Sistemi Esperti - CLIPS

Carlo Alberto Barbano, Valentino Di Cianni, Francesco Iodice

1 Introduzione

Lo scopo di questo progetto è sviluppare alcune strategie sensate per una variante del classico gioco "Battaglia navale" in modalità giocatore singolo. Nonostante progettare delle strategie da far eseguire ad un sistema esperto in condizioni di conoscenza limitata e capacità di azione ridotta non è stato semplice, siamo riusciti a raggiungere risultati promettenti, con performance molto probabilmente simili a quelle che otterrebbe una persona nello stesso contesto.

2 Metodo

Di seguito sono riportate le strategie implementate per la risoluzione del problema del gioco "Battaglia navale" proposto per l'esame. Abbiamo implementato 3 strategie di gioco e le abbiamo testate in diversi scenari raccogliendo i risultati ottenuti. Abbiamo deciso di mantenere l'organizzazione dei moduli come proposta nel materiale fornito, facendo solo alcune modifiche al codice sorgente riguardanti principalmente il calcolo del punteggio finale (in quanto ci è sembrato leggermente sbilanciato per il gioco in questione) e alcuni punteggi di salience assegnati alle regole.

2.1 Strategia 1 - Baseline

Per questa prima strategia abbiamo deciso di cercare un punteggio di baseline da cui partire per sviluppare strategie più complesse. Con questo approccio, il più semplice, partiamo dai dati già presenti nella base di conoscenza, senza modificarli ed andando ad aggiungere solo fatti utili al proseguimento del gioco. Partendo da questo presupposto, l'unica mossa possibile risulta essere la guess: infatti, questa strategia di base consiste in una serie di guess in ordine decrescente di probabilità, esaurite le quali il gioco termina. Il metodo che utilizziamo per ottenere la posizione sulla quale fare *guess* è costruito basandosi sull'ipotesi che nell'intersezione riga/-colonna con il numero di celle occupate più grande c'è una probabilità maggiore di trovare una nave. Questa procedura restituisce quindi in ordine di probabilità tutte le coppie righe/colonne con presunte caselle nave in esse e, in maniera puramente iterativa, vengono asserite le *guess*.

Con questa prima strategia *Baseline* andiamo a modificare la base di conoscenza asserendo semplicemente le caselle su cui viene fatta la *guess* in modo tale da non ripetere la mossa due volte sulla stessa casella.

Modellazione della conoscenza Come detto nel paragrafo precedente, in questa prima strategia abbiamo solo modellato un fatto non ordinato:

```
(deftemplate v-cell
    (slot x)
    (slot y)
    (slot content
        (allowed-values water
                        guess
                        boat))
)
```

Grazie a questo fatto, possiamo tener traccia della posizione delle *guess* effettuate durante l'esecuzione della strategia. Il campo content contiene diversi campi valore non utilizzati in questo caso, ma li abbiamo comunque tenuti per uniformità con le altre strategie.

Per quanto riguarda la determinazione delle coordinate per la *guess* successiva abbiamo definito diverse regole ordinate in base alla salience. In particolare la prima (*tryMaxGuess*) unifica fatti *k-per-col* e *k-per-col* tali che siano i maggiori stretti rispetto agli altri fatti con ugual nome utilizzando la funzione *compareValMax*. Con il maggiore stretto siamo sicuri di avere un ordinamento decrescente, ma un possibile problema si presenta quando questi fatti terminano. Perciò, per evitare che la strategia termini prima del dovuto, abbiamo definito altre 3 regole (*try3Guess*, *try2Guess*, *try1Guess*) che, ordinate anche queste per salience, unificano fatti *k-per-row* e *k-per-col* con almeno 3, 2 o 1 caselle con navi. Tutte queste regole ovviamente impongono come vincoli che: *i)* le celle non siano già state segnate con *guess ii)* non siano celle conosciute a priori (*k-cell*) *iii)* ci siano ancora *guess* disponibili.

Questo criterio di scelta per le caselle su cui effettuare una *guess* è stato utilizzato anche nelle successive strategie.

Strategia di conflict resolution Per questa strategia, come per le successive abbiamo deciso di utilizzare la strategia di conflict resolution Breadth che ci permette di far attivare le regole attive più di recente dopo regole di pari salience. Utilizzando questa strategia siamo sicuri che le regole con salience maggiore vengano eseguite prima e quindi che vengano scelte come celle per la *guess* quelle con intersezione maggiore per prime.

