PROGETTO CLIPS
COLPITO E AFFONDATO...?

Docente: Roberto Micalizio
Studenti: Michele Metta e Pietro Sangermano

Obiettivo

- Sviluppare un sistema esperto che giochi ad una versione semplificata della famigerata Battaglia Navale.
- Il gioco è nella versione “in-solitario”, per cui c’è un solo giocatore che dovrà indovinare la posizione di una flotta di navi distribuite su una griglia 10x10
- L’obiettivo principale è quello di annotare come “guessed” le venti caselle sotto le quali si trovano le navi
- Il punteggio finale verrà calcolato con la seguente formula:

$$(15 * gok + 20 * sink) - (10 * gko + 10 * safe + 20 * nf + 20 * ng)$$

Ove:

- gok è il numero di celle guessed corrette

- sink è il numero di navi totalmente affondate

- gko è il numero di celle guessed errate

- safe è il numero di celle che contengono una porzione di nave e che sono rimaste involate (nè guessed nè firesd)

- nf è il numero di fire non usate

- ng è il numero di guess non usate

Griglia di gioco

Conterrà 10 navi che potranno essere posizionate in verticale o in orizzontale, che potranno essere dei seguenti tipi:

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

La griglia contiene per ogni riga e per ogni colonna il numero di celle che contengono pezzi di nave.
Questa informazione è nota al giocatore e potrà essere sfruttata a proprio vantaggio.

Vincoli da rispettare

- Deve esserci almeno una cella libera (cioè con dell'acqua) tra due navi.
- Le azioni eseguibili dal giocatore sono: fire, guess, unguess e solve.
- Numero di azioni: 5 fies, 20 guess, 1 solve

Ove:

- Fire: permette di vedere il contenuto di una cella [x,y]
- Guess: permette di far indicare al sistema esperto la possible presenta di un pezzo di nave in una cella [x,y]
- Unguess: permette di ritrattare il posizionamento di una bandierina
- Solve: permette al sistema esparto di poter richiedere la terminazione della partita

Introduzione

Sono stati creati 3 agenti:

- ***Agente1-RANDOM***: è un agente che è stato creato per essere utilizzato sostanzialmente come BASELINE poiché esso di fatti non implementa una vera e propria strategia "intelligente" ma quello andrà a fare sarà posizionare in maniera del tutto casuale le bandierine a disposizione fino a quando queste non verranno terminate senza sfruttare minimamente eventuali informazioni di partenza.
- ***Agente2***: è un agente che è in grado di sfruttare anche eventuali informazioni di partenza e che utilizza una strategia di piazzamento delle bandierine più complessa che è suddivisa in 2 fasi.
- ***Agente3***: è una variante dell'agente precedente in quanto assegna una priorità diversa alle navi che cercherà di posizionare all'interno della griglia.

Agente1-Random

- Tutte la logica di funzionamento dell'agente è inserita all'interno del modulo AGENT

Templates

```
; TEMPLATES DI AGENT:
(deftemplate cella_guess_richiesta
    (slot x)
    (slot y)
)
(deftemplate cella_guess ; serve all'agente per ricordarsi in quali celle ha già inserito le bandierine
    (slot x)
    (slot y)
)
(deftemplate start_guess ; serve all'agente per sapere quando deve generare una nuova cella sulla quale posizionare
; una bandierina e quando invece deve restituire il controllo al modulo MAIN per permettere all'ambiente di poter
; posizionare una bandierina
    (slot start (allowed-values false true))
)

; FATTI INIZIALI:
(deffacts fatti_iniziali
    (start_guess (start true))
)
```

Regole di experties

Nome regola	Descrizione
genera_guess_cella	Genera tramite l'ausilio della funzione built-in "random" la coordinata x e la coordinata y della cella sulla quale verrà richiesta la guess.
eseguo_guess	Scatta non appena la regola precedente non avrà generato una cella(x,y) sulla quale è possibile richiedere la guess e quando andrà in **esecuzione farà l'assert dell'azione guess richiesta e dopodichè farà il pop del modulo AGENT in modo tale da permettere all'ambiente di poter posizionare la bandierina nella cella(x,y)**

Agente-2 Modellazione della conoscenza

Fatti non ordinati presenti nel modulo AGENT:

- **k-per-row-bandierine-posizionate** e **k-per-col-bandierine-posizionate**: utili per sapere il numero di bandierine posizionate in riga/colonna dall'agente
- **k_cell_agent**: per ricordare in quali celle sono state posizionate le bandierine
- **corazzata, incrociatori, cacciatorpedinieri, sottomarini**: per ricordare a quali navi fanno riferimento le bandierine posizionate dall'agente e per sapere quante navi di un certo tipo ancora devono essere piazzate

```
(deftemplate corazzata
    ;; il multislot conterrà i fatti di tipo "k_cell_agent"
    ;; che permetteranno all'agente di ricordarsi in quali celle ha posizionato
    ;; una bandierina pensando che essa appartenga alla corazzata
    (multislot celle_con_bandierina)
    ;; mi dirà quante corazzate mi mancano da piazzare (all'inizio 1)
    (slot mancanti)
)
```

- **nave_piazzata_gestore**: per permettere ad AGENT di poter capire se uno dei gestori ha piazzato o meno una nave
- **cella_middle, cella_bot, cella_top, cella_left, cella_right, cella_water**: per sapere quale tipo di fatto iniziale bisognerà gestire e quindi di conseguenza quale modulo-gestore invocare

