2048プレイヤへの準貪欲強化学習の適用と評価

松崎 公紀[1,a)]

概要：ゲーム「2048」は，スライド＆マージ型の確率的一人ゲームである．これまでに「2048」のコンピュータプレイヤとして，N タプルネットワークを評価関数とし，その評価関数を貪欲的自己プレイによる強化学習を用いて調整したものが優れた結果 (3 層の Expectimax 探索と併用して平均得点 40 万点以上) を残している．本研究では，その貪欲的な強化学習手法を改良すると期待される 3 つの手法，すなわち，(1) 確率的に不成功に終わったプレイを除いて結果の良かったプレイを学習する，(2) 強化学習する手の選択を確率的にランダムとする (ε-Greedy)，(3) Expectimax 探索により選ばれたより良い手を用いて強化学習する，を適用する．さまざまなパラメータを試した実験の結果，いずれの手法も学習結果を悪化することとなった．学習結果が悪化した要因を 2048 のゲームの特徴をもとに考察する．

キーワード：2048, 強化学習, N タプルネットワーク

1. はじめに

確率的一人ゲーム「2048」は，類似のスライド＆マージ型のゲームの中でも非常に多くの人にプレイされたゲームである．開発者によると，公開からの最初の 3 週間でプレイされた総時間は 3000 年以上にもなる．また，多数の派生ゲームが作られ，公開されている [*1]．「2048」がこれだけ多くの人を惹きつける理由のひとつは，ゲームのルールを理解するのは容易であるが極めのが困難であるという性質にある．「2048」のコンピュータプレイヤも複数の研究者・開発者により作られてきた [2,3,5–8,10,11,14,17–20]．

これまでに作られた強いコンピュータプレイヤの多くでは，N タプルネットワークが評価関数として用いられ，その N タプルネットワークは TD 学習などの強化学習により調整されている [6–8,10,14,17,19]．その中でも，Jaśkowski による最先端のコンピュータプレイヤ [6] は，Temporal Coherence 学習をベースにマルチステージ化，重み昇進，冗長エンコーディングなどの手法を適用することにより，1 手 1 秒の時間制限のもとで平均得点 609,104 点を達成した．著者の先行研究 [8] では，後退 Temporal Coherence 学習とリスタート方策を提案し，N タプルネットワークによる評価関数の学習を改善できることを示した．

N タプルネットワークに対する強化学習において，これまで提案された手法はいずれも，貪欲法による自己プレイの結果から学習を行っている．一般に，ランダムプレイからの学習に比べて，貪欲的な自己プレイからの学習はより早く優れた学習結果を得ることができるのが利点である．一方で，局所最適な解に留まってしまう，未知の状態において適切に振る舞うことができない，といった欠点もある．実際，AlphaGo [13] の評価関数など他のゲームに対する強化学習では，ある程度の

[1] 高知工科大学
[a)] matsuzaki.kiminori@kochi-tech.ac.jp

[*1] 「2048」自身も，Threes や 1024 の派生ゲームである．

ランダム性を加えたプレイから学習を行い，よい結果が得られている．

そこで本研究では，著者の先行研究 [8] で用いたリスタート方策ありの後退 Temporal Coherence 学習から始めて，以下の 3 つの準貪欲な方法を試行する．

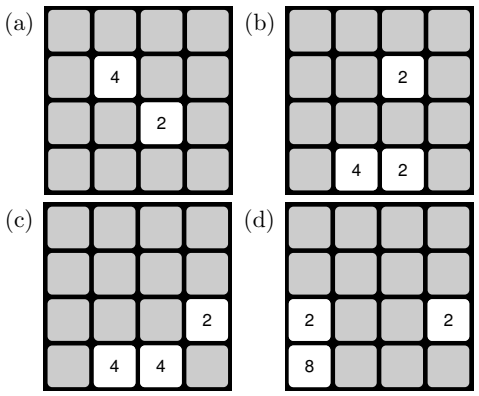
良いプレイのみの学習 「2048」では，確率的に頓死もしくは非常に不利な形になることがある．そのような不成功に終わったゲームを学習対象から外すことで，楽観的に学習を進める．

自己プレイへのランダム性の導入 準貪欲なプレイ方法として，確率 ϵ でランダムにプレイし，確率 $1-\epsilon$ で貪欲にプレイする ϵ-Greedy 法が知られている．いくつかの小さな ϵ について ϵ-Greedy を適用して，ランダム性を導入した自己プレイから学習を行う．

自己プレイへの探索の導入 確率的ゲーム「2048」では，Expectimax 探索が有効に働くことがすでに示されている [6, 7, 17]．そこで，自己プレイにおいて Expectimax 探索を実施し，より優れた手をもとに学習を行う．

これらの 3 つの方法のそれぞれについて，N タプルネットワークの評価関数の学習と評価を行う．実験を行ったところ，残念ながら，いずれの方法も有効でないという結果となった．そこで，実験の結果をふまえ，「2048」においてこれらの手法がうまくいかなかった理由として考えられることを考察する．

本論文の構成は以下のとおりである．第 2 節では，ゲーム「2048」のルールと特徴について簡単に示す．第 3 節では，N タプルネットワークによる評価関数と，それを強化学習により調整する手法を導入する．本研究では，著者による先行研究 [8] で提案した手法をベースラインとする．第 4 節では，良いプレイのみを学習する手法 (試行 1) について，そのアイデアと結果を述べる．第 5 節では，自己プレイへのランダム性の導入 (試行 2) について，そのアイデアと結果を述べる．第 6 節では，自己プレイへの探索の導入 (試行 3) について，そのアイデアと結果を述べる．これらの結果を受けて，第 7 節では，本研究で提案した手法が「2048」に対してうまく働かなかった理由について考察する．最後に，関連研究について第 8 節でまとめて示し，第 9 節で本論文をまとめる．



(a) 初期状態の例．2 つのタイルがランダムに出現．

(b) 1 手目として下を選択．2 つのタイルが下端へと移動し，(3,2) のマスに新しく 2 のタイルが出現．

(c) 2 手目として下を選択．2 つの 2 のタイルが併合されて 4 のタイルとなり，得点 4 を得る．(4,3) のマスに新しく 2 のタイルが出現．

(d) 3 手目として左を選択．2 つの 4 のタイルが併合されて 8 のタイルとなり，得点 8 を得る．(4,3) のマスに新しく 2 のタイルが出現．

図 1 ゲーム「2048」の流れ

2. ゲーム「2048」

2.1 ルール

ゲーム「2048」は，4×4 の盤面でプレイされる．ゲームの初期盤面では，2 のタイル (確率 0.9) か 4 のタイル (確率 0.1) がランダムに 2 つ置かれる (図 1 (a))．プレイヤが上下左右のいずれかの方向を選択すると，全てのタイルがその方向へと移動する (図 1 (b))．もし，同じ値をもつ 2 つのタイルが進行方向に沿って並んだ場合には，それらの 2 つのタイルは併合され和の値を持つタイル 1 つとなる (図 1 (c)，(d))．この併合により，プレイヤは新しくできたタイルの数だけ得点を得る．ここで，併合は奥側から起き，また，併合によってできたタイルは連鎖的に併合することはない．し

