Vol.2019-GI-41 No.16
2019/3/9

# MCTS および Bitboard 計算を用いた GOLAD をプレイする AI の提案

鈴木崇雅[†1]　伊藤毅志[†1]

**概要**：デジタルゲーム AI のコンペティションを開催する Web 上のプラットフォーム Riddles.io では，二人零和有限確定完全情報ゲームである Game Of Life And Death（GOLAD）の AI コンペティションが行われている．GOLAD は，AI に許容される思考時間が非常に短い上に分岐が非常に多いため，乱数対戦を行うだけの単純なモンテカルロ木探索では十分なシミュレーション回数が得られなかった．本研究では，Bitboard 計算を用いて局面計算の高速化を行うとともに，プレイアウトに工夫を加えることでモンテカルロ木探索の効率化を図ることで，少ない計算量で妥当な探索ができるシステムを提案した．その結果、単純なモンテカルロ木探索の AI に対して有意に強くなることを確認した．また，現在 CNN を用いた改良も行っており，発表までに間に合えば，その結果についても言及する．

**キーワード**：AI，モンテカルロ木探索，ビットボード，ゲーム， GOLAD

# AI Bot on GOLAD by Using MCTS and Bitboard

TAKAMASA SUZUKI[†1]　TAKESHI ITO[†1]

***Abstract***: The codesports platform Riddles.io has held AI competitions for the game 'Game of life and death (GOLAD)' all times on their server. GOLAD is a two-players, zero-sum, and logical perfection information game. The simple Monte-Carlo Tree Search (MCTS) which gives good solutions for almost two-players games generally does not work on GOLAD efficiently because GOLAD AI agents need short think-time and complicated searches for enormous moves. In this paper, we propose the fast calculated MCTS in which bitboard calculations and fast playout simulations are implemented. Our proposal AI agent achieved high win-rate against a simple MCTS AI agent. We are also building Convolutional Neural Networks (CNN) for GOLAD. We will show you the result in the presentation if we have finished to build them

***Keywords***: AI, Monte-Carlo Tree Search, MCTS, Bitboard, Game, Game of life and death, GOLAD

## 1. はじめに

近年，デジタルゲーム AI のコンペティションが多く開催されており，MS.Packman の AI の性能を競う The Ms. Pac-Man Vs Ghost Team Competition や，Super Mario Bros. に似たゲームの AI の性能を競う Mario AI Competition 等が開催されている．しかしこれらのコンペティションは，開催期間が短いため対戦数が十分ではなく，コンペティションの結果が AI の性能を十分に表しているとは言い難い．一方，Web 上で常時 AI コンペティションを開催している Riddles.io[1]では，参加者は自作 AI をサーバー上にアップロードするだけでコンペティションに参加することが可能であり，定期的にサーバー上にて自動で AI 同士の対戦を実行し続けるため，多くの対戦結果から AI の性能を正確に評価することができる．

Riddles.io の提供するゲームの中に二人零和有限確定完全情報ゲームである Game Of Life And Death（GOLAD）が存在する．GOLAD は，有名なシミュレーションゲームであるライフゲームを二人ゲーム用に改良したゲームであり，縦 16×横 18 マスからなるフィールド上でマスを取り合うゲームである．各マスは生状態と死状態とに分類され，生状態のマスはいずれかのプレイヤに属する．マスの状態は，ターン経過毎にライフゲームのルールに従って伴って変化するほか，ターン毎に各プレイヤが行う行動によっても変化する．ターン毎にプレイヤが選択可能な行動の中に，任意の 3 マスを指定する行動が存在し，各ターンにおけるプレイヤの合法手数は，10 の 6 乗オーダーに及ぶ．また，Riddles.io の提供する GOLAD の特徴として，1 手あたりに許容される思考時間が 100ms と短いことも挙げられる．

以前より，囲碁などの二人ゲームをプレイする AI として，モンテカルロ木探索（MCTS）が有効であることが知られている[2][3]．しかし，GOLAD は先に挙げた膨大な合法手数と短い思考時間を要求するため，単純なモンテカルロ木探索が有効に機能しない．モンテカルロ木探索は，ノードの評価値の計算をするために乱数対戦による膨大なシミュレーションを必要とする．局面変化が複雑で合法手が多く，更に許容される思考時間が短い GOLAD では十分なシミュレーション回数が得られない．

