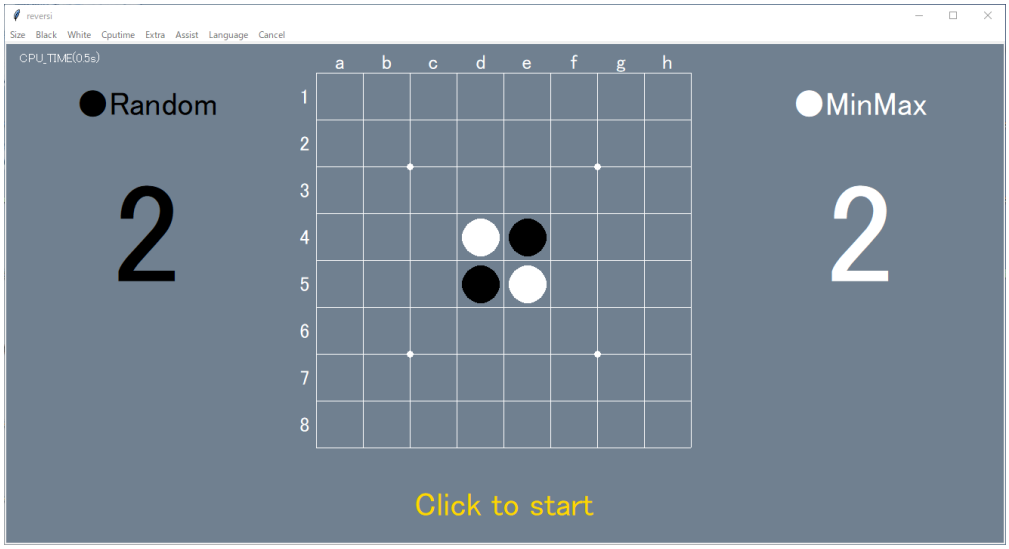


reversi

[English | 日本語]

