Vol.2020-GI-43 No.20
2020/3/14

ガイスターゲームにおけるモンテカルロ法を利用した駒推定およびブラフ手の生成可能性の検証

鴛淵 隆斗[1]　佐藤 直之[1]

概要：不完全情報ゲームでは情報の予測や騙しあい等の，完全情報ゲームとは違った課題が重要となる事が多い．本研究では不完全情報ゲームの一種である「ガイスター」を対象に，敵の駒の色の予測とブラフ手の生成を試みた．駒色の予測のために，「相手から見えるゲーム状態を開始局面としたモンテカルロ法」による探索を行い，実際の着手選択との整合性をヒントに予測を立てるアルゴリズムを利用した．このアルゴリズムを利用したモンテカルロ法プレイヤは，利用しないプレイヤに対しての対戦実験において 56% の勝率で勝ち越した．またブラフ手の生成のために，同様に「相手から見えるゲーム状態からのモンテカルロ法」を基にした手法を提案し，あるゲーム局面 1 つにおいてブラフの手が生成される様子を観察した．

キーワード：ゲーム情報学，不完全情報ゲーム，ガイスター，モンテカルロ法

Abstract:
It is important to guess and deceive opponent in imperfect information games. This research proposed an estimation algorithm to guess colors of opponent pieces. Monte Carlo algorithm is used in this proposed method, and result in winning against a naive Monte Carlo player over 500 matches with 56% win rate. Additionally, this paper discussed a way to produce bluff moves with the proposed method in geister game.

Keywords: Game informatics, Imperfect information game, Geister, Monte Carlo method

1. 序論

ゲームのコンピュータプレイヤの強さの向上については，多くの研究がなされてきた．完全情報ゲームの分野ではチェスの DeepBlue[1] から始まり，将棋 [2][3] や囲碁 [4] ではプロの人間プレイヤに匹敵する実力のプレイヤが開発された．不完全情報ゲームにおいても研究は盛んに進められていて，乱数が大きく作用するゲームとしてはバックギャモンを扱ったプレイヤ開発が有名であり [5]，相手プレイヤに対する情報の秘匿があるゲームとしては麻雀 [6]，人狼 [7] などが例えば研究の対象になっている．

不完全情報ゲームには完全情報ゲームとは違った課題があり，確率的な状態遷移を探索でどのように扱うかといった点や，相手と自分の情報の不平等に対してどうアプローチしていくかといった点なども興味の対象となる．特に後者に関しては，そもそもゲームの解析を行ってナッシュ近郊を利用した最適戦略を追求したり，相手のみが知っている情報を機械学習によって推測したりと，アプローチが幅広い．しかし，それぞれのアプローチには制限もあって，最適戦略の導出をするためにはある程度簡潔なゲームを対象にする必要があり，機械学習もデータを用いた学習が必要になる．

そこで本研究では不完全情報ゲームのガイスターを題材として，「相手の立場になってモンテカルロ法を適用する」事による不完全情報の推測を試みた．相手の体験した状態と，その状態で選択した着手を手掛かりにして「相手の駒色がどのようであればその着手が合理的に強い着手であるのか」をモンテカルロ法で計算する事によって，相手のみが知る駒色の情報を推測していく．

さらに本研究では情報の不完全性の更なる応用として，相手をだます「ブラフ」の着手にも着目する．ある特定の局面で相手プレイヤの立場からのモンテカルロ法を行ってみて，提案手法の応用による「ブラフ手」の生成が可能であるかを議論する．

