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

19-1-037-0110

石倉慎

近畿大学理工学部情報学科

令和４年２月２日提出

概要

リバーシは８×８の盤面を用い黒と白の石を交互に打っていく二人零和完全情報ゲームである.リバーシは、自石で敵石を挟めばその敵石をひっくり返して自石にできる．このとき角のマスに置かれている石はゲーム終了まで絶対に裏返すことができず，また，辺に置かれている石もひっくり返されにくいため，リバーシでは角のマスおよび辺のマスを取ることが重要である．このため既存のリバーシ AI では角や辺の裏返しにくいマスを重要視している.

リバーシには様々なバリエーションがあり，その一つにトーラス型リバーシがある．トーラス型のリバーシは８×８の盤面を用いるのは同じだが,盤面の上下左右はつながっているものと考える．このため，角と辺のマスに置かれている石も他の石と同様に裏返すことができるリバーシとなっている.トーラス型のリバーシには既存のリバーシの AI を適用することができず,また,マイナーなゲームなため既存のトーラス型リバーシ AI も存在していない.そこで本研究では,トーラス型リバーシの AI を作成する.

本研究で作成するトーラス型リバーシ AI は,研究初期の段階ではマス目の一つ一つに評価値を決めて,そこから先読みをし,石を打つマスを決めていた.しかし,研究が進むと,モンテカルロ法を用いたほうが思考時間が短くなった.よって,本研究で作成されたトーラス型リバーシ AI はモンテカルロ法が使われている.

目次

1　序論

1.1 本研究の背景

リバーシとは，８×８の盤面と，片面が黒，もう片面が白に塗られた石を用いる 2 人用のボードゲームである．空いているマスに自石を打ったときに，自石で敵石を挟むとその敵石をひっくり返して自石にできる．二人で白と黒の石を交互に打ち合い,最終的に盤面に多く存在している色の石のプレイヤーが勝利となる．

1.2 リバーシの戦略

リバーシは長年遊ばれており，様々な戦略が確立している．

1.1.1. 定石

リバーシには，鼠定石,牛定石,兎定石といった序盤定石が存在する.本小節では，代表的な定石について述べる．まず，定石を説明するために図１のようにリバーシのマスに座標を設定しておく．以下の定石の説明で●を黒石,○を白石とする．

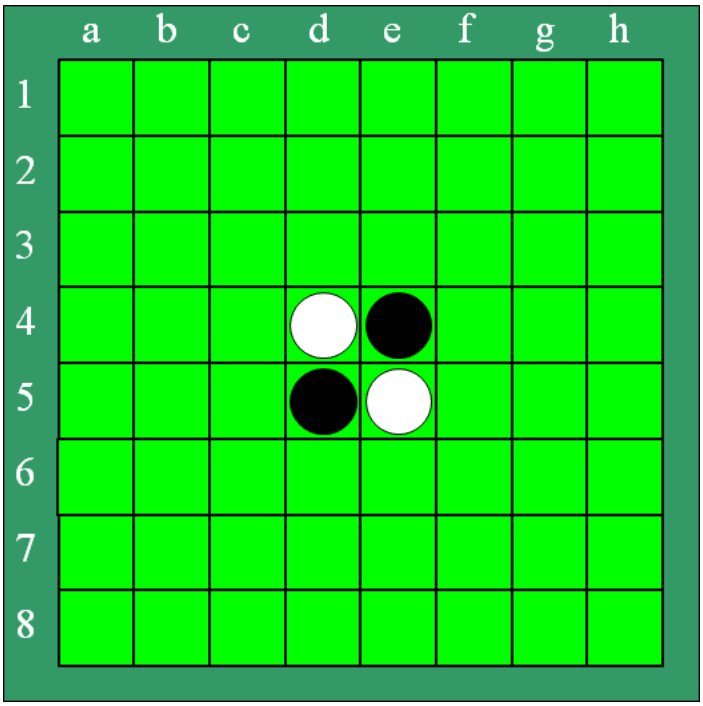

図 1 リバーシの初期状態

- **兎定石**
  兎定石は，●f5→○d6→●c5→○f4→●e3 と打っていく定石である．
- **虎定石**
  虎定石は，●f5→○d6→●c3→○d3→●c4 と打っていく定石である．
- **牛定石**
  牛定石は，●f5→○f6→●e6→○f４→●e3→○c5 と打っていく定石である．