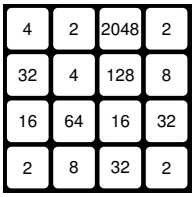


図 2 終了盤面の例

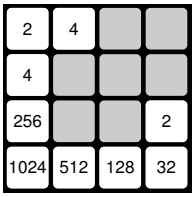


図 3 盤面の例．上・右はほぼ明らかに良くない．下・左はおそらく同程度に良い．

たがって，222␣，422␣，2222 というタイル列が右に動くと，それぞれ，␣␣24，␣␣44，␣␣44 となる．プレイヤは，移動・併合がまったく起きない方向を選択することはできない．これらの移動の後に，空きマスのうち 1 つがランダムに選ばれ，そこに 2 のタイル (確率 0.9) か 4 のタイル (確率 0.1) が出現する．

どの方向にも移動・併合ができなくなったとき，ゲームが終了する (図 2)．オリジナルの「2048」の目標は，以上のルールに従ってタイルを移動・併合して 2048 のタイルを作ることである．本研究での目標は，2048 のタイルができた後もゲームを続け，より多くの総得点をとることである．

2.2 特徴

これまで，ゲーム研究ではチェス・将棋・囲碁などのゲーム [*2] を対象として多くの研究がなされてきた．「2048」は，それらのゲームと比べ，以下に示す点が大きく異なる．

ランダム性の存在 「2048」では，プレイヤが方向 (手) を選択するたび，ランダムな位置にタイルが出現する．特に，10% の確率で 4 のタイルが出現することは，「2048」を難しくまた面白くしている．実際，minimax 探索よりも Expectimax 探索が有効であることに，このランダム性が影響していると考える．

分枝数が少なく，手数が多い プレイヤが選択できる方向 (手) は高々 4 通りである．一方，終了状態までの手数は，強いプレイヤの場合 25,000 以上 (得点は 80 万点程度) にもなる．この分枝数と手数との比は，他のゲームにはあまり見られない．Expectimax 探索などでは，ランダムに置かれる 2 と 4 のタイルの全ての可能性を考える必要があり，平均分枝数は 30 程度となる．

手の優劣 プレイヤは 4 つの方向から 1 つを選択するが，最善手と次善手が同じくらいの良さとなる局面が多いのではないかと著者は考える．例えば，図 3 において，上・右はほぼ明らかに良くない手であるが，下・左のどちらが良いかは自明ではなく，おそらく同程度である．

[*2] チェス・将棋・囲碁は二人完全情報確定零和ゲームに分類される．「2048」は，一人確率的（不完全情報）ゲームである．

3. N タプルネットワークと強化学習による 2048 プレイヤ

本節ではまず，N タプルネットワークによる「2048」の評価関数と，それを強化学習により調整する考え方について説明する．次に，本研究でベースラインとして用いる学習アルゴリズムとして，著者の先行研究 [8] で提案したリスタート方策ありの後退 Temporal Coherence 学習のアルゴリズムを示す．

3.1 N タプルネットワークによる評価関数

本節で説明する，「2048」に対する N タプルネットワークの考え方は，Szubert と Jaśkowski [14] によって最初に与えられたものである．また，ゲームの進行に沿って，複数の N タプルネットワークを使い分けるアイデアは，Wu ら [17] により提案されたものである．

N タプルネットワークは，いくつかの N タプルとそれらに対応した特徴重みの表からなる (図 4 (a))．各タプルが覆うセルの数を N，各セルの取り得る値の種類を K とすると，特徴重みの数は

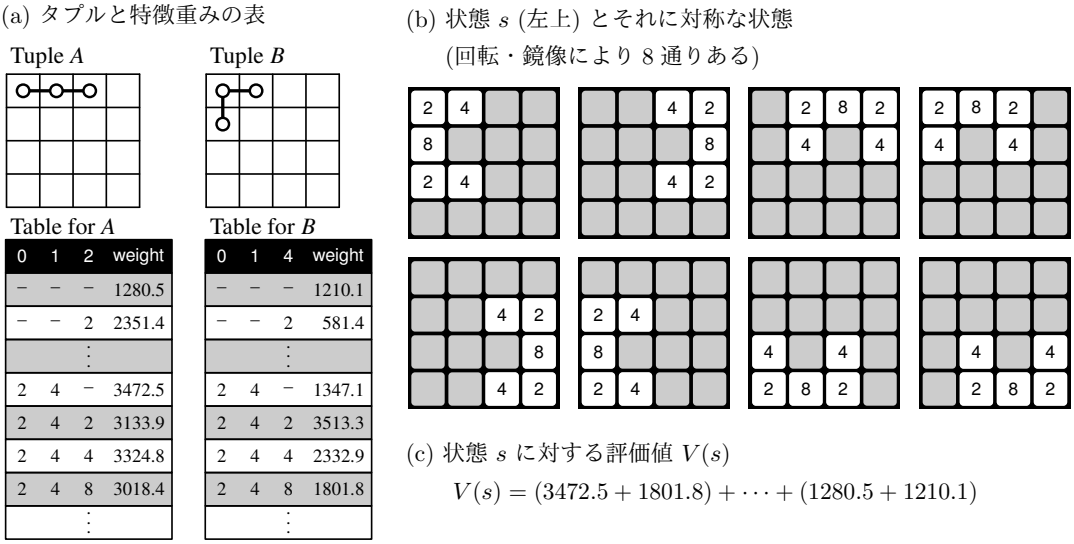


図 4 評価値の計算の例 [8]

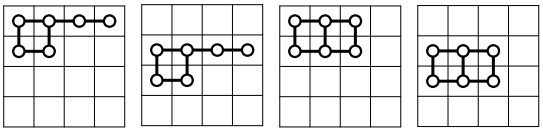


図 5 本研究で用いる N タプル [17]

K^N となる．本研究では，図 5 に示す 4 つの 6 タプルを用い，取り得る値の種類は $K = 16$ (最大のタイルの数は 32,768) とした．

第 3.2 節で示す手法などを用いて，適切な N タプルネットワーク (N タプルと特徴重みの表) が与えられるとする．そのとき，局面の評価値は，N タプルが示す位置にあるタイルの数に対応して特徴重みを表引きし，それらの和をとることで計算される．例として，図 4 に 2 つの 3 タプルの場合を示す．「2048」の盤面は 90 度回転・鏡像について対称であるので，8 つの対称な盤面のそれぞれについて各 N タプルの特徴重みを表引きする (図 4 (c)) ことに注意せよ．本研究で用いる 4 つの 6 タプルの場合，評価関数は 32 個の特徴重みの和として与えられる．