本研究では，Bitboard 計算を採用することで局面の変化計算を高速化し，更に乱数対戦によるシミュレーションを改良することでシミュレーションにかかる時間を短縮する

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

手法を示す．これらの改良により，提案手法を利用した AI の有効性を示す．

## 2. Game Of Life And Death（GOLAD）

GOLAD は二人零和有限確定完全情報ゲームであり，図 1 のような 16×18 個のセルを敷き詰めた長方形のフィールド上で行われる．

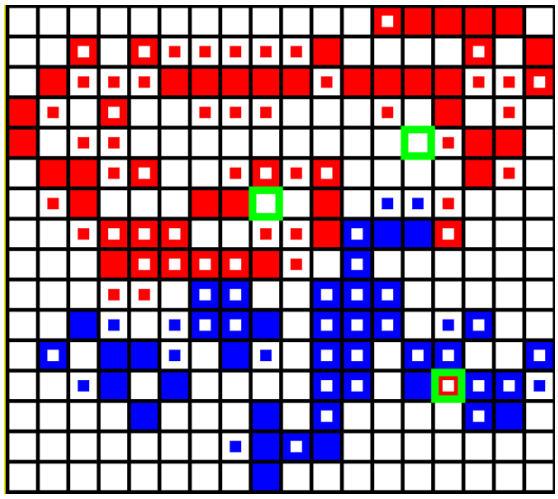

図 1　GOLAD の局面

各セルには，生状態か死状態が設定され，生状態のセルはいずれかのプレイヤの色に属する（図 1）．プレイヤが 1 手指すごとにターンが経過していき，ターンの経過に応じて各セルの状態が変化する．

図 1 におけるマスの状態の見方

- 赤の生のマスで次のターンも生き残るマス
- 青の生のマスで次のターンも生き残るマス
- 次のターンに死のマスに変化する赤のマス
- 死のマス
- 次のターンに青のマスに変化する死のマス

なお，緑の枠で囲まれたマスは，そのターンにプレイヤが行った行動を表している．

■マスの状態変化ルール

- 生のマスの周り（最大 8 か所）に存在する生のマスが 2 個未満または 4 個以上の時，そのマスは次のターンに死ぬ
- 生のマスの周りに存在する生のマスが 2 個か 3 個の時，そのマスは次のターンも生き残る
- 死のマスの周りに存在する生のマスが 3 個の時，そのマスは次の世代で生になる．ただし，そのマスの色は，周りの生のマスの色のうち，数が多い方の色になる．

■自分のターンに選択可能なプレイヤの行動

- Kill Move：任意の生のマスを殺す（いずれの色のマスでも良い）
- Birth Move：任意の自分の生のマスを 2 マス殺す代わりに，任意の死のマス 1 つを自分の色の生のマスにする
- Pass Move：何もしない

初期フィールドは，上半分にランダムで 40 マス赤マスを配置し，その配置をコピーしたものを 180 度回転させ，下半分に青マスとして配置することで作成される．

ゲームの勝敗は，先に自分の色の生のマスが無くなった方が負けとなる．両プレイヤの生のマスが同時に無くなる，または 200 ターン経過しても両プレイヤのマスの数が 0 にならない場合引き分けとなる．

## 3. 関連技術

### 3.1 モンテカルロ木探索（MCTS)

モンテカルロ木探索（MCTS）は，2006 年に現れた探索木アルゴリズムで，以前のミニマックスゲーム木探索アルゴリズムでは難しかった碁のような合法手数が多く，評価関数の設計が難しいゲームにおいても有効であることが示されている．MCTS も，ほかの木探索アルゴリズムと同様，根ノードから遷移先の状態の数だけ子ノードを展開していくことで探索木を作成していく．MCTS の処理を以下の 4 ステップに分けて説明する．

●選択（Selection）

各ノードにおいて，探索を行う自分の子ノードの選択を行う工程．この工程は，根ノードからスタートし，子ノードを持たないノードに到達するまで行われる．ノードの選択は、基本的には UCB 1 アルゴリズム[4]に従って選出される．UCB 1 は，各ノード期待報酬とそのノードの訪問回数に基づいて選択を行うアルゴリズムであり，各ノードにおいて，UCB1 の値が大きいノードを選択していく．UCB1 の値は以下の式（1）で表される．

$$\mathrm{UCB1} = X_j + C\sqrt{\frac{2\ln N}{N_j}} \qquad (1)$$

$Xj$ はそのノード（ノード j とする）の期待報酬を表しており，期待報酬が大きいノードほど，UCB1 の値が大きくなることが分かる．C は定数，$Nj$ はノード $j$ が選択された回数，$N$ は全選択回数の合計であり，被選択回数が少ないノードほど UCB1 の値が大きくなることが分かる．つまり，期待報酬が大きいノードを重点的に探索しつつも，期待報酬が小さくて探索が不十分なノードにも最低限の探索を行うようにするアルゴリズムであると言える．

