確率的ゲーム 2048 の強化学習の研究

山下修平

2024 年 1 月 5 日

目次

第 1 章

はじめに

近年の深層学習および深層強化学習の技術の進歩はめざましい．AlphaZero 一方で囲碁や将棋に代表される二人零和有限完全確定情報ゲームや研究の盛んな Atari は， いずれも環境のダイナミクスが決定的なゲームであり， ランダム性は介入しない．2048 は確率的ゲームである．

第 2 章

2048

2.1　2048 のルールと用語説明

2048 は， Gabrirle Cirulli によって公開された 1 人用のパズルゲームである [1]．ゲームは 16 マスからランダムに選ばれた 2 マスに 2 か 4 の数字タイルが置かれた盤面から始まる．プレイヤが行うことは上下左右いずれかの方向を選択することである．プレイヤがある 1 つの方向を選ぶと， 盤面上のすべての数字タイルは選択した方向に向かってスライドして移動する．スライドする数字タイルは空きマスを通過し， 異なる数字タイルの直前か盤面の端で停止する．スライドして移動する際に 2 つの同じ数字のタイルが衝突すると， これらは合体してその合計の数字の 1 つのタイルへ変化し， プレイヤはその数値を得点として獲得する．そのため， ゲームには 2 の累乗の数字タイルしか現れない．図 2.1 にある盤面から上下左右を選択したときの， 数字タイルのスライドの仕方の具体例を示す．

数字タイルのスライド後， 空きマスから等確率に選択されたある 1 マスに 90% の確率で 2 のタイルが， 10% の確率で 4 のタイルが置かれる．ゲームはプレイヤの行動による数字タイルのスライドと新たな数字タイルの出現を交互に繰り返して進行する．盤面上の数字タイルが市松模様のようになると， プレイヤが選択可能な行動がなくなったときにゲームは終了する (図 2.2 を参照)．

ここでプレイヤが行動を選択する盤面を**状態**， 行動を選択して新たな数字タイルが出現する直前の盤面を *afterstate* と呼ぶ．またプレイヤがゲームを開始する盤面を初期状態， ゲームが終了した盤面を終了状態と呼ぶ．図 2.4 に状態 s から afterstate s' を経由して， 次の状態 s_{next} に遷移する例を示す．

プレイヤの一般的な目標はゲームのタイトルが示す $2^{11} = 2048$ のタイルを完成させることだが， それ以降もゲームを続けることができる．

2.2　ゲームの進行上の性質

2048 はゲームの性質上， 状態から afterstate への遷移において盤面上の数字タイルの合計値は不変である (図 2.1 を参照)．盤面上の数字タイルの合計値は afterstate から次の状態への遷移においてのみ変

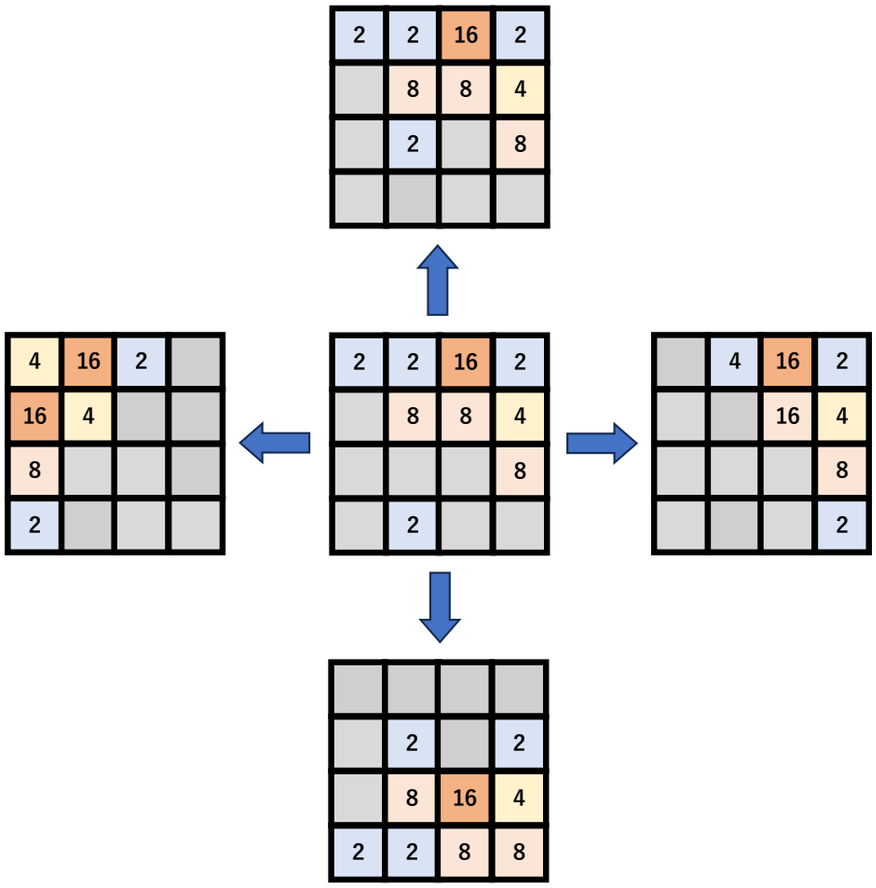


図 2.1: 上下左右それぞれへのスライドの例

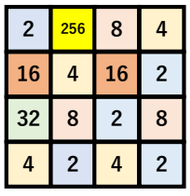


図 2.2: 終了状態の例

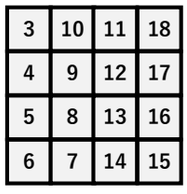


図 2.3: 最大到達盤面の例 (数字は指数部のみ)

化する．新しい数字タイルとして 2 か 4 のタイルが出現することで， 数字タイルの合計値はその値の分だけ必ず増加する．すなわちプレイヤが 1 回行動するたびに， 盤面上の数字タイルの合計値は 2 か 4 ずつ単調に増加する．

よって盤面上の数字タイルの合計値をゲームの進行度合いとして用いることができる．以降これを**時刻**と呼ぶ．例えば図 2.4 の遷移では時刻 $2 \times 4 + 4 + 8 \times 3 + 16 = 52$ の状態 s が時刻 $52 + 2 = 54$ の状態 s_{next} に遷移している．ゲームの時刻はプレイヤが行動するたびに必ず増加するため， 2048 はサイクル

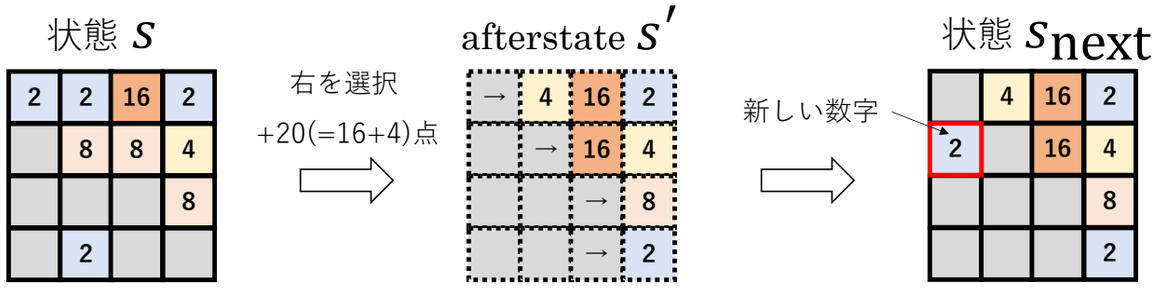


図 2.4: 状態遷移の例

の出現しないゲームであることがわかる．

また 2048 は必ず終了するゲームである．2^n のタイルを 1 つ完成させるには，盤面上に 2 つの 2^{n-1} のタイルを同時に存在させる必要がある．2 つの 2^{n-1} のタイルを同時に存在させるには，1 つの 2^{n-1} のタイルと 2 つの 2^{n-2} を同時に存在させる必要がある．新しい数字タイルとして 2 と 4 が出現することを踏まえると，帰納的に考えて 2^n のタイルを完成させるには最小でも $n-2$ マスを必要とすることが分かる．よって 16 マスの 2048 では最高でも 2^{18} のタイルまでしか完成させられず，運や選択手に関わらずゲームを永久に続けることはできないことが分かる．

第 3 章

2048 と強化学習

これまでに 2048 を対象とした強化学習の研究は数多くなされてきた．本章では強化学習の概要，および 2048 に対する強化学習の先行研究について記述する．

3.1 強化学習の概要

まず本節では 2048 との関係を踏まえつつ，一般的な強化学習の概要について記述する．なお本節の内容は全体に文献 [2] を参照して書かれた．

3.1.1 マルコフ決定過程

強化学習は与えられた環境において試行錯誤することを通して，目標を達成するための戦略や意思決定を学習するための手法である．学習や意思決定を行う主体はエージェントと呼ばれる．エージェントは離散タイムステップに従って行動を選択し続け，環境とやり取りを行う．

