卒業研究報告書

題目

遺伝的アルゴリズムによる
テトリス AI の戦略比較

指導教員

石水 隆　講師

報告者

20-1-037-0147

黒山 泰希

近畿大学理工学部情報学科

令和 6 年 1 月 29 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ちてくるテトリミノと呼ばれるブロックを左右に動かしたり, 回転させたりして配置し, 積み上げていくゲームである. テトリミノで横一行のラインをつくると, その行のテトリミノが消えてスコアがもらえる. テトリミノの一部がフィールドの上限を超えた場合, ゲームが終了する. ラインを消したときに, 同時に複数のラインを消すと高得点となり, 特に 4 行を同時に消したときは大きなスコアが得られる. このため高いスコアを目指すときは多段消しを狙う必要がある. しかし, 多段消しを狙うと落ちてくるミノのパターンによってはなかなかラインを消すことができず, ゲームオーバーになる可能性が高くなる. そのため, 状況によってはラインを 1 行ずつ消してフィールド内のミノを減らすことも必要になる. テトリスの戦略は 2 つ考えられる. 1 つ目は 4 行をまとめて消していくことを優先してミノを積んでいく多段消し戦略である. 2 つ目はできるだけ早くミノを消すことを優先する積み方をする速攻戦略である. そこで本研究では, 2 つの戦略をテトリス AI に組み込み, どのような条件のときにそれぞれの戦略が有利となるかを検証する. この結果から, 遺伝的アルゴリズムでは, どのような戦略が成長するのに最適かを検証する.

目次

1.1 序論

1.2 本研究の背景

テトリスは,縦 20 行,横 10 行のフィールドに画面上部から落ちてくる 4 つの正方形を組み合わせた 7 種類の形のテトリミノ(ブロック)を左右に動かしたり,回転させたりして配置し，積み上げていくゲームである.図 1 にテトリスで出現する 7 種類のテトリミノを示す．テトリミノを並べて横一列のラインをつくると,テトリミノが消えてスコアがもらえる.テトリミノの一部がフィールドの上限を超えた場合,ゲームが終了する.

テトリスでは最大 4 列をまとめて消すことが出来る.これが最大のスコアを得ることが出来,高いスコアを目指すときは多段消しを狙う必要がある.しかし,どの種類のテトリミノが出現するかはランダムであり，出現するテトリミノの種類は，現在操作中のテトリミノの次に出現するテトリミノしかわからない．出現するテトリミノの種類はランダムであるため,多段消しを狙ってテトリミノを積み上げているといつまでも狙った種類のテトリミノが出現せず,ゲームオーバーになる可能性が高くなる.そのため,どのように積めば,最適となるかは今後出現するブロックに依存するため,画面上の情報だけでは最適解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テトリミノの種類が分かると仮定しても，最も消せる列の数が多くなるテトリミノの並べ方，および最もスコアが高くなるテトリミノの並べ方を求める問題は NP 完全である[4]．そのため,常に 4 段消しを狙っていく戦略が最もスコアを高くとれるかはわからない.

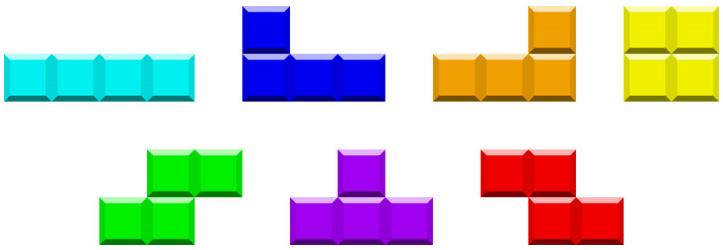

図 1 テトリミノの種類

1.3 テトリスに関する既知の結果

テトリスは，今後出現する全てのテトリミノの種類が分かっていたとしても，最も消せる列の数が多くなるテトリミノの並べ方，および最もスコアが高くなるテトリミノの並べ方を求める問題は NP 完全であることが Demeine により示されている[4]．また，テトリミノ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 字型のテトリミノ(以降 Z ミノ，S ミノと呼ぶ)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[5].

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[6]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[7]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 ミノと 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 ミノと Z ミノの出現頻度を高めゲームが終了しやすくする[7]など学習を工夫する必要がある.

テトリス AI は最適化問題として定式化することが可能であるため,これまでに様々な最適化手法が適応された.その中で最も良い結果を報告しているのが Thiery らおよび Boumaza である.Thiery らは,表 1 に示す 8 個の特徴量を用いた線型の評価関数を cross-entropy method を用いて最適化することで,平均 3500 万ラインの性能を示すテトリスコントローラの開発に成功した.また Boumaza は covariance matrix adaptation evolution strategy を適用することで,平均 3630 万ラインを達成した[8].

表 1 テトリスコントローラで用いられる特徴量[8]

名称	説明
LandingHeight	直前に置いたピースの高さ
ErodedpieceCells	消えたラインの数×ピースの中で消えたブロックの数
RowTransitions	横方向にスキャンした時,セルの内容が変わる回数
ColTransitions	縦方向にスキャンした時,セルの内容が変わる回数
NumHoles	穴の数
CumulativeWells	井戸の高さの階和
HoleDepth	穴の上のブロック数
RowsWithHoles	穴のある行

1.4 本研究の目的

テトリスの戦略は 2 つ考えられる.1 つ目は最大 4 列をまとめて消していくことだけを考えて,テトリミノを積んでいき,多段消しを狙う戦略である. 2 つ目は 1 行消し以上を消し続ける積み方をする戦略である.1.1 節で述べたように, テトリスで最もスコアが高くなるテトリミノの詰み方を求める問題は NP 完全であり, 最適な詰み方を得ることは難しい. そこで本研究では, 遺伝的アルゴリズムを用いてどのような戦略が有効であるかを検証する.

1.5 本報告書の構成

2 節以降の各節の内容を簡潔に記述する.
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と遺伝的アルゴリズムについて説明し,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,本研究の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社アレクセイ・パジトノフが開発した落ちものパズルとされるコンピュータゲームである[6]. テトリスは,縦 20 行,横 10 行のフィールドに画面上部から落ちてくる図 1 に示す 4 つの正方形を組み合わせた 7 種類の形のテトリミノ(ブロック)を左右に動かしたり,回転させたりして,配置し,積み上げていくゲームである.テトリミノで横一列のラインをつくるとその列のテトリミノが消えてスコアがもらえる.テトリミノの一部がフィールドの上限を超えた場合,ゲームが終了する. ラインを消したときに, 同時に複数のラインを消すと高得点となり, 特

に 4 行を同時に消したときは大きなスコアが得られる.

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.以下に本研究でのテトリスのルールを挙げる.

- 本研究では,テトリスのフィールドを縦 20 行,横 10 行とする.
- テトリミノは落下中のものまで確認できるものとする.
- プレイヤーができる操作は左移動,右移動,左に 90° 回転,右に 90° 回転である.
- テトリミノの種類は図 1 の 4 つの正方形を組み合わせた 7 種類である.
- ラインを消した時には, 消したライン数に応じて表 2 に示すスコアが得られる.
- テトリミノの一部がフィールドの上限を超えた場合ゲームオーバーとする.

表 2 スコア表

消したライン数	得られるスコア
1 ライン	100
2 ライン	300
3 ライン	500
4 ライン	800

2.3 遺伝的アルゴリズム

遺伝的アルゴリズムとは生物の進化の仕組みを模した,最適化のためのアルゴリズムである. 生物の進化の過程で起きる「環境に適応し,より強い個体が生き残り,環境に適応できない弱い個体は淘汰される」という現象から,プログラム上で優秀な個体を次世代へと受け継ぐ仕組みが遺伝的アルゴリズムだ.遺伝的アルゴリズムには「組合せ最適化問題」というのがある.「組合せ最適化問題の例としてはナップサック問題がある.ナップサックに入れたいいくつかの候補があらかじめ用意されているとする.そして,その候補の価値と重さ,大きさなどは事前にわかっている状態で,容量に制限のあるナップサックにトータルで最も価値の高いものの入れ方を求めるという問題である.ナップサック問題に代表される組み合わせ最適化問題は, 多くが NP 完全問題であり, 最適解を得ることは難しいとされる. しかし, 遺伝的アルゴリズムを用いることで, 最適解では無いが最適解に近い解を求めることができる.