「2048」では，1 ゲームが長くまたゲームの進行とともに難易度が変化する．それにうまく対応するため，ゲームの進行とともに複数の N タプルネットワークを切り替えて用いるマルチステージ化を用いる．これまでに複数のマルチステージ化の手法が提案されている [6, 9, 17, 19] が，本研究では最大のタイルが 32768，16384，8192，4096，2048，1024，512 の各場合，および 256 以下の場合の計 8 つのステージに分ける手法を用いた．したがって，本研究で用いる N タプルネットワークの特徴重みの総数は $4 \times 16^6 \times 8 = 536{,}870{,}912$ である．

ここで，本論文の以降で用いる表記法をいくつか導入する．局面は s で表し，特に時間 t における局面を s_t と書く．局面 s に対して，i 番目の表引きで得られる特徴重みを $V_i[s]$ とし，それらの和である評価値を $V[s]$ と書く．表引きの回数 (すなわち，N タプルの個数の 8 倍) を m とする．

3.2 TD 誤差と Temporal Coherence 学習

強化学習を適用することにより，N タプルネットワークの特徴重みを適切に定めることができる．本研究で用いる Temporal Coherence 学習は，Jaśkowski [6] により最初に「2048」に適用され良い結果を得た．

強化学習の手法のうち，Temporal Coherence 学習およびその元となる Temporal Difference 学習では，時系列に沿った特徴重み間の誤差 (以下，TD 誤差と呼ぶ) をもとに特徴重みを更新する．本研究で

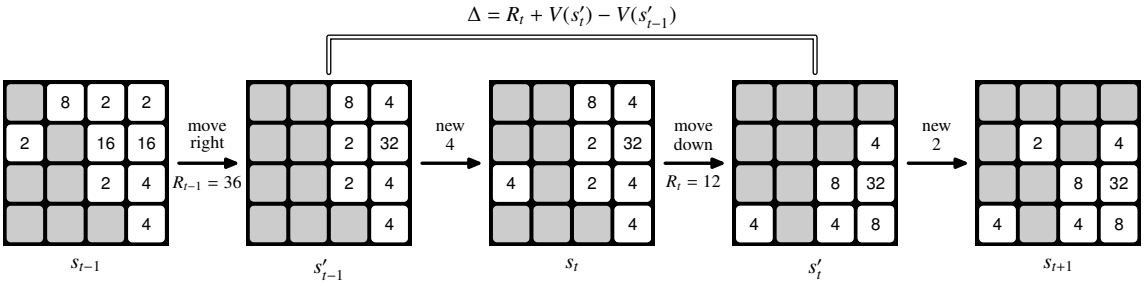


図 6 状態遷移と TD 誤差の定義

は，TD 誤差として，Szubert と Jaśkowski [14] により提案された TD-AFTERSTATE を用いるので，それを図 6 を用いて説明する．時間 $t-1$ の局面を s_{t-1} とする．プレイヤが方向を選ぶと，タイルが移動・併合して，得点 (報酬) R_{t-1} と局面 s'_{t-1} を得る．その後，ランダムな位置にタイルが出現して，時間 t の局面 s_t となる．同様に，局面 s_t においてプレイヤが方向を選択して，局面 s'_t (タイルの出現の前) を得る．ここで，誤差 Δ を

$$\Delta = R_t + V(s'_t) - V(s'_{t-1}) .$$

と定義する．最も単純な TD(0) アルゴリズムでは，自己プレイの結果から，局面 s に対応する特徴重みを

$$V'(s'_{t-1}) \leftarrow V(s'_{t-1}) + \alpha\Delta$$

と更新する．ここでパラメタ α は学習率と呼ばれ，それを適切に設定することで学習の速度や精度が改善される．

Temporal Coherence 学習 [1] は，学習率を自動で調整するアルゴリズムのひとつである．学習率は，それまでの誤差の和 $E_i[s]$ の絶対値と，それまでの誤差の絶対値の和 $A_i[s]$ との比によって与えられる（Algorithm 1 の 17 行目）．すると，学習率は最初 1 であり徐々に 0 へと近づく．学習時には N タプルネットワークの特徴重みごとにこれらの値を保持する必要があるため，3 倍 [*3] のメモリが必要となることに注意が必要である．

[*3] 実装では，特徴重みを小数部 10 ビットの 32 ビット固定小数点数，誤差の和 $E_i[s]$ と誤差の絶対値の和 $A_i[s]$ をそれぞれ 32 ビットの浮動小数点数で表した．

3.3 リスタート方策ありの後退 Temporal Coherence 学習

本研究ではベースラインの学習アルゴリズムとして，Algorithm 1 と Algorithm 2 とに示すリスタート方策ありの後退 Temporal Coherence 学習 [8] を用いる．この学習アルゴリズムは，Jaśkowski [6] による (オンライン) Temporal Coherence 学習に対して次の点を変更したものである．

後退 Temporal Coherence 学習 「2048」では 1 ゲームがとても長いため，ゲームの終わり近くで得られた報酬 (得点) がゲーム開始まで伝播するのに長い時間がかかる．そこで，1 ゲームのプレイを記憶しておき，そのゲームの終了時にまとめて後ろから前へと学習を行うことでこの問題を解決する．

リスタート方策 「2048」ではゲームの後ろにいくほど難易度が向上するため，ゲームの後半からより多くの局面を学習するほうがよいと考えた．そこで，次のゲームの自己プレイを，前のゲームの開始時点と終了時点の中間から再帰的に再開することとした．ただし，前の自己プレイのゲームの長さが $L = 10$ 以下であったときには，初期状態から再開する．

リスタート方策ありとなしのそれぞれについて，後退 Temporal Coherence 学習により学習した結果を [8] より再掲する．それぞれの場合について，最大 4×10^{10} 手分の自己プレイを行い学習を行った．途中，5×10^8 手ごとに，Greedy プレイと 3-ply Expectimax によるプレイを複数回行い，その平均スコアをプロットしたものがそれぞれ図 7 と 8 である．リスタート方策は，Greedy プレイの平均得

