

学習によるスパイダソリティアの 自動クリアルーチンの試み

東京工業大学 情報科学科

佐々 政孝 研究室所属

03_07939

菊池 巧

平成 22 年度卒業論文

目次

第1章 はじめに

1.1 背景	4
1.2 研究動機	5
1.3 研究概要	5

第2章 スパイダソリティアとは

2.1 開始状態	7
2.2 ゲームの進め方	8
2.2.1 カード移動	8
2.2.2 空き列の処理	11
2.2.3 カードの配布	12
2.2.4 カードの除去	13
2.3 スパイダソリティアの難易度	14
2.3.1 初級	14
2.3.2 中級	14
2.3.3 上級	14

第3章 学習手法とクリアルーチンの作成方法

3.1 教師付き学習	15
3.2 重みの調整	15
3.3 評価関数	16
3.4 クリアルーチンの作成方法	16

第4章 クリアアルゴリズムの作成

4.1 クリアアルゴリズムの作成手法	18
4.2 基本的なクリアアルゴリズムの作成	19
4.2.1 カードの移動	19
4.2.2 空き列の処理	19
4.2.3 基本的なアルゴリズムの実装と評価	19

4.3 クリアアルゴリズムの改良	20
4.3.1 改良アルゴリズム 1 の実装方法	21
4.3.2 改良アルゴリズム 2 の実装方法	21
4.3.3 改良アルゴリズムの性能評価と考察	21
4.4 高レベル・クリアアルゴリズムの作成	23
4.4.1 前処理アルゴリズム	24
4.4.2 前処理アルゴリズムの利点	24
4.4.3 空き列の処理	25
4.4.4 高レベル・クリアアルゴリズムの実装	26
4.4.5 高レベル・クリアアルゴリズムの評価と考察	27

第5章 学習によるクリアルーチンの作成

5.1 ゲームの進み具合の評価	28
5.2 各場における列の優先順位の判定	30
5.3 学習	32

第6章 評価と考察

6.1 クリアルーチンの性能評価	35
6.2 更なる改良をするために	36
6.3 最後に	36

第7章 関連研究

第8章 まとめ

第9章 参考文献

目次

1. 開始状態	9
2. 1枚のカード移動	10
3. 複数枚のカード移動	10
4. 空き列の処理	11
5. カード配布	12
6. カード除去	13

表目次

1. 基本アルゴリズムと改良アルゴリズムの性能比較	21
2. 高レベル・クリアアルゴリズムと他のアルゴリズムの性能比較	27
3. 各列の評価要素	31
4. 学習によるクリアルーチンと各アルゴリズムとの性能比較	35

1. はじめに

1.1 背景

将棋や囲碁に代表される対人ゲームやトランプを用いたカードゲームの思考ルーチン¹の研究が、昨今において進められている。

オセロやチェス・将棋といった完全情報ゲーム²ではコンピュータゲームプレイヤーが人間のトップクラスのプレイヤーと同等以上の実力を持つなど目覚ましい進歩を遂げている。

その一方で不完全情報ゲーム³では、ポーカーやブリッジ・麻雀などにおいてコンピュータゲームプレイヤーについての研究が行われているが、完全情報ゲームと比べるとあまり良い結果は出ていない。

不完全情報ゲームでは将来の局面を展開していくといったゲーム木を用いた探索手法を利用することが困難なことがその理由の1つである。

¹ コンピュータゲームのプログラムの一部。将棋や麻雀などのボードゲームの対戦相手など、プレイヤーと対戦する側の行動を決定するプログラム。

一般的には、一手ずつ考えて行動するターン制のゲームに対して主に用いる用語。

ボードゲームの領域では特に人工知能研究の一環として思考ルーチンが研究されている。

² 各プレイヤーが自分の手番において、これまでの各プレイヤーの行った選択(あるいは意思決定)について知ることができるゲーム。

将棋やチェス、囲碁など多くのボードゲームでは、各プレイヤーが他のプレイヤーの状況を常に把握でき、また、どのような手を差したのかも明確にわかるため完全情報ゲームといえる。

³ 完全情報ゲームに分類されないゲーム。

麻雀やポーカーなどのカードゲームの多くでは、各プレイヤーの手牌や手札を他のプレイヤーは見る事ができず、どのような状況でその牌やカードを切ったのかを知ることができないため不完全情報ゲームとなる。

1.2 研究動機

不完全情報ゲームは未知の情報が存在する環境に置いて、既知の情報を用いることで人工知能に最善の行動をさせることに対する有用なモデルである。

「限られた情報から全体を予測した上で、最善の手を判断し実行する」という不完全情報ゲームにおける思考プロセスは「全ての情報から最善の手を判断し実行する」という完全情報ゲームの思考プロセス⁴に比べて人間の思考により近いものであるため、作成するのも一般的には困難であると言われている。このことが完全情報ゲームに比べて不完全情報ゲームの研究が進んでいない大きな要因となっている。

本研究では、未知の情報が存在する環境下に置ける人工知能の学習モデルとして、教師あり学習を用いることによる不完全情報ゲームにおけるコンピュータプレイヤーのパラメータの調整を行うことで、不完全情報ゲームの研究とする。

スパイダソリティアのクリアルーチンを作成することで、不完全情報ゲームの研究の進歩の一助になればと思い、本研究を行うことにした。

⁴ 完全情報ゲームは探索を用いることで実装することができる。ゲームの規模によって探索範囲が膨大になる場合もあるため、効率化が課題となっている。具体的な手法としては、ミニマックス木、ミニマックス探索、 $\alpha\beta$ 探索、実現確率探索、ハッシュ法を利用した探索などである。

1.3 研究概要

本研究では、不完全情報ゲームの1つである、トランプを用いたカードゲームの1つであるスパイダソリティアの思考ルーチンの作成を試みることを目的とする。

現在に至るまで、スパイダソリティアの研究はほとんどされてこなかったように思える。少なくとも探した限りでは、スパイダソリティアを自動でクリアするアルゴリズムや思考ルーチンの研究は見つけることができなかった。

そこで本研究では、教師あり学習によってスパイダソリティアのクリアルーチンを作成し、その性能を評価することを目的とする。

本研究では、まずスパイダソリティアを高確率でクリアすることができるアルゴリズムを作成する。その後、作成したアルゴリズムを教師として使い、スパイダソリティアのクリアルーチンを作成する。

クリアルーチンを作るにあたり、評価要素とそれに対応する重みの積からなる評価関数を使用する。

評価要素とは、「注目している列における表向きのカードが n 枚ある」や「裏向きのカードが m 枚ある」など、注目している列の特徴を表す要素のことである。本研究では、場の評価要素 (s で表す) と各列の評価要素 (t で表す) の2種類を扱う。

場の各評価要素とそれに対応する重み (w で表す) の積を全て足したものを、場の評価関数とし、 S で表す。具体的に記述すると、場の各評価要素を s_1, s_2, s_3 とし、それに対応する重みを w_1, w_2, w_3 とすると、場の評価関数 S は

$$S = s_1 \times w_1 + s_2 \times w_2 + s_3 \times w_3$$

という形で計算することができる。

同様に、各列の評価要素とそれに対応する重み (v で表す) の積を全て足したものを、各列の評価関数とし、 $T(i)$ で表す。 (i は列を表す番号。 $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ 。) 具体的に記述すると、第 i 列の場の評価要素を $t_1(i), t_2(i), t_3(i), \dots$ とし、それに対応する重みを v_1, v_2, v_3, \dots とすると、第 i 列の評価関数 $T(i)$ は、

$$T(i) = t_1(i) \times v_1 + t_2(i) \times v_2 + t_3(i) \times v_3 + \dots$$

という形で計算することができる。

2. スパイダソリティアとは

有名なトランプゲームの1つに、ソリティア⁵というものがある。

スパイダソリティアはソリティアから派生したゲームの1つで、2組のトランプ計104枚（ジョーカーを除く）を用いて行うものである。

カードの移動を繰り返すことで 同じ柄のカードを順番に重ねていき、1～13まで揃えたカードを場から取り除く。

計8組のカードの束を場から取り除くとクリアとなる。

2.1 開始状態

ゲームが開始した時、場の状態は以下のようになっている。

- ・ 10列にトランプが並べられている。
- ・ 各列に、裏向きのカードが複数枚（裏返しのカードが5枚なのが4列、4枚なのが6列）その上に、表向きのカードが1枚ずつ置かれている。
- ・ よって、開始時に場に置かれているカードの数は全部で $5 \times 4 + 4 \times 6 + 10 = 54$ 枚である。
- ・ 残り50枚のカードは、場の隅に置かれる。これはカードを配る時に使う。

図1 (p9) は、スパイダソリティアの開始状態である。向かって上部に10列のトランプカードの列があり、左から4列は裏向きのカードが5枚、右6列は裏向きのカードが4枚それぞれ重なって置かれており、その上に、表向きのカードが1枚置かれている。

向かって下部の一番左には、残り50枚のカード（すなわち、配るために使われるカード）が置かれている。その右にある8つの空白マスは、1～13のカードの束ができたときに置く場所である。

⁵ 元々はトランプの一人用ゲームを総称して「ソリティア」と呼ぶ。一般に「ソリティア」という名前で知られているカードゲームは正確には「クロンダイク」という名称であり、スパイダソリティアは正確にはクロンダイクから派生したゲームである。

2.2 ゲームの進め方

スパイダソリティアをプレイするにあたり、プレイヤーが行うことは大きく分けて以下の3つのことである。

- | |
|--|
| <ul style="list-style-type: none">① カードを移動させる。② 空き列の処理③ カードを配る |
|--|

