

Linguaggi di Programmazione: Paradigmi di Programmazione

Corso di Laurea in Informatica

Algoritmo di risoluzione del gioco $1\frac{1}{4}$

30 ottobre 2007

Docente
Gianni Aguzzi

Relazione di
Fabio Duchi
2751316
(fabio.duchi@gmail.com)

ANNO ACCADEMICO 2007-2008

Indice

1	Prefazione	2
1.1	Descrizione e Regole	2
1.2	Interfaccia	4
2	Strategie di implementazione	6
2.1	Rappresentazione dei dati	6
2.2	Coppie da eliminare	7
2.3	Strategia di individuazione della miglior coppia	8
2.3.1	Individuazione delle mosse non applicabili	8
2.3.2	Valutazione della qualità delle mosse	10
2.4	deadlock detection / deadlock avoiding	11
2.5	Risoluzione	12
2.6	Stampa	13
3	Esempio	14
4	Listato	18

Prefazione

1

Il progetto ha come obiettivo quello di individuare una strategia vincente per risolvere il gioco 14, a tale scopo l'algorithmo deve valutare la bontà delle mosse possibili. Questa parte del progetto è quella che ha richiesto la maggior attenzione in quanto necessita di un buon bilanciamento tra efficienza e qualità delle valutazioni.

1.1 Descrizione e Regole

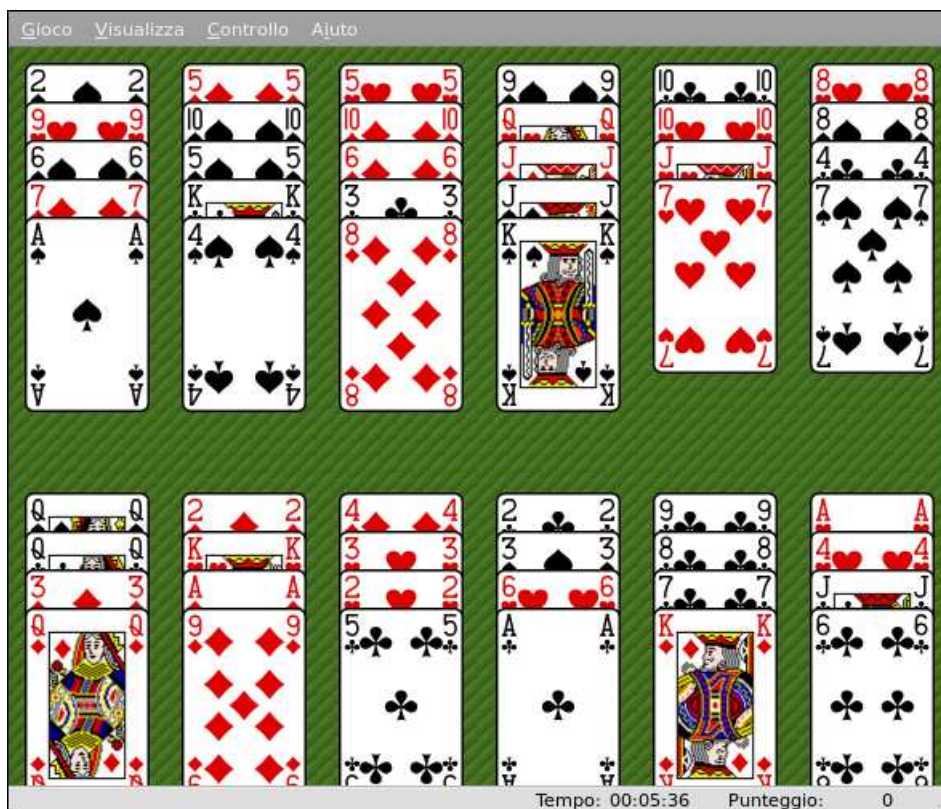


Figura 1.1: Esempio di gioco

Il gioco 14 o *fourteen* utilizza un mazzo di 52 carte, distribuite su 12 colonne; in particolare le prime 4 colonne sono costituite da 5 carte, le successive da solo 4 carte.

L'obiettivo è eliminare le 52 carte sul tavolo, 2 alla volta, scelte tra le carte che stanno nella testa delle colonne, rispettando il vincolo che la somma dei valori associati alle 2 carte sia 14; per le carte dal 3 al 10 i valori sono i valori nominali delle stesse, mentre l'asso vale 1, il *Jack*, *Donna* e il *Re*, valgono rispettivamente 11, 12 e 13.

La difficoltà principale di questo gioco sono le situazioni di *deadlock* che si possono presentare in seguito a scelte sbagliate, come nella seguente immagine:

deadlock

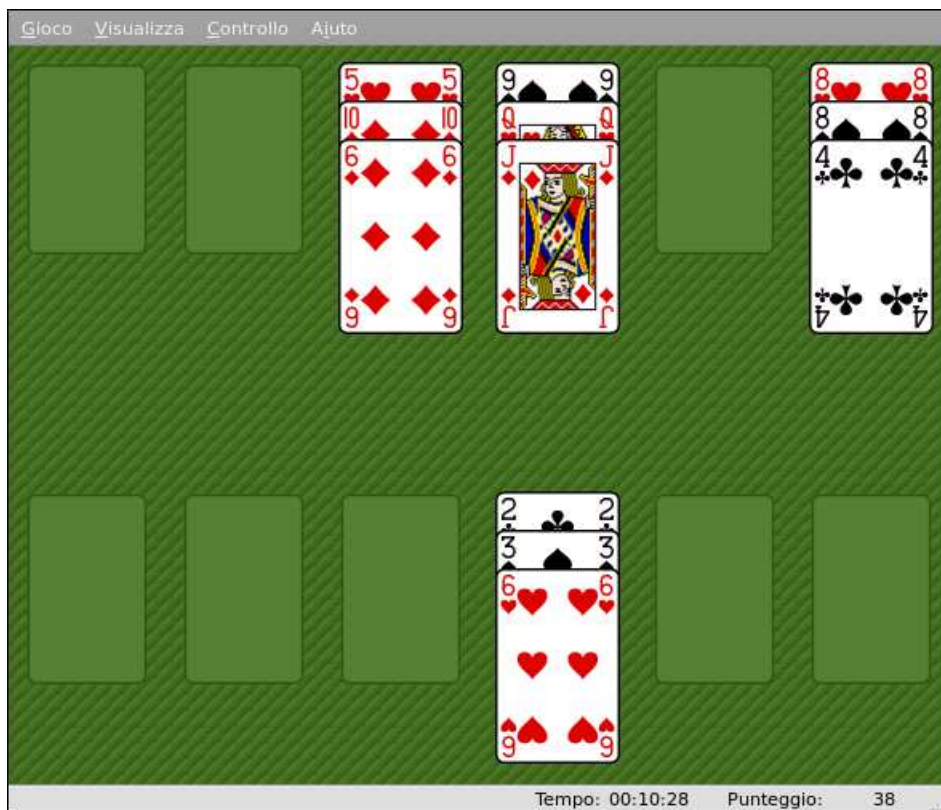


Figura 1.2: Sconfitta per deadlock

Consideriamo la 3° colonna contenente le carte 6, 10, 5, per poter eliminare la carte in testa alla colonna è necessario un 8; troviamo l'8 nella 6° colonna il quale per essere utilizzato richiede che prima il 4 debba essere eliminato; per eliminare il 4 è necessario il 10 della 3° colonna che è bloccato dal 6.

