Vol.2021-GI-46 No.16
2021/6/20

自己対戦を用いたガイスターAI における行動優先度の学習

志賀薫[1]　伊藤毅志[1]

概要：本研究は，GAT2021 のガイスターAI 大会において優勝した機械学習を用いたゲーム AI のアルゴリズムを紹介する．本手法の有効性を示すために，学習回数を重ねるごとに自己対戦のみならず，他プログラムに対しても強くなる傾向があることを対戦実験を通じて確認した．

キーワード：行動優先度，自己対戦，強化学習，方策勾配法

Learning of Action Preferences in Geister AI Using Self-Play

KAORU SHIGA[†1]　TAKESHI ITO[†2]

Abstract: This report presents an algorithm for game AI using machine learning that won the Geister AI competition at GAT2021. To demonstrate the effectiveness of this program method, we confirmed through competitive experiments that the algorithm tend s to become stronger not only against itself but also against other programs as it learns more and more.

Keywords: action preferences, self-play, reinforcement learning, policy gradient method

1. はじめに

ガイスターは，不完全情報確定ゼロ和ゲームに分類されるゲームであり，近年ゲーム情報学の分野でも興味を集める研究テーマになっている．筆者は，2020 年度に電気通信大学の情報工学工房において，ガイスターAI を作る課題に取り組んできた．その成果として，2021 年 3 月 7 日に開催された GAT(Game AI Tournament)において，優勝することができた．本報告では，プログラムのアルゴリズムを紹介するとともに，本手法の有効性を示す簡単な実験を行ったので，その実験結果を紹介する．

情報工学工房は，電気通信大学の I 類（情報系）の１～４年生を対象にした科目であり,「プログラミングを修行する工房」という形態を通してものづくりの面白さを理解し，ソフトウェア制作の実践力を身につけることを目的としている．複数の教員がそれぞれテーマを持ち寄り，学生はその中から希望するテーマを選択して受講する．2020 年度は，12 のテーマが掲げられ，第一著者の志賀は，第二著者の伊藤が提案した「不完全情報ゲーム AI を作ろう！」というテーマに応募して受講した．

工房では，最初はガイスターのルールを学ぶところから始まり，ゲーム AI の基礎的な考え方を学び，その後，サンプルコードが配られて，各自自由に改良を進めることとなった．

2 章の関連研究も紹介されたが，ガイスターの AI においては，機械学習の成功例があまり見られなかったので，本研究では，あえて機械学習に挑戦してみることにした．

2. 関連研究

2.1 ガイスターに関する研究

ガイスターAI を強くする研究は近年行われている．研究の方向性を見ると 3 つの方向性に大別される．1 つ目は，ゲーム木探索を用いた手法，2 つ目はルールベースによる手法，3 つ目は機械学習を用いる手法である．それぞれ順に見ていこう．

2.1.1 ゲーム木探索を用いる手法

川上らはゲーム木探索を用いたガイスターAI の開発を試みている[1]．ゲーム木探索は完全情報ゲームであることを想定した手法であるので，そのままではガイスターには使えない．そこで，可能性のあるすべての状態それぞれを根として持つようなゲーム木を作成し，それぞれの探索結果を集計することを試みている．集計方法として，以下の 2 つの手法を試みている．

① ゲーム木ごとに最高の評価値を取る着手を求め，多数決で決める
② すべてのゲーム木の中で最小評価値が最大となる着手を選ぶ

これら 2 つの手法を直接戦わせた場合に有意な差は見られなかったものの，それ以外の単純な戦略に対する AI に対する勝率で②の方が有力であると結論づけている．

また川上らは別の論文で，以下の 2 つの手法も検討して

1 電気通信大学
The University of Electro-Communications

いる[2].

③　すべてのゲーム木の評価値の平均が最大となる着手

④　相手の未判明の駒を紫駒としたゲーム木探索

紫駒は，佐藤が提案したガイスターを完全情報ゲームに変換する手法で，脱出するときには青駒と同じとみなし，取ると赤駒と判定されるような敵の駒と定義される [3]. すなわち，最悪の状況を想定する手法である.

前述の①～④を比較して，②と④が強く，特に④は探索時間を大幅に削減できることを示した．また，評価関数についても検討しており，青駒の残り個数と出口マスまでの距離までのどちらを重視すべきかについて比較している．その結果，青駒の残り個数を重視すべきであると結論づけている.