reversiはリバーシ(オセロ)のPython用ライブラリです。
手軽にリバーシAIをプログラミングして、アプリケーションが作れます。

目次

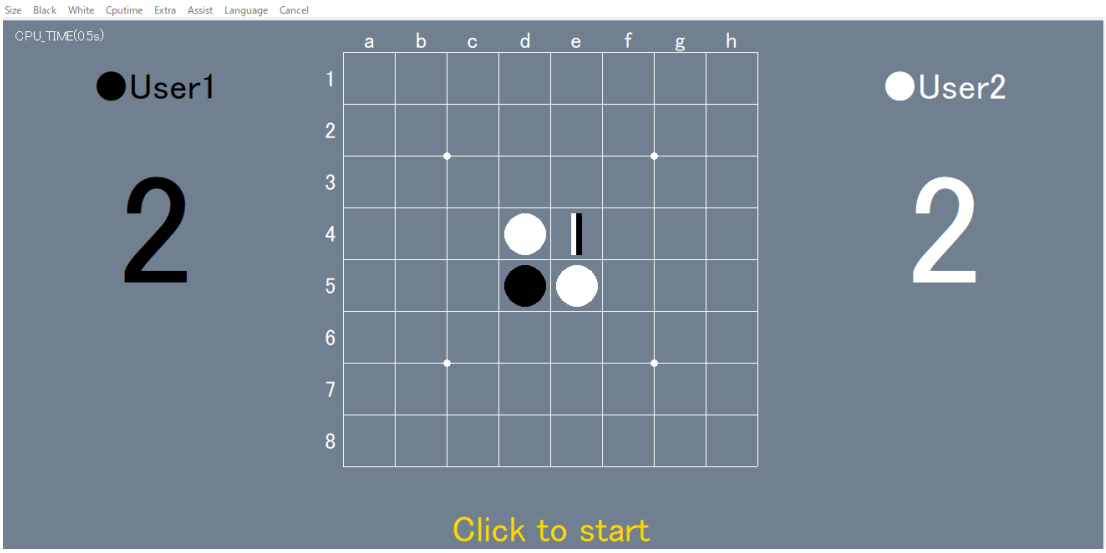
概要

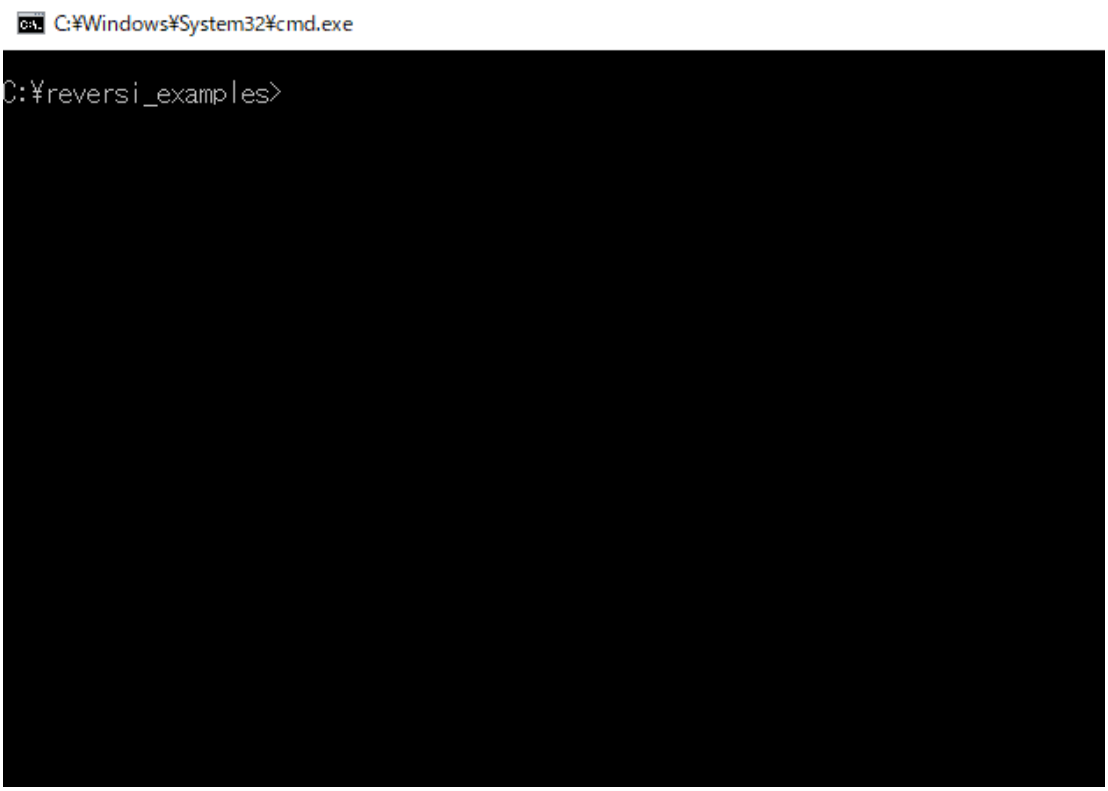
reversiはPythonで作られた[1]Pythonで使えるリバーシのライブラリです。
reversiをインストールすると、リバーシAIのプログラミングを手軽に試せるようになります。

他にも、以下のような用途に使えます。

- アプリケーションを作って、自作したAIと対戦し遊ぶ
- シミュレータを使って、AI同士をたくさん対戦させ強さを調べる

また、本ライブラリを使って作成したWindows版アプリケーションも用意しています。
こちらはダウンロード後、インストール不要で、すぐにリバーシのゲームを無料でプレイできます。





動作環境

- Windows10 64bit
- ディスプレイサイズ 1366x768 以上
- プロセッサ 1.6GHz 以上
- メモリ 4.00GB 以上

- Python 3.7.6以上
  - cython 0.29.15
  - pyinstaller 3.6
- Microsoft Visual C++ 2019(Python3.7.6以外の場合)

インストール方法

1. Python 3.7.6をインストールしてください。
2. 下記を実行して**reversi**をインストールしてください。

```
$ py -3.7 -m pip install git+https://github.com/y-tetsu/reversi
```

アンインストール方法

reversiをアンインストールする場合は、下記を実行してください。

```
$ py -3.7 -m pip uninstall reversi
```

サンプル

reversiをインストール後、任意のフォルダで下記コマンドを実行すると、サンプルをコピーできます。

```
$ install_reversi_examples
```