このような操作を繰り返していき、最終的に場からカードが全て無くなったらクリアとなる。

2.2.1 カード移動

場のカードを他の列に移動させる。この操作が、スパイダソリティアをプレイする上でメインとなる。

3のカードの場合は4の上に、6のカードの場合は7の上に、など、降順に順番に並ぶような形になるとき、移動することができる。このようにしてできた同じマーク（ダイヤならダイヤのみ、ハートならハートのみ、など）の複数のカードが降順に重ねられているまとまりを、以後「カード束」と呼ぶ。

同じマークのカードは束として扱うことができるが、異なるマークのカードは束として扱うことができない。ただし、異なるマークのカードの上に移動させることは可能である。

移動することができるのは、列の一番上にあるカード、もしくは同じマークからなるカード束のみである。同じマークからなるカード束の場合、束として移動することができる他に、上から1枚だけ、もしくは上から数枚だけ、のような形で移動させることもできる。

（例として、クローバーが3，4，5と重なっているカード束が列の一番上にあった場合、①3，4，5のまとまりとして6の上に移動させる。②3，4のまとまりとして、他の列の5の上に移動させる。③3のみを、他の列の4の上に移動させる。のいずれかのカード移動を行うことができる。）

図2 (p10)は、カードを1枚移動する様子である。まず、上図の左から5列目にある4のカード、及び左から6列目にある5のカードに着目して欲しい。ここで、4は5の上に移動できるので移動する。

さらに、この操作によって4を移動した列の一番上に裏向きのカードが出現する。列の一番上にきた裏向きのカードは即座に表向きにする必要がある。この例では、裏向きのカードを表向きにしたことで8のカードが出現した。そこで、上図の左から3番目にある7のカードを8のカードの上に移動している。

図3 (p10)は、複数のカードを一度に移動する様子である。

まず、上図の一番左の列に注目していただきたい。5のカードの下に6のカードがある

ので、この2枚は合わせて1つの束とみなすことができる。よって、左から2列目の7のカードの上にまとめて移動することができる。まとめて移動したものが下図である。



図 1. 開始状態

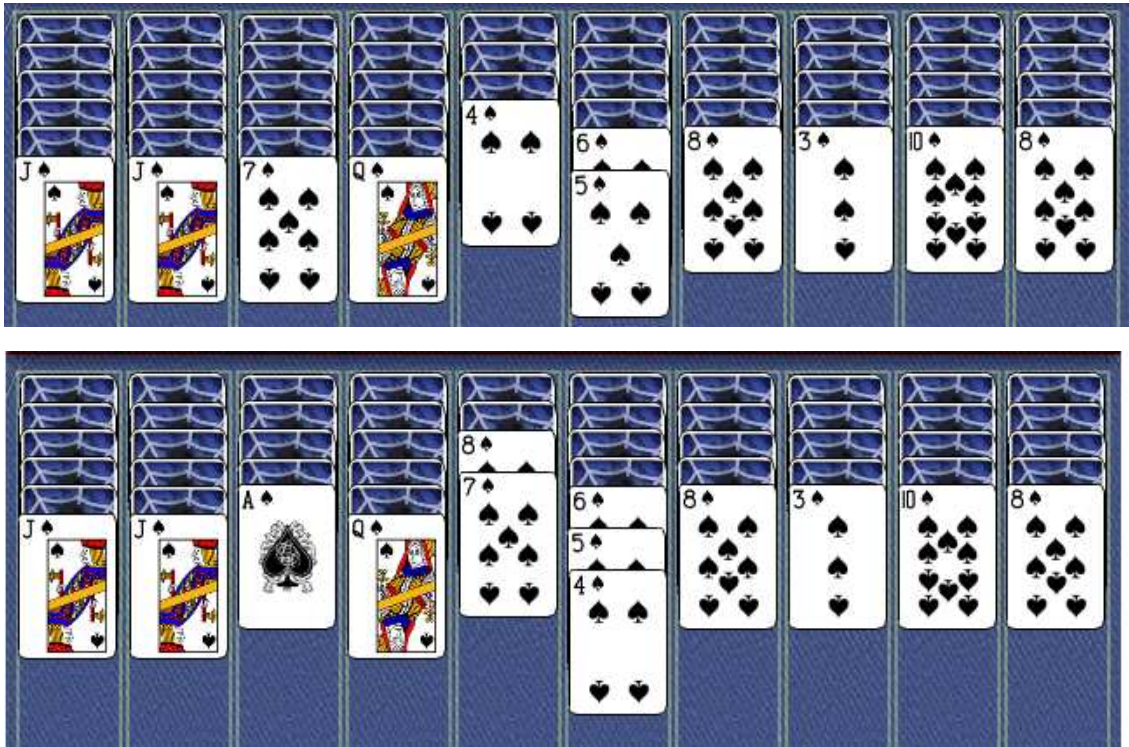


図 2. 1 枚のカード移動

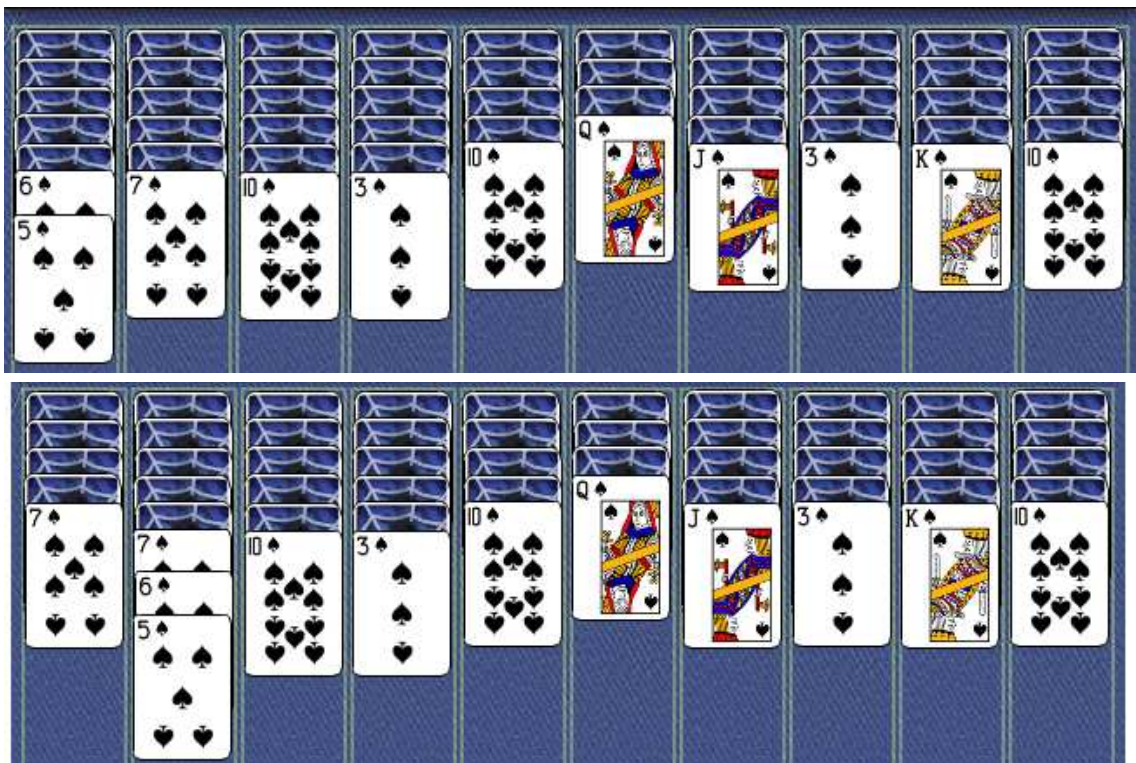


図 3. 複数枚のカード移動

2.2.2 空き列の処理

カードの移動を繰り返すと、1枚もカードが残されていないという状況の列ができる。これを以後「空き列」と呼ぶ。

空き列には、任意のカードを移動させることができる。

空き列がある状態だとカードを配ることができない。

図4 (p11)は、空き列の処理についての様子である。

カードの移動を繰り返したことによって、上図の右から3番目の列が空き列となっている。この場合、任意のカード束を移動させることができる。束の一番下の数字が13の場合、その束を移動できる先は空き列だけである。図4の場合は、上図の右から5番目にあるカード束（一番下の数字が13になっている）を空き列に移動させている。