2.1.2 ルールベースによる手法

末續らはルールベースによるガイスターAI の構築を試みている[4]．初期配置のルールから相手の駒の推定，駒の動きなど，数多くのルールを記述することで強い AI を実現している．実際 AI 同士の大会で複数回優勝しており，有力な手法の一つであると言える.

2.1.3 機械学習を用いる手法

佐藤らは，自己対戦により行動価値関数を学習することで強いガイスターAI の構築を試みている[3]．この研究では，行動価値関数の近似として自己対戦によるニューラルネットワークの学習を行っている．しかし，学習したネットワークを用いたプレイヤは，ランダムプレイヤにすら勝ち越すことができず，それだけでは十分な強さは得られなかった.

また，木村ら[5]は Alpha Zero のアルゴリズムをガイスターに適用する手法を試みた．相手の駒の色が互いにわかっているという完全情報ガイスターを定義し，これに Alpha Zero アルゴリズムを適用しネットワークの学習を行った．実際のガイスターへの適用では，ルールベースによる駒色の推定を行い，推定値を用いて局面を完全情報ガイスターへと変換してゲーム木探索と組み合わせている．これは，機械学習だけでは十分な強さを実現できなかったことが理由であった.

2.2 強化学習の研究

強化学習がボードゲーム AI において有用であることを示す先行事例として AlphaGo が挙げられる．AlphaGo は深層強化学習を用いた囲碁 AI である[6]．AlphaGo では，インターネット囲碁道場 KGS の 6 段～9 段のレベルのプレイヤの 16 万局にも及ぶ棋譜を教師データとして深層ニューラルネットワーク（DCNN）を用いて教師あり学習を行い，ポリシーネットワークを構築した．さらに，そのポリシーネットワークを初期値として DCNN を用いて自己対戦により強化学習を行い，より強力なポリシーネットワークを構築した．さらにそのポリシーネットワークを用いて，局面の勝率を計算する DCNN であるバリューネットワークを作成することで，当時のトッププロ棋士を破るほどの棋力を実現した.

囲碁のような完全情報ゲームにおける強化学習の成功例は多くあるが，不完全情報ゲームにおいては必ずしもうまく行っていない．例えば Minecraft を用いた研究では，DQN という強化学習手法を用いた結果 3 項目中 2 項目でランダムプレイヤ以下の性能に留まった[7]. また, Doom というゲームでも AI が右往左往するだけでほとんどその場から動かないという報告がなされている[8]. いずれのゲームも 3D の一人称視点ゲームであり，障害物の向こう側や背後の状況を把握できないという不完全情報ゲームである．ガイスターにおける DNN を用いた強化学習の先行研究でもあまり成功しているとは言えない.

3. 提案手法

提案手法を 2 段階に分けて説明する．第一段階は，行動優先度の説明と計算方法で，第二段階は行動優先度の使い方とパラメータの学習についてである.

ガイスターの対戦では，まず駒の初期配置を決める必要がある．本報告のアルゴリズムでは，あらゆる初期配置の組み合わせの中からランダムに選ぶ手法を取っている．初期配置が決まれば，ある局面から次の一手を選ぶ手法について検討するだけで良い．以下では，その手法についてのみ議論する.

3.1 行動優先度の計算方法

ある局面から次の一手を選ぶために，行動優先度という指標を考える．現在の局面の合法手を列挙して，それぞれについて行動優先度(action preference)を以下のように求める．行動優先度は指し手の良さを数値化したものである．行動優先度の計算は以下の三つの段階から成る．

(1) **次の一手の二値ベクトル表現 s を求める**.

(2) **s から二駒関係 x を求める**.

(3) **x と重み θ の線形和から行動優先度を求める**.

以下では，(1)～(3)についてそれぞれ説明する．なお，重み θ については 3.2 で詳述する.

(1)　**次の一手のベクトル表現 s**

ここでは，次の一手を二値化して表現する手法について述べる.