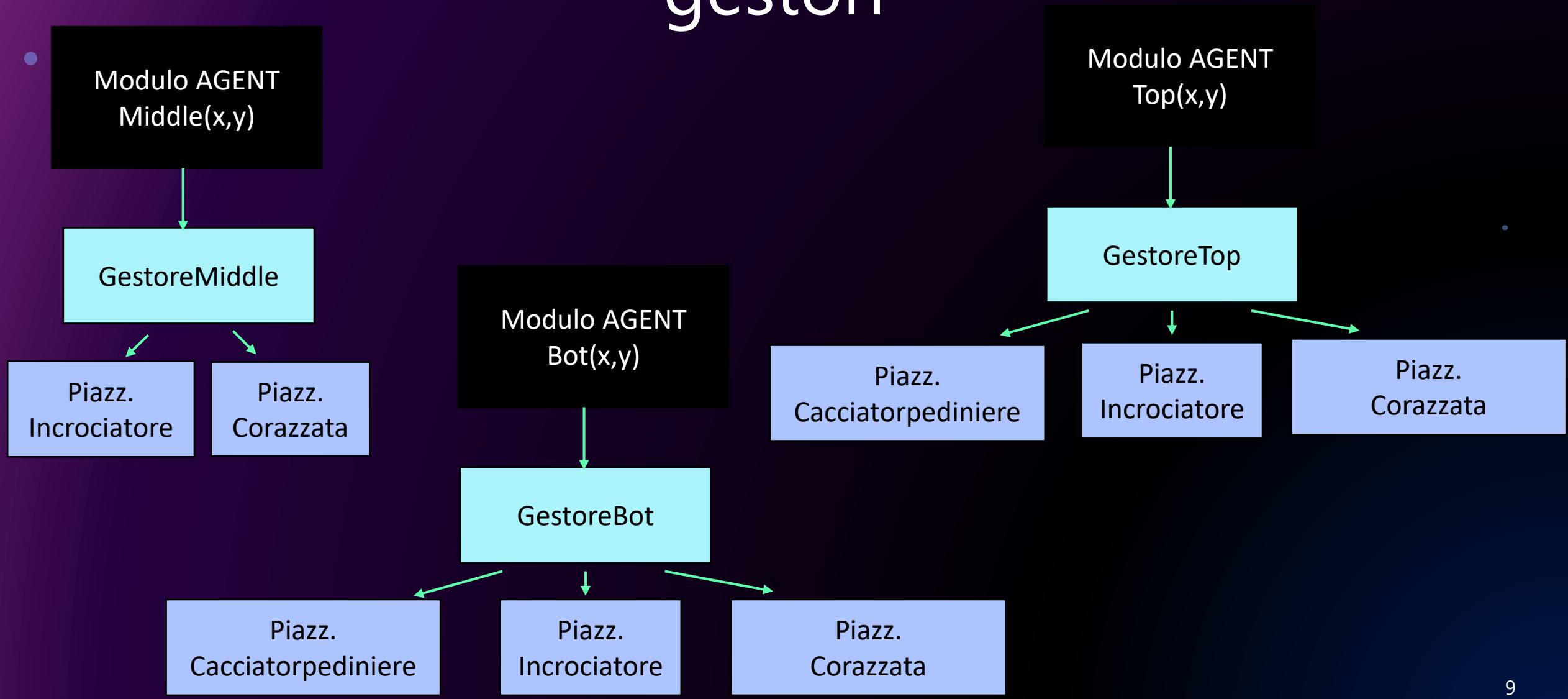
Agente-2 Fase1

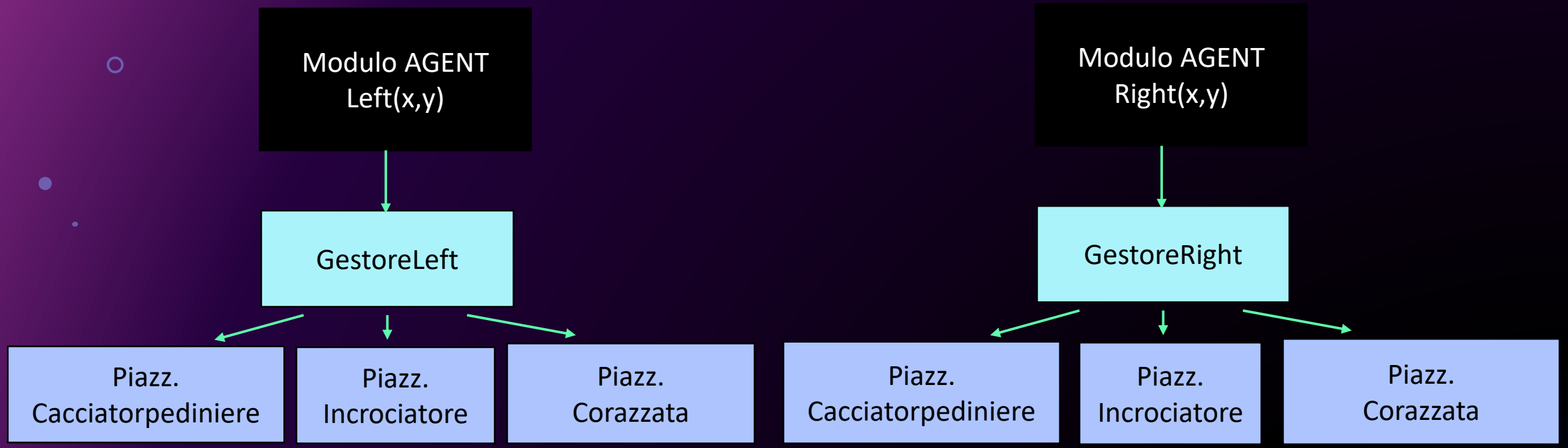
All'interno del modulo AGENT vengono scanditi tutti i fatti iniziali secondo questo ordinamento:

- **sub(x,y), (in questo caso si posiziona subito la bandierina richiamando l'ENV)**
- **top(x,y),**
- **bot(x,y)**
- **left(x,y)**
- **right(x,y)**
- **middle(x,y)**
- **water(x,y)**

- Ogni volta che viene letto un nuovo fatto iniziale, il modulo AGENT posizionerà in cima al Focus il modulo-gestore che si preoccuperà di gestire quel determinato fatto iniziale

Gestione invocazioni dei moduli-gestori





- **Ogni Modulo-Gestore contiene al proprio interno delle regole di controllo che permettono al modulo AGENT di poter sapere se ed eventualemente in quale cella una bandierina è stata posizionata (i fatti riferiti alle bandierine verranno asseriti dal modulo gestore in modo tale che AGENT possa vederli in WM e eseguire l'azione guess di una bandierina per volta).**
- **Ogni Modulo-Piazzamento contiene al proprio interno sia delle regole di controllo per avvertire il proprio gestore di un eventuale piazzamento di eventuali bandierine e sia delle regole di dominio che permettono di poter modificare la conoscenza di dominio come ad esempio sapere il numero di navi di un certo tipo che l'agente dovrà ancora individuare.**

Regole di Expertise di AGENT
per gestire il fatto iniziale (k-cell (x ?x) (y ?y) (content top))

```
; La regola di sotto si preoccupa di prendere un fatto iniziale "top" e di asserire (cella_top (x ?x) (y ?y) (considerata false)) in modo tale
; che la regola "start_gestione_top" possa scattare e una volta eseguita potrà richiamare il modulo "11_GESTORE_TOP":
(defrule creo_nuovo_fatto_iniziale_top (declare (salience 508))

    (k-cell (x ?x) (y ?y) (content top) )
    (status (step ?s)(currently running))
    (not (exec  (action guess) (x ?x) (y ?y)))

    ; Il not qui sotto mi assicura il fatto che non vengano gestiti 2 o più "top" iniziali, altrimenti
    ; ci sarebbero dei conflitti e le cose non funzionerebbero correttamente:
    (not (k_cell_agent (x ?x_esistente) (y ?y_esistente) (content top) (considerato false) (current yes)))

=>
    (assert (exec (step ?s) (action guess) (x ?x) (y ?y)))
    (assert (k_cell_agent (x ?x) (y ?y) (content top) (considerato false) (current yes))) ; asserisco la cella nella quale l'agente posizionerà la bandierina
    ; in modo tale che l'agente possa ricordarsi di questo posizionamento

    ; asserisco che questa cella top dovrà essere considerata per piazzare una qualche nave:
    (assert (cella_top (x ?x) (y ?y) (considerata false)))

    (pop-focus) ;; in modo tale che l'ENV possa posizionare la bandierina in cella(x ?x) (y ?y) dove siamo certi esserci un "top"
)

; Adesso, la regola di sotto fa questo:
; Se c'è un top non ancora considerato
; Allora, chiamo semplicemente il modulo "gestore_caso_top" inserendolo in cima allo stack
(defrule start_gestione_top (declare (salience 508))

    ; controllo che ci sia ancora una qualche cella_top in WM che non è stata ancora considerata
    ?cella_top <- (cella_top (x ?x) (y ?y) (considerata false))

=>
    ; dico che adesso questa cella_top è stata considerata perchè subito sotto
    ; il modulo "GESTORE_TOP" la prenderà in cosiderazione per piazzare una nave:
    (modify ?cella_top(considerata true))

    ; Invoco il modulo qui sotto che si occuperà di cercare di piazzare una qualche nave
    ; sfruttando l'informazione sulla cella top trovata nell'antecedente:
    (focus GESTORE_TOP) ; quando terminerà, il controllo tornerà al MODULO AGENT
)
```