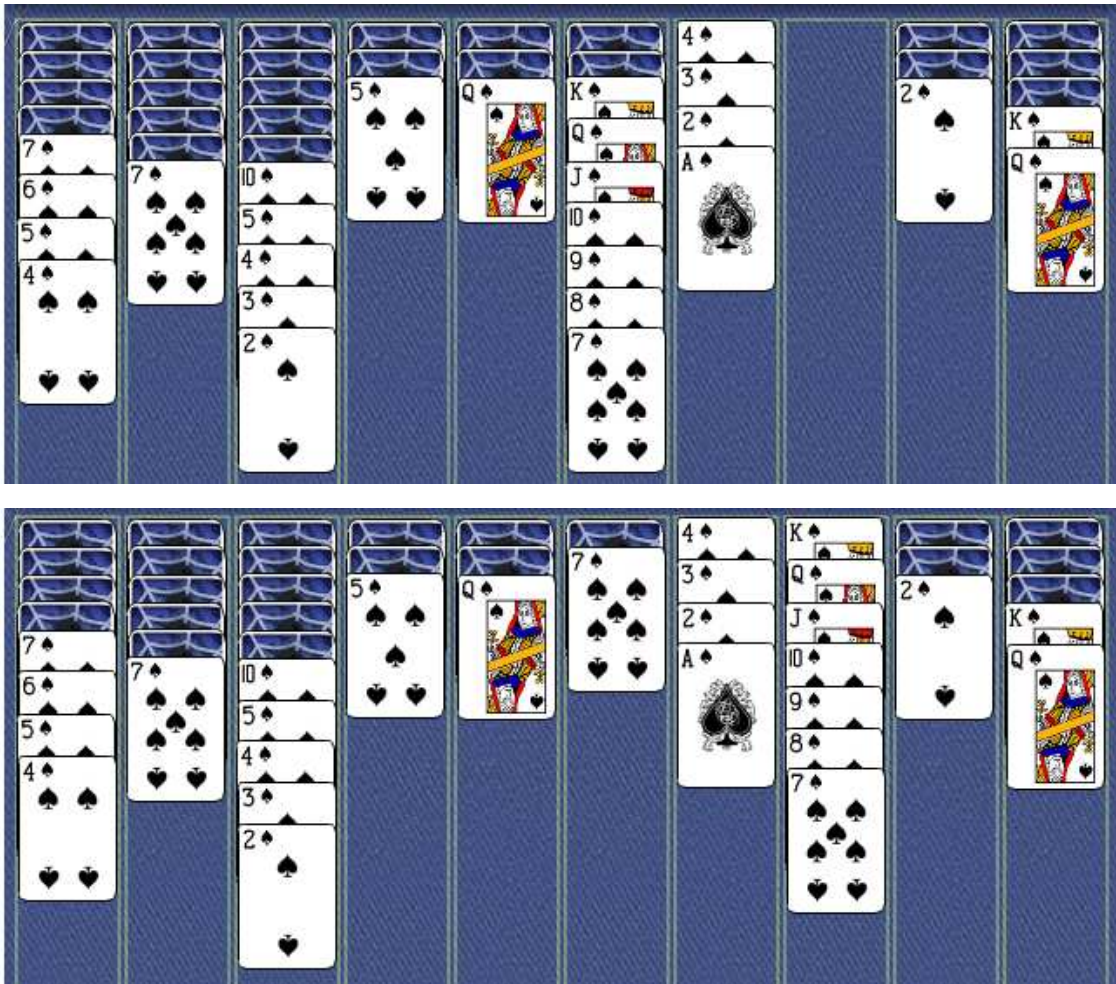


図4. 空き列の処理

2.2.3 カードの配布

開始状態の時に隅に残しておいたカードを、任意のタイミングで配ることができる。

一度のカード配りで配られるカードは各列に1枚ずつ、計10枚。開始状態において隅に残しておいたカードは全部で50枚だったので、計5回、カードを配ることができる。

カードを配るためには、場に空き列が1つもない状態でなければならない。このような状態のとき、各列の一番上に、表向きで1枚ずつ配る。

図5 (p12) は、カードを配る様子である。上図の状態でカードを配ると、各列の一番上にそれぞれ1枚ずつ表向きにカードが配られる。その結果が下図である。

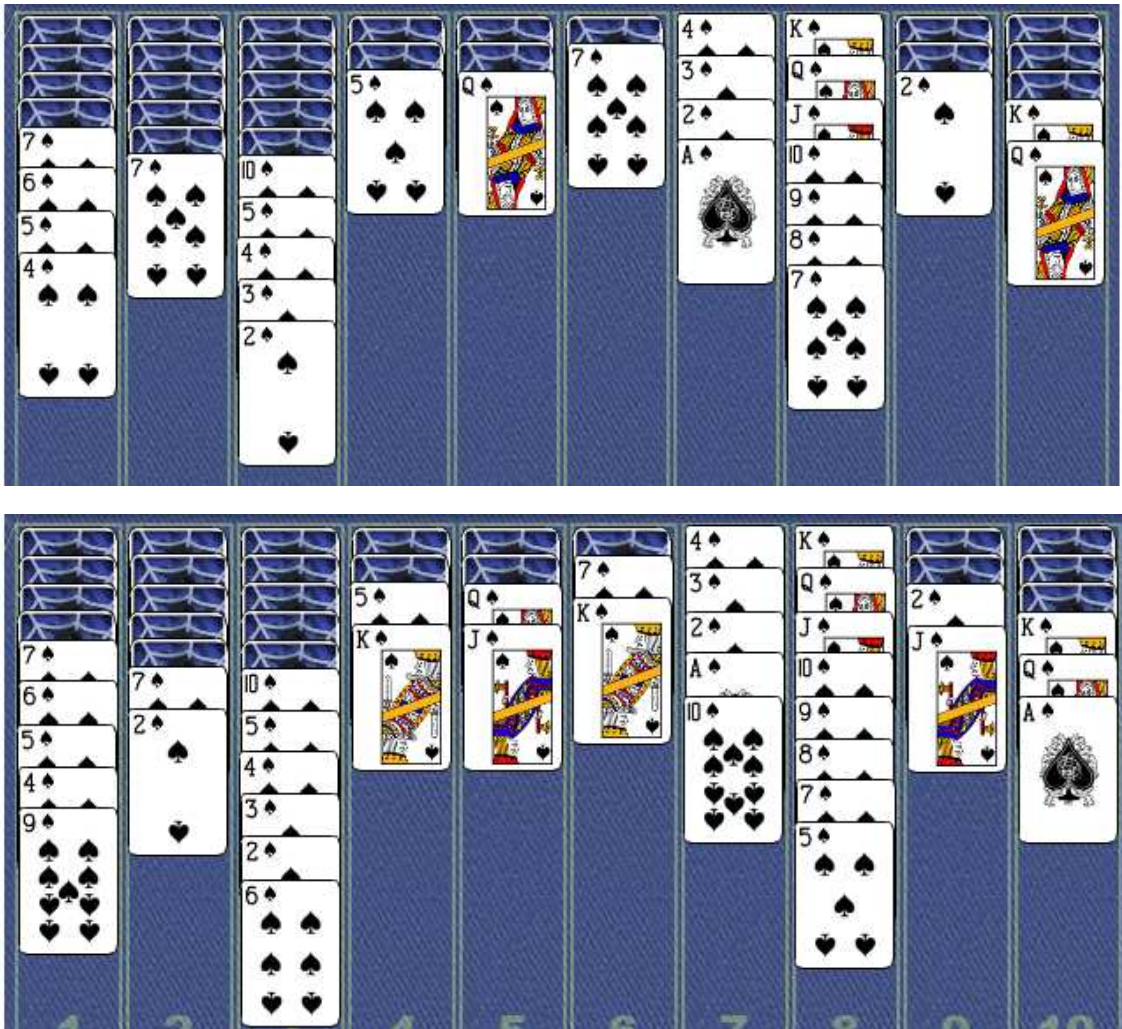


図5. カード配布

2.2.4 カードの除去

カード移動によって、同じマークで1から13までの束を作った場合、即座に場から取り除く。(これは、プレイヤーの判断で場に残しておくことはできない。)

取り除いたカードはその後場に戻すことはできないので、1～13の束を完成させるタイミングは重要である。

図6 (p13)は、カード除去についてである。

上図の右から2番目の列に5から13が、右から5番目の列に1から4がそれぞれ束としてある。よって、右から5番目の1から4の束を右から2番目の列に移動させることで1から13の束ができるので、この束を場から取り除く。

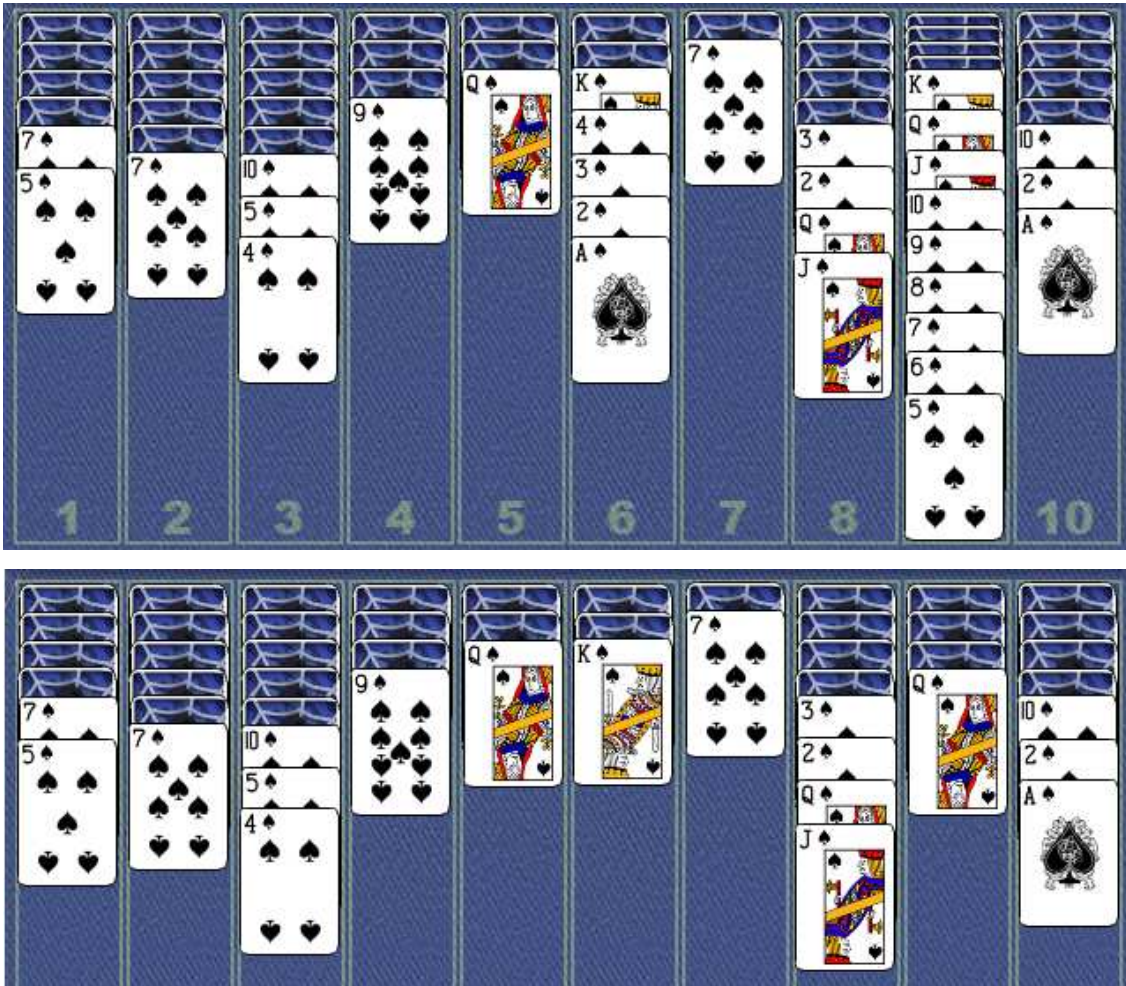


図6. カード除去

2.3 スパイダソリティアの難易度

スパイダソリティアには3種類の難易度が設定されており、ゲームの開始時にどの難易度でプレイするかを選択することができる。

2.3.1 初級

名前の通り、難易度設定の中でもっとも簡単にゲームをクリアできる。具体的には、1種類のマークを用いてゲームを行う。この場合、スペードのみ1から13の束を計8組、104枚を使ってプレイする。