兎定石，虎定石，牛定石の棋譜を図 2 に示す．定石にも上記のもの以外にもまだあり，また，今回最終状態としたものの続きの進行も多数存在している．

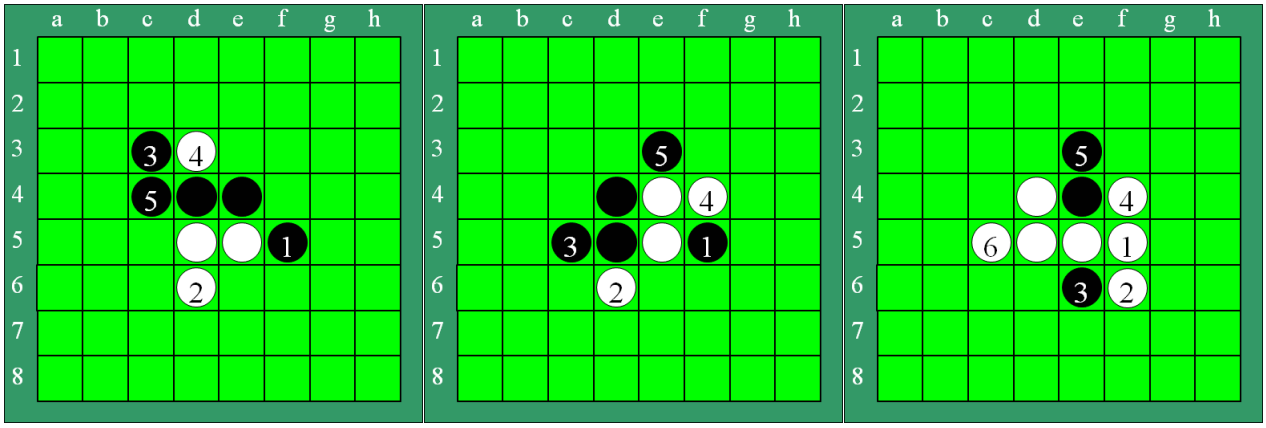

図 2 兎定石，虎定石，牛定石

1.1.2. 確定石と角の重要性

リバーシは,縦横斜めの直線上で相手の石を挟みこむことで,裏返して自分の石にすることができる.しかし,角のマスだけについては,どのような状態の盤面においても,直線上で挟むことができない.挟むことができないので,裏返すことができず,角を取るだけで,最大 4 マスのアドバンテージを取ることができる.このことから,角のマスをとることが,重要なのである.角に置いた石のようにゲーム終了までひっくり返すことができない石は確定石と呼ばれる．また，角に自石があるときに，その自石と隣接する辺上の石も確定石となる．リバーシではできるだけ多くの自石を確定石とすることが重要とされている．図 3 に確定石の例を示す．図 3 の a8 の石は角にあるためひっくり返すことができない．また，a8 に隣接する辺上にある a2…a7 の石もひっくり返すことができない確定石である.

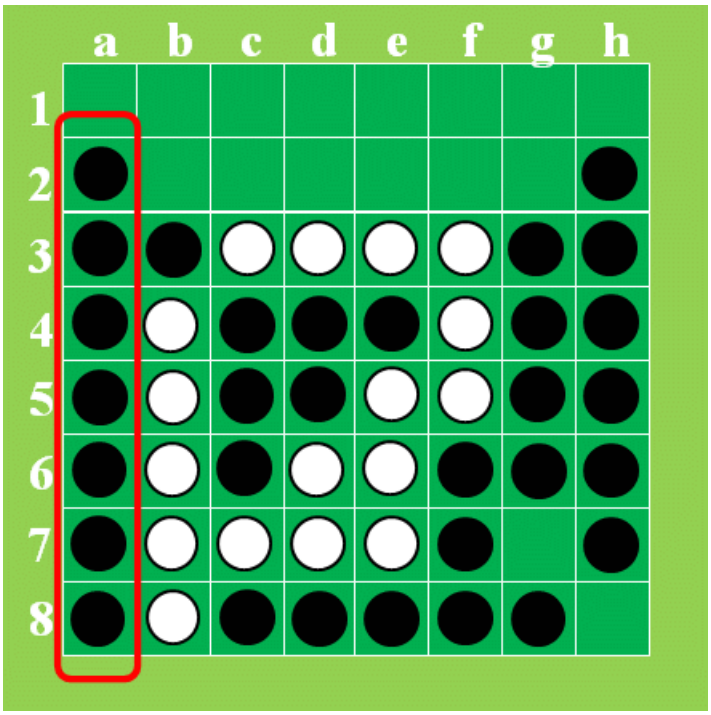

図 3　確定石の例

リバーシではいくつかのマスには名前が付いており，辺上のマスは中心に近いものから順に A マス，B マス，C マスと呼ばれる．また，角マスから斜めに 1 マス内側のマス X マス呼ばれる．図 4 に各マスの名前を示す.