[1] 佐世保工業高等専門学校
Sasebo, Nagasaki 857–1174, Japan

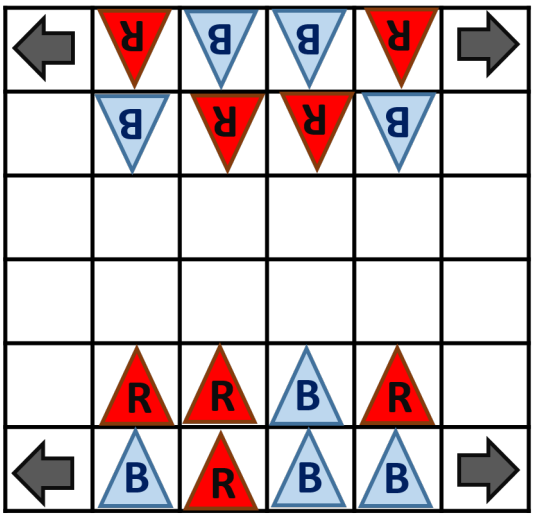

図 1　ガイスターの初期局面．所定の自陣 8 マスの中で赤と青を，相手に見えないように自由に配置できる．対戦相手陣地の脇に見える矢印は脱出マスであり，青の駒をそこから盤外に逃がせば即座にゲームに勝利する．

2. 研究対象

2.1 ガイスター

研究対象のゲームであるガイスターについて述べる．ガイスター（Geister）は，おばけの形をした 2 種類の駒を使って戦うドイツ発祥の二人用チェスゲームであり，背中に青い印のついた駒と赤い印のついた駒を取り合う．

各プレイヤはこれら 2 種類の駒を 4 つずつ持ち，ゲーム開始時に図に示す自陣 8 マスに自由に，相手に印の色が見えないように配置することができる．各手番においてプレイヤは駒を前後左右に 1 マス動かすことができる．自分の駒があるマスには移動できず，相手の駒がある場合には将棋と同じように相手の駒を取ることができる．相手の駒を取る時にその駒の色を確認できる．また，相手プレイヤー側にある矢印が描かれたマスは自分の出口であり，出口に「青いおばけ」が乗っている時に自分のターンに脱出させることができる．

ゲームの勝利条件は次の 3 つの条件の内 1 つを満たすことである．1 つ目は相手の青の駒を 4 つ取る事である．2 つ目は自分の赤の駒を 4 つ取らせる事である．3 つ目は自分の青の駒を相手側の出口から脱出させる事である．

2.2 関連研究

不完全情報ゲームには多様な先行研究がみられるが，本稿ではガイスターに関する物のみ述べる．Balakrishinan らはガイスターを対象とした MiniMax 木探索の提案を行った [8]．三塩らは Using Past Playout を用いた不完全情報への対処を行い，ガイスターのコンピュータプレイヤの設計および評価を行った [9][10]．Using Past Playout のアプローチは，モンテカルロ法のシミュレーションにおいて「相手の着手の勝率が高くなるような駒色の配置」を探索において重視する点が本稿の提案手法に類似している．しかし三塩らの手法では敵の駒色の配置についての推測を陽には行わない点が異なる．

川上らはガイスターから不完全情報性を除いた場合のゲームにおける，既存手法の性能を調査した [11]．Chen らは盤のサイズを縮小したルールにおける最適戦略の解析を，Counterfactual Regret Minimizaion アルゴリズムを用いて行い [12]，末続らはガイスターに特有の知識を利用したルールベース型のプレイヤを設計し，コンピュータプレイヤの大会における上位への入賞を報告した [13]．

また，不完全情報ゲームにおいて探索手法を適用するときのアプローチとして，不完全情報に対する適当な割り当てを行ってそれぞれの評価値を統計的に扱う方法はよく採られるが，川上らは各割り当てにおける評価値の平均値を扱った場合と最小値を扱った場合の比較を行った [14]．木村らは縮小されたゲームボード上で Deep Q Network を用いたプレイヤ作成を行った [15]．ゲームの詰め探索については，石田らが Df-PN アルゴリズムを利用して行った例がある [16]．また伊藤らはガイスターの「キーパー戦略」により，ある特定の状況下で相手の勝ち筋を潰せることを確かめた [17]．

ガイスターのブラフ手に関連した研究は佐藤らによるものが挙げられる [18]．この研究は強化学習を用いたプレイヤ作成の研究であるが，その学習により導出された戦略上に，ブラフを利用した着手が現れる事を確認している．さらに，強いプレイヤの作成を推進する試みとして，コンピュータプレイヤの競技会が定期的に開催されている [19]．