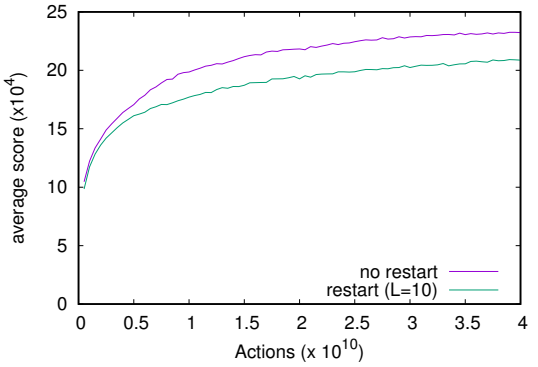


図 7　後退 Temporal Coherence 学習の結果: Greedy

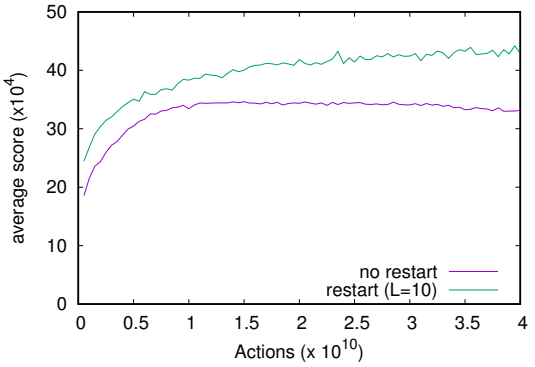


図 8　後退 Temporal Coherence 学習の結果: Expectimax

Algorithm 1 後退 Temporal Coherence 学習 [8]

function LearnFromSelfplay()
  $t \leftarrow 0$; $s \leftarrow$ InitialState()
  while not Terminal(s_t) **do**
    $a_t \leftarrow \mathrm{argmax}_{a \in A}$Evaluate($s_t, a$)
    $(R_t, s'_t, s_{t+1}) \leftarrow$ MakeAction(s_t, a_t)
    $t \leftarrow t + 1$
  BackwardLearning(1, t)
function Evaluate(s, a)
  $(R, s', s'') \leftarrow$ MakeAction(s, a)
  return $R + V(s')$
function BackwardLearning(t_0, t)
  TC-Update($t - 1$, $-V(s'_{t-1})$)
  for $\tau = t - 1$ **downto** t_0
    TC-Update($\tau - 1$, $R_\tau + V(s'_\tau) - V(s'_{\tau-1})$)
function TC-Update(t, Δ)
  for $i = 1$ **to** m **do**
    $\alpha =$ **if** $A_i[s'_t] = 0$ **then** 1 **else** $|E_i[s'_t]|/A_i[s'_t]$
    $\delta = \alpha\Delta/m$
    $V_i[s'_t] \leftarrow V_i[s'_t] + \delta$
    $E_i[s'_t] \leftarrow E_i[s'_t] + \delta$
    $A_i[s'_t] \leftarrow A_i[s'_t] + |\delta|$

* 関数 InitialState は初期状態を返す．関数 Terminal は，引数として与えられた状態がゲームの終了状態であるかを判定する．関数 MakeAction は状態と方向を入力とし，得点，タイルの移動後の状態，新しいタイルが出現した状態からなる 3 つ組を返す．

点を少し下げるが，3-ply Expectimax ではより高得点を達成する．

Algorithm 2 リスタート方策あり学習 [8]

function LearnFromSelfplayWithRestart()
  $t_{\mathrm{start}} \leftarrow 1$; $t_{\mathrm{end}} \leftarrow \infty$; $s_0 \leftarrow$ InitialState()
  while $t_{\mathrm{end}} - t_{\mathrm{start}} > L$ **do**
    $t \leftarrow t_{\mathrm{start}}$
    while not Terminal(s_t) **do**
      $a_t \leftarrow \mathrm{argmax}_{a \in A}$Evaluate($s_t, a$)
      $(R_t, s'_t, s_{t+1}) \leftarrow$ MakeAction(s_t, a_t)
      $t \leftarrow t + 1$
    BackwardLearning(t_{start}, t)
    $t_{\mathrm{end}} \leftarrow t$; $t_{\mathrm{start}} \leftarrow (t_{\mathrm{start}} + t_{\mathrm{end}})/2$

4. 試行 1: 良いプレイのみの学習

第 2 節で述べたように「2048」は確率的ゲームである．とくに，出現するタイルの位置とその数によっては，頓死もしくは非常に不利な形になって結果として少ない得点で終了することがある．これまでの研究により，そのような最悪ケースを考えて探索する minimax 法よりも，期待得点を最大化するように探索する Expectimax 法のほうが，平均得点をより大きくすることができることが分かっている．つまり，楽観的に探索を行う方法がうまくいっているのである．

そこで本研究では，試行 1 として，上記のように不成功に終わったゲームを学習対象から外すことで，楽観的に学習を行う方法を試みた．具体的には，それまでの自己プレイによって得た最大得点を覚えておき，行った自己プレイの得点が最大得点の P $(0 < P < 1)$ 倍以上であるゲームについてのみ学習を行う．ただし，リスタート方策では，

Algorithm 3 良いプレイのみ学習 (リスタートなし)

```
 1: function LearnGoodSelfplay()
 2:     t ← 1; s_0 ← InitialState()
 3:     while not Terminal(s_t) do
 4:         a_t ← argmax_{a∈A} Evaluate(s_t, a)
 5:         (R_t, s'_t, s_{t+1}) ← MakeAction(s_t, a_t)
 6:         t ← t + 1
 7:     maxSc ← max(maxSc, Σ_{i=0}^{t} R_i)
 8:     if P * maxSc < Σ_{i=0}^{t} R_i then
 9:         BackwardLearning(1, t)
```

Algorithm 4 良いプレイのみ学習 (リスタートあり)

```
 1: function LearnGoodPlayWithRestart()
 2:     t_start ← 1; t_end ← ∞; s_0 ← InitialState()
 3:     while t_end − t_start > L do
 4:         t ← t_start
 5:         while not Terminal(s_t) do
 6:             a_t ← argmax_{a∈A} Evaluate(s_t, a)
 7:             (R_t, s'_t, s_{t+1}) ← MakeAction(s_t, a_t)
 8:             t ← t + 1
 9:         maxSc ← max(maxSc, Σ_{i=0}^{t} R_i)
10:         if P * maxSc < Σ_{i=0}^{t} R_i then
11:             BackwardLearning(1, t); break
12:         t_end ← t; t_start ← (t_start + t_end)/2
```

ある (再開した) プレイが上記の条件を満たした際に，初期局面までの学習を一度だけ行い，その次のゲームは初期局面に戻るようにした．Algorithm 3 にリスタートなしの場合の変更後のアルゴリズムを，Algorithm 4 にリスタートありの場合の変更後のアルゴリズムを示す．

評価実験として，学習を行う閾値を $P = 0.75$，$P = 0.5$，$P = 0.25$ の 3 通りとし，リスタート方策をありなしの 2 通り，計 6 通りを試した．それぞれの場合について，自己プレイ 2×10^8 手ごとに Greedy と 3-ply Expectimax で評価プレイを行い，自己プレイ 5×10^{10} 手まで学習を行った．各条件について 5 回ずつ実験を行い，それぞれ平均値と平均値の 95%信頼区間とを求めた．

図 9 と図 11 に Greedy による評価，図 10 と図 12 に Expectimax による評価結果の推移を示す．また，表 1 に，1×10^{10} 手時点と 5×10^{10} 手時点のそれぞれにおいて，Greedy による平均得点，3-ply Expectimax による平均得点，実際に学習した手の数を示す．