C マスおよび X マスは，角が空いているときにここに自石を置くと相手に角マスを取られやすくなるため，置かない方がいいとされている．一方角に自石がある場合は C マスに自石を置けば確定石となるため，一転して積極的に置いた方がいいマスとなる．

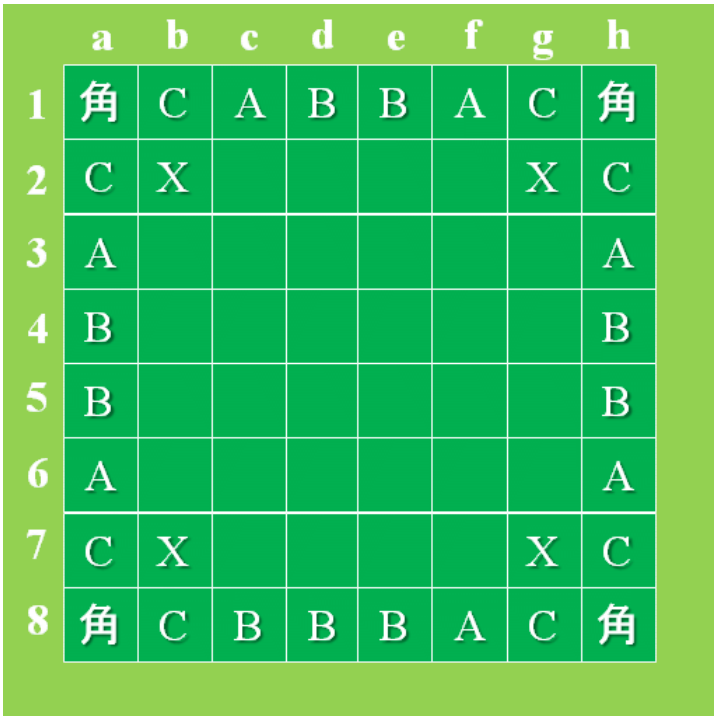

図 4　各マスの名前

1.3 リバーシのバリエーション

リバーシには様々なバリエーションが存在する．盤面サイズの小さいミニリバーシ，盤面が八角形で角が 8 個あるエイトスターズリバーシや,盤面が円形で角がないニップ等がある.図 5 に各バリエーションの盤面と初期局面の石の配置を示す．ミニリバーシは盤面サイズが異なるだけで，通常のリバーシと同様に角マスに置いた石はひっくり返せない確定石となる．一方，エイトスターズリバーシやニップはリバーシと同じルールでありながら,リバーシとは全く異なった戦略が必要となってくる.通常のリバーシは長年遊ばれているため前節で述べた通り様々な戦略が確立している．一方，リバーシの各バリエーションはマイナーなゲームであるためプレイ人口も少なく有望な定石等は作られていない．

リバーシのバリエーションの一つにトーラス型リバーシがある．トーラス型リバーシは盤面の上下左右が繋がっており，角のマスが存在しない．このため，トーラス型リバーシも通常のリバーシとは異なる戦略が必要となる．