マークを考慮に入れるに入れる必要がなく、単純に数字のみを追っていけばいいので、その分簡単にクリアできる。

人が初級をプレイしたときのクリア確率は、約80%ほどであると言われている。

(上に見てきた例で使われているのは全て初級の場合である)

2.3.2 中級

初級が1種類のマークを使っていることに対して、中級は2つのマーク（スペードとハート）を使ってプレイする。スペードの1から13を4組、ハートの1から13を4組、計8組104枚使ってゲームを行う。

初級に比べてマークの概念がある分クリアするのが難しく、その戦略も初級に比べてはるかに複雑なものとなる。

人がプレイした場合のクリア確率は40~50%程度だと言われている。

2.3.3 上級

2組のトランプ計104枚を使ってプレイする。4種類全てのマークを使う分、ゲームはさらに複雑となる。

人がプレイした場合のクリア確率は非常に低く、ある程度のスパイダソリティアの経験者でも上級はクリアしたことがない、といった例も少なくない。

本研究では、初級について考えていく。できれば中級に挑戦したかったのだが、マークの概念が加わった途端にアルゴリズムが非常に複雑なものとなり、とてもではないが收拾がつかないような状態となってしまったため断念した。

3. 学習手法と

クリアルーチンの作成方法

1.3 節で記述した通り、クリアルーチンを作るにあたり、評価要素とそれに対応する重みの積の和からなる評価関数を使用する。

評価要素 s と t 、 s に対応する重み w と t に対応する重み v を用いると、場の評価関数 S は

$$S = s_1 \times w_1 + s_2 \times w_2 + s_3 \times w_3$$

という形で、第 i 列の評価関数 $T(i)$ は、

$$T(i) = t_1(i) \times v_1 + t_2(i) \times v_2 + t_3(i) \times v_3 + \dots$$

という形でそれぞれ計算することができることも記述した通りである。

ここでは、学習の手法と重みの調整方法、評価関数の詳細について詳しく述べる。

3.1 教師あり学習

一般的に教師あり学習とは、幾つかの学習例と各学習例に対する目標出力を与え、目標出力と実際の出力が一致するように重みを調整する方法である。

この研究に関して言えば、評価関数 S と $T(i)$ はそれぞれ別の出力を目標とする。

S は場の評価関数であるが、各評価要素が「場を見る限り、ゲームが進んでいる」という場合に S の値が小さくなるように重みを調整する。すなわち、 S が小さければ小さいほど場が進んでいると判断できるように調整する。

$T(i)$ は第 i 列の評価関数であるが、これは各列が移動するカードの選択肢としてどれだけ優れているかを評価する基準となる。具体的には、 $T(i)$ を $i = 1, 2, \dots, 10$ でそれぞれ算出し、その結果が最も大きかった i を選び、カード移動元の候補とする。ゆえに、移動するカードの候補として優れていれば優れているほど、 $T(i)$ が大きくなるように重みを調整する。

3.2 重みの調整

3.1 節で w と v を準備したが、この値を調整することが学習の大きな目的となる。まず w_1, w_2, w_3 および v_1, v_2, v_3, \dots に、初期値として全て同じ値を入れる。

次に、教師を用いて学習させていく。

場の評価関数に関しては、 S が小さければ小さいほど場が進んでいる、という評価をするのが目的なので、教師のゲーム進行に合わせて徐々に S が小さくなっていくように w_1, w_2, w_3 を調整する。

各列の評価関数に関しては、教師が各場面で移動の候補として選んだ列に着目する。その列の特徴を調べ、特徴と合致する評価要素の重みを大きくし、逆に、教師が選んだ列の特徴としてあまり見られないものの重みは小さくする。

このように学習させることで、教師が選んだ列に顕著に見られる特徴は、その出現回数に比例して重みが大きくなる。

3.3 評価関数

上記のような手順で学習させ重みを調節させることで、実際に評価関数の値を算出することができるようになる。

例えば、第 i 列から「表向きのカードが5枚ある」、「裏向きのカードが2枚ある」、という2つの特徴のみを見つけることが出来た場合を考える。評価要素については表 3. 各列の評価要素 (p31) に記載しているが、「表向きのカードが5枚ある」という評価値は t_5 、「裏向きのカードが2枚ある」という評価値は t_{10} である。この時、 $t_5 = t_{10} = 1$ となり、 $t_k = 0$ ($k \neq 5, 10$) となるので、その評価関数は

$$T(i) = t_5(i) \times v_5(j) + t_{10}(i) \times v_{10}(j) = v_5(j) + v_{10}(j)$$

となる。つまり、「表向きのカードが5枚ある」及び「裏向きのカードが2枚ある」といった特徴がもし教師に顕著に見られる特徴であれば、重みの調整によって $v_5(j)$ および $v_{10}(j)$ は他の評価要素に比べて相対的に大きな数となっている。よって、 $T(i)$ の値もまた、他の列に比べて相対的に大きくなる。

すなわち、教師に顕著に見られる特徴と同じ特徴を持っている列の評価関数が高くなる。

3.4 クリアルーチンの作成方法

以上のように評価要素を設定し、重みを調節することで、評価関数の値を求めることが可能となった。

本研究では、このようにして求めた S と $T(i)$ を用いてスパイダソリティアのクリアルーチンを作成する。以下、その具体的な方法を説明する。

ゲームが開始されると、まず移動するカードを選択する必要がある。この時、まず S を求め、カード移動の選択が迫られた場（以後、「注目している場」と呼ぶ）におけるゲーム進行状況を判定する。

この時、注目している場のゲームの進行状況が第 j 段階であったとする。

次に、注目している場における、各列の評価関数 T を求める。

場の列の左から第 1 列、第 2 列、…第 10 列と呼ぶことにする。

まず、第 1 列の評価関数 $T(1)$ を求める。注目している場のゲームの進行状況は第 j 段階であったことから、それに対応する重みを使って、

$$T(1) = t(1) \times w(1, j) + t(2) \times w(2, j) + t(3) \times w(3, j) + \dots + t(29) \times w(29, j)$$

を求める。 $T(2), T(3), \dots, T(10)$ も同様にして求める。

$T(i)$ が大きいほど、カード移動の選択肢として優れている列となるように設定していたので、求めた $T(1), T(2), \dots, T(10)$ を比較し、一番大きい列を探す。その上で、一番大きい列のカードを移動させる。

このような手順を、移動するカードの選択が迫られる度に行うのが、本研究で作成するスパイダソリティアのクリアルーチンである。

4. クリアアルゴリズムの作成

1.3 節で記述した通り、本研究では

- ① クリア確率の大きなアルゴリズムを作成する。
- ② ①のアルゴリズムを教師として、クリアルーチンを作成する。

という手順でスパイダソリティアの自動クリアルーチンを作成する。

そこで、この章では効率的なクリアアルゴリズムを作成することについて考えていく。

まず 4.1 節で基本的なクリアアルゴリズムを作成してその性能を評価する。その基本的なクリアアルゴリズムを基に、4.2 節ではアルゴリズムにいくつかの改良を加え、その性能を評価する。

最後に今回学習の教師として使用する高レベルなクリアアルゴリズムについて 4.3 節で紹介する。クリアアルゴリズムを学習の教師として使うためには、高い確率でゲームをクリアするアルゴリズムを作成する必要がある。本研究で作成した高レベル・クリアアルゴリズムでは、スパイダソリティアを 90.8% の確率でクリアすることができ、学習の教師としては十分なものである。

4.1 クリアアルゴリズムの作成手法

スパイダソリティアをクリアするためには、プレイヤーは大きく 2 つのことを考える。1 つは「移動するカードの選択」であり、もう 1 つは「空き列の処理」である。

プレイヤーはもう 1 つ、「カードを配るタイミング」について考えるが、これは一般的には「移動できるカードがない」かつ「空き列がない」状態になった時にするのが一番良いとされている。よって今回作成するアルゴリズム全てにおいてこの方法を適用する。

4.2 基本的なクリアアルゴリズムの作成

「移動するカードの選択」と「空き列の処理」に関して最低限の処理をするようなアルゴリズムを作成することで、基本となるクリアアルゴリズムを作成する。

4.2.1 カード移動

カード移動は、移動できるカードを見つけたらすぐに移動させる、という方法をとる。

具体的には、まず一番左の列に注目し、列の一番上にあるカード束の一番下の数字に着目する。次に、左から2番目の列、3番目の列というように順に見ていき、一番左の列のカードを移動することができる列を見つけた地点で探索を終了し、見つけた場所にカードを移動させる。