結果より，リスタートなしの場合とリスタートありの場合のいずれにおいても，学習を行うかどうかを判定する閾値 P が 0.25 と小さいほうが良い結果となった．閾値 P が大きい場合に学習した手の数がとても少ないため，学習が不十分であることがひとつの理由と考えられる．リスタートの有無による比較では，リスタートありのほうが学習した手の数が多いにもかかわらず，平均得点が小さい．また，ベースライン (図 7，8) と比べると，学習した手の数が同等である場合においても明らかに悪い結果となっている．

以上の結果より，楽観的に良いプレイのみを学習するのは良くなく，むしろ悪い結果を含めてすべての自己プレイの結果から学習するべきであることが分かる．

5. 試行 2: 自己プレイへのランダム性の導入

貪欲な自己プレイの結果のみから強化学習を行った場合の問題点のひとつに，自己プレイが偏ってしまうことにより未知の局面に適切に対応できないことが考えられる．人間のトッププレイヤに勝つほどの強さを得た AlphaGo [13] などにおいても，自己対戦を行う際にプレイにある程度のランダム性を入れてこの問題に対処している．

そこで本研究では，試行 2 として，ϵ-Greedy 法により自己プレイにランダム性を導入することを試みた．この ϵ-Greedy 法では，確率 ϵ でランダムに手を選択し，確率 $1 - \epsilon$ で貪欲に手を選択する．プログラムにおいては，これまでの手の選択

$a_t \leftarrow \mathrm{argmax}_{a \in A} \textsc{Evaluate}(s_t, a)$

の部分を

if Rand() $< \epsilon$ **then**
  $a_t \leftarrow \textsc{RandomPlay}(s_t)$
else
  $a_t \leftarrow \mathrm{argmax}_{a \in A} \textsc{Evaluate}(s_t, a)$

に変えるだけである．ただし，ここでランダムに選ばれた手は学習の対象から外し，TC-Update 関数 (Algorithm 1 の 15–21 行目) を呼ばない．

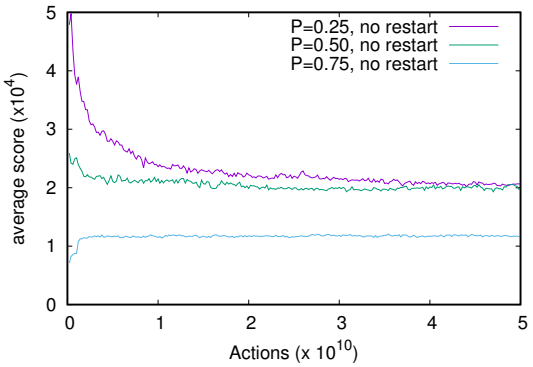


図 9 良い結果だけ学習した結果: リスタートなし，Greedy

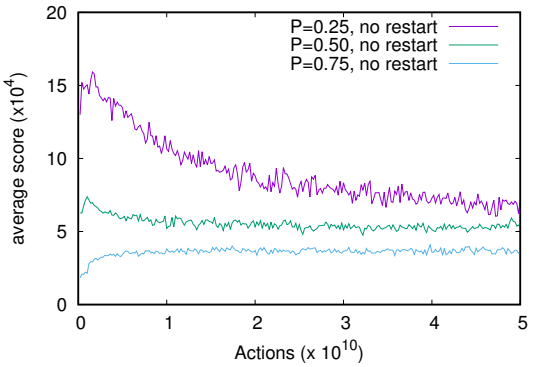


図 10 良い結果だけ学習した結果: リスタートなし，Expectimax

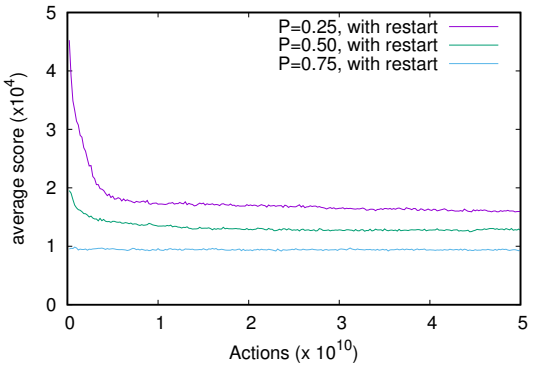


図 11 良い結果だけ学習した結果: リスタートあり，Greedy

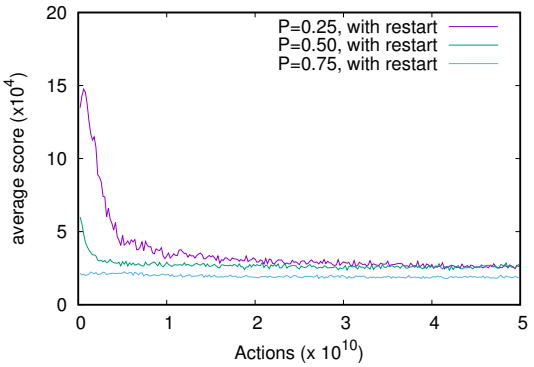


図 12 良い結果だけ学習した結果: リスタートあり，Expectimax

表 1 良い結果だけを学習した結果

		1×10^{10} 手時点			5×10^{10} 手時点		
リスタート	P	Greedy	Expectimax	学習数	Greedy	Expectimax	学習数
なし	0.75	11,507±2,648	38,154±11,484	0.14×10^8	11,735±3,053	34,249±11,963	0.20×10^8
なし	0.50	21,491±8,295	52,131±16,750	3.57×10^8	19,776±8,484	52,669±19,685	4.51×10^8
なし	0.25	23,606±6,642	106,515±28,999	17.7×10^8	20,614±6,046	72,373±17,404	24.6×10^8
あり	0.75	9,586±5,721	20,416± 9,033	0.38×10^8	9,421±5,683	18,060± 7,416	0.68×10^8
あり	0.50	13,438±7,489	29,007±13,080	9.52×10^8	12,733±6,458	26,907±12,036	13.1×10^8
あり	0.25	17,194±4,624	33,362±13,230	48.7×10^8	15,964±3,713	26,354± 7,311	98.9×10^8