このような問題設定はマルコフ決定過程 (MDP) というモデルによって定式化されている．MDP は以下の 4 つの要素で構成される．

- 状態集合 $\mathcal{S}$
- 行動集合 $\mathcal{A}$
- 状態遷移関数 $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$
- 報酬関数 $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$

エージェントはステップ t で状態 $S_t \in \mathcal{S}$ から行動 $A_t \in \mathcal{A}$ を選択する．そして確率 $p(S_{t+1}|S_t, A_t)$ で次の状態 S_{t+1} に遷移し，$R_{t+1} = r(S_t, A_t, S_{t+1})$ を即時報酬として獲得する．状態遷移関数と報酬関数は環境のダイナミクスと呼ばれることがある．図 3.1 に MDP の概念図を示す．

状態集合と行動集合が有限である MDP を有限 MDP と呼ぶ．2048 は有限 MDP にそのまま当てはま

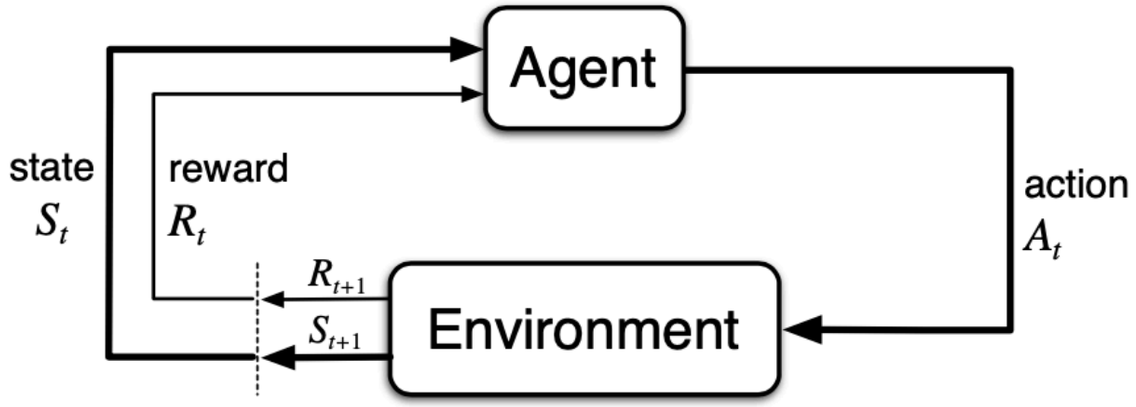


図 3.1: MDP の模式図 (文献 [2] より引用)

るゲームである．行動集合 A はプレイが選ぶ上下左右に対応し， 報酬はプレイヤが獲得する得点に直接対応する．

一般に強化学習で扱う問題には, エージェントと環境のやり取りが終わる終了状態が存在する episodic task と終了状態が存在しない continuing task が存在する．episodic task ではエージェントと環境のやり取りを初期状態から終了状態までのエピソードと呼ばれる単位で分割することができる．2.2 節で説明したように 2048 は必ず終了するゲームであるため， 以降 episodic task での定義を確認する．

3.1.2　方策と価値関数

エージェントがある状態において行動を決定する際の戦略， すなわち確率分布 $\pi : S \times A \to [0, 1]$ を方策と呼ぶ．状態価値関数 $v_\pi(s)$ は状態 s から方策 π に従って行動を選択し続けた場合の累積報酬和の期待値であり， 次のように定義される．

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[\sum_{k=0}^{T} R_{t+k+1} | S_t = s \right] \tag{3.1}$$

同様に状態 s から行動 a を選択し， その後方策 π に従って行動を選択し続けた場合の累積報酬和の期待値である行動価値関数 $q_\pi(s, a)$ の定義は以下のようになる．

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[\sum_{k=0}^{T} R_{t+k+1} | S_t = s, A_t = a \right] \tag{3.2}$$

強化学習の目標は多くの報酬を獲得できるような良い方策を見つけることである．価値関数の定義より 2 つの方策 π と π' があるとすると， すべての状態 $s \in S$ について $v_\pi(s) \geq v_{\pi'}(s)$ が成り立つならば π は π' と等価か π' よりも良い方策だと言える．ここで他のすべての方策と比べて等価であるか， それよ

りも良い方策が少なくとも 1 つ存在する．これは最適方策 π_* と呼ばれる方策である．π_* に従うときの状態価値関数は最適状態価値関数と呼ばれ， v_* で表される．同様に π_* に従うときの行動価値関数は最適行動価値関数と呼ばれ， q_* で表される．それぞれの具体的な定義を式 3.3， 3.4 に示す．

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{for all } s \in S \tag{3.3}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s) \tag{3.4}$$

このとき $v_*(s) = \max_{a \in A(s)} q_*(s,a)$ であるから， 以下の式が導かれる (図 3.2 を参照) ．

$$v_*(s) = \max_{a \in A(s)} q_*(s,a) \tag{3.5}$$

$$= \max_{a \in A(s)} \mathbb{E}[R_{t+1} + v_*(S_{t+1}) | S_t = s, A_t = a] \tag{3.6}$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + v_*(S_{t+1}) | S_t = s, A_t = a] \tag{3.7}$$

$$= \mathbb{E}[R_{t+1} + \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \tag{3.8}$$

式 3.6， 3.8 はベルマン最適方程式と呼ばれる．

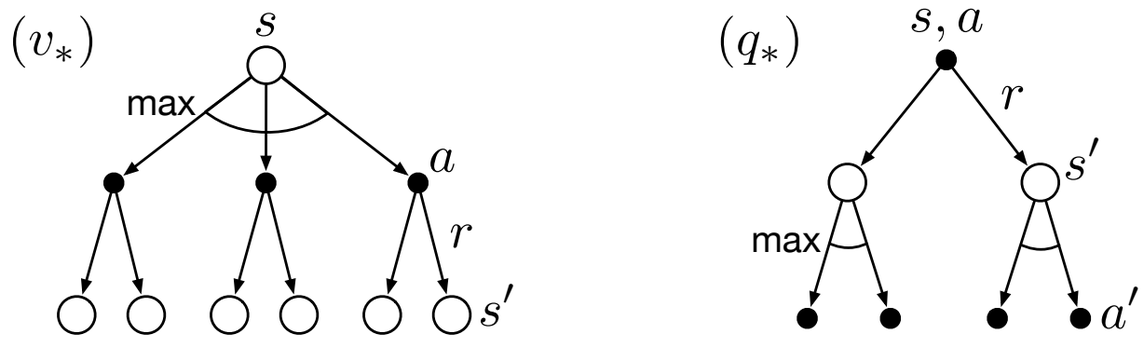


図 3.2: 最適価値関数のバックアップ図 (文献 [2] より引用)

3.1.3 価値ベースな手法

状態 s における最適方策に従った具体的な行動は $\arg\max \mathbb{E}[R_{t+1} + v_*(S_{t+1}) | S_t = s, A_t = a]$ と， 1 ステップ先の状態を探索してそれらの最適状態価値関数を参照することで計算できる．最適行動価値関数が参照できれば， 1 ステップ先の状態を探索する必要すらなく， 単に $\arg\max q_*(s,a)$ を選択すればよい．最適方策を得るために価値関数を学習する手法は価値ベースな手法と呼ばれる．

Q 学習は最も基本的な価値ベースな手法の 1 つで， 最適行動価値関数 q_* の推定値 Q を得られた経験から学習する．状態 S_t から行動 A_t を選択し， 報酬 R_{t+1} を獲得して次の状態 S_{t+1} に遷移したとする．

このとき Q 学習は以下の更新式 3.9 に従って $Q(S_t, A_t)$ を更新する．

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3.9}$$

式 3.9 は現在の推定値 $Q(S_t, A_t)$ を，ベルマン最適方程式 3.8 の右辺の推定値である $R_{t+1} + \max_a Q(S_{t+1}, a)$ に近づけていると解釈できる．任意の $(s, a) \in \mathcal{S} \times \mathcal{A}$ について $Q(s, a)$ が更新され続けるという条件の下で，Q 学習は収束が保証されている．

状態集合や行動集合が大きい場合や有限でない場合には，テーブル形式で Q 値を保持することができないため何らかの関数近似を行う必要がある．近年では深層学習 [3] の研究の発展により，関数近似の方法としてニューラルネットワークが用いられることが多い．価値関数や方策をニューラルネットワークで近似する手法は，まとめて深層強化学習と呼ばれる．Q 値をニューラルネットワークで近似する，Deep Q Network (DQN) [4] は深層強化学習の先駆けとして有名な手法である．DQN の発表以降，様々な工夫が提案され続けている．詳細は文献 [5] の 4 節などを参照されたい．

