UCTに方策勾配法を用いるガイスターAIの研究

青木蓮樹[1,a)]　橋本剛[1]

概要：ガイスターはチェスに似たルールのボードゲームであり，2017年からAI大会が開催され，不完全情報ゲームのテストベッドとして注目されている．2021年に開催された大会では，方策勾配法を用いたAIが優勝している．このAIは探索を用いないため，探索を用いることでより強いAIを実現できると考えられる．本研究では，探索アルゴリズムのUCT(Upper Confidence Tree)に方策勾配法で求まる行動確率を用いる手法をいくつか提案し，対戦実験によってそれらの性能を評価する．実験結果から，優勝したAIを上回る強さの手法が確認できた．

A Study on Geister AI using the policy gradient method for UCT

Aoki Renju[1,a)]　Hashimoto Tsuyoshi[1]

Abstract: Geister is a board game with rules similar to chess, and has attracted attention as a test bed for incomplete information games, with AI competitions beginning in 2017. The 2021 competition was won by an AI using the policy gradient method. Since this AI does not use search, it is believed that using search can achieve a stronger AI. In this study, we propose several methods that use action probabilities obtained by the policy gradient method for UCT(Upper Confidence Tree) of search algorithms, and evaluate their performance through competitive experiments. The experimental results confirmed a method stronger than the AI that won the competition.

1. はじめに

ガイスターは，チェスに似たルールのボードゲームであり，2017年からAI大会が開催され，不完全情報ゲームのテストベッドとして注目されている．ガイスターは，青駒と赤駒の二種類を用いる，相手の駒の種類がわからない特徴を持つ二人用の不完全情報ゲームである．ガイスターAIには，相手の駒の種類を推測する[1]，深層強化学習のDQN (Deep Q Network)を利用する[2]，青駒と赤駒の両方の性質を持つ「紫駒」を用いて最悪なケースをαβ探索をする[3]などがある．志賀らは，強化学習の一つである方策勾配法を利用し，当時最強とされたAIの「早指しAgent」を開発した[4]．強化学習を用いたガイスターAIはそれまでにも開発されていたが，強いAIは実現されていなかった．その中で，方策勾配法というアプローチにより，強化学習を用いた強いAIを実現した．

完全情報ゲームであるチェスや将棋では，深く探索するとAIがより強くなることが知られている．不完全情報ゲームであるガイスターでも，探索をすることでAIが強くなると考えられる．しかし，ガイスターは，相手の駒の種類によって局面の評価が大きく異なるため，αβ探索のように局面から評価値を得て探索を行う手法は難しい．そこで，局面から評価値を得ずに探索できるモンテカルロ法を用い，代表的な探索方法であるUCT (Upper Confidence Tree)を使用する．強いAIの早指しAgentに用いられる手法を，UCTに用いることでより強いAIを実現を目指す．ここで，方策勾配法は行動確率を求めるため，UCTにおけるノードでの行動選択やプレイアウトにて，方策勾配法で求まる行動確率を応用できる．こうすると，ランダムな行動確率で探索するよりも有意義な探索ができ，強くなると考えられる．そこで，UCTにおけるノードでの行動選択やプレイアウトで方策勾配法を用いる方法を三種類実装し，それぞれの強さを対戦実験により調べる．

[1] 松江工業高等専門学校
National Institute of Technology, Matsue College
[a)] s2101@matsue-ct.ac.jp

2. ガイスターについて

2.1 ガイスターのルール

ガイスター (Geister) は，オバケの形をした駒を用いるドイツ発祥のチェスに似た二人用の不完全情報ゲームである．ガイスターでは，背中に青い印のついた良いオバケ駒（青駒）と，赤い印のついた悪いオバケ駒（赤駒）の 2 種類を用いる．ガイスターの初期配置例を図 1 に示す．はじめに，各プレイヤはそれぞれの駒を 4 個ずつ自陣の 8 マスに相手にわからないように自由に配置する．試合はチェスと同様に交互に駒を動かしていく．1 ターンに動かせる駒は 1 個であり，移動可能な範囲は青駒と赤駒どちらも同じで縦横に隣接する 4 マスのうち自分の駒がないマスに移動できる．また，相手の駒があるマスに移動すると，その駒を取ることができ，取った駒の種類を知ることができる．勝利条件は，相手の青駒をすべて取る・自分の赤駒がすべて取られる・自分の青駒を 1 個脱出させる，のいずれかである．脱出は青駒のみができ，相手側の矢印マスから 1 手かけて脱出できる．

