福山大学工学部紀要
第28巻 2004年12月

カルキュレーションを解くプログラム

南崎 好律* 新谷 敏朗**

A Program to Solve Calculation

Yoshinori Nanzaki* and Toshio Shintani**

ABSTRACT

Calculation is a solitaire game with one deck of cards. The game is played with 3 stacks or 4 stacks .It is more difficult with 3 stacks than with 4 stacks. The rate of success is about 60% with 3 stacks when human expert plays the game. In this paper, we use a simple priority queue of hands to play calculation on personal computers. The priorities in the queue are made of our experience. The rate of success on our program is about 56% which is not so much less than that of the human experts.

キーワード:優先順位表,成功率,スタック,テーブル
Key words : Priority table, Rate of success, .Stack ,Table

1. まえがき

カルキュレーションはトランプの一人遊びの一種であり，情報処理学会プログラミングシンポジウムのGPCC[1]でたびたび取り上げられてきた。カードの待避場所である「スタック」の台数を3あるいは4としてプレイすることができる。人間がプレイすると、最初はなかなか成功しないが慣れてくると成功率が上がる。熟練者ではスタック4台の場合で95%、3台の場合では60%程度の成功率であるといわれている。運に左右されることが少なく、戦略の善し悪しによって成功率が大きく変わるゲームである。最近の研究で、田中哲朗氏は十分な計算機資源を用いた大がかりなプログラムによってスタック4台の場合に約99%、3台の場合に約72%という成果をあげている。[2] 筆者らもこれまで、カルキュレーションをパソコンレベルの計算機でプレイさせるプログラムを作成してきた。人間の経験をもとにした「手」の候補を優先度順に表にしたものにしたがってプレイする比較的単純なアルゴリズムを用いているが、スタック4台の場合は90%程度の成功率を達成している。しかし、スタック3台の場合には20%程度の成功率にとどまっていた。本論文では、その優先順位表に改良を加えることにより、スタック3台の場合に人間の熟練者のレベルよりはやや低いが約 56%の成功率を達成することができたことを報告する。

2. ルール

(1) 52 枚のカードをシャッフルして伏せて置く。これを山と呼ぶ。

(2) 山からカードを1枚引く。

(3) 引いたカードの値に従って、テーブルと呼ばれる4つの場所に図 1 のような順序に従って置いていく。（左から右に）

(4) (3)で、山から引いたカードが4つのテーブルのどれについても次に置くべきカードでない場合、一時的な避難場所としてスタックに積むことができる。スタックの個数は3または4のどちらかである。山か

*大学院情報処理工学専攻 **情報処理工学科

ら引いたカードがすぐにテーブルに置くことができる場合でもそのカードをスタックに積んでよい。

テーブル1： A 2 3 4 5 6 7 8 9 10 J Q K
テーブル2： 2 4 6 8 10 Q A 3 5 7 9 J K
テーブル3： 3 6 9 Q 2 5 8 J A 4 7 10 K
テーブル4： 4 8 Q 3 7 J 2 6 10 A 5 9 K

図1　カルキュレーションのテーブル配置

(5) スタックトップのカードが 4 つのテーブルのどれかについて次に置くべきカードならば、そのテーブルに移動してもよい。

(6) (2)～(5)の動作を繰り返して、52 枚のカードをすべてテーブルに置くことができれば成功である。山のカードがなくなった後で、スタックトップの 4 枚のカードのどれもテーブルへ移動させることができなければ失敗である。

このゲームはカードのスートは意味を持たずカードの値(A,2,3,...,9,10,J,Q,K)のみが意味を持つ。なお以後本論文においては便宜上、10 を 0 と表記し，スタックは Z, X, C の 3 台とする。