3.1.4　方策ベースな手法

方策を直接パラメタライズされた関数で近似し，パラメータを学習する手法を全般に方策ベースな手法と呼ぶ．なお方策ベースな手法においても価値関数の学習を行うことはあるが，良い方策のパラメータを学習するために必要とするものであり，直接的な行動の選択には関与しない．

方策のパラメータ $\boldsymbol{\theta}$ は目的関数 $J(\boldsymbol{\theta})$ を最大化するように更新される．$J(\boldsymbol{\theta})$ は，方策 $\pi_{\boldsymbol{\theta}}$ に従ったときの初期状態 s_0 の価値 $v_{\pi_{\boldsymbol{\theta}}}(s_0)$ と定義することができる．このとき方策 π のもとで状態 s を訪れる確率を $\mu(s)$ とすると，$\nabla J(\boldsymbol{\theta})$ は以下の方策勾配定理によって求めることができる．

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \boldsymbol{\theta}) \tag{3.10}$$

さらに以下のように変形する．

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \boldsymbol{\theta}) \\
&= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \boldsymbol{\theta}) \right] \\
&= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \right]
\end{aligned} \tag{3.11}$$

よって学習率を α とすると， パラメータ $\boldsymbol{\theta}$ の更新式が導かれる．

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \left[q_\pi(S_t, A_t) \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \right] \tag{3.12}$$

ただし未知の値 $q_\pi(S_t, A_t)$ を， 学習データから推定する必要がある．Williams [6] は $q_\pi(S_t, A_t)$ の推定値として， S_t から A_t を選択して実際に獲得した報酬の総和 $G_t = \Sigma_{k=0}^{T} R_{t+k+1}$ を用いることを提案した．G_t は $q_\pi(S_t, A_t)$ の不偏推定量としての性質をみたす一方で， 分散が大きく学習が安定しない．またパラメータの更新をエピソードが終了するまで行うことができない．

ここで式 3.10 において， 行動 a に依存しない定数を $b(s)$ として $\Sigma_a b(s) \nabla \pi(a|s, \boldsymbol{\theta}) = b(s) \nabla \Sigma_a \pi(a|s, \boldsymbol{\theta}) = 0$ に注目すると， 以下の式 3.13 が成り立つ．

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s, a) - b(s) \right) \nabla \pi(a|s, \boldsymbol{\theta}) \tag{3.13}$$

$b(s)$ として s の価値関数 $v_\pi(s)$ を考えることが一般的である．$q_\pi(s, a) - v_\pi(s)$ は， 状態 s で行動 a をとることが他の行動をとることに比べて， どれくらい良いかを示す値で一般的にアドバンテージと呼ばれる．よってパラメータ更新式 3.12 中の $q_\pi(S_t, A_t)$ の代わりに， アドバンテージ $q_\pi(S_t, A_t) - v(S_t)$ を推定する．そのために方策関数の学習と同時に， 価値関数 v_π の推定値 V を学習する．$q_\pi(S_t, A_t) = \Sigma_{k=0}^{n-1} R_{t+k+1} + v_\pi(S_{t+n})$ であるから， 以下の方策パラメータ $\boldsymbol{\theta}$ の更新式 3.14 が導かれる．

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \left[\sum_{k=0}^{n-1} \big(R_{t+k+1} + V(S_{t+n}) - V(S_{t+n}) \big) \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \right] \tag{3.14}$$

ここで $\Sigma_{k=0}^{n-1} R_{t+k+1} + v_\pi(S_{t+n})$ は n ステップリターンと呼ばれる．n をエピソードの終了時刻 T とすると， n ステップリターンは G_t に一致する．このように学習した価値関数を行動の良さを評価するために用いる手法は， 全般に actor-critic な手法と呼ばれる．

Mnih ら [7] が提案した asynchronous advantage actor-critic (A3C) は，方策関数と価値関数をニューラルネットワークにより近似した actor-critic な手法で， Atari や物理シミュレーションの実験で高い成果を収めた．A3C の発表以降，方策ベースな深層強化学習の研究は数多くなされている．特に Schulman らが提案した Proximal Policy Optimization (PPO) [8] は有名手法の 1 つである．

3.1.5 AlphaZero と MuZero

Silver らが提案した AlphaZero [9] は， 囲碁やチェスなどの二人零和完全確定情報ゲームを対象とした有力な深層強化学習手法である．AlphaZero は盤面の特徴量を入力として， その盤面における方策と価値を出力するニューラルネットワークを持つ．これをモンテカルロ木探索 (MCTS) による自己対戦を通して得たデータから学習する．そのため人間の棋譜などの教師データを使うことなく， 囲碁， 将棋，チェスにおいて当時の有力なプログラムを上回る強さを示して注目を集めた．

AlphaZero は MCTS を行うにあたって， ゲームのルールや環境のダイナミクスが既知であることを前提としている．すなわち現在の状態 s_t から行動 a_t を選択した後の状態 s_{t+1} を， s_t のみ観測している状況から知ることができる必要がある．そのため AlphaZero は複雑なダイナミクスを持つ Atari などのゲームには適用することが難しい．Schrittwieser らが提案した MuZero [10] は， ニューラルネットワークで環境のダイナミクスをモデル化し， 学習することを提案した．MuZero は二人零和完全確定情報ゲームにおいて AlphaZero と同等の強さ， Atari において当時の有力手法と同等以上の性能を示した．3.2.3 節で MuZero を確率的な環境に対応できるように拡張した， Stochastic MuZero について述べる．そこで MCTS の具体的な方法や， ニューラルネットワークの学習方法について述べる．

3.2　2048 への強化学習の応用

3.1 節では強化学習一般の概要について説明した．これを踏まえて本節では， 2048 を対象とした強化学習研究の先行研究について概説する．

3.2.1　TD afterstate 学習

Szubert ら [11] は TD afterstate 学習と呼ばれる価値ベースの強化学習手法を提案した．2048 はゲームの性質上， 状態 s から行動 a をとって獲得する得点 $r(s,a)$ と遷移する afterstate s' は決定的である．よって afterstate s' の最適価値を $v'_*(s')$ と表すと， 行動価値 $q_*(s,a)$ は $q_*(s,a) = r(s,a) + v'_*(s')$ と分解できる．そこで TD afterstate 学習は， Q 値ではなく afterstate の価値 v'_* の推定値 V' を式 3.15 に従って更新する．

$$V'(S'_t) \leftarrow V'(S'_t) + \alpha[R_{t+1} + V'(S'_{t+1}) - V'(S'_t)] \tag{3.15}$$

式 3.15 は Q 学習の更新式 3.9 において， $Q(S_t, A_t) = R(S_t, A_t) + V'(S'_t)$ として書き直したものと見ることができる．

Szubert らは N-tuple ネットワークというヒューリスティックな盤面評価関数を用いて V' を表現し，2048 のタイルを 98% の確率で到達させることに成功した．Jaskowski [12] はこれに様々な工夫を加えた学習方法を提案し， 最終的に expectimax 探索を 1 手 1 秒の制限で行うことで平均 609, 104 点を獲得するプレイヤを開発した．松崎 [13] はニューラルネットワークによって表現した V' を学習させ， 3-ply expectimax 探索を行うことで平均 406, 927 点を達成した．

Szubert らは文献 [11] で N-tuple ネットワークを用いた Q 学習の実験も行ったが， TD afterstate 学習の性能を大きく下回る結果になったことを報告している．TD afterstate 学習は観測している状態 s から afterstate s' への遷移までをシミュレートする， いわば 0.5 手読みの探索を行っているため， 0 手読みの Q 学習と比べて学習しやすいと考えられる．

3.2.2 Afterstate PPO

3.2.1 節では， 価値ベースな手法である TD afterstate 学習が 2048 において主流な強化学習手法の 1 つであることを説明した．一方で， 2048 において目立った方策ベースな強化学習手法は山下ら [14] は PPO [8] を 2048 に工夫して適用する， Afterstate PPO という手法を提案した．すなわち状態 s から行動 a をとったときの afterstate をニューラルネットワークの入力とし， その出力を行動 a の選択確率のロジットとする．通常の PPO では全く学習が進まなかった一方で， Afterstate PPO は学習後 2048 のタイルを平均約 50% の割合で完成させられることを示した．