2.2 Strategia 2 - Safe

Per questa seconda strategia abbiamo optato per un approccio strategico, ragionato e cauto, dove le *fire* vengono utilizzate solo se necessarie per portare avanti il gioco mentre le *guess* vengono asserite solo nei casi in cui la sicurezza di colpire il bersaglio è sufficientemente alta. La strategia è riportata nell'algoritmo 1.

Come detto in precedenza, abbiamo voluto pianificare una strategia "attenta" a non utilizzare azioni se non in presenza di una sicurezza abbastanza alta.

Modellazione della conoscenza Abbiamo creato quattro nuovi tipi di dati non ordinati:

- *boat*: template con uno slot per il nome e uno per il numero di barche disponibili. Questo dato ci serve per tenere traccia delle tipologie di navi ancora in gioco secondo quanto dedotto fino al quel momento.
- *kVisited*: con coordinate (x, y) rappresenta le caselle conosciute k-cell già visitate. Sfruttando questo dato evitiamo di andare a fare supposizioni su caselle conosciute già analizzate, concentrandoci solo su quelle nuove scoperte con la fire.
- *fired*: con coordinate (x, y) ci serve per tenere traccia delle caselle in cui è già stata effettuata una fire per evitare di sparare due volte in uno stesso punto.
- *guessed*: con coordinate (x, y) ci serve per tenere traccia delle caselle in cui abbiamo già effettuato una *guess* per non sprecare una mossa e per eventuale *unguess*.

Abbiamo deciso di non operare direttamente sui fatti creati dal modulo Env o dal modulo Main, ma di tenere traccia dell'andamento del gioco direttamente nella WM dell' Agent asserendo i fatti *kVisited*, *guessed*, *fired*.

Algorithm 1 Strategia 2

1. Considerare tutte le caselle conosciute inizialmente.
2. Se ci sono delle celle boat conosciute (k-cell) :
   - Se sono sottomarini viene aggiornato il conteggio interno delle navi abbattute aggiungendo il sottomarino in questione.
   - Se sono caselle *top*/*bottom* o *left*/*right* si controlla, a seconda dell'orientamento, il numero N di navi presenti nella colonna/riga. Vengono segnate con *guess* le $N-1$ caselle successive (tale che $N-1 \leq 3$ in quanto la nave più grande ha solo 4 celle), sempre tenendo conto dell'orientamento della nave. Viene poi aggiornato il fatto *boat* relativo a seconda della nave abbattuta decrementando il campo *num*.
   - Se invece si tratta di parti *middle*, si analizza se sono presenti più navi nella riga o nella colonna. Si seleziona il numero più alto, e vengono distribuite le *guess* in modo uniforme a sinistra/destra o sopra/sotto a seconda della scelta precedente (una da una parte e una dall'altra).
   - Una volta finite le *k-cell* con navi si procede al punto 3.
3. Se non ci sono celle boat conosciute:
   - Si scelgono le coordinate x e y tali che il numero di navi sulla rispettiva riga/colonna sia maggiore e non siano ancora state visitate (*known*, *guess* o *fire*). Dopo aver unificato le coordinate, viene eseguita una *fire* con quei valori. Se la *fire* trova una nave, vengono attivate le regole del punto 2. Se non trova nessuna nave, si passa alla coppia riga/colonna successiva. Si continua finché non si finiscono le *fire* o non si trova un'altra nave.
4. Qualora terminassero le *guess* a disposizione, ma fossero presenti ancora delle *k-cell* non analizzate scoperte tramite una *fire*, allora viene eseguita una *unguess* sulla casella intersezione riga/colonna con meno navi per permettere di continuare con la nave appena scoperta.
5. Al termine delle *fire* si decide di concludere il gioco, per evitare di perdere ulteriori punti con potenziali *guess* errate.

Strategia di conflict resolution Anche per questa strategia, abbiamo optato per la strategia di conflict resolution Breadth che ci permette di far attivare le regole attive più di recente dopo regole di pari salience. Utilizzando questa strategia siamo sicuri che le regole con salience maggiore vengano eseguite prima, considerando prima le celle *k-cell* e, in assenza di queste, facendo unificare la regola di *fire* per allargare la base di conoscenza.

