卒業研究報告書

題目

トーラス型リバーシの AI 作成

指導教員

石水 隆　講師

報告者

19–1–037–0140

嶋田 鉄馬

近畿大学理工学部情報学科

令和 6 年 1 月 31 日提出

概要

リバーシは 8 × 8 の盤面で黒と白の石を交互に打ち，相手の石を自分の石で挟むことによって自分の石へと換えていき，最終的な盤上の石の個数を競う二人零和有限確定完全情報ゲームである．リバーシでは，四隅の角 (以下「角」と呼ぶ) に置かれている石は裏返すことができないため，角を取ることが重視される．そのため既存のリバーシ AI では角に高い評価値を設定するなどのように，マス目ごとに評価値を設定する手法を用いている．

トーラス型リバーシは通常のリバーシと違い，ゲームの盤面の左右，上下がそれぞれ接続されているものと考える．その性質上，トーラス型リバーシには角に相当するものが存在しないため，通常のリバーシと同様の手法を使うことができず，なおかつトーラス型リバーシがマイナーなゲームであるため，AI がほとんど存在していない．そこで本研究ではトーラス型リバーシの AI 開発を目指す．

目次

1　序論

1.1　本研究の背景

1.1.1　リバーシ

リバーシは 8 × 8 の盤面で黒と白の石を交互に打ち，相手の石を自分の石で挟むことによって自分の石へと換えていき，最終的な盤上の石の個数を競う二人零和有限確定完全情報ゲームである．日本ではオセロの名称でも親しまれており，現在のオセロ愛好者は全世界で 6 億人にものぼるほどの国際的なボードゲームである．[1] そのためリバーシの AI は，囲碁，将棋，チェスなどの AI と並び多くの研究と開発がなされてきた．そして，1997 年のオセロ AI”Logistello” と当時の世界チャンピオンの村上健氏との 6 回戦で村上氏が 6 連敗した時に，オセロでコンピュータが人間の能力を上回ったと見做されるようになった．[2] また，リバーシは理論的な解析も行われてきた．6x6 のミニリバーシでは，完全解析により双方最善手を打つと 16 対 20 で後手が勝つことが判明している [3]．その他のサイズのミニリバーシでも解析が行われている [4]．表 1 にミニリバーシの完全解析結果を示す．8x8 の通常のリバーシに関しては，2023 年 10 月に弱解析が行われ，初期局面より双方最善手を打つと引き分けとなるという論文が滝沢により公開された [5]．ただし現時点ではこの論文は追試および査読が行われていないため真偽は不明である．

リバーシには，通常 2 人で対戦するリバーシを 4 人版に拡張した Yonin や盤面が円形で角が存在しないニップ，盤面サイズの小さいミニリバーシ，10 × 10 の大きな盤を用いるグランドオセロ，盤面が八角形で角が 8 個あるエイトスターズオセロなど様々なヴァリエーションがある．図 1 に各ヴァリエーションの盤面を示す．本研究ではそれらの中の一つであるトーラス型リバーシを題材とする．

表 1　ミニリバーシの完全解析結果 [4]

初期配置	サイズ	勝敗	石数
○●	4x4	後手勝ち	黒 3 白 11
●○	4x6	先手勝ち	黒 20 白 4
	4x8	先手勝ち	黒 26 白 0
	4x10	先手勝ち	黒 39 白 0
	6x6	後手勝ち	黒 16 白 20
●○	4x4	後手勝ち	黒 6 白 9
●○	4x6	先手勝ち	黒 21 白 3
	4x8	先手勝ち	黒 28 白 0
	4x10	先手勝ち	黒 32 白 0
	6x6	後手勝ち	黒 17 白 19

1.1.2　トーラス型リバーシ

図 2 にトーラス型リバーシの盤面と初期配置を示す．トーラス型リバーシは通常のリバーシと違い，ゲームの盤面の左右、上下がそれぞれ接続されているものと考える．例えば，図 3 の局面では，a6 に黒石を打つことで△の白 2 石をひっくり返すことができる．その性質上，トーラス型リバーシには角に相当するものが存在しないため，通常のリバーシとは異なる戦略をとらなければならない．

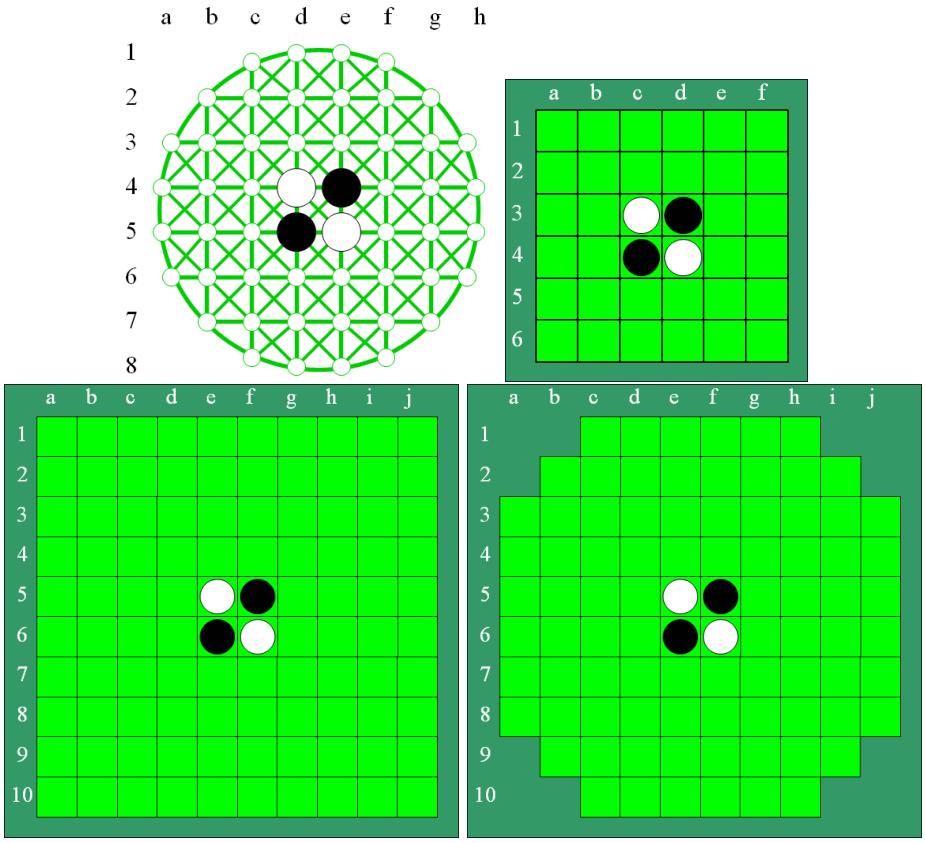

図 1　ニップ，ミニオセロ，グランドオセロ，エイトスターズオセロの盤面

1.2　本研究の目的

通常のリバーシにおける AI は強さと速さの両面においてまだ改良の余地があるものの，人間の能力をはるかに凌駕しており，人間相手に圧勝できるほど強くなっている．[2] しかしトーラス型リバーシは非常にマイナーなゲームであるため，通常のリバーシのような定石や戦術が存在しておらず，通常のリバーシの AI のように評価値を用いる手法も使うことができないため，トーラス型リバーシの AI はほとんど存在していない．そこで本研究では，強いトーラス型リバーシの AI を作成することを目的とする．

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 章で本研究で作成するトーラス型リバーシの AI のプログラムについて説明し，3 章で実行結果とそれを受けての考察を述べ，4 章で結論および今後の課題について述べる．

2　研究内容

本章では，本研究で作成するトーラス型リバーシの AI のプログラムについて述べる．付録に本研究で作成するトーラス型リバーシの AI のプログラムのソースを示す．

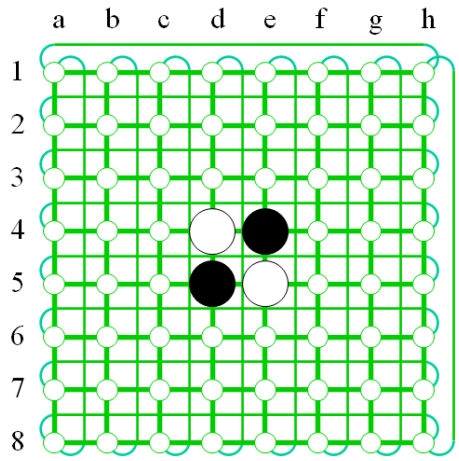

図 2　トーラス型リバーシの盤面と初期状態

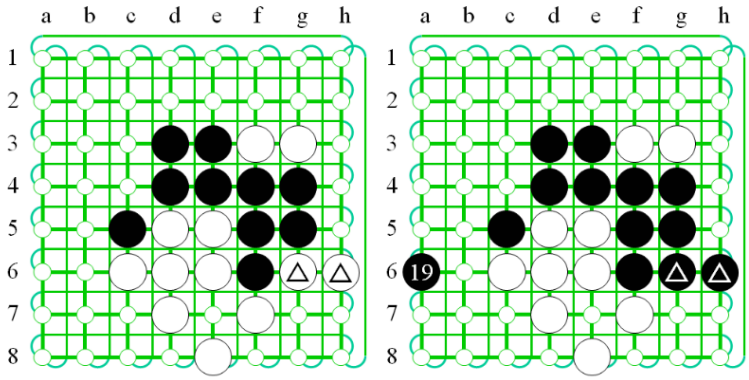

図 3　局面例

2.1　トーラス型リバーシのルール

本節では，トーラス型リバーシのルールについて述べる．トーラス型リバーシは通常のリバーシと同じく 8 × 8 の盤面を用いるが，盤の上下と左右は繋がっているとみなす．黒石が先手で，盤上の石が多い色が勝ちという基本的なルールは通常のリバーシと同じであるが，上下左右が繋がっているとみなされるため，角や辺に置かれた石であっても反対側のマスに石を置くことで裏返すことができる．この角や辺に置かれた石も反対側に石を置くことで裏返すことができるのがトーラス型リバーシの特徴で，その他のリバーシと違うルールとなっている．

2.2　トーラス型リバーシの AI の戦略

本節では，本研究で作成するトーラス型リバーシの AI が用いる戦略について述べる．

通常のリバーシでは，着手選択をするための手法は様々なものが提案されている．リバーシは古くからプレイされているゲームであるため，定石が確立している．リバーシの序盤定石には鼠定石，牛定石，兎定石等が

あり，定石をデータベースとして持つことで序盤は定石通りに打つことができる．また，通常のリバーシでは角に置いた石はゲーム終了までひっくり返されることがないため，角に石を置くことは重要である．通常のリバーシでは辺上のマスは角のマスから順に角マス，C マス，B マス，A マス，角から斜め方向に 1 マス内側の位置のマスは X マスと呼ばれる．図 4 に通常のリバーシの各マスの呼び名を示す．角に隣接する C マスと呼ばれるマスは，角が空いているときに C マスが自石だと相手に角を取られやすくなるため，C マスに打つのは危険とされる．一方角に自石が置かれている場合，C マスの自石もひっくり返されることが無い．このようなゲーム中に絶対にひっくり返されることが無い石を確定石と呼ぶ．リバーシではできるだけ確定石を増やすことが中盤の重要戦略とされる．一方，トーラス型リバーシでは，角マスに石を打っても確定石とならないため，角マスを取ることはさほど重要ではない．逆に，通常のリバーシでは危険とされる C マス，X マスへの石打ちは，トーラス型リバーシでは特に危険とは考えられない．

