卒業研究報告書

題目

TD 学習によるテトリス AI の開発

指導教員

石水 隆　講師

報告者

19-1-037-0073

藤本　晴生

近畿大学理工学部情報学科

令和 05 年 01 月 26 日提出

概要

テトリスは,縦 20 行,横 10 列のフィールドに画面上部から落下してくる 4 つの正方形を組み合わせて作られたテトロミノと呼ばれるブロックを操作し配置していくゲームである.配置したブロックが横一列に揃えられるとその列のブロックが消えスコアを得る.ブロックの一部がフィールドの上限を超えた場合,ゲームが終了となる.

テトリスの最高得点を得る積み方を求める問題は NP 完全であり,最適なアルゴリズムを得ることはできないと考えられている[1].一方,テトリスによる強化学習は数多く行われているが,どの手法にも一長一短がありどれが最適かは一概には言えない.そこで本研究では強化学習アルゴリズム別に幾つかのテトリス AI を作成し, 得られたスコアを比較することで最適な強化学習アルゴリズムを見つけ出す.

目次

1. 序論

1.1 本研究の背景

テトリスは,縦 20 行,横 10 列のフィールドに画面上部から落下してくる 4 つの正方形を組み合わせて作られたテトロミノと呼ばれるブロックを操作し配置していくゲームである.プレイヤーは落下中のブロックを左右に移動させ，90 度ごとに回転させることができる．配置したブロックが横一列に隙間無く揃えられるとその列のブロックが消えスコアを得る.ブロックの一部がフィールドの上限を超えた場合,ゲームが終了となる.

ブロックの置き方によっては，複数の段をまとめて消す多段消しができる．多段消しをすると得点が多く得られるため，得点を稼ぐためには多段消しができるようにブロックを積んでいく必要がある．どのようにブロックを積むのが最適となるかは今後出現するブロックの形に依存するが，テトリスではブロックが落下しているときは次に出現するブロックの形のみが分かり，それ以降どのようなブロックが現れるかは分からない．このため，表示されている情報だけでは最適な詰み方を求めることはできない．また，仮に表示される情報の条件を緩めて今後出現する全てのブロックの形が分かると仮定しても、最も消せる列の数が多くなるブロックの並べ方，および最も 4 段消しの回数が多くなるブロックの並べ方を求める問題は NP 完全である[1].

また，ブロックの出現する順番によっては，永久にプレイすることはできず，必ずゲームオーバーになる．Burgiel は，Z 字型と S 字型のブロック（以下 Z ミノ，S ミノとする）が交互に降ってくる場合，70000 個以内に必ずゲームオーバーになることを示した[9].

一方,強化学習によるテトリス AI の作成は数多く行われているが,どの手法にも一長一短がありどれが最適かは一概には言えない.フィールドを評価する関数に非線形のニューラルネットワーク(以下 NN)を使用し,NN の最適化に TD 学習を用いた手法[11]ではラインを消去し始めるまでのゲーム試行回数は数十回と非常に少ない結果を得られていたが,その後消去段数に関して右肩上がりとはならず停滞する結果となっている.また,遺伝的アルゴリズム(以下 GA)と NN を組み合わせた手法[12]では 6000 万ライン消去の性能を示したが,学習時間が長い欠点があり,学習時間の短縮のため S と Z ミノの出現頻度を高めている.テトリスのような状態数が膨大となるゲームでは学習時間が長くなってしまうため,S と Z ミノの出現頻度を高めゲームが終了しやすくする[12]など学習を工夫する必要がある.そこで,本研究ではテトリスをスケールダウンさせる[3]などの工夫を行い状態数を減らすことで短時間での学習を可能にする.そして幾つかの強化学習アルゴリズムを検証し,最大のスコアが得られるテトリス AI を作成する.

1.2 本研究の目的

テトリスは，Z ミノと S ミノのみが出現し続けた場合，無限にゲームを続けることができず必ずゲームオーバーになる．しかし，テトリスの各ブロックの出現率はほぼ均等であるため，Z ミノ，S ミノのみが出現し続けることはまずありえない。このため，テトリスは最適な行動をとり続ける限りは実質上無限に続けることができる.そのため状況に応じて行動を学習することができる強化学習の対象としてテトリスはよく用いられる.また,テトリスは現在のブロック,次のブロック,現在のフィールド状況の 3 つの情報しか必要としないため,学習用プログラムを組むことが容易で新たな手法を試すことに利用しやすい.これら利点から本研究ではテトリスで強化学習を行っていく.概要でも述べた通りテトリスによる強化学習は数多く行われているが,強化学習アルゴリズムは一長一短であり,どれが最適であるかは一概には言えない.そこで本研究の目的は,テトリス AI をいくつかの強化学習アルゴリズム別で作成し,テトリスにおいて最適と考えられる強化学習アルゴリズムを見つけ出すことである.

1.3 本報告書の構成

本報告書の構成は以下の通りである.2 章で本研究対象であるテトリスについて説明し.続く 3 章で,

本研究で作成したテトリスのプログラムについて述べる. 4 章において結果を示し, 5 章では結論について述べる.

2　研究内容

本章では, 本研究対象であるテトリスについて説明する.

2.1 テトリスの概要

テトリスは 1984 年にソ連の計算機科学者アレクセイ・パジトノフが開発した落ち物パズルの元祖とされるコンピュータゲームである[10]. テトリスは, 縦 20 行, 横 10 列のフィールドに画面上部から落下してくる 4 つの正方形を組み合わせて作られたテトロミノと呼ばれるブロックを操作し配置していくゲームである. 配置したブロックが横一列に揃えられるとその列のブロックが消えスコアを得る. 配置したブロックが横一列に隙間無く揃えられるとその列のブロックが消えスコアを得る. ブロックの置き方によっては, 複数の段をまとめて消す多段消しができ, 多段消しをするとその段数に応じて得点が多く得られる. ブロックの一部がフィールドの上限を超えた場合, ゲームが終了となる.

2.2 テトリスのルール

本節では, 本研究でのテトリスのルールについて述べる.

- 本研究ではテトリスのフィールドを縦 5 行, 横 5 列で行う.
- 現在落下中のブロックに加え, 次に出現予定のブロックまで確認できる.
- テトリスのブロックは 7 種類で, 7 種類のブロックが 1 セットになり, 1 セットの中でまだ出現していないブロックのうちの一つがランダムに出現する. 7 つブロックが出現した後はそのセットが繰り返される.
- ブロックの落下は自由落下では行わず、即時に配置するものとするため, 図 1 のように下まで落下させた後に横に移動させるようなブロックの配置はできないものとする. 同様に T 字のブロックを回転させるなどしてブロックの隙間に入り込ませる T-spin などの配置は行えない.
- 本研究では落下してくるブロックが本来のテトリスと同じテトロミノのバージョンと, 3 つの正方形を組み合わせて作られたトリオミノのバージョンで研究を行う.