2.3 Strategia 3 - Context-Aware

Sulla base dell'esperienza ottenuta con le prime due strategie, abbiamo elaborato questa nuova tattica di gioco cercando di andare a migliorare ogni aspetto delle precedenti. L'approccio utilizzato è sempre "safe" eseguendo solo azioni di cui si abbia una sicurezza molto alta. La differenza sostanziale di questa strategia è data da semplici assunzioni e deduzioni che vengono fatte durante la partita, in particolare:

- Da quanto detto durante una registrazione, si assume che intorno ad ogni nave ci sia almeno una cella *water* su tutti i lati.
- Qualora una riga/colonna avesse *k-per-col* o *k-per-row* uguale a 0, si deduce che nella riga/colonna selezionata tutte le celle siano *water*.

Fatte queste premesse la strategia è riportata nell'algoritmo 2.

Modellazione della conoscenza Abbiamo creato quattro nuovi tipi di dati non ordinati:

- *v-cell*: con coordinate (x, y) e content (*allowed-values water guess boat*) che rappresenta le caselle visitate. Sfruttando questo dato evitiamo di andare a considerare celle già analizzate.
- *fired*: con coordinate (x, y) ci serve per tenere traccia delle caselle in cui è già stata effettuata una fire per evitare di sparare due volte in uno stesso punto.
- *fireOk*: con slot numerico numFire che ci serve per tenere traccia del numero di fire andare a buon fine.

Strategia di conflict resolution Anche per questa strategia, come nelle due precedenti, abbiamo optato per la strategia di conflict resolution Breadth che ci permette di far attivare le regole attive più di recente dopo regole di pari salience. In questo caso specifico è molto importante, agendo sui valore di salience, che vengano attivate prima le regole che modificano il numero di celle occupate per riga e colonna e quelle che marcano una intera colonna o riga come *water*, cosicché le altre regole facciano affidamento su fatti aggiornati.

Algorithm 2 Strategia 3

1. Vengono segnate come *water* tutte le celle della riga/colonna che hanno *k-per-row* o *k-per-col* uguale a 0
2. Si prendono in considerazione le celle conosciute: se sono celle *boat* vengono segnate con *water* tutte le celle intorno e, a seconda del tipo di cella *boat*, viene segnata con *guess* la cella adiacente (le celle sinistra/destra o top/bottom a seconda dell'orientamento se si tratta di cella *mid*).
3. Qualora non ci fossero celle conosciute si esegue una *fire* seguendo le logiche della strategia 1 e 2.
4. A differenza delle prime due strategie, qualora venga colpita una nave tramite *fire*, vengono decrementati i counter di *k-per-col* e *k-per-row* di conseguenza, tenendo in considerazione non solo la cella *fire*
5. Una volta terminate le *fire*, vengono effettuate le *guess* rimanenti secondo il criterio probabilistico esposto in precedenza.

3 Discussione

Metodologia di sperimentazione Abbiamo testato le strategie su diversi campi di battaglia. Nella sezione Appendice sono riportate le immagini dei diversi terreni di gioco utilizzati.

Le quattro mappe sono state create sia in versione con *k-cell* sia senza. Nelle versioni con *k-cell* abbiamo usato due celle conosciute. Ci rendiamo conto che un numero maggiore di *k-cell* avrebbe sicuramente aumentato i punteggi ottenuti con le ultime due strategie, ma non volevamo che questo fattore influenzasse troppo i risultati. Nel primo terreno di gioco (*T/B*) le navi sono tutte posizionate sul limite superiore ed inferiore della mappa. Per quanto riguarda il secondo (*L/R*), le navi sono sul limite destro e sinistro, mentre nel terzo terreno di gioco (*C*) sono concentrate tutte nella zona centrale. L'ultima versione (*EQ*), quella più "realistica", ha le navi sparse per la mappa in modo casuale per simulare una situazione di gioco standard. Una visualizzazione delle mappe è riportata in appendice A.

Tabella 1: Punteggi strategia *Baseline*

	k-cell		no k-cell	
Mappa	**std**	**custom**	**std**	**custom**
T/B	-260	-30	-140	40
L/R	-105	90	-140	55
C	-35	130	115	250
EQ	0	150	-35	115

Tabella 2: Punteggi strategia *Safe*

	k-cell		no k-cell	
Mappa	**std**	**custom**	**std**	**custom**
T/B	-165	10	-125	55
L/R	-70	70	-90	40
C	-170	25	-140	5
EQ	-105	50	-85	80

Abbiamo distinto i test tra quelli eseguiti con la funzione standard fornita e quelli che utilizzano invece la funzione da noi modificata per un miglior bilanciamento dei punti. Di seguito i due metodi a confronto.