3. 提案手法

本研究では，相手の駒の色の推測にモンテカルロ法を利用する．本項ではモンテカルロ法およびそれを利用した駒推定の方法について述べる．

3.1 モンテカルロ法

モンテカルロ法は乱数を利用したシミュレーションを複数回繰り返し，その平均値などを参照する事で，求めたい数値の近似値を得るアプローチである．ゲームプレイヤの作成においては状態の価値や行動の価値を近似するために用いられる事が多い．

まず一般的な実装における最初の処理として，手版局面から可能な着手それぞれを行ってみた次状態を作り出す．それから各（次）状態を開始状態として両プレイヤの着手をランダムに生成してゲームを進めていき，そのシミュレーションプレイによる最終局面が先手番の勝ちとなったか負けとなったかを観察する．このシミュレーションプレ

イはプレイアウトと呼ばれ，プレイアウトによる勝ちまたは負けの数[*1]を集計する事で，各開始局面，ひいては初期局面における各可能着手の「良さ」を近似的に計算する．

この手法には多くの改造や拡張が試みられており，シミュレーション中の行動生成に確率的な偏りを持たせたり，または木探索と組み合わせた形であるモンテカルロ木探索も広く利用されている．

3.2 モンテカルロ法による不完全情報の推定

本研究では三塩らが試みたアプローチ [9][10] を参考に，「相手プレイヤの立場にたって行ったモンテカルロ法の探索結果」と「実際に相手が行った着手」との整合性を基にして，不明な駒の色の推定を行う．疑似コードを Algorithm1，2 に示す．

Algorithm 1 MakePrediction(...)

引数: last_state, last_move
返り値: piece_ID, piece_color

```
pieceID = getPieceID(last_move)
  /* 相手がさっき動かした駒の “青っぽさ” */
blue_feasible = calcEvalDV(last_state, last_move, pieceID, BLUE)
  /* “赤っぽさ” */
red_feasible = calcEvalDV(calcEvalDV(last_state, last_move, pieceID, RED)
if (blue_feasible − red_feasible ≥ THRESHOLD) then
    return {pieceID, BLUE}
else if (red_feasible − blue_feasible ≥ THRESHOLD) then
    return {pieceID, RED}
else
    return {pieceID, UNKNOWN}
end if
```

Algorithm1 の makePrediction は以下の 2 つの情報を引数にとる．

- 相手プレイヤの手番局面．
- その局面で選択した着手．

そして以下の情報をセットにして出力する．

- 引数で与えられた着手で移動した駒の色の推定値．赤，青，不明の 3 値．

仕組みとしては，後述する calcEvalDV(·) 関数によって特定の駒の「赤らしさ」と「青らしさ」を数値化して，その差が一定値を超えていれば色の予測値を赤または青として返す．色の推定が行われるのは，その着手により移動した 1 つの駒に対してのみである．1 回の着手から全ての駒の色の推定を行う設計にする事も可能ではあるが，十分な推定の精度の達成には多くの局面（特に残存する駒が多い局面）にて困難が予想されるため，今回は 1 駒のみ色の予測をする設計とした．

[*1] 実装によっては状態評価関数による局面の価値を代わりに用いる場合もある．

Algorithm 2 CalcEvalDV(...)

引数: state, move, pieceID, color_arg
返り値: deviation_value

```
 1: eval_list = double[]
 2: for i_mv = 0 to legal_moves.length do
 3:     win_count = 0
 4:     legal_move = legal_moves[i_mv]
 5:     if (legal_move==move) then
 6:         i_mv* = i_mv
 7:     end if
        /* 子局面生成 */
 8:     child_state = AddMove(state, move)
 9:     for j = 0 to 10 do
10:         /*引数 pieceID 番目の駒の色を引数の color_arg に変更*/
11:         child_state.pieces[pieceID].color = color
            /*pieceID 番目の駒を除いた敵駒の色をランダムに決定*/
12:         ShuffleColor_opponentPieces(child_state)
13:         win_count += PlayOut(child_state, 1000)
14:     end for
15:     eval_list[i_mv]=win_count
16: end for
    /*全合法手の勝率のリストの中における，
      引数 move の勝率の偏差値*/
17: DevValue = calcDV(eval_list[i_mv*], eval_list)
18: return DevValue
```