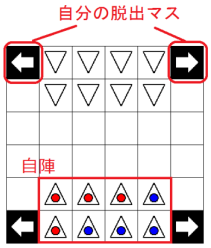

図 1 ガイスターの駒の初期配置
Fig. 1 Initial placement of Geister's pieces

2.2 ガイスターの戦略

ガイスターでは，相手に駒の種類を読まれないような戦略や，相手の駒を推測する戦略を要求される．例えば，赤い駒が 1 つ相手に知られると，相手はその駒を取らないようにするだけで「赤駒を取ることによる負け」にならないため，相手に有利な状況になってしまう．このように相手の駒の種類は試合をする上で重要な情報となる．そのため，ブラフを掛ける AI[5] や，相手の駒がどの種類かを推測する AI[1] なども開発されている．

3. 早指し Agent

志賀らは，方策勾配法を利用する AI「早指し Agent」を開発した [4]．探索を用いないが，大会で優勝する強さの AI である．方策勾配法は方策ベースの強化学習の一つであり，方策勾配定理に基づいて学習を行う．志賀らは，指し手を駒の位置，駒を取った数，駒を取られた数，駒を取る動きであるか，脱出する動きであるかという盤面の情報を 122 桁のビット列で表現し，このうちの二つの情報の連言を特徴量とし方策勾配法に用いた（図 2）．この特徴量を扱うことで，将棋 AI に用いられる三駒関係のように位置関係を考慮でき，それに加えて相手の赤駒の数や駒取りなどを考慮することで扱う情報量を増やし，不完全情報ゲームであるガイスターに対応している．また，盤面の価値を求める学習では，不明である相手の駒の種類によって有利不利が異なるため，同じような盤面での価値が安定せず学習が難しくなるが，方策勾配法は盤面の価値ではなく行動確率を求めるため，相手の駒の種類に左右されにくく安定した学習ができ，強い AI が実現できたと考えられる．この手法で学習された評価関数を HAF(Hayazashi Agent Function) と呼ぶ．

早指し Agent は探索を用いずに強いため，HAF と探索を組み合わせるとより強い AI になることが予想される．ここで，代表的な探索方法の一つである α β 探索は盤面の価値を用いるため，盤面の価値ではなく行動確率を求める HAF と組み合わせるのは難しい．そのため，盤面評価が不要である乱択アルゴリズムの UCT に HAF を用いることで強い AI の実現を目指す．

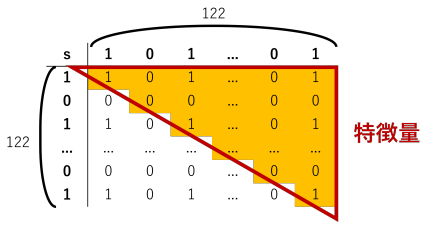

図 2 早指し Agent が用いる特徴量
Fig. 2 Features used by Hayazashi Agent

4. 比較する手法

UCT をベースにし，HAF を UCT のプレイアウトやノードの選択に応用する．UCT は，モンテカルロ木探索を改良したアルゴリズムであり，UCB 値 (Upper Confidence Bound) が大きいノードの選択を葉ノードまで繰り返すことで探索を行う．葉ノードにたどり着くと，そのノードからランダムプレイアウトを行う．このプレイアウトの結果を利用し，通ってきたノードの UCB 値を更新する．ここで UCB 値は，プレイアウトの勝率が高いほど大きくなり，また，そのノードの探索回数が少ないほど大きくなる値である．UCB 値が初期値である場合やプレイアウトでは行動確率が均等である．そこで，HAF によって求められる行動確率を用いることで，探索の質が良くなり，真に勝率が高いノードをより早く探索できると考えられる．また，HAF の計算結果に加えて未来の情報を持つことになるため，早指し Agent よりも強くなることが予想される．そこで，UCT に HAF を応用する方法を三種類実装し，強さを

比較する．それぞれの方法について HAF を応用する箇所を以下に示す．

方法 A.　UCB 値，プレイアウト
方法 B.　UCB 値
方法 C.　根ノードにいるときの UCB 値

4.1　方法 A

方法 A は UCB 値とプレイアウトに HAF を用いる方法である（図 3）．UCB 値に HAF を用いることで強い行動を主に探索するため，少ない探索回数で良い結果が求まり，プレイアウトの行動確率に HAF を用いることで，ランダムプレイアウトよりも求まる勝敗結果の質が良くなると考えられる．