Funzione di score standard:

$$S_{std} = (10f_{ok} + 10g_{ok} + 15s_{sink}) - (25f_{ko} + 25g_{ko} + 10s_{safe}) \quad (1)$$

Funzione di score custom:

$$S_{custom} = (20f_{ok} + 10g_{ok} + 25s_{sink}) - (25f_{ko} + 5g_{ko} + 5s_{safe}) \quad (2)$$

dove f_{ok} e g_{ok} rappresentano rispettivamente il numero di *fire* e *guess* andate a buon fine e s_{sink} il numero di navi affondate, mentre f_{ko} e g_{ko} rappresentano il numero di *fire* e *guess* errate e s_{safe} il numero di navi non affondate.

Le motivazioni che ci hanno portato a modificare la funzione di scoring sono le seguenti:

1. Il reward per una *fire* andata a buon fine, secondo noi, va premiato con un punteggio superiore, soprattutto a confronto con quelli che sono i punteggi negativi per le azioni sbagliate.
2. Come per il *fire*, affondare una nave ci sembrava essere meritevole di un aumento di punteggio, vista la "rarità" dell'evento e quello che è lo scopo del gioco.
3. I punteggi negativi di *guess* e *safe* ci sembravano troppo sbilanciati, in quanto: per la *guess*, essendo una supposizione, non è corretto togliere così tanti punti; per le *safe* abbiamo leggermente abbassato il punteggio poiché abbattere una nave intera è un task molto difficile con le risorse a disposizione del gioco.

Analisi dei risultati In Tabella 1 sono riportati i risultati per la prima strategia *Baseline*. Con *k-cell* e *no k-cell* è indicata la presenza di celle conosciute a priori nella mappa, mentre *std* e *custom* indicano la funzione di punteggio utilizzata, rispettivamente quella standard e quella custom.

Come si può notare dai risultati, con entrambi i punteggi, questi valori cambiano di molto al variare della mappa, con un trend che sembra confermare un punteggio maggiore nel caso in cui la disposizione delle navi sia uniforme o spostata verso il centro, mentre un punteggio minore nei casi in cui le navi siano disposte verso i bordi della mappa.

Tabella 3: Punteggi strategia *Context Aware*

	k-cell		no k-cell	
Mappa	**std**	**custom**	**std**	**custom**
T/B	95	285	-115	100
L/R	-85	285	-30	160
C	145	345	75	260
EQ	115	370	-10	175

Interessante osservare come si ottenga un risultato diametralmente opposto con la seconda strategia *Safe*, riportata in Tabella 2: si ottengono punteggi migliori rispetto alla baseline in mappe con navi disposte in modo sbilanciato, mentre si peggiorano i risultati rispetto a mappe con navi uniformi. Dal momento che il criterio per le *fire* della seconda strategia è il medesimo per i *guess* della *Baseline*, si può supporre che piazzando le *guess* in prossimità delle *fire* si dimuisca la probabilità di successo delle future *fire*, dal momento che dovranno essere effettuate in luoghi della mappa con una minore concentrazione di navi. Per questo motivo otteniamo risultati peggiori della baseline con mappe come la *C* e *EQ* dove le navi sono concentrate nel centro oppure sono disposte in modo uniforme. Inoltre, poiché non vengono utilizzate le *fire* per la strategia *Baseline*, il punteggio non subisce crolli drastici dovuti alle fko, cosa che non avviene invece con la strategia 2. Di contro, nella baseline i punteggi totali ottenuti nei caso in cui fossero indovinate tutte le navi (best case) sono limitati superiormente dall'assenza di fok, discorso opposto per la strategia 2.

Per quanto riguarda la terza strategia *Context-Aware*, i risultati ottenuti, riportati in Tabella 3, sono nettamente migliori sia rispetto a *Baseline* che a *Safe*. In particolare, nella mappa *C* si ottengono, con il punteggio custom, dei valori vicini a quelli massimi. Come ci aspettavamo, nelle

mappe senza celle conosciute a priori si nota un peggioramento deciso sulle performance, che pur si mantiene superiore ai punteggi ottenuti con le altre strategie. Questo è, purtroppo, un limite che non possiamo evitare, poiché l'unico modo che abbiamo per aumentare la nostra base di conoscenza è eseguire le *fire*, che non sono mai a colpo sicuro.