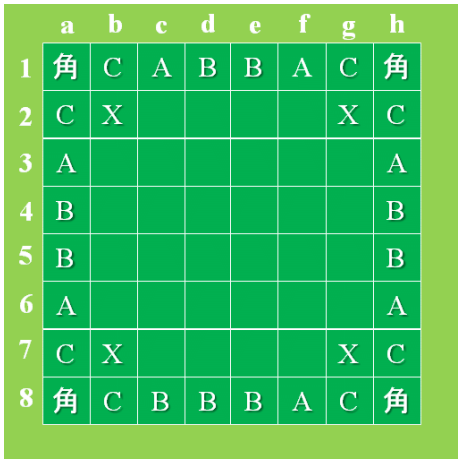

図 4　通常のリバーシの各マスの呼び名

また，既存のリバーシプログラムでは $\alpha\beta$ 法を用いたものや棋譜データを読み込ませて深層学習を行う方法が主流である．$\alpha\beta$ 法は，マスに設定した評価関数を基にゲーム木を探索していき，その時点での最善手を導き出す Mini-Max 法を改良し，不必要なノードの探索をカットすることで計算量を削減した手法である．しかし，トーラス型リバーシはその性質上評価関数の設定が難しく，マイナーなゲームであり棋譜も存在しないため，これらの手法を適用することはできない．

そのため，本研究で作成する AI の戦略はモンテカルロ法を用いている．モンテカルロ法とは，着手可能手に対し，その手から終局までをランダムに打ち，判定を行うものである．モンテカルロ法は最も勝率の高い手を選択することはできるが，その選択が必ずしも最良の選択となるとは限らない手法である．しかし，モンテカルロ法は対象のゲームに関する戦略を必要とせず，ランダムに手を打つだけで着手選択できるため，様々なゲームに応用できる手法である [6]．石倉の研究結果 [7] より，トーラス型リバーシはモンテカルロ法を用いて着手選択すると後手が有利となることが示されている．そこで本研究では，先手の勝率を伸ばすことを重視してトーラス型リバーシ AI を作成する．自分が先手の場合，図 5，6 のように自分が石を置いた時点で相手の石が縦，横，斜め，いずれかの直線状になるように石を置くことで，図 7 のように高確率で詰み状態を実現できる．しかし，上記の方法でも試合展開によっては図 8 のように相手の石が直線状にならなくなる，つまり詰み状態に導けなくなる場合があるため，基本は自分が石を置いた時点で相手の石が直線状になるように石を置き，相手の石が直線状にならなくなった場合はモンテカルロ法に切り替えるという戦略をとる．

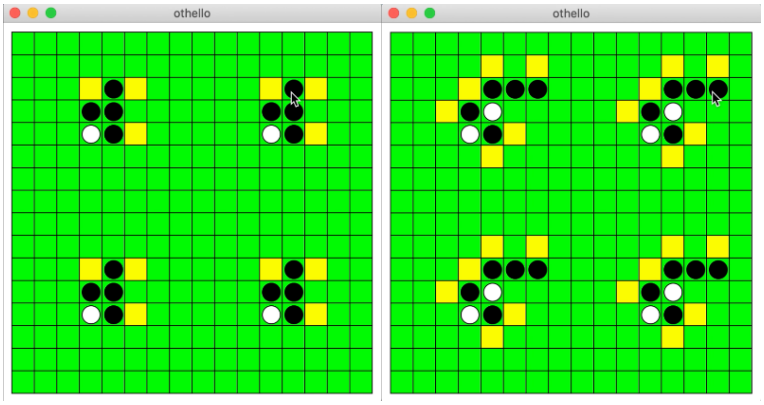

図 5　黒の一手目および二手目

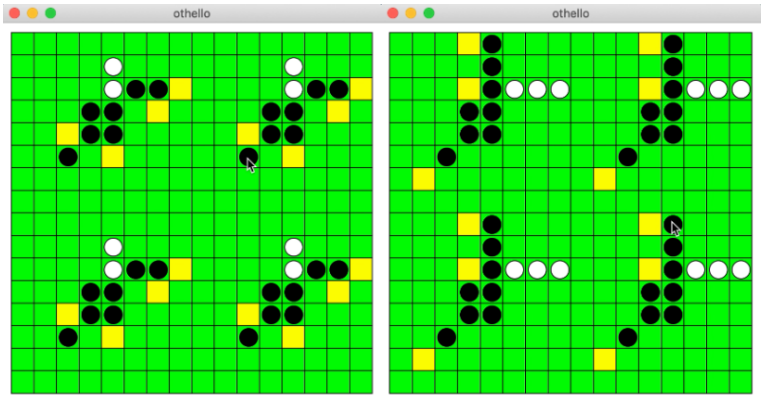

図 6　黒の三手目および四手目

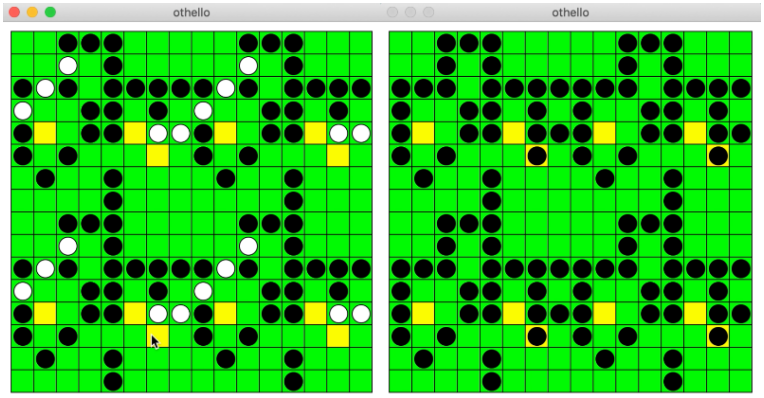

図 7　白の十一手目および黒の十二手目

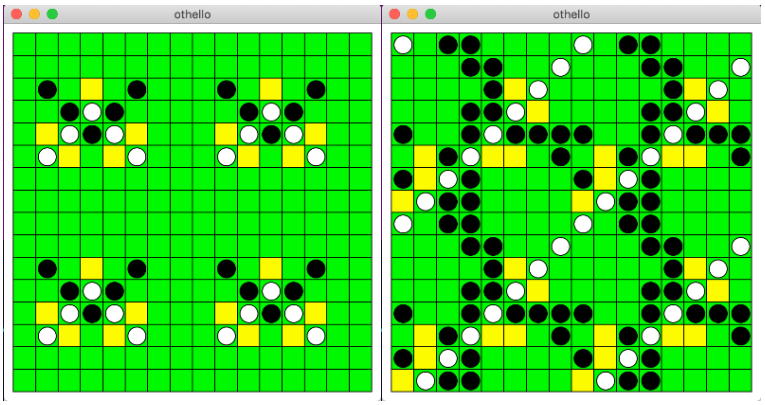

図 8　どのマスに黒石を置いても詰み状態に導けなくなる場合の例

2.3　トーラス型リバーシの AI のプログラム概要

本節では，本研究で作成するトーラス型リバーシの AI のプログラムの概要について述べる．通常の 8 × 8 の盤面では，石を置いた際に盤面の境目を超えて裏返る石がどの程度存在するのかを確認することが一見しただけでは分かりづらいため，本研究で作成したプログラムは，同じ盤面が 4 つ隣接する 64 × 64 の盤面となっている．また，石を置くことができるマスは黄色く描写される．

2.4　プログラムの構造

本節では，プログラムの構造について述べる．本研究では python を用いてプログラムを作成した．
以下に，本研究で作成したプログラムで用いているメソッドについて述べる．

1. 定数
   CANVAS_SIZE　キャンバスの横方向・縦方向のサイズ（px）．値は 400.
   NUM_SQUARE　横方向・縦方向のマスの数．値は 16.
   WIN　モンテカルロ法で使用する勝利数．値は 0.
   BOARD_COLOR　盤面の背景色．値は’green’.
   M_COM_COLOR　M_COM の石の色．先手，後手によって値が’black’，’white’ のどちらかになる．
   COM_COLOR　COM の石の色．先手，後手によって値が’black’，’white’ のどちらかになる．
   PLACABLE_COLOR　次に石を置ける場所を示す色．値は’yellow’.
   M_COM　トーラス型リバーシの AI プレイヤーを示す値．値は 1.
   COM　ランダムに石を置くプレイヤーを表す値．値は 2.
   fOrS　先手後手を格納する値．先手の場合の値は True.
2. コンストラクタ
   __init__(self, master)　ウィジェットの作成やオセロゲームの初期化を行う．
3. メソッド
   createWidgets(self)　ウィジェットを作成・配置する．

initOthello(self)　ゲームの初期化を行う．

drawDisk(self, x, y, color)　(x,y) に色が color の石を置く（円を描画する）．

getPlacable(self)　次に置くことができる石の位置を取得する．

checkPlacable(self, x, y)　(x,y) に石が置けるかどうかをチェックする．

showPlacable(self, placable)　placable に格納された次に石が置けるマスの色を変更する．

place(self, x, y, color)　(x,y) に色が color の石を置く．

reverseList(self, x, y)　(x,y) に石が置かれた時に裏返す必要のある石をリストに格納する．

reverse(self)　リストを基に (x,y) に石が置かれた時に裏返す必要のある石を裏返す．

nextPlayer(self)　次に石を置くプレイヤーを決める．

showResult(self)　ゲーム終了時の結果を表示する．

com(self)　COM に石を置かせる．

m_com(self)　モンテカルロ法を使う戦略．

countNone(self)　盤面リストの None の数を数える．

load(self)　モンテカルロ法用の盤面に現在の盤面を読み込む．

getPlacableM(self)　次に置くことができる石の位置を取得（モンテカルロ法専用）．

checkPlacableM(self, x, y)　(x,y) に石が置けるかどうかをチェック（モンテカルロ法専用）．

placeM(self, x, y, color)　(x,y) に色が color の石を置く（モンテカルロ法専用）．

reverseListM(self, x, y)　(x,y) に石が置かれた時に裏返す必要のある石をリストに格納する（モンテカルロ法専用）．

reverseM(self)　リストを基に (x,y) に石が置かれた時に裏返す必要のある石を裏返す（モンテカルロ法専用）．

memorizeColor(self, x, y, color)　石の色を管理リストに記憶させておく．

nextPlayerM(self)　次に石を置くプレイヤーを決める（モンテカルロ法専用）

endGame(self)　モンテカルロ法でのゲーム終了後勝った場合は勝ち点を足す．

com1(self)　モンテカルロ法で相手が石を置く．

com2(self)　モンテカルロ法で自分が石を置く．

straight(self)　一直線に石を置いていく直線戦略．

copy(self)　コピー盤面に現在の盤面を読み込む．

judgeStraight(self)　直線状になっているかを判断する．

placeC(self, x, y, color)　(x,y) に色が color の石を置く（直線戦略専用）．

reverseListC(self, x, y)　(x,y) に石が置かれた時に裏返す必要のある石をリストに格納する（直線戦略専用）．

reverseC(self)　リストを基に石が置かれた時に裏返す必要のある石を裏返す（直線戦略専用）．

memorizeColorC(self, x, y, color)　描画した円の色を管理リストに記憶させておく（直線戦略専用）．

これらの中で石を置ける場所の決定と石をひっくり返す処理とモンテカルロ法，相手の石が直線状になるように石を置く戦術それぞれの基盤となるメソッドについて述べる．

2.4.1　石を置ける場所の決定と石を裏返す処理の基盤となるメソッド

石を置ける場所の決定と石を裏返す処理の基盤となるのは主に checkPlacable(self, x, y)，reverseList(self, x, y) メソッドの 2 つである．checkPlacable(self, x, y) メソッドでは引数の x,y 座標に石を置いた際に，その石の周囲 8 方向をチェックしていくメソッドである．その際に，範囲外を参照しないようあらかじめ値を調節する（例えば x 座標が 0 で方向が-1 の場合，そのままでは範囲外の-1 を参照してしまうため x の値を 16 にすることで 16-1=15 となり，反対側の 15 の位置を参照するようになる）．そしてその方向を全て確認し，条件を満たしているかどうかで石を置くことができる場所を決定する．reverseList(self, x, y) メソッドは石を裏返す処理であり，盤面を第一象限，第二象限，第三象限，第四象限のそれぞれに分けて行っている．処理の内容は checkPlacable(self, x, y) メソッドと同じく置く石の周囲 8 方向をチェックしていき裏返すことができる石を裏返していく．