コピーされるサンプルは下記のとおりです。

- 01_tkinter_app.py - tkinterを使ったGUIアプリケーション(#遊び方)
- 02_console_app.py - コンソール上で遊ぶアプリケーション(#遊び方)
- 03_create_exe.bat - GUIアプリケーションのexeファイルを作成するバッチファイル
- 04_reversi_simulator.py - AI同士を対戦させて結果を表示するシミュレータ
- 05_manual_strategy.py - 自作したAIを実装するサンプル
- 06_table_strategy.py - テーブルによる重みづけで手を選ぶAIを実装するサンプル
- 07_minmax_strategy.py - MinMax法で手を選ぶAIを実装するサンプル
- 08_alphabeta_strategy.py - AlphaBeta法で手を選ぶAIを実装するサンプル
- 09_genetic_algorithm.py - 遺伝的アルゴリズムを使ってテーブルの重みを求めるサンプル
- 10_x_elucidator.py - 変則ボードの解析ツール

サンプルの実行方法はそれぞれ下記のとおりです。

```
$ cd reversi_examples
$ py -3.7 01_tkinter_app.py
$ py -3.7 02_console_app.py
$ 03_create_exe.bat
$ py -3.7 04_reversi_simulator.py
```

```
$ py -3.7 05_manual_strategy.py
$ py -3.7 06_table_strategy.py
$ py -3.7 07_minmax_strategy.py
$ py -3.7 08_alphabeta_strategy.py
$ py -3.7 09_genetic_algorithm.py
$ py -3.7 10_x_elucidator.py
```

ライブラリの使い方

基本編

本ライブラリの基本的な使い方を、コーディング例を元に説明します。

アプリケーションを起動させる

まず最初に、リバーシのGUIアプリケーションを起動させる方法を示します。

下記のコードを実行してください。

```
from reversi import Reversi

Reversi().start()
```

アプリケーションが起動し、そのまま二人対戦で遊ぶ事ができます。
(この場合、選択できるプレイヤーはユーザ操作のみとなります)

アプリケーションにAIを追加する

次に、AIをアプリケーションに追加する方法を示します。

例として、ライブラリにあらかじめ組み込まれている下記AIをアプリケーションに追加します。

- ランダムな手を打つAI : `Random`
- できるだけ多く石が取れる手を打つAI : `Greedy`

```
from reversi import Reversi
from reversi.strategies import Random, Greedy

Reversi(
    {
        'RANDOM': Random(),
        'GREEDY': Greedy(),
    }
).start()
```

上記を実行すると、ユーザ操作に加えて"RANDOM"と"GREEDY"をプレイヤーとして選択できるようになります。

組み込みのAIは、すべてreversi.strategiesよりインポートすることができます。
他に使用可能なAIについての詳細はAIクラス編を参照してください。
また、追加するプレイヤーの情報(以後、"プレイヤー情報")は、下記フォーマットに従ってください。
プレイヤー名については任意に設定可能です。

```
{
    'プレイヤー名1': AIクラスのオブジェクト1,
    'プレイヤー名2': AIクラスのオブジェクト2,
    'プレイヤー名3': AIクラスのオブジェクト3,
}
```

AIをプログラミングする

続いて、本ライブラリを使って独自のAIを自作し、アプリケーションに追加する方法を示します。

AIクラスの作り方

下記のようにコーディングすると、AIクラスが完成します。

```
from reversi.strategies import AbstractStrategy

class OriginalAI(AbstractStrategy):
    def next_move(self, color, board):
        #
        # 次の一手(X, Y)を決めるロジックをコーディングして下さい。
        #

        return (X, Y)
```

next_moveメソッドには特定の手番および盤面の時に、どこに石を打つか(次の一手)を返すロジックを実装します。
next_moveメソッドの引数は下記を参照して下さい。

引数	説明
color変数	blackかwhiteのstr型の文字列が渡され、それぞれ黒番か白番かを判別することができます。
boardオブジェクト	リバーシの盤面情報を持ったオブジェクトが渡されます。黒と白の石の配置情報のほか、石が置ける位置の取得などゲームを進行するために必要となる、パラメータやメソッドを持っています。