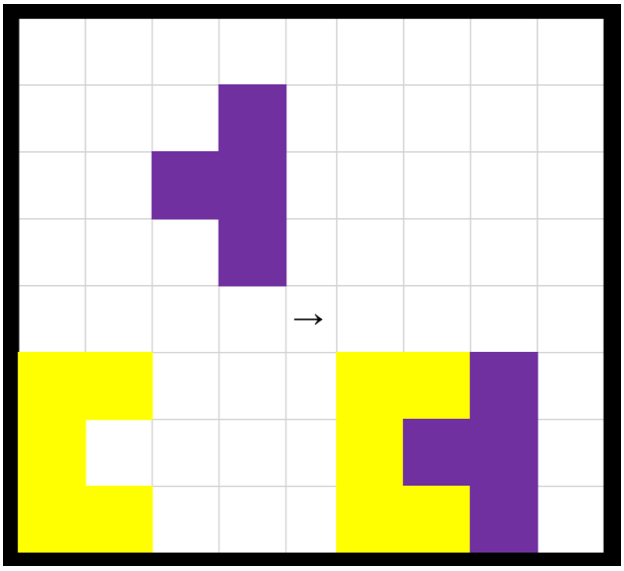

図 1 ブロックの設置

2.3 強化学習

本節では，本研究で用いている強化学習について述べる.

本研究では Q-learning と Sarsa を用いている.Q-learning と Sarsa は行動価値関数 Q の更新式で異なる.Sarsa の Q の更新式を(1)に示す.Sarsa は Q の更新時に次の行動a_{t+1}を求め更新に使用している.一方で,Q-learning の Q の更新式を(2)に示す.Q-learning では Q の更新時に次の状態s_{t+1}の Q の値のうち,最も大きいものを更新に使用する[13].

$$Q(s_t, a_t) = Q(s_t, a_t) + \eta\left(R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right) \tag{1}$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \eta\left(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right) \tag{2}$$

2.4 スケールダウン

本節では,スケールダウンについて述べる.

スケールダウンとは物事や計画の規模を縮小させることである.難しい学習対象に対し,簡単な状態のみで学習を行い,次第に難しい状態を増やしていくといった,段階を設けて学習を行う手法が提案されており,良い性能を示している[4]. 本研究で行うテトリスのスケールダウンはフィールドの状態を縦 20 ブロック,横 10 ブロックから縦 5 ブロック,横 5 ブロックにフィールドサイズを小さくし,学習を行っていく上で簡単な状態に変更することである.テトリスにおいてはこのようにフィールドサイズの変更を行ったとしてもゲームの状態をブロック単位で扱えるという特徴からゲームとして成立する.テトリスのゲーム AI にも適用しフィールドサイズが 5*5 の学習結果を 7*7 に利用している研究[3]では初めから 7*7 で学習させた方がライン消去数が多くなる結果が出ているが,5*5 で学習させる方がフィールドの状態数を減らすことができるため学習時間の短縮を図れると考え,本研究ではテトリスにスケールダウンを行っている.

3 テトリスのプログラム

本章では,本研究で作成したテトリス AI について説明する.付録に本研究で作成したテトリス AI のソースコードを示す.

3.1 プログラムの仕様

本節では,本研究で作成したテトリス AI のプログラム仕様について述べる.

本研究では python を用いて学習用プログラムを作成した.今回検証する強化学習アルゴリズムは現在落下中のブロックの他に次に落下してくるブロックが表示される特性を活かし,Q-learning と Sarsa の TD 学習と呼ばれる現在より先の状態を使用する学習手法を用いる.また強化学習を行なわずランダムにブロックを配置する場合も検証し,強化学習の有用性を確かめる.今回テトリスのフィールドを 5*5 で学習するため本来の 4 つのブロックが繋がったテトロミノだとフィールドが狭く学習があまり進まないことが危惧されるため,テトロミノでの学習に加え,3 つのブロックが繋がったトリオミノでの学習も行っていく． 図 2 のようにテトロミノは 7 種類,トリオミノは 2 種類となる.ブロックがフィールド上限を超えるまでを 1 回とし今回学習させる回数は,テトロミノで学習させる場合 Q-learning,Sarsa,ランダムのそれぞれで 2000 万回,トリオミノで学習させる場合 Q-learning,Sarsa,ランダムのそれぞれで 800 万回学習を行っていく.作成したテトリス AI の性能を評価するためにゲーム終了までに消去したライン数,配置したブロック数,プログラムの実行時間,フィールドの状態数の 4 つの指標を用いた.消去したライン数と配置したブロック数はテトリスのスコアに関わる指標であり,消去したライン数と配置したブロック数が多い方が高性能と言える.プログラム実行時間とフィールドの状態数はテトリス AI を学習させる際の効率に関わる指標であり,プログラム実行時

間が短く,フィールドの状態数が少ない方が学習効率が高いと言える.学習状況及び結果を標準出力させ,内容をテキストファイルに保存させる.また学習の最後の一回に関しては出現したブロックの履歴とフィールド情報を保存し,最後に動画の作成を行う.

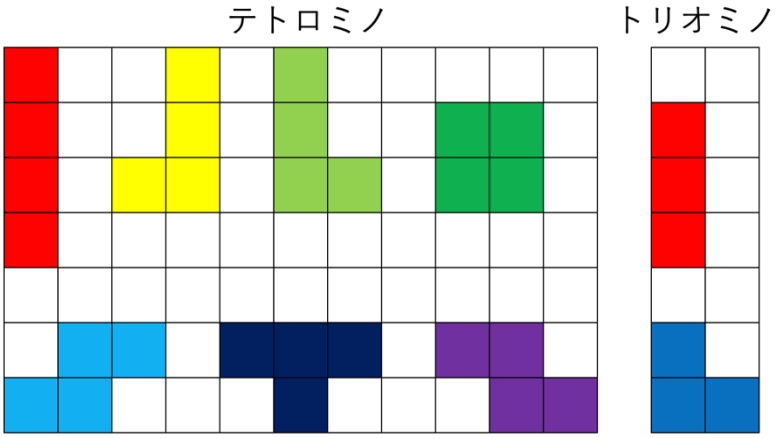

図 2 ブロックの種類

3.2 プログラムの構造

本節では,本研究で作成したテトリス AI の構造について述べる.