3. ゲームの性質

このゲームにおいては，スタックにカードを積む場合の戦略が重要である。そのことは次のように考えると明らかである。山から引いたカードをすぐにテーブルに置くことができない場合にスタックに積むことが必要になる。その際同じテーブルのカードを同一のスタックにテーブル上の順序と逆順に積むことができれば最終的にスタックからテーブルにテーブル上の順番に従って移動することができる。しかし例えば，テーブル1のカード 5 をスタック V に積んだ後にテーブル 1 のカード 6 を同じスタック V に積んだとする。するとカード 6 をテーブルに移すためにはカード5がテーブル1に置かれている必要があるが，カード 5 をテーブルに移すためにはカード6を先にスタックから取り出さなくてならない。この状況はオペレーティングシステムのプロセス競合に関する「デッドロック」と似ている。よって以後この状況をデッドロックと呼ぶことにする。デッドロックが生じると当然このゲームは失敗する。上に上げた例は，同じテーブルの2枚のカードが同じスタックでテーブルでの順序と同じ順序で積まれたことによるものであるが，2つのテーブルの 4 枚のカードが 2 つのスタックに積まれた場合などもっと複雑なデッドロックも考えられる。同じ値のカードが 4 枚あるので，デッドロックが生じないように最終的に移動するテーブルの予定を割り当てながら52枚のカードを3台あるいは4台のスタックにすべて積むことができればこのゲームは成功する。そのことは次のように考えると明らかである。なお，人間がプレーする場合はスタック上のカードと4つのテーブルの対応づけは途中で変更する場合も多いが本論文ではスタックに積むときに決めた後は変更しないものとする。

52 枚のカードをスタックに積んだ状態でスタック上のカードをグラフの節点に対応させる。スタック上でカード X がカード Y のすぐ上に積まれているときカード Y に対応する節点からカード X に対応する節点に向かう有向の枝を付ける。またスタック上のカードはどれかのテーブルに対応付けられているのでその対応付けに従って，あるテーブルでカード A がカード B の直前に位置しているとき，カード B に対応する節点からカード A に対応する節点に向かう有向の枝を付ける。スタック上のカードに対応する節点すべてにそのような有向枝を付け加えたグラフを G とする。つまり，G において節点 1 から節点 2 に向かう有向枝がある場合には節点 1 に対応するカードをテーブルに移動するためにはその前に節点2に対応する節点をスタックからテーブルに移動する必要がある。

上記のデッドロックが生じていれば G に有向の閉路が存在して，あるカードをテーブルに移動するためには他のカードを移動する必要があるがそのためには自分自身を先に移動しなければならないという矛盾を生じていることになる。テーブルの左端のカードはルール上直ちにテーブルに置くことができるので対応する G の節点から出ている枝は高々1 本（スタックトップにない場合に 1 本でスタックトップにある場合は 0 本）であることを考慮すると，次の性質が成り立つ。

性質1　52 枚のカードがすべてスタックに積まれた状態でデッドロックが生じていなければ4つのテーブルの左端のカードのうち少なくとも1枚はスタックのトップにある。

証明　4 つのテーブルの左端のカードに対応するカードがどれもスタックトップにないとする。するとすべてのスタックのトップにはテーブルの左端以外のカードが積まれている。当然テーブルにはまだカードは1枚も置かれていないので，どれかのスタックの底の方にあるカードを先に移動しなければスタックトップのカードをテーブルに移動することはできない。これは最も単純なデッドロックの例である。

もし山から引いたカードをすぐテーブルに置く，またはスタックトップのカードをテーブルに移動するなどして，52 枚のカードのすべてがスタック上にあるのではない状態でも，山のカードがなくなった状態で上記のグラフ G を考える。するとその場合の G では 4 つのテーブルで次に置くべきカードに対応する節点が性質 1 のテーブルの左端のカードに対応する。従ってデッドロックが生じていなければ(すなわち G に有向の閉路が存在しなければ)，性質 1 によりそれらのうち少なくとも1枚はスタックのトップにある。よってそのカードはテーブルに移動することができる。そのカードに対応する節点を G から除去したグラフについても同様に考えることができるのは明らかなので，スタックトップのカードのうち1枚をテーブルに移動する操作を繰り返すことによってスタック上のすべてのカードをテーブルに移動することができる。従って次の系1が成り立つ。