Richiediamo la guess all'ENV

In modo tale che MAIN possa chiamare ENV per posizionare la bandierina nella cella con "top"

Inseriamo in cima al focus il modulo "GESTORE_TOP" che cercherà di piazzare una qualche nave sfruttando l'info iniziale

Templates di GestoreTop

```
(deftemplate decremento_corazzate
    ; solo se tutti questi cambi saranno a true allora verrà eseguita la regola
    ; che si occuperà di decrementare il numero tot di incrociatori ancora da piazzare

    ; "slot cella_corazzata_1" mi dice se l'agente ha memorizzato (nella nostra struttura)
    ; la cella che contiene il primo pezzo della corazzaza (true) o meno (false)
    (slot cella_corazzata_1 (allowed-values false true))
    (slot cella_corazzata_2 (allowed-values false true)) ; mi dice se l'agente ha memorizzat
    (slot cella_corazzata_3 (allowed-values false true)) ; mi dice se l'agente ha memorizzat
    (slot cella_corazzata_4 (allowed-values false true)) ; mi dice se l'agente ha memorizzat
)
(deftemplate decremento_incrociatori
    ; solo se tutti questi cambi saranno a true allora verrà eseguita la regola
    ; che si occuperà di decrementare il numero tot di incrociatori ancora da piazzare
    (slot cella_incrociatore_1 (allowed-values false true)) ; mi dice se l'agente ha memori
    (slot cella_incrociatore_2 (allowed-values false true)) ; mi dice se l'agente ha memori
    (slot cella_incrociatore_3 (allowed-values false true)) ; mi dice se l'agente ha memori
)
(deftemplate decremento_cacciatorpedinieri
    ; solo se tutti questi cambi saranno a true allora verrà eseguita la regola
    ; che si occuperà di decrementare il numero tot di incrociatori ancora da piazzare
    (slot cella_cacciatorpediniere_1 (allowed-values false true)) ; mi dice se l'agente ha
    (slot cella_cacciatorpediniere_2 (allowed-values false true)) ; mi dice se l'agente ha
)
```

```
; template che permette di poter specificare se bisogna esegure o meno
; l'azione di unguess
(deftemplate fai_unguess
    (slot unguess (allowed-values false true))
)

; template che permette di poter memorizzare le righe delle celle
; che si trovano sotto la cella_top(x,y)
(deftemplate celle_sotto_a_top_in_gestore_top
    (slot x_row_sotto_1) ; riga subito sotto al top
    (slot x_row_sotto_2) ; riga +2 sotto al top
    (slot x_row_sotto_3) ; riga +3 sotto al top
)
```

```
; I templates qui sotto, qualora fosse stata richiesta l'unguess,
; permettono di poter far attivare solo la regola
; che si preoccuperà di aggiornare il numero di navi di un certo tipo
; (corazzata o incrociatori o cacciatorpedinieri o sottomarini)
; in base a quale nave era stata assegnata la bandierina sulla
; quale è stata richiesta l'unguess.
(deftemplate unguess_corazzata
    (slot unguess (allowed-values false true))
)
(deftemplate unguess_incrociatori
    (slot unguess (allowed-values false true))
)
(deftemplate unguess_cacciatorpedinieri
    (slot unguess (allowed-values false true))
)
(deftemplate unguess_sottomarini
    (slot unguess (allowed-values false true))
)
```

Regola di GestoreTop

```
(defrule controllo_piazzamento_corazzata (declare (salience 150))

    (cella_top (x ?x) (y ?y) (considerata true)) ; controllo che ci sia una cella_top che il modulo AGENT mi ha detto di considerare
    ; ASSUNZIONE:
    ; se è già stato trovato un modo per posizionare il cacciatorpediniere o l'incrociatore,
    ; allora l'agente non proverà neanche a piazzare la corazzata.
    (nave_piazzata_gestore (piazzamento false))
=>

    (assert (decremento_corazzate (cella_corazzata_1 false) (cella_corazzata_2 false) (cella_corazzata_3 false) (cella_corazzata_4 false)))
    ; richiamo il sotto-modulo che si preoccuperà di posizionare la corazzata
    (focus PIAZZAMENTO_CORAZZATA_TOP)
)
```

- La regola di sopra permette al modulo GESTORE_TOP di poter richiamare il sotto-modulo di piazzamento della corazzata qualora non fosse stata già trovata una nave precedente (che in questo caso per l'agente1 potrà essere o il cacciatorpediniere o l'incrociatore)
- Qualora l'agente non riuscisse a pizzare né un cacciatorpediniere, né un incrociatore e neanche una corazzata, allora all'interno di questo gestore verrà eseguita una delle regole che verranno tra poco descritte per andare ad eseguire l'unguess su una qualche cella.

Template e regole di expertise di PIAZZAMENTO_CORAZZATA_TOP

```
(deftemplate piazzamento_corazzata_verticale
    (slot x)
    (slot y)
)
```

Qualora fosse possibile posizionare la corazzata sfruttando l'informazione cella_top(x,y), questo template verrà istanziato tramite l'esecuzione di una assert specificando come coordinate x,y quelle appartenenti alla cella più in basso dalla quale potrà essere posizionata la corazzata.

Nome regola	Descrizione
Posizionamento_corazzata_in_verticale_conoscendo_top	Verifica che sia possibile posizionare una corazzata supponendo cella_top(x,y)
Gestione_caso_solo_verticale_conoscendo_top	Sarà attivabile solo se la regola precedente è riuscita a trovare un modo per piazzare la corazzata e quando andrà in esecuzione asserirà le 3 celle (la bandierina in cella_top(x,y) è già stata posizionata) di tipo k_cell_agent nella quale il modulo AGENT dovrà inserire le bandierine che faranno riferimento alla corazzata
Memorizzo_corazzata_1	Data k_cell_agent(x1,y1), Incrementa di 1 le strutture interne all'agente **k-per-row-bandierine-posizionate** e **k-per-col-bandierine-posizionate (dove row == x1 e col == y1) e aggiunge nella struttura corazzata il fatto** k_cell_agent(x1,y1)
Memorizzo_corazzata_2	Data k_cell_agent(x2,y2), Incrementa di 1 le strutture interne all'agente **k-per-row-bandierine-posizionate** e **k-per-col-bandierine-posizionate (dove row == x2 e col == y2) e aggiunge nella struttura corazzata il fatto** k_cell_agent(x2,y2)
Memorizzo_corazzata_3	Data k_cell_agent(x3,y3), Incrementa di 1 le strutture interne all'agente **k-per-row-bandierine-posizionate** e **k-per-col-bandierine-posizionate (dove row == x3 e col == y3) e aggiunge nella struttura corazzata il fatto** k_cell_agent(x3,y3)
Memorizzo_corazzata_4	Data k_cell_agent(x4,y4), Incrementa di 1 le strutture interne all'agente **k-per-row-bandierine-posizionate** e **k-per-col-bandierine-posizionate (dove row == x4 e col == y4) e aggiunge nella struttura corazzata il fatto** k_cell_agent(x4,y4)
Decremento_corazzata	Decrementa di 1 il numero di corazzate ancora da trovare