なお、戻り値の(X, Y)座標は盤面左上を(0, 0)とした時の値となります。
盤面サイズが8の場合の各マス目の座標を下図に示します。

Size = 8

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)	(6, 0)	(7, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)	(6, 1)	(7, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)	(6, 2)	(7, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)	(6, 3)	(7, 3)
(0, 4)	(1, 4)	(2, 4)	(3, 4)	(4, 4)	(5, 4)	(6, 4)	(7, 4)
(0, 5)	(1, 5)	(2, 5)	(3, 5)	(4, 5)	(5, 5)	(6, 5)	(7, 5)
(0, 6)	(1, 6)	(2, 6)	(3, 6)	(4, 6)	(5, 6)	(6, 6)	(7, 6)
(0, 7)	(1, 7)	(2, 7)	(3, 7)	(4, 7)	(5, 7)	(6, 7)	(7, 7)

`board`オブジェクトについてはここでは簡単のため、 石が置ける位置を取得する`get_legal_moves`メソッドと、
盤面のサイズを取得する`size`パラメータの、2つを取り上げます。
より詳しい説明はboardオブジェクトの使い方を参照してください。

石が置ける位置の取得方法

ある盤面の石が置ける位置(座標)は`board`オブジェクトの`get_legal_moves`メソッドで取得できます。
`get_legal_moves`呼び出し時の引数には、黒か白のどちらかの手番(`color`変数)を与えてください。

```
legal_moves = board.get_legal_moves(color)
```

`get_legal_moves`の戻り値"石が置ける座標のリスト"となっています。

初期状態(盤面サイズ8)での黒手番の結果は下記のとおりです。

```
[(3, 2), (2, 3), (5, 4), (4, 5)]
```

盤面のサイズ

本アプリケーションは、盤面のサイズとして4～26までの偶数が選べる仕様となっております。 必要に応じて、いずれの場合でも動作するよう盤面のサイズを考慮するようにしてください。

盤面のサイズは下記で取得できます。

```
size = board.size
```

「角が取れる時は必ず取る」AIの実装

それでは、AIの作成例として4角が取れる時は必ず取り、 そうでない時はランダムに打つ、`Corner`というAIを実装する例を示します(プレイヤー名は"CORNER"とします)。

```
import random

from reversi import Reversi
from reversi.strategies import AbstractStrategy

class Corner(AbstractStrategy):
    def next_move(self, color, board):
        size = board.size
        legal_moves = board.get_legal_moves(color)
        for corner in [(0, 0), (0, size-1), (size-1, 0), (size-1, size-1)]:
            if corner in legal_moves:
                return corner

        return random.choice(legal_moves)

Reversi({'CORNER': Corner()}).start()
```

上記を実行すると、対戦プレイヤーに"CORNER"が選択可能となります。
実際に対戦してみると、角が取れる時に必ず取ってくることが分かると思います。

AI同士の対戦をシミュレートする

本ライブラリのシミュレータを使うと、AI同士を複数回対戦させて勝率を出すことができます。
自作したAIの強さを測るために活用してください。
また、AIの打つ手が特定の盤面に対して固定となる場合は、別途後述のrandom_openingパラメータを設定することで、結果の偏りを減らせます。

シミュレータの実行例として、 これまでに登場した"RANDOM"、"GREEDY"、"CORNER"を総当たりで対戦させ、結果を表示するまでを示します。

シミュレータの実行

下記を実行すると、シミュレーションを開始します。

```
from reversi import Simulator

if __name__ == '__main__':
    simulator = Simulator(
        {
            'RANDOM': Random(),
```

```
            'GREEDY': Greedy(),
            'CORNER': Corner(),
        },
        './simulator_setting.json',
    )
    simulator.start()
```

シミュレータは必ずメインモジュール(__main__)内で実行するようにしてください。
シミュレータの引数には、"プレイヤー情報"と"シミュレータの設定ファイル"を指定してください。

シミュレータの設定ファイル

シミュレータの設定ファイル(JSON形式)の作成例は下記のとおりです。 Simulatorの第二引数に、本ファイル名(上記例では`./simulator_setting.json`ですが任意)を指定してください。

```
{
    "board_size": 8,
    "board_type": "bitboard",
    "matches": 100,
    "processes": 1,
    "parallel": "player",
    "random_opening": 0,
    "player_names": [
        "RANDOM",
        "GREEDY",
        "CORNER"
    ]
}
```