A)　次の一手を指した後の駒の有無
- 自分の青駒の位置[6*6]
- 自分の赤駒の位置[6*6]
- 敵の駒の位置[6*6]

B)　次の一手を指す前の取られた駒の数
- 自分の青駒の取られた数[3] <one-hot 表現>
- 自分の赤駒の取られた数[3] <one-hot 表現>
- 相手の青駒の取られた数[3] <one-hot 表現>

- 相手の赤駒の取られた数[3] <one-hot 表現>

C) 特殊な駒の移動

- 駒を取るかどうか[1] 取る 1 取らない 0
- 青駒が脱出するかどうか[1] 脱出 1 脱出しない 0

*上述の[ ]内の数値は表現に必要なベクトルの要素数を表している．例えば，「自分の青駒の位置(6*6)」は，6*6=36 個のベクトルで表現されることを意味する．上述の[ ]内の数値をすべて合計すると 122 となり，次の一手は 122 個の数値ベクトルで表現されることがわかる．*

A)は次の一手を指した後の駒の位置を表現している．表現すべき駒の種類は自分の青駒と赤駒，相手の駒の 3 種類でそれぞれの位置は, 6*6 のマス上に存在するか否かを 0,1 で示す．B)は次の一手を指す前の取られた駒の数を表現していて，one-hot 表現を用いることで，取られた駒の数を表現する．例えば，自分の青駒の取られた数が 0 個,1 個,2 個,3 個のとき，それぞれ「0 0 0」「1 0 0」「0 1 0」「0 0 1」と表現される．表現すべき駒の種類は，自分と相手の青駒と赤駒の 4 種類であるので，3*4 個のベクトルで表現できる．C)は特殊な指し手（相手の駒を取るか否か，青駒を脱出するか否か）を表現している． s は事後状態と呼ばれている．

(2) 二駒関係からの特徴量 x の抽出

ここで，2 つの状態 s の連言を用いて，その特徴量 x を表現する．例えば x のある要素が 1 のとき，それは例えば『「自分の青駒が盤の左上隅に存在」かつ「敵の駒が盤の左下隅に存在する」』を意味したり，『「自分の青駒が盤の左上隅に存在」かつ「相手の駒を取る」』を意味したり，『「自分の青駒の取られた数が 3」かつ「相手の赤駒の取られた数が 0」』などを表現できる．

これに似た表現として将棋の二駒関係もあるが，ガイスターは不完全情報ゲームであるので駒の位置関係以外の要素の組み合わせも考慮する必要があるため，ここでは二駒関係と呼ぶことにする．また, 機械学習における polynomial も類似の手法であるが，やや異なっており本手法のほうが x ベクトルのサイズを小さくすることができると考えている．

二駒関係の具体的なアルゴリズムは次のようになる．

```
s ベクトルの大きさを n とする(n=122)
x を空の配列とする
Loop : i=0, …, n
    Loop : j=0, …, i
        x に要素 s_i s_j を追加する
    End Loop
End Loop
x をベクトルに変換する
```

s の i 番目の要素と s の j 番目の要素の連言を考えたものを x の要素とする．x ベクトルの大きさは n(n+1)/2=7503 となることを図 1 とアルゴリズムを比較して確認してほしい．

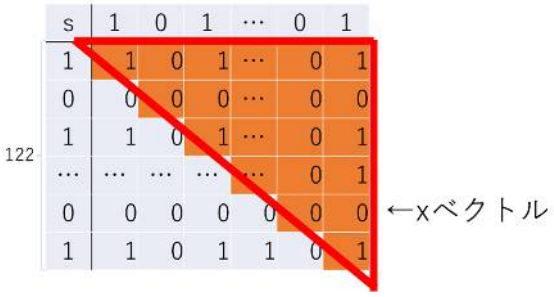

図 1 二駒関係の図式化

Figure 1 Diagrammatic representation of binary relationships

(3) 線形和と行動優先度

行動優先度 h は，特徴量 x，パラメータ θ を用い，

$$h = \Sigma_i \theta_i x_i = \theta \cdot x$$

と表現される．θ は重みとも呼ばれている． θ の値は，3.2 で説明する学習により求められる．

3.2 重みの学習

重み θ の学習は，自己対戦と θ の更新を指定回数繰り返すことで行われる．自己対戦で自分自身を対戦相手として対戦を行い，その勝敗結果を用いて θ の更新を行う．このような自分自身を対戦相手とした対局のシミュレーションを自己対戦(self-play)と呼ぶ．

3.2.1 学習中の指し手選択方法

自己対戦は，初期配置をランダムに決定するところから始まり，勝敗が決まるか 200 ターンで引き分けとなるまで対戦を続ける．自己対戦を行っているときの指し手は行動優先度をもとに計算した確率（行動確率）に従って選ばれる．i 番目の指し手の事後状態をs_i, s_iの行動優先度を$h(s_i, \theta)$としたとき，i 番目の指し手を選ぶ確率(行動確率)を$\pi(s_i, \theta)$は次の式で表される．

$$\pi(s_i, \theta) = \frac{exp\left(h(s_i, \theta)\right)}{\Sigma_j exp\left(h(s_j, \theta)\right)}$$