表 1 にインポートしてきたライブラリを示す.

表 1 インポートしたライブラリ

ライブラリ名	説明
numpy	多次元配列を扱うライブラリ
matplotlib.pyplot	グラフ描画のためのライブラリ
random	乱数を生成するモジュール
copy	オブジェクトをコピーするためのライブラリ
time	時間データへのアクセスをするためのライブラリ
psutil	ハードウェアの情報を取得するためのライブラリ

表 2 に設定した定数の一覧を示す.

表 2 定数一覧

定数名	説明	値
F_HEIGT	フィールドの高さ	5
F_WIDTH	フィールドの横幅	5
BLOCK_TYPE	ブロックの種類	トリオミノモード:2 テトロミノモード:3
PLAY_MODE	実行モード	トリオミノモード:3 テトロミノモード:4
LEARN_MODE	学習モード	ランダム:0 Q-learning:1 Sarsa:2
eta	学習率	0.1
gamma	時間割引率	0.9

EPSILON	確率	0.5
EPISODE	エピソード	トリオミノ:8000000 テトロミノモード:20000000
PLAY_COUNT	試行数のまとまり設定	10000

表 3 にメソッド及び各処理の一覧を示す.

表 3 メソッド及び各処理の一覧

処理名,メソッド名(引数)	ソースコードの行	説明
描画用のプログラム	38-66	テトリスのフィールドを描画する
各ブロック ０ Ｔ Ｓ Ｚ Ｉ Ｌ Ｊ	67-96	テトロミノ,トリオミノを配列で表現している
field_history_update (field_history,field)	97-103	フィールドの履歴を更新するメソッド
block_field_history_update (block_field_history,block_field)	103-112	ブロックフィールドの履歴を更新するメソッド
new_Q (theta)	112-122	新しく Q を生成するメソッド
create_key_from_field (field)	123-140	現在の field 状況からキー値を作成するメソッド
get_Q (field,theta,Q)	141-150	現在の field 状況から Q が保存されている Q の配列番号を返すメソッド
rotate_block (r,n)	151-255	ブロックを回転させるメソッド
decision_theta ()	256-269	theta の配列を作成するメソッド
decision_pi (theta)	270-280	pi を生成するメソッド
get_action (n,Q,epsilon,pi)	281-301	行動を決定するメソッド
get_field (n,r,w_p,field)	302-371	決定した行動に基づいてフィールドを更新するメソッド
get_block_field (n,r,w_p,block_field)	372-382	決定した行動に基づいてブロックのフィールドを更新するメソッド
get_R (line,state)	383-393	報酬を与えるメソッド
Q_learning (Q,R,Q_next,state,eta,gamma)	394-403	Q-learning による行動価値関数 Q の更新をするメソッド
Sarsa (Q,R,Q_next,state,eta,gamma)	404-413	Sarsa による行動価値関数 Q の更新をするメソッド
one_game (episode,Q,epsilon,eta,gamma,pi)	414-481	テトリスの 1 ゲームを行うプログラム
last_game (episode,Q,epsilon,eta,gamma,pi)	482-580	テトリスの最後の 1 ゲームを行うプログラム
実行プログラム	581-675	テトリスを実行するプログラム
動画作成用プログラム	676-773	テトリスの実行の様子を動画に収めるためのプログラム

4 結果と考察

テトリス AI をテトロミノで 2000 万回学習させた際の Q-learning,Sarsa の学習によるライン消

去数の推移とランダムに配置した場合の成果を図 3 に示す.図 3 より,Q-learning と Sarsa のライン消去数に大した差は見られないが,Random と比べ明らかに消去数が多く,右肩上がりになっていることから強化学習が正しく行われていることが分かる. 次にテトリス AI をトリオミノで 800 万回学習させた際の Q-learning,Sarsa の学習によるライン消去数の推移とランダムに配置した場合の成果を図 4 に示す.図 4 より,学習数が 400 万回を超えるまで Q-learning と Sarsa に大した学習の差は現れなかったが,それから Sarsa が急激にライン消去数を増加させ始め,最終的には 220 ライン程の差が Q-learning と Sarsa の間に現れた.また,テトロミノ,トリオミノで定めた回数分学習を終え得られた各指標の値を表 4,表 5 に示す.表 4 より,Q-learning と Sarsa に平均ライン消去数,平均ブロック配置数で大した差は見られないものの,プログラムの実行時間は 1.33 倍,フィールドの状態数は 1.23 倍と Sarsa の学習効率の悪さが示される.次に表 5 より,Sarsa は Q-learning と比べ平均ライン消去数は 5.51 倍,平均ブロック設置数は 5.22 倍と好成績を示した.しかし,Sarsa は Q-learning と比べプログラムの実行時間は 2.29 倍,フィールドの状態数は 1.29 倍と学習に難が示される.図 3,表 4 より,平均ライン消去数に学習による変化があまり見られず,Q-learning と Sarsa に差が現れなかったのは 5*5 のフィールドがテトロミノには狭く,ブロックの配置に制限がかかりライン消去にまで至らなかったことが予想される.一方図 4,表 5 のトリオミノでは自由にブロックが配置できたことにより,少ない学習回数で顕著な結果を得ることができたと考えられる.これらの結果から,スケールダウンを用いる手法にはミノのサイズに対して十分スペースを確保しておかなければならず,自由にブロックを配置できる必要がある.また表 4,表 5 より,Q-learning はプログラム実行時間が短く,フィールドの状態数が少ない特徴から,すぐに結果を得ることができるが,平均ライン消去数,平均ブロック配置数が共に少ないため大きな成果は得られない強化学習アルゴリズムであることが分かる.一方,Sarsa は平均ライン消去数,平均ブロック配置数が共に多い特徴から学習による成果を大きく得られるが,プログラム実行時間が長く,フィールドの状態数が多い特徴から,成果を得るのに時間がかかる強化学習アルゴリズムであることがわかる.