系 1　デッドロックが生じていなければ，スタックのトップにあるカードのうち少なくとも1枚はテーブルに移動することができる。

4.　アルゴリズム

前節の性質1と系1から，カルキュレーションをプレイするおおまかな戦略としてとして次のような方針が考えられる。

(おおまかな戦略)

1)　山から引いたカードをテーブルに置くことができる場合は，そのカードをテーブルに置く。

2)　山から引いたカードをテーブルに置くことができない場合は，そのカードを対応させる可能性がある「4 つのテーブルのうちのまだ置かれていないカード」のうち最も後の順序のカードに割り当ててスタックに積む。

3)　1)の操作(又は,3)の操作自身に置いて)によってスタックトップのカードがテーブルに移動できる状態になったらそのカードをテーブルに移動する。

これらのうち操作1)と2)を次のように2文字で表すことにする。操作 3)は可能なときに必ず実行するので特に表記する必要がない。まず，x = 1, 2, 3, 4 によって 4 台のテーブルを表す。S = C, X, V によって 3 台のスタックを表す。そして，テーブルに置くことは T によって表す。すると，

(操作の表記法)

Tx : そのカードをテーブル x に置く。

Sx : そのカードをテーブル x に移動する予定でスタック S に積む。

これらの操作に対して，人間がプレイした経験によって優先順位をつけたカードの値別のキューをあらかじめ作成しておく。例えば，A に対して

(カード別の優先順位リスト)

A : T1 T2 T3 T4 Z4 Z3 Z2 X4 X3 X2 C4 C3 C2

とする。これによると，山から引いた A のうち最初のものはテーブル1に置く。2 枚目の A は「テーブル 2 に置く」。置けなければ，「テーブル 3 に置く」。テーブル 3 にも置けなければ，「テーブル 4 に置く」。テーブル 4 にも置けなければ，「後でテーブル 4 に移動する予定でス

タックXに積む」。以下同様で最後が，「後でテーブル2に移動する予定でスタックVに積む」，となる。スタックに積む際はデッドロックが生じるかどうかのチェックを行う必要がある。そのためには，前節で述べたようなグラフを作成して有向の閉路が生じるかどうかを調べればよい。本論文では，次のような簡略化された方法によっている。

y = A, 2, ..., 0, J, Q, Kによってカードの値を表す。カードyをテーブルxに移動する予定でスタックSに積むとする。

（デッドロックのチェック方法）

① テーブルxでyよりも順序が前のカードがスタックSに既に積まれていれば，デッドロックが生じる。

② テーブルxでyよりも順序が前のカードwがS以外のスタックに積まれているとする。そのスタックでwの上に積まれているカードをuとする。uよりテーブル上の順序が前のカードrがスタックSに既に積まれていればデッドロックが生じる。

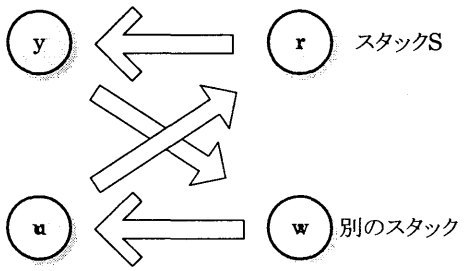


図2　デッドロック

スタックに積む操作によってデッドロックが生じるときはその操作は行わずにキューから次の優先順位の操作を取り出す。デッドロックが生じる操作が続いてキューが空になれば「失敗」と判定する。

デッドロックが生じないで山のカードがなくなったときに，スタックが空であればもちろん「成功」である。スタックにカードが残っていてもスタックトップのカードをテーブルに移動していけば系1によりすべてのカードをテーブルに移動できるので「成功」である。