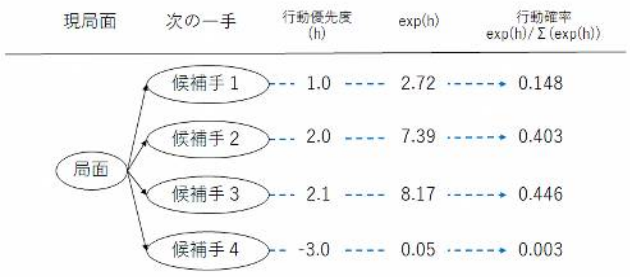

図 2 行動優先度から行動確率の求め方

Figure 2 Calculation of action probability from action priority

3.2.2 θ の更新

このように行動確率の関数を考え，その結果から直接パラメータの更新を行う手法は方策勾配法(policy gradient method)と呼ばれている．方策勾配法は θ の更新方法により複数の手法が知られているが，今回は REINFORCE with baseline と呼ばれる方法を用い，以下の式で θ を更新する[9].

ただし，s_tは先手が t 回目に選んだ指し手の事後状態，α は学習率を示すハイパーパラメータ，$b(s'_t)$は先手が手を選ぶ前の勝率の推定値である．R の値は，先手が勝利したとき R=1,負けたとき R=-1,引き分けのとき R=0 となる．各 t について更新を行うので，一試合で複数回の更新が行われることに注意してほしい．

$$\theta \leftarrow \theta + \alpha\big(R - b(s'_t)\big)\frac{\nabla\pi(s_t,\theta)}{\pi(s_t,\theta)}$$

$\nabla\pi(s_t,\theta)$は$\pi(s_t,\theta)$のθに関する偏微分であり，次の式が成り立つ．ただし，x_tは t 番目に選択した手の事後状態で，$x_{t,j}$は t 番目に選択可能だった手のうち j 番目の事後状態を表している．

$$\frac{\nabla\pi(s_t,\theta)}{\pi(s_t,\theta)} = x_t - \sum_j \pi(x_{t,j},\theta)x_{t,j}$$

θ の学習のアルゴリズムは以下のようにまとめられる．

θを 0 ベクトルで初期化
Loop forever:
　slist をから配列で初期化
　先手と後手の赤駒の配置を決定
　For each t=0, 1, … until 勝敗決定
　　確率$\pi(s_i,\theta)$に従って i を選択
　　s_iを slist に追加
　　i 番目の指し手を指す
　End
　R ← 勝敗結果(勝: 1,負: -1,引き分け:0)
　for each s in slist:
　　$b(s')$ ← sを選択する直前の勝率の推定値
　　$\theta \leftarrow \theta + \alpha\big(R - b(s')\big)\nabla\pi(s,\theta)/\pi(s,\theta)$
　関数 b(s')の更新

3.2.3 学習後の指し手選択

θ の学習後の指し手の選択は学習中とは異なり，行動優先度が最も高くなる手を確定的に選択することにする．これは，学習中とは異なり，局面のバリエーションを考慮する必要が無いからである．

学習後の指し手の選択を図に表したものが，図 3 である．図 2 と同じ局面では，学習後は行動優先度の高い候補手 3 が確定的に選択される．

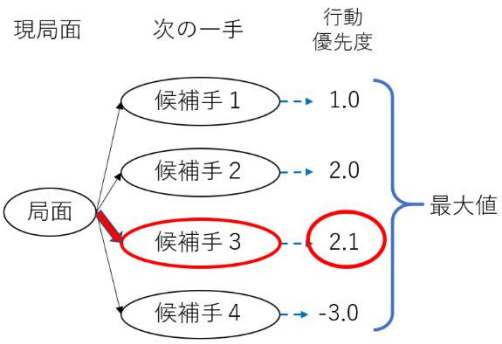

図 3　学習後の指し手選択
Figure 3　Select a next move after learning

4. 実験

4.1 目的と方法

提案手法の学習の効果を確かめるために，学習回数によってどの程度強くなるのかを調べることにした．

具体的には，以下の 3 つの実験を行った．

実験 1）提案手法で，学習回数が増えることによって，勝率が増えるかどうかを確認する実験を行った．提案手法において学習回数を 100 万回にしたプログラムをベースに，0 回，20 万回，40 万回，60 万回，80 万回，100 万回，300 万回，500 万回，700 万回，900 万回学習したプログラムとの対戦した．