遺伝的アルゴリズムを考えるうえで「遺伝子」は必要不可欠である.遺伝的アルゴリズムにおいての遺伝子とは,パラメータや解のことである.遺伝子の選択の仕方はランダムに生成した大量の遺伝子を評価し,成績上位のものだけが,次世代の子孫を作ることができる方法や,上位 20%を残す方法,成績がいいほど次世代に遺伝子を残す確率が高くなる方法など,選択の仕方はいろいろある.

生き延びた遺伝子は,自分のコピーを作って子孫を作るが,ただコピーを作るだけでは,それ以上優秀な遺伝子は残せない.生物には, 遺伝子をシャッフルする仕組みとして,「遺伝子乗り換え」(クロスオーバー) という現象がある.遺伝的アルゴリズムでも,このクロスオーバーと似た仕組みを使って,

遺伝子をシャッフルする.

クロスオーバー以外のシャッフルの仕組みに突然変異がある. 突然変異とは生物体に, 親の系統になかった形質が突然生じ, ある遺伝子がランダムに変わってしまう現象のことだ. 実際にはかなりの低確率でしか起こらない. しかし, まれに優秀な遺伝子に突然変異することがある. 遺伝的アルゴリズムは, 最初に設定する個体の組み合わせによっては, 最適解とは異なる局所最適解に陥ってしまうことがある. そこで適度に遺伝子の突然変異を起こすことで局所最適解から離れ, 新たな最適解を探すことができるようになる.

このようなシャッフルを行なって, 次世代の遺伝子を生成し, 再び採点をして, 優れた個体を子孫として残す. これを繰り返していくことで, 何世代か後には得点が延びなくなっていく. これが収束で解が求められるようになっている.

本研究では, 各世代においてモデルの評価が行われ, 最も優れた個体(上位 20%)はそのまま次世代に残し, 残りの 60%については, 親世代からランダムに選んだ 2 つの個体間で交叉して, 次世代が生成される. また残る 20%については, 親世代とは全く異なる遺伝子を持った突然変異の個体を生成する. 本研究での遺伝の仕方を図 2 に示す.

遺伝的アルゴリズムには二つの欠点が挙げられる. 1 つ目は過剰適応だ. 過剰適応とはある水準以上の解を求めようとしたとき, 偶然その水準に近い解を求められた場合, その求めようとする水準以下で解が収束してしまうことである. 2 つ目はヒッチハイク問題だ. ヒッチハイク問題とは, 生物が交配によって子孫を残すことをモデル化したものにおいて, 最適解の生成を妨げる遺伝をしてしまうことである. [9]

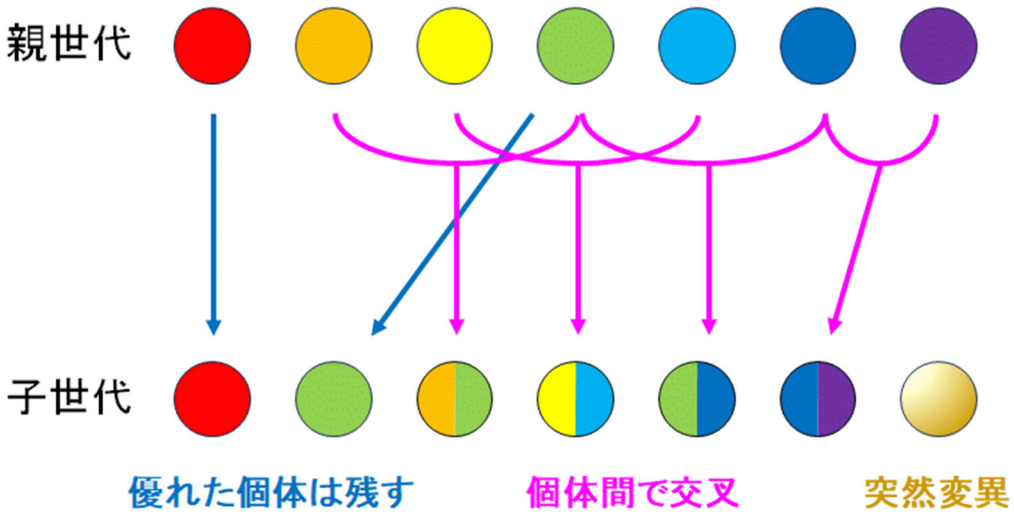

図 2 遺伝の仕方

3　テトリスのプログラム

本研究では, 遺伝的アルゴリズムにより多段消し戦略を学習させたテトリス AI および速攻戦略を学習させたテトリス AI を Python を用いて作成した. 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. このテトリス AI では, 遺伝的アルゴリズムを使用して進化させられるニューラルネットワークモデルを学習する. まず, 遺伝的アルゴリズムの管理をし, 個体数, 交叉, 突然変異を実行し, 次世代のモデルを生成する. 各個体は異なるモデルのパラメータセットを表し, それによってテトリスのプレイの戦略が異なる. 多段消し線略の時は行遷移数と

列遷移数の最小化と凸凹数やブロックが 1 つも無い列数の最小化,最大井戸高さの最小化,穴の数の最小化を行う.行遷移数と列遷移数が少ないほど,テトリミノが盤面に密着しているため,ブロックを積み上げることが容易になる.密着しているほど,隙間なく積み上げることができ,穴や井戸ができにくくなります.凸凹が少なく,ブロックが一切ない列が多いほど,テトリミノを滑らかに積み上げることができる.凸凹があると,テトリミノを配置する際に積み重なりが生じ,ブロックの平坦な配置が難しくなる.　井戸は穴の深さのことであり,最大井戸高さを小さくすることで,多段消しの機会が増える.井戸が深いと,テトリミノを垂直に積むことが難しくなり,消去ラインの形成が制限される.穴が少ないほど,テトリミノを落とす場所が限られ,多段消しのチャンスが増える.井戸とは列の一部が他の列よりも低い位置になっている状態で,図 2 の場合では 9 列目一番右の行に高低差 3 の穴があり,このような状態のことを井戸という.の穴があると,テトリミノを配置するときに穴の位置に注意が必要であり,穴がないほど効率的にテトリミノを積み上げることができる.テトリミノの動きに関しては与えられたモデルと盤面から,テトリミノが落下する最適な位置を計算して,それぞれの位置出のスコアを計算,スコアが最も高い位置を選択して,その位置にテトリミノを落下させる.これにより,テトリミノを盤面に最適に配置し,評価関数に基づいて盤面を良い状態に保つような動きが行われる.多段消しをした場合にはスコアに多段ボーナスを与える.速攻戦略の時は多段消し戦略とは反対に行遷移数と列遷移数,凹凸数やブロックが 1 つもない列数,最大井戸高さ,穴の数を最大化する.　また,消去行数を優先し,その後に他の項目を考慮するようにした.

プログラムを実行すると,図 2 の通りにテトリス AI が自動でテトリスをプレイして,個体数を 50 とし,各世代においてモデルの評価が行われ,最も優れた個体(上位 20%)とランダムな 2 つの個体が選ばれ,最も優れた個体はそのまま次世代に,ランダムな 2 つの個体で交叉して,次世代が生成される.各個体のスコアと世代ごとにテトリスを 5 回プレイさせ,そのスコアと削除したライン数の平均をとり,その世代の結果を出す.上位の個体はそのまま次世代に残し,それ以外の個体はランダムに 2 つの個体を選び,一様交叉を行う.これを第 10 世代まで交叉を繰り返した.

図 3 テトリス AI のプレイ画像

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 3 にインポートしてきたライブラリ，表 4 に設定した定数の一覧，表 5 にクラスおよび各処理の一覧を示す.