おおまかなアルゴリズムは以上であるが，本論文ではスタックに積む操作のうち同じテーブルで隣接するカードを同じスタックに（テーブル上の順序とは逆順に）続けて積む操作を「並べ積み」と呼んで，Nで表す。並べ積みをすればデッドロックが生じる可能性が低くなることは容易に想像できるし，経験的にもそう考えられる。よって優先順位表には「並べ積み」の操作も含まれている。今回採用した優先順位表を以下に示す。[3]

表1　カードごとの操作に対する優先順位表

A: T1 T2 N4 N3 N2 T3 T4 Z4 Z3 Z2 X4 X3 X2 C4 C3 C2

2: T1 T2 T3 T4 N4 N3 N1 Z4 Z3 Z1 X4 X3 X1 C4 C3 C1

3: T3 T1 T4 T2 N2 N4 N1 Z2 Z4 Z1 X2 X4 X1 C2 C4 C1

4: T4 T1 T2 T3 N3 N1 N2 Z3 Z1 Z2 X3 X1 X2 C3 C1 C2

5: T1 T3 T2 T4 N4 N2 N3 N1 X4 X2 Z3 Z1 C4 C2 X3 X1 Z4 Z2 C3 C1

6: T3 T2 T1 N4 N1 N2 N3 T4 Z4 Z1 Z2 Z3 X4 X1 X2 X3 C4 C1 C2 C3

7: T4 T1 T2 T3 N3 N2 N1 N4 X3 X2 Z1 Z4 C3 C2 X1 X4 Z3 Z2 C1 C4

8: T4 T2 N1 N3 N4 N2 T3 T1 Z1 Z3 Z2 Z4 X1 X3 X2 X4 C1 C3 C2 C4

9: N4 N2 X4 T3 N3 T1 N1 T2 T4 Z2 Z1 Z3 Z4 X2 X1 X3 C4 C2 C1 C3

0: N3 N1 X3 T2 N2 T4 N4 T1 T3 Z1 Z4 Z2 Z3 X1 X4 X2 C3 C1 C2 C4

J: N2 N1 X2 T4 N4 T3 N3 T1 T2 Z1 Z3 Z4 Z2 X1 X3 X4 C2 C1 C3 C4

Q: N1 T3 N2 T4 N4 N3 X1 T2 T1 Z2 Z3 Z4 Z1 X2 X3 X4 C1 C2 C3 C4

K: C1 C2 C3 C4 T1 T2 T3 T4 X1 X2 X3 X4 Z1 Z2 Z3 Z4

原則として，テーブル上の順序の分布が前に偏っているカードはテーブルに置く操作の優先順位を高くしており，そうでないカードは並べ積みとスタックに積む操作の優先順位を高くしている。そして，テーブルに置く操作ではテーブル上の順序に従って(例えばAではT1, T2,T3,T4の順）優先順位を決めている。スタックに積む操作ではテーブルに置く操作と逆順（例えばAではZ4,Z3,Z2の順)に優先順位を設定している。Kはどのテーブルでも最後に置くので，スタックCを優先的にKに割り当てる。また他の2つのスタックのうちXをよりテーブル上の順序が前のカードを積むために主に使用するように優先順位を決める。これにより図2のようなデッドロックができるだけ生じないようにしている。これらの原則の他に筆者の経験によって詳細を決めている。例えば2のカードについて，テーブル2にはすぐ置けるが，テーブル1に置く操作の方に1番の優先順位を割り当てている。