Regola di GestoreTop per far partire la gestione dell'unguess

```
(defrule non_sono_riuscito_a_piazzare_nessuna_nave_precedente (declare (salience 100))

    ; controllo che ci sia una cella_top che il modulo AGENT mi ha detto di considerare:
    ?cella_top_corrente <- (cella_top (x ?x) (y ?y) (considerata true))

    ; verifico che l'agente non sia riuscito a piazzare nè una corazzata nè un incrociatore e neanche un cacciatorpediniere
    (nave_piazzata_gestore (piazzamento false))

=>
    ;; A questo punto l'agente è certo di aver commesso un errore in precedenza poichè non è riuscito a piazzare nessuna nave
    ;; partendo da una cella nella quale è certo che ci sia il "top" di una nave.
    ;; Quindi quello che farà adesso sarà quello di togliere una bandierina per volta dalla colonna "y" corrente
    ;; fino a quando non riuscirà a piazzare una qualche nave lungo questa colonna (muovendosi ovviamente verso il basso):
    ;; Per fare quello appena detto, verrà asserito qui sotto un fatto che potrà far attivare una delle regole
    ;; chiamata "eseguo_unguess_lungo_....":
    (assert (fai_unguess (unguess true)))

    ; Con l'assert qui sotto, mi preoccupo di settare
    ; tutti i valori che servono alle regole di posizionamento per cercare di scoprire con le regole successive
    ; in quale riga o colonna bisogna togliere una bandierina già inserita che si suppone essere sbagliata:
    (assert (celle_sotto_a_top_in_gestore_top (x_row_sotto_1 (+ ?x 1)) (x_row_sotto_2 (+ ?x 2)) (x_row_sotto_3 (+ ?x 3))))

)
```

- La regola di sopra permette al modulo GESTORE_TOP, qualora non fosse stato possibile posizionare nessuna nave in verticale partendo da cella_top(x,y), di poter asserire le righe nelle quali potenzialmente potrà essere eseguita l'unguess.

- ***Supponendo cella_top(x,y), solo una di queste regole di expertise potrà essere eseguita in modo tale da richiedere l'esecuzione dell'unguess su una particolare cella (qualora scattasse più di una regola presente qui sotto allora verrà eseguita solamente quella che nel file sorgente è stata scritta prima):***

Nome regola	Descrizione
eseguo_unguess_lungo_la_stessa_colonna_di_cella_top	Supponendo che lungo la colonna y non sia possibile piazzare una bandierina: rimanendo fissa la colonna y, verrà cercata una bandierina presente in una qualche cella(x,y) tale che in questa cella non sia noto che sia presente un qualche pezzo di nave oppure un sottomarino sulla quale eseguire l'unguess
eseguo_unguess_su_riga_sotto_a_cella_top	Supponendo che lungo la riga sotto a cella_top non sia possibile piazzare una bandierina: rimanendo fissa la x_row_sotto_1, verrà cercata una bandierina presente in una qualche cella(x_row_sotto_1, y_var) tale che in questa cella non sia noto che sia presente un qualche pezzo di nave oppure un sottomarino sulla quale eseguire l'unguess
eseguo_unguess_su_due_righe_sotto_a_cella_top	Supponendo che due righe sotto a cella_top non sia possibile piazzare una bandierina: rimanendo fissa la x_row_sotto_2, verrà cercata una bandierina presente in una qualche cella(x_row_sotto_2, y_var) tale che in questa cella non sia noto che sia presente un qualche pezzo di nave oppure un sottomarino sulla quale eseguire l'unguess
eseguo_unguess_su_tre_righe_sotto_a_cella_top	Supponendo che tre righe sotto a cella_top non sia possibile piazzare una bandierina: rimanendo fissa la x_row_sotto_3, verrà cercata una bandierina presente in una qualche cella(x_row_sotto_3, y_var) tale che in questa cella non sia noto che sia presente un qualche pezzo di nave oppure un sottomarino sulla quale eseguire l'unguess
aggiorno_struttura_k_per_row_per_col_bandierine_posizionate	**Verrà eseguita se non è stato possibile piazzare né una nave e né eseguire l'unguess su una qualche cella. Questa regola si preoccuperà quindi di aggiornare la struttura interna all'agente per permettere di ricordarsi che comunque ha posizionato una bandierina in cella_top(x,y)**

- ***Dopo che una delle regole di richiesta unguess della slide precedente sarà stata eseguita e supponendo che la cella(x,y) sulla quale è stata richiesta l'unguess appartenga ad un cacciatorpediniere:***

```
;; stessa cosa di sopra ma questa volta verrà fatto per il cacciatorpediniere
(defrule rimuovo_bandierina_da_cacciatorpediniere (declare (salience 98))

    ; mi assicuro che in WM ci sia il fatto che stabilisca la cancellazione della bandierina in cella(x,y)
    (bandierina_da_cancellare (x ?x) (y ?y) (content ?val_content) (considerato ?val_considerato) (current ?val_current))

    ?k_cell_agent <- (k_cell_agent (x ?x) (y ?y) (content ?val_content) (considerato true) (current no)) ; maniglia
    ?cacciatorpedinieri <- (cacciatorpedinieri (celle_con_bandierina $?facts) (mancanti ?m)) ; maniglia
    (test (member$ ?k_cell_agent $?facts)) ; mi assicuro che la ?k_cell_agent(x,y), nella quale l'agente aveva posizionato la bandierina,
    ; faccia parte (sempre secondo l'agente) di un cacciatorpediniere.

=>
  ; con l'istruzione qui sotto, cancello la ?k_cell_agent dal multislot dei cacciatorpedinieri (perchè l'agente sta supponendo
  ; che questa cella in realtà non dovrebbe contenere una bandierina e quindi farà su di essa l'unguess)
  (modify ?cacciatorpedinieri (celle_con_bandierina (delete-member$ $?facts ?k_cell_agent)))

  ; ritratto la ?k_cell_agent perchè ormai l'agente ha tolto la bandierina su di essa:
  (retract ?k_cell_agent)

  (assert (cacciatorpedinieri_incrementati (x ?x) (y ?y) (incrementata true)))
  (assert (unguess_cacciatorpedinieri (unguess true)))
)

;; Questa regola si assicura di poter incrementare di 1 il numero di cacciatorpedinieri da trovare:
(defrule incremento_cacciatorperdinieri_da_trovare (declare (salience 98))

    ?cella_top_corrente <- (cella_top (x ?x) (y ?y) (considerata true))
    (unguess_cacciatorpedinieri (unguess true))
    (not (cacciatorpedinieri_incrementati (x ?x) (y ?y) (incrementata true))) ; in questo modo sono certo che non faccio due volte l'incremento
    ; per la stessa cella top di partenza
    ?cacciatorpedinieri <- (cacciatorpedinieri (celle_con_bandierina $?facts) (mancanti ?m))
    (test (< ?m 3)) ; mi assicuro di poter incrementare i cacciatorpedinieri di 1
=>
    (assert (cacciatorpedinieri_incrementati (x ?x) (y ?y) (incrementata true)))
    (modify ?cacciatorpedinieri (mancanti (+ ?m 1))) ; incremento di 1 il numero di cacciatorpedinieri ancora da trovare.
)
```

- L'incremento del numero totale di cacciatorpedinieri ancora da trovare viene fatto perché poiché è stata eseguita l'unguess su una cella che l'agente pensava appartenesse ad un cacciatorpediniere, allora vuol dire che il cacciatorpediniere che l'agente pensava di aver trovato in realtà era sbagliato

Gestori rimanenti..