3.2.3 Stochastic MuZero

Antonoglou らが提案した Stochastic MuZero [15] は 3.1.5 節で述べた MuZero を， 確率的な環境にも対応できるように拡張した手法である．Stochastic MuZero は確率的ゲームである 2048 とバックギャモンで MuZero を大きく上回るパフォーマンスを発揮した．さらにランダム性のない囲碁においても MuZero と同等のレーティングを達成し， 対象とするドメインが幅広いことを示した．2048 においては， 3.2.1 節で述べた TD afterstate 学習の先行研究 [12] と比較して， 一切のドメイン知識を必要とせず， より優れたパフォーマンスを出したことが強調された．一方で， 論文内で示された Stochastic MuZero の最終的な平均スコアは約 50 万点で， expectimax 探索を行うことで平均 $609,104$ 点を達成した文献 [12] の結果を明確に上回るとはいえない．

Stochastic MuZero は方策・価値ニューラルネットワークに加えて， 環境のダイナミクスモデルをニューラルネットワークにより推論する．これらのニューラルネットワークを， モンテカルロ木探索 (MCTS) によるプランニングを行って得た学習データを用いて訓練する．ここでは Stochastic MuZero の MCTS の方法， および方策・価値ニューラルネットワークの学習方法について説明する．環境のダイナミクスモデルの学習方法については論文 [15] の 4 節を参照されたい．

Stochastic MuZero のモンテカルロ木探索

Stochastic MuZero は MCTS において， *decision* ノードと *chance* ノードという 2 種類のノードから成る探索木を用いる．decision ノードから chance ノードへの遷移は決定的で， chance ノードから decision ノードへの遷移は確率的に行われる．すなわち 2048 の文脈では， decision ノードはプレイヤが行動を選択する状態に対応し， chance ノードは afterstate に対応する．そのため decision ノードの子は chance ノードで， chance ノードの子は decision ノードである．

探索木中のそれぞれのノード s は， $N(s), W(s), Q(s), P(s), R(s)$ という 5 つの統計量を管理する．

- $N(s)\cdots$ 探索中に s を訪問した回数

- $W(s)\cdots$ s を根とする部分木の価値の推定値の合計
- $Q(s)\cdots$ s を根とする部分木の価値の推定値の平均 $(W(s)/N(s))$
- $P(s)\cdots$ s の親から s への遷移確率
- $R(s)\cdots$ s の親から s へ遷移するときに獲得する即時報酬

MCTS は選択， 評価， 逆伝播， 展開の 4 つのステップを繰り返すことで， 探索木を大きくする．この 4 つのステップはまとめてシミュレーションと呼ばれる．すなわちシミュレーションを繰り返すことで深い探索を行い， 良い行動を選択することができる．以下ではそれぞれのステップについて説明する．

■選択　現在のゲーム木の根ノード s_0 (decision ノード) から有望な子ノードを選択し， これを葉ノード s_L に至るまで辿り続ける．decision ノード s_d においては， 以下の PUCT 式 3.16 が最大となるような子 chance ノード s_c を選択する．

$$Q(s_c) + C(s_c)P(s_c)\frac{\sqrt{1+N(s_d)}}{1+N(s_c)} \tag{3.16}$$

ただし $C(s_c)$ は探索率を表す．これは現段階での価値の推定値 $Q(s_c)$ と探索項 $C(s_c)P(s_c)\sqrt{1+N(s_d)}/(1+N(s_c))$ を合計したものである．$N(s_c)$ が小さい子ノードほど探索項は大きな値を示す．一方 chance ノード s_c においては， 単にそれぞれの子 decision ノード s_d への遷移確率 $P(s_d)$ に従って確率的に選択する．

■評価 (図 3.3b)　選択のステップの葉ノード s_L をニューラルネットワークにより評価する．s_L が decision ノードである場合には， s_L における方策 $\pi(\cdot|s_L)$ および s_L の価値を v_L を得る．s_L が chance ノードである場合には， s_L の価値 v_L のみを得る．

■展開 (図 3.3d)　s_L が decision ノードである場合には， s_L に対応する状態から遷移可能な afterstate を s_L の子 chance ノードとして探索木に追加する．それぞれの子 chance ノード s_c について， $P(s_c) = \pi(s_c|s_L)$ とする．

s_L が chance ノードである場合には，s_L に対応する afterstate から遷移可能な状態を s_L の子 decision ノードとして探索木に追加する．それぞれの子 decision ノード s_d について， $P(s_d)$ は s_L に対応する afterstate から s_d に対応する状態へ遷移する環境の確率とする．

■逆伝播 (図 3.3c)　s_L について， $N(s_L,a)=0, W(s_L,a)=0, Q(s_L,a)=0$ と初期化する．さらにさらに s_0 から s_L に至る各ノード s について， 以下のように各統計量の値を更新する．

$$\begin{aligned} N(s) &\leftarrow N(s)+1 \\ W(s) &\leftarrow W(s)+v_L \\ Q(s) &\leftarrow \frac{W(s)}{N(s)} \end{aligned}$$

行動の決定方法

一定回数のシミュレーションを行った後に， 実際に選ぶ行動を決定する．根ノード s_0 のそれぞれの子 chance ノード s_c について， $N(s_c)^{1/\tau}$ に比例した確率で子ノードを選択する．選択した子ノードに対応する afterstate に遷移するような行動をとる．τ は温度パラメータと呼ばれ， 0 に近づくほど訪問回数に関して貪欲な選択をするようになる．多様な学習データを集めるために学習の序盤では τ を大きく設定し， 学習が進むについて 0 に近づける．

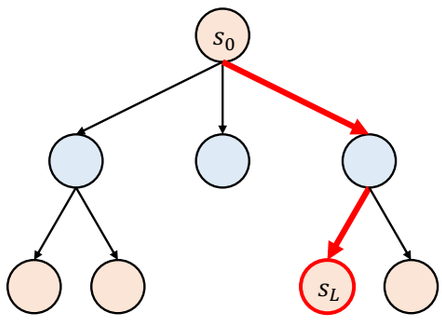
ニューラルネットワークの学習

MCTS によるプレイによって得たデータを使ってニューラルネットワークを訓練する．各訪問回数を正規化した $\boldsymbol{\pi}$ すなわち式 3.17 で表される誤差関数 l を最小化するようにパラメータを更新する．

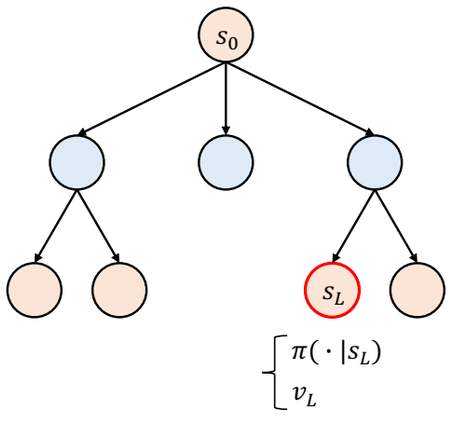
$$l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \mathbf{p} \tag{3.17}$$

ただし実際の価値関数の学習においては付録 A.3.1 節で説明するように， 最小二乗誤差ではなく Cross Entropy 誤差を最小化するようにパラメータを更新する．

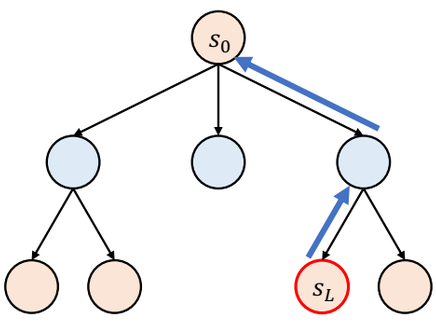
3.2.4　先行研究の考察



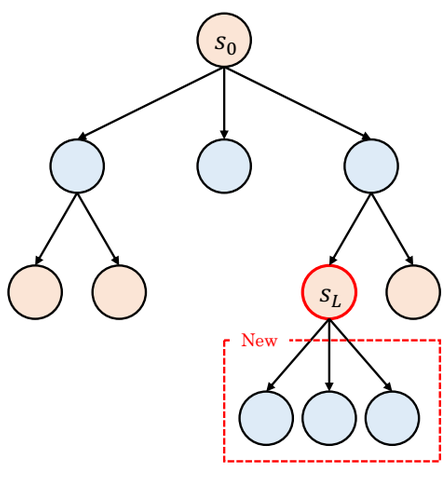
(a) 選択



(b) 評価



(c) 逆伝播



(d) 展開

図 3.3: MCTS の各ステップ (オレンジ色のノードは decision ノード，青色のノードは chance ノード)

第 4 章

提案手法

3.2 節で述べたように， 2048 を対象とした強化学習の研究は数多くなされてきた．もしゲームが完全に解かれていれば， 強化学習手法の良し悪しを定量的に評価することができる．一方で 2048 はゲーム木の大きさから， 完全解析を実行することは計算資源の観点から困難である．そこで本研究では 2048 のミニゲームの完全解析を行うことを提案する．さらに解析したミニゲームをベンチマークとして， 2048 の強化学習手法について詳細に検討する．