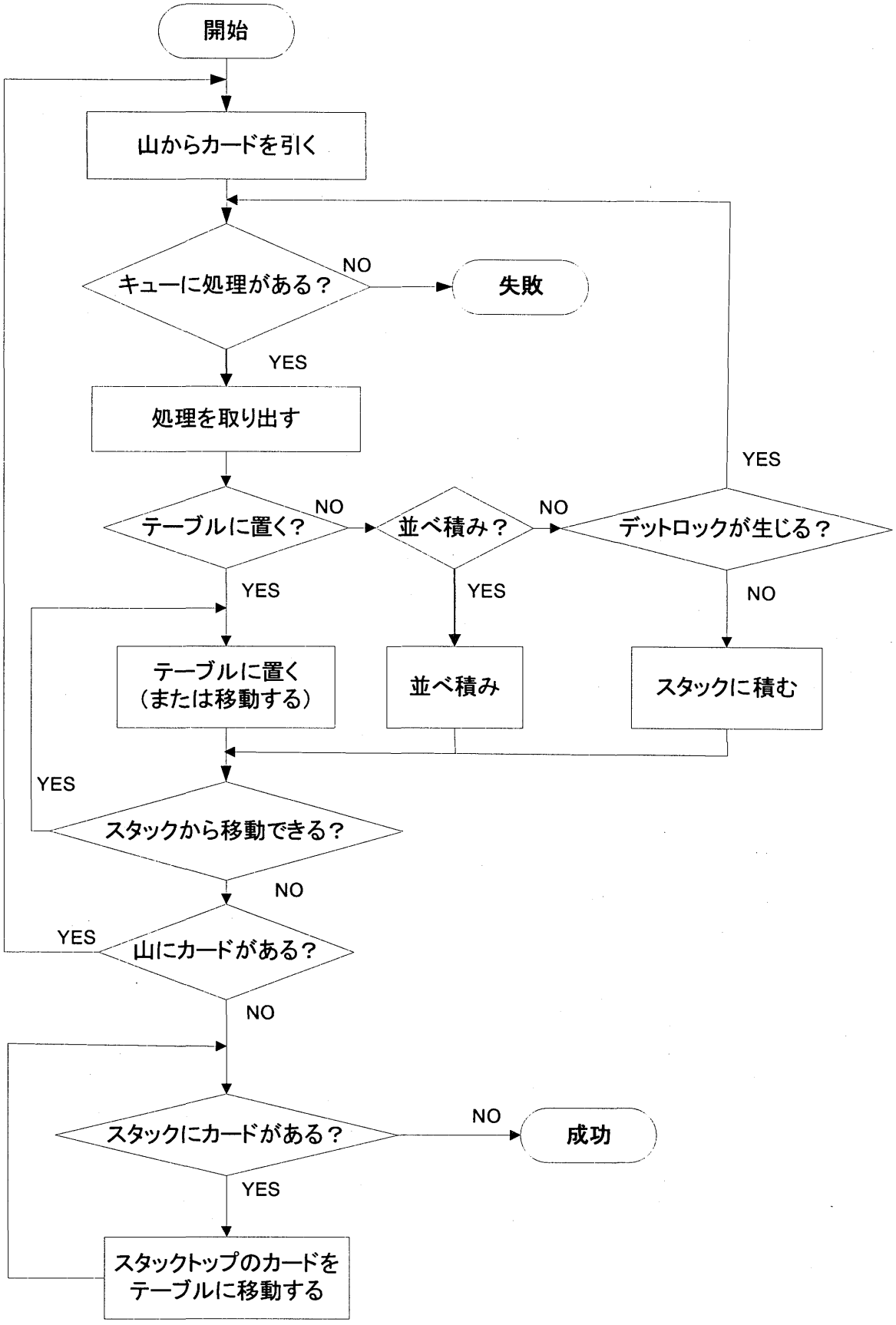