表 3 インポートしたライブラリ

ライブラリ名	説明
pygame	テトリスの画面の表示やゲームの制御とイベントの処理を行うためのライブラリ
numpy	多次元配列を扱うライブラリ
copy	オブジェクトをコピーするためのライブラリ
sys	Python の実行環境やシステムに関連する機能を提供するモジュール
random	乱数を生成するモジュール
torch	深層学習ライブラリである PyTorch の主要なモジュール
torch.nn as nn	PyTorch のニューラルネットワーク（Neural Network）を構築するための主要なツールが含まれているモジュール

表 4 定数一覧

定数名	説明	値
BLOCK_SIZE	各ブロックのサイズ(描画用サイズ)	20
BLOCK_IMG_SIZE	各ブロックのサイズ(原画(tile.png)の各サイズ)	40
elitism_pct	交叉を行う際に優秀な個体としてそのまま世代以降する割合	0.2
mutation_prob	突然変異を行う確率	0.2
weights_mutate_power	突然変異でのノイズ割合	0.5
input_size	PyTouch Network モデルの Input 次数	13
output_size	PyTouch Network モデルの Output 次数	1
weights_init_min	PyTouch 初期ウエイト下限	-1
weights_init_max	PyTouch 初期ウエイト上限	1
device	PyTouch は CPU を利用する	cpu

表 5 クラス及び各処理の一覧

クラス名	クラスの中のメソッド名	説明
Block		1 つのブロックを管理するクラス このブロックの 4 つの集合体でテトリミノを構成する.
	__init__(self, x, y, c)	引数 self は Block クラスのインスタンスで,x,y は初期位置で,c は色インデックスを表している. クラスのインスタンスが生成されるときに呼び出される初期化メソッドである.
	get_pos(self)	ブロックの現在の位置を返すためのメソッドである.
	move(self, x, y)	ブロックを指定された値だけ水平方向及び垂直方向に移動させるためのメソッドである.x と y が引数として渡され,これらの値が現在の位置に加算される.
	rotate(self)	ブロックを時計回りに 90 度回転させるためのメソッドである. 現在のブロックの位置 self.x,self.y を変数 x,y に保存する.次に self.x に-y,self.y に x を代入すると,ブロックの位置が時計回りに 90 度回転する.
	draw(self, screen, colors)	ブロックをゲーム画面に描画するためのメソッドである.
Tetorimino		テトリスのゲーム内で表示される 7 つの異なる形状（O,I,S,Z,J,L,T）のテトリミノ（ブロックの集まり）を管理するためのクラス
	__init__(self, x, y, r, t)	インスタンスの初期化を行うメソッド
	get_blocks(self)	現在のテトリミノのブロック 4 つを返すメソッド
	get_type(self)	テトリミノのタイプを返すメソッド
	collision(self, field)	壁や他のブロックとの接触判定を行うメソッ

		ド
	set_score(self, score)	スコアをセットするメソッド
	get_score(self)	現在のスコアを取得するメソッド
	draw(self, screen, colors)	画面にテトリミノを描画するメソッド 実際の描画処理は Blockk クラス内の draw メソッドで行う.
	clone(self, dx=0, dy=0, dr=0)	指定の移動を行ったあとのクローンを生成する.
Field		テトリスの盤面を管理するためのクラス.盤面上のブロックの配置や得点計算などの機能がある.
	__init__(self):	クラスの初期化を行うメソッド 盤面を表す 2 次元の tiles 配列を生成し,高さ 20 マスの盤面を初期化する. score 属性は得点を保持するための変数. tiles は盤面を表す 2 次元配列で,各セルにはブロックの状態が格納されている.壁(9),空き(-1),ブロック(0-6)で構成される.
	get_tile(self, x, y)	指定された座標のブロックの状態を返す.
	set_blocks(self, blocks)	ブロックを盤面に確定反映させる. 指定されたブロックの位置を self.tiles に反映させる.
	line_erase(self)	埋まった行があればそれを消し,得点を計算する. 消去された行数と追加された得点を返す.
	draw(self, screen, colors)	Pygame の Screen オブジェクトに盤面を描画する.
	get_bit_field(self, candidate_blocks=None)	盤面の 2 値の盤面値（埋まっている: 1, 空いている: 0）を返す.
	get_field_score(self, candidate_blocks=None)	盤面の評価値を返す. candidate_blocks で現在の盤面に特定のテトリミノが次に配置されたと仮定した状況で評価する. 戻り値の 9 つ の要素からなる Numpy 配列.各要素は特定の盤面評価指標を表している. この盤面評価指標を変えることでで戦略を立てている.
	get_peaks(area)	テトリスのフィールド内の各列の最大の高さを計算している. 各列をイテレートし,その列内で最も高い占有セルを見つける. 結果は,各列の最大の高さを含む配列 pesks.
	get_holes(peaks, area)	テトリスのフィールド内の各列の穴（空白のスペース）の数を計算している. get_peaks から得られたピークの高さに関する情報を使用して,各列での穴を確認する開始ポイントを決定する. 結果は,各列の中の穴の数を含む配列 holes.
	get_row_transition(area, highest_peak)	テトリスのフィールド内の合計行遷移数を計算している.

		行遷移は,同じ行内で占有セルから空のセルへの変更,またはその逆が発生したとき. パラメータ highest_peak はテトリスのフィールドの最大の高さであり,関数はその高さ以上の行のみを考慮する.
	get_col_transition(area, peaks)	テトリスのフィールド内の合計列遷移数を計算している. 列遷移は,同じ列内で占有セルから空のセルへの変更,またはその逆が発生したとき. get_peaks から得られたピークの高さに関する情報を使用し,ピークの高さが 1 の列はスキップする.
	get_bumpiness(peaks)	隣接する列間の高さの差の絶対値の合計を計している.
	get_wells(peaks)	各列における最も深い井戸の深さを計算している.
play		Pygame を使用して実装したシンプルなテトリスゲームのクラス
	make_colors()	7 色のブロック画像を読み込み,それぞれの色を表す pygame.Surface オブジェクトを作成する.
	generate_tetrimino()	ランダムにテトリミノを生成する.
	play()	メインのゲームプレイが行われる. Pygame を初期化し,ゲーム画面を設定する. また,テトリスの盤面やテトリミノの初期化などを行う. キーボード操作によってテトリミノの移動や回転を制御し,得点の計算やゲームオーバー判定も含む.
main		遺伝的アルゴリズムを使用してテトリスの AI を進化させるクラス
	get_candidate_list(init_mino, field)	与えられたテトリミノと盤面の状態に基づいて,そのテトリミノが可能な位置における候補のリストを作成する. 各テトリミノ候補は,落下の高さに応じて score 値がセットされる.
	get_next(model, mino, field)	与えられたモデルと盤面の状態に基づいて,テトリミノの落下位置を計算する. モデルはニューラルネットワークであり,各候補の盤面の評価を行って,スコアが最も高いテトリミノを選択する.
	eval_network(model)	モデルを自動で 3 回プレイさせ,その結果のスコアの平均値を返す. これはモデルの適応度（fitness）を評価する.
	preview_ai(model)	モデルを使用してテトリスを自動でプレイし,その結果のスコアと消去行数を返す. 移動のアニメーションは行われず,テトリミノが生成された瞬間にモデルから算出された落下位置に一気に移動する.
	main()	メインの実行関数で,遺伝的アルゴリズムによりモデルを進化させる. 各世代の最もスコアの高い個体に対して,プ

		レビューを 5 回行いその平均スコアを出力する.
Network(nn.Module)		nn.Module クラスを継承している. これは PyTorch のモデルの基本クラス 単純な 1 層の線形関数からなるネットワーク. 入力サイズは 13 (input_size) ,出力サイズは 1 (output_size) バイアスは使用せず. 重みは初期化時に一様分布で設定される.
	__init__(self)	ネットワークの初期化メソッド nn.Linear を使用して 1 層の線形関数を構築する. 入力サイズと出力サイズは引数から取得し, バイアスは使用しない. 逆伝搬は今回不要なので, self.output.weight.requires_grad_(False)で重みの勾配計算を無効にしている. 重みは一様分布で初期化される.
	activate(self, x)	モデルに入力ベクトルを与え, 評価値を取得するメソッド torch.from_numpy(x).float().to(device) で Numpy 配列を PyTorch テンソルに変換し, デバイス (CPU) に送る. モデルによって評価を行い, その結果を 1 次元のテンソルとして返す.
Population		遺伝的アルゴリズムの各世代を管理するクラス
	__init__(self, size=50, old_population=None)	クラスのイニシャライザ size は各世代の個体数を指定する. old_population が与えられた場合, 交叉 (crossover) および突然変異 (mutate) を行って次世代の個体を生成する. それ以外の場合は, 全ての個体を初期値で生成する.
	crossover(self)	交叉 (crossover) メソッド ルーレット選択法を使用して, 各個体の評価値に基づいて交叉を行う. 上位の一部の優秀な個体はそのまま (エリート主義) ,それ以外の個体はランダムに選ばれた 2 つの個体から交叉によって新しい個体を生成する.
	mutate(self)	突然変異 (mutate) メソッド 各個体に対して一定の確率で突然変異を行う. 突然変異が発生すると, モデルの重みに一定のノイズが加えられる.