4.1 2048 のミニゲームの完全解析

3 章で述べた強化学習は環境 (ゲーム) と何度もやり取りすることで， 最適な方策を学習するための手法である．一方で小さなゲームであれば， 力ずくの計算によってゲームを完全に解くこともできる．本節では 2048 を解析的なアプローチによって解くことについて述べる．なお本節の内容は文献 [16] および文献 [17] を元に執筆された．

4.1.1 2048 の完全解析とは

2048 は 1 人用のゲームであるため， 勝敗のようなプレイヤの明確な目標は存在しない．そのためプレイヤが何を目標とするかによって， プレイヤの最善手の定義は変化する．また 2.1 節で述べたようにゲームはランダム性を伴うため， 同じ状態から毎回同じ手を選んでも結果は確率的に変動する．

そこで本稿ではある状態 s における最善手を「s から獲得できる得点の合計の期待値が最も高くなるような手」と定義する．これは 3 節で述べた強化学習の最適状態価値と等価なものである．よって状態 s から最善手を選び続けて獲得できる得点の合計の期待値を状態 s の最適価値と呼び， $v_*(s)$ で表すことにする．

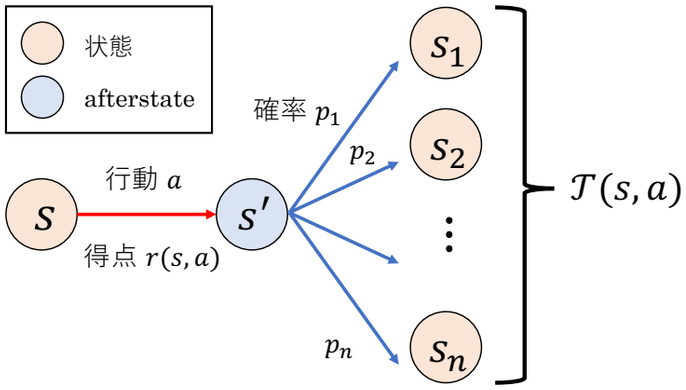
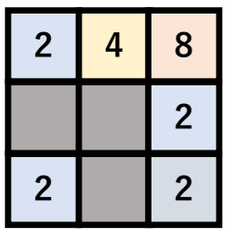
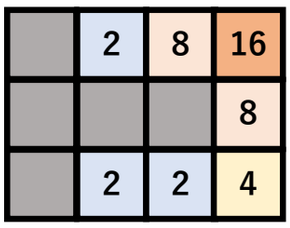


図 4.1: 式 4.1 の補足図



(a) 3×3 盤面の 2048



(b) 4×3 盤面の 2048

このとき $v_*(s)$ は式 4.1 のように再帰的な形式で書くことができる．

$$v_*(s) = \begin{cases} 0 & (s \text{ が終了状態}) \\ \max_a \left(r(s,a) + \mathbb{E}_{s_{\text{next}} \in \mathcal{T}(s,a)} v_*(s_{\text{next}}) \right) & (\text{otherwise}) \end{cases} \tag{4.1}$$

ただし $r(s,a)$ は状態 s から行動 a をとって獲得する得点，$s_{\text{next}} \in \mathcal{T}(s,a)$ は状態 s から行動 a をとって遷移しうる次の状態の集合を表す (図 4.1 を参照)．式 4.1 の $r(s,a) + \mathbb{E}_{s_{\text{next}} \in \mathcal{T}(s,a)} v_*(s_{\text{next}})$ は，強化学習における最適行動価値 $q_*(s,a)$ に対応する．また $\mathbb{E}_{s_{\text{next}} \in \mathcal{T}(s,a)} v_*(s_{\text{next}})$ は，s から a をとって遷移する afterstate s' の価値といえる．

ゲームに現れうるすべての状態の最適価値を計算すれば，任意の状態において最善手を選ぶことができる．本稿ではこれを 2048 の完全解析ということにする．

完全解析をすることで，ゲームの任意の状態の最適価値・最善手を明かし，最善手を選び続けるプレイヤの戦略を解析することができる．さらに 2048 を対象とした強化学習手法の良し悪しを，定量的な指標によって評価することができると考えられる．一方で 2048 を完全解析することは，そのゲーム木の大きさによる計算コストの観点から現状難しいと考えられる．そこで本研究では本来 4×4 盤面上で行われる 2048 のミニゲームとして，盤面サイズを縮小した 2048 を完全解析することを提案する．

Algorithm 1 すべての状態の列挙

```
1: function ENUMERATE
2:     INITIALIZE(S_4, S_6, S_8)
3:     for t = 4 to t_max do
4:         for all s_t ∈ S_t do
5:             for all s_{t+2} ∈ T(s_t) do
6:                 if s_{t+2} ∉ S_{t+2} then
7:                     S_{t+2} = S_{t+2} ∪ {s_{t+2}}
8:             for all s_{t+4} ∈ T(s_t) do
9:                 if s_{t+4} ∉ S_{t+4} then
10:                    S_{t+4} = S_{t+4} ∪ {s_{t+4}}
```

4.1.2 盤面サイズが小さな 2048 の完全解析

基本的なルールは 2048 と同じで盤面サイズを 4×4 から縮小したゲームを完全解析することを考える．盤面サイズに関わらず，以下の 2 つのステップを順番に行うことで完全解析を実行することができる．

1. 初期状態から到達し得るすべての状態の列挙
2. 列挙した状態の最適価値の計算

初期状態から到達し得るすべての状態の列挙

完全解析の第 1 ステップとしてゲームに現れうるすべての状態を列挙する．これまでに発見した状態の集合 S から状態を 1 つ取り出し，s から遷移可能な次の状態 $s_{\text{next}} \in \mathcal{T}(s)$ の内，未発見の状態を S に追加する．初期状態を S に入れて列挙を開始し，新たに発見する状態がなくなるまで繰り返すことで，すべての状態を列挙することができる．

素朴な方法ではこれまでに発見した状態の集合 S をメモリ上で管理することが考えられるが，状態数が非常に大きな場合にはメモリの容量を超えてしまう．そこで 2.2 節で説明した時刻によって，ゲーム木を整理しこれを解決する．時刻 t の状態は時刻 $t+2$ か $t+4$ の状態にしか遷移しないため，時刻 $t+2$ と $t+4$ の発見した状態の集合 S_{t+2} と S_{t+4} をメモリ上で管理すれば十分である．よって時刻が最小の 4 の状態から時刻 2 刻みで順番に列挙を行うことで，ディスクを効率的に活用することができる．以上を踏まえた疑似コードを Algorithm 1 に示す．

Algorithm 2 後退解析による価値計算

```
function RETROGRADE
    for t = t_max to 4 do
        for all s_t ∈ S_t do
            if s_t is gameover then
                v_*(s_t) = 0
            else
                v_*(s_t) = max_a (r(s_t, a) + E_{s_next ∈ T(s_t,a)} v_*(s_next))
```

盤面サイズ	状態数	初期状態の価値
$2 \times 2 = 4$	110	67.6
$3 \times 2 = 6$	$21,752$	480.9
$4 \times 2 = 8$	$4,980,767$	$2,642.6$
$3 \times 3 = 9$	$48,713,519$	$5,468.4$
$4 \times 3 = 12$	$1,152,817,492,752$	$50,724.2$

表 4.1: 盤面の大きさと解析結果に関する表

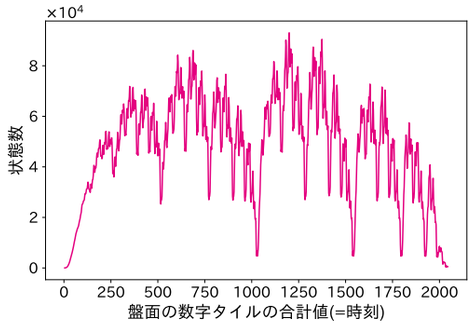
後退解析による状態の最適価値の計算

4.1.2 節で列挙した状態の価値を，式 4.1 に従って後退解析を行い計算する．状態列挙のときと同様に，時刻に従って状態を管理することで効率的に後退解析を行える．すなわち時刻 t の状態の価値は，時刻 $t+2$ と $t+4$ の状態の価値が計算済みであれば必ず計算できる．よって時刻が最大の状態から順番に走査することで，無駄なくすべての状態の価値を計算できる．疑似コードを Algorithm 2 に示す．