1.2 Interfaccia

Uno dei metodi pubblici fornito dal programma è *risolvi(+Stato, ?Mosse)*, che come suggerisce il nome stesso del metodo restituisce una strategia vincente per completare il gioco:

- *Stato*, come ne suggerisce il nome, rappresenta lo stato attuale del gioco ossia le carte sul tavolo e la loro disposizione; più nel dettaglio le 12 colonne vengono rappresentate come una lista di 12 elementi, ciascuno dei quali è ancora una lista, la cui testa rappresenta l'elemento immediatamente raggiungibile della colonna (ovviamente non è esplicitamente richiesto che lo stato corrisponda allo stato iniziale del gioco);
- *Mosse*, rappresenta la lista di azioni da eseguire sullo stato fornito per risolvere il gioco.

Per concentrarmi sul problema principale, alcuni aspetti legati all'usabilità sono stati trascurati, benchè possano essere facilmente implementati, ad esempio il Jack, la Donna, ed il Re sono rappresentati dai numeri 11, 12 e 13, ossia dai valori ad essi associati; nella lista delle Mosse per risolvere il gioco, sono elencate solamente le colonne dalle quali recuperare le carte da eliminare, questo perchè per ogni colonna può essere giocata solo la carta in cima alla colonna.

Per verificare la correttezza del risultato è stato implementato anche un semplice interfaccia testuale che visualizza l'evoluzione del sistema mano a mano che le mosse vengono applicate. Questa schema è fornito separatamente alla funzione *risolvi*, tramite le regole *stampa(+Stato, +Mosse)*. Un esempio del risultato prodotto è dato:

```

...

      - -
8      7
7 7 9 7
12 10 1 2      5      5 6      3
9 9 2 1 11 3 4 10 11 6      1
10 8 5 11 3 13 12 13 4 13      4

```

```
      8      7      -      -
      7 7 9
12 10 1 7      5      5 6      3
  9 9 2 2      3 4 10 11 6      1
10 8 5 1 11 13 12 13 4 13      4
      ...
```

Le colonne rappresentano le colonne di carte nello stato attuale, gli underscore sopra le colonne indicano invece quali carte verranno eliminate e quindi da un'anticipazione sull'evoluzione del sistema.

La regola *risolvi_stamp*(**+Stato**), raggruppa le due operazioni precedentemente distinte.

Strategie di implementazione

2

2.1 Rappresentazione dei dati

Per astrarre lo stato di gioco dalla rappresentazione dello stesso nel calcolatore ho definito un set di istruzioni a questo scopo.

```
1 colonna( [Pilal, _, _, _, _, _, _, _, _, _, _, _], 1, Pilal).
2 /*...*/
3 colonna( [_ , _ , _ , _ , _ , _ , _ , _ , _ , _ , _ , _], 12, Pilal2).
4
5 decapita([[_|Pilal], P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, Step], 1,
6         [Pilal, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, Step]).
7 /*...*/
8 decapita([P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, [_|Pilal2], Step],
9         12, [P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, Pilal2, Step]).
10
11 testa(Stato, N, X) :- colonna(Stato, N, [X|_]),!.
12
13 passo([_,_,_,_,_,_,_,_,_,_,_,_,[Step]], Step).
14 passo_successivo([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,[Step]],[P1,P2,P3,P4,
15                   P5,P6,P7,P8,P9,P10,P11,P12,[Step0]]):-
16     Step0 is Step + 1.
```

Listing 2.1: Astrazione dai dati

Lo stato del gioco viene internamente rappresentato mediante una lista di liste, ciascuna con uno scopo preciso: le prime 12 liste rappresentano le carte presenti sul tavolo di gioco, ogni colonna è un mazzetto di carte. La lista è ordinata in modo tale da avere nel primo elemento la prima carta del mazzetto (cioè quella che può essere utilizzata nelle mosse), nel secondo elemento la seconda carta, e così via. Le carte sono a loro volta rappresentate da un numero; poiché il gioco non fa distinzione tra i semi delle carte, ma unicamente tra i valori che esse rappresentano. Il numero con cui è rappresentata la carta rappresenta il valore della carta stessa, come già detto per il gobbo, la donna ed il re si utilizzano i valori 11, 12 e 13.

La 13° lista che costituisce lo stato del gioco contiene informazioni “accessorie”, informazioni che vengono utilizzate internamente dall’algoritmo ma di cui un giocatore umano non avrebbe bisogno. Un esempio di queste informazioni è

l'indice del passo corrente; le regole per modificare questa informazione sono alla righe 11 e 12.

La rappresentazione dello stato con una lista da interpretare si è rivelata particolarmente buona, sia sotto un profilo di “possibilità di arricchire” l’algoritmo (è stato infatti estremamente semplice aggiungere l’informazione sul passo, che inizialmente non era prevista), sia sotto il profilo della chiarezza del sorgente, che è stato tra i problemi maggiori.

Ad ogni passo del gioco sono necessarie due operazioni fondamentali, l’individuazione delle possibili coppie di carte da eliminare, e l’individuazione della miglior mossa (ossia la coppia di carte da eliminare che ci permette di progredire più velocemente nel gioco).

2.2 Coppie da eliminare

Sfruttando la rappresentazione delle colonne come liste, la cui testa è l’elemento direttamente accessibile, in modo semplice si definiscono le regole:

```

1 coppia_candidata(_,_, [], L_append, L_append).
2 coppia_candidata(Stato, X, [X|L2], L3, L_append) :-
3   coppia_candidata(Stato, X, L2, L3, L_append).
4 coppia_candidata(Stato, X, [Y|L2], [[X,Y]|L3], L_append) :-
5   X<Y,
6   testa(Stato, X, X1), testa(Stato, Y, Y1),
7   X1+Y1 =:= 14, coppia_candidata(Stato, X, L2, L3, L_append).
8 coppia_candidata(Stato, X, [_|L2], L3, L_append) :-
9   coppia_candidata(Stato, X,L2,L3, L_append).

11 removable1(Stato, [X|L], L2, Li, Lo) :-
12   coppia_candidata(Stato, X,L2,L3, Li),
13   removable1(Stato, L,L2,L3, Lo).
14 removable1(_, [], _, L_append, L_append).
15 candidati_eliminazione(Stato, X) :-
16   removable1(Stato, [1,2,3,4,5,6,7,8,9,10,11,12], [1,2,3,4,5,6,
      7,8,9,10,11,12], [], X),!.