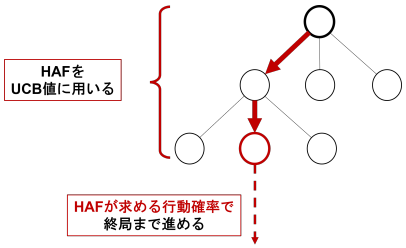

図 3　方法 A のアルゴリズム
Fig. 3　Algorithm of Method A

4.2　方法 B

方法 B は UCB 値のみに HAF を用いる方法である（図 4）．方法 A と同様に UCB 値に HAF を用いることで強い行動を主に探索するため，少ない探索回数で良い結果が求まると考えられる．また，プレイアウトには HAF を用いないことで，一回の探索に要する計算量が小さくなるため，探索できる回数が多くなり深くまで探索できる．

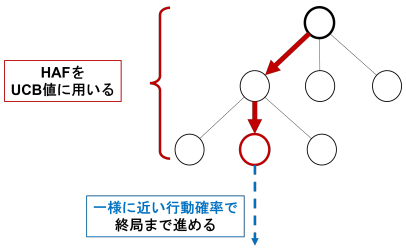

図 4　方法 B のアルゴリズム
Fig. 4　Algorithm of Method B

4.3　方法 C

方法 C は根ノードにいるときの UCB 値のみに，HAF を用いる方法である（図 5）．HAF の応用を最小限に抑えることで，方法 B よりも多く探索できる．

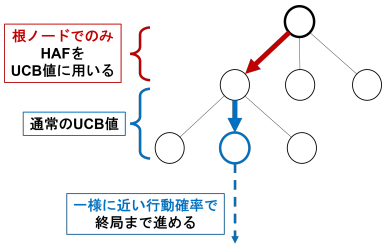

図 5　方法 C のアルゴリズム
Fig. 5　Algorithm of Method C

5.　実装

UCT に HAF を用いた AI「AokiUCT」の実装について説明する．まず，UCT を用いるガイスター AI を実装する．そして，方法 A，B，C それぞれの応用方法で HAF を組み込む．また，UCT についてランダムプレイアウトの高速化と相手の駒の種類を選ぶ方法を工夫したため，それらについて説明する．

5.1　UCT を用いるガイスター AI

UCT のアルゴリズムは，選択・展開・プレイアウト・結果の伝播に分けられる．探索木の初期状態は現在のゲームの状態を表すノードがただ一つ存在し，そのノードから探索を行う．ここで，あらかじめ相手の駒の種類を決めておき，その状態から探索を行う．

UCT はまず，現在のノードが葉ノードでなければ選択を行う．選択では，現在のノードから UCB 値が最大となる子ノードに遷移する．ここで，UCB 値は (1) 式により求める．

$$\mathrm{UCB} = \bar{w} + c\sqrt{\frac{\ln N}{n}} \qquad (1)$$

$\bar{w}$:　プレイアウト勝率
n:　プレイアウト回数
c:　$\sqrt{2}$（定数）
N:　総プレイアウト回数

葉ノードにたどり着くまで選択を繰り返し，葉ノードが条件を満たしていれば展開を行う．展開を行う条件は，プレイアウトを 1 回以上行ったノードであり，かつゲームが終了していないこととする．展開では，現在のノードの状態から任意の合法手を指した後の状態を調べ，それらを子ノードにする．ここで，HAF を用いない場合はランダムに選択した子ノードに遷移し，HAF を用いる場合は HAF により求まる行動確率が最大となる子ノードに遷移する．葉ノードにたどり着き，そのノードが展開を行う条件を満たしていなければプレイアウトを行う．プレイアウトでは，繰り返し確率的に行動を選択することで，現在のノードの状態から終局まで局面を進める．このときの勝敗結果を，たどってきたノードすべてに伝播しそれぞれのノードの UCB 値を変更する．そして，再び根ノードから選択を

行う．一定時間で探索を終了し，探索の結果から指し手を決定する．

5.2 UCT への方策勾配法の応用

HAF は，早指し Agent の学習回数 900 万回時点のもの[*1]を，方法 A，B，C の三種類で UCT に応用する．HAF を用いる際は，盤面情報を，122 桁に加え空白のビットとバイアス項を合わせた 127 桁のビット列の連言，8128 桁のビット列に変換する．そして，変換した値のビットが 1 となる箇所について，学習された重みを加算することで行動優先度が求まる．候補手それぞれの行動優先度を求め，(2) 式により行動確率が求まる．

$$\pi(s_i, \theta) = \frac{\exp h(s_i, \theta)}{\sum_j \exp h(s_j, \theta)} \tag{2}$$