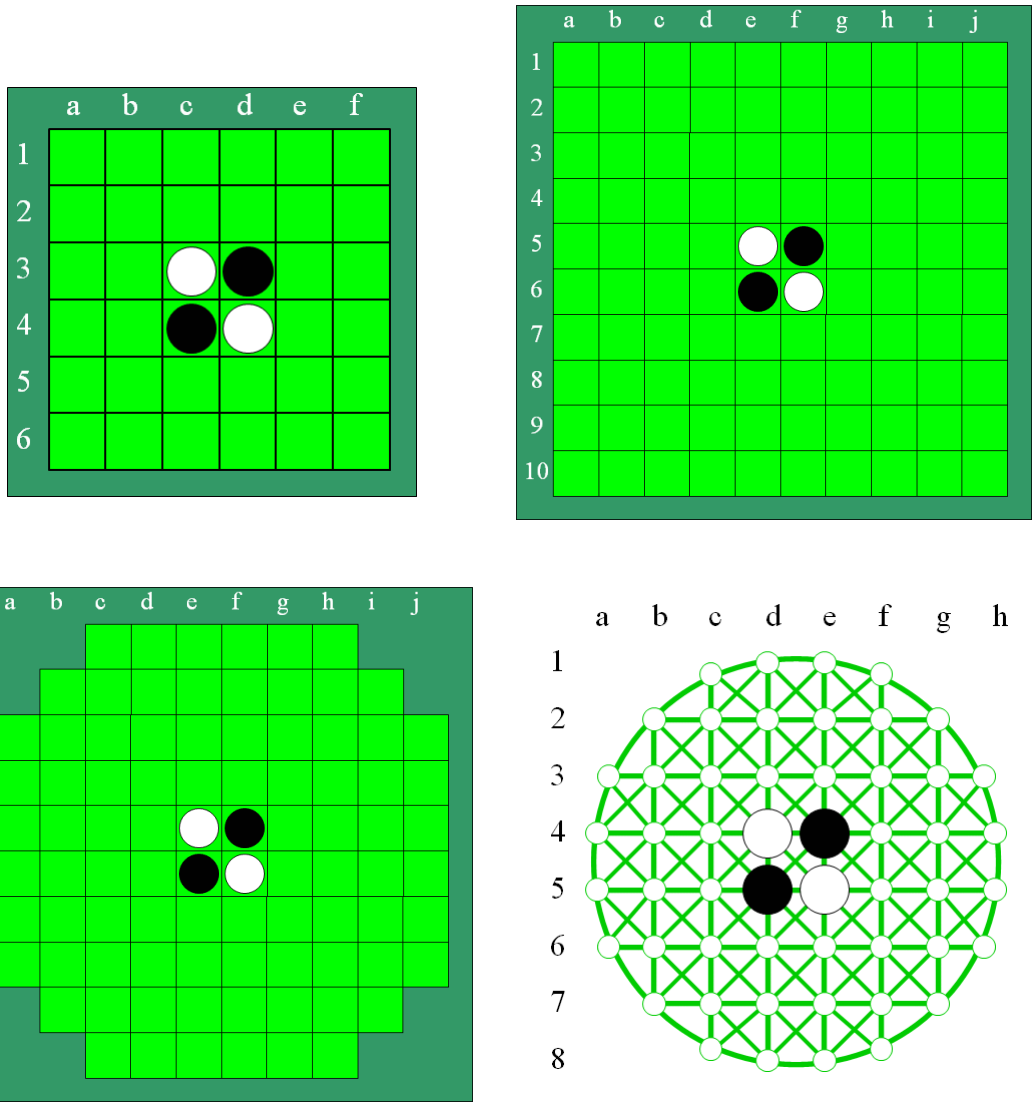

図 5　ミニリバーシ，グランドリバーシ，エイトスターズリバーシ，ニップの盤面と初期配置

1.4 リバーシの既知の結果

リバーシのバリエーションのうち，ミニリバーシの中には完全解析されており，勝敗が確定しているものがある．表 1 にミニリバーシの既知の完全解析結果を示す．

他のバリエーションについては,ニップやエイトスターズリバーシがあるが,どちらも完全解析するには至っていない．

表 1 ミニリバーシ完全解析結果[6][7]

初期配置	サイズ	勝敗	黒石	白石
○● ●○	4x4	後手必勝	3	11
	4x6	先手必勝	20	4
	4x8	先手必勝	28	0
	4x10	先手必勝	39	0
	6x6	後手必勝	16	20
●○ ●○	4x4	後手必勝	6	9
	4x6	先手必勝	21	3
	4x8	先手必勝	28	0
	4x10	先手必勝	32	0
	6x6	後手必勝	17	19

1.5 本研究の目的

トーラス型リバーシはマイナーなゲームであるため, 通常のリバーシのような戦術は確立しておらず定石も無い．また，トーラス型リバーシの AI も存在しない．そこで本研究では，トーラス型リバーシの AI を作成し,その有用性を検証する.

1.6 本報告書の構成

本報告書の構成は以下の通りである.２章でトーラス型リバーシのルールについて説明する.３章では,本研究で作成したプログラムについてのべる.４章で本研究の結果を示し,考察を行う.最後に５章で結論および今後の課題についてのべる.

2　トーラス型リバーシ

本章では，本研究の対象であるトーラス型リバーシについて述べる.