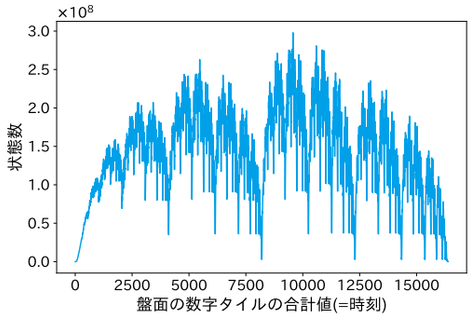
4.1.3 実験結果

本研究では 2×2 盤面から 4×3 盤面までの 2048 を完全解析することに成功した．完全解析した結果を表 4.1 に示す．盤面サイズが大きくなるに従って指数関数的に状態数は大きくなることが分かる．そのため 4×4 盤面の 2048 は完全解析を行うには状態数が非常に大きいことが予想される．一方で初期状態の最適価値は，盤面サイズが n マス増える度に $2^n \sim 2^{n+1}$ 倍になっていることが見て取れる．よって 4×4 盤面の 2048 では初期状態の最適価値は，少なく見積もっても $800,000$ 点程度はあるのではないかと推測される．

3×3 盤面と 4×3 盤面の 2048 の各時刻における状態数のグラフを図 4.1 に示す．多くのゲームでは進行に従って多様な盤面が存在するため状態数は大きくなり続ける．一方で 2048 は盤面が数字タイルで



(a) 3×3 盤面の 2048 の時刻と状態数



(b) 4×3 盤面の 2048 の時刻と状態数

埋まるとゲームオーバーになりやすく， その時刻の状態数は少なくなる．大きな数字タイルを完成させると盤面上に空きマスが増え， ゲームは再び複雑性を増す．実際， 図 4.1 からは 2^n の前後の時刻で状態数が大きく増減していることが見て取れる．よって 2048 はゲームの進行に従って， ゲームの複雑性が増減するという特徴を持つといえる．参考として， 2×2 盤面の 2048 のゲーム木全体を図 4.4 に示す．ゲーム木が拡大と縮小を繰り返す様子が見て取れる．

4.2　2048 のミニゲームの完全解析と強化学習

4.1.1 節では強化学習のベンチマークとしての 2048 のミニゲームの提案， およびその完全解析を行った．本研究では 3.2.3 節で述べた， Stochastic MuZero [15] について詳細に研究する．ただし Stochastic MuZero のように， 環境のダイナミクスモデルを学習するには多くの計算資源を要する．そこで AlphaZero [9] のように環境のダイナミクスは既知として， 方策・価値ネットワークの訓練のみを行う手法を考える．これを本稿では 2048-AlphaZero と呼ぶことにする．

本節では 3×3 盤面の 2048 および 4×3 盤面の 2048

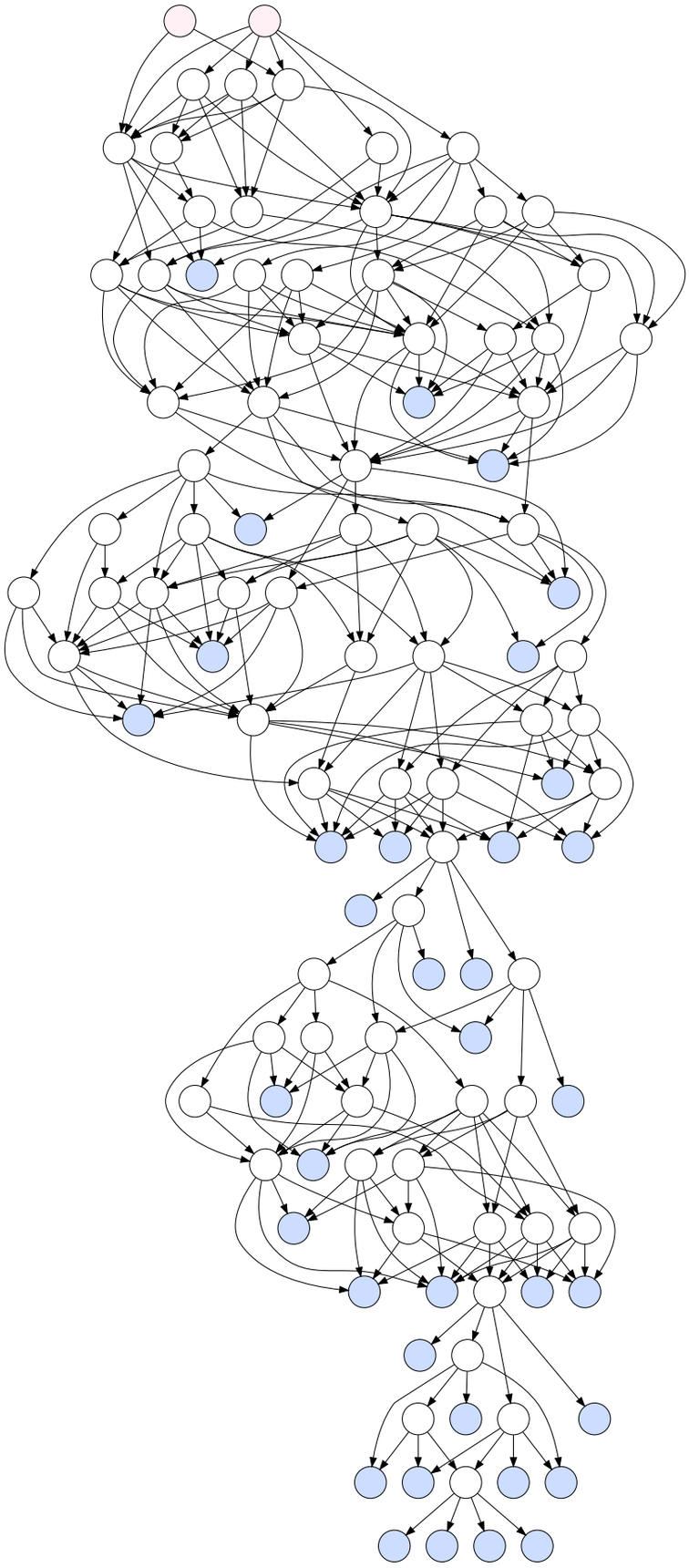


図 4.4: 2×2 盤面の 2048 のゲーム木 (赤色のノードは初期状態,　青色のノードは終了状態)

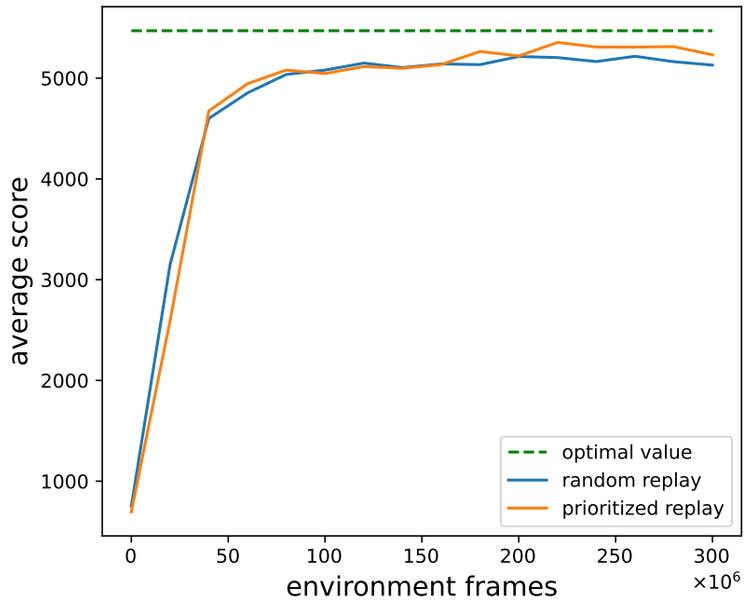


図 4.5: 3 × 3 盤面の 2048-alphazero

第 5 章

まとめと今後の展望

一方で 2048 を対象とした強化学習の先行研究は, 内発的報酬を利用した探索の促進が考えられる [18].

謝辞

参考文献

[1] G Cirulli. 2048. http://gabrielecirulli.github.io/2048/, 2014.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[3] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, Vol. 518, No. 7540, pp. 529–533, 2015.

[5] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, Vol. abs/1811.12560, , 2018.

[6] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, Vol. 8, pp. 229–256, 1992.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, Vol. abs/1602.01783, , 2016.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. https://arxiv.org/abs/1707.06347.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, Vol. 362, No. 6419, pp. 1140–1144, 2018.

[10] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned

model. *Nature*, Vol. 588, No. 7839, pp. 604–609, Dec 2020.

[11] Marcin Szubert and Wojciech Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. In *2014 IEEE CIG*, pp. 1–8, 2014.

[12] Wojciech Jaskowski. Mastering 2048 with delayed temporal coherence learning, multi-state weight promotion, redundant encoding and carousel shaping. *CoRR*, Vol. abs/1604.05085, , 2016.

[13] Kiminori Matsuzaki. Developing value networks for game 2048 with reinforcement learning. *Journal of Information Processing*, Vol. 29, pp. 336–346, 2021.