一番左の列のカードを移動する先が見つからなかったら、次に左から2番目の列のカードの移動先を探す。

このように順次移動先を探していき、1つ見つけた地点ですぐ探索を終了し、カードを移動させる。

移動させる先が全く見つからなかった場合は、空き列の処理をする。空き列も無い場合は、カードを配る。

4.2.2 空き列処理

空き列には、任意のカードを置いてよいというルールであった。

そこで、空き列に関しては、適当なカードを移動させる。

実際は一番左の列の一番上にあるカード束を、空き列に移動させる。このとき、一番左の列にカードが1枚しかない場合に注意する。1枚しかないカードを空き列に移動させると、一番左側の列が今度は空き列となり、余計な移動してしまうばかりか、実装の方法によっては無限ループに陥る危険もある。そこで実装の方法としては、「空き列を見つけたら、2枚以上のカードが重なっている任意の列からカードを1枚持ってきて埋める」とすると良い。

4.2.3 基本的なアルゴリズムの実装と評価

「カード移動」「空き列の処理」に関して上記のような簡単なアルゴリズムを実装すると、スパイダソリティアをクリアすることができるアルゴリズムを作成することができる。

このアルゴリズムを実際に実装し、スパイダソリティアを1000回プレイさせてみたところ、クリア確率は42.6%だった。

4.3 クリアアルゴリズムの改良

4.2 節で作成したクリアアルゴリズムは最低限の実装しかしていないため、クリア確率が低かった。そこで、「カードの移動」に着目して、いくつか改良の手段を考えていく。

最低限の実装の場合は、移動できるカードを見つけ次第そのカードを移動させるという単純なものだった。

実際のゲームでは、各場面で移動できるカードが複数存在する 경우가多く、どのカードを優先的に移動させるかの選択がゲームをクリアする上で重要な要素となる。そこで、どのカードを優先的に移動させるかについていくつか改良の候補を挙げ、実装した上で最初のアルゴリズムと比較し、その性能を評価する。

こうすることで、効率的にクリアするために必要な要素はないかを探るのが目的である。今回優先させる基準とするのは以下の2つである。

- ・ **移動する候補のカードのすぐ下に、裏返しのカードが置かれているものを優先する。**

このようなカードを優先することで、カード移動によってある列の一番上のカードが取り除かれた場合に、裏返しのカードを表向きにする確率が高くなる。

不完全情報ゲームは場の状態から集めることのできる情報をいかに多くするかが鍵になるので、裏返しのカードを表向きにする確率が高くなればその分だけ場の状態から集めることのできる情報が増えるので、効率よくクリアできるようになるのでは、という考えである。

- ・ **移動する候補のカードの列にある裏返しのカードの数を比較し、多いものを優先する。**

場の状態から集めることのできる情報を増やすのが目的、というのは上記と同じであるが、このアルゴリズムの場合は裏向きのカードが多く残っている列を優先的にカード移動の候補として選ぶ。こうすることで、たくさん裏返しのカードが残っている列のカードを優先的に移動させることができ、それにより、裏返しのカードを表向きにする確率が高くなる。

以上、2つの優先基準についてそれぞれ実装し、それを「改良アルゴリズム1」「改良アルゴリズム2」と呼ぶことにし、それぞれについての実装方法を説明した上で、実際に実装した時の性能を評価する。また、この実装によっていくつかの予測が立てられるので、それについても述べる。

4.3.1 改良アルゴリズム 1 の実装方法

改良アルゴリズム 1 を実装するには、移動できるカード（束）のすぐ下のカードが表か裏かを判定するメソッドを追加すればよいだけである。

そのようなメソッドによって、すぐ下のカードが表か裏かがわかるので、あとは、すぐ下のカードが裏返しになっているようなカード移動の候補を探してくるだけで実装することができる。

4.3.2 改良アルゴリズム 2 の実装方法

改良アルゴリズム 2 の実装には、ある列の裏返しのカードの枚数を数えるメソッドを追加する。

裏返しのカードは必ず列の一番下から数枚、という形で固まっているので、ある列の一番下のカードが表か裏か調べ、裏だったら下から 2 番目のカードを調べ…といった形で裏返しのカードを数え上げていくとよい。

4.3.3 改良アルゴリズムの性能評価と考察

最初に作った基本的なアルゴリズムと、改良アルゴリズム 1、改良アルゴリズム 2 について、同じスパイダソリティアの問題を解かせるというテストを 1000 回行い、その性能を比較する。

以下は、その結果である。

	基本アルゴリズム	改良アルゴリズム 1	改良アルゴリズム 2
クリア確率	47.2%	57.4%	61.3%
このアルゴリズムでのみ解けた問題	6.1%	5.4%	8.2%

表 1. 基本アルゴリズムと改良アルゴリズムの性能比較

考察

基本アルゴリズムに比べて、改良アルゴリズム 1、2 共に若干クリア確率が上がったことは確認できたが、明確な性能の違いはなかった。

また、1つのアルゴリズムで解けた問題が、他の問題では解けない、といった例が少なからず見受けられた。特筆すべきは、改良アルゴリズム 1、2 では解けない問題が基本アルゴリズムで解ける、といった例が 6.1% もあったことである。これは、基本アルゴリズ

ムが改良アルゴリズムに比べて必ずしも劣っているわけではない、ということの意味している。

この結果から、いくつかの予測をたてることができる。

予測①

スパイダソリティア自体がランダム性の高いゲームである可能性がある。

初期状態のカードの配布状態によって、有効なカード選択の手段が変わってくるのではないだろうか。実際、上記の3つのアルゴリズムにおいて、1つのアルゴリズムで解けた問題が他のアルゴリズムでは解けなかった、という例が3つのアルゴリズムについて同じような確率で存在していることから、その可能性は高いと言える。

予測②

ゲームの進行具合によって、有効なカード選択の手段が変わってくるのではないだろうか。ゲーム中盤から終盤にかけて、裏返しのカードの枚数は少なくなっていく。特に終盤は裏返しのカードが全くなくなるので、裏返しのカードに依存したカード選択の手段は効果がなくなる。よって、改良アルゴリズム1と2はいずれも裏返しのカードに依存したクリアアルゴリズムなので、ゲームの中盤から終盤にかけてその効果が薄くなる。逆に、終盤にかけて有効な手段が存在する可能性がある。

これらの予測から、1つの手段に限定してスパイダソリティアをクリアしようとしても、なかなかクリア確率が上がらない可能性が示唆できる。よって、ゲームの進行具合によってカード移動の候補を選択する手段を色々に変更していくのが効率的であると判断できる。

この予測は、クリアルーチンの作成時に考慮していくとして、別の視点からクリアアルゴリズムの改良方法を考える。

4.4 高レベル・クリアアルゴリズムの作成

スパイダソリティアは不完全情報ゲームであるということに立ち返る。

不完全情報ゲームは、場から得ることができる情報を最大限に利用して、プレイヤーが行動を決定していく、というものである。当然、プレイヤーが得ることのできる情報が多ければ多いほど、ゲームを効率的に進めることができる。

ここで、スパイダソリティアにおける「伏せられた情報」について考える。

スパイダソリティアの初期状態を思い出して欲しい。この地点でプレイヤーが場から得ることができる情報は、各列の一番上で1枚だけ表向きになっているカード計10枚である。

裏返しになっているカード44枚と、配布用のカード50枚についてはプレイヤーは知ることができない。ここで、配布用のカードは実際に配るまでその内容を知ることができず、また、無意味に配ると非効率的であることから、配布用のカードの情報は伏せられたまま得ることができない、としておく。

よって、プレイヤーが得ることのできる情報を増やすためには、「場で裏返しになっているカードをより多く表向きにする」ということを考えればよい。

ここで、「場で裏返しになっているカードの情報をより多く得る」ための方法について考えていく。

スパイダソリティアには、「**特別な場合を除き、移動させたカードを移動させる前の状態に戻すことができる**」、というルールがある。

これは、例えば将棋や囲碁のようなゲームをコンピュータと行う場合の「待った」の機能と同等のものであると考えるとよい。すなわち、特別な場合を除き、一度行った手をなかったことにできるというものである。この機能は「特別な場合」以外では続けて使用することができ、例えば、カード移動を10回繰り返したあとで10手前の状態に戻すということも可能である。

ここで言う「特別な状態」とは、「カード配布の前後」と「カードを取り除く操作の前後」である。すなわち、カードを1度配布したら、配布する前の状態には戻せない。また、1から13までのカードを揃えて場から取り除いた場合、取り除く前の状態に戻すことはできない、といった内容である。

この2つの特別な場合を除きカード移動後の状態をカード移動前の状態に戻すことができるので、次のような手段をとることで、裏返しになっているカードの情報を得ることができる。

- ・まず、移動できるカードを適当に1枚選び、移動させる。
- ・その移動によって裏返しのカードが表向きになった場合、そのカードの情報（すなわちカードの柄と数字）を記憶した後に、移動したカードを移動前の状態に戻す。

このような手段をとることで、裏返しのカードの情報を得ることができる。実際は移動

前の状態に戻すことで、1度表向きにしたカードが裏向きの状態に戻るが、プレイヤーはそのカードの柄と数字をすでに得ることができたので問題はない。

これを複数の移動できるカードについて行うことで、場の状態を進めることなく裏返しのカードの情報を集めることができ、プレイヤーに有利な状況が生まれる。

このスパイダソリティアのルールを最大限に用いて、以下のようなアルゴリズムを考える。なお、このアルゴリズムは各カード移動の選択の前に行うことになるので、「前処理アルゴリズム」と呼ぶことにする。