$\pi(s_i, \theta)$: 重みベクトル θ のとき状態 s_i となる行動確率
s_i: 状態
θ: 重みベクトル
$h(s_i, \theta)$: 重みベクトル θ のとき状態 s_i となる行動優先度

ノードでの選択に HAF を応用する場合，UCB 値を HAF により求まる行動確率を用いて変更する．また，相手が行動する場面のノードで UCB 値を求めるとき，HAF が出力する行動優先度の正負を逆にして扱う．HAF を用いる場合の UCB 値の計算式は，AlphaZero[6] が用いる計算式を引用し (3) 式より求める．

$$\mathrm{UCB} = \bar{w} + C(s)P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)} \tag{3}$$
$$C(s) = \ln(\frac{N(s)+c_{base}+1}{c_{base}} + c_{init})$$

s: 状態
a: ノード遷移に行う行動
$\bar{w}$: プレイアウト勝率
P(s,a): 状態 s における行動 a の行動確率
N(s): 親ノードの訪問回数
N(s,a): 訪問回数
c_{base}: 19652.0（定数）
c_{init}: 1.25（定数）

プレイアウトに HAF を応用する場合，HAF により求まる行動確率に従って行動選択しプレイアウトを行う．相手の行動選択では，UCB 値を求めるときと同様に，HAF が出力する行動優先度の正負を逆にして扱う．

指し手は，UCT の探索結果と HAF の行動確率を合わせて決定する．根ノードが持つ子ノードのうち (4) 式の値が最大になる手を指し手にする．

$$\mathrm{Score} = \bar{w} + c\pi(s_i) \tag{4}$$

[*1] https://github.com/KaoruShiga/geister_rl

$\bar{w}$: プレイアウト勝率
c: 0.1（定数）
$\pi(s_i)$: HAF により求まる状態 s_i となる行動確率
s_i: 状態

HAF が求める行動確率を比率で合わせることで，UCT の探索結果に基づいて行動しつつ，勝率が同じ程度のノードが複数個ある場合に HAF に従って行動するため，UCT で強い行動が見つからなかい場合でも弱い行動選択を抑えられる．また，定数 c は予備実験を行い強かったためこの値を用いる．

5.3 ランダムプレイアウトの高速化

一般的にプレイアウトでは，一様分布に従って行動を選択する．ここで，BitBoard を応用することで，一様分布に近い確率分布での行動選択が高速に行える．

BitBoard とは，盤面情報を 2 進数で扱うことで高速化を図る技法である．通常，盤面と同じ大きさの BitBoard を扱うが，ガイスターの盤面サイズの 6×6 の BitBoard ではなく 8×8 の BitBoard を扱う．8×8 の BitBoard の内側 6×6 をガイスターの盤面として使用する．こうすると，値が 64 ビット整数型にちょうど収まるため扱いやすく，外側の余白に脱出先を設けることで脱出処理を BitBoard 上で高速に行うことができる．

BitBoard を応用するプレイアウトの方法は，まずプレイヤが動かす駒の移動先としてあり得るマスを取得し，その中からランダムに一つマスを選ぶ．次に，そのマスに隣接する駒を取得し，その中からランダムに一つ駒を選ぶ．そして選んだ駒を，選んだマスに進めるように状態遷移を行う．これを操作するプレイヤを入れ替えながら終局まで行う．駒の移動先としてあり得るマスを取得するには，プレイヤの駒に隣接するマスのうち，自身の駒が存在しないマスを取ればよい．プレイヤの駒が位置するビットが 1 である値を持っていれば，隣接するマスはシフト演算と論理和によって求められ，そこから自身の駒があるマスを除いたマスは論理否定と論理積によって求められる．求めた値からランダムにマスを一つ選ぶには，1 であるビットの数未満の乱数を取得し，その値の回数だけ 1 になっている一番下のビットを 0 にする．その後，1 になっている一番下のビットを取得することでランダムにマスを一つ選ぶことができる．次に，選んだマスに隣接する駒からランダムに一つ選ぶには，マスを選んだときと同様に，隣接マスを求め，駒のあるマスとの論理積からランダムに一つ選ぶ．こうして，1 回の行動選択にかかる計算量は二回の乱数取得と複数回の論理演算とループのみとなり，高速に手が選択できる．