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．図 6 にトーラス型リバーシの盤面および石の初期配置を示す．トーラス型リバーシは８×８の盤面を用いるリバーシである.基本的なルールは通常のリバーシと同じになっているが,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.例えば,図 7 の左側の図の局面の場合において,トーラス型リバーシなら,△で示される g6,h6 の２つの白石は１９手目に a6 に黒石を置くことで右側の図のように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.図７がこの特徴の裏返しの様子である.

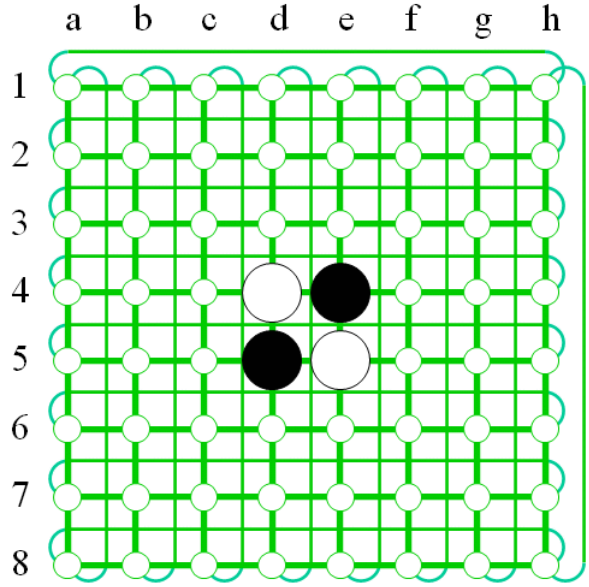

図 6 トーラス型リバーシの盤面と初期状態

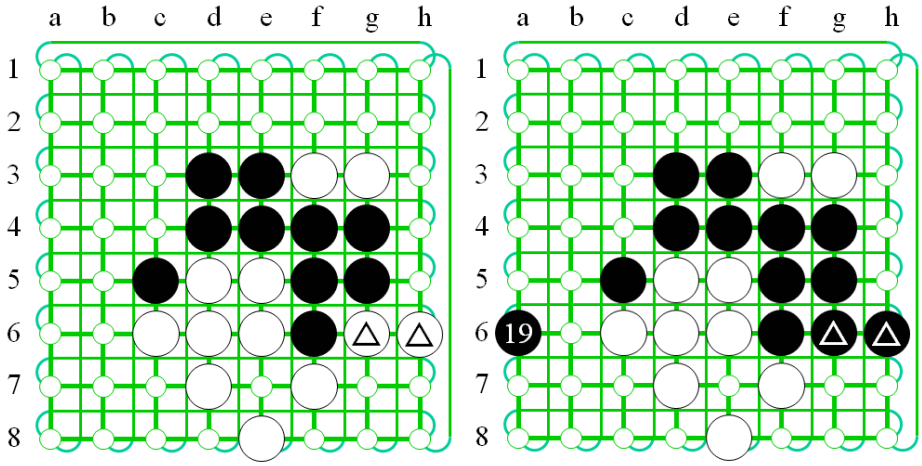

図 7 トーラス型リバーシの端の裏返し

3　トーラス型リバーシの AI

本章では，本研究で作成するトーラス型リバーシの AI について述べる．

3.1 AI の戦略

本節では，本研究で作成する AI が用いている戦略について述べる．

従来のリバーシでは角マスは確定石となるため，いかにして角を確保するかが重要となる．このため，各マスに評価値を設定し，角マスに高い評価値，角に隣接する C マスおよび X マスに低い評価値を設定することで着手を選択する手法などが良く用いられる．しかし，トーラス型リバーシでは実質的に角マスが無いため、このような手法を取ることができない．

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.しかし,モンテカル法は,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.

3.2 プログラム

本節では，本研究で作成したトーラス型リバーシの AI のプログラムについて述べる．本研究では python を用いてプログラムを作成した．付録に本研究で作成したプログラムのソースを示す.また,

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.