makePrediction(·) の下流にあたる関数 calcEvalDV() の概要を Algorithm2 に示す．この関数は，まずある特定の（色がまだ不明な）駒の色を赤または青と仮定した上で，「可能な着手それぞれの価値」の中から「実際に選ばれた着手の価値」の偏差値（Deviation Value）を算出する．

まず与えられた局面の全合法手をループして，それぞれの合法手を着手した後の局面を生成する．そしてその局面に含まれている，色が未知の駒たちに対して，「引数で色が指定された駒」以外の駒すべてにランダムな色を割り当てる．ただし赤の駒が 5 つ以上になるなど，ルールで許されない割り当ては除外する．そのように全ての駒に色が割り当てられたらプレイアウトを行って，勝率を集計する．

末尾の calcDV(·) 関数は，与えられた数値の集合を標本とみなして，その中から特定の値の偏差値を計算する．着手の良さを表す指標として偏差値をあえて採用する必然性はないが，標本に含まれる値の定数倍やバイアスの追加に対しても不変となる指標として，今回ひとまず採用した．

本提案手法は，ガイスターに特有の専門知識を利用しない点でルールベースド型の手法よりも適用可能な対象が広いが，専門知識を利用しない分，精度に劣ると予想される．また機械学習型の手法と比較した場合には，学習データを用意しなくてもある程度の性能が発揮できる点が優越してるが，一方で対戦相手の着手選択について「モンテカルロ法と同様の選択を行う」という，やや強めの仮定を置いているため，性能の上限については対戦相手のデータを利用した機械学習より劣る事が予想される．

4. 実験１：駒色の推定実験

提案手法の有効性を検証するため，駒色の推測精度を確かめる．

4.1 手順

本実験は以下の手順で行われる．

(1) ガイスターのランダムなゲーム状態を生成する．
(2) 3.1 項で示した原始モンテカルロ法で着手を生成する．
(3) その状態と着手を入力とし，3.2 項の推定を行う．

プログラムの実装上では盤上の全ての駒の色は既知であるが，推定の際には，駒の推定を受ける側の陣営の駒の色を未知として推定する．

4.2 ランダム状態の生成

ランダムな状況を生成するために，ゲームの開始局面から駒の色をランダムに割り振ってから，ランダムな着手を一定回数加えた．空の盤面からランダムな座標を選んで全８駒を配置するやり方も考えられるが，ゲームの進行上不自然な局面の生成を抑制できると考え，その方法を選んだ．

4.3 モンテカルロ法プレイヤ

本稿では「着手から駒の色を生成されるプレイヤ」として，モンテカルロ法のプレイヤを用いる．他の方法のプレイヤであっても，ある程度合理的に着手を選択するプレイヤであれば推定は機能すると予想されるが，最も推定がうまくいきそうなプレイヤとして今回ひとまず選択した．

本実験で用いるモンテカルロ法プレイヤは，ほとんど独自の工夫を含まない単純なモンテカルロ法を用いるが，いくつか説明するべき点がある．まず相手プレイヤの未知な色の駒の扱い方である．本稿では，ルート局面に 10 通りのランダムな色の割り当てのパターンを生成し，それぞれの割り当てからプレイアウトを開始する．各割り当てごとにプレイアウト回数と勝率の重みは平等である．

そして，プレイアウト中における行動選択は基本的に全合法手からの一様にランダムな選択であるが，そのターン中に「青いオバケを脱出できる」場合のみ必ずその行動を選ぶ．またプレイアウト中の手数がどれだけ伸びても，プレイアウトを途中で打ち切らない．