```

Listing 2.2: Ricerca coppie candidate

per non eseguire due volte il solito controllo, una con l’ordine degli addendi invertiti rispetto all’altra, si limita in *linea 5* al solo caso in cui $X < Y$.

La struttura scelta, in particolare semplifica la regola *coppia_candidata*, poiché è sufficiente controllarne solo la testa per verificare che la colonna rappresentata dalla lista sia, oppure no, coinvolta in una possibile eliminazione.

Ovviamente il controllo deve essere fatto su tutte le liste non vuote.

2.3 Strategia di individuazione della miglior coppia

Quando il numero di coppie candidate per la successiva azione è maggiore di 1, è necessario eseguire una scelta. Questo passo è molto importante per l'efficienza del metodo in quanto mosse differenti portano il gioco in stati differenti, più o meno semplici da risolvere, se non addirittura impossibile da risolvere. In casi di fallimento (impossibilità di risolvere il gioco) sono necessari dei rollback, che per quanto siano semplici in prolog, sono operazioni costose.

Per scegliere la miglior coppia nel pool, si cerca di valutarne la bontà in funzione di proprietà e/o osservazioni sui loro effetti, inizialmente era stata implementata un funzione euristica che cercava di determinare la "qualità" del sistema se l'azione fossa stata eseguita. Il vantaggio di questo approccio è l'efficienza nell'operazione di scelta, fondamentale qui poiché tali operazioni verranno eseguite diverse volte per ciascuna azione eseguita. Per l'alta probabilità di incappare in situazioni fallimentari ho, successivamente, preferito scomporre il problema in due problemi, non più semplici in assoluto, ma per lo meno distinti tra di loro e tali da poterne scrivere due set di funzioni specifici e indipendenti.

Ho cercato anzitutto di individuare un algoritmo per rilevare le mosse che avrebbero portato ad uno stato non accettabile il sistema, ossia alla sconfitta; un secondo algoritmo avrebbe, mediante funzioni basate su mie osservazioni, associato a ciascuna mossa un fattore che ne rappresenta la "qualità" della stessa, ossia la probabilità che l'applicazione di tale mossa porti ad uno stato più semplice da risolvere.

2.3.1 Individuazione delle mosse non applicabili

```

1 notdeadlock3(_ ,X, Y, _, _, Exclude) :- member([X,Y], Exclude), !, fail.
2 notdeadlock3(Stato, _, [], Stato, _, _).
3 notdeadlock3(Stato, X, Y, StatoFinal, AlreadyVisited, Exclude) :-
4   testa(Stato, Y, X1),
5   (
6     X:=X1, !, StatoFinal=Stato;

8     coppia_candidata2(Stato, Y, [1,2,3,4,5,6,7,8,9,10,11,12], [[Y,Z]|_]),
9     applica_regola(Stato, Y, Z, Stato0),
10    notdeadlock3(Stato0, X, Y, StatoFinal, AlreadyVisited, Exclude)
11   ).
12 notdeadlock3(Stato, X, Y, StatoFinal, AlreadyVisited, Exclude) :-
13   testa(Stato, Y, Y0),
14   Y0=\=X,
15   Y1 is 14-Y0,
16   occorrenze_nelle_colonne(Stato, Y1, YListPrec0),
17   elimina_occorrenze(Y, YListPrec0, YListPrec),

```

```

18 notdeadlock2(Stato, Y1, YListPrec, Stato0, AlReadyVisited, [[X,Y]|Exclude])
19     ,
19 coppia_candidata2(Stato0, Y, [1,2,3,4,5,6,7,8,9,10,11,12], [[Y,Z]|_]),
20 applica_regola(Stato0, Y, Z, Stato1),
21 notdeadlock3(Stato1, X, Y, StatoFinal, AlReadyVisited, Exclude).

24 notdeadlock25(Stato, X, [Y|XListPrec], FinalStato, AlReadyVisited, Exclude)
25     :-
26     (
26     notdeadlock3(Stato, X, Y, FinalStato, AlReadyVisited, Exclude);
27     notdeadlock25(Stato, X, XListPrec, FinalStato, AlReadyVisited, Exclude)
28     ).
29 notdeadlock25(_, _, [], _, _, _) :- fail.

31 notmember(X, [X|_]) :-
32     !, fail.
33 notmember(X, [_|L]) :-
34     notmember(X,L).
35 notmember(_, []).

37 notdeadlock2(Stato, X, XListPrec, FinalStato, AlReadyVisited, Exclude) :-
38     notmember(X, AlReadyVisited),!,
39     notdeadlock25(Stato, X, XListPrec, FinalStato, [X|AlReadyVisited], Exclude)
39     ,!.

42 notdeadlock1(Stato, [X|L], Acc, Conta) :-
43     occorrenze_nelle_colonne(Stato,X, XListPrec),
44     length(XListPrec, Len),
45     (
46     Len == 0;
47     notdeadlock2(Stato, X, XListPrec, _,[],[])
48     ),
49     Conta0 is Acc+1,
50     notdeadlock1(Stato, L, Conta0, Conta).

52 notdeadlock1(Stato, [_|L], Acc, Conta) :-
53     notdeadlock1(Stato, L, Acc, Conta).
54 notdeadlock1(_, [], Conta, Conta).

56 notdeadlock(Stato, Conta) :-
57     notdeadlock1(Stato, [1,2,3,4,5,6,7,8,9,10,11,12,13], 0, Conta),!.

```

Listing 2.3: Deadlock detection

Per determinare se una mossa è non applicabile si applica la mossa stessa allo stato corrente in un'ambiente sandbox, si valuta quindi se tale ambiente presenta dei deadlock. Per deadlock in questo contesto intendo una situazione come descritta in §1.

La funzione che determina se una carta è in deadlock è *notdeadlock3*. Per determinare se la carta **X** è in deadlock si cerca almeno un percorso valido che permetta di avere tale carta in testa alla colonna che la contiene; questo significa tentare di eliminare ciascuna carta (se ne esistono) che precede la carta **X**.

Se la carta non è immediatamente accessibile si cerca di eliminare la prima carta della colonna **Y** (cercandone la sua complementare); se la carta complemen-

tare è immediatamente accessibile, allora la coppia viene eliminata dallo stato e si ritorna a verificare se la carta **X** sia ora in testa. Se la carta complementare non fosse accessibile, allora prima di dichiarare lo stato di deadlock della carta **X** si tenta di eliminare le carte che precedono la carta complementare rendendo questa (se possibile) accessibile.

Anche se le funzioni utilizzate sono pressappoco le stesse del metodo *risolvi*, in questo caso non si richiede che il percorso sia il migliore, ma solo che il percorso esista; inoltre l'operazione non coinvolge solo le carte in testa alle colonne ma ricerca per ciascun valore distinto, almeno una carta accessibile che lo rappresenti.

Due strutture di appoggio vengono utilizzate per motivi di efficienza ed efficacia dell'algoritmo; nel particolare la struttura Visited serve per indicare le carte che non sono immediatamente accessibili, ma ciascuna di esse lo sarà se viene individuato un percorso che elimina tutte le carte che la precedono; conseguentemente se viene individuata una carta che richiede un elemento che già appartiene a tale struttura allora si è individuato un deadlock.