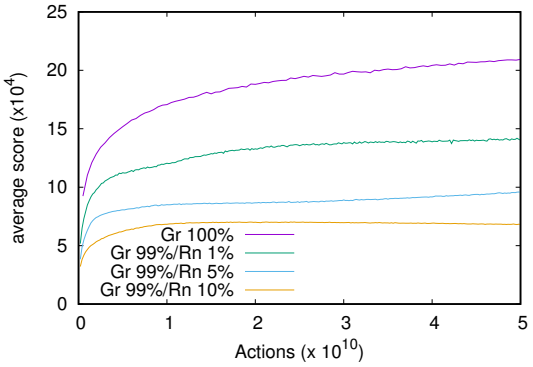


図 13　ϵ-Greedy 法の結果: Greedy

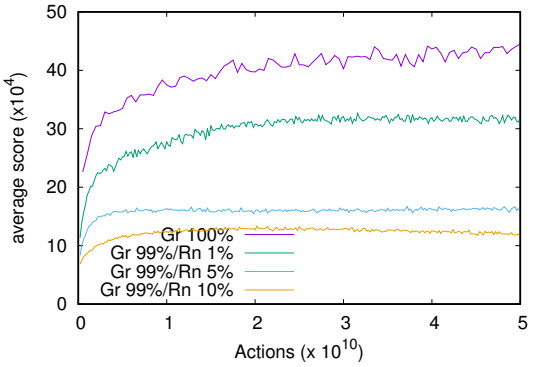


図 14　ϵ-Greedy 法の結果: 3-ply Expectimax

表 2　ϵ-Greedy 法の結果 (5×10^{10} 手の自己プレイ後)

プレイヤ	Greedy	Expectimax
Gr 100%	209,400±4,421	445,120±17,917
Gr 99%/Rn 1%	140,399±2,705	318,071±13,187
Gr 95%/Rn 5%	95,594± 808	161,944± 2,903
Gr 90%/Rn 10%	68,120± 441	118,126± 4,465

本研究では，ランダムに手を選択する確率 ϵ を $\epsilon = 0.01$ (Gr 99%/Rn 1%)，$\epsilon = 0.05$ (Gr 95%/Rn 5%)，$\epsilon = 0.10$ (Gr 90%/Rn 10%) の 3 通り試した [*4]．評価実験として，これらの ϵ-Greedy 法とベースライン (Gr 100%) のそれぞれの場合について，自己プレイ 2×10^8 手ごと (Gr 100%は 5×10^8 ごと) に Greedy と 3-ply Expectimax で評価プレイを行い，自己プレイ 5×10^{10} 手まで学習を行った．各条件について 5 回ずつ実験を行い，それぞれ平均値と平均値の 95%信頼区間とを求めた．

図 13 に Greedy による評価の結果，図 14 に 3-ply Expectimax による評価の結果，表 2 に 5×10^{10} 手の自己プレイを行った後の結果を示す．これらの結果より，ランダムプレイの確率 ϵ を大きくすると平均得点が小さくなり，「2048」においては ϵ-Greedy 法による自己プレイへのランダム性の導入がうまく働かないことが分かる．

6. 試行 3: 自己プレイへの探索の導入

「2048」では，評価値を用いた貪欲法 (Greedy) に対して，Expectimax 探索を行うことでより良いプレイができることがすでに知られている [6,17]．実際，ベースラインの学習結果のグラフ (図 7，8) では，貪欲法が 20–23 万点であるのに対し，3-ply Expectimax では 40 万点を超える平均得点を得ている．このことから，強化学習の際にも Expectimax 探索を行うことで，より良い手に基づいて学習を行うことが有用ではないかと考えた．

そこで本研究では，試行 3 として，強化学習における自己プレイ (の一部) を Expectimax 探索により得られた手に置き換えることを試した．具体的にはベースライン (Gr 100%) に加えて以下の 5 つの学習方法を試した．

Gr 75%/Ex 25%　75% の確率で貪欲法 (リスタートあり) の自己プレイを行い，25% の確率で Expectimax 探索 (2-ply，リスタートなし) の自己プレイを行う．

Gr 50%/Ex 50%　50% の確率で貪欲法 (リスタートあり) の自己プレイを行い，50% の確率で Expectimax 探索 (2-ply，リスタートなし) の自己プレイを行う．

Gr 25%/Ex 75%　25% の確率で貪欲法 (リスタートあり) の自己プレイを行い，75% の確率で Expectimax 探索 (2-ply，リスタートなし) の自己プレイを行う．

Ex After 25h　最初貪欲法の自己プレイ (リスタートあり) による学習を 9×10^{10} 手分 (約 25 時間) 行い，その後は Expectimax 探索 (2-ply，リスタートなし) の自己プレイを行う．

[*4] これは，他のゲームにおいて用いられている確率よりもずっと小さい．「2048」は手数の長いゲームであるため，小さな確率でも十分な回数ランダムな局面が生成されると考えた．

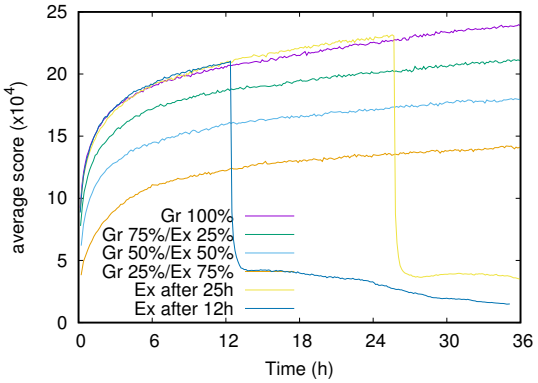


図 15　Expectimax 探索併用の結果: Greedy

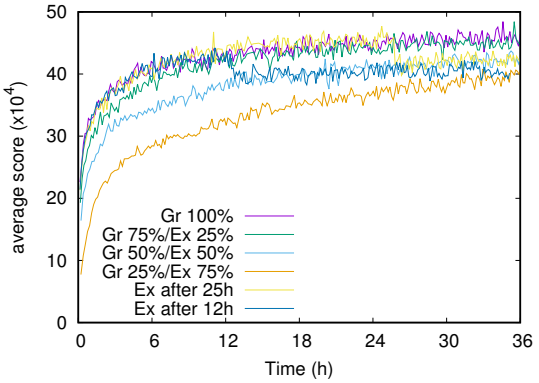


図 16　Expectimax 探索併用の結果: 3-ply Expectimax

表 3　Expectimax 探索併用の結果 (36 時間の学習後)

プレイヤ	Greedy	Expectimax	学習局面数
Gr 100%	238,297±12,457	458,626±26,431	12.7×10^{10}
Gr 75%, Ex 25%	210,082±12,696	459,371±24,527	7.4×10^{10}
Gr 50%, Ex 50%	179,955±15,215	428,166±52,089	4.5×10^{10}
Gr 25%, Ex 75%	141,206±20,771	408,619±35,635	2.5×10^{10}
Ex after 25h	37,532± 2,379	423,203±32,302	9.5×10^{10}
Ex after 12h	16,880± 4,185	41,2141±27,552	5.5×10^{10}