- banmen メソッド
  このメソッドで,リバーシの盤面を作成している.8×8 の盤面を生成しており,画面下部に現在,どちらのターンであるのかを表示するための空白も用意している.
- click メソッド
  このメソッドで,実際のプレイ時にクリックした場所に応じて,そのクリックした場所がどのマスに該当するのかを決定している.また,石を置くことができる場所よりも外側をクリックした時は,一番外側のマスをクリックしたことにする処理も行なっている.
- ban_syokika メソッド
  このメソッドで盤面を初期化して,リバーシの初期状態に設定している.
- hitStone メソッド
  このメソッドで打った石がどの石を裏返すことができるのかを確認している.
- hitable メソッド
  このメソッドで石を打てるなら,どれだけ裏返すことができるのかを数えている.
- result メソッド
  このメソッドは黒石と白石がどれだけあるのか数えて最終的にどちらが勝っているのかを判定している.
- save メソッド
  ここで,現在の盤面を back に保存している.
- load メソッド
  保存した盤面を読み込んでいる.
- uchiau メソッド
  このメソッドで,黒石と白石をランダムに最後まで打ち合う処理をしている.
- computer_2 メソッド
  このメソッドがモンテカルロ法による戦略の中心となっている.ここでランダムに打ち合い勝敗を調べているので,save で盤面を保存している.この時に win[]に勝った回数をプラスしていき,勝った回数が多いマスを選択している.

4 実行結果及び考察

図８がトーラス型リバーシ AI との対戦の様子である.自分の番に回ってきた時に,どの場所に打つことができるのかをわかりやすくするために青丸を描写している.図８より,反対側の石であっても裏返されていることが確認できる.また,青丸で提示されている次に打つことができるマスを確認すると,反対側にある白石を裏返すことができると判定されている.

表１は,ランダムに石を打つ AI との対戦結果である.本研究で作成した AI が先手のときと後手のときをそれぞれ 100 回ずつ対戦させた.その結果,先手のときは,約 60%の勝率で勝利し,後手のときは 100%勝利している.このことから,本研究の AI はランダムに打っているものより,強い手を打てていることがわかる.また,先手と後手の勝率に大きな違いがあることにより,後手の方が有利であると考えられる.

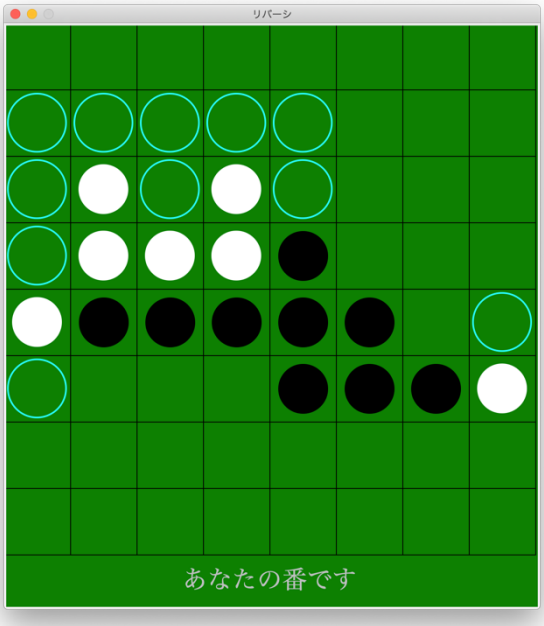

図 8 トーラス型リバーシ実行結果

表 1 対戦戦績

	先手	後手	合計
勝利	62	100	162
敗北	37	0	37
引き分け	1	0	1
勝率(%)	62	100	81

5 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.角や辺のマスを反対側のマスから裏返すことに成功し,対局を最後まで打ち合うことに成功した.また,ランダムに打つ戦略のものとの対戦結果により,本研究の AI は想定通りに思考して手を進めていると考えられる.

しかし,現在のままだと,一見しただけでは反対側のマスはどれだけ裏返すことができるのかわかりにくい状態となっている.そこで,今後の課題として,角や辺のマスに置かれた石を裏返すことができるのか確認しやすくするために,盤面を一回り大きく用意して,対応したマスに石を置けるようにしたいと考えている.

謝辞

本研究をするにあたり,指導してくださった石水隆講師にはとても感謝しています.また,本研究を発表までに完成させられなかったことを申し訳なく思い,ここに謝罪いたします.

今後は,指導していただいたことを活かしながら,本研究の完成に向けて取り組んでいきたいと思います.ありがとうございました.

参考文献

[1] Seal software：リバーシのアルゴリズム C++ & Java 対応，工学社 (2003).