実験 2）提案手法が，自分以外のプログラムに対して学習回数が増えることによってどの程度の効果があるのかを比較する実験を行った．具体的な対戦相手として，GAT2021 準優勝ソフト「Naotii2021」(川上直人氏)を用いて， 0 回，100 万回，200 万回，300 万回，400 万回，500 万回，600 万回，700 万回，800 万回，900 万回学習させたプログラムと各 1 万回の対戦を行った．

実験 3）実験 2 と同様に，GAT2021 3 位のソフト「selfest」(伊藤篤志氏)と 0 回，30 万回，100 万回，400 万回，900 万回学習させたプログラムとの各 1 万回の対戦．

ただし，selfest との対戦では大会出場時の思考時間だと一試合に相応の時間がかかり，十分な実力差の評価がするための対戦試合数を確保することが困難であるために，約 1/20 の思考時間に減らしている．そのため，大会時よりも弱くなっていることが想定されるが，本実験では，他プログラムに対する学習の効果を調べることが目的であるため，この条件で実行した．

4.2 実験結果

4.2.1 実験 1

本手法で 100 万回学習したプログラムに対して，学習回数を変えたプログラムに対する勝率と引分率をグラフにしたものが図 4 である．横軸は万回を表し，縦軸は率を表す．なお，勝率の計算方法は，勝率=勝ち総数/（勝ち数＋負け

数）とし，引分率は，引き分け総数/総試合数で計算した．すべての対戦の先手後手の勝ち数, 負け数, 引き分け数は，付録 1 に掲載しているので参照されたい．

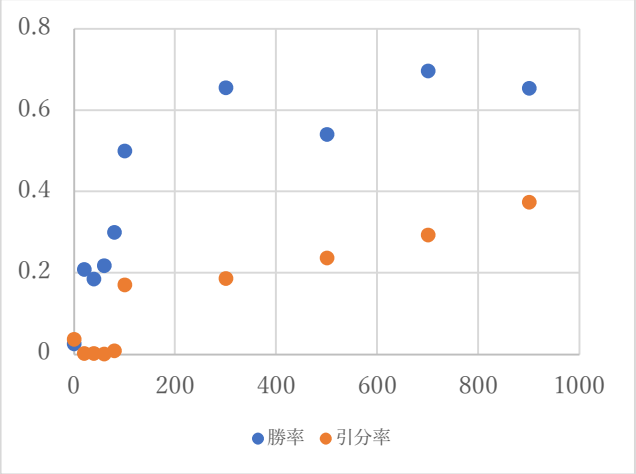

図 4　学習回数を変えた時の 100 万回学習したものに対する勝率及び引分率

Figure 4　Win and draw rate against 1,000,000 training by different training times

このグラフを見ると，100 万回以前では学習回数が増えるにつれて，概ね右肩上がりのグラフとなった．100 万回を超えると，500 万回の時に落ち込みが見られるが，100 万回よりも強くなっていることがわかる．700 万回，900 万回では，引き分けの数が増えており，勝率は 7 割程度で頭打ちになっている．

4.2.2 実験 2

続いて，他プログラムに対する学習回数の効果を調べるために，GAT 杯で準優勝であった Naotti を対戦相手とした実験を行った．学習回数を 100 万回単位で増やしたものと Naotti を対戦させて，勝敗と引き分け数を調べた．

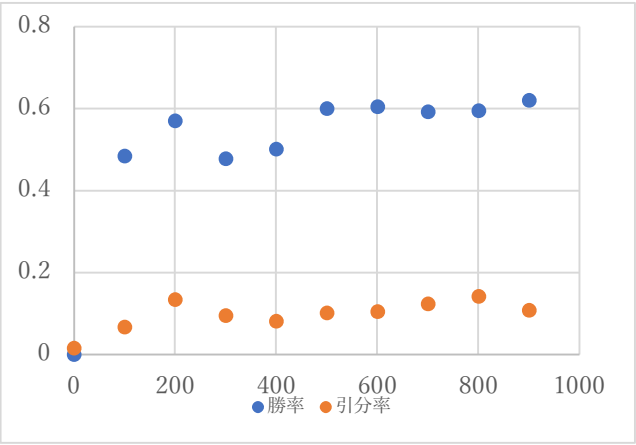

図 5　学習回数を変えたときの Naotti2021 に対する勝率及び引分率

Figure 5　Win and draw rates against Naotti2021 by different number of training times