Possibili miglioramenti potrebbero riguardare lo sviluppo di funzioni euristiche decisionali piú complesse rispetto all'ordinamento probabilistico utilizzato, o anche produrre una serie di configurazioni di navi, che soddisfano i vincoli posti dal gioco, usando strategicamente le azioni di tipo *fire* per decidere la configurazione piú probabile.

A Appendice

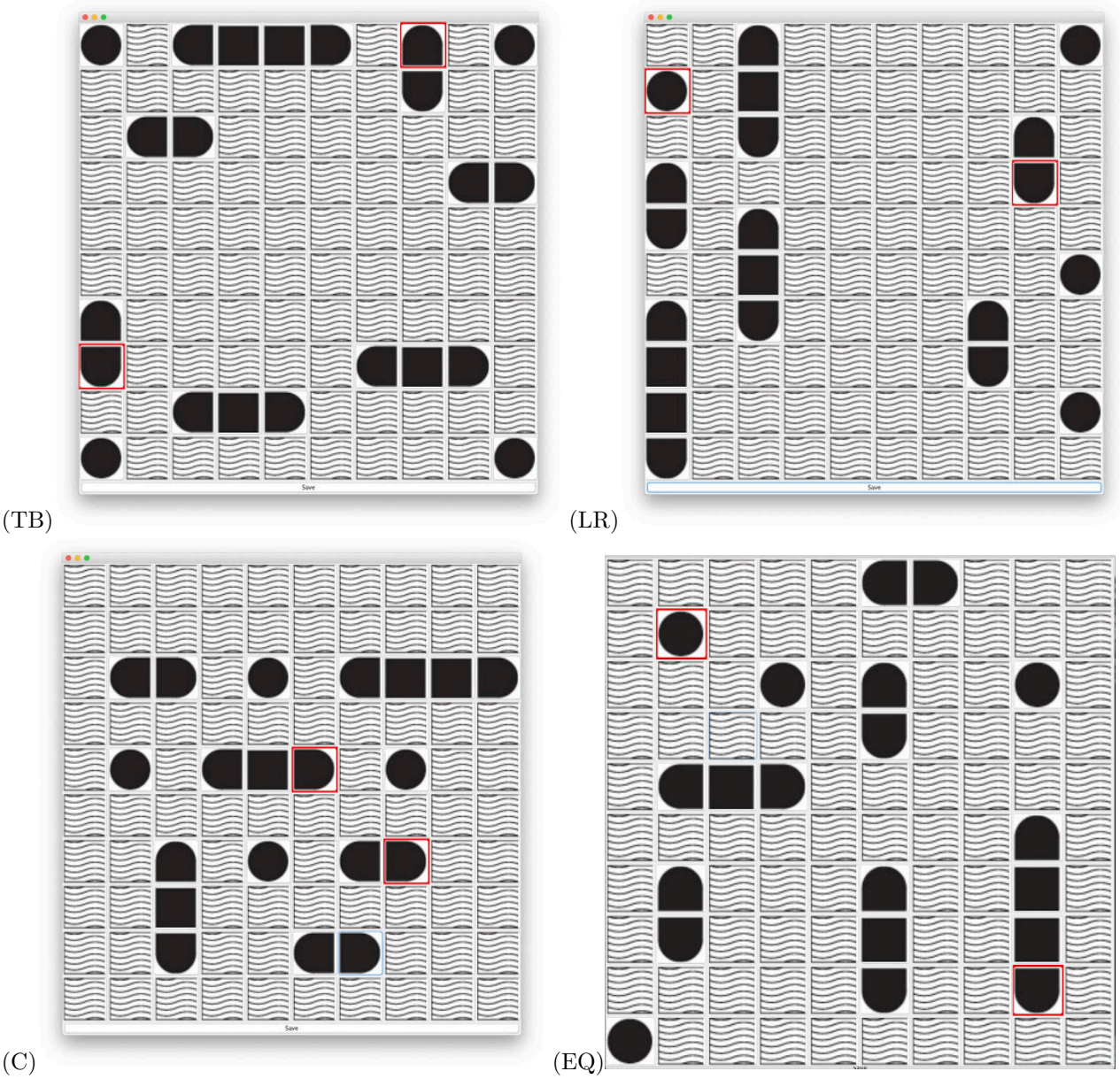

Figura 1: Visualizzazione delle mappe testate. In rosso sono evidenziate le *k-cell*