4 結果とまとめ

本研究では, 多段消し戦略を学習させたテトリス AI および速攻戦略を学習させたテトリス AI それぞれに対して 5 回プレイし, 得られたスコアおよび消したライン数を計測した. 表 6 に各戦略に対する世代ごとの平均スコアおよび平均ライン数を示す. 多段消し戦略では, 世代が進むにつれて平均スコアと平均ライン数が増加している. 第 6 世代まではスコアが順調に向上している. しかし, 第 7 世代以降は成長があまり見られないので, 第 6 世代で成長限界に達したと考えられる. 具体的には, 積み上げが

過剰になり,ゲームオーバーに至っていると考えられる.多段消し戦略は局所的な最適解に収束しやすい傾向があることが考えられ.速攻戦略では,初期から高いスコアを達成しており,第 2 世代以降から,成長をしていないと考えられる. 速攻戦略はラインを優先的に消す傾向があり,これが初期段階での高スコアを得られていると考えられる.成長の鈍化は,この戦略が特定の局面において最適解に収束していると考えられる.多段消し戦略の方が速攻戦略よりも進化が制約されていると考えられる.これらのことから,多段消し戦略では順調にスコアが上がっているので,初期の段階では速攻戦略が有効であるが,ゲームが進行するにつれて多段消し戦略が有効になる可能性がある.

表 6　世代ごとの平均スコアと平均ライン数(試行回数各 5 回)

	多段消し戦略		速攻戦略	
世代	スコア	ライン	スコア	ライン
1 世代	150	0.0	200	0.0
2 世代	3500	24	81000	540
3 世代	23000	170	21000	130
4 世代	7500	54	56000	350
5 世代	13000	90	32000	200
6 世代	28000	200	66000	420
7 世代	13000	97	43000	290
8 世代	42000	300	80000	540
9 世代	31000	220	74000	510
10 世代	13000	90	42000	290

5　結論・今後の課題

本研究では遺伝的アルゴリズムを用いた機械学習により自動的にテトリスをプレイする AI を作成した. 多段消し戦略では,第 6 世代まではスコアが順調に向上している.しかし,第 7 世代以降は成長があまり見られないので,第 6 世代で成長限界に達したと考えられる.速攻戦略では,初期の世代から高いスコアを達成しているが,その後の成長がやや鈍化している.本研究から,他のゲーム AI で遺伝的アルゴリズムを使った時にどのような戦略のほうが成長するか推測できる.本研究で提案された速攻戦略と多段消し戦略は,他のゲームにも応用可能であると考えられる.具体的な例として,ぷよぷよやパズル玉などの落ち物系ゲームにおいては,同様の戦略が適用できると考えられる.これらの戦略は即座な行動と積み上げに焦点を当てており,そのままの形で利用できる可能性が高い.他の例として,セブンブリッジといったカードゲームでは,速攻戦略が手札を迅速に出す戦略に相当し,一方で多段消し戦略は手札をまとめて高得点を狙う戦略に適用できる可能性がある.
更なる応用として,格闘ゲームにおいても考えられる.小技で相手の体力を少しずつ削る速攻戦略と,大技で一発逆転を狙う多段消し戦略の対比が存在し,これらの戦略が適用可能であるかもしれない.

今後の課題としては,他の戦略や特徴を導入することで,より高度な戦略を学習でき,成績の向上が期待できる.他の戦略はブロックの段差を最小限に抑える戦略や穴を作らないようにすることを目指す戦略などが考えられる.また,実験時間を測ることで戦略の新たな特徴を知ることができると考えられる.

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] 藤本晴生：TD 学習によるテトリス AI の開発，近畿大学理工学部情報学科 2022 年度卒業研究報告 (2023).

[2] 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,情報処理学会 (2017).

[3] hiyuzawa:遺伝的アルゴリズムでテトリス風ゲームを PLAY させる, https://hiyuzawa.jp/archives/553 (2021-06-10)

[4] 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

[5] 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

[6] 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)

[7] 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

[8] 宮崎真奈美,荒川正幹: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(2012-03-06)

[9] 牧野 武文:遺伝的アルゴリズムとは?用いられる場面やアルゴリズムの欠点徹底解説! https://staff.persol-xtech.co.jp/corporate/security/article.html?id=63(2022-10-24)

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.

config.py を示す.

```
1   BLOCK_SIZE = 20              # 各ブロックのサイズ(描画用サイズ)
2   BLOCK_IMG_SIZE = 40          # 各ブロックのサイズ(原画(tile.png)の各サイズ)
3
4   elitism_pct = 0.2            # 交叉を行う際に優秀な個体としてそのまま世代以降する割合
5   mutation_prob = 0.2          # 突然変異を行う確率
6   weights_mutate_power = 0.5   # 突然変異でのノイズ割合
7
8   input_size = 13               # PyTouch NetworkモデルのInput次数
9   output_size = 1              # PyTouch NetworkモデルのOutput次数
10  weights_init_min = -1        # PyTouch 初期ウエイト下限
11  weights_init_max = 1         # PyTouch 初期ウエイト上限
12  device = 'cpu'               # PyTouch は CPU を利用する
```

block.pyを示す.

```
1  from config import BLOCK_SIZE
2
3
4  class Block:
5      """ 1つのブロックをこれで管理するクラス
6
7      テトリスのゲーム内ブロック(４ブロックの塊)は Tetrimino で集合体とするが
8      このBlock４つでそれが構成される。
9      上下左右の動作および回転はこのクラスで１ブロック単位で行う
10     """
11
12     def __init__(self, x, y, c):
13         """ イニシャライザ
14
15         :param x: 初期位置 x
16         :param y: 初期位置 y
17         :param c: 色インデックス
18         """
19         self.x = x
20         self.y = y
21         self.c = c
22
23     def get_pos(self):
24         """ 現在の位置を返す
25
26         :return: ブロックの位置
27         """
28         return self.x, self.y
29
30     def move(self, x, y):
31         """x, y づつ移動する
32
33         :param x: x移動値
34         :param y: y移動値
35         :return:
36         """
37         self.x += x
38         self.y += y
39
40     def rotate(self):
41         """ 回転させる
42         時計回りのみサポート
43         :return:
44         """
45         x = self.x
```

```
46              y = self.y
47              self.x = -y
48              self.y = x
49
50          def draw(self, screen, colors):
51              """ 盤面にブロックを描画する
52              :param screen: PyGame Screen オブジェクト
53              :param colors: PyGame Surface オブジェクト(ブロック用の色配列)
54              :return:
55              """
56              screen.blit(colors[self.c], (BLOCK_SIZE*self.x, BLOCK_SIZE*self.y, BLOCK_SIZE,
57      BLOCK_SIZE))
```