●拡張（Expansion）

このステップは，まだ拡張していない子ノードが存在するノードに到達したときに行われる．どの子ノードを拡張

するかはランダムで決められる．

●シミュレーション

ノードが拡張されると，そのノードの評価値をシミュレーションによって求める．具体的には，そのノードからゲーム終了状態まで乱数対戦を行い，ゲームの終了状態における勝敗に応じて，そのノードの評価値を決める．乱数対戦とは，各プレイヤの行動をランダムに決定してゲームを進行することを示す．シミュレーションによってノードの評価値を求めることをプレイアウトと呼ぶ．

●更新（Backpropagation）

上記シミュレーションによって特定の子ノードの評価値が決定することで，その親ノードの評価値が変化する．各ノードにおける評価値の変化を，根ノードまで遡って更新していくステップがこれに該当する．

### 3.2 Bitboard 計算

Bitboard とは，ゲーム局面の状態を 2 進数で表したものであり，オセロなどの 2 人ゲームにおいてその有効性が確認されている[5]．例えばオセロであれば，8×8 のフィールドを構成する各マスに 1 ビットを割り当てる．そして、各マスに駒が存在する場合は 1 を，存在しない場合は 0 を割り当てる．想定する駒の状態は自駒か敵駒の 2 種類であるので，自駒用の Bitboard と敵駒用の Bitboard の 2 種類を用意するだけですべての局面を表すことができる．つまり，ゲーム局面を，64bit の数 2 つで表すことができる．

Bitboard を用いた計算を採用すると，ゲーム局面の表現を簡素化できるだけでなく，局面の変化を論理演算で求めることで計算を大幅に高速化することもできる．

## 4. 提案手法

### 4.1 単純 MCTS

まず前段階として，GOLAD に 3.1 にて説明した MCTS を適用することを試みた. 本文中は, この MCTS を単純 MCTS と表記することにする．

■単純 MCTS の設定

- プレイアウト数：100
- 1 ノードあたりの子ノード最大数：15
- 選択工程のアルゴリズム：USB1
- シミュレーション： Kill Move，Birth Move，Pass Move をランダムで選択し，更にそれぞれの Move におけるマスの選択をランダムで行う．シミュレーションは上限 10 ターン分行い，終局しない場合は双方のマスの色の割合で評価値を算出する．

ここで，子ノードの数を制限しているのは，GOLAD の各局面における合法手の数が 10 の 6 乗オーダーであり，そのすべてを子ノードとして作成することが不可能だからだ．子ノード 15 個の作成方法について以下に示す．

●Birth Move：

生贄セル 2 個

1. 全ての自セルの中から，そのセルを殺した場合，何もしない場合に比べて次のターンの自セルの相手セルに対する相対数が多くなるセルの集合を求める．
2. 上記集合の中から，ランダムに２つ自セルを選び，その２つを殺した時に，何もしない場合に比べて次のターンの自セルの相対数が多くなる場合，その２つの組み合わせを生贄候補として記録する．

生成セル 1 個

1. 全ての死セルの中から, そのセルを自セルにした場合, 何もしない場合に比べて次のターンの自セルの相対数が多くなるセルの集合を求める．

上記生贄セル 2 個と，生成セル 1 個の組み合わせによる Move が，何もしない場合に比べて次のターンの自セルの相対数が多くなる場合，これら 3 個のセルの組み合わせを Birth Move として登録する．このような Birth Move をランダムに 15 個作成する．

●Kill Move：

上記の基準を満たす Birth Move の数が 15 個に満たない場合，Kill Move を追加する．相手セルの中から，そのセルを殺した場合，何もしない場合に比べて次のターンの自セルの相対数が多くなるセルの集合を求める．当該集合からランダムに 15－上記 Birth Move の個数だけ作成する．