Ex After 12h　最初貪欲法の自己プレイ (リスタートあり) による学習を 4.5×10^{10} 手分 (約 12 時間) 行い，その後は Expectimax 探索 (2-ply，リスタートなし) の自己プレイを行う．「2048」ではある局面における有効手は少ないものの，ランダムに配置されるタイルを全通り列挙すると分枝数は平均でおよそ 30 となる．したがって，Expectimax の先読みの段数を 1 増やすごとに計算時間がおよそ 30 倍となる．十分な数の自己プレイを行うことができるよう，自己プレイで併用する Expectimax 探索の先読みの段数は 2 (貪欲法に比べて 1 手先を見る) とした．評価実験として，それぞれの場合について，約 10 分の学習ごとに Greedy と 3-ply Expectimax で評価プレイを行い，総学習時間が 36 時間程度まで学習を行った (学習局面数は学習方法ごとに異なる)．各条件について 5 回ずつ実験を行い，それぞれ平均値と平均値の 95%信頼区間とを求めた．

図 15 に Greedy による評価結果を，図 16 に 3-ply Expectimax による評価結果を示す．また，36 時間の学習後の各プレイヤの平均得点とそれまでの学習局面数を表 3 に示す．まず，図 15 より，Expectimax 探索の割合を増やすと Greedy によるプレイの平均得点が低下している．これは，Expectimax 探索を併用することでより精度の高い学習を行うという，著者の事前の予想に反する結果であった．さらに，途中で貪欲法による自己プレイから探索による自己プレイに変えたプレイヤでは，その変更の直後に 5 万点程度まで平均得点が大きく低下し，その後平均得点はさらに徐々に低下した．一方，3-ply Expectimax による評価 (図 16) では，Greedy による評価のときほどの差は見られない．自己プレイに探索を 25%導入したプレイヤの平均得点はベースラインと同程度であるものの，自己プレイに探索を加える割合が大きくなると平均得点は小さくなっている．また，途中で貪欲法による自己プレイから探索による自己プレイに変えたプレイヤでは，その変更の直後に 5–10 万点ほど平均得点が下がっている．

同程度の学習局面数において比較してみると，例えばプレイヤ Gr 25%/Ex 75% の場合 2.5×10^{10} 局面を学習した結果 3-ply Expectimax による評価

で平均 408,619 であるが，これはベースラインにおいて同数の局面を学習した結果 414,440 に至っていない．

7. 考察

本研究では，貪欲な自己プレイに基づく強化学習を開始点とし，(1) 良いプレイのみの学習，(2) 自己プレイへのランダム性の導入，(3) 自己プレイへの探索の導入 の 3 つの変更を試行した．実験の結果，これらの試行はいずれも不成功に終わったが，そのことは「2048」の特性だけでなく学習手法に関していくつかの重要な示唆を与えると考える．

特に 1 人ゲームの場合では，そのゲームの探索空間を調べる際により有望なところに計算資源を集中することがよく行われる．これはモンテカルロ法など一般にうまくいっているが，「2048」のように対象とするゲームにランダム要素がある場合には必ずしもうまくいかないことが分かった．著者らは別の研究 [4] において，既存の「2048」プレイヤのプレイログをもとに教師あり学習を適用したが，その予備実験において平均得点の高かったゲームを選んで学習を行っても良い結果が得られなかった．本研究の結果は，その結果と一貫性がある．

第 5 節で述べたように，貪欲な自己プレイの結果のみから強化学習を行った場合の問題点のひとつに，自己プレイが偏ってしまうことにより未知の局面に適切に対応できないことがある．本研究では，プレイにランダム性を導入する単純な手法のひとつである ϵ-Greedy 法を適用したが，1%というわずかなランダム性を入れた場合でも貪欲な自己プレイからの学習に劣る結果となった．この理由として以下のふたつが考えられる．

- もともと「2048」のゲームでは出現するタイルにランダム性がある．そのため，貪欲な自己プレイに基づく学習でも十分多くの局面を列挙できており，結果として上記の問題が発生しなかった．
- 本研究では N タプルネットワークによる評価関数を用いており，それぞれの N タプルでは盤面の一部分のみを参照している．貪欲な自己プレイにおいて列挙された局面は偏っていたとしても，その部分盤面は十分網羅されていた．

しかし，これらの理由は ϵ-Greedy により学習が改善しないことは示唆するが，本研究の結果のようにランダム性を導入することで結果が大きく悪化することの理由とはならない．その解明には，より深く原因の調査を行うことが必要である．

最後に，自己プレイにおいて Expectimax 探索を導入して，より良いプレイを学習する試行についても良い結果が得られなかった．自己プレイと学習に用いた時間をそろえた評価だけでなく，学習局面数をそろえた評価においても，(2-ply) Expectimax 探索を行うことで性能向上が得られなかった．この理由として以下の 3 つが考えられる．

- 自己プレイに Expectimax 探索を使用した場合では，自己プレイに貪欲法を使用した場合に比べてより良い (長い) プレイとなることが予想されたため，本研究では Expectimax 探索の際にはリスタート方策を用いなかった．リスタート方策なしのベースラインとの比較では，3-ply Expectimax による評価が向上しており，適切なリスタート方策を併用することで改善する可能性がある．
- 貪欲な自己プレイの場合に比べて，Expectimax 探索による自己プレイでは，学習データの見た目の一貫性が低下する可能性がある．本研究で用いた N タプルネットワークでは，局所的な盤面情報をもとに値を計算するため，そのような一貫性の欠如が学習を悪化させせか可能性がある．
- 第 2 節で述べたように，「2048」のゲームにおいて最善手と次善手が同じくらいの良さとなる局面が多いと著者は考える．そのような場合に，その小さな差が Expectimax 探索により必要以上に広がってしまった可能性もある．

これらの理由のうち，その 2 つ目は学習手法の問題，3 つ目は学習の対象の問題である．本研究でうまくいかなかった原因を探ることによりそれらの問題提起を行うことが重要であると考えるが，そのためにはより深い調査が必要である．

8. 関連研究

「2048」に対するコンピュータプレイヤにおいて，N タプルネットワークと強化学習を組み合わせる手法が有用であることが様々な研究により示されている．そのアイデアは，Szubert と Jaśkowski [14] により最初に提案された．その後 Wu ら [17] により，N タプルネットワークがマルチステージ化を用いて拡張された．Yeh ら [19] は，このアイデアに基づき，3 ステージの N タプルネットワークを評価関数とした場合，貪欲プレイにより平均 143,958 点クリア率 96.6%，3-ply Expectimax 探索により平均 350,394 点クリア率 99.9% を達成した．また，より多くのステージと人手で設計した評価項目とを併用した場合，3-ply Expectimax 探索にて平均 443,526 点を達成した．著者らが知る限りの最も良いコンピュータプレイヤは，Jaśkowski によるもの [6] である．そこでは，Temporal Coherence 学習，マルチステージ化，冗長な N タプルなどの手法が適用され，3-ply Expectimax により平均 511,759 点，1 手 1 秒の制限時間のもとで平均 609,104 点を達成した．本研究で用いたベースラインは著者による先行研究 [8] としたが，それは Jaśkowski の結果を踏まえ学習方法を少し改良することで少しの性能向上を達成したものである (ただし，メモリ使用量の制限から使用した N タプルの個数やステージ数が少ないため，最終的な平均得点は劣る)．