La struttura Exclude è invece utilizzata per non tentare di eliminare più volte la stessa carta, quando la funzione *notdeadlock3* è costretta a cercare di liberare la complementare della carta **X**, impedisce alle funzioni chiamanti di modificare la colonna che la contiene bloccandone la carta, quindi parzialmente lo stato del sistema coinciderà con quello antecedente la chiamata ricorsiva (indiritta).

L'algoritmo elimina dal paniere delle possibili mosse successive quelle che, certamente, portano ad uno stato di deadlock. Così facendo non solo si evitano situazioni "problematiche", ma anche si riduce il numero di dati da dover elaborare nella valutazione della migliore delle mosse possibili.

2.3.2 Valutazione della qualità delle mosse

```

1  prioritazione(Stato, X, Y, Priorita) :-
2    applica_regola(Stato, X, Y, Stato2),
3    notdeadlock(Stato2, Priorita0),
4    Priorita0:=13,
5    colonna(Stato, X, P1),
6    colonna(Stato, Y, P2),
7    length(P1, X1),
8    length(P2, Y1),
9    (
10     X1>Y1, !, Priorita is X1;
11     Priorita is Y1
12    ).

```

Listing 2.4: Valutazione mossa

Tramite delle osservazioni ho notato che per aver maggior probabilità di successo in questo gioco è opportuno avere il più ampio ventaglio di carte accessibili,

questa osservazione si traduce nella preferenza di applicare prima le mosse che coinvolgono mazzi di carte + numerosi; tale scelta permette infatti di avere un maggior numero di mazzi da cui poter scegliere le carte e di ridurre i mazzi più numerosi.

Altre euristiche si possono accompagnare a questa analisi, ad esempio in una versione precedente (di cui riporto un brandello di codice) avevo inserito un controllo sul numero di carte diverse accessibili dopo l'applicazione della mossa

```
1 testa_distinte_carte0(_, [], []).
2 testa_distinte_carte0(Stato, [X|Colonne], [X1|Valori]) :-
3   testa(Stato, X, X1),
4   testa_distinte_carte0(Stato, Mosse, Valori).
5 testa_distinte_carte(Stato, Colonne, Valori) :-
6   testa_distinte_carte0(Stato, Colonne, Valori0),
7   unique(Valori0, Valori).
```

Ovviamente anche questa euristica potrebbe essere utilizzata, da sola, o in congiunzione con quella implementata.

Per lasciare l'algorithmo più leggibile possibile ho preferito non appesantire ulteriormente il sorgente eliminando codice non necessario (oltre questo set di funzione era necessario inserire anche un set di funzioni per comporre la valutazione basata sulla dimensione dei mazzetti, con questa valutazione).

A questo punto la mossa da applicare può essere facilmente individuata ordinando le mosse possibili in base alla priorità assegnata, ed ovviamente tentare di risolvere il puzzle scegliendo per prima la mossa con maggior priorità.

2.4 deadlock detection / deadlock avoiding

Il set di funzioni per la rilevazione delle mosse non applicabili costituiscono un'analisi di deadlock semplificato, poiché si accontenta di accertarsi che almeno una carta di ciascuno dei 13 distinti valori sia accessibile. Questa scelta è stata fatta per motivi di efficienza. E' possibile vedere l'algorithmo come una versione semplificata di deadlock avoiding, che per la voluta semplicità non è completo, ossia ci sono delle situazioni di deadlock che non vengono individuate. L'algorithmo è però corretto, infatti ad ogni risposta affermativa corrisponde una situazione di deadlock.

Questa snella individuazione delle mosse non applicabili, ha il costo di potersi rivelare alle volte sbagliata, costringendo l'applicazione ad operazioni di backtracking. Da analisi fatte sull'andamento dell'algorithmo, mi sono accorto che, spesso, dopo una decisione sbagliata ne seguono diverse altre prima che l'algorithmo stesso si accorga di aver sbagliato.

In questo caso è esoso, in termini prestazionali, appoggiarsi al sistema di back-

tracking integrato in prolog, poiché prima di individuare una nuova scelta giusta è necessario esaminare decine e decine di altre scelte certamente sbagliate.

Ho quindi implementato un algoritmo di deadlock-detection, il cui kernel è condiviso con l'algoritmo precedente, esteso a tutte le 52 carte. Vengono analizzate tutte le carte alla ricerca di almeno una carta in deadlock, questo viene fatto andando a individuare per ciascuna carta una sequenza di mosse che permettono di eliminarla dal tavolo. Se esiste una carta per la quale non esiste alcuna sequenza che permette di eliminarla dal gioco, allora certamente siamo in una situazione di deadlock che impone all'algoritmo di ritornare sui suoi passi.

Benché la funzione che individua il deadlock è molto vicina alla funzione del deadlock avoiding, le funzioni di preparazione dei dati sono profondamente diverse dalle rispettive controparti; in questo caso, tra le tante piccole ma sostanziali differenze, si tengono in considerazione che le carte sono distinte anche per il seme e per la loro posizione nei mazzi, quindi se prima ci si accontentava di prendere la prima carta del valore desiderato nel mazzetto corrente, adesso si esaminano tutte le occorrenze della carte di stesso valore nel medesimo mazzo costringendo l'algoritmo a trattare anche stati dipendenti all'interno della sandbox (se due carte di medesimo valore si presentano nel solito mazzo è ovvio che è necessario eliminare la prima occorrenza per eliminare la seconda, quindi per non fare gli stessi calcoli due volte, una volta verificata che la prima occorrenza non è in deadlock si procede tentando di eliminare la seconda partendo dal precedente stato raggiunto). Un'altra piccola, ma fondamentale differenza, è la gestione di tutte le possibili mosse effettuabili per eliminare le carte che precedono la carta a cui siamo interessati.

Alla luce di quanto esposto, si potrebbe obiettare che essendo quest'ultimo algoritmo corretto e completo, diversamente dal precedente, potrebbe essere utilizzato per determinare la successiva mossa, evitando completamente i rollback. Purtroppo determinare se tutte le carte sono in deadlock richiede, la risoluzione del puzzle con gravi, ed ovvie, conseguenze sull'impatto prestazionale dell'algoritmo; diventa però utile nelle ultime fasi di elaborazione quando il numero di carte in gioco è esiguo, e sono sufficienti un numero ridotto di confronti per determinare se siamo di fronte ad una situazione irrisolvibile.