●Pass Move：

上記の基準を満たす Birth Move と Kill Move が存在しない場合にのみ，Pass Move を採用する．

このように単純 MCTS を作成したが，GOLAD の AI エージェントとして動かすことができなかった．具体的には 1 手の思考時間が 15,000ms を超えてしまい，Riddles.io の GOLAD が要求する一手あたりの思考時間 100ms に遠く及ばなかった．その原因を以下の 2 点と判断した．

**1）局面変化計算に時間がかかる点**

GOLAD の各マスの状態は，周囲 8 マスの状態に基づいて変化する．したがって，16×18 個のすべてのマスが、自身の次のターンの状態を求めるために周囲 8 マスの状態を参照しなければならない．この参照に要する時間が膨大だと考えた．

**2）シミュレーションに時間がかかる点**

シミュレーションを乱数対戦で行う場合，終局までシミュレーションを実行しなければ評価値を求められないため，時間がかかる．また，プレイヤの行動を完全にランダムにしてしまうと，局面が終局に向かうとは限らないため時間がかかる．

### 4.2 Bitboard 計算

4.1 の単純 MCTS の思考時間を短縮するために，3.2 にて紹介した Bitboard 計算を採用した. 局面の変化計算を bit 演算で再現する必要があるが，本研究ではその再現に成功した．計算手順を以下に説明する．

1. 赤と青それぞれのマスの状態を表すために 2 種類の Bitboard，RedBB と BlueBB を用意する．全マスを int 型の数 1 つで表すには桁数が足りないので，行ごとに int 型の整数を 1 つずつ割り当てる. したがって, RedBB と BlueBB はそれぞれ 16 個の要素を持つ int 型の配列として表現される．配列を構成する各要素は，横方向のマスの状態を 1,0 で表し，マスが自色であれば 1 をそうでなければ 0 を立てる．したがって，配列の各要素は 18 桁の 2 進数で表される．
2. 赤と青のマスを合算した Bitboard を AllBB とすると，AllBB は RedBB と BlueBB の論理和で求められる．AllBB において，論理演算を用いて下記の S2 と S3 の値を求める．
   S2 ：そのセルの周りに生のセルが 2 個隣接しているセルにビットを立てたもの
   S3 ：そのセルの周りに生のセルが 3 個隣接しているセルにビットを立てたもの
3. さらに, RedBB において, 下記の S0r と S1r を求める．
   S0r : そのセルの周りに赤セルが存在しないセルにビットを立てたもの
   S1r : そのセルの周りに赤セルが 1 個隣接しているセルにビットを立てたもの
4. 以下の式で局面変化計算，つまり次のターンの RedBB と BlueBB を求めることができる．

$$RedBB_{next} = \left(\overline{AllBB} \cdot \left((S3 \cdot \overline{S0r}) \cdot (S3 \cdot \overline{S1r})\right)\right) + \left(RedBB \cdot (S2 + S3)\right)$$

$$BlueBB_{next} = \left(\overline{AllBB} \cdot \left((S3 \cdot S0r) + (S3 \cdot S1r)\right)\right) + \left(BlueBB \cdot (S2 + S3)\right)$$

1. ～4. の手順により，局面変化計算のために必要な各セルへの参照回数が大幅に削減されるため．計算を高速化できた．しかし，単純 MCTS に当該 Bitboard 計算を採用しただけでは，安定して 100ms 以内に計算を納めることができなかった．

### 4.3 改良 MCTS

さらに計算速度を早くし，より強い AI エージェントを作成するため，MCTS 自体に以下の改良を加えた（図 2）.

**(1) シミュレーション**

プレイアウト時に選択する動作をランダムではなく，PassMove に変更．これにより，乱数対戦が終局に向かう可能性を高めた．乱数対戦の上限ターン数も 10 ターンから 5 ターンに短縮して，局面変化計算のコストを削減した．

**(2) UCB1 の改良**

思考時間が短く，十分なプレイアウト数を確保できない環境においても，妥当な探索木が生成されるように，探索木の UCB1 値の計算方法を以下の式（２）に変更した．

$$\text{newUCB} = X_j + C\sqrt{\frac{2\ln(N+\varepsilon)}{N_j+\varepsilon}} + RN \times \varepsilon \qquad (2)$$