2.4.2　モンテカルロ法の基盤となるメソッド

モンテカルロ法の基盤となるのは主に m_com(self)，countNone(self)，load(self) メソッドの 3 つである．load(self) メソッドでその時点での盤面をモンテカルロ法専用の二次元リストに読み込ませ，m_com(self) メソッドで一定回数の試行を繰り返して最も勝率の高いマスを選ぶ．試行を繰り返す回数は countNone(self) メソッドで残りの空きマスの数を求め，その数によって決定される．今回は最序盤，序盤，中盤，終盤，最終盤 1，最終盤 2，最終盤 3 と 7 つの局面に分けており，序盤の方は試行回数を少なく設定することで探索時間を抑え，終盤は試行回数を多く設定することでより良い手を探すようにしている．

2.4.3　相手の石が直線状になるように石を置く戦術の基盤となるメソッド

相手の石が直線状になるように石を置く戦術の基盤となるのは主に straight(self)，copy(self)，judgeStraight(self) メソッドの 3 つである．copy(self) メソッドでその時点での盤面を本戦術専用の二次元リストに読み込ませ，straight(self) で相手の石が直線状になるマスを選ぶ．この際，詰み状態にできるマスがあればそちらを優先しておくようにしている．直線状の判定には judgeStraight(self) メソッドを使用し，このメソッドは基準となる石を一つ設定してその周囲 8 マスに一つ以上石が置かれている場合，その方向一列を一次元リストに保存していき，それらと盤面の白石の数を比較して直線かどうかを判断するものである．

3　実行結果および考察

本研究で作成した AI の有用性を確かめるため，ランダムに石を打つ AI を相手に先手のときと後手のときでそれぞれ 100 回ずつ対戦させた．表 2 はその結果を示したものである．なお完全勝利数は詰み状態で勝利した際の勝利数である．この結果より，先手の際に相手の石が直線状になるように石を置く戦術が有効であることが実証された．また先手の際の勝利数と完全勝利数の比率から，先手の際の勝利数は直線状になるように石を置く戦術に依存していないことも判明したため，モンテカルロ法と組み合わせた戦術が有効であることも実証された．先手の勝率が 100% なのに対し，後手の勝率が 100% ではない原因として考えられるのは，負けた際の対局で相手の石が直線状になるように石を置く戦術が比較的長くとられており，モンテカルロ法に移行したのが遅かったためと考えている．このため，先手が中盤までにモンテカルロ法を使うなどの対策を講じなければ，後手の方が有利であるということが考えられる．

表 2　対戦結果

	先手	後手	合計
勝利数（完全勝利数）	90（51）	100（0）	190（51）
敗北数	10	0	10
勝率 (%)	90	100	95

4　結論・今後の課題

本研究ではトーラス型リバーシ AI の作成を行った．通常のリバーシにおける既存の戦術が使えないため，独自の戦術を編み出し実装した結果，とても高い勝率を記録することに成功した．

しかし，本研究ではランダムに石を置く AI だけを相手にしているため，相手が戦術をとる場合に勝率を伸ばすことができるかというような検証も今後必要になる．また今回は一手先だけを見てモンテカルロ法と直線状になるように石を置く戦術を切り替えていたため，二手先三手先まで見通せるようになれば，さらに勝率を伸ばすことができるのではないかと考えている．

謝辞

本研究を行うにあたり，卒業研究発表概要や卒業研究報告書の推敲，資料の提供など終始多大なご指導を賜った，石水隆講師に深謝いたします．

参考文献

[1] 株式会社メガハウス：オセロってなに？ — オセロ公式サイト
https://www.megahouse.co.jp/othello/what/

[2] 山名　琢翔：深層学習による圧縮を利用した強力なオセロ AI の制作, 研究報告ゲーム情報学 (GI) Vol.2022-GI-48(5), 情報処理学会 (2022)

[3] Joel Feinstein： Amenor Wins World 6x6 Championships!, Forty billion noted under the tree, pp.6-8,British Othello Federation's newsletter. (1993) http://www.britishothello.org.uk/fbnall.pdf

[4] 竹下拓輝，池田論, 坂本眞人, 伊藤隆夫：縮小盤オセロにおける完全解析, 情報処理学会九州支部火の国情報シンポジウム, No.1A-2, pp.1-6 (2015) https://www.ipsj-kyushu.jp/page/ronbun/hinokuni/1004/1A/1A-2.pdf

[5] Hiroshi Takizawa： Othello is Solved (2023) https://arxiv.org/abs/2310.19387

[6] 美添一樹，山下宏，松原仁 編：コンピュータ囲碁 —モンテカルロ法の理論と実践—，共立出版 (2012)

[7] 石倉　慎：トーラス型リバーシの AI 作成, 近畿大学理工学部情報学科 2022 年度卒業報告 (2023).

付録 A　ソースコード

本研究で作成した主なプログラムのソースコードを以下に示す.

付録 1　ソースコード

```
# -*- coding:utf-8 -*-
import tkinter
import tkinter.messagebox
import random

'''先手後手で戦術を変えるクラス'''

# キャンバスの横方向・縦方向のサイズ（ px ）
CANVAS_SIZE = 400

# 横方向・縦方向のマスの数
NUM_SQUARE = 16

# モンテカルロ法で使用する勝利数
WIN = 0

# 色の設定
BOARD_COLOR = 'green' # 盤面の背景色
M_COM_COLOR = 'black' # M_COM の石の色
COM_COLOR = 'white' # COM の石の色
PLACABLE_COLOR = 'yellow' # 次に石を置ける場所を示す色

# プレイヤーを示す値
M_COM = 1
COM = 2

# 先手後手を格納する値
fOrS = True

class Othello():
    def __init__(self, master):
        '''コンストラクタ'''