Per i gestori rimanenti:

- ✓ ***GESTORE_BOT : simile al GESTORE_TOP con la differenza che l'agente proverà a posizionare una qualche nave partendo dal basso e muovendosi verso l'alto***
- ✓ ***GESTORE_LEFT : stessa cosa di GESTORE_TOP ma proverà a posizionare una qualche nave spostandosi verso destra***
- ✓ ***GESTORE_RIGHT : stessa cosa di GESTORE_TOP ma proverà a posizionare una qualche nave spostandosi verso sinistra***
- ✓ ***GESTORE_MIDDLE : diverso rispetto ai gestori precedenti in quanto conoscendo cella_middle(x,y), l'agente valuterà differenti posizionamenti di una nave (incrociatore o corazzata)…***

regole di expertise di **PIAZZAMENTO_CORAZZATA_MIDDLE**

Nome regola	Descrizione
Posizionamento_corazzata_in_orizzontale_left_subito_sinistra	Verificherà che sia possibile posizionare la corazzata supponendo che a sinistra di cella_middle(x,y) ci sia il "left"
Posizionamento_corazzata_in_orizzontale_right_subito_a_destra	Verificherà che sia possibile posizionare la corazzata supponendo che a destra di cella_middle(x,y) ci sia il "right"
Gestione_conflitto_solo_orizzontale	In caso di conflitto lungo la direzione orizzontale, questa regola tramite il calcolo dello score mostrato nella slide successiva permetterà ad altre regole di poter stabilire qualle delle due direzioni orizzontali precedenti è da preferire per il piazzamento della corazzata
Posizionamento_corazzata_in_verticale_bot_subito_sotto	Verificherà che sia possibile posizionare la corazzata supponendo che subito sotto a cella_middle(x,y) ci sia il "bot"
Posizionamento_corazzata_in_verticale_top_subito_sopra	Verificherà che sia possibile posizionare la corazzata supponendo che subito sopra a cella_middle(x,y) ci sia il "top"
Gestione_conflitto_solo_verticale	In caso di conflitto lungo la direzione verticale, questa regola tramite il calcolo dello score mostrato nella slide successiva permetterà ad altre regole di poter stabilire quale delle due direzioni verticali precedenti è da preferire per il piazzamento della corazzata
Calcolo_score	In caso di conflitto lungo sia una direzione orizzontale che una verticale, questa regola tramite il calcolo dello score mostrato nella slide successiva sotto permetterà ad altre regole di poter stabilire quale delle due direzioni è da preferire per il piazzamento della corazzata

Calcolo scores

Score_posizionamento_corazzata_in_orizzontale_left_subito_a_sinistra =

[

(k_per_col_left − k_per_col_left_bandierine_posizionate) +
(k_per_col_middle1 − k_per_col_middle1_bandierine_posizionate) +
(k_per_col_middle2 − k_per_col_middl2_bandierine_posizionate) +
(k_per_col_right − k_per_col_right_bandierine_posizionate)

]

- ***Stessa cosa per il "posizionamento_corazzata_in_orizzontale_right_subito_a_destra" e in caso di conflitto verrà scelta la direzione con lo score max***

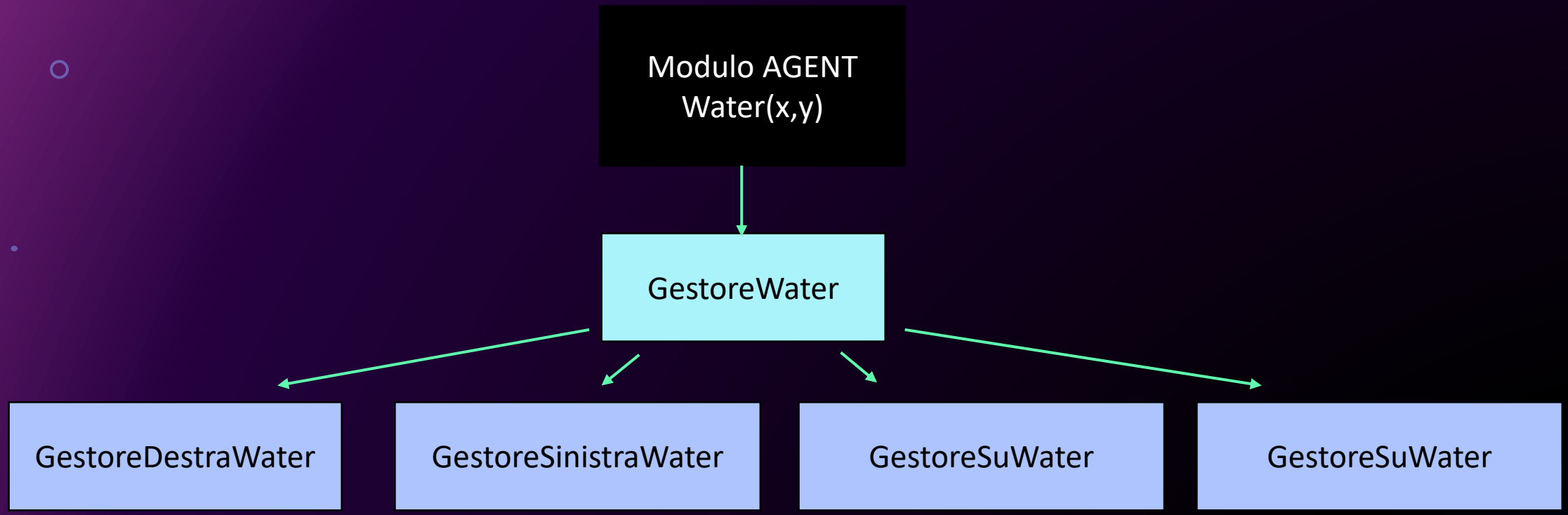
Score_posizionamento_corazzata_in_verticale_bot_subito_sotto =

[

(k_per_row_bot − k_per_row_bot_bandierine_posizionate) +
(k_per_row_middle1 − k_per_row_middle1_bandierine_posizionate) +
(k_per_row_middle2 − k_per_row_middl2_bandierine_posizionate) +
(k_per_row_right − k_per_row_right_bandierine_posizionate)

]

- ***Stessa cosa per il "posizionamento_corazzata_in_verticale_top_subito_sopra" e in caso di conflitto verrà scelta la direzione con lo score max***
- ***Anche In caso di conflitto tra una direzione orizzontale e verticale verrà scelta la direzione con lo score max***



- Nel caso di una cella_water, invece quello che accade è questo:

1) Vengono controllate tutte le celle adiacenti (su, giù, sinistra, destra) per vedere quali potrebbero essere le celle nelle quali l'agente potrebbe partire con il cercare di posizionare una qualche nave. Quindi se ad esempio sarà possibile posizionare la bandierina nella cella superiore, allora verrà calcolato lo score di questa cella in questo modo:

$$Score\_cella = (k\_per\_row - k\_per\_row\_bandierine\_posizionate) + (k\_per\_col - k\_per\_col\_bandierine\_posizionate)$$

2) Dopodichè, verrà asserita in WM la cella_max che specificherà in quale direzione è più conveniente provare a piazzare una nave in base al valore di score

3) Verrà richiamato il sotto-modulo assegnato alla direzione vincente (ad es. GestoreDestraWater se la direzione migliore rispetto a cella_water(x,y) è quella destra)

4) Infine, **il sotto-modulo vincente proverà a posizionare una qualche nave seguendo questo ordine: 1-sottomarino (solo se ci troviamo nella fase2), 2-cacciatorpediniere, 3-incrociatore, 4-corazzata**

Agente-2 Fase2

- **- A questo punto dopo aver cercato di posizionare le diverse bandierine sfruttando tutti i fatti iniziali disponibili, l'agente entrerà nella FASE 2 poiché scatterà la regola di controllo presente nel modulo AGENT chiamata** *"start_fase_2".*
- **L'idea di base di quest'ultima fase è questa:**

  "ANDANDO A CONSIDERARE OGNI CELLA DELLA GRIGLIA (presente nei fatti iniziali del modulo "GESTORE_FASE_3"), SI VERIFICHERA' CHE QUESTA CELLA SIA AMMISSIBILE (rispetti 2 vincoli fondamentali), QUALORA QUESTO DOVESSE ACCADERE ALLORA QUESTA CELLA VERRA' ASSERITA istanziando il template chiamato "cella_ammissibile" ANDANDO A SPECIFICARE LO SCORE DI QUESTA CELLA che verrà calcolato con la formula mostrata sotto".