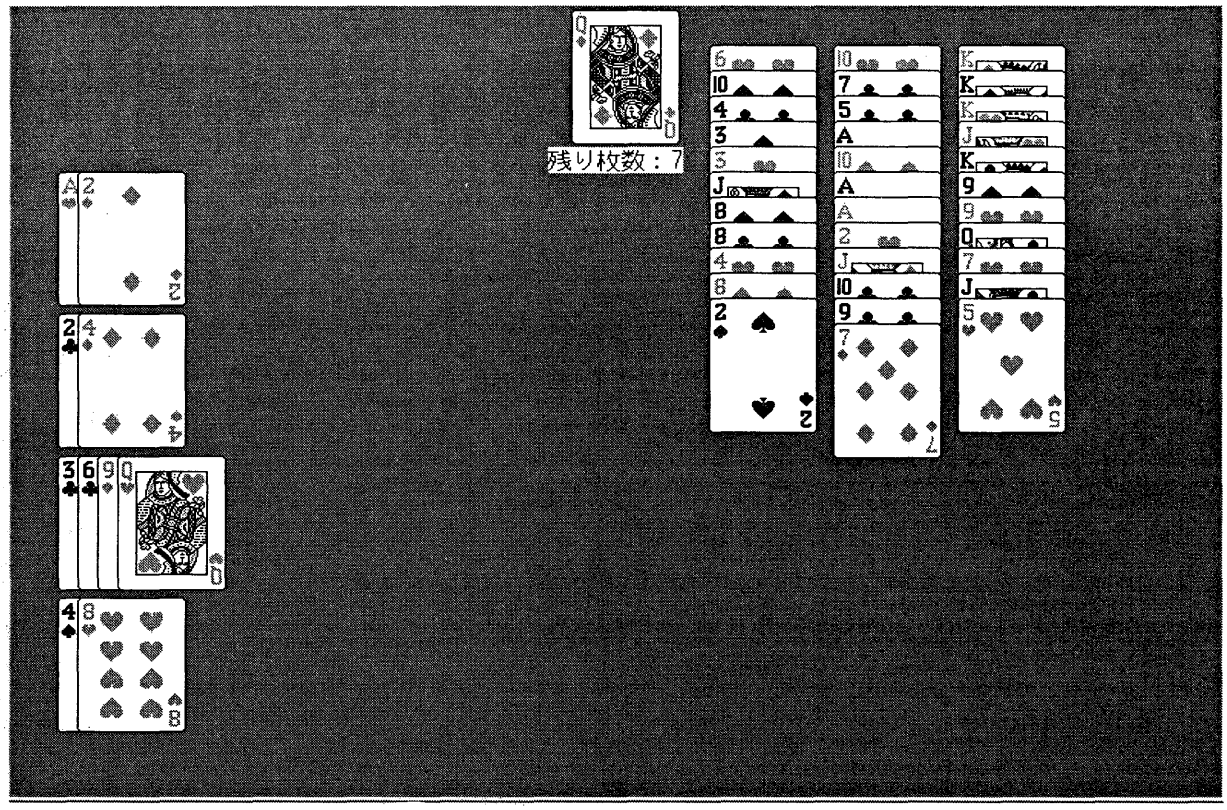