```

```
34          self.master = master # 親ウィジェット
35          self.player = COM # 次に置く石の色
36          self.board = None # 盤面上の石を管理する 2 次元リスト
37          self.mBoard = None # モンテカルロ法を用いる際の盤面上の石を管理する 2 次元リ
                ストト
38          self.boardCopy = None # 盤面をコピーする 2 次元リスト
39          self.revBoard = None # 裏返す石を管理する 2 次元リスト
40          self.straightBoard = None # 一直線判定用の 1 次元リスト
41          self.color = { # 石の色を保持する辞書
42              M_COM : M_COM_COLOR,
43              COM : COM_COLOR
44          }
45
46          # ウィジェットの作成
47          self.createWidgets()
48
49          # オセロゲームの初期化
50          self.initOthello()
51
52      def createWidgets(self):
53          '''ウィジェットを作成・配置する'''
54
55          # キャンバスの作成
56          self.canvas = tkinter.Canvas(
57              self.master,
58              bg=BOARD_COLOR,
59              width=CANVAS_SIZE+1, # +1 は枠線描画のため
60              height=CANVAS_SIZE+1, # +1 は枠線描画のため
61              highlightthickness=0
62          )
63          self.canvas.pack(padx=10, pady=10)
64
65      def initOthello(self):
66          '''ゲームの初期化を行う'''
67
68          # 盤面上の石を管理する  2 次元リストを作成（最初は全て None ）
69          self.board = [[None] * NUM_SQUARE for i in range(NUM_SQUARE)]
70
71          # 裏返す石を管理する 2 次元リストを作成（最初は全て None ）
72          self.revBoard = [[None] * NUM_SQUARE for i in range(NUM_SQUARE)
```

```
            ]

        # 1 マスのサイズ（ px ）を計算
        self.square_size = CANVAS_SIZE // NUM_SQUARE

        # マスを描画
        for y in range(NUM_SQUARE):
            for x in range(NUM_SQUARE):
                # 長方形の開始・終了座標を計算
                xs = x * self.square_size
                ys = y * self.square_size
                xe = (x + 1) * self.square_size
                ye = (y + 1) * self.square_size

                # 長方形を描画
                tag_name = 'square_' + str(x) + '_' + str(y)
                self.canvas.create_rectangle(
                    xs, ys,
                    xe, ye,
                    tag=tag_name
                )

        # あなたの石の描画位置を計算

        # 第一象限
        mCom_init_pos1_1_x = NUM_SQUARE // 4 * 3
        mCom_init_pos1_1_y = NUM_SQUARE // 4
        mCom_init_pos1_2_x = NUM_SQUARE // 4 * 3 - 1
        mCom_init_pos1_2_y = NUM_SQUARE // 4 - 1

        # 第二象限
        mCom_init_pos2_1_x = NUM_SQUARE // 4
        mCom_init_pos2_1_y = NUM_SQUARE // 4
        mCom_init_pos2_2_x = NUM_SQUARE // 4 - 1
        mCom_init_pos2_2_y = NUM_SQUARE // 4 - 1

        # 第三象限
        mCom_init_pos3_1_x = NUM_SQUARE // 4
        mCom_init_pos3_1_y = NUM_SQUARE // 4 * 3
        mCom_init_pos3_2_x = NUM_SQUARE // 4 - 1
```

```
112         mCom_init_pos3_2_y = NUM_SQUARE // 4 * 3 - 1
113
114         # 第四象限
115         mCom_init_pos4_1_x = NUM_SQUARE // 4 * 3
116         mCom_init_pos4_1_y = NUM_SQUARE // 4 * 3
117         mCom_init_pos4_2_x = NUM_SQUARE // 4 * 3 - 1
118         mCom_init_pos4_2_y = NUM_SQUARE // 4 * 3 - 1
119
120
121         # 第一象限
122         mCom_init_pos1 = (
123             (mCom_init_pos1_1_x, mCom_init_pos1_1_y),
124             (mCom_init_pos1_2_x, mCom_init_pos1_2_y)
125         )
126
127         # 第二象限
128         mCom_init_pos2 = (
129             (mCom_init_pos2_1_x, mCom_init_pos2_1_y),
130             (mCom_init_pos2_2_x, mCom_init_pos2_2_y)
131         )
132
133         # 第三象限
134         mCom_init_pos3 = (
135             (mCom_init_pos3_1_x, mCom_init_pos3_1_y),
136             (mCom_init_pos3_2_x, mCom_init_pos3_2_y)
137         )
138
139         # 第四象限
140         mCom_init_pos4 = (
141             (mCom_init_pos4_1_x, mCom_init_pos4_1_y),
142             (mCom_init_pos4_2_x, mCom_init_pos4_2_y)
143         )
144
145         # 計算した描画位置に石（円）を描画
146
147         # 第一象限
148         for x, y in mCom_init_pos1:
149             self.drawDisk(x, y, self.color[M_COM])
150
151         # 第二象限
```

```
152             for x, y in mCom_init_pos2:
153                 self.drawDisk(x, y, self.color[M_COM])
154 
155             # 第三象限
156             for x, y in mCom_init_pos3:
157                 self.drawDisk(x, y, self.color[M_COM])
158 
159             # 第四象限
160             for x, y in mCom_init_pos4:
161                 self.drawDisk(x, y, self.color[M_COM])
162 
163             # 対戦相手の石の描画位置を計算
164 
165             # 第一象限
166             com_init_pos1_1_x = NUM_SQUARE // 4 * 3 - 1
167             com_init_pos1_1_y = NUM_SQUARE // 4
168             com_init_pos1_2_x = NUM_SQUARE // 4 * 3
169             com_init_pos1_2_y = NUM_SQUARE // 4 - 1
170 
171             # 第二象限
172             com_init_pos2_1_x = NUM_SQUARE // 4 - 1
173             com_init_pos2_1_y = NUM_SQUARE // 4
174             com_init_pos2_2_x = NUM_SQUARE // 4
175             com_init_pos2_2_y = NUM_SQUARE // 4 - 1
176 
177             # 第三象限
178             com_init_pos3_1_x = NUM_SQUARE // 4 - 1
179             com_init_pos3_1_y = NUM_SQUARE // 4 * 3
180             com_init_pos3_2_x = NUM_SQUARE // 4
181             com_init_pos3_2_y = NUM_SQUARE // 4 * 3 - 1
182 
183             # 第四象限
184             com_init_pos4_1_x = NUM_SQUARE // 4 * 3 - 1
185             com_init_pos4_1_y = NUM_SQUARE // 4 * 3
186             com_init_pos4_2_x = NUM_SQUARE // 4 * 3
187             com_init_pos4_2_y = NUM_SQUARE // 4 * 3 - 1
188 
189             # 第一象限
190             com_init_pos1 = (
191                 (com_init_pos1_1_x, com_init_pos1_1_y),
```

```
192             (com_init_pos1_2_x, com_init_pos1_2_y)
193         )
194
195         # 第二象限
196         com_init_pos2 = (
197             (com_init_pos2_1_x, com_init_pos2_1_y),
198             (com_init_pos2_2_x, com_init_pos2_2_y)
199         )
200
201         # 第三象限
202         com_init_pos3 = (
203             (com_init_pos3_1_x, com_init_pos3_1_y),
204             (com_init_pos3_2_x, com_init_pos3_2_y)
205         )
206
207         # 第四象限
208         com_init_pos4 = (
209             (com_init_pos4_1_x, com_init_pos4_1_y),
210             (com_init_pos4_2_x, com_init_pos4_2_y)
211         )
212
213         # 計算した描画位置に石（円）を描画
214
215         # 第一象限
216         for x, y in com_init_pos1:
217             self.drawDisk(x, y, self.color[COM])
218
219         # 第二象限
220         for x, y in com_init_pos2:
221             self.drawDisk(x, y, self.color[COM])
222
223         # 第三象限
224         for x, y in com_init_pos3:
225             self.drawDisk(x, y, self.color[COM])
226
227         # 第四象限
228         for x, y in com_init_pos4:
229             self.drawDisk(x, y, self.color[COM])
230
231         # 最初に置くことができる石の位置を取得
```

```
232             placable = self.getPlacable()
233
234             # その位置を盤面に表示
235             self.showPlacable(placable)
236
237             if self.player == M_COM:
238                 # 次のプレイヤーが M_COM の場合は 1 秒後に M_COM に石を置く場所を決めさ
                        せる
239                 self.master.after(1000, self.straight)
240             else:
241                 global fOrS
242                 fOrS = False
243                 # 次のプレイヤーが COM の場合は 1 秒後に COM に石を置く場所を決めさせる
244                 self.master.after(1000, self.com)
245
246         def drawDisk(self, x, y, color):
247             '''(x,y)に色が color の石を置く（円を描画する）'''
248
249             # (x,y)のマスの中心座標を計算
250             center_x = (x + 0.5) * self.square_size
251             center_y = (y + 0.5) * self.square_size
252
253             # 中心座標から円の開始座標と終了座標を計算
254             xs = center_x - (self.square_size * 0.8) // 2
255             ys = center_y - (self.square_size * 0.8) // 2
256             xe = center_x + (self.square_size * 0.8) // 2
257             ye = center_y + (self.square_size * 0.8) // 2
258
259             # 円を描画する
260             tag_name = 'disk_' + str(x) + '_' + str(y)
261             self.canvas.create_oval(
262                 xs, ys,
263                 xe, ye,
264                 fill=color,
265                 tag=tag_name
266             )
267
268             # 描画した円の色を管理リストに記憶させておく
269             self.board[y][x] = color
270
```

```
271     def getPlacable(self):
272         '''次に置くことができる石の位置を取得'''
273 
274         placable = []
275 
276         for y in range(NUM_SQUARE):
277             for x in range(NUM_SQUARE):
278                 # (x,y) の位置のマスに石が置けるかどうかをチェック
279                 if self.checkPlacable(x, y):
280                     # 置けるならその座標をリストに追加
281                     placable.append((x, y))
282 
283         return placable
284 
285     def checkPlacable(self, x, y):
286         '''(x,y)に石が置けるかどうかをチェック'''
287 
288         # その場所に石が置かれていれば置けない
289         if self.board[y][x] != None:
290             return False
291 
292         if self.player == M_COM:
293             other = COM
294         else:
295             other = M_COM
296 
297         # (x,y)座標から縦横斜め全方向に対して相手の石が裏返せるかどうかを確認
298         for j in range(-1, 2):
299             for i in range(-1, 2):
300 
301                 # 真ん中方向はチェックしてもしょうがないので次の方向の確認に移る
302                 if i == 0 and j == 0:
303                     continue
304 
305                 originX = x # 本来の x 座標を保持
306                 originY = y # 本来の y 座標を保持
307 
308                 # x + i が 0 を下回る場合は、
309                 # x の値を NUM_SQUARE + 1 にすることで範囲外を参照しないようにする
                    (操作 1 )
```

```
310                 # 例えば x = 0 、i = -1 の場合は x の値を 16 にすることで
                        x + i が 15 になる
311                 if x + i < 0:
312                     x = NUM_SQUARE
313
314                 # x + i が NUM_SQUARE を上回る場合は、
315                 # x の値を -1 にすることで範囲外を参照しないようにする（操作 1 ）
316                 # 例えば x = 15 、i = 1 の場合は x の値を -1 にすることで
                        x + i が 0 になる
317                 if x + i >= NUM_SQUARE:
318                     x = -1
319
320                 # x の場合と同様（操作 1 ）
321                 if y + j < 0:
322                     y = NUM_SQUARE;
323
324                 # y の場合と同様（操作 1 ）
325                 if y + j >= NUM_SQUARE:
326                     y = -1
327
328                 # 隣が相手の色でなければその方向に石を置いても裏返せない
329                 if self.board[y + j][x + i] != self.color[other]:
330                     # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
331                     if originX != x:
332                         x = originX
333
334                     if originY != y:
335                         y = originY
336                     continue
337
338                 # 置こうとしているマスから遠い方向へ 1 マスずつ確認
339                 for s in range(2, NUM_SQUARE):
340
341                     # x + i * s が 0 を下回る場合は、
342                     # x の値を NUM_SQUARE + s にすることで範囲外を参照しないように
                            する（操作 2 ）
343                     # 例えば x = 5 、i = -1 、s = 7 の場合は、
344                     # x の値を 16 + 5 で 21 にすることで x + i * s が 14 になる
345                     if x + i * s < 0:
346                         x = NUM_SQUARE + x
347
```

```
348                     # x + i * s が NUM_SQUARE を上回る場合は、
349                     # x + i * s の値をから NUM_SQUARE を引いて 1 を足すことで範囲
                            外を参照しないようにする（操作 2 ）
350                     # 例えば x = 10 、 i = 1 、 s = 7 の場合は、
351                     # x の値を 17 - 16 - 7 にすることで x + i * s が 1 になる
352                     if x + i * s >= NUM_SQUARE:
353                         x = x + i * s - NUM_SQUARE - (i * s)
354
355                     # x の場合と同様（操作 2 ）
356                     if y + j * s < 0:
357                         y = NUM_SQUARE + y
358
359                     # y の場合と同様（操作 2 ）
360                     if y + j * s >= NUM_SQUARE:
361                         y = y + i * s - NUM_SQUARE - (i * s)
362
363                     if self.board[y + j * s][x + i * s] == None:
364                         # 自分の石が見つかる前に空きがある場合
365                         # この方向の石は裏返せないので次の方向をチェック
366                         # 操作 1 もしくは操作 2 を行った場合、 x の値と y の値を本来
                                の値に戻す
367                         if originX != x:
368                             x = originX
369
370                         if originY != y:
371                             y = originY
372                         break
373
374                     # その方向に自分の色の石があれば石が裏返せる
375                     if self.board[y + j * s][x + i * s] == self.color[
                            self.player]:
376                         # 操作 1 もしくは操作 2 を行った場合、 x の値と y の値を本来
                                の値に戻す
377                         if originX != x:
378                             x = originX
379
380                         if originY != y:
381                             y = originY
382                         return True
383
384                 # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
```

```
385                 if originX != x:
386                     x = originX
387 
388                 if originY != y:
389                     y = originY
390 
391                 # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
392                 if originX != x:
393                     x = originX
394 
395                 if originY != y:
396                     y = originY
397 
398         # 裏返せる石がなかったので(x,y)に石は置けない
399         return False
400 
401     def showPlacable(self, placable):
402         ''' placable に格納された次に石が置けるマスの色を変更する '''
403 
404         for y in range(NUM_SQUARE):
405             for x in range(NUM_SQUARE):
406 
407                 # fill を変更して石が置けるマスの色を変更
408                 tag_name = 'square_' + str(x) + '_' + str(y)
409                 if (x, y) in placable:
410                     self.canvas.itemconfig(
411                         tag_name,
412                         fill=PLACABLE_COLOR
413                     )
414 
415                 else:
416                     self.canvas.itemconfig(
417                         tag_name,
418                         fill=BOARD_COLOR
419                     )
420 
421     def place(self, x, y, color):
422         '''(x,y)に色が color の石を置く'''
423 
424         if x >= NUM_SQUARE // 2 and y < NUM_SQUARE // 2: # 第一象限を選択し
```

```
            た場合
425
426             # (x,y)に石が置かれた時に裏返る石を裏返す
427             self.reverseList(x, y) # 第一象限で裏返る石のリストを作る
428             self.reverseList(x - 8, y) # 第二象限で裏返る石のリストを作る
429             self.reverseList(x - 8, y + 8) # 第三象限で裏返る石のリストを作る
430             self.reverseList(x, y + 8) # 第四象限で裏返る石のリストを作る
431             self.reverse() # リストをもとに裏返す
432
433             # (x,y)に石を置く（円を描画する）
434             self.drawDisk(x, y, color) # 第一象限
435             self.drawDisk(x - 8, y, color) # 第二象限
436             self.drawDisk(x - 8, y + 8, color) # 第三象限
437             self.drawDisk(x, y + 8, color) # 第四象限
438
439         elif x < NUM_SQUARE // 2 and y < NUM_SQUARE // 2: # 第二象限を選択し
            た場合
440
441             # (x,y)に石が置かれた時に裏返る石を裏返す
442             self.reverseList(x + 8, y) # 第一象限で裏返る石のリストを作る
443             self.reverseList(x, y) # 第二象限で裏返る石のリストを作る
444             self.reverseList(x, y + 8) # 第三象限で裏返る石のリストを作る
445             self.reverseList(x + 8, y + 8) # 第四象限で裏返る石のリストを作る
446             self.reverse() # リストをもとに裏返す
447
448             # (x,y)に石を置く（円を描画する）
449             self.drawDisk(x + 8, y, color) # 第一象限
450             self.drawDisk(x, y, color) # 第二象限
451             self.drawDisk(x, y + 8, color) # 第三象限
452             self.drawDisk(x + 8, y + 8, color) # 第四象限
453
454         elif x < NUM_SQUARE // 2 and y >= NUM_SQUARE // 2: # 第三象限を選択
            した場合
455
456             # (x,y)に石が置かれた時に裏返る石を裏返す
457             self.reverseList(x + 8, y - 8) # 第一象限で裏返る石のリストを作る
458             self.reverseList(x, y - 8) # 第二象限で裏返る石のリストを作る
459             self.reverseList(x, y) # 第三象限で裏返る石のリストを作る
460             self.reverseList(x + 8, y) # 第四象限で裏返る石のリストを作る
461             self.reverse() # リストをもとに裏返す
462
```

```
463                 # (x,y)に石を置く（円を描画する）
464                 self.drawDisk(x + 8, y - 8, color) # 第一象限
465                 self.drawDisk(x, y - 8, color) # 第二象限
466                 self.drawDisk(x, y, color) # 第三象限
467                 self.drawDisk(x + 8, y, color) # 第四象限
468
469             else: # 第四象限を選択した場合
470
471                 # (x,y)に石が置かれた時に裏返る石を裏返す
472                 self.reverseList(x, y - 8) # 第一象限で裏返る石のリストを作る
473                 self.reverseList(x - 8, y - 8) # 第二象限で裏返る石のリストを作る
474                 self.reverseList(x - 8, y) # 第三象限で裏返る石のリストを作る
475                 self.reverseList(x, y) # 第四象限で裏返る石のリストを作る
476                 self.reverse() # リストをもとに裏返す
477
478                 # (x,y)に石を置く（円を描画する）
479                 self.drawDisk(x, y - 8, color) # 第一象限
480                 self.drawDisk(x - 8, y - 8, color) # 第二象限
481                 self.drawDisk(x - 8, y, color) # 第三象限
482                 self.drawDisk(x, y, color) # 第四象限
483
484             # 次に石を置くプレイヤーを決める
485             before_player = self.player
486             self.nextPlayer()
487
488             # 裏返す石を管理する 2 次元リストを初期化
489             self.revBoard = [[None] * NUM_SQUARE for i in range(NUM_SQUARE)
                    ]
490
491             if before_player == self.player:
492                 # 前と同じプレイヤーであればスキップされたことになるのでそれを表示
493                 if self.player != M_COM:
494                     tkinter.messagebox.showinfo('結果', ' M_COM のターンをスキッ
                            プしました')
495                 else:
496                     tkinter.messagebox.showinfo('結果', ' COM のターンをスキップし
                            ました')
497
498             elif not self.player:
499                 # 次に石が置けるプレイヤーがいない場合はゲーム終了
```

```
500             self.showResult()
501             return
502 
503         # 次に石がおける位置を取得して表示
504         placable = self.getPlacable()
505         self.showPlacable(placable)
506 
507         if self.player == M_COM:
508             if fOrS:
509                 # 次のプレイヤーが M_COM で先手の場合は 1 秒後に M_COM に直線戦略で石を置く場所を決めさせる
510                 self.master.after(1000, self.straight)
511             else:
512                 # 次のプレイヤーが M_COM の後手の場合は 1 秒後に M_COM にモンテカルロ法で石を置く場所を決めさせる
513                 self.master.after(1000, self.m_com)
514         else:
515             # 次のプレイヤーが COM の場合は 1 秒後に COM に石を置く場所を決めさせる
516             self.master.after(1000, self.com)
517 
518     def reverseList(self, x, y):
519         '''(x,y)に石が置かれた時に裏返す必要のある石をリストに格納する'''
520 
521         if self.board[y][x] != None:
522             # (x,y)にすでに石が置かれている場合は何もしない
523             return
524 
525         if self.player == COM:
526             other = M_COM
527         else:
528             other = COM
529 
530         for j in range(-1, 2):
531             for i in range(-1, 2):
532                 # 真ん中方向はチェックしてもしょうがないので次の方向の確認に移る
533                 if i == 0 and j == 0:
534                     continue
535 
536                 originX = x # 本来の座標を保持x
537                 originY = y # 本来の座標を保持y
```

```
538
539
540                 # x + i が 0 を下回る場合は、
541                 # x の値を NUM_SQUARE + 1 にすることで範囲外を参照しないようにする
                        （操作 1 ）
542                 # 例えば x = 0 、i = -1 の場合は x の値を 16 にすることで
                        x + i が 15 になる
543                 if x + i < 0:
544                     x = NUM_SQUARE
545
546                 # x + i が NUM_SQUARE を上回る場合は、
547                 # x の値を -1 にすることで範囲外を参照しないようにする（操作 1 ）
548                 # 例えば x = 15 、i = 1 の場合は x の値を -1 にすることで
                        x + i が 0 になる
549                 if x + i >= NUM_SQUARE:
550                     x = -1
551
552                 # x の場合と同様（操作 1 ）
553                 if y + j < 0:
554                     y = NUM_SQUARE
555
556                 # y の場合と同様（操作 1 ）
557                 if y + j >= NUM_SQUARE:
558                     y = -1
559
560                 # 隣が相手の色でなければその方向で裏返せる石はない
561                 if self.board[y + j][x + i] != self.color[other]:
562                     # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
563                     if originX != x:
564                         x = originX
565
566                     if originY != y:
567                         y = originY
568                     continue
569
570                 # 置こうとしているマスから遠い方向へ 1 マスずつ確認
571                 for s in range(2, NUM_SQUARE):
572
573                     # x + i * s が 0 を下回る場合は、
574                     # x の値を NUM_SQUARE + s にすることで範囲外を参照しないように
                            する（操作 2 ）
```

```
575                 # 例えば x = 5 、i = -1 、s = 7 の場合は x の値を 16 + 5 で
                        21 にすることで x + i * s が 14 になる
576                 if x + i * s < 0:
577                     x = NUM_SQUARE + x
578
579                 # x + i * s が NUM_SQUARE を上回る場合は、
580                 # x + i * s の値をから NUM_SQUARE を引いて 1 を足すことで範囲
                        外を参照しないようにする（操作 2 ）
581                 # 例えば x = 10 、 i = 1 、 s = 7 の場合は x の値を
                        17 - 16 - 7 にすることで x + i * s が 1 になる
582                 if x + i * s >= NUM_SQUARE:
583                     x = x + i * s - NUM_SQUARE - (i * s)
584
585                 # x の場合と同様（操作 2 ）
586                 if y + j * s < 0:
587                     y = NUM_SQUARE + y
588
589                 # y の場合と同様（操作 2 ）
590                 if y + j * s >= NUM_SQUARE:
591                     y = y + i * s - NUM_SQUARE - (i * s)
592
593                 if self.board[y + j * s][x + i * s] == None:
594                     # 自分の石が見つかる前に空きがある場合
595                     # この方向の石は裏返せないので次の方向をチェック
596                     # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
597                     if originX != x:
598                         x = originX
599
600                     if originY != y:
601                         y = originY
602                     break
603
604                 # その方向に自分の色の石があれば石が裏返せる
605                 if self.board[y + j * s][x + i * s] == self.color[
                        self.player]:
606
607                     for n in range(1, s):
608
609                         # 盤面の石の管理リストを石を裏返した状態に更新
610                         self.revBoard[(y + j * n) % NUM_SQUARE][(x
                                + i * n) % NUM_SQUARE] = self.color[self
```

```
                                .player]
611
612                         # 操作 1 もしくは操作 2 を行った場合、 x の値と y の値を本来
                                の値に戻す
613                         if originX != x:
614                             x = originX
615
616                         if originY != y:
617                             y = originY
618
619                         break
620
621                 # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
622                 if originX != x:
623                     x = originX
624
625                 if originY != y:
626                     y = originY
627
628                 # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
629                 if originX != x:
630                     x = originX
631
632                 if originY != y:
633                     y = originY
634
635     def reverse(self):
636         '''リストを基に(x,y)に石が置かれた時に裏返す必要のある石を裏返す'''
637
638         for j in range(0, NUM_SQUARE):
639             for i in range(0, NUM_SQUARE):
640                 if self.revBoard[j][i] != None:
641                     # 盤面の石の管理リストを石を裏返した状態に更新
642                     self.board[j][i] = self.revBoard[j][i]
643
644                     # 石の色を変更することで石の裏返しを実現
645                     tag_name = 'disk_' + str(i) + '_' + str(j)
646                     self.canvas.itemconfig(
647                         tag_name,
648                         fill=self.color[self.player]
```

```
649                     )
650
651
652     def nextPlayer(self):
653         '''次に石を置くプレイヤーを決める'''
654
655         before_player = self.player
656
657         # 石を置くプレイヤーを切り替える
658         if self.player == M_COM:
659             self.player = COM
660         else:
661             self.player = M_COM
662
663         # 切り替え後のプレイヤーが石を置けるかどうかを確認
664         placable = self.getPlacable()
665
666         if len(placable) == 0:
667             # 石が置けないのであればスキップ
668             self.player = before_player
669
670             # スキップ後のプレイヤーが石を置けるかどうかを確認
671             placable = self.getPlacable()
672
673             if len(placable) == 0:
674                 # それでも置けないのであれば両者とも石を置けないということ
675                 self.player = None
676
677     def showResult(self):
678         '''ゲーム終了時の結果を表示する'''
679
680         # それぞれの色の石の数を数える
681         num_mcom = 0
682         num_com = 0
683
684         for y in range(NUM_SQUARE):
685             for x in range(NUM_SQUARE):
686                 if self.board[y][x] == M_COM_COLOR:
687                     num_mcom += 1
688                 elif self.board[y][x] == COM_COLOR:
```

```
689                     num_com += 1
690
691         num_mcom /= 4
692         num_com /= 4
693
694         # 結果をメッセージボックスで表示する
695         tkinter.messagebox.showinfo('結果', 'M_COM' + str(num_mcom) + ':
                COM' + str(num_com))
696
697     def com(self):
698         ''' COM に石を置かせる '''
699
700         # 石が置けるマスを取得
701         placable = self.getPlacable()
702
703         # 置くマスをランダムに選ぶ
704         x, y = random.choice(placable)
705
706         # 石を置く
707         self.place(x, y, COM_COLOR)
708
709
710     def m_com(self):
711         ''' M_COM に石を置かせる '''
712
713         # 石が置けるマスを取得
714         placable = self.getPlacable()
715
716         mostWin = -1 # 最高勝利数
717         mostWinX = -1 # 最高勝利数の時の X 座標
718         mostWinY = -1 # 最高勝利数の時の Y 座標
719
720         for j in range(8):
721             for i in range(8):
722                 # 座標に石を置くことができる場合
723                 if self.checkPlacable(i,j) == True:
724
725                     # 盤面リストの None の数が 176 以上の時（最序盤）
726                     if self.countNone() > 176:
727
```

```
728                         # 試行回数は 5 回
729                         for x in range(5):
730                             # モンテカルロ法用の盤面を読み込み
731                             self.load()
732 
733                             self.placeM(i,j,M_COM_COLOR)
734 
735                     # 盤面リストの None の数が 136 以上 176 未満の時（序盤）
736                     elif self.countNone() <= 176 and self.countNone()
                            >= 136:
737 
738                         # 試行回数は 8 回
739                         for x in range(8):
740                             # モンテカルロ法用の盤面を読み込み
741                             self.load()
742 
743                             self.placeM(i,j,M_COM_COLOR)
744 
745                     # 盤面リストの None の数が 96 以上 136 未満の時（中盤）
746                     elif self.countNone() < 136 and self.countNone() >=
                            96:
747 
748                         # 試行回数は 10 回
749                         for x in range(10):
750                             # モンテカルロ法用の盤面を読み込み
751                             self.load()
752 
753                             self.placeM(i,j,M_COM_COLOR)
754 
755                     # 盤面リストの None の数が 50 以上 96 未満の時（終盤）
756                     elif self.countNone() < 96 and self.countNone() >=
                            50:
757 
758                         # 試行回数は 20 回
759                         for x in range(20):
760                             # モンテカルロ法用の盤面を読み込み
761                             self.load()
762 
763                             self.placeM(i,j,M_COM_COLOR)
764 
```

```
765                     # 盤面リストの None の数が 32 以上 50 未満の時（最終盤 1 ）
766                     elif self.countNone() < 50 and self.countNone() >=
                            32:
767
768                         # 試行回数は 75 回
769                         for x in range(75):
770                             # モンテカルロ法用の盤面を読み込み
771                             self.load()
772
773                             self.placeM(i,j,M_COM_COLOR)
774
775                     # 盤面リストの None の数が 16 以上 32 未満の時（最終盤 2 ）
776                     elif self.countNone() < 32 and self.countNone() >=
                            16:
777
778                         # 試行回数は 150 回
779                         for x in range(150):
780                             # モンテカルロ法用の盤面を読み込み
781                             self.load()
782
783                             self.placeM(i,j,M_COM_COLOR)
784
785                     # 盤面リストの None の数が 24 未満の時（最終盤 3 ）
786                     elif self.countNone() < 24:
787
788                         # 試行回数は 300 回
789                         for x in range(300):
790                             # モンテカルロ法用の盤面を読み込み
791                             self.load()
792
793                             self.placeM(i,j,M_COM_COLOR)
794
795                     global WIN
796
797                     # 勝利数が直前の最高勝利数より多い場合、座標と共に更新
798                     if WIN >= mostWin:
799                         mostWin = WIN
800                         mostWinX = i
801                         mostWinY = j
802
```

```
803                     # 勝利数はリセット
804                     WIN = 0
805
806         # 石を置く
807         self.place(mostWinX, mostWinY, M_COM_COLOR)
808
809     def countNone(self):
810         '''盤面リストの None の数を数える'''
811
812         numOfNone = 0 # 初期状態は 0 最大は 252
813
814         for j in range(16):
815             for i in range(16):
816                 if self.board[j][i] == None:
817                     numOfNone += 1
818
819         return numOfNone
820
821     def load(self):
822         '''モンテカルロ法用の盤面に現在の盤面を読み込む'''
823
824         # モンテカルロ法を用いる際の盤面上の石を管理する 2 次元リストを作成（最初は全て
                None ）
825         self.mBoard = [[None] * NUM_SQUARE for i in range(NUM_SQUARE)]
826
827         for j in range(16):
828             for i in range(16):
829                 self.mBoard[j][i] = self.board[j][i]
830
831     def getPlacableM(self):
832         '''次に置くことができる石の位置を取得（モンテカルロ法専用）'''
833
834         placable = []
835
836         for y in range(NUM_SQUARE):
837             for x in range(NUM_SQUARE):
838                 # (x,y) の位置のマスに石が置けるかどうかをチェック
839                 if self.checkPlacableM(x, y):
840                     # 置けるならその座標をリストに追加
841                     placable.append((x, y))
```

```
842
843             return placable
844
845         def checkPlacableM(self, x, y):
846             '''(x,y)に石が置けるかどうかをチェック（モンテカルロ法専用）'''
847
848             # その場所に石が置かれていれば置けない
849             if self.mBoard[y][x] != None:
850                 return False
851
852             if self.player == M_COM:
853                 other = COM
854             else:
855                 other = M_COM
856
857             # (x,y)座標から縦横斜め全方向に対して相手の石が裏返せるかどうかを確認
858             for j in range(-1, 2):
859                 for i in range(-1, 2):
860
861                     # 真ん中方向はチェックしてもしょうがないので次の方向の確認に移る
862                     if i == 0 and j == 0:
863                         continue
864
865                     originX = x # 本来の x 座標を保持
866                     originY = y # 本来の y 座標を保持
867
868                     # x + i が 0 を下回る場合は、
869                     # x の値を NUM_SQUARE + 1 にすることで範囲外を参照しないようにする
                            （操作 1 ）
870                     # 例えば x = 0 、i = -1 の場合は x の値を 16 にすることで
                            x + i が 15 になる
871                     if x + i < 0:
872                         x = NUM_SQUARE
873
874                     # x + i が NUM_SQUARE を上回る場合は、
875                     # x の値を -1 にすることで範囲外を参照しないようにする（操作 1 ）
876                     # 例えば x = 15 、i = 1 の場合は x の値を -1 にすることで
                            x + i が 0 になる
877                     if x + i >= NUM_SQUARE:
878                         x = -1
879
```

```
                # x の場合と同様（操作 1 ）
                if y + j < 0:
                    y = NUM_SQUARE;

                # y の場合と同様（操作 1 ）
                if y + j >= NUM_SQUARE:
                    y = -1

                # 隣が相手の色でなければその方向に石を置いても裏返せない
                if self.mBoard[y + j][x + i] != self.color[other]:
                    # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                    if originX != x:
                        x = originX

                    if originY != y:
                        y = originY
                    continue

                # 置こうとしているマスから遠い方向へ 1 マスずつ確認
                for s in range(2, NUM_SQUARE):

                    # x + i * s が 0 を下回る場合は、
                    # x の値を NUM_SQUARE + s にすることで範囲外を参照しないようにする（操作 2 ）
                    # 例えば x = 5 、 i = -1 、 s = 7 の場合は、
                    # x の値を 16 + 5 で 21 にすることで x + i * s が 14 になる
                    if x + i * s < 0:
                        x = NUM_SQUARE + x

                    # x + i * s が NUM_SQUARE を上回る場合は、
                    # x + i * s の値をから NUM_SQUARE を引いて 1 を足すことで範囲外を参照しないようにする（操作 2 ）
                    # 例えば x = 10 、 i = 1 、 s = 7 の場合は、
                    # x の値を 17 - 16 - 7 にすることで x + i * s が 1 になる
                    if x + i * s >= NUM_SQUARE:
                        x = x + i * s - NUM_SQUARE - (i * s)

                    # x の場合と同様（操作 2 ）
                    if y + j * s < 0:
                        y = NUM_SQUARE + y
```

```
                # y の場合と同様（操作 2 ）
                if y + j * s >= NUM_SQUARE:
                    y = y + i * s - NUM_SQUARE - (i * s)

                if self.mBoard[y + j * s][x + i * s] == None:
                    # 自分の石が見つかる前に空きがある場合
                    # この方向の石は裏返せないので次の方向をチェック
                    # 操作 1 もしくは操作 2 を行った場合、 x の値と y の値を本来の値に戻す
                    if originX != x:
                        x = originX

                    if originY != y:
                        y = originY
                    break

                # その方向に自分の色の石があれば石が裏返せる
                if self.mBoard[y + j * s][x + i * s] == self.color[self.player]:
                    # 操作 1 もしくは操作 2 を行った場合、 x の値と y の値を本来の値に戻す
                    if originX != x:
                        x = originX

                    if originY != y:
                        y = originY
                    return True

            # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
            if originX != x:
                x = originX

            if originY != y:
                y = originY

            # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
            if originX != x:
                x = originX