4.4 実験条件

本実験にはパラメータが２つある．１つは，ランダム局面を生成するために初期局面から加えるランダム着手の回数である．これが大きくなるほど，ゲームが終盤に近付くので，直感的には駒色の推定は易しくなると予想される．もう１つは，3.2 項の推定で用いる threshold 値である．“赤っぽさ” と “青っぽさ” の差が threshold 値以上ならば，駒の色が赤または青であると判定を行う．

各パラメータごとにランダム局面の生成と色の推定を 1,000 回繰り返した．モンテカルロ法のシミュレーション回数は 1 着手生成の 1 合法手ごとに 10,000 回とした．実験には Windows10 搭載の，メモリが 16GB，クロックが 3.7GHz で，64bit の CPU を搭載した CP を利用した．実験環境としては Java による自作環境を用いた．この条件で 1,000 回の駒色推定に約２時間程度を要した．

4.5 結果

まず，表 1 に示すように各パラメータの各局面で駒がきちんと赤または青に推定された割合を算出した．推定手法の threshold 値が 0 のとき[*2]は，100% となり，threshold 値の増加とともに割合が減少した．また推定する局面のターン数が大きくなるほど，色の推定が不明となるケースが減少する傾向があった．

表 2 には各パラメータごとの色の推定の正答率を示す．threshold 値が大きくなるほど曖昧な局面が減り，正答率が向上していると解釈できる．また，今回は 3.2 の推定アルゴリズムを用いたが，この手法では着手を生成する方法のモデルとして原始モンテカルロ法を想定している．そして実際の着手生成の手法として，同様の原始モンテカルロ法を採用した．そのため 8 割を超える高い正答率を達成できたと解釈できる．

5. 実験２：駒色推定を含むプレイヤの対戦実験

4 項の駒色の推定アルゴリズムを利用したプレイヤと利用しないプレイヤの対戦実験によって，ゲームの強さに対する提案手法の有効性を評価した．

5.1 駒色の推定つきモンテカルロ法プレイヤ

この実験では，3.2 節の駒色推定と 4.3 節のモンテカルロ法を組み合わせたプレイヤを用意した．このプレイヤは，相手プレイヤ E が着手により駒 P を動かすと，3.2 節のアルゴリズムによって駒 P の色を推定する．もし色が赤または青らしいと推定した場合（つまり推定結果が「不明」ではなかった場合）は駒 P の色を確定して，プレイヤ E の他の駒の色のみにランダムな割り当てをしてプレイアウトを行う．

この駒色の推定結果はターンをまたいで利用しない．すなわち，$(n-1)$ ターン目の相手プレイヤの着手から得られた駒色の推定結果は，直後の n ターン目の提案プレイヤのモンテカルロ法にのみ利用され，推定の履歴は破棄される．

[*2] threshold 値が 0 で，なおかつ “赤らしさ” と “青らしさ” の値が等しい場合，本稿で実装したアルゴリズムは色の判定を保留しない．コードの処理順序の都合により「駒の色が青である」という推定結果を返す．

表 1 駒色推定アルゴリズムが駒色の判定を「不明」にしなかった割合．アルゴリズム内の Threshold 値が大きな影響を持つ．また経過ターン数の大きい局面ほど色の判別が不明になりにくい事が見てとれる．

	T=0	1	2	5	10	20	30
ターン経過:0	100%	95.3%	91.6%	82.0%	65.0%	33.7%	9.3%
10	100%	92.0%	86.7%	73.0%	55.0%	26.3%	9.6%
20	100%	92.2%	86.1%	71.7%	50.9%	24.6%	9.1%
40	100%	89.3%	84.7%	69.4%	49.0%	24.7%	10.1%
60	100%	89.9%	84.7%	71.5%	52.7%	29.4%	13.1%
100	100%	90.1%	86.0%	72.6%	57.1%	31.7%	14.9%
150	100%	87.4%	83.0%	71.9%	57.0%	31.7%	13.4%
200	100%	89.4%	85.9%	75.3%	61.6%	37.7%	14.2%