tetorimino.pyを示す.

```
1  from block import Block
2
3
4  class Tetrimino:
5      """ Tetrimino(テトリスでは各ブロック(全７種) をこう呼ぶらしいを管理するクラス
6
7      O(四角)
8      I(棒)
9      S
10     Z
11     J
12     L
13     T
14     """
15
16     def __init__(self, x, y, r, t):
17         """ イニシャライザ
18
19         初期位置とタイプでインスタンス化する
20         どのTetriminoもBlockクラス4つの集合（初期ブロックの配置で各々の形をつくる）
21
22         :param x: 初期位置 x
23         :param y: 初期位置 y
24         :param r: 初期回転 r
25         :param t: テトリミノのタイプ
26         """
27         self.x = x
28         self.y = y
29         self.r = r
30         self.t = t
31         self.s = 0
32         if t == 0:
33             # O (四角)の形のブロック
34             self.shape = [Block(-1, -1, t), Block(-1, 0, t), Block(0, 0, t), Block(0, -
35 1, t)]
36         if t == 1:
37             # I の形のブロック
38             self.shape = [Block(-2, 0, t), Block(-1, 0, t), Block(0, 0, t), Block(1, 0,
39 t)]
40         if t == 2:
41             # S の形のブロック
42             self.shape = [Block(-1, 0, t), Block(0, 0, t), Block(0, -1, t), Block(1, -1,
43 t)]
44         if t == 3:
45             # Z の形のブロック
```

```
46                  self.shape = [Block(-1, -1, t), Block(0, -1, t), Block(0, 0, t), Block(1, 0,
47  t)]
48          if t == 4:
49              # J の形のブロック
50              self.shape = [Block(0, -2, t), Block(0, -1, t), Block(-1, 0, t), Block(0, 0,
51  t)]
52          if t == 5:
53              # L の形のブロック
54              self.shape = [Block(-1, -2, t), Block(-1, -1, t), Block(-1, 0, t), Block(0,
55  0, t)]  # L
56          if t == 6:
57              # T の形のブロック
58              self.shape = [Block(-1, 0, t), Block(0, 0, t), Block(0, -1, t), Block(1, 0,
59  t)]     # T
60
61          for _ in range(r%4):
62              for b in self.shape:
63                  b.rotate()
64
65          for b in self.shape:
66              b.move(x, y)
67
68      def get_blocks(self):
69          """ 現在のブロック4つを返す
70
71          :return: 4つのBlockインスタンス配列
72          """
73          return self.shape
74
75      def get_type(self):
76          """ タイプを返す
77
78          :return: タイプ
79          """
80          return self.t
81
82      def collision(self, field):
83          """ 壁や他のブロックとの接触判定
84          field(盤面を引数にとり現在位置のBlockでfieldとの障害があれば当たり判定とする
85
86          :param field: 盤面
87          :return: True 当たり判定あり / False なし
88          """
89          for b in self.shape:
90              x, y = b.get_pos()
91              if x < 0 or x > 10:
```

```
92                  return True
93              tile = field.get_tile(x, y)
94              if tile != -1:
95                  return True
96          return False
97
98      def set_score(self, score):
99          """ スコアをセットする
100         :param score: スコア値(落下の高さ)
101         :return:
102         """
103         self.s = score
104
105     def get_score(self):
106         """ スコアを得る
107         :return: スコア値
108         """
109         return self.s
110
111     def draw(self, screen, colors):
112         """ 画面にブロックを描画する
113         実際の描画処理はBlock内で行うのでここではそれを呼び出すだけ
114
115         :param screen: PyGame Screen オブジェクト
116         :param colors: PyGame Surface オブジェクト(色ブロックの配列)
117         :return:
118         """
119         for b in self.shape:
120             b.draw(screen, colors)
121
122     def clone(self, dx=0, dy=0, dr=0):
123         """ 指定の移動を行った後のクローンを生成する
124
125         :param dx: x移動値
126         :param dy: y移動値
127         :param dr: r回転値
128         :return: インスタンスから指定の移動回転を行った後のTetriminoオブジェクト
129         """
130         return Tetrimino(self.x+dx, self.y+dy, self.r+dr, self.t)
```