また，マスを選んでから駒を選ぶことで，一様分布に従った行動選択にはならないが，移動できない駒を選ぶ可能性がなくなるため，選び直しがなくなり高速化につながる．

5.4 相手の駒の種類

UCT ではプレイアウトで終局まで手番を進め，その勝敗を見るためそれまでに相手の駒の種類を決める必要がある．ここで，相手の駒の種類は探索前に自分の手番が回ってくるたび決める．探索前に駒の種類を決める方法として，様々な組み合わせについて探索する方法が考えられるが，探索木が非常に大きくなるため深くまで探索できないことが懸念される．そこで，探索する組み合わせを固定することで深くまで探索できるようにする．ただし，取った駒が想定していた駒と違う場合，探索を続けられないため手番ごとに種類を決め直す．また，プレイアウトの前に決める方法もあるが，予備実験より探索前に決める方法が強かったためこちらを採用する．

探索する駒の種類が実際の駒の種類と一致していなければ弱い行動選択につながる．そこで独自の赤駒推定を用いて，駒の種類を決める．赤駒推定のアルゴリズムは，自駒に隣接する相手の駒は赤駒らしさが増し，隣接する箇所から離れる相手の駒は赤駒らしさが減る．また，脱出できる相手の駒が脱出しなかった場合は赤駒らしさを上限まで増やす．この赤駒らしさを確率分布に用いて，駒の種類を確率的に決める．予備実験よりこの方法が強かったため用いる．

6. 実験方法

対戦実験によって方法 A，B，C それぞれについて AokiUCT の強さを確認する．対戦は，先手後手入れ替えで 100 試合ずつ，合計 200 試合行う．300 手で決着がつかない場合は引き分けとする．

6.1 実験 1

AokiUCT の探索時間を 1 秒間として，方法 A，B，C について学習回数 900 万回時点の早指し Agent との対戦実験を行う．探索に HAF を用いることで，早指し Agent より強くなるかを確認する．

6.2 実験 2

AokiUCT の探索時間を 10 秒間として，方法 A，B，C について学習回数 900 万回時点の早指し Agent との対戦実験を行う．実験 1 と比較して，探索時間を長くすることでどれだけ強くなるかを確認する．

6.3 実験 3

AokiUCT の探索時間を 1 秒間として，方法 A，B，C について「Naotti-2020」との対戦実験を行う．Naotti-2020 は探索時間ではなく探索深さを指定する必要があり，1 秒程度で探索が終了するように探索深さを 6 に指定する．Naotti-2020 は，紫駒と α β 探索を用い，GAT2020 ガイスター大会で優勝した AI である．早指し Agent とは違う手法の AI を相手にして，どれだけ強いかを確認する．

7. 実験結果

7.1 実験 1 の結果

対戦結果を表 1 に示す．方法 A は負け越し，方法 B と方法 C は勝ち越している．方法 B と方法 C は勝率が同等であることがわかる．

表 1 探索時間 1 秒での対早指し Agent の勝敗表 (行プレイヤ勝ち数-列プレイヤ勝ち数-引き分け数)

Table 1 Win/loss table vs. Hayazashi Agent in search time of 1 second (number of column player wins - number of column player wins - number of draws)

	早指し Agent
AokiUCT 方法 A	32 - 68 - 0 36 - 54 - 0
AokiUCT 方法 B	61 - 38 - 1 53 - 44 - 3
AokiUCT 方法 C	62 - 36 - 2 50 - 48 - 2

7.2 実験 2 の結果

対戦結果を表 2 に示す．実験 1 と同様に，方法 A は負け越し，方法 B と方法 C は勝ち越している．実験 1 の結果と大きな差がないことがわかる．

表 2 探索時間 10 秒での対早指し Agent の勝敗表 (行プレイヤ勝ち数-列プレイヤ勝ち数-引き分け数)

Table 2 Win/loss table vs. Hayazashi Agent in search time of 10 second (number of column player wins - number of column player wins - number of draws)

	早指し Agent
AokiUCT 方法 A	35 - 64 - 1 29 - 69 - 2
AokiUCT 方法 B	51 - 40 - 9 62 - 38 - 0
AokiUCT 方法 C	56 - 37 - 7 55 - 43 - 2