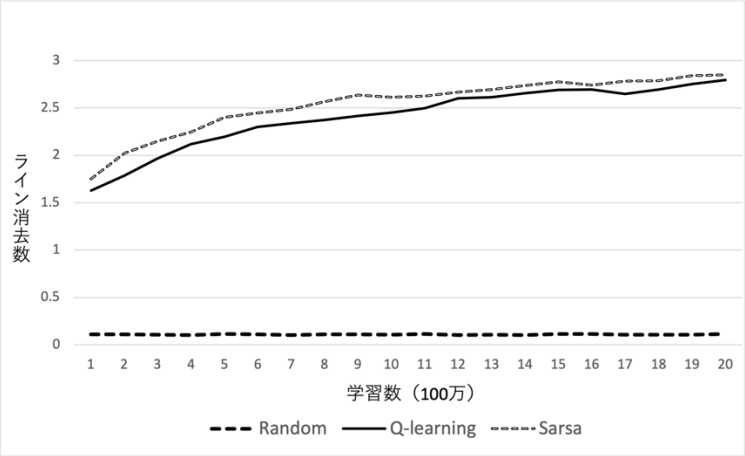

図 3 テトロミノでの学習数とライン消去数の関係

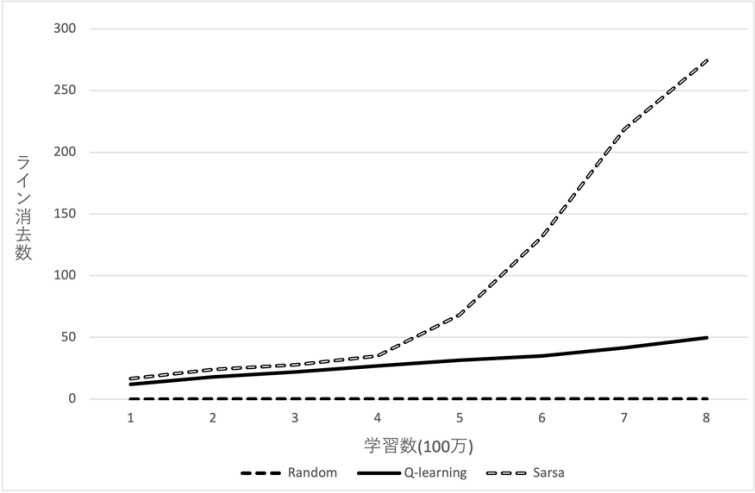

図 4 トリオミノでの学習数とライン消去数の関係

表 4 テトロミノで 2000 万回学習した際の各指標の値

	Q-learning	Sarsa	ランダム
平均ライン消去数	2.79	2.84	0.11
平均ブロック配置数	6.85	6.87	2.53
プログラムの実行時間(秒)	139826	186020	33536
フィールドの状態数	859373	1056461	372582

表 5 トリオミノで 800 万回学習した際の各指標の値

	Q-learning	Sarsa	ランダム
平均ライン消去数	49.67	274.09	0.17
平均ブロック配置数	88.48	462.25	3.75
プログラムの実行時間(秒)	138688	318024	35480
フィールドの状態数	243400	314684	310065

5 結論・今後の課題

本研究では Python を用いてテトリス AI を Q-learning と Sarsa で作成し,それぞれの学習による特徴を得ることができた.Q-learning,Sarsa にはそれぞれ利点があり,研究目的に応じて使い分ける必要があることがわかった.今後の課題は,トリオミノでの学習は顕著な結果を得られたが,テトロミノでは 5*5 のフィールドサイズは狭すぎると考えられ学習が思うように進まなかったため,今後はもう少し大きいサイズを用いて研究を行うか,もしくはスケールダウンを行わずに本来のサイズを用いられるよう学習用プログラムの高速化を行っていきたい.また,より多くの強化学習アルゴリズムを実装し,DQN など深層強化学習も調査していきたい.

謝辞

本研究を行うにあたり, 石水隆講師にはレジュメや卒論の添削などのご指導を受けました. ここに感謝の意を表します.

参考文献

[1] Erik D. Demaine, Susan Hohenberger, David Liben-Nowell：Tetris is Hard, Even to Approximate, Computer Science Vol.2002, No.20 pp. 1-56, Cornell University Library (2002) https://erikdemaine.org/papers/Tetris_COCOON2003/paper.pdf

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Mar-tin Riedmiller: Playing Atari with Deep Reinforcement Learning, In NIPS Deep Learning Workshop(2013) https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[3] 青木勢馬,橋本剛:テトリスを題材にしたスケールダウンを利用した学習手法の開発,ゲームプログラミングワークショップ 2017 論文集,pp.99-103,ゲームプログラミングワークショップ(2017) http://id.nii.ac.jp/1001/00183754/

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston: Curriculum learning, In ICML, 2009.

[5] 田伏未来, 萩原将文:ファジィ推論ニューラルネットワークを用いたテトリスのスキル獲得のための自動学習, 日本ファジィ学会誌 Vol.11, No,6, pp.1089-1097, 日本ファジィ学会 (1999) https://doi.org/10.3156/jfuzzy.11.6_201

[6] Simón Algorta and Özgür Şimşek：The Game of Tetris in Machine Learning, Computer Science,ArXiv (2019) https://arxiv.org/pdf/1905.01652.pdf

[7] Matt Stevens and Sabeek Pradhan：Playing Tetris with Deep Reinforcement Learning, Computer Science (2016) http://cs231n.stanford.edu/reports/2016/pdfs/121_Report.pdf

[8] Hanyuan Liu and Lixin Liu：Learn to Playing Tetris with Deep Reinforcement Learning, IERG5350 Reinforcement Learning Course Project (2020) https://openreview.net/pdf?id=8TLyqLGQ7Tg

[9] H.Burgiel:How to lose at Tetris, The Mathematical Gazette, Vol.81, No.491, pp.194-200, (1997) https://research.amanote.com/publication/P57A3XMBKQvf0BhiW5SZ/how-to-lose-at-tetris

[10] Dan Ackerman 著, 小林啓倫 訳：テトリス・エフェクト：世界を惑わせたゲーム, 白揚社, (2017)

[11] 中山亮士,TD 学習を用いたテトリス解放アルゴリズム, 法政大学大学院紀要. 理工学・工学研究科編,Vol.57,pp.1-8,法政大学院,(2016) https://hosei.repo.nii.ac.jp/?action=repository_action_common_download&item_id=13290&item_no=1&attribute_id=22&file_no=1

[12] 荒川正幹, 宮崎真奈実：ニューラ ルネットワークと遺伝的アルゴリズムを 用いたテトリスコントローラの開発, 情報 処理学会 第 74 回全国大会講演論文, pp. 539-540, (2012) http://id.nii.ac.jp/1001/00109944/

[13] 小川雄太郎,つくりながら学ぶ！深層強化学習,マイナビ出版, (2018)