図3　カルキュレーションをプレイするアルゴリム

以上のアルゴリズムをフローチャートで示したものが図3である。フローチャートが煩雑になるので省略してあるが，実際のプログラムではキューが空になって可能な操作がなくなり「失敗」と判定された後もスタックCに積んで山のカードを最後まで引いてその後スタックトップのカードをテーブルにできるだけ移動するようにしている。

5. 実行結果

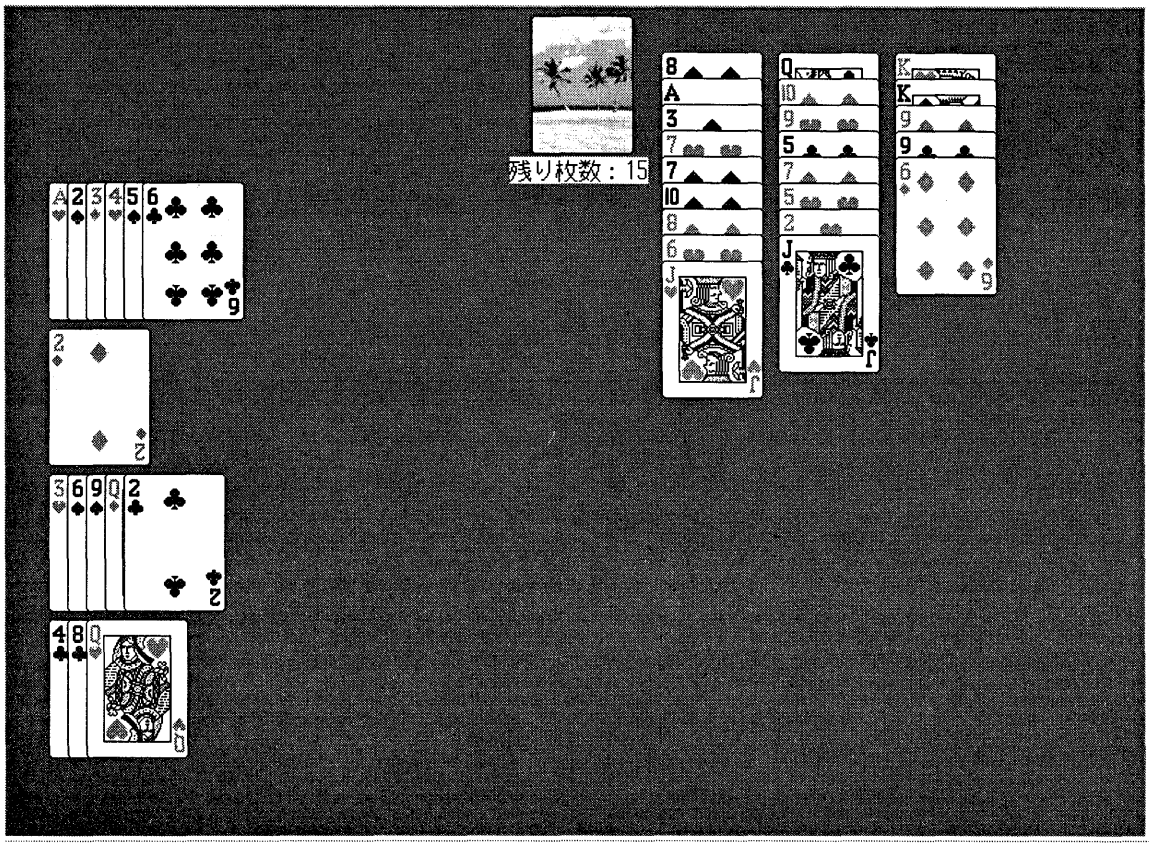
異なる系列からなる3種類の疑似乱数を用いて作成した30000個の山データに対して前節のアルゴリズムを用いてカルキュレーションをプレイさせた。その結果，16720 個(55.73%)が成功した。これは人間の熟練者の成功率にはやや及ばないがほぼ匹敵する成功率である。成功した場合の操作と途中の状態の例を図4に，失敗した場合の操作と途中の状態の例を図5にそれぞれ示す。図で画面左側の部分がテーブル（上から1,2,3,4で左から右に置いていく）を右側の部分がスタック（左からZ, X, Cで上から下に積んでいく）を表す。操作手順は優先順位表の操作手順と似ているがスタック上のカードとテーブルの対応を把握しやすいように，次のような表記法にした。



操作手順

6Z4 0X3 33 KC1 0Z1 44 4Z3 3Z3 7X3 KC2 KC3 3Z4 5X4 JC2 A1 KC4 9C4 9C2 JZ3 QC1
84 AX4 7C2 8Z3 8Z1 4Z1 JC1 8Z2 0X4 AX3 AX2 21 22 2Z3 42 2X4 JX4 9Z3 0X2 63
Z93 9X1 5C2 7X1 Q3 Z23 Q4 QC2 62 Z82 31 Z41 51 7C4 53 61 X71 Z11 Z83 Z33
Z34 X91 X02 C74 XJ4 X24 CQ2 XA2 Z32 XA3 Z43 Z01 Z64 X04 XA4 X54 X73 X03 C52 CJ1
C72 CQ1 C92 C94 CK4 CJ2 CK3 CK2 CK1