[14] 山下修平, 金子知適. 2048 への方策勾配法の適用. ゲームプログラミングワークショップ 2021 論文集, 第 2021 巻, pp. 179–185, nov 2021.

[15] Ioannis Antonoglou, Julian Schrittwieser, Sherjil Ozair, Thomas K Hubert, and David Silver. Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*, 2022.

[16] 山下修平, 金子知適, 中屋敷太一. 3 × 3 盤面の 2048 の完全解析と強化学習の研究. ゲームプログラミングワークショップ 2022 論文集, 第 2022 巻, pp. 1–8, nov 2022.

[17] 山下修平, 金子知適. 4 × 3 盤面の 2048 の完全解析. ゲームプログラミングワークショップ 2023 論文集, 第 2023 巻, pp. 1–5, nov 2023.

[18] Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. *CoRR*, Vol. abs/1810.12894, , 2018.

[19] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[20] GPCC2015 問題. 対戦型 2048. `https://hp.vector.co.jp/authors/VA003988/gpcc/gpcc15.htm#g2`, 2015.

付録 A

実装の詳細

A.1　ゲーム環境の実装

2048 は状態から afterstate への遷移において，各行 (列) の変化は独立に考えることができる．また回転と反転を考慮することで上下左右は等価な盤面変化を起こす．よって 1 行の全パターンについて，ある一方向を選択したときの遷移先を前もって計算することで，全方向に対する盤面全体の遷移を高速に行える．

A.2　完全解析の実装

図 A.1 に示すように，回転・反転に関して同じ盤面は 1 つの状態として扱った．またそれぞれの状態は 64 ビット整数で表現された．完全解析にはメモリ 256GB でプロセッサはコア数 32 の AMD Ryzen Threadripper 3970X のマシンを用いた．プログラムはすべて C++ 言語で実装された．

4×3 盤面の 2048 の完全解析は状態列挙に約 45 日，価値計算に約 20 日を要した．ただし実装の細かい工夫の仕方で改善することができると考えられる．また列挙した状態とその価値の情報を保持するために合計で 16.8TB の記憶領域を必要とした．本研究では状態を表す 64 ビットと，価値を表す 64 ビットを素朴に並べることでデータを記録した．簡潔データ構造の工夫を導入し，データを圧縮することは今後の課題としたい．

A.3　強化学習の実装

A.3.1　ニューラルネットワークの詳細

ニューラルネットワークは盤面の特徴量を入力として，方策と価値を出力する．盤面サイズ $H \times W$ のルールの下では理論上の最高到達タイルは $2^{H \times W + 1}$ である．このとき入力は空きマス $H \times W + 2$ チャネルの $H \times W$ から成る．n 番目のチャネルの (i, j) 成分には盤面の (i, j)

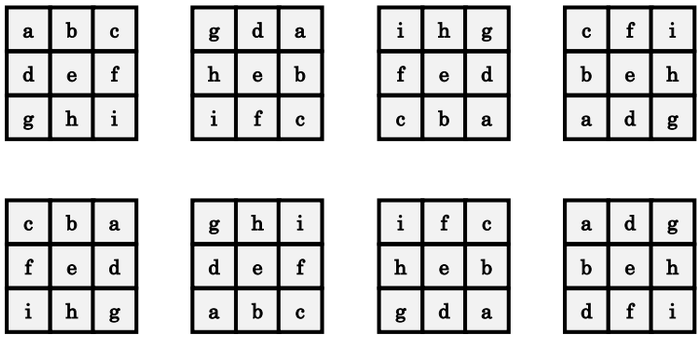


図 A.1: 3×3 盤面の 2048 の対称盤面

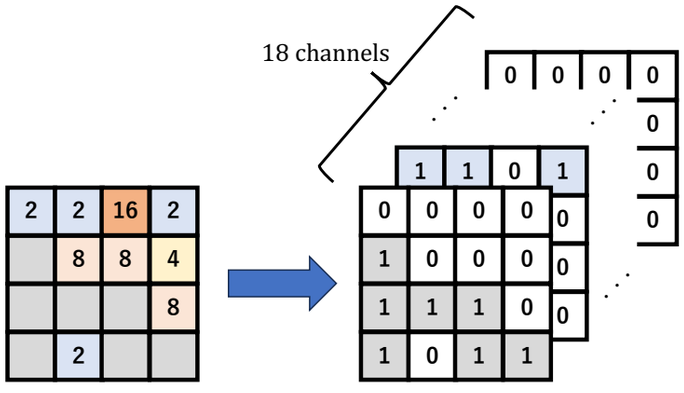


図 A.2: ニューラルネットワークへの入力特徴量

また Stochastic MuZero [15] に倣って， ニューラルネットワークは価値を D 次元のベクトル値として出力する．価値の学習ターゲット x (スカラー値) は， まず $h(x) = \sqrt{x+1} - 1 + \epsilon x$ $(\epsilon = 0.001)$ によってスケールを調整する (図 A.3 を参照)．さらに $h(x)$ は two-hot という， 特定の 2 つの要素以外は全ての 0 であるようなベクトル値に変換される．たとえば $h(x) = 3.7$ の場合， $3.7 = 3 \times 0.3 + 4 \times 0.7$ であるため， 3 の重みが 0.3， 4 の重みが 0.7， それ以外は 0 である D 次元ベクトルに変換される．これをニューラルネットワークの価値の学習ターゲットとして， Cross Entropy 誤差を最小化するように訓練される．推論時にはニューラルネットワークの価値の出力を， softmax により全体の総和を 1 にする．これを 0 から D までのそれぞれの重みとして， 重み付け平均 y を計算する．最後に $h(x)$ の逆関数である $h^{-1}(y) = \epsilon^{-1}(y + 0.5\epsilon^{-1} + 1.0) - 0.5\sqrt{4.0\epsilon^{-3}y + 1.004\epsilon^{-4}}$ によって， $x = h^{-1}(y)$ を得る．3×3 盤面の 2048 の実験では $D = 200$， 4×3 盤面では $D = 400$， 4×4 盤面では $D = 600$ とした．

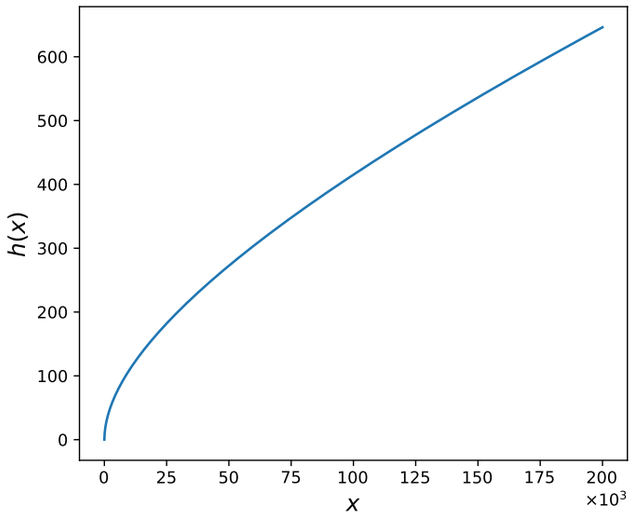


図 A.3: $h(x)$ のグラフ

A.3.2 Prioritized Experience Replay の実装

Prioritized Experience Replay [19] は学習に使用するデータを一様ランダムではなく，priority と呼ばれる重みに従ってサンプルするリプレイバッファである．リプレイバッファの i 番目のデータの priority を p_i とする．このとき i 番目のデータはハイパーパラメータ α を用いて， 確率 $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ でサンプルされる．

Prioritized Experience Replay からのデータのサンプル， および priority の更新は， sum-tree という二分木でデータを管理することで $\mathcal{O}(\log n)$ で行うことができる．sum-tree の葉ノードは priority の値を保持し， 各ノードは左右の子ノードの合計値を保持する．そのため根ノードは priority 全体の合計値 S を持つ．サンプルする際には， まず 0 から S までの値をランダムに生成し， 根ノードから葉ノードに至るまでたどることで選ぶ．図 A.4 に sum-tree と具体的なサンプルの仕方を例示する．