付録　ソースコード

以下に本研究で作成したプログラムのソースコードを示す.

```
import numpy as np
import matplotlib.pyplot as plt
import random
import copy
import time
import psutil

%matplotlib inline
#  フィールドの高さ
F_HEIGHT = 5

#  フィールドの横幅
F_WIDTH = 5

#  ブロックの種類（２ or ７） ※トリオミノモード時 BLOCK_TYPE:2,テトロミノモード時 BLOCK_TYPE:7
BLOCK_TYPE = 2

# 実行モード(トリオミノ:3,テトロミノ:4)
PLAY_MODE = 3

# 学習モード(Random:0,Q-learning:1,Sarsa:2)
LEARN_MODE = 2

# 学習率(default:0.1)
eta = 0.1

# 時間割引率(default:0.9)
gamma = 0.9

#  ε-greedy 法の初期値(default:0.5)
EPSILON = 0.5

#  エピソード数
EPISODE = 10000

# 試行数のまとまり設定
PLAY_COUNT = 1000


#描画用のプログラム
fig = plt.figure(figsize=(F_WIDTH,F_HEIGHT+PLAY_MODE + 3))
ax = plt.gca()

#  縦線
for i in range(1,F_WIDTH):
    plt.plot([i,i],[0,F_HEIGHT+PLAY_MODE],color='black',linewidth=2)

#  横線
for i in range(1,F_HEIGHT+PLAY_MODE+1):
    if i != (F_HEIGHT):
        plt.plot([0,F_WIDTH],[i,i],color='black',linewidth=2)
    else:
         plt.plot([0,F_WIDTH],[i,i],color='red',linewidth=2)

#  次のブロック用マス（縦線）
for i in range(1,PLAY_MODE + 1):
    plt.plot([i,i],[F_HEIGHT + PLAY_MODE+1,F_HEIGHT + PLAY_MODE+3],color='black',linewidth=2)

#  次のブロック用マス（横線）
for j in range(0,2):
    plt.plot([0,PLAY_MODE],[F_HEIGHT + PLAY_MODE +1 + j,F_HEIGHT + PLAY_MODE + 1 +
j],color='black',linewidth=2)