- **cella(x,y) ammissibile :**
  - **Vincolo 1: sulla cella non deve esserci una bandierina**
  - **Vincolo 2: deve essere possibile posizionare una bandierina nella cella (controllando sia la riga x che la colonna y)**

- **score_cella(x,y) =** ***[*** (k_per_row_x + k_per_col_y) - (k_per_row_bandierine_pos_x + k_per_col_bandierine_pos_y) ***]***

Regole di experties di Gestore_Fase_2..

- La regola qui sotto permette al gestore della fase 2 di poter asserire come cella_max la cella che tra tutte le possibili celle ammissibili risulta essere quella con lo score maggiore (in caso di parità viene scelta la cella ammissibile più recente):

```
(defrule asserisco_cella_max_fase_2 (declare (salience 498))

    ; verifico che possa prendere la cella_max_fase_2 successiva:
    ?next_cella_max_fase_2 <- (next_cella_max_fase_2 (next_max true))

    ;; 1) Verifico che ci sia una cella_ammissibile che potenzialmente sarà quella max:
    ?cella_ammissibile_max <- (cella_ammissibile (x ?x_max) (y ?y_max) (score ?score_max) (aggiunta false))

    ;; 2) Tra tutte le celle ammissibili correnti la "?cella_ammissibile_max"
    ;; DEVE ESSERE QUELLA CON LO SCORE MAGGIORE (in caso di conflitti verrà scelta l'ultima presente in WM):
    (not (cella_ammissibile (x ?x_other) (y ?y_other) (score ?score_other &:(> ?score_other ?score_max)) (aggiunta false)))
=>

    ; asserisco la cella_max_fase_2 corrente:
    (assert (cella_max_fase_2 (x ?x_max) (y ?y_max) (considerata false)))

    ; lo riporto a false per non far riscattare di nuovo la regola:
    (modify ?next_cella_max_fase_2 (next_max false))

    ;; ritratto la cella_ammissibile_max che ho asserito come "cella_max_fase_2" in modo tale che non venga
    ;; più considerata successivamente:
    (retract ?cella_ammissibile_max)
)
```

Vincolo field constraint

Regole di experties di Gestore_Fase_2..

Nome regola	Descrizione
non_ho_posizionato_gia_una_bandierina_in_cella_max_fase_2	Scatterà quando avrò la certezza che nella cella_max non è stata posizionata nessuna bandierina e quindi nel conseguente **verrà richiesta la fire (qualora ne l'agente ne avesse ancora una disponibile) sulla cella_max**
scopro_di_aver_posizionato_gia_una_bandierina_in_cella_max_fase_2	Scatterà quando l'agente si accorgerà di aver già posizionato una bandierina in questa cella_max e quindi essa verrà ritrattata in modo da poter trovare la successiva cella_max
fires_terminate_ma_guess_ancora_no	Questa regola verrà eseguita quando le fires a disposizione sono terminate , in questo caso l'agente continuerà a posizionare tutte le bandierine rimanenti nelle celle_max che verranno asserite mano mano perché saranno quelle che con maggiore "probabilità" potranno contenere dei pezzi di una qualche nave
fires_e_guess_terminate	Quando l'agente si accorgerà che sia le fires che le guess sono terminate, richiederà la terminazione dellla partita settando l'attributo "termina game" del template "termina_partita" a "true" e successivamente farà il pop del modulo corrente dal focus in modo tale che il controllo torni al modulo AGENT che potrà eseguire l'azione "solve" per far calcolare il punteggio finale e terminare la partita.
richiedo_la_fire_a_AGENT	Richiede al modulo AGENT l'esecuzione della fire sulla cella_max(x,y) corrente tramite il setting dell'attributo "considerata" al valore "eseguiFire" dell'istanza del template chiamato "cella_max_fase_2" presente nel modulo AGENT ed esportato verso l'esterno

```
;; servirà al modulo GESTORE_FASE_2 per far capire ad Agent in quale cella dovrà eseguire la fire o posizionare una bandierina:
(deftemplate cella_max_fase_2
    (slot x)
    (slot y)
    (slot considerata (allowed-values false fireRichiedibile eseguiFire FireEseguita completata FiresTerminate BandierinaSenzaFireInserita completataSenzaFire))
)
```

Note conclusive sul modulo AGENT..

- A questo punto una volta tornati al modulo AGENT, se il modulo "GESTORE_FASE_2" aveva richiesto una fire su una certa cella, la regola "eseguo_fire_in_agent" si preoccuperà di richiedere questa fire e dopo che il modulo ENV avrà terminato e il controllo sarà ritornato al modulo AGENT allora in base a quale sarà il contenuto della cella fire, verrà richiamato il modulo gestore opportuno seguendo la classica sequenza di invocazioni vista precedentemente

- Se invece, il modulo "GESTORE_FASE_2" aveva richiesto una guess su una certa cella, allora la regola "posiziono_direttamente_bandierina_in_agent" si preoccuperà di richiedere la guess e dopo che il modulo ENV avrà terminato e il controllo sarà ritornato al modulo AGENT allora tramite un'opportuna regola di controllo verrà immediamente rimesso in cima al focus il modulo "GESTORE_FASE_2" in modo tale che possa riprendere la sua esecuzione individuando la successiva cella_max ammissibile... E così via fino a quando sia le fires che le guess a disposizione dell'agente non saranno terminate e quindi verrà richiesta la terminazione della partita

Agente-3

- L'agente-3 corrisponde ad una variazione dell'agente-2 in quanto la differenza sostanziale tra i due agenti è che l'agente-3 si basa su una preferenza di posizionamento delle navi opposta rispetto all'agente-2:
  - **Precedenza navi più piccole Agente-2: sottomarino (solo in fase2) – cacciatorpediniere – incrociatore – corazzata**
  - **Precedenza navi più grandi Agente-3: corazzata – incrociatore – cacciatorpediniere – sottomarino (solo in fase 2)**