図 5 はその結果をグラフにしたものである．横軸の単位は万回，縦軸は率を表す．すべての対戦の勝敗表は，付録 2 に掲載しているので参照されたい．

これを見ると，100 万回，300 万回，400 万回のときに落ち込んではいるものの，概ね 500 万回以上で約 6 割程度の勝率になっていることがわかる．500 万回以上では，学習回数を増やしてもあまり勝率は伸びなかった．

4.2.3 実験 3

GAT 杯で 3 位となった Selfest に対する学習回数の効果を調べた結果を示す. Selfest は思考に時間がかかるために，大会バージョンよりも 1 手あたり概ね 1/20 程度の思考時間のプログラムを用意した．用意したプログラムとの対戦結果をグラフにしたものが，図 6 である．すべての対戦の勝敗表は，付録 3 に掲載されているので参照されたい．

Selfest に対しては，30 万回以上で勝ち越しており，その後，対戦回数が増えるに伴って微増する傾向が見られた．また，対戦回数は少ないが，Selfest に対しては，30 万回以上で引き分けにはならなかった．

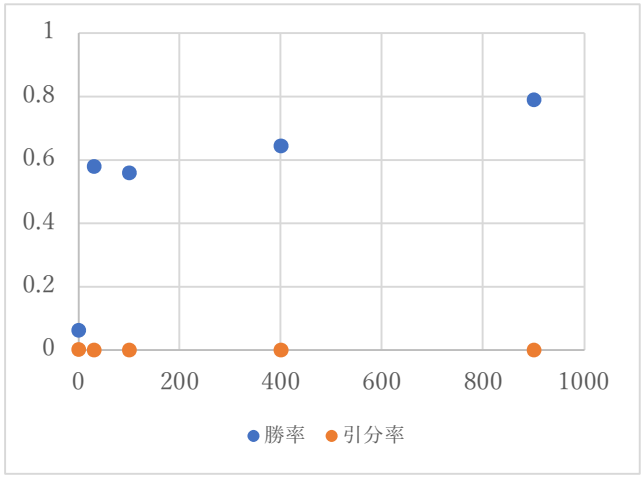

図 6　学習回数を変えたときの selfest に対する勝率及び引分率

Figure 6　Win and draw rates against selfest by different number of training times

4.3 考察

実験 1 から，概ね学習回数が増えるにしたがって強くなっていく傾向があるが，100 万回学習のプログラムに対して，それ以上学習しても劇的な勝率の上昇は見られず，900 万回学習しても 7 割程度の勝率にしかならなかった．これは，この手法における学習の限界を表しているのかも知れない．また，学習が進むにつれて，引分率が上昇する傾向が見られた．さらに詳しく調べるために，学習回数を 100 万回～900 万回に増やしたもの同士を自己対戦させたときの引分率を調べてみたものが図 7 である．

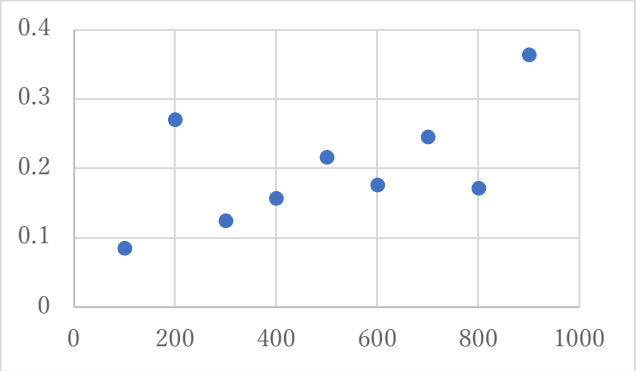

図 7　学習回数を変えたときの自己対戦の引分率
Figure 7　Draw rate in self-competition against training times

図 7 を見ると，上下を繰り返しながらやや引き分け率は上昇しているように見える．この原因は，よくわからないが，学習が進むにつれて複数の戦略が盛衰するために起こっている可能性が考えられる．今後，棋譜の詳細をしらべて原因を明らかにしていきたい．