4.4.1 前処理アルゴリズム

「カードの移動後の状態から移動以前の状態に戻すことができる」というルールを利用し、以下のようなアルゴリズムを作成する。

- ・ある状態（初期状態と呼ぶ）について、ある1つの列に注目する。（通常はまず、一番左の列に注目する。）
- ・その列について、移動できるカードがなくなるまでカード移動をさせる。（まずは一番上のカードを移動する。それにより新たに一番上にきたカードが移動できるようならそれも移動する、といったことを、移動できるカードがなくなるまで繰り返す）。
- ・移動できるカードがなくなった段階で、移動したカードを全て移動以前に戻す。

このような処理を一番左の列から順に全ての列にほどこすことで、場のカードの並びの状態は初期状態のまま、本来裏返しになっているカードの情報も得ることができる。

場の状態に戻すことで表向きにしたカードも裏返しに戻るが、一度情報を得ることができたカードは裏向きでも表向きと同様にあつかうことができる。

4.4.2 前処理アルゴリズムの利点

前処理アルゴリズムを用いることで場の情報をより多く知ることができるのはもちろんだが、その最大の利点は、「各列で上から何枚（カード束であることも考慮すると上からなん束）のカードを移動させることができるか」を知ることができる点にある。

仮に、上から5枚が移動可能なカードで構成されているような列があった場合、その列のカードを順次移動させるだけで最低でもゲームを5手分進めることができる。

上からカードが1枚しか移動できない列をカード移動の候補として選択するよりも、上から複数枚のカードを移動することができる列をカード移動の候補として選択した方が効率よくゲームを進めることができるのは自明なので、前処理アルゴリズムをほどこした上

で、移動できるカード（束）が一番多く重なっている列をカード移動の候補として選択することで、良いクリアアルゴリズムを作ることができそうである。

4.4.3 空き列の処理

基本アルゴリズムの作成以降全く触れてこなかったが、空き列の処理を改良することも重要である。

実際人がスパイダソリティアをプレイする場合には、この空き列を最大限に利用してゲームを進めていく、といった手段をとる。例えば空き列がある状態で、それとは別に一番上から4, 3, 5とカードが乗っている列（これを*i*列と呼ぶ）があるとす。この場合、次のような操作をすることで、カードを束にすることができる、すなわち、

- ・ 4のカードを空き列に移動させる。
- ・ 4を移動させたことにより、*i*列の一番上が3になり、空き列には4が置かれた状態になる。ここで、*i*列の一番上の3を空き列に置かれた4の上に移動させる。
- ・ 3を移動させたことにより、*i*列の一番上が5になり、空き列だった場所には3, 4の順でカードが置かれている。3, 4はカード束なので、最後にこの束を *i*列の一番上に移動させる。束の一番下の数字が4であり、*i*列の一番上のカードが5であることから、この移動は可能である。
- ・ このような操作をすると、空き列だった場所は空き列に戻り、上から4, 3, 5と重なっていた *i*列のカードは、3, 4, 5という束を作る順に並べ直すことができる。

空き列を用いるとこのように他の列のカードを入れ替え整えることができる。

このようなカードの入れ替えを実装することができればより効率的なアルゴリズムを作成することができるが、カードの入れ替えのパターンは膨大であり、効率的に見極める手段を思いつうことができなかつたので、本研究では実装には及んでいない。

本研究による空き列の処理の方法として「各列の一番上にあるカードもしくはカード束の中で、一番下の数字が最も大きいものを優先的に空き列に移動させる」というものを採用する。

基本アルゴリズムでの空き列の処理は「適当なカードを空き列に移動させる」といった程度にとどめてあつたので、条件付けがしてある分改良されていると言える。

条件を「一番下の数字が最も大きいものを優先的に空き列に移動させる」としたのは、先に説明した「1 3のカードは空き列にしか移動できない」というルール上の理由に起因する。

このように条件付けすることで、列の一番上に1 3のカードがある場合には、率先して空き列に移動させるようになり、次の段階での移動可能なカードの数を増やすことができる。それにより、ゲームを効率的に進めることが期待できる。

以上のような「前処理アルゴリズム」と「空き列の処理」を用いて、さらに効率のよい

クリアアルゴリズムを作成する。なお、このアルゴリズムは本研究におけるスパイダソリティアのクリアアルゴリズムとして最も高いクリア確率を納めることができたアルゴリズムなので、「高レベル・クリアアルゴリズム」と呼ぶことにする。

4.4.4 高レベル・クリアアルゴリズムの実装

高レベル・クリアアルゴリズムを実装する上で最も重要となるのは、先ほど見てきた「前処理アルゴリズム」である。各カード移動のフェーズの際に、まず前処理アルゴリズムを行い、場の情報をできるだけ得る。その上で、各列における移動可能なカードの枚数を調べ、最も移動可能カード枚数が多い列を、カード移動の候補として選択する。

空き列ができた場合は、空き列の処理を行う。

その他のフェーズに関しては、基本アルゴリズムと同様である。

以下は、ゲームのスタートからゲームクリア、もしくはゲームオーバーになるまでの流れである。

【高レベル・クリアアルゴリズムを用いた実際のゲームの流れ】

1. ゲームスタート
2. 移動できるカードがあるか？ない場合は7に進む。
3. 移動できるカードがある場合、前処理アルゴリズムを使用。
4. 前処理アルゴリズムにより情報を得た後、移動できるカードが一番多い列を選択し、その列の移動できるカードを全て移動させる。
5. 空き列はあるか？ない場合は2に戻る。
6. 空き列がある場合、空き列の処理をし、2に戻る。
7. 空き列はあるか？ある場合は空き列の処理をし、2に戻る。空き列がない場合は8に進む。
8. ここに到達した場合、移動できるカードも空き列もないという状態である。配るカードがあるか？ある場合はカード配布を1回行い、2に戻る。配るカードがない場合は9に進む。
9. ここに到達した場合、移動できるカードも空き列も、配るカードも無いという状態である。場にカードは残っているか？残っている場合は10に、1枚も残っていない場合は11に進む。
10. GAME OVER
11. GAME CLEAR

このような手順でゲームを進めるのが高レベル・クリアアルゴリズムである。前処理アルゴリズムを追加した以外は、ゲームの流れ自体は他のクリアアルゴリズムと同じである。

このようなアルゴリズムを実装し、実際にスパイダソリティアを1000回プレイさせた。以下はその結果と考察である。

4.4.5 高レベル・クリアアルゴリズムの

実行結果と考察

以下が、基本アルゴリズムと改良アルゴリズム1、改良アルゴリズム2、および高レベル・クリアアルゴリズムのそれぞれに対して同じスパイダソリティアの問題を1000通り解かせ、そのクリア確率を測定した結果である。

	クリア確率
基本アルゴリズム	44.7%
改良アルゴリズム1	53.8%
改良アルゴリズム2	64.5%
高レベル・クリアアルゴリズム	90.8%

表2. 高レベル・クリアアルゴリズムと他のアルゴリズムの性能比較

一見してわかる通り、高レベル・クリアアルゴリズムは90.8%という非常に高い確率でゲームをクリアしており、その性能は他のクリアアルゴリズムに比べてはるかに優れている。

空き列処理に関してはあまり効果的な実装はしていないので、高レベル・クリアアルゴリズムのクリア確率を跳ね上げることができた要因は前処理アルゴリズムを追加したと大きく関係する、と考えることができる。

このことから、いかに多くの情報を場の状況から引き出し効率的にゲームを進めていくかということが、スパイダソリティアをプレイする上で重要な要素か、ということを見ることが出来る。

目標としていたクリア確率90%を超えることができたので、この高レベル・クリアアルゴリズムを教師として使用することで、以下、スパイダソリティアの自動クリアルーチンを作成していく。

5. 学習による

クリアルーチンの作成

4.4 節で作成した高レベル・クリアアルゴリズムを使って、クリアルーチンを作成していく。大まかな手順は以下の通り。

【クリアルーチンの作成手法】

1. ゲームの進み具合を 10 段階にわけろ。
2. 各段階について、各列の評価要素とそれに対応する重みを作成する。
3. 改良アルゴリズムを使い、重みを調整する。
 - ・ ゲームの各状態において、改良アルゴリズムがカード移動の候補として選択した列に着目。
 - ・ その列に見られた特徴と合致する評価要素の重みは大きくし、特徴として見られなかった評価要素の重みは小さくする。

ここで、各手順について説明していく。

5.1 ゲームの進み具合の評価

改良アルゴリズム 1, 2 の実装の項での考察として、「ゲームの進み具合によって効率的なカード選択の方法が違うのではないか」といったことを挙げた。

よって、自動クリアルーチンを作る上で、「ゲームの進み具合」という要素を考えることにする。具体的には、ゲームの進み具合をいくつかの段階に分け、それぞれの段階において違ったカード選択の方法をとる、ということである。

スパイダソリティアをプレイするにあたり、「カードの移動を 1 回行う」及び「カードを 1 度配る」という操作をそれぞれ 1 手と数えたとき、高レベル・クリアアルゴリズムはゲーム開始からゲームをクリアするまで平均して 200 手程度かかる。

すなわち、ゲームの進み具合を 10 段階ほどに分ければ各段階に平均 20 手ずつ収まることになり、ゲームの進み具合を判定するのに十分な段階分けであると判断した。

ここで、ゲームの進み具合をどのように調べるかについて言及する。ゲームを進めていくと、次のような変化が生じる。

1. ゲームを進めるにつれて、場のカードの束が増えていく。
2. ゲームを進めるにつれて、場の裏返しのカードが減っていく。
3. ゲームを進めるとカード配布も行うので、配布できるカードの残り枚数が減っていく。