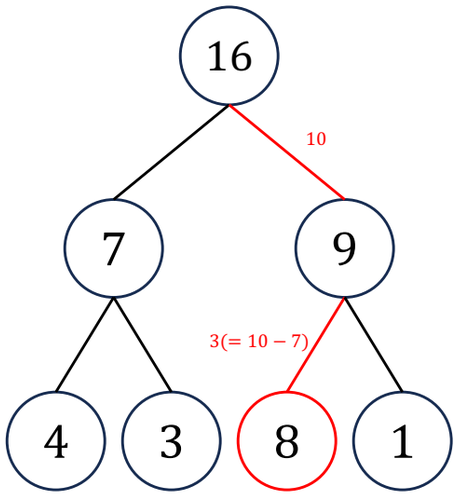


図 A.4: sum tree の例

付録 B

2048 のゲーム性とプレイヤの戦略の検証

ここでは 3×3 盤面の 2048 を主な題材として， 2048 のゲーム性とプレイヤの戦略について様々な面から検証する．

B.1　2 と 4 の出現確率とプレイヤの得点

2.1 節で述べたように， 2048 の通常のルールでは afterstate が次の状態へ遷移する際に出現する， 新しいタイルの数字と位置はランダムに決まる．すなわち afterstate の空きマスから等確率に選択されたある 1 マスに 90% の確率で 2 のタイルが， 10% の確率で 4 のタイルが置かれる．

ここで 2 のタイルと 4 のタイルの出現確率を変更した場合にゲーム性がどう変わるか検証する．表 B.1 に 3×3 盤面の 2048 において， 4 の出現確率を通常の 10% から増減させたときの完全解析の結果を示す．表から分かるように， 2 と 4 の出現確率がいずれか一方に傾くほど期待値は大きくなる傾向があることが分かる．また 4 の出現確率が 0% の場合と 100% の場合には， 最適な行動をし続ければ常に 図 2.3 のような理論上の最終盤面に到達できることが分かる．2048 は 2 と 4 の 2 種類の数字タイルが良い割合で出現することが， 人間がプレイするにあたってゲームを面白くしていると考えられる．

B.2　2048 と minmax 法

環境が常にプレイヤにとって最も都合の悪くなるように，新しい数字タイルを出現させるゲームを考える．これは二人零和有限完全確定情報ゲームと同様に， minmax 法によって完全解析を行える．よって本稿ではこの場合の環境を minmax 環境と呼ぶことにする．minmax 環境の 2048 は， 得点を最大化したいプレイヤと得点を最小化したい環境の対戦ゲームのように考えることができるため，「対戦型 2048」とも呼ばれている [20]．minmax 環境における状態の価値 v_{minmax} は， 以下の式 B.1 に従って， 4.1.1

表 B.1: 4 の出現確率を増減させたときのゲームの期待値

4 の確率	初期状態の期待値	4 の確率	初期状態の期待値
0.00	7172.00	0.55	3206.00
0.05	6161.17	0.60	3171.24
0.10	**5468.49**	0.65	3165.36
0.15	4932.54	0.70	3194.44
0.20	4515.42	0.75	3269.18
0.25	4182.44	0.80	3399.20
0.30	3919.20	0.85	3607.78
0.35	3704.44	0.90	3993.30
0.40	3531.46	0.95	4938.20
0.45	3390.19	1.00	14344.00
0.50	3278.70		

節と同様に後退解析を行い計算できる．

$$v_{\text{minmax}}(s) = \begin{cases} 0 & (s \text{ が終了状態}) \\ \max_a \left(r(s,a) + \min_{s_{\text{next}} \in \mathcal{T}(s,a)} v_{\text{minmax}}(s_{\text{next}}) \right) & (\text{otherwise}) \end{cases} \tag{B.1}$$

3×3 盤面の minmax 環境におけるプレイヤと環境の最善手順を図 B.1 に示す．ゲームは 21 手で終了し， プレイヤは 164 点しか獲得することができない．環境がプレイヤの妨害をすると， プレイヤの得点は最善手を選んだとしてもごく僅かになることがわかる．そのため対戦型 2048 において環境がプレイヤにとって最悪手を選択する場合， 通常の 2048 と比べてプレイヤ視点でのゲーム性に欠けるだろう．対戦型 2048 に良いゲーム性を持たせるには， 環境側に何らかの制約を加えるなどの工夫をする必要があると考えられる．

minmax 環境と同様に， 環境がプレイヤの得点を最大化させるように振る舞うようなゲームを考えることもできる．これは式 B.1 中の min を max に置き換えることで完全解析を行える．よってこの場合の環境を maxmax 環境と呼ぶことにする．直感的にも明らかなように， maxmax 環境では必ず図 2.3 に示したような理論上の最大盤面に到達することができる．そのため 3×3 盤面ではプレイヤは $16,352$ 点を獲得することができる．

さらに v_{minmax} や v_{maxmax} を評価関数として， 通常のルールの環境でプレイさせることを考える．それぞれ minmax プレイヤ， maxmax プレイヤと呼ぶことにする．minmax プレイヤが通常のルールの環境で獲得する得点の期待値を $v_{\text{minmax-eval}}$ とする．$v_{\text{minmax-eval}}$ は， 以下の式 B.2 に従って後退解析を行

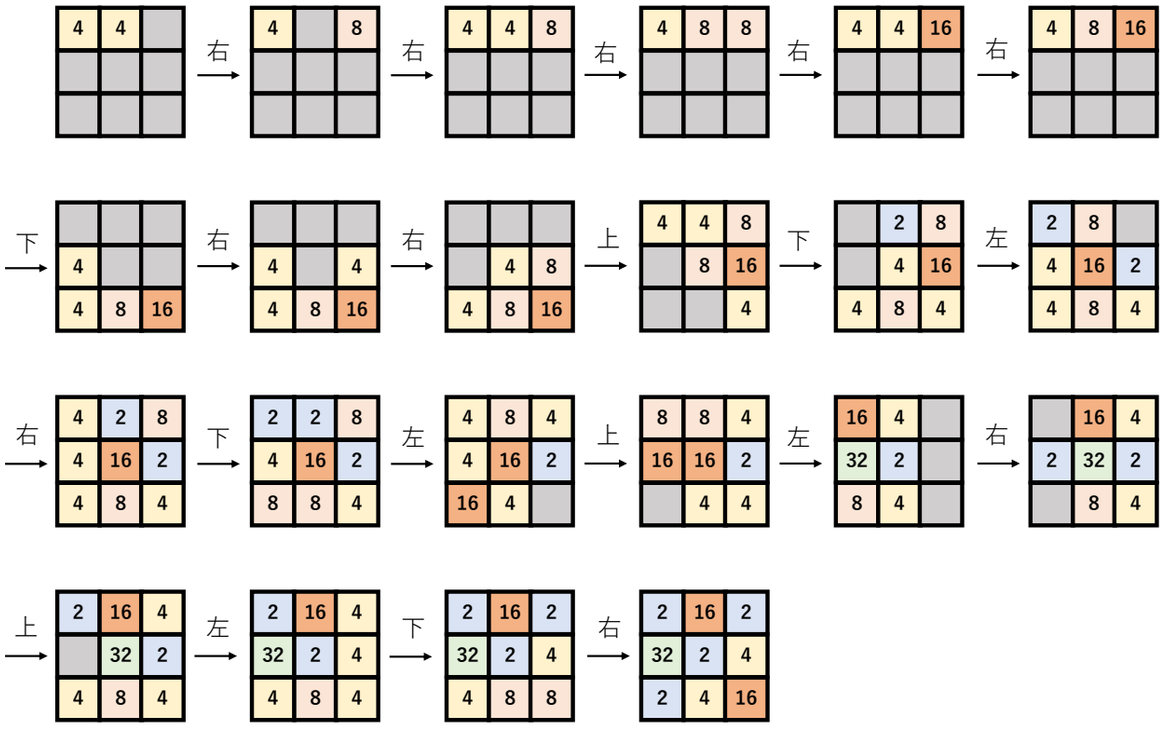


図 B.1: minmax 環境における双方の最善手順

い計算できる．ただし a_{minmax} を， v_{minmax} を評価関数としたときに選択する行動とする．

$$v_{\text{minmax-eval}}(s) = \begin{cases} 0 & (s \text{ が終了状態}) \\ \left(r(s, a_{\text{minmax}}) + \mathbb{E}_{s_{\text{next}} \in \mathcal{T}(s, a_{\text{minmax}})} v_{\text{minmax-eval}}(s_{\text{next}})\right) & (\text{otherwise}) \end{cases} \tag{B.2}$$

maxmax プレイヤが通常のルールの環境で獲得する得点の期待値 $v_{\text{maxmax-eval}}$ についても， 全く同じ方法で計算できる．

初期状態の $v_{\text{minmax-eval}}$ は $1,363.44$ 点， $v_{\text{maxmax-eval}}$ は minmax プレイヤは常に最悪な場合を想定して行動を選択する， 保守的な戦略をとるプレイヤといえる．ゲームオーバーにつながるような行動を避ける一方で， チャンスも逃し続けてしまうため v_* に従って行動を選択する場合に比べて獲得できる得点も大きく減少する．maxmax プレイヤは常に楽観的な場合を想定して行動を選択する， ギャンブル的な戦略をとるプレイヤといえる．