$\varepsilon$ は定数で 1.0 を設定しており，$RN$ は 0～1.0 の値を取る乱数としている．ほかの値については先に UCB1 について説明した時と同じである. この値を利用することにより，ノードの選択が高いランダム性を持って行われる．プレイアウト数が少ない環境においては，ノードの選択に高いランダム性を設けることによって，バランスの良い探索木を生成することができる．

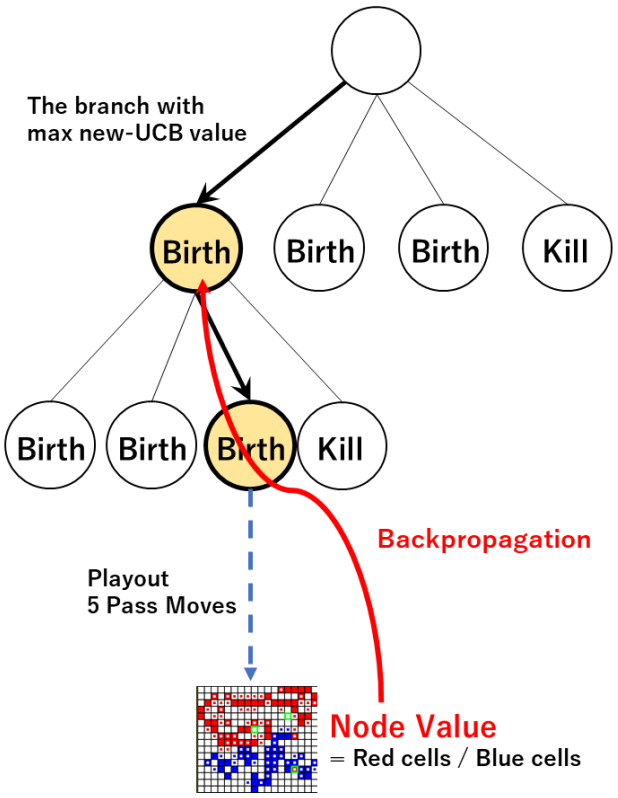


図 2　改良 MCTS

**(3) 子ノード数の制限**

子ノード数が多いと深い探索木を生成できないので，子ノードの数を 4 個に限定した．生成方法は以下のとおりである．

●Birth Move：3 個

3 つのセルの選ぶ順番で 3 通りの Birth Move を作成する．

1. 生成セル　→　生贄セル×２
2. 生贄セル　→　生成セル　→　生贄セル
3. 生贄セル×２　→　生成セル

・生贄セル 2 個の候補：全ての自色セル
・生成セル 1 個の候補：全ての死のセル

評価方法

1 つ目のセル選択 → 評価 → 2 つ目のセル選択 → 評価 → 3 つ目のセル選択 → 最終評価

各評価及び最終評価は，実際に各セル選択を行った状態で 3 ターン先の局面（すべて Pass Move を選択）をシミュレートし，その局面のマスの色の割合で評価を決定する．1 つ目のセル選択で最適なセル選択が確定したら，そのセル選択を維持した状態で次のセル選択を行う．これを 3 回繰り返して最終評価を行う．最終評価が最大の Move を上記 3 通りのセル選択それぞれで求め，3 通りの Move を生成する．

●Kill Move：1 個

全ての生セルに対し，そのセルを選択した状態で同じように 4 ターン先の局面をシミュレートし，一番評価値の高い Move を採用する．

●Pass Move：

上記 Birth Move と Kill Move がいずれも存在しない場合にのみ，Pass Move を選択する．

### (4) プレイアウト数

実際に Riddles.io のサーバー上で，100ms の思考時間に収まるプレイアウト数を計測し，50 回に設定している．

## 5. 性能評価

提案手法の性能を評価するために，以下の 3 つの AI エージェントを想定した．

**AI1**：4.1 の単純 MCTS

**AI2**：4.1 の単純 MCTS＋4.2 の Bitboard 計算

**AI3**：4.3 の改良 MCTS＋4.2 の Bitboard 計算

### 5.1 AI1 と AI2 の計算速度比較

AI1 と AI2 は MCTS のアルゴリズムは同じであるため，強さは変わらず，計算速度のみが異なる．AI1 と AI2 の対戦を行い，Bitboard を採用したことによる計算速度の差を計測した．AI1 を先手，AI2 を後手として，10 手指すのにかかる時間を計測した．