```

```
955                 if originY != y:
956                     y = originY
957
958         # 裏返せる石がなかったので(x,y)に石は置けない
959         return False
960
961     def placeM(self, x, y, color):
962         '''(x,y)に色が color の石を置く（モンテカルロ法専用）'''
963
964         if x >= NUM_SQUARE // 2 and y < NUM_SQUARE // 2: # 第一象限を選択した場合
965             '''(x,y)に色がの石を置くcolor'''
966
967             # (x,y)に石が置かれた時に裏返る石を裏返す
968             self.reverseListM(x, y) # 第一象限で裏返る石のリストを作る
969             self.reverseListM(x - 8, y) # 第二象限で裏返る石のリストを作る
970             self.reverseListM(x - 8, y + 8) # 第三象限で裏返る石のリストを作る
971             self.reverseListM(x, y + 8) # 第四象限で裏返る石のリストを作る
972             self.reverseM() # リストをもとに裏返す
973
974             # (x,y)に石を置く（円を描画する）
975             self.memorizeColor(x, y, color) # 第一象限
976             self.memorizeColor(x - 8, y, color) # 第二象限
977             self.memorizeColor(x - 8, y + 8, color) # 第三象限
978             self.memorizeColor(x, y + 8, color) # 第四象限
979
980         elif x < NUM_SQUARE // 2 and y < NUM_SQUARE // 2: # 第二象限を選択した場合
981
982             # (x,y)に石が置かれた時に裏返る石を裏返す
983             self.reverseListM(x + 8, y) # 第一象限で裏返る石のリストを作る
984             self.reverseListM(x, y) # 第二象限で裏返る石のリストを作る
985             self.reverseListM(x, y + 8) # 第三象限で裏返る石のリストを作る
986             self.reverseListM(x + 8, y + 8) # 第四象限で裏返る石のリストを作る
987             self.reverseM() # リストをもとに裏返す
988
989             # (x,y)に石を置く（円を描画する）
990             self.memorizeColor(x + 8, y, color) # 第一象限
991             self.memorizeColor(x, y, color) # 第二象限
992             self.memorizeColor(x, y + 8, color) # 第三象限
```

```
993                 self.memorizeColor(x + 8, y + 8, color) # 第四象限
994
995             elif x < NUM_SQUARE // 2 and y >= NUM_SQUARE // 2: # 第三象限を選択
                    した場合
996
997                 # (x,y)に石が置かれた時に裏返る石を裏返す
998                 self.reverseListM(x + 8, y - 8) # 第一象限で裏返る石のリストを作る
999                 self.reverseListM(x, y - 8) # 第二象限で裏返る石のリストを作る
1000                self.reverseListM(x, y) # 第三象限で裏返る石のリストを作る
1001                self.reverseListM(x + 8, y) # 第四象限で裏返る石のリストを作る
1002                self.reverseM() # リストをもとに裏返す
1003
1004                # (x,y)に石を置く（円を描画する）
1005                self.memorizeColor(x + 8, y - 8, color) # 第一象限
1006                self.memorizeColor(x, y - 8, color) # 第二象限
1007                self.memorizeColor(x, y, color) # 第三象限
1008                self.memorizeColor(x + 8, y, color) # 第四象限
1009
1010            else: # 第四象限を選択した場合
1011
1012                # (x,y)に石が置かれた時に裏返る石を裏返す
1013                self.reverseListM(x, y - 8) # 第一象限で裏返る石のリストを作る
1014                self.reverseListM(x - 8, y - 8) # 第二象限で裏返る石のリストを作る
1015                self.reverseListM(x - 8, y) # 第三象限で裏返る石のリストを作る
1016                self.reverseListM(x, y) # 第四象限で裏返る石のリストを作る
1017                self.reverseM() # リストをもとに裏返す
1018
1019                # (x,y)に石を置く（円を描画する）
1020                self.memorizeColor(x, y - 8, color) # 第一象限
1021                self.memorizeColor(x - 8, y - 8, color) # 第二象限
1022                self.memorizeColor(x - 8, y, color) # 第三象限
1023                self.memorizeColor(x, y, color) # 第四象限
1024
1025            # 次に石を置くプレイヤーを決める
1026            before_player = self.player
1027            self.nextPlayerM()
1028
1029            # 裏返す石を管理する 2 次元リストを初期化
1030            self.revBoard = [[None] * NUM_SQUARE for i in range(NUM_SQUARE)
                    ]
```

```
1031
1032            if not self.player:
1033                # 次に石が置けるプレイヤーがいない場合はゲーム終了
1034                self.endGame()
1035                self.player = M_COM # 次に置く石の色
1036                self.mBoard = None # 盤面上の石を管理する次元リスト2
1037                return
1038
1039            if self.player == M_COM:
1040                # 次のプレイヤーが M_COM の場合は M_COM に石を置く場所を決めさせる
1041                self.com2()
1042            else:
1043                # 次のプレイヤーが COM の場合は COM に石を置く場所を決めさせる
1044                self.com1()
1045
1046        def reverseListM(self, x, y):
1047            '''(x,y)に石が置かれた時に裏返す必要のある石をリストに格納する（モンテカルロ法専用）
                '''
1048
1049            if self.mBoard[y][x] != None:
1050                # (x,y)にすでに石が置かれている場合は何もしない
1051                return
1052
1053            if self.player == COM:
1054                other = M_COM
1055            else:
1056                other = COM
1057
1058            for j in range(-1, 2):
1059                for i in range(-1, 2):
1060                    # 真ん中方向はチェックしてもしょうがないので次の方向の確認に移る
1061                    if i == 0 and j == 0:
1062                        continue
1063
1064                    originX = x # 本来の x 座標を保持
1065                    originY = y # 本来の y 座標を保持
1066
1067
1068                    # x + i が 0 を下回る場合は、
1069                    # x の値を NUM_SQUARE + 1 にすることで範囲外を参照しないようにする
```

```
                （操作 1 ）
# 例えば x = 0 、 i = -1 の場合は x の値を 16 にすることで x + i が 15 になる
if x + i < 0:
    x = NUM_SQUARE

# x + i が NUM_SQUARE を上回る場合は、
# x の値を -1 にすることで範囲外を参照しないようにする（操作 1 ）
# 例えば x = 、15i = 1 の場合は x の値を -1 にすることで x + i が 0 になる
if x + i >= NUM_SQUARE:
    x = -1

# x の場合と同様（操作 1 ）
if y + j < 0:
    y = NUM_SQUARE

# y の場合と同様（操作 1 ）
if y + j >= NUM_SQUARE:
    y = -1

# 隣が相手の色でなければその方向で裏返せる石はない
if self.mBoard[y + j][x + i] != self.color[other]:
    # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
    if originX != x:
        x = originX

    if originY != y:
        y = originY
    continue

# 置こうとしているマスから遠い方向へ 1 マスずつ確認
for s in range(2, NUM_SQUARE):

    # x + i * s が 0 を下回る場合は、
    # x の値を NUM_SQUARE + s にすることで範囲外を参照しないようにする（操作 2 ）
    # 例えば x = 5 、 i = -1 、 s = 7 の場合は x の値を 16 + 5 で 21 にすることで x + i * s が 14 になる
    if x + i * s < 0:
        x = NUM_SQUARE + x
```

```
                # x + i * s が NUM_SQUARE を上回る場合は、
                # x + i * s の値をから NUM_SQUARE を引いて 1 を足すことで範囲外を参照しないようにする（操作 2 ）
                # 例えば x = 10 、 i = 1 、 s = 7 の場合は x の値を17 - 16 - 7 にすることで x + i * s が 1 になる
                if x + i * s >= NUM_SQUARE:
                    x = x + i * s - NUM_SQUARE - (i * s)

                # x の場合と同様（操作 2 ）
                if y + j * s < 0:
                    y = NUM_SQUARE + y

                # y の場合と同様（操作 2 ）
                if y + j * s >= NUM_SQUARE:
                    y = y + i * s - NUM_SQUARE - (i * s)

                if self.mBoard[y + j * s][x + i * s] == None:
                    # 自分の石が見つかる前に空きがある場合
                    # この方向の石は裏返せないので次の方向をチェック
                    # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                    if originX != x:
                        x = originX

                    if originY != y:
                        y = originY
                    break

                # その方向に自分の色の石があれば石が裏返せる
                if self.mBoard[y + j * s][x + i * s] == self.color[self.player]:

                    for n in range(1, s):

                        # 盤面の石の管理リストを石を裏返した状態に更新
                        self.revBoard[(y + j * n) % NUM_SQUARE][(x + i * n) % NUM_SQUARE] = self.color[self.player]

                    # 操作 1 もしくは操作 2 を行った場合、 x の値と y の値を本来の値に戻す
```

```
                        if originX != x:
                            x = originX

                        if originY != y:
                            y = originY

                        break

                # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                if originX != x:
                    x = originX

                if originY != y:
                    y = originY

                # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                if originX != x:
                    x = originX

                if originY != y:
                    y = originY

    def reverseM(self):
        '''リストを基に石が置かれた時に裏返す必要のある石を裏返す（モンテカルロ法専用）'''

        for j in range(0, NUM_SQUARE):
            for i in range(0, NUM_SQUARE):
                if self.revBoard[j][i] != None:
                    # 盤面の石の管理リストを石を裏返した状態に更新
                    self.mBoard[j][i] = self.revBoard[j][i]

    def memorizeColor(self, x, y, color):
        '''描画した円の色を管理リストに記憶させておく（モンテカルロ法専用）'''

        self.mBoard[y][x] = color

    def nextPlayerM(self):
        '''次に石を置くプレイヤーを決める（モンテカルロ法専用）'''

        before_player = self.player
```

```
1181
1182            # 石を置くプレイヤーを切り替える
1183            if self.player == M_COM:
1184                self.player = COM
1185            else:
1186                self.player = M_COM
1187
1188            # 切り替え後のプレイヤーが石を置けるかどうかを確認
1189            placable = self.getPlacableM()
1190
1191            if len(placable) == 0:
1192                # 石が置けないのであればスキップ
1193                self.player = before_player
1194
1195                # スキップ後のプレイヤーが石を置けるかどうかを確認
1196                placable = self.getPlacableM()
1197
1198                if len(placable) == 0:
1199                    # それでも置けないのであれば両者とも石を置けないということ
1200                    self.player = None
1201
1202        def endGame(self):
1203            '''ゲーム終了後勝った場合は勝ち点を足す（モンテカルロ法専用）'''
1204
1205            # それぞれの色の石の数を数える
1206            num_mcom = 0
1207            num_com = 0
1208
1209            for y in range(NUM_SQUARE):
1210                for x in range(NUM_SQUARE):
1211                    if self.mBoard[y][x] == M_COM_COLOR:
1212                        num_mcom += 1
1213                    elif self.mBoard[y][x] == COM_COLOR:
1214                        num_com += 1
1215
1216            num_mcom /= 4
1217            num_com /= 4
1218
1219            global WIN
1220
```

```
1221            # 相手より自石が多い（勝った）場合は勝ち点を 1 足す
1222            if num_mcom > num_com:
1223                WIN += 1
1224
1225        def com1(self):
1226            '''モンテカル法で相手が石を置く'''
1227
1228            # 石が置けるマスを取得
1229            placable = self.getPlacableM()
1230
1231            # 置くマスをランダムに選ぶ
1232            x, y = random.choice(placable)
1233
1234            # 石を置く
1235            self.placeM(x, y, COM_COLOR)
1236
1237        def com2(self):
1238            '''モンテカル法で自分が石を置く'''
1239
1240            # 石が置けるマスを取得
1241            placable = self.getPlacableM()
1242
1243            # 置くマスをランダムに選ぶ
1244            x, y = random.choice(placable)
1245
1246            # 石を置く
1247            self.placeM(x, y, M_COM_COLOR)
1248
1249        def straight(self):
1250            '''一直線に石を置いていく直線戦略'''
1251
1252            # 石が置けるマスを取得
1253            placable = self.getPlacable()
1254
1255            # 座標格納用変数
1256            copyX = -1
1257            copyY = -1
1258
1259            # 最小の白石の数
1260            minWhiteNum = 100
```

```
1261
1262            for j in range(8):
1263                for i in range(8):
1264                    # 座標に石を置くことができる場合
1265                    if self.checkPlacable(i,j) == True:
1266
1267                        # 全体の白石の数
1268                        whiteSum = 0
1269
1270                        # コピー盤面を読み込み
1271                        self.copy()
1272
1273                        # 石を置く
1274                        self.placeC(i,j,M_COM_COLOR)
1275
1276                        if self.judgeStraight():
1277                            for b in range(8):
1278                                for a in range(8):
1279                                    if self.boardCopy[b][a] == 'white':
1280                                        whiteSum += 1
1281
1282                            # 全体の白石の数が最小の白石の数よりも少ない場合、最小の白石と座
                                標を更新する
1283                            if minWhiteNum > whiteSum:
1284                                minWhiteNum = whiteSum
1285                                copyX = i
1286                                copyY = j
1287
1288            for j in range(8):
1289                for i in range(8):
1290                    # 座標に石を置くことができる場合
1291                    if self.checkPlacable(i,j) == True:
1292
1293                        # コピー盤面を読み込み
1294                        self.copy()
1295
1296                        # 石を置く
1297                        self.placeC(i,j,M_COM_COLOR)
1298
1299                        # 全体の白石の数
```

```
                    whiteNum = 0

                    # 盤面上の白石の数を数える
                    for s in range(NUM_SQUARE):
                        if 'white' in self.boardCopy[s]:
                            whiteNum += 1