パラメータ名	説明
board_size	盤面のサイズを指定してください。
board_type	盤面の種類(board または bitboard)を選択してください。bitboardの方が高速で通常はこちらを使用してください。
matches	AI同士の対戦回数を指定してください。100を指定した場合、AIの各組み合わせにつき先手と後手で100試合ずつ対戦する動作となります。
processes	並列実行数を指定してください。お使いのPCのコア数に合わせて、設定を大きくするほど、シミュレーション結果が早く得られる場合があります。
parallel	並列実行する単位を指定してください。"player"(デフォルト)を指定した場合、AI対戦の組み合わせ毎に並列処理を実施します。また、"game"を指定した場合は、matchesの対戦回数をprocessesの数で分割して並列処理を実施します。シミュレートするAI対戦の組み合わせの数が、お使いのPCのコア数より少ない場合は、"game"を指定することで、より早く結果を得られる場合があります。

パラメータ名	説明
random_opening	対戦開始から指定した手数までは、AI同士ランダムな手を打ち試合を進行します。指定された手数を超えるとAIは本来の手を打ちます。対戦開始の状況をランダムに振ることで、結果の偏りを減らしAIの強さを測りやすくします。不要な場合は0を指定してください。
player_names	対戦させたいAI名をリストアップして下さい。指定する場合は第一引数の"プレイヤー情報"に含まれるものの中から選択してください。省略すると第一引数の"プレイヤー情報"と同一と扱います。リストアップされた全てのAI同士の総当たり戦を行います。

実行結果

シミュレーション結果はシミュレーション実行後(startメソッド実行後)、下記のようにprint表示で確認できます。

```
print(simulator)
```

実行例

ライブラリ組み込みのAIを用いたプレイヤー"RANDOM"、"GREEDY"と、 自作のAIによる"CORNER"の、それぞれを対戦させるようシミュレータを実行して、 結果を出力するまでのコード例を下記に示します。

```
import random

from reversi import Simulator
from reversi.strategies import AbstractStrategy, Random, Greedy

class Corner(AbstractStrategy):
    def next_move(self, color, board):
        size = board.size
        legal_moves = board.get_legal_moves(color)
        for corner in [(0, 0), (0, size-1), (size-1, 0), (size-1, size-1)]:
            if corner in legal_moves:
                return corner

        return random.choice(legal_moves)

if __name__ == '__main__':
    simulator = Simulator(
        {
            'RANDOM': Random(),
            'GREEDY': Greedy(),
            'CORNER': Corner(),
        },
        './simulator_setting.json',
    )
```

```
    simulator.start()

    print(simulator)
```

"RANDOM"、"GREEDY"、"CORNER"を総当たりで、先手/後手それぞれ100回ずつ対戦したところ、下記の結果になりました。

```
Size : 8
                        | RANDOM                    GREEDY
CORNER
-------------------------------------------------------------------------------
-----------------------
RANDOM                  | ------                     32.5%
21.5%
GREEDY                  |  66.0%                    ------
29.5%
CORNER                  |  76.0%                     68.5%                    --
----
-------------------------------------------------------------------------------
-----------------------

                        | Total   | Win   Lose  Draw  Match
-------------------------------------------------------------
RANDOM                  |  27.0% |   108   284     8   400
GREEDY                  |  47.8% |   191   202     7   400
CORNER                  |  72.2% |   289   102     9   400
-------------------------------------------------------------
```

ランダムに打つよりも毎回多めに取る方が、さらにそれよりも角は必ず取る方が、より有利になりそうだという結果が得られました。

対戦結果を1試合ごとにAIへ通知する

AIに以下のget_resultメソッドを実装することで、シミュレータ実行時に1試合ごとの対戦結果を、AIに渡し何らか処理させることができます。

```
from reversi.strategies import AbstractStrategy

class OriginalAI(AbstractStrategy):
    def next_move(self, color, board):
        #
        # 次の一手(X, Y)を決めるロジックをコーディングして下さい。
        #

        return (X, Y)

    def get_result(self, result):
        #
        # 1試合終わるごとにSimulatorからコールされます。
```

```
        #
        # (resultには以下の情報が格納されています)
        # result.winlose : 対戦結果(0=黒の勝ち、1=白の勝ち、2=引き分け)
        # result.black_name : 黒のAIの名前
        # result.white_name : 白のAIの名前
        # result.black_num : 黒の石の数
        # result.white_num : 白の石の数
        #
```