よって、各場の状況から以上に挙げた要素について調べていき、そこからゲームの進み具合を判定する関数を作成すればいいように思える。すなわち、

- ・ 場にある表向きのカードの中で、束になっているものがどの程度あるか。
(表向きのカードが同じ枚数あるような2つの状態を比べたとき、束を成しているものが多ければ多いほど、ゲームが進んでいると判断することができる)
- ・ 場に裏向きのカードが何枚あるか。
(単純に、裏向きのカードが少なければ少ないほど、プレイヤーが得ることのできる情報は増えるので、場が進んでいると判断することができる)
- ・ 配ることができるカードが何枚残っているか。
(カードを配った直後は「場にある表向きのカードの中で束を成しているもの」の割合が小さくなってしまうので、配るカードが何枚残っているかについても、ゲームの進み具合を判断する要素として挙げる必要がある。)

ゲームの進み具合を算出する評価関数

ゲームの進み具合を算出する評価関数として、 S というものを作る。

この S の値によって、ゲームの進み具合を判定するようにしていく。

まず、前述の3つの「ゲームの進み具合を判断する基準」を具体的に評価要素 s_1, s_2, s_3 として書き下していく。なお、 s_1, s_2, s_3 にはそれぞれ0から1の間の実数が入り、それぞれが小さければ小さいほどゲームが進んでいると評価できるように設定する。 s_1, s_2, s_3 それぞれの具体的な計算方法は以下の通りである。

- ・ $s_1 \dots$ (場にあるカード束の数) \div (場の表向きのカードの数)
- ・ $s_2 \dots$ (裏返しのカードの枚数) \div (初期状態での裏返しのカードの枚数 = 44枚)
- ・ $s_3 \dots$ (配ることができるカードの枚数) \div (初期状態での配ることができるカードの枚数 = 50枚)

この評価要素に対応する重み w_1, w_2, w_3 を設定し、場の進み具合を判定する S を計算する。

$$S = s_1 * w_1 + s_2 * w_2 + s_3 * w_3$$

実際の S の使い方としては、

- ・ ある場において、その状況から s_1, s_2, s_3 を計算する。

- ・ それぞれに重みを掛けたものを全て足し、 S を算出する。
 - ・ S の値によって、ゲームの進行状況を判断する。
- といった感じである。

5.2 各場における列の優先順位の判定

ゲームの進み具合の判定と同じように、列に関して評価基準 $t_1, t_2, t_3 \dots$ を設定し、そのおのおのに対応する重み $v_1, v_2, v_3 \dots$ を設定することで列の評価とする。

具体的な方法として、まず、列に関する評価基準を設定する。これは、注目している列が持つ特徴と一致する評価基準は値を 1 に、一致しない評価基準は 0 にするという、真偽値の形をとる。

以上、計 29 個の評価基準を設け、各列を評価していく。

この評価は場の進行具合によっても変わるので、各段階において重みを使い分ける。

第 i 列の、第 j 段階における評価関数は以下ようになる。

$$T(i, j) = t_1(i) \times v_1(j) + t_2(i) \times v_2(j) + t_3(i) \times v_3(j) + \dots + t_{29}(i) \times v_{29}(j)$$

ここで、 t_1, t_2, t_3, \dots は各評価基準であり、列によって値が変わるので、 $t_1(i)$ のように列番号を表す i の関数となっている。 v_1, v_2, v_3, \dots は各評価基準に対応する重みであり、列によって値は変わらないが、ゲームの進行具合によって重みは変わるので、 $v_1(j)$ のようにゲームの進行具合を表す j の関数となっている。

例えば、ゲームの具合が第 5 段階であり、第 3 列について調べたところ、 t_3 と t_7 と t_{16} と t_{24} が特徴として合致し、その他の評価基準は特徴として見受けられなかったと仮定する。このとき、各 t は真偽値であることから、 $t_3 = t_7 = t_{16} = t_{24} = 1$ であり、その他の t は全て 0 になる。

よって、この場合の $T(i, j) = T(3, 5)$ は、以下ようになる。

$$\begin{aligned} T(3, 5) &= t_3(3) \times v_3(5) + t_7(3) \times v_7(5) + t_{16}(3) \times v_{16}(5) + t_{24}(3) \times v_{24}(5) \\ &= v_3(5) + v_7(5) + v_{16}(5) + v_{24}(5) \end{aligned}$$

この式より、 t_3, t_7, t_{16}, t_{24} がそれぞれ第 5 段階において顕著に見られる特徴だとすれば、それに対応する重みも大きいはずなので、 $T(3, 5)$ の値が大きくなる。

このように定義した評価関数 $T(i, j)$ を各列ごとに算出し、 $T(i, j)$ が最も大きかった列を優先的にカード移動の候補として選ぶ、というのが今回作成する自動クリアルーチンの枠組みである。

t1	表向きのカード(束)が1組である。
t2	表向きのカード(束)が2組である。
t3	表向きのカード(束)が3組である。
t4	表向きのカード(束)が4組である。
t5	表向きのカード(束)が5組である。
t6	表向きのカード(束)が6組である。
t7	裏向きのカードの枚数が0枚である。
t9	裏向きのカードの枚数が1枚である。
t10	裏向きのカードの枚数が2枚である。
t11	裏向きのカードの枚数が3枚である。
t12	裏向きのカードの枚数が4枚である。
t13	裏向きのカードの枚数が5枚である。
t14	表向きのカードのうち移動できるカード(束)が一番上から数えて0組ある。
t15	表向きのカードのうち移動できるカード(束)が一番上から数えて1組ある。
t16	表向きのカードのうち移動できるカード(束)が一番上から数えて2組ある。
t17	表向きのカードのうち移動できるカード(束)が一番上から数えて3組ある。
t18	表向きのカードのうち移動できるカード(束)が一番上から数えて4組ある。
t19	表向きのカードのうち移動できるカード(束)が一番上から数えて5組以上ある。
t20	一番上のカード束の、一番下の数字は1である。
t21	一番上のカード束の、一番下の数字は2である。
t22	一番上のカード束の、一番下の数字は3である。
t23	一番上のカード束の、一番下の数字は4である。
t24	一番上のカード束の、一番下の数字は5である。
t25	一番上のカード束の、一番下の数字は6である。
t26	一番上のカード束の、一番下の数字は7である。
t27	一番上のカード束の、一番下の数字は8である。
t28	一番上のカード束の、一番下の数字は9である。
t29	一番上のカード束の、一番下の数字は10である。
t30	一番上のカード束の、一番下の数字は11である。
t31	一番上のカード束の、一番下の数字は12である。
t32	一番上のカード束の、一番下の数字は13である。

表3. 各列の評価要素

表3は、各列の評価要素の一覧である。

t1 から t6 までは、列にある表向きのカード及びカード束が何組あるかを評価する。カード移動は、あるカードを他の列に移動して束を作っていく操作なので、1度目のカード配布の前に各列にあるカード束は高々1つである。これに5回のカード配布が上乘せされるので、列ごとのカード束は最大6組となる。

t7 から t13 までは、列にある裏向きのカードが何枚あるかを評価する。裏向きもカードは初期状態で高々5枚であり増えることはないので、最大5枚である。

t14 から t19 までは、表向きのカードのうち移動できるカード(束)が一番上から数えて何組あるかを評価する。計算上、1つの列にある移動できるカードは最大11枚となるが、テストの結果を見る限り、3枚以上になることはごく稀である。そのため、ここでは評価要素として5枚以上というようにまとめた。

t20 から t32 までは、列一番上にあるカード(束)の一番下の数字を評価するものである。この数字が何かによって、他の列の一番上の数字が何ならば移動できるか、が決まるので、評価基準の1つとした。

5.3 学習

以上のように、スパイダソリティアの自動クリアルーチンを作成するために2つの評価関数を定義した。

S …ゲームの進行具合を評価する関数。

$T(i, j)$ …各列がカード移動の候補としてどの程度ふさわしいかを評価する関数。

2つの評価関数の使い方としては、

- ・移動するカードの候補を選択する必要がある状況の時に使用する。
- ・まずその場におけるゲームの進行具合を調べるため、 S を計算する。
- ・ S を計算することにより、その結果からゲームの進行具合を判断することができる。ゲームの進行具合を判断したら、次に、その進行具合の段階における各列の評価関数の重みを準備する。(第 j 段階のときには、重みは $v1(j), v2(j), \dots$ である。)
- ・準備した重みを使って、各列がどの程度カード移動の候補としてふさわしいかを、全ての列について計算する。
- ・計算された値 $T(1, j), T(2, j), \dots, T(10, j)$ を比較し、最も大きな値を出した列をカード移動の候補として選ぶ。

このような手順で効率的にゲームをクリアできるように、 S および T の重みを調整していく。

Sの重み、w1, w2, w3の学習

w1, w2, w3 に関しては、

$$w1+w2+w3 = 1 \quad (**)$$

を満たすように設定する。

高レベルクリアアルゴリズムにおいて、最初の20手が第1段階に、次の20手が第2段階に…最後の20手が第10段階に含まれるように重みを調整する。

s1, s2, s3 はそれぞれ0から1の間の実数であり、w1, w2, w3 は(**)より足すと1なので、

$$0 \leq S = s1 \times w1 + s2 \times w2 + s3 \times w3 \leq w1 + w2 + w3 = 1$$

より

$$0 \leq S \leq 1$$

となる。

よって、ゲームの進行具合を評価する関数Sは0から1の間の実数であり、値が小さければ小さいほど、ゲームが進んでいるということができる。

T(i, j)の重み t1(j), t2(j), …の調整

各列の評価関数の重みを調整するためには、Sについての学習が済んでいる必要がある。そこでまずはSについての学習をし、その後で各列の評価関数について学習させていく。

学習には高レベル・クリアアルゴリズムを使う。

実際にあるスパイダソリティアの問題を高レベル・クリアアルゴリズムに解かせ、各カード移動の選択時に、アルゴリズムがカード移動の候補として選択する列にはどのような特徴があるかを調べていく。それによって、アルゴリズムがカード移動の候補として選択する列に見られる特徴に対応する評価要素の重みは大きくし、逆に、アルゴリズムが選択したカード列に見られない特徴に対応する評価要素の重みは小さくする。