これらの研究とは独立して，評価関数として用いる N タプルの系統的な選択法について，Oka と Matsuzaki [7,10] による研究がある．著者らは，Matsuzaki [7] により選択された N タプルを用いて，「2048」および「対戦型 2048」[21] プレイヤの強さを評価した [23]．

一方，「2048」に対して，多層ニューラルネットワークを用いた評価関数を用いることは，これまであまり研究されていない．著者らが知る限り，論文として報告されているものは Gui によるもの [5] だけである．著者らは，Gui によるニューラルネットワークを拡張した「2048」プレイヤの開発にも取り組んでいる [22]．論文として報告はされていないが，オープンソースとして公開されているプログラムには，ニューラルネットワークに基づくもの [15]，深層 Q 学習を用いるもの [16]，深層リカレントニューラルネットワークによる Q 学習を用いるもの [12] などがある．このうち，プログラム [15] は，クリア率 94.152% を達成したと報告している．

プログラム [15] では，学習にあたって，ヒューリスティックに基づく評価関数と Expectimax 探索を用いたプレイヤ nneonneo (平均得点 387,222) [18] のプレイを教師データとして用いる教師あり学習を行っている．その結果，上記のとおり良い結果が得られている．一方，著者らが取り組んだニューラルネットワークプレイヤの場合は，本研究で用いたベースラインプレイヤの Expectimax 探索プレイの結果を教師データとして用いる教師あり学習を行った．その結果は，文献 [22] で示すようにあまり良いものではなかった．これらの違いがニューラルネットワークの構成によるものか，学習データの特性によるものかについて調査することは今後の課題である．

9. まとめ

「2048」に対するコンピュータプレイヤにおいて，N タプルネットワークと貪欲な自己プレイに基づく強化学習を組み合わせる手法が有用であることが様々な研究により示されてきた．本研究では，そのような貪欲な自己プレイに基づく強化学習に改善の余地があるのではないかと考え，(1) 良いプレイのみの学習，(2) 自己プレイへのランダム性の導入，(3) 自己プレイへの探索の導入 の 3 つの変更を試行した．実験の結果，いずれの試行もベースラインよりも悪い結果となり，不成功に終わった．

実験結果は不本意ではあったものの，本研究ではそのような結果が得られた理由について考察を行った．それらの考察で導かれる結論と確かめるにはより深い実験調査が必要となるが，それらの解明およびその結果による強化学習の改善が今後の課題である．

謝辞

本稿に掲載した研究成果は，高知工科大学のIACP クラスタ計算機を使用して得られたものである．

参考文献

[1] Beal, D. F. and Smith, M. C.: Temporal coherence and prediction decay in TD learning, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Vol. 1, pp. 564–569 (1999).

[2] Chabin, T., Elouafi, M., Carvalho, P. and Tonda, A.: Using Linear Genetic Programming to Evolve a Controller for the game 2048, `http://www.cs.put.poznan.pl/wjaskowski/pub/2015-GECCO-2048-Competition/Treecko.pdf` (2015).

[3] Dedieu, A. and Amar, J.: Deep Reinforcement Learning for 2048 (2017). Available online: `http://www.mit.edu/~adedieu/pdf/2048.pdf`.

[4] Fujita, R. and Matsuzaki, K.: Improving 2048 Player with Supervised Learning, *Proceedings of 6th International Symposium on Frontier Technology*, pp. 353–357 (2017).

[5] Guei, H., Wei, T., Huang, J.-B. and Wu, I.-C.: An early attempt at applying deep reinforcement learning to the Game 2048, *the Workshop Neural Networks in Games in the International Conference on Computers and Games (CG 2016)* (2016).

[6] Jaśkowski, W.: Mastering 2048 with Delayed Temporal Coherence Learning, Multi-Stage Weight Promotion, Redundant Encoding and Carousel Shaping, *IEEE Transactions on Computational Intelligence and AI in Games* (2017).

[7] Matsuzaki, K.: Systematic Selection of N-tuple Networks with Consideration of Interinfluence for Game 2048, *Proceedings of the 2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2016)* (2016).

[8] Matsuzaki, K.: Developing 2048 Player with Backward Temporal Coherence Learning and Restart, *Proceedings of Fifteenth International Conference on Advances in Computer Games (ACG2017)* (2017).

[9] Matsuzaki, K.: Evaluation of Multi-staging and Weight Promotion for Game 2048, Technical Report KUTBTR2017-001, Kochi University of Technology (2017).

[10] Oka, K. and Matsuzaki, K.: Systematic Selection of N-tuple Networks for 2048, *Proceedings of 9th International Conference on Computers and Games (CG2016)*, pp. 81–92 (2016).

[11] Rodgers, P. and Levine, J.: An Investigation into 2048 AI Strategies, *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–2 (2014).

[12] Samir, M.: 2048 Deep Recurrent Reinforcement Learning. `https://github.com/georgwiese/2048-rl`.

[13] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D.: Mastering the game of Go with deep neural networks and tree search, *Nature*, Vol. 529, pp. 484–489 (2016).

[14] Szubert, M. and Jaśkowski, W.: Temporal Difference Learning of N-Tuple Networks for the Game 2048, *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8 (2014).

[15] tjwei: A Deep Learning AI for 2048. `https://github.com/tjwei/2048-NN`.

[16] Wiese, G.: 2048 Reinforcement Learning. `https://github.com/georgwiese/2048-rl`.

[17] Wu, I.-C., Yeh, K.-H., Liang, C.-C., Chang, C.-C. and Chiang, H.: Multi-Stage Temporal Difference Learning for 2048, *Technologies and Applications of Artificial Intelligence*, pp. 366–378 (2014).

[18] Xiao, R.: nneonneo/2048-ai, `https://github.com/nneonneo/2048-ai` (2015).

[19] Yeh, K., Wu, I., Hsueh, C., Chang, C., Liang, C. and Chiang, H.: Multi-stage temporal difference learning for 2048-like games., *IEEE Transactions on Computational Intelligence and AI in Games* (2016).

[20] Zaky, A.: Minimax and Expectimax Algorithm to Solve 2048, `http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/Makalah2014/MakalahIF2211-2014-037.pdf` (2014).

[21] 寺田　実：対戦型 2048，情報処理学会 夏のプログラミング・シンポジウム [2015] 報告集，pp. 19–22 (2015).

[22] 近藤直季，松崎公紀：2048 におけるニューラルネットワークプレイヤの育成報告，第 59 回プログラミングシンポジウム予稿集 (2018).

[23] 岡　和人，松崎公紀：システム的選択による N-tuple networks の”対戦型 2048”への適用，第 58 回プログラミングシンポジウム予稿集 (2017).