                    # 盤面上の白石が 0 の場合、勝ちが確定なのでその座標に更新する
                    if whiteNum == 0:
                        copyX = i
                        copyY = j

        # 座標が見つからなかった場合
        if copyX == -1:
            # モンテカルロ法に移行する
            self.m_com()
        # 座標が見つかった場合
        else:
            # 石を置く
            self.place(copyX, copyY, M_COM_COLOR)

    def copy(self):
        '''コピー盤面に現在の盤面を読み込む'''

        # 盤面上の石を管理する 2 次元リストを作成（最初は全て None ）
        self.boardCopy = [[None] * NUM_SQUARE for i in range(NUM_SQUARE
            )]

        for j in range(16):
            for i in range(16):
                self.boardCopy[j][i] = self.board[j][i]

    def judgeStraight(self):
        '''直線状か判断する'''

        # 座標格納用変数
        whiteX = -1
        whiteY = -1

        # 全体の白石の数
```

```
1339            whiteNum = 0
1340
1341            # 基準とする白石を決め、全体の白石の数を数える
1342            for j in range(8):
1343                for i in range(8):
1344                    if self.boardCopy[j][i] == 'white':
1345                        whiteX = i
1346                        whiteY = j
1347                        whiteNum += 1
1348
1349            # 直線の数
1350            straight = 0
1351
1352            # 一つの直線における白石の数
1353            straightWhite = 0
1354
1355            # 全体の白石がの場合1
1356            if whiteNum == 1:
1357                return True
1358            # 全体の白石が 2 以上の場合
1359            else:
1360                for j in range(-1, 2):
1361                    for i in range(-1, 2):
1362                        # 真ん中方向はチェックしてもしょうがないので次の方向の確認に移る
1363                        if i == 0 and j == 0:
1364                            continue
1365
1366                        # 一直線判定用の 1 次元リストを作成（最初は全て None ）
1367                        self.straightBoard = [None] * 7
1368
1369                        # 隣が白石でなければ直線ではないので処理を飛ばす
1370                        if self.boardCopy[(whiteY + j) % 8][(whiteX + i) %
                                8] != COM_COLOR:
1371                            continue
1372
1373                        # 石をリストに追加していく
1374                        for s in range(1, 7):
1375                            self.straightBoard[s - 1] = self.boardCopy[(
                                    whiteY + j * s) % 8][(whiteX + i * s) % 8]
1376
```

```
1377                        # 一つの直線における白石の数を数える
1378                        for a in range(7):
1379                            if self.straightBoard[a] == COM_COLOR:
1380                                straightWhite += 1
1381
1382                        # 基準の隣が白の場合
1383                        if self.straightBoard[0] == COM_COLOR:
1384                            for b in range(1, 6):
1385                                # 黒や空白が現れた場合
1386                                if self.straightBoard[b] == M_COM_COLOR or
                                       self.straightBoard[b] == None:
1387                                    if b == 6:
1388                                        if whiteNum - 1 == straightWhite:
1389                                            straight += 1
1390                                        straightWhite = 0
1391                                        break
1392                                    for c in range(b + 1, 6):
1393                                        if  self.straightBoard[c]==
                                               COM_COLOR:
1394                                            break
1395                                    else:
1396                                        if whiteNum - 1 == straightWhite:
1397                                            straight += 1
1398                                            straightWhite = 0
1399                                            break
1400                                        else:
1401                                            straightWhite = 0
1402                                            continue
1403                                    break
1404
1405            # 直線が 1 つの場合
1406            if straight ==1:
1407                return True
1408            else:
1409                return False
1410
1411    def placeC(self, x, y, color):
1412        '''(x,y)に色が color の石を置く（直線戦略専用）'''
1413
1414        if x >= NUM_SQUARE // 2 and y < NUM_SQUARE // 2: # 第一象限を選択し
```

```
            た場合
1415            '''(x,y)に色がの石を置くcolor'''
1416
1417            # (x,y)に石が置かれた時に裏返る石を裏返す
1418            self.reverseListC(x, y) # 第一象限で裏返る石のリストを作る
1419            self.reverseListC(x - 8, y) # 第二象限で裏返る石のリストを作る
1420            self.reverseListC(x - 8, y + 8) # 第三象限で裏返る石のリストを作る
1421            self.reverseListC(x, y + 8) # 第四象限で裏返る石のリストを作る
1422            self.reverseC() # リストをもとに裏返す
1423
1424            # (x,y)に石を置く（円を描画する）
1425            self.memorizeColorC(x, y, color) # 第一象限
1426            self.memorizeColorC(x - 8, y, color) # 第二象限
1427            self.memorizeColorC(x - 8, y + 8, color) # 第三象限
1428            self.memorizeColorC(x, y + 8, color) # 第四象限
1429
1430        elif x < NUM_SQUARE // 2 and y < NUM_SQUARE // 2: # 第二象限を選択し
            た場合
1431
1432            # (x,y)に石が置かれた時に裏返る石を裏返す
1433            self.reverseListC(x + 8, y) # 第一象限で裏返る石のリストを作る
1434            self.reverseListC(x, y) # 第二象限で裏返る石のリストを作る
1435            self.reverseListC(x, y + 8) # 第三象限で裏返る石のリストを作る
1436            self.reverseListC(x + 8, y + 8) # 第四象限で裏返る石のリストを作る
1437            self.reverseC() # リストをもとに裏返す
1438
1439            # (x,y)に石を置く（円を描画する）
1440            self.memorizeColorC(x + 8, y, color) # 第一象限
1441            self.memorizeColorC(x, y, color) # 第二象限
1442            self.memorizeColorC(x, y + 8, color) # 第三象限
1443            self.memorizeColorC(x + 8, y + 8, color) # 第四象限
1444
1445        elif x < NUM_SQUARE // 2 and y >= NUM_SQUARE // 2: # 第三象限を選択
            した場合
1446
1447            # (x,y)に石が置かれた時に裏返る石を裏返す
1448            self.reverseListC(x + 8, y - 8) # 第一象限で裏返る石のリストを作る
1449            self.reverseListC(x, y - 8) # 第二象限で裏返る石のリストを作る
1450            self.reverseListC(x, y) # 第三象限で裏返る石のリストを作る
1451            self.reverseListC(x + 8, y) # 第四象限で裏返る石のリストを作る
1452            self.reverseC() # リストをもとに裏返す
```

```
1453
1454                # (x,y)に石を置く（円を描画する）
1455                self.memorizeColorC(x + 8, y - 8, color) # 第一象限
1456                self.memorizeColorC(x, y - 8, color) # 第二象限
1457                self.memorizeColorC(x, y, color) # 第三象限
1458                self.memorizeColorC(x + 8, y, color) # 第四象限
1459
1460            else: # 第四象限を選択した場合
1461
1462                # (x,y)に石が置かれた時に裏返る石を裏返す
1463                self.reverseListC(x, y - 8) # 第一象限で裏返る石のリストを作る
1464                self.reverseListC(x - 8, y - 8) # 第二象限で裏返る石のリストを作る
1465                self.reverseListC(x - 8, y) # 第三象限で裏返る石のリストを作る
1466                self.reverseListC(x, y) # 第四象限で裏返る石のリストを作る
1467                self.reverseC() # リストをもとに裏返す
1468
1469                # (x,y)に石を置く（円を描画する）
1470                self.memorizeColorC(x, y - 8, color) # 第一象限
1471                self.memorizeColorC(x - 8, y - 8, color) # 第二象限
1472                self.memorizeColorC(x - 8, y, color) # 第三象限
1473                self.memorizeColorC(x, y, color) # 第四象限
1474
1475            # 裏返す石を管理する 2 次元リストを初期化
1476            self.revBoard = [[None] * NUM_SQUARE for i in range(NUM_SQUARE)
                ]
1477
1478        def reverseListC(self, x, y):
1479            '''(x,y)に石が置かれた時に裏返す必要のある石をリストに格納する（直線戦略専用）'''
1480
1481            if self.boardCopy[y][x] != None:
1482                # (x,y)にすでに石が置かれている場合は何もしない
1483                return
1484
1485            if self.player == COM:
1486                other = M_COM
1487            else:
1488                other = COM
1489
1490            for j in range(-1, 2):
1491                for i in range(-1, 2):
```

```
                # 真ん中方向はチェックしてもしょうがないので次の方向の確認に移る
                if i == 0 and j == 0:
                    continue

                originX = x # 本来の x 座標を保持
                originY = y # 本来の y 座標を保持


                # x + i が 0 を下回る場合は、
                # x の値を NUM_SQUARE + 1 にすることで範囲外を参照しないようにする（操作 1 ）
                # 例えば x = 0 、 i = -1 の場合は x の値を 16 にすることで x + i が 15 になる
                if x + i < 0:
                    x = NUM_SQUARE

                # x + i が NUM_SQUARE を上回る場合は、
                # x の値を -1 にすることで範囲外を参照しないようにする（操作 1 ）
                # 例えば x = 15 、i = 1 の場合は x の値を -1 にすることで x + i が 0 になる
                if x + i >= NUM_SQUARE:
                    x = -1

                # x の場合と同様（操作 1 ）
                if y + j < 0:
                    y = NUM_SQUARE

                # y の場合と同様（操作 1 ）
                if y + j >= NUM_SQUARE:
                    y = -1

                # 隣が相手の色でなければその方向で裏返せる石はない
                if self.boardCopy[y + j][x + i] != self.color[other]:
                    # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                    if originX != x:
                        x = originX

                    if originY != y:
                        y = originY
                    continue
```

```
                # 置こうとしているマスから遠い方向へ 1 マスずつ確認
                for s in range(2, NUM_SQUARE):

                    # x + i * s が 0 を下回る場合は、
                    # x の値を NUM_SQUARE + s にすることで範囲外を参照しないようにする（操作 2 ）
                    # 例えば x = 5 、 i = -1 、 s = 7 の場合は x の値を 16 + 5 で 21 にすることで x + i * s が 14 になる
                    if x + i * s < 0:
                        x = NUM_SQUARE + x

                    # x + i * s が NUM_SQUARE を上回る場合は、
                    # x + i * s の値をから NUM_SQUARE を引いて 1 を足すことで範囲外を参照しないようにする（操作 2 ）
                    # 例えば x = 10 、 i = 1 、 s = 7 の場合は x の値を 17 - 16 - 7 にすることで x + i * s が 1 になる
                    if x + i * s >= NUM_SQUARE:
                        x = x + i * s - NUM_SQUARE - (i * s)

                    # x の場合と同様（操作 2 ）
                    if y + j * s < 0:
                        y = NUM_SQUARE + y

                    # y の場合と同様（操作 2 ）
                    if y + j * s >= NUM_SQUARE:
                        y = y + i * s - NUM_SQUARE - (i * s)

                    if self.boardCopy[y + j * s][x + i * s] == None:
                        # 自分の石が見つかる前に空きがある場合
                        # この方向の石は裏返せないので次の方向をチェック
                        # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                        if originX != x:
                            x = originX

                        if originY != y:
                            y = originY
                        break

                    # その方向に自分の色の石があれば石が裏返せる
                    if self.boardCopy[y + j * s][x + i * s] == self.
                       color[self.player]:
```

```
                        for n in range(1, s):

                            # 盤面の石の管理リストを石を裏返した状態に更新
                            self.revBoard[(y + j * n) % NUM_SQUARE][(x
                                + i * n) % NUM_SQUARE] = self.color[self
                                .player]

                        # 操作 1 もしくは操作 2 を行った場合、 x の値と y の値を本来
                            の値に戻す
                        if originX != x:
                            x = originX

                        if originY != y:
                            y = originY

                        break

                # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                if originX != x:
                    x = originX

                if originY != y:
                    y = originY

                # 操作 1 を行った場合、 x の値と y の値を本来の値に戻す
                if originX != x:
                    x = originX

                if originY != y:
                    y = originY

    def reverseC(self):
        '''リストを基に石が置かれた時に裏返す必要のある石を裏返す（直線戦略専用）'''

        for j in range(0, NUM_SQUARE):
            for i in range(0, NUM_SQUARE):
                if self.revBoard[j][i] != None:
                    # 盤面の石の管理リストを石を裏返した状態に更新
                    self.boardCopy[j][i] = self.revBoard[j][i]
```

```
1603
1604        def memorizeColorC(self, x, y, color):
1605            '''描画した円の色を管理リストに記憶させておく（直線戦略専用）'''
1606
1607            self.boardCopy[y][x] = color
1608
1609    # スクリプト処理ここから
1610    app = tkinter.Tk()
1611    app.title('othello')
1612    othello = Othello(app)
1613    app.mainloop()
```