図4 成功した場合の例（No.11）



操作手順

QX1 A1 8Z1 44 21 33 AZ4 KC1 31 41 0X3 9X4 51 5X4 3Z2 7X3 5X2 KC2 7Z1 22
7Z4 9Z3 84 2X4 63 Z93 9C2 Q3 23 JX4 Q4 0Z2 9C1 61 8Z2 6Z2 6C4 JZ3 **0C1** 8Z3
KC3 53 Z83 ZJ3 0C4 42 Z62 Z82 Z02 34 Z74 Z71 XJ4 X24 4X3 JC2 AX3 XA3 X43 Q2
KC4 A2 Z32 X52 X73 JC1 72

図5 失敗した場合の例（No.31）

yx: カードyをテーブルxに置く。

ySx: カードyを後でテーブルxに移動する予定でスタックSに積む。

Syx: スタックSのトップにあるカードyをテーブルxに移動する。

図4では操作手順の45手目のQ3（下線で表示）までの状態である。その後56手目の61で山のカードを引き終え,その後スタック上のカードをすべてテーブルに移動して成功している。図5は，38手目のJZ3までの状態である。この状態で次のカード10は既にスタック上に2枚積まれておりそれらはテーブル2と3に割り当てられいる。(11手目の0X3と22手目の0Z2) したがってカード10を割り当てられるのはテーブル1と4のどちらかであるが，テーブル1では8Z1，9C1，テーブル4では7Z4, JX4, 6C4の手で，テーブル2と4で10より順序が前のカードが既にスタックに積まれている。これらは前節で挙げたデッドロックのチェック方法のうち①の単純な場合である。残された手であるスタックXにテーブル1に移動する予定で積む操作0X1を行ったとする。するとカード10をテーブルに移動するためにはその前にスタックZの8を（テーブル1で10より順序が前だから）テーブル1に移動しておく必要がある。そのためには，スタックZで8の上に積まれているAをテーブル4

に移動しなければならない。しかし，そのためにはスタックXのJをテーブル4に移動しておく必要があり，そのためにはたった今その上に積んだ10をテーブル1に移動しなければならない。これは前節の図2の例である。よって可能なすべての操作のどれを行ってもデッドロックを避けることができない。

プログラムではその後もそのカードをスタックCに強制的に積むことにより山のカードを引き終わるまで実行を継続して65手目の72まで至っている。そのようにして図5以降山のカードを引き終わるまで操作を続けても最終的にスタックトップのどのカードもテーブルに移動できない状態になるのでこの場合は失敗である。

6. あとがき

トランプの一人遊びであるカルキュレーションをプレイするプログラムを作成した。使用したアルゴリズムは優先順位がつけられた操作の手順を次々に試していくという単純なものである。優先順位は筆者らの経験に基づいて作成されている。その結果，約56%の成功率が得られた。これは人間の熟練者よりはやや低いがほぼ同程度の成功率である。また，計算機による結果の中でも田中氏の結果に比べると15%以上低いが，用いたアルゴリズムの単純さや身近なパーソナルコンピュータで実行可能であることを考えれば十分な成果と考えられる。本論文で用いたアルゴリズムではスタックに積んだカードとテーブルとの対応関係は1対1で固定である。それを変更可能にするあるいは複数の対応関係を保持できるようにすれば，デッドロックが生じる前に対応関係を変更する，または別の対応関係に切り替えるなどの対策を取ることによりデッドロックが避けられる可能性がある。また，山に残っているカードの枚数がある程度少ない状態までデッドロックが生じないようにプレイできれば，残りの山のカードの出方をすべて考慮した全探索によりその状態が成功可能かどうかを調べることもできると考えられる。今後そのような改良を施して，成功率の上限がどの程度かを検討する予定である。

参考文献

[1] 小谷善行：「GPCC　ウルトラナノピコ問題」，Bit 共立出版社株式会社 Vol.20、No.4、pp99-100 (1988年)

[2] 田中哲朗：「部分ゲームの解析を用いたカルキュレーションの戦略」，情報処理学会論文誌 Vol.43，No.10，pp3064-3073(2002年)

[3] 南崎好律:「カルキュレーションの成功率に関する考察」，福山大学卒業論文，(2003年)