2.5 Risoluzione

I metodi per la risoluzione utilizzano l'insieme di coppie candidate per tentare di risolvere il gioco applicando allo stato corrente, la mossa scelta. Per non appesantire il calcolo sulla scelta delle mosse da applicare non si assicura che tutte le mosse portino alla risoluzione del sistema, la regola *risolvi*, deve pertanto

recuperare da situazioni “indesiderate” considerando una delle alternative della mossa utilizzata all'ultimo passo.

```

1  risolvil(Stato, [[_,X,Y]|_], [[X,Y]|Finale]) :-
2    applica_regola(Stato, [X,Y], Stato0),
3    risolvi(Stato0, Finale).

5  risolvil(Stato, [_|Candidates], Finale) :-
6    notdeadlock_fullcheck(Stato),
7    risolvil(Stato, Candidates, Finale).

9  risolvil(_, [], _) :- fail.

11 risolvi([], [], [], [], [], [], [], [], [], [], [], [], []).

13 risolvi(Stato, Finale) :-
14  candidati_eliminazione(Stato, Candidates),
15  candidati_con_priorita(Stato, Candidates, Queue),
16  risolvil(Stato, Queue, Finale).

```

Listing 2.5: Risoluzione

La variabile **Finale**, conterrà la lista, in ordine di esecuzione, delle mosse applicate per la risoluzione del sistema.

Per rendere efficiente queste operazioni è stata utilizzata sempre la ricorsione in coda, costruendo ad ogni passo un solo elemento della lista **Finale**. In alcune funzioni ricorsive l'utilizzo del costrutto *cut* è stato reso necessario per i problemi legati al backtracking in grandi strutture, senza il *cut*, l'occupazione di memoria sarebbe troppo elevata producendo, per sistemi complessi, errori tipo “Out of Stack”.

2.6 Stampa

La funzione *stampa*, visualizza gli stati attraversati dal sistema durante lo operazioni di risoluzione del sistema stesso, il jack, la donna ed il re, vengono rappresentati con i loro valori nominali, questa scelta è stata fatta per favorire la leggibilità delle scelte effettuato.

Per evidenziare quali mosse vengano fatte dall'algorithm sulle colonne coinvolte durante le operazioni viene posto un underscore.

3

Esempio

Gli esempi qui riportati vogliono essere delle discussioni sul comportamento dell'algorithm, quindi per non sprecare fogli inutilmente vengono riportati solo alcuni dei 26 passaggi che compongono la soluzione.

```
?- risolvi_stampa(  
  [[7,10,10,1,4],  
   [13,6,11,5,4],  
   [8,11,7,5,11],  
   [3,12,7,3,3],  
   [8,2,4,1],  
   [6,11,10,8],  
   [9,5,4,12],  
   [2,9,7,1],  
   [12,13,13,13],  
   [10,1,9,12],  
   [8,9,2,6],  
   [5,2,6,3]])).
```

```
      -      -  
4  4  11  3  
1  5  5  3  1  8 12  1 13 12  6  3  
10 11  7  7  4 10  4  7 13  9  2  6  
10  6 11 12  2 11  5  9 13  1  9  2  
7 13  8  3  8  6  9  2 12 10  8  5  1
```

```
      -      -  
4  4    3  
1  5 11  3  1    12  1 13 12  6  3  
10 11  5  7  4  8  4  7 13  9  2  6  
10  6  7 12  2 10  5  9 13  1  9  2  
7 13 11  3  8 11  9  2 12 10  8  5  2
```

```

                                - -
4 4
1 5 11 3 1      12 1 13 12 6 3
10 11 5 3 4     4 7 13 9 2 6
10 6 7 7 2 8 5 9 13 1 9 2
7 13 11 12 8 10 9 2 12 10 8 5 3

```

```

...           ...           ...           ...

```

```

                                - -

```

```

11 3      8                6      25

```

```

- -

```

```

11 3

```

```

26

```

In questo caso ad ogni passo sono state effettuate le scelte corrette che hanno permesso di evitare situazioni di deadlock; se durante l'elaborazione l'algoritmo individua delle situazioni irrisolvibili, dovute ad errate scelte precedenti, informa l'utilizzatore di quanto avvenuto e al tempo stesso individua un percorso diverso di soluzione.

Di seguito è riportato un esempio che mostra il caso sopra descritto:

```

?- risolvi_stampa(
[[11, 13, 11, 11, 3],
[7,12,1,12,11],
[8,13,2,13,5],
[9,5,8,6,8],
[3,2,7,10],
[12,9,6,7],
[9,1,4,4],
[6,5,3,5],
[12,2,6,9],

```

[10,1,1,4],
 [7,3,8,10],
 [2,13,10,4]]).

```
/*deadlock detected-----
 3           4           4
11      5           4           1 10 4
11 11 13 8 10     1 5 9 1 8 10
13 12 2 6 7 7 9 3 6 10 3 13 11
-----*/
```

```
/*deadlock detected-----
 3           4           4
11      5 8           4           1 10 4
11 11 13 6 10 7 1 5 9 1 8 10
13 12 2 8 7 6 9 3 6 10 3 13 10
-----*/
```

```
3 11 5 8           -
11 12 13 6 10 7 4 5 9 4 10 4
11 1 2 8 7 6 4 3 6 1 8 10
13 12 13 5 2 9 1 5 2 1 3 13
11 7 8 9 3 12 9 6 12 10 7 2 1
```

```
3 11           -           -
11 12 5 6 10 7 4           9 4 10 4
11 1 13 8 7 6 4 5 6 1 8 10
13 12 2 5 2 9 1 3 2 1 3 13
11 7 13 9 3 12 9 5 12 10 7 2 2
```

...