field.pyを示す.

```
1  import pygame
2  import numpy as np
3  import copy
4  from config import BLOCK_SIZE
5
6
7  class Field:
8      """ テトリスの盤面全体を管理するクラス
9
10     盤面の個々のブロックを
11     ・9 = 壁
12     ・-1 = 空き
13     ・0-6 = 埋まっているブロック(値はブロック色)
14     で表現する
15     """
16
17     EMPTY_LINE = [9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 9]
18     FLOOR_LINE = [9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
19
20     def __init__(self):
21         """ イニシャライザ
22
23         self.tile が盤面を表す2次元配列
24         高さは20マス
25         """
26         self.tiles = [self.FLOOR_LINE]
27         for _ in range(20):
28             self.tiles.insert(0, copy.deepcopy(self.EMPTY_LINE))
29
30         self.score = 0  # 得点を保持する属性
31
32     def get_tile(self, x, y):
33         """ 指定のタイルのブロック状況を返す
34
35         :param x: x座標
36         :param y: y座標
37         :return: 与えられた座標にセットされている盤面の値
38         """
39         return self.tiles[y][x]
40
41     def set_blocks(self, blocks):
42         """ ブロックを盤面に確定反映する
43
44         :param blocks:
45         :return:
```

```
46          """
47          for block in blocks:
48              self.tiles[block.y][block.x] = block.c
49
50      def line_erase(self):
51          """ 埋まった行があればそれを消す関数
52
53          :return: 消去された行数
54          """
55          erase = []
56          for y in range(len(self.tiles)-1):  # 全ての行について捜査する
57              flag = True
58              for x in range(len(self.tiles[0])):   # 横一列ブロックがすべて埋まっているか
59  調べる
60                  if self.tiles[y][x] == -1:
61                      flag = False
62              if flag:  # flag = True である場合には埋まっているので消去リストに追加
63                  erase.append(y)
64          # 埋まったラインがあった場合
65          if erase:  # 埋まったラインがあった場合
66              for i in erase:
67                  self.tiles.pop(i)  # そのラインを取り除く
68                  self.tiles.insert(0, copy.deepcopy(self.EMPTY_LINE))  # 最上部に新たなラ
69  インを追加する
70                  #ここから
71                  # 追加の得点を計算し、得点に加算する
72                  erased_lines = len(erase)
73                  additional_score = 0
74                  if erased_lines > 0:
75                      # 消したライン数に応じて得点を設定
76                      if erased_lines == 1:
77                          additional_score = 100
78                      elif erased_lines == 2:
79                          additional_score = 300
80                      elif erased_lines == 3:
81                          additional_score = 500
82                      elif erased_lines == 4:
83                          additional_score = 800
84
85                  self.score += additional_score
86                  return len(erase), additional_score#ここまで変更点
87          else:
88              return 0, 0  # 消去された行がない場合はゼロを返す
89
90      def draw(self, screen, colors):
91          """ 画面に盤面を描画する
```

```
92
93          :param screen: pygame の Screen オブジェクト
94          :param colors: pygame の Surface (描画用の色ブロック)
95          :return:
96          """
97          for y in range(len(self.tiles)):
98              for x in range(len(self.tiles[0])):
99                  if self.tiles[y][x] == 9:
100                     pygame.draw.rect(screen, (80, 80, 80), (x*BLOCK_SIZE, y*BLOCK_SIZE,
101 BLOCK_SIZE, BLOCK_SIZE), 0)
102                 elif self.tiles[y][x] == -1:
103                     pygame.draw.rect(screen, (80, 80, 80), (x*BLOCK_SIZE, y*BLOCK_SIZE,
104 BLOCK_SIZE, BLOCK_SIZE), 1)
105                 else:
106                     screen.blit(colors[self.tiles[y][x]], (BLOCK_SIZE*x, BLOCK_SIZE*y,
107 BLOCK_SIZE, BLOCK_SIZE))
108
109     def get_bit_field(self, candidate_blocks=None):
110         """ 盤面の評価用の-1, 1 の2値の盤面値を返す関数
111
112         :param candidate_blocks: 現在の盤面に特定のテトリミノが次に配置されたと仮定した
113 状況で応答する
114         :return: 盤面の2次元配列(埋まっている:1, 空いている:0). (壁は含めない)
115         """
116         bit_field = [[0 if self.tiles[y][x] == -1 else 1 for x in range(1,
117 len(self.tiles[0])-1)] for y in range(0, len(self.tiles)-1)]
118         if candidate_blocks:
119             for block in candidate_blocks:
120                 bit_field[block.y][block.x-1] = 1
121         return bit_field
122
123     def get_field_score(self, candidate_blocks=None):
124         """ 盤面の評価値を返す
125
126         :param candidate_blocks: 現在の盤面に特定のテトリミノが次に配置されたと仮定した
127 状況で評価する
128         :return: 9つの1次元配列 (Numpy array型)
129         https://ichi.pro/identeki-arugorizumu-de-tetorisu-gb-no-sekai-kiroku-o-yaburu-
130 118795294920347
131
132         戻り配列詳細:
133         0: 高さ総合計
134         1: 穴数合計
135         2: 少なくとも1つ以上穴のある列数
136         3: 行遷移数
137         4: 列遷移数
```

```
138             5: 凸凹数
139             6: ブロックが1つも無い列数
140             7: 最大井戸高さ
141             8: 消去行数
142             """
143             area = np.array(self.get_bit_field(candidate_blocks))
144             peaks = self.get_peaks(area)
145             highest_peak = np.max(peaks)
146             agg_height = np.sum(peaks)
147 
148             holes = self.get_holes(peaks, area)
149             n_holes = np.sum(holes)
150             n_cols_with_holes = np.count_nonzero(np.array(holes) > 0)
151 
152             row_transitions = self.get_row_transition(area, highest_peak)
153             col_transitions = self.get_col_transition(area, peaks)
154 
155             bumpiness = self.get_bumpiness(peaks)
156 
157             num_pits = np.count_nonzero(np.count_nonzero(area, axis=0) == 0)
158 
159             wells = self.get_wells(peaks)
160             max_wells = np.max(wells)
161 
162             cleard = area.min(axis=1).sum()
163             #ここから
164             #多段消し戦略
165            # multilayer_bonus = np.count_nonzero(peaks >= 3)*200
166 
167             #return np.array([agg_height, n_holes, n_cols_with_holes,
168             #                 row_transitions, col_transitions, bumpiness,
169 num_pits,max_wells, cleard + multilayer_bonus,
170             #                  -row_transitions, -col_transitions, -bumpiness, -n_holes])
171 
172             #速攻戦略
173             return np.array([ cleard,agg_height, n_holes, n_cols_with_holes,
174                               row_transitions, col_transitions, bumpiness,
175 num_pits,max_wells,
176                               row_transitions, col_transitions, bumpiness, n_holes])#ここまで
177 変更点
178     @staticmethod
179     def get_peaks(area):
180         """高さ計算
181 
182         :param area: 盤面
183         :return: 各列での最大のブロックの高さ
```

```
184             """
185             peaks = np.array([])
186             for col in range(area.shape[1]):
187                 if 1 in area[:, col]:
188                     p = area.shape[0] - np.argmax(area[:, col], axis=0)
189                     peaks = np.append(peaks, p)
190                 else:
191                     peaks = np.append(peaks, 0)
192             return peaks
193 
194         @staticmethod
195         def get_holes(peaks, area):
196             """ 穴数計算
197 
198             :param peaks: get_peaksで得られた値(各列での最大の高さ)
199             :param area: 盤面
200             :return: 各列での穴(空白)の数
201             """
202             holes = []
203             for col in range(area.shape[1]):
204                 start = -peaks[col]
205                 if start == 0:
206                     holes.append(0)
207                 else:
208                     holes.append(np.count_nonzero(area[int(start):, col] == 0))
209             return holes
210 
211         @staticmethod
212         def get_row_transition(area, highest_peak):
213             """行遷移数
214 
215             :param area: 盤面
216             :param highest_peak: 最大のブロック高さ
217             :return: 各行の占有タイルから非占有タイルへの遷移の合計数
218             """
219             s = 0
220             for row in range(int(area.shape[0] - highest_peak), area.shape[0]):
221                 for col in range(1, area.shape[1]):
222                     if area[row, col] != area[row, col - 1]:
223                         s += 1
224             return s
225 
226         @staticmethod
227         def get_col_transition(area, peaks):
228             """列遷移数
229 
```

```
230             :param area: 盤面
231             :param peaks: get_peaksで得られた値(各列での最大の高さ)
232             :return: 各列の占有タイルから非占有タイルへの遷移の合計数
233             """
234             s = 0
235             for col in range(area.shape[1]):
236                 if peaks[col] <= 1:
237                     continue
238                 for row in range(int(area.shape[0] - peaks[col]), area.shape[0] - 1):
239                     if area[row, col] != area[row + 1, col]:
240                         s += 1
241             return s
242 
243         @staticmethod
244         def get_bumpiness(peaks):
245             """ 凸凹値
246 
247             :param peaks: get_peaksで得られた値(各列での最大の高さ)
248             :return: 隣り合う列間の絶対的な高さの差の合計
249             """
250             s = 0
251             for i in range(9):
252                 s += np.abs(peaks[i] - peaks[i + 1])
253             return s
254 
255         @staticmethod
256         def get_wells(peaks):
257             """ 最も深い井戸
258 
259             :param peaks: get_peaksで得られた値(各列での最大の高さ)
260             :return: 各列で最も深い井戸
261             """
262             wells = []
263             for i in range(len(peaks)):
264                 if i == 0:
265                     w = peaks[1] - peaks[0]
266                     w = w if w > 0 else 0
267                     wells.append(w)
268                 elif i == len(peaks) - 1:
269                     w = peaks[-2] - peaks[-1]
270                     w = w if w > 0 else 0
271                     wells.append(w)
272                 else:
273                     w1 = peaks[i - 1] - peaks[i]
274                     w2 = peaks[i + 1] - peaks[i]
275                     w1 = w1 if w1 > 0 else 0
```

```
                    w2 = w2 if w2 > 0 else 0
                    w = w1 if w1 >= w2 else w2
                    wells.append(w)
            return wells


if __name__ == "__main__":
    import pprint

    fields = Field()
    fields.tiles[16] = [9, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, 9]
    fields.tiles[17] = [9, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, 9]
    fields.tiles[18] = [9, -1, 1, 1, 1, -1, 1, -1, 1, -1, -1, 9]
    fields.tiles[19] = [9, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 9]
    print(fields.get_field_score())

    fields = Field()
    fields.tiles[16] = [9, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 9]
    fields.tiles[17] = [9, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 9]
    fields.tiles[18] = [9, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 9]
    fields.tiles[19] = [9, -1, 1, -1, -1, 1, 1, 1, -1, 1, -1, 9]
    print(fields.get_field_score())

    fields = Field()
    fields.tiles[16] = [9, -1, -1, -1, 1, -1, -1, -1, -1, -1, -1, 9]
    fields.tiles[17] = [9, -1, -1, -1, 1, -1, -1, -1, 1, 1, 1, 9]
    fields.tiles[18] = [9, -1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 9]
    fields.tiles[19] = [9, 1, 1, -1, 1, 1, 1, 1, -1, -1, 1, 9]
    print(fields.get_field_score())

    fields = Field()
    fields.tiles[16] = [9, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 9]
    fields.tiles[17] = [9, 1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 9]
    fields.tiles[18] = [9, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 9]
    fields.tiles[19] = [9, -1, 1, -1, -1, 1, 1, 1, -1, 1, -1, 9]
    print(fields.get_field_score())

    fields = Field()
    fields.tiles[15] = [9, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, 9]
    fields.tiles[16] = [9, -1, -1, 1, -1, 1, -1, 1, -1, -1, 1, 9]
    fields.tiles[17] = [9, -1, 1, 1, -1, 1, -1, 1, -1, -1, 1, 9]
    fields.tiles[18] = [9, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 9]
    fields.tiles[19] = [9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 9]
    print(fields.get_field_score())

    fields = Field()
```

```
322         fields.tiles[16] = [9, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, 9]
323         fields.tiles[17] = [9, -1, -1, 1, 1, 1, 1, 1, -1, -1, -1, 9]
324         fields.tiles[18] = [9, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 9]
325         fields.tiles[19] = [9, 1, 1, -1, 1, 1, 1, 1, -1, -1, -1, 9]
326         print(fields.get_field_score())
327 
328         fields = Field()
329         fields.tiles[16] = [9, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, 9]
330         fields.tiles[17] = [9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 9]
331         fields.tiles[18] = [9, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 9]
332         fields.tiles[19] = [9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 9]
333         print(fields.get_field_score())
334 
335         fields = Field()
336         print(fields.get_field_score())
```