オブジェクト編

本ライブラリに用意されている各種オブジェクトについて説明します。

boardオブジェクトの使い方

ここでは、リバーシの盤面を管理するboardオブジェクトの使い方について説明します。

boardオブジェクトの生成

boardオブジェクトは**reversi**より、Boardクラスまたは BitBoardクラスを インポートすることで、生成できるようになります。

BoardクラスとBitBoardクラスの違いは、盤面を表現する内部データの構造のみで、使い方は同じです。
BitBoardクラスの方が処理速度がより高速なため、通常はこちらをご使用下さい。

Boardクラスをインスタンス化する際の引数に、数値を入れることで盤面のサイズを指定できます。 サイズは4～26までの偶数としてください。省略時は8となります。 また、sizeプロパティにて盤面のサイズを確認することができます。

コーディング例は下記のとおりです。

```
from reversi import Board, BitBoard

board = Board()
print(board.size)

bitboard = BitBoard(10)
print(bitboard.size)
```

上記の実行結果は下記となります。

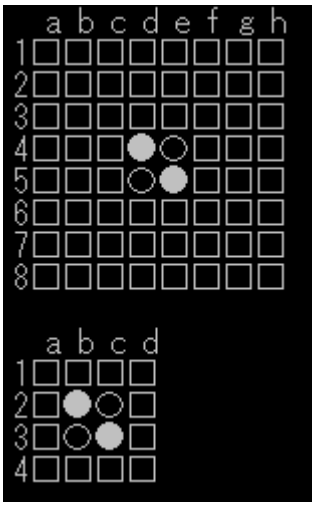
```
8
10
```

boardオブジェクトの標準出力

boardオブジェクトをprintすると盤面の状態が標準出力されます。

```
from reversi import BitBoard

board = BitBoard()
print(board)

board = BitBoard(4)
print(board)
```

上記の実行結果は下記となります。



boardオブジェクトのメソッド

`board`オブジェクトの使用可能なメソッドを紹介します。

get_legal_moves

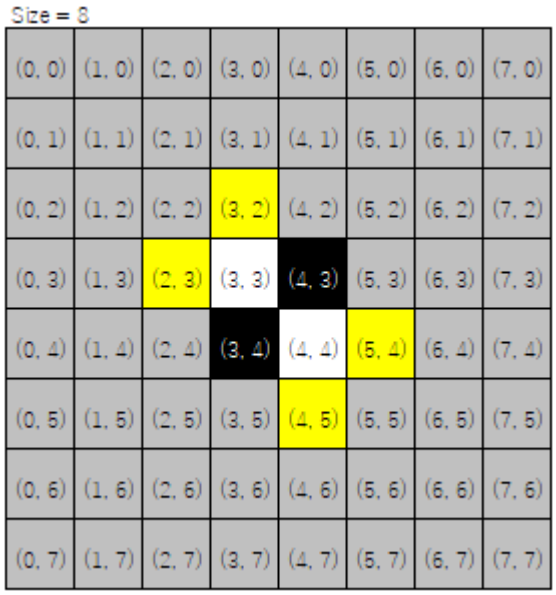
黒番または白番での着手可能な位置を返します。 着手可能な位置は"XY座標のタプルのリスト"となります。
引数には`black`(黒番)または`white`(白番)の文字列(以後`color`と呼びます)を指定してください。

```
from reversi import BitBoard

board = BitBoard()
legal_moves = board.get_legal_moves('black')

print(legal_moves)
```

上記の実行結果は下記となります。

```
[(3, 2), (2, 3), (5, 4), (4, 5)]
```

この場合、下図の黄色のマスの位置が、着手可能な位置として返されます。



get_flippable_discs

指定位置に着手した場合の、ひっくり返せる石を返します。 ひっくり返せる石は"XY座標のタプルのリスト"となります。 第一引数に`color`、第二引数に石を置くX座標、第三引数にY座標を指定してください。

```
from reversi import BitBoard

board = BitBoard()
flippable_discs = board.get_flippable_discs('black', 5, 4)

print(flippable_discs)
```