- -

3 11 5 10 9 4 24

-

-

3 11 5 9 25

- -

3 11 26

Listato

```

1  /* struttura per la rappresentazione di una colonna di carte */

3  colonna( [Pila1, _, _, _, _, _, _, _, _, _, _, _], 1, Pila1).
4  colonna( [_, Pila2, _, _, _, _, _, _, _, _, _, _], 2, Pila2).
5  colonna( [_, _, Pila3, _, _, _, _, _, _, _, _, _], 3, Pila3).
6  colonna( [_, _, _, Pila4, _, _, _, _, _, _, _, _], 4, Pila4).
7  colonna( [_, _, _, _, Pila5, _, _, _, _, _, _, _], 5, Pila5).
8  colonna( [_, _, _, _, _, Pila6, _, _, _, _, _, _], 6, Pila6).
9  colonna( [_, _, _, _, _, _, Pila7, _, _, _, _, _], 7, Pila7).
10 colonna( [_, _, _, _, _, _, _, Pila8, _, _, _, _], 8, Pila8).
11 colonna( [_, _, _, _, _, _, _, _, Pila9, _, _, _], 9, Pila9).
12 colonna( [_, _, _, _, _, _, _, _, _, Pila10, _, _], 10, Pila10).
13 colonna( [_, _, _, _, _, _, _, _, _, _, Pila11, _], 11, Pila11).
14 colonna( [_, _, _, _, _, _, _, _, _, _, _, Pila12, _], 12, Pila12).

18 decapita([[_|Pila1], P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, Step], 1,
           [Pila1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, Step]).
19 decapita([P1, [_|Pila2], P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, Step], 2,
           [P1, Pila2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, Step]).
20 decapita([P1, P2, [_|Pila3], P4, P5, P6, P7, P8, P9, P10, P11, P12, Step], 3,
           [P1, P2, Pila3, P4, P5, P6, P7, P8, P9, P10, P11, P12, Step]).
21 decapita([P1, P2, P3, [_|Pila4], P5, P6, P7, P8, P9, P10, P11, P12, Step], 4,
           [P1, P2, P3, Pila4, P5, P6, P7, P8, P9, P10, P11, P12, Step]).
22 decapita([P1, P2, P3, P4, [_|Pila5], P6, P7, P8, P9, P10, P11, P12, Step], 5,
           [P1, P2, P3, P4, Pila5, P6, P7, P8, P9, P10, P11, P12, Step]).
23 decapita([P1, P2, P3, P4, P5, [_|Pila6], P7, P8, P9, P10, P11, P12, Step], 6,
           [P1, P2, P3, P4, P5, Pila6, P7, P8, P9, P10, P11, P12, Step]).
24 decapita([P1, P2, P3, P4, P5, P6, [_|Pila7], P8, P9, P10, P11, P12, Step], 7,
           [P1, P2, P3, P4, P5, P6, Pila7, P8, P9, P10, P11, P12, Step]).
25 decapita([P1, P2, P3, P4, P5, P6, P7, [_|Pila8], P9, P10, P11, P12, Step], 8,
           [P1, P2, P3, P4, P5, P6, P7, Pila8, P9, P10, P11, P12, Step]).
26 decapita([P1, P2, P3, P4, P5, P6, P7, P8, [_|Pila9], P10, P11, P12, Step], 9,
           [P1, P2, P3, P4, P5, P6, P7, P8, Pila9, P10, P11, P12, Step]).
27 decapita([P1, P2, P3, P4, P5, P6, P7, P8, P9, [_|Pila10], P11, P12, Step],
           10, [P1, P2, P3, P4, P5, P6, P7, P8, P9, Pila10, P11, P12, Step]).
28 decapita([P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, [_|Pila11], P12, Step],
           11, [P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, Pila11, P12, Step]).
29 decapita([P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, [_|Pila12], Step],
           12, [P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, Pila12, Step]).

31 /* funzioni di supporto */

33 % QuickSort.prolog — quicksort implementation
35 quicksort([], []).

```



```

92  occorrenze_nelle_colonne([], _, [], _).
93  occorrenze_nelle_colonne(CodaCol, X, Prec):-
94    occorrenze_nelle_colonne(CodaCol, X, Prec, 1),!.

96  occorrenze_multiple1([X|Col], X, [Conta|MultiOcc], Conta) :-
97    occorrenze_multiple1(Col, X, MultiOcc, Conta).

99  occorrenze_multiple1([_|Col], X, MultiOcc, Conta) :-
100   occorrenze_multiple1(Col, X, MultiOcc, Conta).

102  occorrenze_multiple1([], _, [], _).

104  occorrenze_multiple0([Col|CodaCol], X, [Conta|Prec0], [MultiOcc|Prec], Conta)
    :-
105    occorrenze_multiple1(Col, X, MultiOcc, Conta),
106    Conta0 is Conta + 1,
107    occorrenze_multiple0(CodaCol, X, Prec0, Prec, Conta0).

109  occorrenze_multiple0([_|CodaCol], X, [Conta|Prec0], Prec, Conta) :-
110    Conta0 is Conta + 1,
111    occorrenze_multiple0(CodaCol, X, [Conta|Prec0], Prec, Conta0).

113  occorrenze_multiple0(_, _, [], [], _).
114  occorrenze_multiple0([], _, _, [], _).

116  occorrenze_multiple(CodaCol, X, Prec):-
117    occorrenze_nelle_colonne(CodaCol, X, Prec0, 1), !,
118    occorrenze_multiple0(CodaCol, X, Prec0, Prec, 1), !.

120  applica_regola(Stato, [X,Y], StatoOut) :-
121    applica_regola(Stato, X, Y, StatoOut).
122  applica_regola(Stato, X, Y, StatoOut) :-
123    decapita(Stato, X, Stato0),
124    decapita(Stato0, Y, StatoOut),!.

126  /* funzioni di ricerca della mossa successiva */

128  coppia_candidata(_,_, [], L_append, L_append).
129  coppia_candidata(Stato, X, [X|L2], L3, L_append) :-
130    coppia_candidata(Stato, X, L2, L3, L_append).
131  coppia_candidata(Stato, X, [Y|L2], [[X,Y]|L3], L_append) :-
132    X<Y,
133    testa(Stato, X, X1), testa(Stato, Y, Y1),
134    X1+Y1 =:= 14, !, coppia_candidata(Stato, X, L2, L3, L_append).
135  coppia_candidata(Stato, X, [_|L2], L3, L_append) :-
136    coppia_candidata(Stato, X,L2,L3, L_append).

138  coppia_candidata20(_,_,[], []).
139  coppia_candidata20(Stato, X, [X|L2], L3) :-
140    coppia_candidata20(Stato, X, L2, L3).
141  coppia_candidata20(Stato, X, [Y|L2], [[X,Y]|L3]) :-
142    testa(Stato, X, X1), testa(Stato, Y, Y1),
143    X1+Y1 =:= 14, X=\=Y, coppia_candidata20(Stato, X, L2, L3).
144  coppia_candidata20(Stato, X, [_|L2], L3) :-
145    coppia_candidata20(Stato, X, L2, L3).

147  coppia_candidata2(Stato, X, L2, L3) :- coppia_candidata20(Stato, X, L2, L3)
    ,!.

149  removable1(Stato, [X|L], L2, Li, Lo) :-
150    coppia_candidata(Stato, X,L2,L3, Li),
151    removable1(Stato, L,L2,L3, Lo).