```

```
62
63          ax.set_xlim(0,F_WIDTH)
64          ax.set_ylim(0,F_HEIGHT+PLAY_MODE + 3)
65          plt.tick_params(axis='both',which='both',bottom='off',top='off',
66                          labelbottom='off',right='off',left='off',labelleft='off')


67          #  各ブロック ０ Ｔ Ｓ Ｚ Ｉ Ｌ Ｊ
68          if PLAY_MODE == 4:
69              blocks = [
70                  [[1,1],
71                   [1,1]],
72
73                  [[0,2,0],
74                   [2,2,2]],
75
76                  [[0,3,3],
77                   [3,3,0]],
78
79                  [[4,4,0],
80                   [0,4,4]],
81
82                  [[5,5,5,5]],
83
84                  [[0,0,6],
85                   [6,6,6]],
86
87                  [[7,0,0],
88                   [7,7,7]]
89              ]
90          else:
91              blocks = [
92                  [[1,0],
93                   [1,1]],
94
95                  [[2,2,2]]
96              ]


97          #  フィールドの履歴を更新するメソッド
98          def field_history_update(field_history,field):
99
100             for i in range(0,F_HEIGHT):
101                 field_history.append([field[i][0],field[i][1],field[i][2],field[i][3],field[i][4]])
102
103             return field_history


104         #  ブロックフィールドの履歴を更新するメソッド
105         def block_field_history_update(block_field_history,block_field):
106
107             for i in range(0,PLAY_MODE):
108
109         block_field_history.append([block_field[i][0],block_field[i][1],block_field[i][2],block_field[i][3],
110                                     block_field[i][4]])
111
112             return block_field_history


113         #  新しく Q を生成するメソッド
114         def new_Q(theta):
115
116             Q_sub = np.zeros([1,BLOCK_TYPE,4,F_WIDTH])
117             for i in range(0,BLOCK_TYPE):
118                 for j in range(0,4):
119                     for k in range(0,F_WIDTH):
120                         Q_sub[0][i][j][k] = theta[i][j][k] * random.random()
121
122             return Q_sub
```

```
123         #  現在の field 状況からキー値を作成するメソッド
124         def create_key_from_field(field):
125
126             count = 0
127             s = ""
128
129             for i in range(F_HEIGHT - 1,-1,-1):
130                 for  j in range(F_WIDTH - 1,-1,-1):
131                     if field[i][j] != 0:
132                         count += 1
133                         s += "1"
134                     else:
135                         s += "0"
136
137             if count == 0:
138                 return "clear"
139
140             return s


141         #  現在の field 状況から Q が保存されている Q の配列番号を返すメソッド
142         def get_Q(field,theta,Q):
143             s = create_key_from_field(field)
144
145             if Q_search.get(s) is None:
146                 Q_search[s] = len(Q)
147                 Q_sub = new_Q(theta)
148                 Q = np.concatenate([Q,Q_sub])
149
150             return [Q_search.get(s),Q]


151         #  ブロックを回転させるメソッド r:回転させる回数(時計回り 0-3) n:block の番号
152         def rotate_block(r,n):
153             if PLAY_MODE == 4:
154                 #  0 ブロックの場合
155                 if n == 0 or r == 0:
156                     return blocks[n]
157
158                 (height,width) = np.shape(blocks[n])
159
160                 if (r % 2) ==  1:
161                     block = [[0] * height for i in range(width)]
162
163                 else:
164                     block = [[0] * width for i in range(height)]
165
166                 #  右に 90 度
167                 if r== 1:
168                     if n == 4:
169                         block[0][0] = blocks[n][0][0]
170                         block[1][0] = blocks[n][0][1]
171                         block[2][0] = blocks[n][0][2]
172                         block[3][0] = blocks[n][0][3]
173                     else:
174                         block[0][1] = blocks[n][0][0]
175                         block[1][1] = blocks[n][0][1]
176                         block[2][1] = blocks[n][0][2]
177                         block[0][0] = blocks[n][1][0]
178                         block[1][0] = blocks[n][1][1]
179                         block[2][0] = blocks[n][1][2]
180
181                 #  右に 180 度
182                 elif r==2:
183                     if n == 2 or n==3 or n== 4:
184                         return blocks[n]
185                     else:
186                         block[1][0] = blocks[n][0][2]
187                         block[1][1] = blocks[n][0][1]
```

```
                    block[1][2] = blocks[n][0][0]
                    block[0][0] = blocks[n][1][2]
                    block[0][1] = blocks[n][1][1]
                    block[0][2] = blocks[n][1][0]

            #  左に 90 度
            elif r==3:
                if n == 4:
                    block[0][0] = blocks[n][0][0]
                    block[1][0] = blocks[n][0][1]
                    block[2][0] = blocks[n][0][2]
                    block[3][0] = blocks[n][0][3]
                else:
                    block[2][0] = blocks[n][0][0]
                    block[1][0] = blocks[n][0][1]
                    block[0][0] = blocks[n][0][2]
                    block[2][1] = blocks[n][1][0]
                    block[1][1] = blocks[n][1][1]
                    block[0][1] = blocks[n][1][2]

            return block
        else:
            if r == 0:
                return blocks[n]

            (height,width) = np.shape(blocks[n])

            if (r % 2) ==  1:
                block = [[0] * height for i in range(width)]

            else:
                block = [[0] * width for i in range(height)]

            #  右に 90 度
            if r== 1:
                if n == 1:
                    block[0][0] = blocks[n][0][0]
                    block[1][0] = blocks[n][0][1]
                    block[2][0] = blocks[n][0][2]
                else:
                    block[0][1] = blocks[n][0][0]
                    block[1][1] = blocks[n][0][1]
                    block[0][0] = blocks[n][1][0]
                    block[1][0] = blocks[n][1][1]

            #  右に 180 度
            elif r==2:
                if n == 1:
                    return blocks[n]
                else:
                    block[1][0] = blocks[n][0][1]
                    block[1][1] = blocks[n][0][0]
                    block[0][0] = blocks[n][1][1]
                    block[0][1] = blocks[n][1][0]

            #  左に 90 度
            elif r==3:
                if n == 1:
                    block[0][0] = blocks[n][0][0]
                    block[1][0] = blocks[n][0][1]
                    block[2][0] = blocks[n][0][2]
                else:
                    block[1][0] = blocks[n][0][0]
                    block[0][0] = blocks[n][0][1]
                    block[0][1] = blocks[n][1][1]
                    block[1][1] = blocks[n][1][0]

            return block
    #  theta の配列を作成するメソッド
    def decision_theta():
```

```
258
259             #  theta の配列を生成し、全てを nan にしている
260             theta = np.zeros([BLOCK_TYPE,4,F_WIDTH])
261             theta[:,:,:] = np.nan
262             for h in range(0,BLOCK_TYPE):
263                 for i in range(0,4):
264                     (height,width) = np.shape(rotate_block(i,h))
265
266                     for j in range(0,F_WIDTH + 1 - width):
267                         theta[h][i][j] = 1
268
269             return theta


270         #  pi を生成するメソッド
271         def decision_pi(theta):
272
273             pi = np.zeros([BLOCK_TYPE,4,F_WIDTH])
274             for h in range(0,BLOCK_TYPE):
275                 for i in range(0,4):
276                     pi[h,i,:] = theta[h,i,:] / np.nansum(theta[h,i,:])
277
278             pi = np.nan_to_num(pi)
279
280             return pi


281         #  行動を決定します
282         def get_action(n,Q,epsilon,pi):
283
284             position = np.arange(F_WIDTH)
285
286             #  ランダムに動きます
287             if (np.random.rand() < epsilon) or (LEARN_MODE == 0):
288                 r = random.randint(0,3)
289                 p = np.random.choice(position,p=pi[n,r,:])
290
291             #  Q の最大値の行動を採用します
292             else:
293                 value = np.nanmax(Q[n,0,:])
294                 r = 0
295                 p = np.nanargmax(Q[n,0,:])
296                 for i in range(1,4):
297                     if np.nanmax(Q[n,i,:]) >= value:
298                         r = i
299                         p = np.nanargmax(Q[n,i,:])
300
301             return [r,p]


302         #  決定した行動に基づいてフィールドを更新します
303         def get_field(n,r,w_p,field):
304
305             block = rotate_block(r,n)
306             [h,w] = np.shape(block)
307
308             count = 0
309             h_p = -1
310
311             while(1):
312
313                 for i in range(0,w):
314                     value  = field[F_HEIGHT-1-count][w_p + i]