play.pyを示す.

```
1  import pygame
2  import sys
3  import random
4  from field import Field
5  from tetrimino import Tetrimino
6  from config import BLOCK_SIZE, BLOCK_IMG_SIZE
7
8
9  def make_colors():
10     """7色のブロックをそれぞれ抽出する関数
11
12     tile.png に 40x280の画像があり、40x40で切り出し、最終的には20x20でリサイズ
13     :return: 7色のブロック画像配列 (pygame.Surface オブジェクト)
14     """
15     colors = []
16     image = pygame.image.load("./tile.png")
17     for i in range(7):
18         tmp_color_surface = pygame.Surface((BLOCK_IMG_SIZE, BLOCK_IMG_SIZE))
19         color_surface = pygame.Surface((BLOCK_SIZE, BLOCK_SIZE))
20         tmp_color_surface.blit(image, (0, 0), (BLOCK_IMG_SIZE * i, 0, BLOCK_IMG_SIZE,
21 BLOCK_IMG_SIZE))
22         pygame.transform.scale(tmp_color_surface, (BLOCK_SIZE, BLOCK_SIZE),
23 color_surface)
24         colors.append(color_surface)
25     return colors
26
27
28 def generate_tetrimino():
29     """テトリミノをランダムに1つ生成する関数
30
31     :return: ランダムに生成されたTetrimino インスタンス
32     """
33     random.seed()
34     return Tetrimino(5, 2, 0, random.randrange(0, 7))
35
36
37 def play():
38     """ Play関数
39
40     キーボード操作でテトリスを遊べる関数
41     :return:
42     """
43     pygame.init()  # pygameを初期化数する
44     screen = pygame.display.set_mode((640, 480))  # 画面サイズを640, 480に
45     pygame.display.set_caption("tetris_ai")
```

```
46      clock = pygame.time.Clock()  # フレームレート(描画頻度)を更新するためのタイマー
47      font = pygame.font.SysFont("Arial", 20)
48
49      # 7色ブロックの準備
50      colors = make_colors()
51
52      # 盤面の定義
53      field = Field()
54
55      # 1つめ(初期)のTetrimino(mino)
56      mino = generate_tetrimino()
57
58      # minoの落下が確定して次のminoを生成する必要がある場合にTrueとする
59      generate_flag = False
60
61      # ゲームオーバの場合にTrue
62      game_over_flag = False
63
64      # トータル経過フレーム数
65      num_frame = 0
66
67      # 得点(消したライン数)
68      score = 0
69      while True:   # ゲームのメインループ
70          num_frame += 1
71          screen.fill((0, 0, 0))  # マイループで画面を真っ黒に一度する
72
73          # 盤面の描画
74          field.draw(screen, colors)
75          mino.draw(screen, colors)
76
77          moved_flag = False # キー操作等によりminoが移動したことを表すflag
78          for event in pygame.event.get():  # イベントは一括で取得
79              if event.type == pygame.QUIT:  # 終了ボタンを押したときにpygameを閉じるのは
80  お作法
81                  pygame.quit()
82                  sys.exit()
83              if event.type == pygame.KEYDOWN:  # これよりキーボードのイベントを捕捉
84                  if event.key == pygame.K_RIGHT:  # 右が押された場合...
85                      next_mino = mino.clone(1, 0, 0)  # 右に移動したminoを仮に作成
86                      if not next_mino.collision(field):  # 当たり判定
87                          mino = next_mino  # 当たり判定無しであれば移動確定
88                          moved_flag = True
89                  if event.key == pygame.K_LEFT:
90                      next_mino = mino.clone(-1, 0, 0)  # 左に移動したminoを仮に作成(以下
91  同様)
```

```
                    if not next_mino.collision(field):
                        mino = next_mino
                        moved_flag = True
                if event.key == pygame.K_UP:
                    next_mino = mino.clone(0, 0, 1)  # 1回転したminoを仮に作成(時計回
り)(以下同様)
                    if not next_mino.collision(field):
                        mino = next_mino
                        moved_flag = True
                if event.key == pygame.K_DOWN:  # 下に移動(落下)は盤面的に可能なところま
で落ちる
                    next_mino = mino.clone(0, 1, 0)
                    while not next_mino.collision(field):  # 1ブロックづつ落下させ当たり
判定を繰り返す
                        mino = next_mino
                        next_mino = mino.clone(0, 1, 0)
                if event.key == pygame.K_SPACE:
                    pass

        if game_over_flag:  # game overであれば何もしない
            continue

        if moved_flag:  # キー操作(左右、回転）があった場合は 一旦その移動を完了し落下は
次フレームでやる（落下は除く）
            continue

        next_mino = mino.clone(0, 1, 0)  # minoを下に1つずらしてみる
        if next_mino.collision(field):  # 盤面との接触判定
            field.set_blocks(mino.get_blocks())  # 盤面にブロックを置く(落下停止)
            s = field.line_erase()
            if s != 0:
                score += s
                print("Score: {}".format(score))
            generate_flag = True
        else:
            mino = next_mino  # 盤面との接触が無いので下に1つ移動を確定する

        if generate_flag:
            mino = generate_tetrimino()
            if mino.collision(field):
                print("GAME OVER")
                game_over_flag = True
            generate_flag = False

        pygame.display.update()  # 描画を更新する
        clock.tick(3)  # フレームレートを3（秒間3回更新するぐらいの頻度に設定）
```

```
138
139
140 if __name__ == "__main__":
141     play()
```