**AI1：233,424ms**

**AI2：10,665ms**

Bitboard の採用により，非採用時に比べて 20 倍以上計算が高速化されたことが確認できた（図 3）．

### 5.2 AI2 と提案手法の計算速度比較

AI2 と AI3 は同じ Bitboard 計算を採用しているが，MCTS のアルゴリズムが異なるため，計算速度が異なる．AI2 を先手，AI3 を後手として，10 手指すのにかかる時間を計測した．

**AI2：12,891ms**

**AI3：3,865ms**

MCTS を改良したことにより，3 倍以上計算が高速化されたことが確認できた（図 4）．

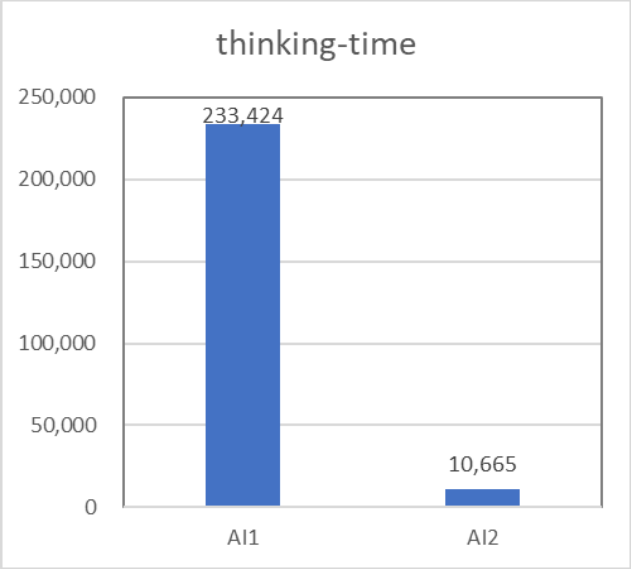


図 3　AI1 と AI2 の 10 手あたりの思考時間の比較

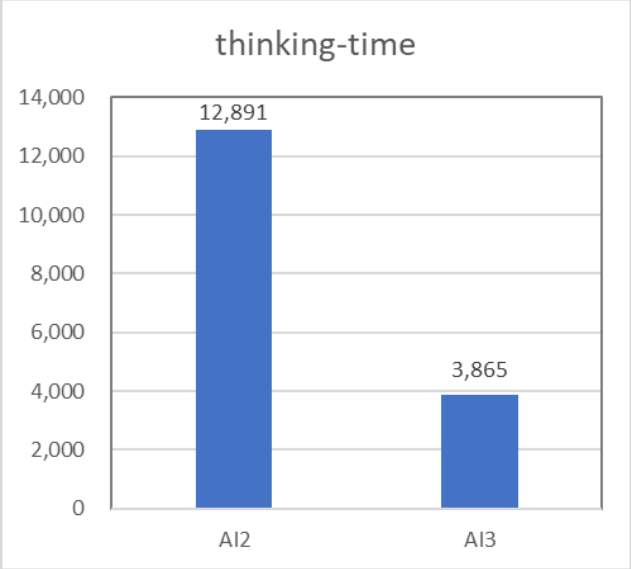


図 4　AI2 と AI3 の 10 手あたりの思考時間比較

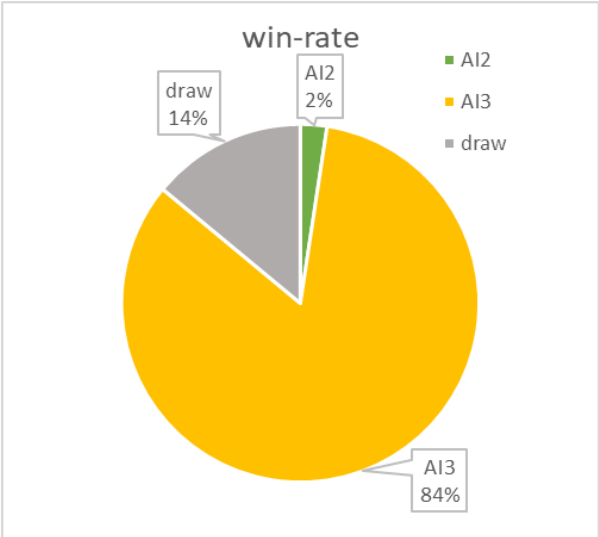


図 5　AI2 と AI3 の勝敗

### 5.3 AI2 と AI3 の勝率比較

AI2 と AI3 の対戦を先手後手それぞれ 250 回，合計 500 回実行したところ，AI3 から見て，418 勝 12 敗 70 分となった．

**この結果はから，**AI3 の改良によって AI2 に比べて有意に強くなったことが確認できた（図 5）．

## 6. 考察

### 6.1 Bitboard の効果について

5.1 の性能評価実験において Bitboard 計算の有効性を示した．計算を高速化できた理由として，局面を管理する配列の値の参照回数を大幅に削減できたことが考えられる．Bitboard 計算を未採用の場合，各マスの状態を個別に配列の要素として管理するため，局面の状態を更新する際に，約 16×18×8 回の配列要素への参照が必要になる．それに対して，Bitboard 計算を採用した場合，フィールド上のマスの各行を 1 つの int 型の整数で表し，尚且つ論理演算で次のターンの局面を計算するため，16×3 回の参照で事足りる．これにより，計算を高速化できたものと考察する．

### 6.2 MCTS の改良の効果について

5.2 及び 5.3 の性能評価実験において，改良 MCTS が単純 MCTS に比べて，計算速度の高速化，及びプログラムの強化においてともに優れていることが確認できた．

まず，計算速度について考察すると，計算速度を高速化できた 1 つ目の要因はプレイアウト数を削減したこと，2 つ目の要因は 1 回のプレイアウトにかかる時間を削減したことにあると考える．1 つ目については，プレイアウト数を 100 から 50 に削減したことにより，単純に計算時間を半減できた．2 つ目については，プレイアウトにおける局面の読み込み数を 10 から 5 に削減したこと，そして，Pass Move のみを選択することで，ランダムに Move を選択するよりも早く終局に近づけたことが考えられる．