実験２，３の結果から，自身のプログラムだけでなく，他プログラムに対しても学習回数によって勝率が増加傾向にあることが見られたが，対 Naotti2021 については，６割を大きく超えるような勝率には至らなかった．500 万回を超えると学習回数を増やす効果があまり期待できないのかも知れない．

自己対戦による学習であるために，学習の結果局所解に陥っている可能性も否定できない．対戦相手が極端なプレイをするプレイヤに対する有効性はこの実験では明確ではわからない．猪突戦法（青駒が出口に向かってくる戦法）やキーパー戦略（出口をガッチリ固めて青駒を出口から出させないようにプレイする戦略）などの極端なプレイヤとの対戦を行うことで，本手法の有効性をさらに検証する必要があるだろう．

5.　おわりに

本研究では，行動優先度という概念を導入し，強化学習の手法を組み合わせることで, 従来のものよりも強い AI を作ることが可能であることを示した．最後に，今後改善できる点を三つ提示する．

一つ目は特定の相手以外にも対応できるような手法を用いることである．具体的には自己対戦において過去の自分と対戦することで，自分とは違う方策の相手に対応できると予想される．例えば 100 万回学習した時点において，一定確率で 50 万回や 70 万回学習のプログラムと対戦すれば様々な相手を想定できる．

二つ目はゲーム木探索やモンテカルロ木探索などの探索手法を組み合わせることである．この場合は相手の駒の色の確率を予測する必要がある．

三つ目は行動優先度の関数を改良することである．今回使った二駒関係をより複雑な関数である三駒関係やニューラルネットワークに変えることで改良できるかもしれない．こうした手法を試すことで，さらに AI を強化していきたい．

謝辞

本研究の実験のために，快くプログラムの提供をしていただいた川上直人さんと伊藤篤志さんにはこの場を借りて深く感謝申し上げます．

参考文献

[1] 川上直人，橋本剛，ガイスターAI の研究，情報処理学会研究報告，Vol.2018-GI-39 No.6 (2018).
[2] 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).
[3] 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，電気通信大学修士論文(2016).
[4] 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，GAT2018 論文集,pp.13-16 (2018).
[5] 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).
[6] Silver, D.,et.al.: Mastering the game of go with deep neural networks and tree search, Nature, Vol. 529, pp. 445-446 (2016).
[7] William H. Guess, et al.; MineRL: A Large-Scale Dataset of Minecraft Demonstrations, arXiv:1907.13440vl [cs.LG] (2019).
[8] Michal Kenpka, et al.: ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning, arXiv: 1605.02097v2 [cs.LG] (2016).
[9] Richard S. Sutton and Andrew G. Barto, “Reinforcement Learning, second edition: An Introduction”, Bradford Books, (2018).

付録

１．100 万回学習に対するそれぞれの学習回数における勝敗引分数

	先後	勝	負	分
0回	先	0	4930	70
	後	0	4912	88
100万回	先	2239	2421	340
	後	2279	2385	336
200万回	先	2485	1842	673
	後	2485	1888	681
300万回	先	2142	2418	440
	後	2187	2297	516
400万回	先	2367	2238	395
	後	2232	2342	426
500万回	先	2683	1775	542
	後	2711	1818	471
600万回	先	2709	1747	544
	後	2696	1795	509
700万回	先	2571	1762	667
	後	2617	1811	572
800万回	先	2497	1757	746
	後	2616	1706	678
900万回	先	2797	1625	578
	後	2741	1760	499

２．Naotti2021 に対する学習回数に対する勝敗引き分け数

	先後	勝	負	分
0回	先	4718	106	176
	後	4669	138	193
20万回	先	3919	1073	8
	後	3986	1001	13
40万回	先	4019	969	12
	後	4115	877	8
60万回	先	3853	1145	2
	後	3965	1033	2
80万回	先	3449	1502	49
	後	3481	1474	45
100万回	先	4501	4647	852
	後	4647	4501	852
300万回	先	3127	5945	928
	後	3106	5949	945
500万回	先	3864	4907	1229
	後	4223	4630	1147
700万回	先	2547	5985	1468
	後	2634	5898	1468
900万回	先	2751	5342	1907
	後	2879	5291	1830

３．Selfest に対する学習回数に対する勝敗引き分け数

	先後	勝	負	分
0回	先	6	81	13
	後	5	82	13
30万回	先	59	41	0
	後	57	43	0
100万回	先	52	48	0
	後	60	40	0
400万回	先	64	36	0
	後	65	35	0
900万回	先	82	18	0
	後	76	24	0