表 2 駒色推定アルゴリズムが駒色の判定を正答した割合．推定結果が「不明」を返したケースは除外し，残ったケースの中での正答の割合である．アルゴリズムの Threshold 値が大きくなる程，経過ターン数の大きい局面を扱うほど，推定の精度が向上している．

	T=0	1	2	5	10	20	30
ターン経過:0	80.8%	81.7%	82.2%	83.8%	85.2%	89.3%	87.1%
10	76.2%	78.2%	79.7%	83.2%	89.1%	95.1%	97.9%
20	72.1%	73.5%	73.9%	77.1%	84.3%	92.7%	93.4%
40	72.9%	75.5%	76.4%	69.4%	88.8%	93.1%	95.0%
60	73.2%	75.3%	76.5%	71.5%	85.4%	91.5%	95.4%
100	77.9%	81.2%	81.0%	72.6%	90.9%	94.3%	98.0%
150	78.7%	81.5%	83.3%	71.9%	90.9%	96.2%	97.8%
200	83.9%	87.7%	85.8%	75.3%	92.4%	95.8%	98.6%

5.2 実験条件

5.1 節で示した提案プレイヤと 4.3 節で示したナイーブなモンテカルロ法プレイヤの対戦を行った．パラメータは駒色推定のアルゴリズムが用いる threshold 値であり，値が 0，10，30 の場合それぞれ 500 試合ずつ対戦した．

ナイーブなモンテカルロ法プレイヤの 1 ターンの着手生成あたりのプレイアウト回数は 6,000 回，提案手法プレイヤのプレイアウト回数は 2,000 回とした．また，提案プレイヤは相手の駒の “赤らしさ” の推定 1 回のためにプレイアウトを 2,000 回，“青らしさ” の推定 1 回のためにプレイアウトを 2,000 回行うようにした．このため両プレイヤが 1 ターンごとに行うプレイアウトの総数，および着手の生成時間はほぼ等しくなる．実験に利用した計算機は 4.4 節で述べたものと同じで，対戦 500 ごとに約 2 時間を費やした．

表 3 駒色推定アルゴリズムつきプレイヤと通常のモンテカルロ法プレイヤの 500 戦対戦結果．

T=0	10	30
56.0%	56.6%	47.6%

5.3 結果

対戦の結果を表 3 に示す．対戦 500 回における勝率 56% および 56.6% は，この場合の 95% 信頼区間の (±4%) の幅を考慮しても，有意な性能向上であると結論できる．Threshold が 30 の時のみ勝率が有意に向上しなかったが，threshold 値の増加によって色の推定が「不明」になる事が増えたためと思われる．この場合に提案手法プレイヤは，敵の駒の色に対する情報が利用できないばかりでなく，色の推定アルゴリズムのために消費した計算時間が無駄となる．

6. ブラフ手生成の可能性の検証

ポーカー等のナッシュ均衡による解析が可能なゲームでは，最適戦略を求めれば，その戦略の実行によって自然とブラフが実行されうる．また，履歴のデータを用いて対戦相手をモデル化する事によりブラフに引っかけるアプローチも，ポーカーの先行研究に例がある [20]．

しかし，フルサイズのゲームにおける解析がまだ行われていないガイスターにおいても，提案手法を以下の手順で応用すれば，履歴のデータを用いずに，人間プレイヤが行うような「ブラフ手」の生成も可能であると考える．

(1) 現在状態 s_t に対し，合法手 $m_i(\in M_t)$ を着手した次局面を $s'_{t,i}$ とおく．
- ただし M_t は全合法手の集合とする．