次に，強さについて考察すると，短い計算にも関わらず強い AI を作成できた理由は，質の高い子ノードを生成できたことにあると考える．GOLAD では多くの局面において Birth Move が最善手になる可能性が高いことが経験上知られているため，質の高い Birth Move の生成方法が重要になる．Birth Move では 3 つのセルを選択することになるが，単純 MCTS では，個々のセルの評価を基準にセルの選択をしており，選択されたセル同士の相乗効果を考えていなかった．一方，提案手法では，順番にセルを選択していき，1 つ目あるいは 2 つ目のセルが選択された状態で次のセルを選択した場合の評価を算出するので，選択された 3 つのセルの関係性を強く意識して子ノードを作成できている．このような，質の高い子ノードの生成が強さに影響しているものと考察する．

## 7. まとめ

GOLAD の AI エージェントとして，Bitboard 計算と改良 MCTS を用いた AI エージェントを提案した．そして，提案手法が，従来の単純 MCTS による AI に対して，有意に強いことが確認できた．

なお，提案手法の AI エージェントを Riddles.io において実際に稼働させており，2019 年 2 月 12 日現在，26 エージェント中 7 位の成績を収めている．

## 8. 今後の予定

更に強い GOLAD エージェントを作成するため，現在，Convolutional Neural Network (CNN)を利用した AI エージェントを作成中である．CNN が囲碁などの二人ゲームに有効であることは，AlphaGo Zero [6]で周知であり，囲碁以外のゲームにおける採用可能性についても Alpha Zero[7]で示されている．

本論文で作成した提案手法も，当該 CNN における自己学習に利用することを想定して作成したものである．発表当日までに CNN による AI エージェントが完成すれば，その内容についても言及したい．

## 参考文献

[1] “The codesports platform Riddles.io”. https://www.riddles.io/, (参照 2019-02-12).
[2] R´emi Coulom, “Computing Elo ratings of move patterns in the game of Go.”, Computer Games Workshop, 2007.
[3] Guillaume, C. Sander, B, Istvan, S. and Pieter, S., “Monte-Carlo Tree Search: A New Framework for Game AI.”, Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, 2008
[4] Peter, A. Nicol, C. and Paul F., “Finite-time Analysis of the Multiarmed Bandit Problem”, Machine Learning, Vol.47, No.2-3, pp.235-256, 2002
[5] Jack Chen , “Applications of Artificial Intelligence and Machine Learning in Othello “, tjhsst Computer Systems Lab, 2009–2010
[6] David Silver,” Mastering the game of Go without human knowledge”, nature, 2017
[7] David Silver, “A general reinforcement learning algorithm that masters chess, shogi and Go through self-play”, nature, 2018