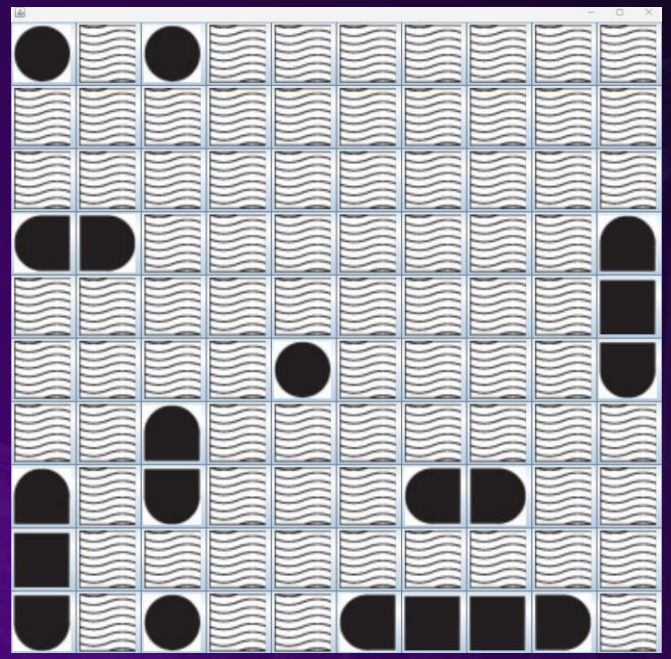
- Questa diversità di preferenza (euristica) permette all'agente-3 di seguire una strategia di esecuzione delle guess-unguess-fires che nella maggior parte dei casi risulta essere totalmente diversa da quella attuata dall'agente-2 producendo in termini di score risultati differenti tra i due agenti

Test eseguiti

- **Per testare il comportamento dei 3 agenti sono stati eseguiti dei test su 2 mappe differenti:**
  - **Mappa1: con navi posizionate per lo più verso la parte esterna**
  - **Mappa2: con navi posizionate per lo più nella parte centrale**

- **Inoltre, per ciascuna mappa sono stati eseguiti dei test partendo da una diversa quantità di fatti iniziali conosciuti:**
  - **Senza conoscenza iniziale**
  - **Con 3 celle conosciute**
  - **Con 5 celle conosciute**

Risultati Mappa 1 – Senza conoscenza iniziale

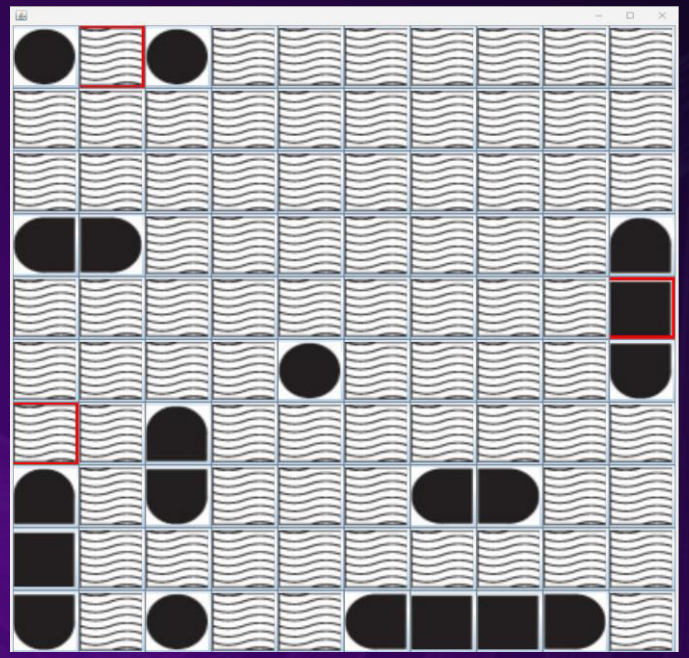


Senza conoscenza iniziale

	N° Fire OK	N° Guess OK	N° Navi distrutte	Score
Agente1-Random	0	4	1	-334
Agente-2	4	10	6	150
Agente-3	4	11	7	205

- L'Agente-Random è stato eseguito 10 volte per ogni mappa e dopodichè sono stati presi i valori medi dei campi della tabella, inoltre, poiché esso non sfrutta in nessun modo i fatti iniziali, i valori riportati saranno uguali per ogni configurazione di una stessa mappa

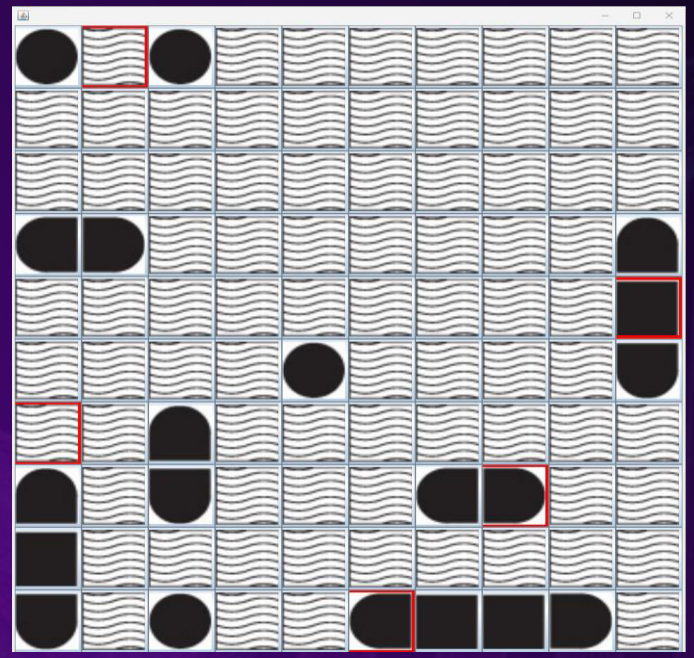
Risultati Mappa 1 – con 3 celle conosciute



Con 3 celle conosciute

	N° Fire OK	N° Guess OK	N° Navi distrutte	Score
Agente1-Random	0	4	1	-334
Agente-2	4	11	7	225
Agente-3	4	11	7	225

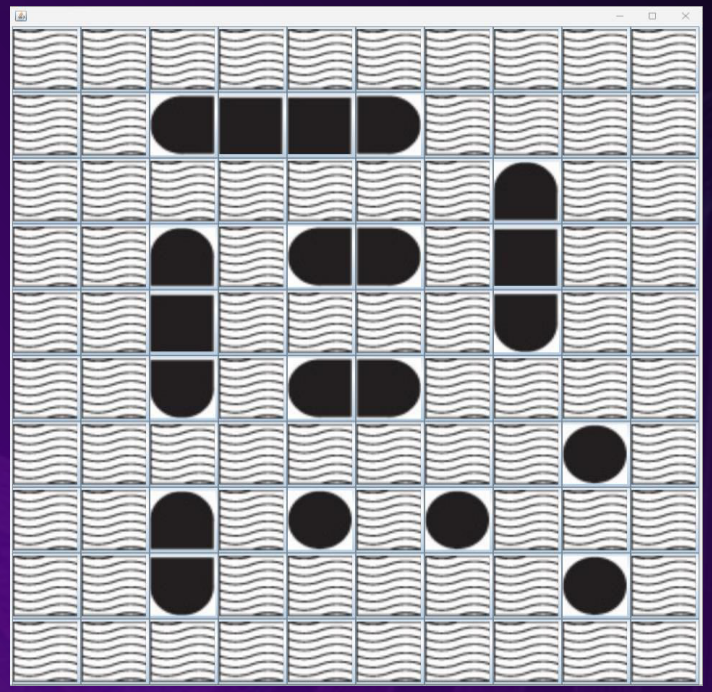
Risultati Mappa 1 – con 5 celle conosciute



Con 5 celle conosciute

	N° Fire OK	N° Guess OK	N° Navi distrutte	Score
Agente1-Random	0	4	1	-334
Agente-2	4	10	7	230
Agente-3	3	12	8	300