315                     if value != 0:
316                         if block[h-1][i] != 0:
317                             h_p = count
318                             break
319                         else:
320                             if h_p == -1:
321                                 if PLAY_MODE == 1:
```

```
                                if (n ==5 and r == 3) or (n == 6 and r == 1):
                                    h_p = count + 2
                                else:
                                    h_p = count + 1
                            else:
                                h_p = count + 1

                if h_p == count:
                    break
                if count == F_HEIGHT -1:
                    h_p = F_HEIGHT
                    break

                count += 1

            if (h- h_p) > 0:
                return [field,0,False]

            for i in range(0,h):
                for j in range(0,w):
                    if block[h-1-i][j] != 0:
                        field[F_HEIGHT-h_p + i][w_p + j] = block[h - 1 - i][j]

            #  ライン消去
            line = 0
            delete_line_height = []
            for i in range(0,F_HEIGHT):
                count = 0
                for j in range(0,F_WIDTH):
                    if field[i][j] != 0:
                        count += 1

                if count == F_WIDTH:
                    delete_line_height.append(i)

            if len(delete_line_height) != 0:

                field_sub = np.zeros([F_HEIGHT,F_WIDTH])

                count = 0
                for i in range(0,F_HEIGHT):
                    for j in range(0,F_WIDTH):
                        if (i in delete_line_height) == True:
                            count += 1
                            break
                        else:
                            field_sub[i - count][j] = field[i][j]
                return [field_sub,count,True]

            return [field,0,True]


        #  決定した行動に基づいてブロックのフィールドを更新します
        def get_block_field(n,r,w_p,block_field):

            block = rotate_block(r,n)
            [h,w] = np.shape(block)

            for i in range(0,h):
                for j in range(0,w):
                    block_field[i][j + w_p] = block[i][j]

            return block_field


        #  報酬を与えるメソッド
        def get_R(line,state):

            if state == False:
                return -1
```

```
388
389             #  ラインを消すことができれば5点
390             if line != 0:
391                 return line
392
393             return 0


394         #  Q-learningによる行動価値関数Qを更新するメソッド
395         def Q_learning(Q,R,Q_next,state,eta,gamma):
396
397             if state == False:
398                 Q = Q + eta * (R - Q)
399
400             else:
401                 Q = Q + eta * (R + gamma * np.nanmax(Q_next) - Q)
402
403             return Q


404         #  Sarsaによる行動価値関数Qを更新するメソッド
405         def Sarsa(Q,R,Q_next,state,eta,gamma):
406
407             if state == False:
408                 Q = Q + eta * (R - Q)
409
410             else:
411                 Q = Q + eta * (R + gamma * Q_next - Q)
412
413             return Q


414         #  テトリスの1ゲームを行うプログラム
415         def one_game(episode,Q,epsilon,eta,gamma,pi):
416
417             #  フィールドの初期化
418             field = np.zeros([F_HEIGHT,F_WIDTH])
419
420             #  ラインの消去数
421             one_game_line_total = 0
422
423             #  初めのブロックを決める
424             block_list = random.sample(range(BLOCK_TYPE),BLOCK_TYPE)
425             n = block_list[0]
426
427             count = 0
428
429             while True:
430
431                 #  ブロックを決める
432                 if count == 0:
433                     n_next = block_list[1]
434
435                 else:
436                     if (count + 1)% BLOCK_TYPE == 0:
437                         block_list = random.sample(range(BLOCK_TYPE),BLOCK_TYPE)
438                     n_next = block_list[(count + 1) % BLOCK_TYPE]
439
440                 #  Q値の保存されている配列番号を得る
441                 [v,Q] = get_Q(field,theta,Q)
442
443                 if LEARN_MODE == 0 or LEARN_MODE == 1:
444                     #  返り値で回転数とポジションを得る(Q-learning and Random)
445                     [r,p] = get_action(n,Q[v],epsilon,pi)
446
447                 if LEARN_MODE == 2:
448                     #  返り値で回転数とポジションを得る(Sarsa)
449                     if count == 0:
450                         [r,p] = get_action(n,Q[v],epsilon,pi)
451                     else:
```

```
                        r = r_next
                        p = p_next

                #  フィールドを更新する
                [field,line,state] = get_field(n,r,p,field)

                #  報酬を与える
                R = get_R(line,state)

                #  次のQ値の保存されている配列番号を得る
                [v_next,Q] = get_Q(field,theta,Q)

                if LEARN_MODE == 1:
                    #  Q-learningによる行動価値関数Qの更新
                    Q[v][n][r][p] = Q_learning(Q[v][n][r][p],R,Q[v_next][n_next],state,eta,gamma)

                if LEARN_MODE == 2:
                    #  Sarsaによる行動価値関数Qの更新
                    [r_next,p_next] = get_action(n_next,Q[v_next],epsilon,pi)
                    Q[v][n][r][p] = Sarsa(Q[v][n][r][p],R,Q[v_next][n_next][r_next][p_next],state,eta,gamma)

                one_game_line_total += line

                if state == False:
                    break
                v = v_next
                n = n_next
                count += 1

        return [one_game_line_total,count,Q]


#  テトリスの最後の1ゲームを行うプログラム
def last_game(episode,Q,epsilon,eta,gamma,pi):

    #  フィールド
    field = np.zeros([F_HEIGHT,F_WIDTH])

    #  ブロックのフィールド
    block_field = np.zeros([PLAY_MODE,F_WIDTH])

    #  フィールドの履歴
    field_history = []
    field_history = field_history_update(field_history,field)

    #  ブロックのフィールドの履歴
    block_field_history = []
    block_field_history = block_field_history_update(block_field_history,block_field)

    #  ブロックの履歴
    block_history = []

    #  ラインの消去数
    one_game_line_total = 0

    #  初めのブロックを決める
    block_list = random.sample(range(BLOCK_TYPE),BLOCK_TYPE)
    n = block_list[0]

    block_history.append(n)

    count = 0

    while True:

        #  ブロックを決める
        if count == 0:
            n_next = block_list[1]

        else:
```

```
                    if (count + 1)% BLOCK_TYPE == 0:
                        block_list = random.sample(range(BLOCK_TYPE),BLOCK_TYPE)
                    n_next = block_list[(count + 1) % BLOCK_TYPE]

                block_history.append(n_next)

                #  Q値の保存されている配列番号を得る
                [v,Q] = get_Q(field,theta,Q)

                if LEARN_MODE == 0 or LEARN_MODE == 1:
                    #  返り値で回転数とポジションを得る(Q-learning and Random)
                    [r,p] = get_action(n,Q[v],epsilon,pi)

                if LEARN_MODE == 2:
                    #  返り値で回転数とポジションを得る(Sarsa)
                    if count == 0:
                        [r,p] = get_action(n,Q[v],epsilon,pi)
                    else:
                        r = r_next
                        p = p_next

                #  フィールドを更新する
                [field,line,state] = get_field(n,r,p,field)

                #  報酬を与える
                R = get_R(line,state)

                #  ブロックのフィールドを更新する
                block_field = get_block_field(n,r,p,block_field)

                #  ブロックのフィールド履歴を更新する
                block_field_history = block_field_history_update(block_field_history,block_field)

                #  ブロックのフィールドを初期化する
                block_field = np.zeros([PLAY_MODE,F_WIDTH])

                #  フィールドの履歴を更新する
                field_history = field_history_update(field_history,field)

                #  次のQ値の保存されている配列番号を得る
                [v_next,Q] = get_Q(field,theta,Q)

                if LEARN_MODE == 1:
                    #  Q-learningによる行動価値関数Qの更新
                    Q[v][n][r][p] = Q_learning(Q[v][n][r][p],R,Q[v_next][n_next],state,eta,gamma)

                if LEARN_MODE == 2:
                    #  Sarsaによる行動価値関数Qの更新
                    [r_next,p_next] = get_action(n_next,Q[v_next],epsilon,pi)
                    Q[v][n][r][p] = Sarsa(Q[v][n][r][p],R,Q[v_next][n_next][r_next][p_next],state,eta,gamma)

                one_game_line_total += line

                if state == False:

                    break
                v = v_next
                n = n_next
                count += 1

        return [field_history,block_history,block_field_history,one_game_line_total,count,Q]


    #プログラムの実行
    #  theta
    theta = decision_theta()

    #  pi
    pi = decision_pi(theta)
```

```
588             #  Q値の索引用辞書
589             Q_search = {'clear':0}
590
591             #  Q値
592             Q = np.zeros([1,BLOCK_TYPE,4,F_WIDTH])
