'inizio necessario per l'esecuzione del **BACKTRACKING INFORMATO**. E' un fatto non ordinato mantenuto nella **CONOSCENZA DI CONTROLLO**.
- **k-per-row-agent**: copia del template **k-per-row** del module **ENV**. Rappresenta il numero di celle occupate da una nave su singola riga. E' un fatto non ordinato mantenuto nella **CONOSCENZA DI CONTROLLO**.
- **k-per-col-agent**: copia del template **k-per-col** del module **ENV**. Rappresenta il numero di celle occupate da una nave su singola colonna. E' un fatto non ordinato mantenuto nella **CONOSCENZA DI CONTROLLO**.
- **exec-agent**: copia del template **exec** del module **MAIN**. Rappresenta la mossa eseguita al passo *?step*. E' un fatto non ordinato mantenuto nella **CONOSCENZA**

## DI CONTROLLO.

- **boat-agent**: Rappresenta una nave specifica a cui viene assegnato un codice univoco. Sarà necessario per inizializzare le navi nella *CONOSCENZA DI CONTROLLO* e verificare, nei moduli successivi, se sono state affondate o meno facendo pattern matching con le celle. Soluzione blind, non vengono riportate le celle ma solo il tipo di nave.
- **update-score-row**: Serve al modulo **UPDATE\_KC** per aggiornare gli score sulla riga in cui è stata effettuata un'azione. Mantiene le celle aggiornate nella *CONOSCENZA DI CONTROLLO*.
- **update-score-col**: Serve al modulo **UPDATE\_KC** per aggiornare gli score sulla colonna in cui è stata effettuata un'azione. Mantiene le celle aggiornate nella *CONOSCENZA DI CONTROLLO*.

## FATTI ORDINATI:

- **first-pass-to-init**: questo fatto ordinato permette di passare l'esecuzione una volta sola al modulo di **INIT**, che inizierà la *CONOSCENZA DI CONTROLLO*.

**REGOLE**: le regole permettono di passare l'esecuzione nell'ordine corretto tra i moduli **INIT**, **UPDATE\_KC**, **DELIBERATION** e **ACTION**.

# Init

Il modulo **INIT** è responsabile di creare TUTTI i FATTI INIZIALI nella *CONOSCENZA DI CONTROLLO*. Oltre ciò, verifica lo stato delle celle se, all'interno dell'ambiente, dovesse esserci della conoscenza, e riporta la medesima all'interno dei propri fatti inizializzati.

## REGOLE:

- **init-\***: i fatti con radice *init* inizializzano i fatti **cell-agent**, **k-per-row-agent** e **k-per-col-agent** nella *CONOSCENZA DI CONTROLLO*.
- **update-cell-water**: vado ad aggiornare il contenuto delle cell-agent presente nella *CONOSCENZA DI CONTROLLO* se la k-cell, presente nella *CONOSCENZA DI DOMINIO*, contiene *water*.
- **add-water-cell**: contrassegna tutte le caselle in cui la moltiplicazione tra l'indice di possibili navi nella riga e l'indice nella colonna sia uguale a zero, ossia quando sicuramente non potrà esserci una nave ma solo *water*.

- **fill-neighbor-\***: le regole con la radice *fill-neighbor* aggiungono *water* ai lati dei pezzi delle navi quando sono sicuro che non potrà esserci altro. Es. se ho un pezzo di nave *top*, sopra di essa ci sarà sicuramente *water*.

## Update Knowledge Control

Questo **modulo** è responsabile di aggiornare la *CONOSCENZA DI CONTROLLO*, riportando le modifiche fatte allo STEP PRECEDENTE, prima di deliberare la prossima azione.

### FATTI NON ORDINATI:

- **tmp-exec-agent**: copia del template *exec-agent* del module *AGENT*.

### FATTI ORDINATI:

- **update-neighbor-guess**: fatto asserito nelle prime regole del modulo come *STATO* di aggiornamento dopo aver effettuato un'azione di *guess*.