このように繰り返し学習することで、高レベル・クリアアルゴリズムが選択する列に顕著に見られる特徴の重みほど大きくなり、逆に、めったに見られない特徴の重みは小さくなる。

具体的な手順は以下の通りである。

- ・まず、高レベル・クリアアルゴリズムを用いてあるスパイダソリティアをプレイさせる。
- ・移動するカードを選択するフェーズにきたとき、最初にゲームの進行具合を調べるためにSを計算する。
- ・Sの計算結果により、現在見ている場はゲームとしてどの程度進んでいるのかを判定できるので、その段階に対応した列の評価関数の重みを準備する。
- ・ここで高レベルアルゴリズムによって、移動するカードの候補となる列を選択させる。

- ・高レベルアルゴリズムが選択した列に見られる特徴を調べる。
- ・その特徴に対応する評価要素 $t_i(j)$ について、それに対応する重みを大きくする。(高レベルアルゴリズムが選択した列の特徴に対応する評価要素の重みを増やすことで、重みの大きな評価要素の特徴を持つ列が高レベルクリアアルゴリズムによって選ばれやすい、ということが言えるようになるため)。

このように何度も学習をさせていき、重みを調整していく。

実際にはこのように学習させる操作を、スパイダソリティア10000ゲーム分行なった。

列の評価要素とその重み

何度か説明に上がった通り、列の評価要素に対応する重み v は、ゲームの進行状況によってその値が変わる。これについての具体的な説明をここではしていく。

まず、改良アルゴリズム1, 2の考察で言及したように、同じ評価要素でもゲームの進行状況によってその重要性が変わってくるのではないかと推測した。そこで、ゲームの進行状況によって、列の評価要素に対応する重みを変えることにした。

例として、 $v_1(j)$ に着目してみよう。これは「注目している列に、表向きのカードが1枚ある。」という評価要素に対応する重みである。この重みは、場の進行状況によって全部で10種類存在する。 $v_1(1), v_1(2), v_1(3), \dots, v_1(9), v_1(10)$ である。

これらは全て「注目している列に、表向きのカードが1枚ある」という評価要素に対応している重みなのだが、ゲームの進行状況によって使い分ける。実際のゲームを考えてみると、初期状態においては全ての列が「表向きのカードが1枚ある」という条件を満たすので、序盤のうちはこの評価要素に対応する重みも大きくなる。だが、中盤に差し掛かると、1つの列に表向きのカードが何枚も重なっている状態が多くなるため、ゲームが進行するにつれて「表向きのカードが1枚ある」という評価基準を満たす列は少なくなっていく。よって、この要素においては、 $t_1(1)$ は比較的大きいが、 $t_1(2), t_1(3), \dots$ とゲームが進行していくにつれてその重みも小さくなると考えられる。

このように、同じ特徴の評価要素でもゲームの進行状況によってその重みが大きく変わることは容易に予測することができるので、ゲームの進行状況に応じてそれに対応する重みを変えていくことで、より柔軟な評価関数を作成することができる。

6. 評価と考察

6.1 クリアルーチンの性能評価

第5章で作成したクリアルーチンを作成し、その結果として得られた重みを使って1000回ゲームをプレイさせた。結果は下記の通りである。

7.

	クリア確率
基本アルゴリズム	46.8%
改良アルゴリズム1	53.7%
改良アルゴリズム2	61.2%
高レベル・クリアアルゴリズム	90.2%
学習による自動クリアルーチン	75.5%

図4. 高レベル・クリアルーチンと各アルゴリズムとの性能比較

学習による自動クリアルーチンのクリア確率は普通の人間がゲームを行った場合のクリア確率80%には若干劣るものの、十分な結果を得ることができたと判断できる。

また、最初に作成したいくつかのアルゴリズムよりは高いクリア確率を達成することができたので、自動クリアルーチンとしては悪くはない結果なのではないかと判断できる。

6.2 さらに改良をするために

今回の自動クリアルーチンの作成においては、主にカード移動の際の移動するカードの候補の選択に関して研究の大部分を割いた。

始めに述べたように、スパイダソリティアを人がプレイする上で考えることのほとんどは「移動できるカードの候補が複数あった場合、どのカードを選択するか」と「空き列の処理」についてである。今回実装したように、移動するカードの候補の選択は、スパイダソリティアにとって重要な要素であることは確かである。しかし、空き列の処理も、ないがしろにはしてはいけない。

今回は、空き列の処理については詳しくは触れず、また、改良もほとんどしなかったが、空き列を効率的に使えば他の列の束を整理することもでき、ゲームを効率よく進めることも可能になる。そういった意味で空き列処理についてさらに改良すれば、より良い自動クリアルーチンを作成できる。

また、今回はスパイダソリティアの中でも初級の研究に留まった。中級、上級に関しても研究をしてみたかったが、マークの要素が追加されるととたんにパターンが膨大になりクリアアルゴリズムを作るのも困難であったため、今回は断念した。本当であれば、スパイダソリティアといえば4色スパイダソリティアを指すので、「スパイダソリティアの自動クリアルーチン」と言うのであれば、やはり4色スパイダをクリアするルーチンを作りたいと思った。

6.3 最後に

研究に取り掛かる前は、スパイダソリティアも中級くらいまでならすぐに実装できそうだと、などと安易に考えていた。

人間が当たり前のように考えていることでも、プログラムに書き起こすには大変な苦勞が伴うため、初級のスパイダソリティアの自動クリアルーチンを作るだけで手いっぱいになってしまった。

しかし、一見ランダム性の強いゲームに見えるスパイダソリティアでも、自動クリアルーチンを作成することである程度の成果を出せることがわかったのは大きな収穫である。

自分の研究によって不完全情報ゲームの研究がまた1歩進んだ、などと大それたことはとても言えないが、少なくとも、現在に至るまでほとんど（もしくは全くかも知れない）研究されてこなかったであろうスパイダソリティアについて研究し、少しでも結果を出すことができたことを幸せに思う。

7. 関連研究

[1]木カーネルを用いたSVMによる麻雀打ち手の順位学習

三木 理斗、三輪 誠、近山 隆 著

<http://www.logos.ic.i.u-tokyo.ac.jp/~miki/doc/gpw2008.pdf>

[2]麻雀の牌譜からの打ち手評価関数の学習

北川 竜平、三輪 誠、近山 隆 著

<http://www.logos.ic.i.u-tokyo.ac.jp/~kitagawa/papers/gpw07.pdf>

不完全情報ゲームの研究はあまりされておらず、特にスパイダソリティアに関しての研究は探した限り全くされていなかった。

国内では、麻雀の研究が東京大学の近山隆教授を中心として行われているので、関連研究として挙げた。

なお、ゲームプログラミングに関心を持つ研究者の活発な議論の場として「ゲームプログラミングワークショップ」というものが国内で毎年開催されており、完全情報ゲーム、不完全情報ゲーム、思考型ボードゲーム、パズルゲームなどゲーム情報学に関する幅広い研究発表がされている。

8. まとめ

本研究では、不完全情報ゲームの1つとしてのスパイダソリティアのクリアルーチンを作成した。その結果、

- ・ 90%以上の確率でスパイダソリティアをクリアするアルゴリズムの作成
- ・ 75%程度の確率でスパイダソリティアをクリアするアルゴリズムの作成

の2つを達成した。

今後の課題として、

- ・ 空き列処理に関する更なる改良による、より効率的にスパイダソリティアをクリアするルーチンの作成
- ・ スパイダソリティアの中級、上級のクリアルーチンの作成

などが挙げられる。

9. 参考文献

[1]ニューラルネットワーク

http://www.sist.ac.jp/~suganuma/kougi/other_lecture/SE/net/net.htm

[2]ニューラルネットワークの学習理論

http://www.cbrc.jp/~asai/LECTURE/H16SeitaiJouhouRon/NN_learning.pdf

[3]スパイダソリティア 遊び方

<http://windows.microsoft.com/ja-JP/windows-vista?Spider-Solitaire-how-to-play>

[4]スパイダソリティア必勝法

<http://kiriusa.cool.ne.jp/living/diary/f030509.html>

[5]トランプゲームの思考ルーチンの組み方

http://www.geocities.jp/exon_soft/sikou.html

最初、本研究は「ニューラルネットワーク」という計算モデルを使ったクリアルーチンの作成を目的としていたが、スパイダソリティアのゲーム性に用いるのが困難であったため、断念した。これは「系統的に難しい」というわけではなく、私の力不足だったように思える。ニューラルネットワークは、不完全情報ゲームの思考ルーチンを作成する上で高い効果が期待できる計算モデルであり、事実として国内での麻雀研究においてはニューラルネットワークを用いた思考ルーチンの作成が主流である。

本研究では、ニューラルネットワークの考えを簡略化したものをクリアルーチンの枠組みとして使用した。「各状況における特徴の評価」や「教師あり学習を用いた重み調節の方法」など、全体的な流れはニューラルネットワークを用いた思考ルーチンの作成方法と同様である。よって、今回の研究ではニューラルネットワークの学習理論を参考にした。

また、スパイダソリティアのルールや必勝法、トランプゲームの思考ルーチンの組み方なども併せて参考にさせていただいたので、挙げておく。

謝辞

本研究を進めるにあたり多大なるご指導ご鞭撻を頂いた、東京工業大学 数理・計算科学専攻教授の佐々政孝先生に深く感謝の意を表します。

また、佐々研究室の皆さまには、さまざまな面で助力を頂きました。改めまして、ここに深くお礼申し上げます。