```

```

152 removabile1(_, [], _, L_append, L_append).
153 candidati_eliminazione(Stato, X) :-
154   removabile1(Stato, [1,2,3,4,5,6,7,8,9,10,11,12], [1,2,3,4,5,6,
      7,8,9,10,11,12], [], X),!.

156 prioritazione(Stato, X, Y, Priorita) :-
157   applica_regola(Stato, X, Y, Stato2),
158   notdeadlock(Stato2, Priorita0),
159   Priorita0:=13,
160   colonna(Stato, X, P1),
161   colonna(Stato, Y, P2),
162   length(P1, X1),
163   length(P2, Y1),
164   /* Priorita is X1+Y1.*/
165   (
166     X1>Y1, !, Priorita is X1;
167     Priorita is Y1
168   ).

170 candidati_con_priorita0(Stato, [[X,Y]|Candidati], [[W,X,Y]|Queue]) :-
171   prioritazione(Stato, X, Y, W),
172   candidati_con_priorita0(Stato, Candidati, Queue).
173 candidati_con_priorita0(Stato, [_|Candidati], Queue) :-
174   candidati_con_priorita0(Stato, Candidati, Queue).
175 candidati_con_priorita0(_, [], []).

176 candidati_con_priorita(Stato, Candidati, Queue) :-
177   candidati_con_priorita0(Stato, Candidati, Queue0),!,
178   quicksort(Queue0, Queue),!.

181 /* algoritmi di valutazione delle mosse */

183 notdeadlock3(_, X, Y, _, _, Exclude) :- member([X,Y], Exclude), !, fail.
184 notdeadlock3(Stato, _, [], Stato, _, _).
185 notdeadlock3(Stato, X, Y, StatoFinal, AlReadyVisited, Exclude) :-
186   testa(Stato, Y, X1),
187   (
188     X:=X1, !, StatoFinal=Stato;

190     coppia_candidata2(Stato, Y, [1,2,3,4,5,6,7,8,9,10,11,12], [[Y,Z]|_]),
191     applica_regola(Stato, Y, Z, Stato0),
192     notdeadlock3(Stato0, X, Y, StatoFinal, AlReadyVisited, Exclude)
193   ).
194 notdeadlock3(Stato, X, Y, StatoFinal, AlReadyVisited, Exclude) :-
195   testa(Stato, Y, Y0),
196   Y0=\=X,
197   Y1 is 14-Y0,
198   occorrenze_nelle_colonne(Stato, Y1, YListPrec0),
199   elimina_occorrenze(Y, YListPrec0, YListPrec),
200   notdeadlock2(Stato, Y1, YListPrec, Stato0, AlReadyVisited, [[X,Y]|Exclude])
201   ,
202   coppia_candidata2(Stato0, Y, [1,2,3,4,5,6,7,8,9,10,11,12], [[Y,Z]|_]),
203   applica_regola(Stato0, Y, Z, Stato1),
204   notdeadlock3(Stato1, X, Y, StatoFinal, AlReadyVisited, Exclude).

206 notdeadlock25(Stato, X, [Y|XListPrec], FinalStato, AlReadyVisited, Exclude)
207   :-
208   (
209     notdeadlock3(Stato, X, Y, FinalStato, AlReadyVisited, Exclude);
210     notdeadlock25(Stato, X, XListPrec, FinalStato, AlReadyVisited, Exclude)

```

```

210     ).
211 notdeadlock25(_, _, [], _, _, _) :- fail.

213 notmember(X, [X|_]):-
214     !, fail.
215 notmember(X, [_|L]) :-
216     notmember(X,L).
217 notmember(_, []).

219 notdeadlock2(Stato, X, XListPrec, FinalStato, AlReadyVisited, Exclude) :-
220     notmember(X, AlReadyVisited),!,
221     notdeadlock25(Stato, X, XListPrec, FinalStato, [X|AlReadyVisited], Exclude)
        ,!.

224 notdeadlock1(Stato, [X|L], Acc, Conta) :-
225     occorrenze_nelle_colonne(Stato,X, XListPrec),
226     length(XListPrec, Len),
227     (
228         Len == 0;
229         notdeadlock2(Stato, X, XListPrec, _, [], [])
230     ),
231     Conta0 is Acc+1,
232     notdeadlock1(Stato, L, Conta0, Conta).

234 notdeadlock1(Stato, [_|L], Acc, Conta) :-
235     notdeadlock1(Stato, L, Acc, Conta).
236 notdeadlock1(_, [], Conta, Conta).

238 notdeadlock(Stato, Conta) :-
239     notdeadlock1(Stato, [1,2,3,4,5,6,7,8,9,10,11,12,13], 0, Conta),!.

241 /* algoritmo di valutazione dello stato */

243 notdeadlock5_fullcheck(_,_,_, [], _, _):-fail.
244 notdeadlock5_fullcheck(Stato, X, Y, [[Y,Z]|_], StatoFinal, Exclude) :-
245     (
246         testa(Stato, Z, Z1), member([Z1,Z], Exclude), !, fail;

248         applica_regola(Stato, Y, Z, Stato0),
249         notdeadlock3_fullcheck(Stato0, X, Y, StatoFinal, Exclude)
250     ).
251 notdeadlock5_fullcheck(Stato, X, Y, [_|L], StatoFinal, Exclude) :-
252     notdeadlock5_fullcheck(Stato, X, Y, L, StatoFinal, Exclude).

254 primadi(X, _, [X|_]).
255 primadi(_, X2, [X2|_]):-!,fail.
256 primadi(X, X2, [_|L]):-primadi(X,X2,L).

258 purifica0(Stato, X, X2, [Z|L], [Z|L2]) :-
259     colonna(Stato, Z, ColZ),
260     primadi(X2, X, ColZ),
261     purifica0(Stato, X, X2, L, L2).
262 purifica0(Stato, X, X2, [_|L], L2) :-
263     purifica0(Stato, X, X2, L, L2).
264 purifica0(_, _, _, [], []).
265 purifica(Stato, X, X2, L, L2):-purifica0(Stato, X, X2, L, L2), !.

267 notdeadlock3_fullcheck(_, X, Y, _, Exclude) :- member([X,Y], Exclude), !,
        fail.
268 notdeadlock3_fullcheck(Stato, _, [], Stato, _).
269 notdeadlock3_fullcheck(Stato, X, Y, StatoFinal, Exclude) :-

```

```

270  testa(Stato, Y, X1),
271  (   X:=X1, !, StatoFinal=Stato;
272      coppia_candidata2(Stato, Y, [1,2,3,4,5,6,7,8,9,10,11,12], L),
273      notdeadlock5_fullcheck(Stato, X, Y, L, StatoFinal, Exclude)
274  ).

276 notdeadlock3_fullcheck(Stato, X, Y, StatoFinal, Exclude) :-
277     testa(Stato, Y, Y0),
278     (
279         Y0:=X, !, StatoFinal=Stato;

281         Y1 is 14-Y0,
282         occorrenze_nelle_colonne(Stato, Y1, YListPrec2),
283         purifica(Stato, Y0, Y1, YListPrec2, YListPrec1),
284         elimina_occorrenze(Y, YListPrec1, YListPrec0),
285         ordina_per_profondita(Stato, Y1, YListPrec0, YListPrec),
286         notdeadlock2b_fullcheck(Stato, Y1, YListPrec, Stato0, [[X,Y]|Exclude]),
287         coppia_candidata2(Stato0, Y, [1,2,3,4,5,6,7,8,9,10,11,12], L),
288         notdeadlock5_fullcheck(Stato0, X, Y, L, StatoFinal, Exclude)
289     ).

291 notdeadlock25b_fullcheck(Stato, X, [Y|XListPrec], FinalStato, Exclude) :-
292     (
293         notdeadlock3_fullcheck(Stato, X, Y, FinalStato, Exclude);
294         notdeadlock25b_fullcheck(Stato, X, XListPrec, FinalStato, Exclude)
295     ).
296 notdeadlock25b_fullcheck(_, _, [], _, _) :- fail.

298 notdeadlock2b_fullcheck(Stato, X, XListPrec, FinalStato, Exclude) :-
299     notdeadlock25b_fullcheck(Stato, X, XListPrec, FinalStato, Exclude).

301 notdeadlock27_fullcheck(_, _, [], _) :- fail.
302 notdeadlock27_fullcheck(_, _, [], []).
303 notdeadlock27_fullcheck(Stato, X, [[Y,Z]|_], Lista) :-
304     applica_regola(Stato, Y, Z, Stato0),
305     notdeadlock26_fullcheck(Stato0, X, Lista).
306 notdeadlock27_fullcheck(Stato, X, [_|Coppia], Lista) :-
307     notdeadlock27_fullcheck(Stato, X, Coppia, Lista).

310 notdeadlock26_fullcheck(_, _, [13]).
311 notdeadlock26_fullcheck(_, _, [13,_|_]) :-
312     writeln('Stato non ammesso'),halt.

314 notdeadlock26_fullcheck(Stato, X, [Y|Lista]) :-
315     notdeadlock3_fullcheck(Stato, X, Y, Stato0, []),
316     coppia_candidata2(Stato0, Y, [1,2,3,4,5,6,7,8,9,10,11,12], L),
317     notdeadlock27_fullcheck(Stato0, X, L, Lista).

319 notdeadlock25_fullcheck(Stato, X, [Y|XListPrec]) :-
320     notdeadlock26_fullcheck(Stato, X, Y),!,
321     notdeadlock25_fullcheck(Stato, X, XListPrec).
322 notdeadlock25_fullcheck(_, _, []).

324 notdeadlock2_fullcheck(_, _, []).
325 notdeadlock2_fullcheck(Stato, X, [L|XListPrec]) :-
326     !,notdeadlock25_fullcheck(Stato, X, [L|XListPrec]).

329 notdeadlock1_fullcheck(Stato, [X|L]) :-
330     occorrenze_multiple(Stato,X, XListPrec),
331     notdeadlock2_fullcheck(Stato, X, XListPrec),