### REGOLE:

- **update-kc-fire-cell-agent-water**: se l'azione eseguita allo step precedente è una *fire* e la cella su cui è stata effettuata riporta *water* come risultato, allora andrò ad aggiornare la cella nella *CONOSCENZA DI CONTROLLO* (se becco *water* sulla *fire*, copio solo il contenuto).
- **update-kc-guess-cell-agent**: a seguito di una *guess* **SENZA CONTENUTO** aggiorniamo SOLO lo status a *guess* della cella nella *CONOSCENZA DI CONTROLLO*.
- **update-kc-guess-cell-agent-else**: a seguito di una *guess* **CON CONTENUTO**, aggiorniamo il contenuto e lo status a *know* della cella nella *CONOSCENZA DI CONTROLLO*.
- **update-kc-<nome-nave>**: le regole con questa radice si attiveranno quando verranno rilevati dei pezzi di navi che sicuramente formano una nave. Verranno aggiornate le loro celle a score negativo e verrà rimossa dalla *CONOSCENZA DI CONTROLLO* la nave trovata.
- **add-water-after-fire-\***: le regole con questa radice si attiveranno a seguito di una *fire* per andare a posizionare *water* nelle celle adiacenti al pezzo di nave trovato. Es. sopra un *top* vi sarà necessariamente *water* in alto, a destra e a sinistra.

- [update-kc-scores-\\*](#): verranno aggiornati gli score delle righe/colonne che hanno subito modifiche a seguito di un ritrovamento di un pezzo di nave.
- [finish-update-kc-score-\\*](#): una volta finito l'aggiornamento degli score effettuato da [update-kc-scores-\\*](#), ferma l'aggiornamento ritrattando il fatto.
- [update-kc-garbage-\\*](#): le regole con questa radice eliminano il *garbage* creato dai fatti ordinati, necessari per aggiornare lo stato ed il contenuto delle celle.

## Deliberation

Tramite le regole di risoluzione sviluppate nel seguente progetto, viene proposta l'implementazione di **due sistemi esperti**:

- [Dump](#)
- [Intelligent](#)

A livello implementativo, i due sistemi si distinguono SOLO nel modulo di **DELIBERATION**, andando a modificare le due strategie di deliberazione delle azioni intraprese.

### FATTI ORDINATI:

- [state-dfs](#): questo fatto rappresenta lo *STATO* in cui si trova il modulo di *DELIBERATION* in fase di deliberazione dell'azione. Sono tre i possibili valori:
  - [greedy](#): stato di ricerca [GREEDY](#).
  - [explore](#): stato di ricerca [EXPLORE](#)
  - [backtracking](#): stato di ricerca [BACKTRACKING](#)
  - [solve](#): stato di ricerca [SOLVE](#)

### FATTI NON ORDINATI:

- [root-backtracking](#): template definito in fase di [EXPLORE](#)

### REGOLE:

- [enter-in-<name-state>-state](#): permette, secondo determinate precondizioni, di passare da uno stato all'altro quando non ci sono più regole attivabili in agenda per una determinata fase di ricerca;
- [find-guess-with-k-cell](#): delibera le guess sicure. (es. dopo aver effettuato la *fire* nello step precedente, si attiverà la seguente regola dal momento che apparirà un fatto *k-cell* su una determinata cella)

- **find-guess-\***: le regole di deliberazione del sistema esperto cercano di completare prima le navi più grandi, usando la salience per dare priorità alle guess che completano le navi con più pezzi, soprattutto quando mancano pochi pezzi per affondarle. Questo approccio aiuta a ridurre il numero di guess casuali e a velocizzare la ricerca delle navi.
- **find-cell-fire**: la regola delibera una fire in fase di GREEDY, creando una base di conoscenza prima passare alla fase di **EXPLORE**.
- **find-cell-guess-after-backtracking**: delibera una guess dopo aver trovato un root-backtracking, ossia la radice da cui proseguire la ricerca nello spazio degli stati.
- **find-cell-guess**: delibera una guess basandosi sul punteggio di scores più alto.
- **find-unguess-backtracking**: delibera le unguess in fase di BACKTRACKING finché non arriva alla cella corrispondente alla *root-backtracking*.
- **resolution**: si attiva quando non ci sono più azioni possibili da deliberare o si è passati direttamente alla fase di **SOLVE**.

**NOTA:** le regole della fase di ricerca **SOLVE** sono state definite con salience **NEGATIVA** per essere eseguite **SOLO** alla fine della ricerca (o, eventualmente quando la ricerca viene indirizzata in questa fase).