[2] Joel Feinstein：Amenor Wins World 6x6 Championships!, Forty billion noted under the tree (July 1993), pp.6-8, British Othello Federation's newsletter, (1993)

[3] 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

[4] 中村和樹：奇数マスを含む縮小版オセロの完全解析，サレジオ工業高等専門学校 2016 年度卒業研究概要集，p.135 (2017). https://www.salesio-sp.ac.jp/papers/sotsuken/sotsuken2016.pdf

[5] 大筆豊：オセロプログラムの評価関数の改善について，研究報告ゲーム情報学 (GI), Vol.2003-GI-011, pp.15-20，情報処理学会 (2004). http://id.nii.ac.jp/1001/00058554/

[6] 森田悠樹，橋本剛，小林康幸：オセロ求解に向けた単純な縦型探索をベースにする探索方法の研究，ゲームプログラミングワークショップ 2010 論文集，Vol.2010，No.12，pp.36-41，情報処理学会 (2010) http://id.nii.ac.jp/1001/00071311/

[7] 上田陽平，池田心：遺伝的アルゴリズムによる人間のレベルに適応する多様なオセロ AI の生成研究報告ゲーム情報学(GI), Vol.2012-GI-27, No.5, pp.1-8，情報処理学会 (2012). http://id.nii.ac.jp/1001/00080933/

[8] 高木騰也，藤井叙人，片寄晴弘：コンピュータオセロによる自然な手を選択する棋力調整手法の提案，ゲームプログラミングワークショップ 2021 論文集，Vol.2021，pp.9-14，情報処理学会 (2021) http://id.nii.ac.jp/1001/00213315/