main.pyを示す.

```
import pygame
import numpy as np
from population import Population
from field import Field
from play import make_colors, generate_tetrimino


def get_candidate_list(init_mino, field):
    """ 盤面の状況を踏まえて落下可能なテトリミノ候補をすべて返す

    :param init_mino: 与えられたテトリミノ
    :param field: 盤面
    :return: 落下可能な位置にあるテトリミノ配列（落下の高さに応じてscore値をセットする）
    """
    candidate = []
    rotete_num = 1
    mino_type = init_mino.get_type()

    # タイプによって回転のバリエーションが2 or 4 となる
    if 1 <= mino_type <= 3:
        rotete_num = 2
    if mino_type >= 4:
        rotete_num = 4

    # 各回転パターンに応じて
    for i in range(rotete_num):
        mino = init_mino.clone(0, 0, i)

        # まずは最も左に寄せる
        while True:
            next_mino = mino.clone(-1, 0, 0)
            if next_mino.collision(field):
                break
            mino = next_mino
        base_mino = mino

        while True:
            mino = base_mino
            score = 0
            while True:
                # 可能なところまで落下させる
                next_mino = mino.clone(0, 1, 0)
                if next_mino.collision(field):
                    break
                mino = next_mino
```

```
                score += 1
            mino.set_score(score)
            candidate.append(mino)

            # １つづつ右に移動させる
            next_mino = base_mino.clone(1, 0, 0)
            if next_mino.collision(field):
                break
            base_mino = next_mino
    return candidate


def get_next(model, mino, field):
    """ モデルと盤面から与えれたテトリミノが落下位置を計算する

    :param model: モデル
    :param mino:  移動対象のテトリミノ
    :param field: 現在の盤面
    :return: 落下後の盤面のスコアが最も高いテトリミノ(落下後位置)
    """

    # 回転や左右移動を行って落下可能な候補を全て計算する
    candidate_mino = get_candidate_list(mino, field)
    max_score = np.NINF
    best_mino = None
    for m in candidate_mino:
        # 各落下候補において落下後の盤面の評価を行う(スコア計算を行う)
        field_score = field.get_field_score(m.get_blocks())
        s = model.activate(field_score)
        if max_score < s:
            best_mino = m
            max_score = s
    # 最もスコア値の高い落下位置を戻す
    return best_mino


def eval_network(model):
    """ モデルを自動で3回Playさせてスコアの平均を応答する

    :param model: モデル
    :return: 3回プレイしたスコアの平均値
    """
    scores = []
    for i in range(10):
        field = Field()
        i += 1
```

```
92            score = 0
93            while True:
94                mino = generate_tetrimino()
95                if mino.collision(field):
96                    break
97                best_mino = get_next(model, mino, field)
98                field.set_blocks(best_mino.get_blocks())
99                field.line_erase()
100               score += best_mino.get_score()
101               scores.append(score)
102
103       return np.average(scores)
104
105
106   def preview_ai(model):
107       """ モデルを与えて自動でPlayする関数
108       移動のアニメーションは無し、テトリミノが生成されたらモデルから算出された
109       落下位置に一気に移動する
110
111       :param model: モデル
112       :return:
113       """
114       pygame.init()
115       screen = pygame.display.set_mode((640, 480))
116       pygame.display.set_caption("tetris_ai")
117       clock = pygame.time.Clock()
118       colors = make_colors()
119       font = pygame.font.SysFont("Arial", 40)
120
121       field = Field()
122
123       score = 0
124       erase_line = 0
125       game_over_flag = False
126       while True:
127           for event in pygame.event.get():
128               if event.type == pygame.QUIT:
129                   pygame.quit()
130                   return
131           if game_over_flag:
132               return score, erase_line
133
134           # ランダムにテトリミノを生成する
135           mino = generate_tetrimino()
136           screen.fill((0, 0, 0))
137
```

```
138             screen.blit(font.render("score: {}".format(score), True, (255, 255, 255)), (300,
139     100))
140             screen.blit(font.render("erase: {}".format(erase_line), True, (255, 255, 255)),
141     (300, 150))
142 
143             field.draw(screen, colors)
144             mino.draw(screen, colors)
145             pygame.display.update()
146             clock.tick(30)
147 
148             # 生成した時点で当たり判定であればGame Over
149             if mino.collision(field):
150                 game_over_flag = True
151 
152             # モデルと盤面の状況から落下位置を算出する
153             mino = get_next(model, mino, field)
154             field.set_blocks(mino.get_blocks())
155             field.draw(screen, colors)
156 
157             screen.fill((0, 0, 0))
158             field.draw(screen, colors)
159             pygame.display.update()
160 
161             # 消去可能な行があれば消去
162             erase, additional_score = field.line_erase()
163             if erase > 0:
164                 clock.tick(30)
165                 erase_line += erase
166                 score += additional_score
167             else:
168                 clock.tick(30)
169 
170             score += mino.get_score()
171 
172 
173     def main():
174         """メイン関数
175 
176         遺伝的アルゴリズムを用いて世代を繰り返す
177         各世代の最もスコアの高い個体はプレビューを5回行いその平均スコアをアウトプットする
178         :return:
179         """
180         pop_size = 50  # 各遺伝世代における個体数
181         population = Population(size=pop_size)
182         score = 0
183         lines = 0
```

```
184         for i in range(5):
185             s, l = preview_ai(population.models[0])
186             score += s
187             lines += l
188         print("score: {} line: {}".format(score/5, lines/5))
189 
190 
191         iteration = 0
192         while True:
193             iteration += 1
194             print("{} : ".format(iteration), end="")
195             for i in range(pop_size):
196                 population.fitnesses[i] = eval_network(population.models[i])
197                 print("*".format(iteration), end="")
198             print()
199 
200             print(population.fitnesses)
201             best_model_idx = population.fitnesses.argmax()
202             best_model = population.models[best_model_idx]
203             score = 0
204             lines = 0
205             for i in range(5):
206                 s, l = preview_ai(best_model)
207                 score += s
208                 lines += l
209             print("score: {} line: {}".format(score/5, lines/5))
210 
211             population = Population(size=pop_size, old_population=population)
212 
213 
214     if __name__ == "__main__":
215         main()
```

network.pyを示す.

```
1  import torch
2  import torch.nn as nn
3  from config import input_size, output_size, weights_init_max, weights_init_min, device
4
5
6  class Network(nn.Module):
7      """ PyTouchのモデル
8
9      単純な1層の線形関数のネットワーク
10     Input=9, Output=1 (InputはField.get_field_scoreで得られた盤面評価値(9次の
11 Numpy.Array)
12     """
13     def __init__(self):
14         super(Network, self).__init__()
15         # 線形関数１層のみ
16         self.output = nn.Linear(
17             input_size, output_size, bias=False).to(device)
18         # 今回は逆伝搬は必要ない
19         self.output.weight.requires_grad_(False)
20         # ウエイトを,一様分布で初期化
21         nn.init.uniform_(self.output.weight,
22                          a=weights_init_min, b=weights_init_max)
23
24     def activate(self, x):
25         """ 与えられたベクトルでモデル評価値を返す
26
27         :param x: 9次元ベクトル(Numpy.Array)
28         :return: 1次元ベクトル(評価結果)
29         """
30         with torch.no_grad():
31             x = torch.from_numpy(x).float().to(device)
32             x = self.output(x)
33         return x
```

population.pyを示す.

```
1  import numpy as np
2  import torch
3  from network import Network
4  from config import input_size, elitism_pct, mutation_prob, weights_mutate_power, device
5
6
7  class Population:
8      """ 遺伝的アルゴリズムの各世代を管理するクラス
9      """
10
11     def __init__(self, size=50, old_population=None):
12         """ イニシャライザ
13
14         :param size: 各世代の個体数
15         :param old_population: (次世代を生成する場合には前世代のPopulationを与える)
16         """
17         self.size = size
18         if old_population is None:
19             # 前世代なしの場合は、個体数全て初期値でモデルを生成する
20             self.models = [Network() for i in range(size)]
21         else:
22             # 前世代が与えられた場合は交叉(crossover),突然変異(mutate)を行い次世代を生成
23 する
24             self.old_models = old_population.models
25             self.old_fitnesses = old_population.fitnesses
26             self.models = []
27             self.crossover()
28             self.mutate()
29         # 各個体の評価値を保存する配列を初期化
30         self.fitnesses = np.zeros(self.size)
31
32     def crossover(self):
33         """ 交叉(crossover)
34         :return:
35         """
36         # 全ての個体の評価値の合計および各個体評価値を正規化
37         sum_fitnesses = np.sum(self.old_fitnesses)
38         probs = [self.old_fitnesses[i] / sum_fitnesses for i in
39                  range(self.size)]
40
41         sort_indices = np.argsort(probs)[::-1]
42         for i in range(self.size):
43             if i < self.size * elitism_pct:
44                 # 優秀な個体は(上位20%)はそのまま
45                 model_c = self.old_models[sort_indices[i]]
```

```
            else:
                # それ以外はランダムな2つの個体をかけ合わせる
                a, b = np.random.choice(self.size, size=2, p=probs,
                                        replace=False)
                prob_neuron_from_a = 0.5

                # モデルの各ウエイトを50/50の確率で交叉
                model_a, model_b = self.old_models[a], self.old_models[b]
                model_c = Network()

                for j in range(input_size):
                    if np.random.random() > prob_neuron_from_a:
                        model_c.output.weight.data[0][j] = ¥
                            model_b.output.weight.data[0][j]
                    else:
                        model_c.output.weight.data[0][j] = ¥
                            model_a.output.weight.data[0][j]

            self.models.append(model_c)

    def mutate(self):
        """ 突然変異(mutate)
        :return:
        """
        for model in self.models:
            for i in range(input_size):
                # 一定の確率(20%)で突然変異を行ったモデルとする
                if np.random.random() < mutation_prob:
                    # 一定のノイズをモデルに加える
                    with torch.no_grad():
                        noise = torch.randn(1).mul_(
                            weights_mutate_power).to(device)
                        model.output.weight.data[0][i].add_(noise[0])

```