## Sistema esperto dump

Il sistema esperto Dump attua la sua ricerca con una strategia **brute force** finché ha *guess* da eseguire. La strategia, dopo aver terminato lo stato GREEDY, esegue i seguenti passi:

1. Si posizionano, nella fase di **EXPLORE**, tutte le *guess* possibili facendosi guidare dalle **probabilità** create per singola cella
2. Una volta posizionate tutte le *guess* possibili, si controlla se ce ne siano ancora:
  - se ci sono ancora *guess* a disposizione, si entra nello stato di **BACKTRACKING NON INFORMATO** effettuando le *unguess* su tutte le mosse eseguite nello stato di **EXPLORE**;
  - altrimenti, se non restano abbastanza mosse, l'agente viene mandato in stato di **SOLVE**.

## REGOLE:

- **define-root**: trova la radice (root-backtracking), ossia la cella con la *prima guess* in fase di **EXPLORE**. Sarà il lower bound della fase di BACKTRACKING.



- **remain-last-path**: si attiva per effettuare il passaggio di stato in *SOLVE* quando il numero di mosse sta per terminare e l'agente non riuscirebbe ad effettuare una risalita ed una discesa completa (formula di riferimento in **formula ultima discesa**).

## Sistema esperto Intelligente

Nel sistema esperto intelligente è stata proposta una soluzione che integra al suo interno dell'intelligenza, ossia un BACKTRACKING INFORMATO. La strategia, dopo aver terminato lo stato GREEDY, esegue i seguenti passi:

1. Si posizionano, nella fase di **EXPLORE**, tutte le *guess* possibili facendosi guidare dalle **probabilità** create per singola cella
2. Una volta posizionate tutte le *guess* possibili, si controlla se ce ne siano ancora:
  - se ci sono ancora *guess* a disposizione, si entra nello stato di **BACKTRACKING INFORMATO**, risalendo fino alla radice con la minore probabilità iniziale trovata;
  - altrimenti, se non restano abbastanza mosse da eseguire viene fatta scattare la regola di **remain-last-path** e l'agente viene mandato in stato di **SOLVE**.

### REGOLE:

- **define-min-guess-to-unguess**: trova la radice (root-backtracking), ossia la cella con la **probabilità a priori più alta** in fase di *EXPLORE*. Sarà il lower bound della fase di BACKTRACKING.
- **check-solve-root-backtracking**: condizione di uscita quando la radice è l'ultimo nodo su cui eseguire il backtracking.

## Action

Il modulo **ACTION** ha il compito di controllare se nella *CONOSCENZA DI CONTROLLO* è stata deliberata un'azione di exec (tramite il template *exec-agent*) e di che tipologia di azione si tratta. Fatto ciò, manderà l'azione di exec al modulo di **AGENT**. Una volta fatto ciò, verrà eseguita l'azione sull'ambiente. Questo modulo ha con sé delle regole che si attivano in base alla tipologia di *action* (*fire*, *guess*, *unguess* e *solve*).

### REGOLE:

- [exec-action-<name-action>](#): il modulo è responsabile di modificare l'azione, attuandola nella *CONOSCENZA DI DOMINIO*.

# Formule del progetto

---

## Formula Probabilità su singola cella

I sistemi esperti sviluppati sono guidati durante la loro ricerca dalla **PROBABILITA'** sulle singole celle. Di seguito, il calcolo della formula:

$$Prob.Cella = (NPRiga) * (NPColonna)$$

con:

*Prob.Cella* = Probabilità di un pezzo di nave per singola cella

*NPRiga* = Numero di pezzi di navi possibili in quella riga

*NPColonna* = Numero di pezzi di navi possibili in quella colonna

## Formula ultima discesa

Nei due sistemi esperti viene calcolata la seguente formula:

$$MaxDuration \geq StepAttuale + 2 * NGuessRimanenti$$

con:

*MaxDuration* = Numero massimo di azioni che l'agente può eseguire

*StepAttuale* = Step attuale di ricerca

*NGuessRimanenti* = Numero di *guess* rimanenti

Questa formula è necessaria per evitare lo stato di BACKTRACKING nel momento in cui l'agente non ha a disposizione almeno  $2 * NGuessRimanenti$  da poter eseguire per effettuare una risalita (stato BACKTRACKING) e una nuova discesa (stato [EXPLORE](#)).

# Formula original score (probabilità a priori)

All'interno di [cell-agent](#) possiamo trovare il valore di **original-score**. Questo viene calcolato come la [formula della probabilità su una singola cella](#) ma, essendo una **PROBABILITÀ A PRIORI**, viene calcolata solo la prima volta e poi persiste fino alla fine dell'esecuzione.