[9] Python でリバーシ（オセロ）ゲームを作ろう～動画でも解説あり～，https://blogsmile117.com/python_reverse/

[10] 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.

```
#ライブラリ
import tkinter
import tkinter.messagebox
import random

#定数宣言
FS = ("Times New Roman", 30)
FL = ("Times New Roman", 80)
BLACK = 1
WHITE = 2
mx = 0
my = 0
mc = 0
proc = 0
turn = 0
msg = ""
space = 0
color = [0]*2
who = ["あなた", "コンピュータ"]
board = []
back = []

for y in range(8):
    board.append([0]*8)
    back.append([0]*8)

def click(e):
    global mx, my, mc
    mc = 1
    mx = int(e.x/80)
    my = int(e.y/80)
    if mx>7: mx = 7
    if my>7: my = 7

def banmen():
    cvs.delete("all")
    cvs.create_text(320, 670, text=msg, fill="silver", font=FS)
    for y in range(8):
        for x in range(8):
            X = x*80
            Y = y*80
            cvs.create_rectangle(X, Y, X+80, Y+80, outline="black")
            if board[y][x]==BLACK:
                cvs.create_oval(X+10, Y+10, X+70, Y+70, fill="black", width=0)
```

```
            if board[y][x]==WHITE:
                cvs.create_oval(X+10, Y+10, X+70, Y+70, fill="white", width=0)
            if hitable(x, y, BLACK)>0:
                cvs.create_oval(X+5, Y+5, X+75, Y+75, outline="cyan", width=2)
    cvs.update()

def ban_syokika():
    global space
    space = 60
    for y in range(8):
        for x in range(8):
            board[y][x] = 0

    board[3][4] = BLACK
    board[4][3] = BLACK
    board[3][3] = WHITE
    board[4][4] = WHITE

# 石を打ち、相手の石をひっくり返す

def hitStone(x, y, iro):
    board[y][x] = iro
    for dy in range(-1, 2):
        for dx in range(-1, 2):
            k = 0
            sx = x
            sy = y
            while True:
                sx = (sx + dx) % 8
                sy = (sy + dy) % 8
                if sx<0 or sx>7 or sy<0 or sy>7:
                    break
                if board[sy][sx]==0:
                    break
                if board[sy][sx]==3-iro:
                    k += 1
                if board[sy][sx]==iro:
                    for i in range(k):
                        sx = (sx - dx) % 8
                        sy = (sy - dy) % 8
                        board[sy][sx] = iro
                    break

# そこに打つといくつ返せるか数える

def hitable(x, y, iro):
    if board[y][x]>0:
        return -1 # 置けないマス
```

```
    total = 0
    for dy in range(-1, 2):
        for dx in range(-1, 2):
            k = 0
            sx = x
            sy = y
            while True:
                sx = (sx + dx) % 8
                sy = (sy + dy) % 8
                if sx<0 or sx>7 or sy<0 or sy>7:
                    break
                if board[sy][sx]==0:
                    break
                if board[sy][sx]==3-iro:
                    k += 1
                if board[sy][sx]==iro:
                    total += k
                    break
    return total

# 打てるマスがあるか調べる

def uteru_masu(iro):
    for y in range(8):
        for x in range(8):
            if hitable(x, y, iro)>0:
                return True
    return False

# 黒い石、白い石、いくつかあるか数える

def ishino_kazu():
    b = 0
    w = 0
    for y in range(8):
        for x in range(8):
            if board[y][x]==BLACK: b += 1
            if board[y][x]==WHITE: w += 1
    return b, w

# モンテカルロ法による思考ルーチン

def save():
    for y in range(8):
        for x in range(8):
            back[y][x] = board[y][x]

def load():
```

```
    for y in range(8):
        for x in range(8):
            board[y][x] = back[y][x]

def uchiau(iro):
    while True:
        if uteru_masu(BLACK)==False and uteru_masu(WHITE)==False:
            break
        iro = 3-iro
        if uteru_masu(iro)==True:
            while True:
                x = random.randint(0, 7)
                y = random.randint(0, 7)
                if hitable(x, y, iro)>0:
                    hitStone(x, y, iro)
                    break

def computer_2(iro, loops):
    global msg
    win = [0]*64
    save()
    for y in range(8):
        for x in range(8):
            if hitable(x, y, iro)>0:
                msg += "."
                banmen()
                win[x+y*8] = 1
                for i in range(loops):
                    hitStone(x, y, iro)
                    uchiau(iro)
                    b, w = ishino_kazu()
                    if iro==BLACK and b>w:
                        win[x+y*8] += 1
                    if iro==WHITE and w>b:
                        win[x+y*8] += 1
                    load()
    m = 0
    n = 0

    for i in range(64):
        if win[i]>m:
            m = win[i]
            n = i

    x = n%8
    y = int(n/8)

    return x, y
```

```
def main():
    global mc, proc, turn, msg, space
    banmen()

    if proc==0: # タイトル画面
        msg = "先手、後手を選んでください"
        cvs.create_text(320, 200, text="Reversi", fill="gold", font=FL)
        cvs.create_text(160, 440, text="先手(黒)", fill="lime", font=FS)
        cvs.create_text(480, 440, text="後手(白)", fill="lime", font=FS)

        if mc==1: # ウィンドウをクリック
            mc = 0
            if (mx==1 or mx==2) and my==5:
                ban_syokika()
                color[0] = BLACK
                color[1] = WHITE
                turn = 0
                proc = 1

            if (mx==5 or mx==6) and my==5:
                ban_syokika()
                color[0] = WHITE
                color[1] = BLACK
                turn = 1
                proc = 1

    elif proc==1: # どちらの番か表示
        msg = "あなたの番です"
        if turn==1:
            msg = "コンピュータ 考え中."
        proc = 2
    elif proc==2: # 石を打つマスを決める
        if turn==0: # プレイヤー
            if mc==1:
                mc = 0
                if hitable(mx, my, color[turn])>0:
                    hitStone(mx, my, color[turn])
                    space -= 1
                    proc = 3
        else: # コンピュータ
            MONTE = [300, 300, 240, 180, 120, 60, 1]
            cx, cy = computer_2(color[turn], MONTE[int(space/10)])
            hitStone(cx, cy, color[turn])
            space -= 1
            proc = 3
    elif proc==3: # 打つ番を交代
        msg = ""
```

```
            turn = 1-turn
            proc = 4
        elif proc==4: # 打てるマスがあるか
            if space==0:
                proc = 5
            elif uteru_masu(BLACK)==False and uteru_masu(WHITE)==False:
                tkinter.messagebox.showinfo("", "どちらも打てないので終了です")
                proc = 5
            elif uteru_masu(color[turn])==False:
                tkinter.messagebox.showinfo("", who[turn]+"は打てないのでパスです")
                proc = 3
            else:
                proc = 1
        elif proc==5: # 勝敗判定
            b, w = ishino_kazu()
            tkinter.messagebox.showinfo("終了", "黒={}、白={}".format(b, w))
            if (color[0]==BLACK and b>w) or (color[0]==WHITE and w>b):
                tkinter.messagebox.showinfo("", "あなたの勝ち！")
            elif (color[1]==BLACK and b>w) or (color[1]==WHITE and w>b):
                tkinter.messagebox.showinfo("", "コンピュータの勝ち！")
            else:
                tkinter.messagebox.showinfo("", "引き分け")
            proc = 0
        root.after(100, main)
root = tkinter.Tk()
root.title("リバーシ")
root.resizable(False, False)
root.bind("<Button>", click)

cvs = tkinter.Canvas(width=640, height=700, bg="green")
cvs.pack()
root.after(100, main)
root.mainloop()
```