7.3 実験 3 の結果

対戦結果を表 3 に示す．別の実験と同様に，方法 A は負け越し，方法 B と方法 C は勝ち越している．方法 C が最も勝率が高くなっている．

8. 考察

すべての実験で，方法 A は負け越し，方法 B と方法 C は勝ち越している．方法 A はプレイアウトで HAF を応用するが，HAF の応用には一定の計算量を必要とするため，プ

表 3 対 Naotti-2020 の勝敗表 (行プレイヤ勝ち数-列プレイヤ勝ち数-引き分け数)

Table 3 Win/loss table vs. Naotti-2020 (number of column player wins - number of column player wins - number of draws)

	Naotti-2020
AokiUCT	31 - 68 - 1
方法 A	32 - 67 - 1
AokiUCT	51 - 45 - 4
方法 B	52 - 48 - 0
AokiUCT	58 - 38 - 4
方法 C	64 - 32 - 4

レイアウト全体の計算量が非常に大きくなる．そのため，十分な回数の探索ができず，行動決定に用いるプレイアウト結果がプレイアウト回数が少ないため信頼性が小さくなり，弱い行動を選択してしまう．方法 B と方法 C ではプレイアウトで HAF を応用しないため計算量が抑えられ，十分な回数の探索ができるため，この問題が起きないと考えられる．方法 B は UCB 値で HAF を応用し，深く探索した先のノードでも強い行動を選択できるため，探索結果が強くなる．方法 C は，HAF の応用が根ノードにいるときのみに限られているが，プレイアウト回数は方法 B よりも多くなり，UCT が深くまで探索できるため強くなる．また，HAF は探索を用いない環境で学習されているため，深いノードで HAF を用いない方法 C との相性が良い可能性がある．

実験 1 と実験 2 の結果から，探索時間が 1 秒と 10 秒では強さに大きな差がないことがわかる．これは，行動決定が HAF に大きく依存していて，UCT はその補助をするように働いているからだと考えられる．行動を見ると，ゲームのほとんどの場面で，HAF が求めた行動優先度が最大となる行動をしている．序盤では，どの手を指してもプレイアウトの結果が変わりにくいため，HAF に従った行動をする．終盤で確実に脱出できる場面では，HAF が学習しているため，脱出に向かう行動の優先度が大きくなっている．そのため，探索によって脱出できる行動を探すのはメリットが少ない．ただし，HAF だけでは相手の脱出を防がないことがあるため，それを UCT によって防ぐことができる．また，赤駒推定が進むと UCT の結果が変わってくるため，HAF が優先度が大きいと考える行動とは違う行動をするようになる．このように主に HAF に従って行動し，必要な場面だけ UCT が活かされるため，探索時間が強さに大きく影響しなかったと考えられる．

実験 3 の結果から，方法 B と方法 C は Naotti-2020 に勝ち越している．Naotti-2020 は，紫駒を用いることでガイスターを自分に不利な完全情報ゲームとして扱い，α β 探索を行う AI である．早指し Agent と違い，探索を用いる強い AI であるが，勝ち越すことができたため AokiUCT は従来の探索用いる AI よりも強いと言える．また，方法 C は方法 B よりも戦績が良いため，方法 C がより強い方法だと考えられる．

また，方法 C を行う AokiUCT で GAT2022 ガイスター AI 大会に参加し，結果は 4 位であった．手強い AI が多く参加している中，上位の成績を取れたため有効な方法であると言える．しかし，さらに上位の AI があるため，まだ強くできる余地があると考えられる．

9. おわりに

本研究では，不完全情報ゲームであるガイスターを対象とし，探索を用いない強い AI である早指し Agent の手法を，UCT に応用した．早指し Agent や Naotti-2020 といった強い AI に勝ち越す AI が開発でき，UCT に方策勾配法を応用する有効性が示された．GAT2022 ガイスター AI 大会では 4 位であったが，工夫を加えることで優勝できる強さになると考える．今後は，相手の駒の種類を決める方法を工夫することで強くすることを考える．

参考文献

[1] 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.

[2] 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.

[3] 川上直人, 橋本剛：『紫駒』を用いた MinMax 探索によるガイスター AI の研究, 情報処理学会論文誌 Vol.62 No.10 pp.1716–1723, 2021.

[4] 志賀薫, 伊藤毅志：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, 2021-GI-46(16), pp.1-7, 2021.
onoglou, Matthew Lai, Arthur Guez, Marc Lanctot,

[5] 岸野圭汰, 川上直人, 橋本剛：ガイスター AI におけるブラフ戦

[6] David Silver et al.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science,Science,362(6419):1140–1144,s 2018