(2) 5.1 節のプレイヤに，局面 $s'_{t,i}$，着手 m_i，直前の局面 s_t を与えて，返す着手 m^o_j を観察する．

(3) 局面 $s'_{t,i}$ に着手 m^o_j を行った局面を $s''_{t,i}$ とおく．

(4) 全合法手集合 M_t から各合法手 $m_i(\in M_t)$ で導かれる 2 ターン後局面の $s''_{t,i}$ を何らかの状態評価関数で評価

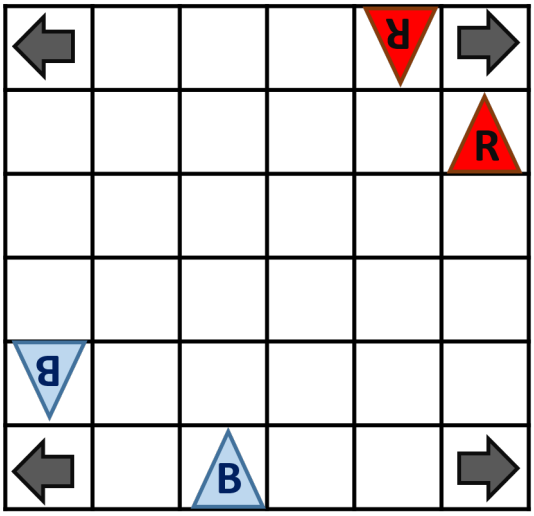

図 2　ガイスターのブラフ手生成の試み．下手側のプレイヤが赤を前進させて脱出口に置く手を生成した．これにより，上手側は「脱出口に前進するのは青に違いない」と判断し，下手側の赤の駒を取ってしまう事で敗北する．

する．

(5) 最も評価値の高い局面を導く合法手を着手として生成する．

上記の方法でブラフ手の生成が観察できた例を紹介する．図 2 の局面で上述の手順を行った結果，下手側から赤の駒を 1 マス前進させる着手が生成された．この着手は，相手プレイヤ（上手側）が「今動かした駒は青に違いない」と誤解すれば，上手側の赤の駒でそれを取ってしまう（その結果，下手側の赤の駒を全て取ってしまって負けになる）ような手である．そのため，人間の目から見てもこれはある程度自然なブラフの手であると言えなくもない．

もちろんこの着手生成が最善である保証は無く，まず対戦相手が 5.1 節の提案プレイヤと “結果として” 同様の着手を選択するようなプレイヤでなければブラフには引っかからない．そして，ブラフに失敗すればかえって不利な状況を招く可能性がある．更に，不完全情報ゲームの性質として，純粋ではない「混合戦略」が最適となる局面も多いはずであるが，このブラフ手の生成手順は混合戦略になりにくい．よって本アプローチの自然な改良によって，ゲーム理論的に最適なプレイヤは完成しにくいと考えられる．

しかし，本手順はゲーム固有の専門知識を陽に組み込む必要もなく，ナッシュ均衡の導出やログの学習も必要とせずに，人間のようなブラフの手を生成できる見込みがある点が長所であると考える．

7. まとめ

本稿では，不完全情報ゲームのガイスターにおいて「相手の視点に立った原始モンテカルロ法の実施」と「実際の着手の観察」から，駒の色を推定する方法を提案した．またこの推定手法の精度を評価するために，ランダム局面からのモンテカルロ法プレイヤの着手を利用し，駒色の推定実験を行った．その結果，多くのパラメータの条件で 80% を超える正答率を達成した．更にこの推定による情報を利用したプレイヤを設計し，思考時間を揃えたモンテカルロ法プレイヤとの対戦を行った結果，有意に性能の向上を確認できた．

今後の課題としては，比較的直近のものとしては以下のものが挙げられる．

- 原始モンテカルロ法プレイヤ以外の生成する着手に対しても本稿の駒色の推定がうまく機能するかの検証．
- モンテカルロ法をモンテカルロ木探索に拡張した場合の性能評価．
- プレイアウト中の方策の改良の検討．
- ブラフ手生成を組み込んだプレイヤの対戦の強さの評価．
- ブラフ手生成を組み込んだプレイヤが人間プレイヤに与える「人間らしさ」等の印象の調査．