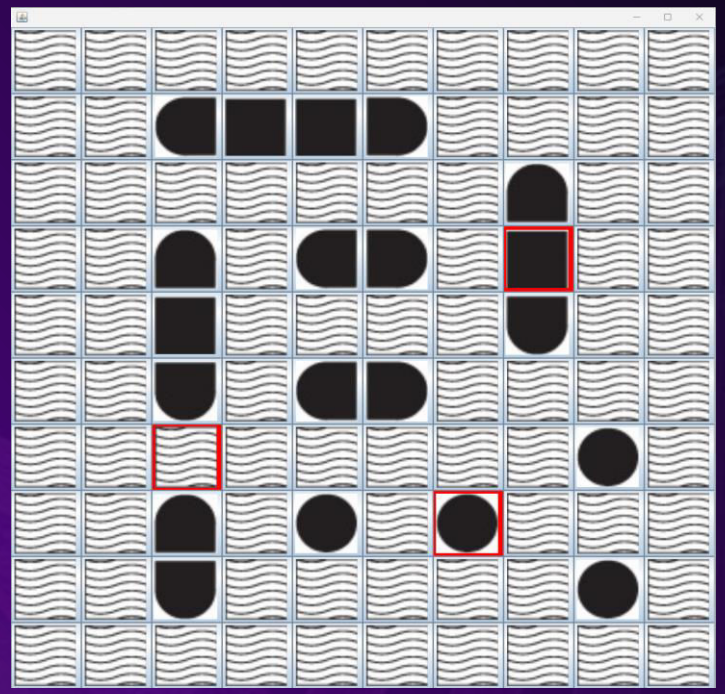
Risultati Mappa 2 – senza conoscenza iniziale



Senza conoscenza iniziale

	N° Fire OK	N° Guess OK	N° Navi distrutte	Score
Agente1-Random	0	5	1	-299
Agente-2	5	9	5	115
Agente-3	5	9	5	115

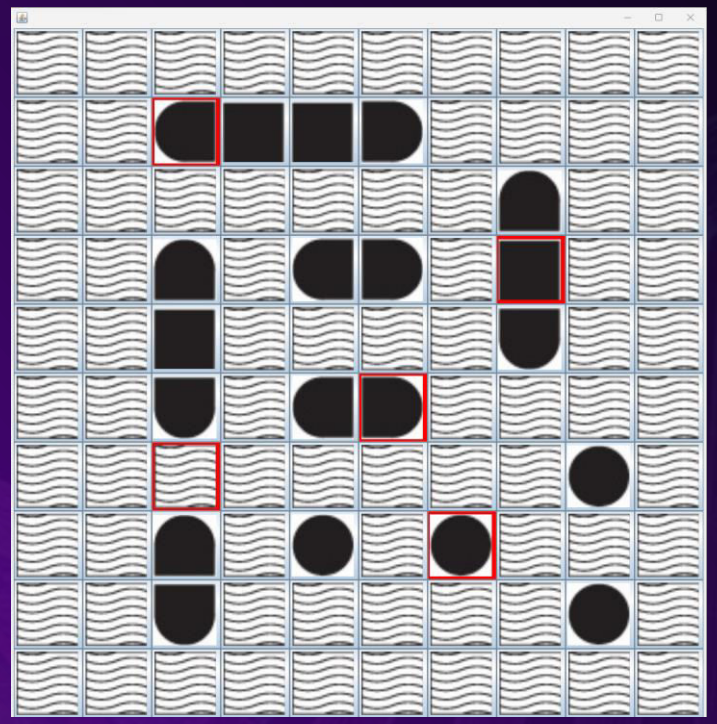
Risultati Mappa 2 – con 3 celle conosciute



Con 3 celle conosciute

	N° Fire OK	N° Guess OK	N° Navi distrutte	Score
Agente1-Random	0	5	1	-299
Agente-2	5	9	7	195
Agente-3	4	11	7	245

Risultati Mappa 2 – con 5 celle conosciute



Con 5 celle conosciute

	N° Fire OK	N° Guess OK	N° Navi distrutte	Score
Agente1-Random	0	5	1	-299
Agente-2	4	7	5	105
Agente-3	3	11	8	285

Conclusioni-1

- ***Agente1-RANDOM***: è l'agente che rispetto agli altri ha il comportamento peggiore in tutti i casi poiché posiziona in maniera totalmente casuale le bandierine e inoltre non sfrutta né le fires e né eventuali info iniziali

- ***Agente2***: è in grado di superare nettamente l'agente precedente sia nei casi in cui la conoscenza iniziale è assente e sia quando essa è presente
  - *Sulla mappa1 all'aumentare della conoscenza iniziale continua a migliorare riuscendo ad ottenere uno score uguale a quello dell'agente-3 (con 3 celle conosciute)*
  - *Sulla mappa2 all'aumentare della conoscenza iniziale non è sempre in grado di riuscire a migliorare lo score finale (da 3 fatti iniziali a 5 fatti iniziali il suo score diminuisce)*

- ***Agente3***: è in grado di superare nettamente l'agente randomico e in **4 configurazioni su 6** è riuscito a battere nettamente l'agente-2 considerando sia casi nei quali c'era della conoscenza iniziale e sia casi nei quali la conoscenza iniziale era assente
  - *A differenza dell'agente-2, sia sulla mappa1 che sulla mappa2, all'aumentare della conoscenza iniziale continua a migliorare riuscendo ad ottenere uno score che è sempre maggiore o al più uguale a quello dell'agente-2*

Osservazioni:

- Dall'analisi dei risultati ottenuti possiamo dire che la preferenza (euristica) sul posizionamento delle navi seguito dall'agente-3 risulta essere migliore rispetto a quella dell'agente-2
- Sia l'agente-2 che l'agente-3 in quasi tutte le configurazioni testate hanno ottenuto performance migliori sulla mappa1, sintomo del fatto che la predisposizione centrale delle navi, dal punto di vista degli agenti, risulta essere più complessa da risolvere

Conclusioni-2

Limiti principali di Agente-2 e Agente-3:

- Dopo aver eseguito un'unguess (o una serie di unguess consecutive), anche se non riescono a posizionare nessuna nave (perché le navi che gli agenti credono di dover ancora piazzare sono troppo grandi), non riposizionano le bandierine che avevano tolto e di conseguenza questa cosa potrebbe portarli ad abbassare il loro score finale (cancellazione bandierina posizionata correttamente)..

✓ Per risolvere questo problema si potrebbero aggiungere ulteriori regole all'interno dei gestori per riuscire a correggere questo comportamento

- Dopo aver eseguito una fire e si scopre ad esempio un "left", se l'agente nella propria struttura interna, crede di poter posizionare, partendo da questa cella, solamente un incrociatore e una corazzata e queste due navi in realtà non può metterle perché a destra di quella cella_left magari può essere posizionato solo il "right", allora poiché l'agente assume di aver già posizionato tutti i cacciatorpedinieri, non sarà in grado di posizionare nessuna nave. L'unica cosa che si limiterà a fare in questo caso sarà posizionare la bandierina in corrispondenza della cella_left che ha scoperto.

✓ Per migliorare questo comportamento si potrebbero ad esempio aggiungere ulteriori regole all'interno dei gestori per poter permettere all'agente di poter scoprire e correggere un eventuale posizionamento errato ad esempio di uno dei cacciatorpedinieri che aveva già posizionato in precedenza

Grazie per l'attenzione