593             for i in range(0,BLOCK_TYPE):
594                 for j in range(0,4):
595                     for k in range(0,F_WIDTH):
596                         Q[0][i][j][k] = theta[i][j][k] * random.random()
597
598             is_continue = True
599             episode = 1
600
601             epsilon = EPSILON
602
603             # ライン消去の総数
604             line_total = 0
605
606             # ブロック設置数の総数
607             block_total = 0
608
609             #  PLAY_COUNT 試行ごとのライン消去平均を求める
610             episode_line_list = np.zeros(PLAY_COUNT)
611
612             # PLAY_COUNT 試行ごとのブロック設置数平均を求める
613             episode_block_list = np.zeros(PLAY_COUNT)
614
615             #  時間計測 start
616             time_sta = time.perf_counter()
617
618             # ファイル書き込み
619             f = open("result.txt","w")
620
621             while is_continue:
622
623                 if episode != EPISODE:
624
625                     [line,block_put,Q] = one_game(episode,Q,epsilon,eta,gamma,pi)
626
627                 else :
628
629                     [field_history,block_history,block_field_history,line,block_put,Q]=
630         last_game(episode,Q,epsilon,eta,gamma,pi)
631
632                 line_total += line
633
634                 block_total += block_put
635
636                 episode_line_list = np.hstack((episode_line_list[1:],line))
637
638                 episode_block_list = np.hstack((episode_block_list[1:],block_put))
639
640                 if (episode % PLAY_COUNT == 0) or (episode == EPISODE):
641                      # 時間計測 end
642                     time_end = time.perf_counter()
643
644                     print("エピソード:" + str(episode))
645                     print("テーブル数:" + str(len(Q))+"テーブル")
646                     print("プログラム実行時間:"+str(time_end - time_sta)+"秒")
647                     print(str(PLAY_COUNT)+"試行ごとの平均ライン消去数:"+ str(episode_line_list.mean())+"ライン")
648                     print(str(PLAY_COUNT)+"試行ごとの平均ブロック設置数:"+str(episode_block_list.mean())+"ブロック")
649                     print()
650
651                     f.write("エピソード:" + str(episode) + "¥n")
652                     f.write("テーブル数" + str(len(Q)) + "テーブル¥n")
653                     f.write(" CPU 使用率は"+str(psutil.cpu_percent())+"%¥n")
654                     f.write("プログラム実行時間:"+str(time_end - time_sta)+"秒¥n")
655                     f.write("メモリ使用量:"+str(psutil.virtual_memory().used / 1024 / 1024)+"MB¥n")
656                     f.write(str(PLAY_COUNT)+"試行ごとの平均ライン消去数:"+ str(episode_line_list.mean())+"ライン¥n")
657                     f.write(str(PLAY_COUNT)+"試行ごとの平均ブロック設置数:"+str(episode_block_list.mean())+"ブロック
```

```
¥n¥n")

        if episode == EPISODE:
            break

    if epsilon > 0.005:
        epsilon /= 1.001

    episode += 1

print()

print("1 ゲームあたりの平均消去列"+str(line_total / episode)+"ライン")
print("1 ゲームあたりの平均ブロック設置数:"+str(block_total / episode)+"ブロック")

f.write("1 ゲームあたりの平均消去列:"+str(line_total / episode)+"ライン")
f.write("1 ゲームあたりの平均ブロック設置数:"+str(block_total / episode)+"ブロック")

# ファイルを閉じる
f.close()


#テトリスの実行の様子を動画に収めるためのプログラム
from matplotlib import animation
from IPython.display import HTML
import matplotlib.cm as cm

def init():
   # line.set_data([],[])
    return (line,)

def animate(i):
    #  フィールドを白で初期化
    for  height in range(0,F_HEIGHT + PLAY_MODE + 3):
        for width in range(0,F_WIDTH):
            line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                            color='white',markersize=50)

    #  次のブロックを表示させる
    [a,b] = np.shape(blocks[block_history[i]])
    for height in range(0,a):
        for width in range(0,b):
            if blocks[block_history[i]][height][width] == 1:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE + 2 - height],marker="s",
                                color='red',markersize=50)
            if blocks[block_history[i]][height][width] == 2:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE + 2 - height],marker="s",
                                color='green',markersize=50)
            if blocks[block_history[i]][height][width] == 3:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE + 2 - height],marker="s",
                                color='blue',markersize=50)
            if blocks[block_history[i]][height][width] == 4:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE + 2 - height],marker="s",
                                color='yellow',markersize=50)
            if blocks[block_history[i]][height][width] == 5:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE + 2 - height],marker="s",
                                color='aqua',markersize=50)
            if blocks[block_history[i]][height][width] == 6:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE + 2 - height],marker="s",
                                color='pink',markersize=50)
            if blocks[block_history[i]][height][width] == 7:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE + 2 - height],marker="s",
                                color='navy',markersize=50)

    #  フィールドに置く一つのブロックを表示させる
    for height in range(0,PLAY_MODE):
        for width in range(0,F_WIDTH):
            if block_field_history[height + i*PLAY_MODE][width] == 1:
                line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE - 1 - height],marker="s",
                                color='red',markersize=50)
```

```
                    if block_field_history[height + i*PLAY_MODE][width] == 2:
                        line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE -1 - height],marker="s",
                                        color='green',markersize=50)
                    if block_field_history[height + i*PLAY_MODE][width] == 3:
                        line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE -1 - height],marker="s",
                                        color='blue',markersize=50)
                    if block_field_history[height + i*PLAY_MODE][width] == 4:
                        line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE -1 - height],marker="s",
                                        color='yellow',markersize=50)
                    if block_field_history[height + i*PLAY_MODE][width] == 5:
                        line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE -1 - height],marker="s",
                                        color='aqua',markersize=50)
                    if block_field_history[height + i*PLAY_MODE][width] == 6:
                        line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE -1 - height],marker="s",
                                        color='pink',markersize=50)
                    if block_field_history[height + i*PLAY_MODE][width] == 7:
                        line, = ax.plot([width + 0.5],[0.5 + F_HEIGHT + PLAY_MODE - 1 - height],marker="s",
                                        color='navy',markersize=50)

            #  フィールドのブロックを表示させる
            for height in range(0,F_HEIGHT):
                for width in range(0,F_WIDTH):
                    if field_history[height + i*F_HEIGHT][width] == 1:
                        line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                                        color='red',markersize=50)
                    if field_history[height + i*F_HEIGHT][width] == 2:
                        line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                                        color='green',markersize=50)
                    if field_history[height + i*F_HEIGHT][width] == 3:
                        line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                                        color='blue',markersize=50)
                    if field_history[height + i*F_HEIGHT][width] == 4:
                        line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                                        color='yellow',markersize=50)
                    if field_history[height + i*F_HEIGHT][width] == 5:
                        line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                                        color='aqua',markersize=50)
                    if field_history[height + i*F_HEIGHT][width] == 6:
                        line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                                        color='pink',markersize=50)
                    if field_history[height + i*F_HEIGHT][width] == 7:
                        line, = ax.plot([width + 0.5],[height + 0.5],marker="s",
                                        color='navy',markersize=50)

            return (line,)

        anim = animation.FuncAnimation(
            fig,animate,init_func=init,frames=int((len(field_history) / F_HEIGHT)),interval=200,repeat=False)

        HTML(anim.to_jshtml())
```