またガイスターで十分に知見が蓄積した後には，ガイスター以外の不完全情報ゲームにも同じような推定やブラフのアルゴリズムを適用してみる事を検討している．

謝辞

本研究は JSPS 科研費（研究課題 19K20429）の助成を受けたものである．

参考文献

[1] Murray Campbell, A. Joseph Hoane Jr. and Feng-hsiung Hsu. “Deep blue.” Artificial intelligence, Vol.134, No.1, pp.57-83, 2002.
[2] Kunihito Hoki and Tomoyuki Kaneko. “Large-Scale Optimization for Evaluation Functions with Minimax Search.” Journal of Artificial Intelligence Research, Vol.49, pp.527-568, 2014.
[3] David Silver, Thomas Hubert, et al. “Mastering chess and shogi by self-play with a general reinforcement learning algorithm.” arXiv preprint arXiv:1712.01815 (2017).
[4] David Silver, Aja Huang, et al. “Mastering the game of Go with deep neural networks and tree search.” Nature, Vol.529, No.7587, pp.484-489, 2016.
[5] Gerald Tesauro. “TD-Gammon, a self-teaching backgammon program, achieves master-level play.” Neural computation 6.2. pp.215-219, 1994.
[6] Naoki Mizukami and Yoshimasa Tsuruoka. “Building a computer mahjong player based on monte carlo simulation and opponent models.” IEEE Conference on Computational Intelligence and Games (CIG), pp. 275-283, 2015.
[7] 片上 大輔, 鳥海 不二夫, 他. “人狼知能プロジェクト (¡ 特集¿ エンターテイメントにおける AI).” 人工知能, 30(1), pp.65-73, 2014.
[8] S. Balakrishnan, K. L. Shunmuganathan, and Raja Sreenevasan. “Amelioration of Artificial Intelligence using Game Techniques for an Imperfect Information

Board Game Geister." International Journal of Applied Engineering Research ISSN 0973-4562 Vol.9(22), pp.11849-11860, 2014.
[9] 三塩 武徳, 小谷 善行. "ゲームの不完全情報推定アルゴリズム UPP とそのガイスターへの応用." ゲーム情報学研究報告 31(4), pp.1-6, 2014.
[10] 三塩 武徳, 藤田 桂英. "UPP による駒価値評価関数に基づいた NEGOgeister AI." ゲーム情報学研究報告 33(7), pp.1-6, 2015.
[11] 川上 直人, 橋本 剛. "完全情報ゲームの探索を用いたガイスター AI の研究." ゲームプログラミングワークショップ 2018 論文集, pp.35-42, 2018.
[12] Chen Chen and Tomoyuki Kaneko. "Counterfactual regret minimization for the board game geister." Game Programming Workshop 2018 Proceedings, pp.137-144, 2018.
[13] 末續 鴻輝, 織田 祐輔. "機械学習を用いないガイスターの行動アルゴリズム開発." Game AI Tournament 2018 論文集, pp.13-16, 2018.
[14] 川上 直人, 橋本 剛. "ガイスター AI の研究." ゲーム情報学研究報告 39(6), pp.1-6, 2018.
[15] 木村 勇太, 伊藤 毅志. "深層強化学習を用いたガイスター AI の構築." ゲームプログラミングワークショップ 2019 論文集, pp.130-135, 2019.
[16] 石井 岳史, 川上 直人, 他. "難しい詰めガイスター問題の生成法." ゲームプログラミングワークショップ 2019 論文集, pp.12-19, 2019.
[17] 伊藤雅士, 大久保壮浩, 他. "ガイスター AI のキーパー戦略の有効性." ゲーム情報学研究報告 42(3), pp.1-7, 2019.
[18] 佐藤佑史. "ガイスターにおける自己対戦による行動価値関数の学習." 平成 27 年度 電気通信大学 修士論文, 2015.
[19] 伊藤毅志, 篠田孝祐, 他. "第 2 回 GAT (Game AI Tournament) 報告." 情報処理学会研究報告 38(2), pp.1-8, 2017
[20] Finnegan Southey, Michael Bowling, et al. "Bayes' Bluff: Opponent Modelling in Poker." arXiv preprint arXiv:1207.1411, 2012.