```

```

332 notdeadlock1_fullcheck(Stato, L).
333 notdeadlock1_fullcheck(_, []).

335 notdeadlock_fullcheck(Stato) :-
336 notdeadlock1_fullcheck(Stato, [1,2,3,4,5,6,7,8,9,10,11,12,13]).

338 /* metodi principali */

340 risolvil(Stato, [[_,X,Y]_|_], [[X,Y]|Finale]) :-
341 applica_regola(Stato, [X,Y], Stato00),
342 passo_successivo(Stato00, Stato0),
343 risolvi(Stato0, Finale).

345 risolvil(Stato, [_|Candidates], Finale) :-
346 (
347 notdeadlock_fullcheck(Stato);
348 write('/*deadlock detected _____'), stampa_stato(Stato),
349 writeln('_____*/'),!,fail
350 ),
351 risolvil(Stato, Candidates, Finale).

353 risolvil(_, [], _) :-
354 fail.

356 risolvi([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12], Finale) :-
357 risolvi([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12, []], Finale).

359 risolvi([], [], [], [], [], [], [], [], [], [], [], [], [], []).

361 risolvi(Stato, Finale) :-
362 candidati_elimina(Stato, Candidates),
363 candidati_con_priorita(Stato, Candidates, Queue),
364 risolvil(Stato, Queue, Finale).

366 /* funzioni di output */

368 stampa_col([X|_], 1) :-
369 ( X<10,!,write(' '),write(X);
370 write(X)
371 ),write(' ').
372 stampa_col([], _) :- write(' ').
373 stampa_col([_|L], N) :- N0 is N-1, stampa_col(L, N0).

375 stampa_riga([Col|Stato], N) :- stampa_col(Col, N), stampa_riga(Stato, N).
376 stampa_riga([], _) :-nl.

378 stampa_stato(Stato) :- stampa_stato(Stato, 5), nl,nl, !.
379 stampa_stato(_, 0).
380 stampa_stato(Stato, N) :- stampa_riga(Stato, N), N0 is N-1, stampa_stato(
381 Stato, N0).

382 stampa_scelta(X,Y) :- stampa_scelta(X,Y,1),nl,!.
383 stampa_scelta(_,_,14).
384 stampa_scelta(X,Y,N) :-
385 write(' '),
386 (
387 N:=X,!,write(' ');
388 N:=Y,!,write(' ');
389 write(' ')
390 ),write(' '),
391 N0 is N+1,

```

```

392 stampa_scelta(X,Y,N0).

394 stampa([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12], Mosse) :-
395 stampa([P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,[1]], Mosse).

397 stampa([[[],[],[],[],[],[],[],[],[],[],[],[], _],_):-nl.
398 stampa(Stato, [[X,Y]|Mosse]) :-
399 stampa_scelta(X,Y),
400 stampa_stato(Stato),
401 applica_regola(Stato, X, Y, Stato1),
402 passo_successivo(Stato1, Stato0),
403 stampa(Stato0, Mosse).

405 risolvi_stampa(Stato) :-
406 risolvi(Stato, Mosse),
407 stampa(Stato, Mosse).

409 mischia([], []).
410 mischia0(Lst, E) :-
411     length(Lst, L),
412     random(0, L, I),
413     nth0(I, Lst, E).

415 mischia([], []).
416 mischia(Carte, [Testa|Scozzate]) :-
417     mischia0(Carte, Testa),
418     delete(Carte, Testa, Carte0),
419     mischia(Carte0, Scozzate).

421 distribuisci([[X1,_],[X2,_],[X3,_],[X4,_],[X5,_]|Carte], [[X1,X2,X3,X4,X5]|
Stato]) :-
422     length(Carte, L),
423     L > 31,
424     distribuisci(Carte, Stato).
425 distribuisci(Carte,Stato):- distribuisci0(Carte,Stato).
426 distribuisci0([],[]).
427 distribuisci0([[X1,_],[X2,_],[X3,_],[X4,_]|Carte], [[X1,X2,X3,X4]|Stato]) :-
428     distribuisci0(Carte,Stato).

431 inventa_risolvi_stampa:-
432     mischia([[1,c],[1,q],[1,p],[1,f],[2,c],[2,q],[2,p],[2,f],[3,c],[3,q],[3,p
], [3,f],[4,c],[4,q],[4,p],[4,f],[5,c],[5,q],[5,p],[5,f],[6,c],[6,q],[6,
p],[6,f],[7,c],[7,q],[7,p],[7,f],[8,c],[8,p],[8,q],[8,f],[9,c],[9,p
],[9,q],[9,f],[10,c],[10,p],[10,q],[10,f],[11,c],[11,p],[11,q],[11,f
],[12,c],[12,q],[12,p],[12,f],[13,c],[13,p],[13,q],[13,f]], Carte),
433     distribuisci(Carte, Stato),
434     risolvi_stampa(Stato).

```

Listing 4.1: Listato completo

Elenco delle figure

1.1	Esempio di gioco	2
1.2	Sconfitta per deadlock	3

Listings

2.1	Astrazione dai dati	6
2.2	Ricerca coppie candidate	7
2.3	Deadlock detection	8
2.4	Valutazione mossa	10
2.5	Risoluzione	13
4.1	Listato completo	18