上記の実行結果は下記となります。

```
[(4, 4)]
```

この場合、下図の黄色のマスの位置が、ひっくり返せる石の位置として返されます。

Size = 8

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)	(6, 0)	(7, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)	(6, 1)	(7, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)	(6, 2)	(7, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)	(6, 3)	(7, 3)
(0, 4)	(1, 4)	(2, 4)	(3, 4)	(4, 4)	(5, 4)	(6, 4)	(7, 4)
(0, 5)	(1, 5)	(2, 5)	(3, 5)	(4, 5)	(5, 5)	(6, 5)	(7, 5)
(0, 6)	(1, 6)	(2, 6)	(3, 6)	(4, 6)	(5, 6)	(6, 6)	(7, 6)
(0, 7)	(1, 7)	(2, 7)	(3, 7)	(4, 7)	(5, 7)	(6, 7)	(7, 7)

get_board_info

盤面に置かれた石の状態を"2次元リスト"で返します。 "1"が黒、"-1"が白、"0"が空きを表します。引数はありません。

```
from pprint import pprint
from reversi import BitBoard

board = BitBoard()
board_info = board.get_board_info()

print(board)
pprint(board_info)
```

上記の実行結果は下記となります。

```
  a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□●○□□□
5□□□○●□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□

[[0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, -1, 1, 0, 0, 0],
 [0, 0, 0, 1, -1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

get_board_line_info

盤面の情報を1行の文字列で返します。

```
from pprint import pprint
from reversi import BitBoard

board = BitBoard()

print(board.get_board_line_info('black'))
```

上記の表示結果は以下です。

```
---------------------------O*------*O---------------------------*
```

盤面のマス目情報 + プレイヤー情報の形式となっています。

引数にはプレイヤーを示す文字列('black'または'white')を指定してください。

デフォルトの文字の割り当ては以下の通りです

- "*" : 黒プレイヤー
- "O" : 白プレイヤー
- "-" : 空きマス

オプション引数の指定で、お好みの文字に変更可能です。

```
print(board.get_board_line_info(player='black', black='0', white='1', empty='.'))
```

上記実行時は、以下の出力になります。

```
.........................10......01.........................0
```

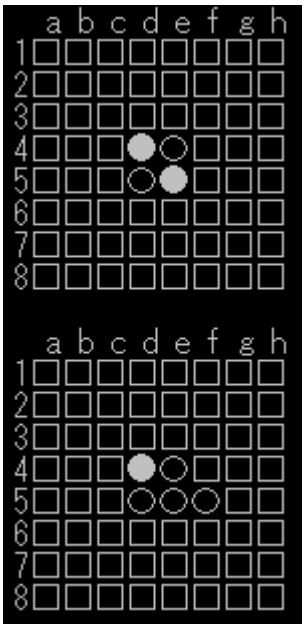
- player : 'black'(黒)か'white'(白)を指定してください。
- black : 黒に割り当てる文字を指定してください
- white : 白に割り当てる文字を指定してください
- empty : 空きマスに割り当てる文字を指定してください

put_disc

指定位置に石を配置し、取れる石をひっくり返します。 第一引数に`color`、第二引数に石を置くX座標、第三引数にY座標を指定してください。

```
from reversi import BitBoard

board = BitBoard()
print(board)

board.put_disc('black', 5, 4)
print(board)
```

上記の実行結果は下記となります。

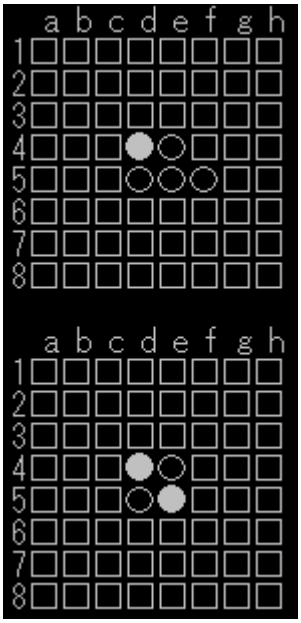


undo

`put_disc`メソッドで置いた石を元に戻します。引数はありません。 `put_disc`メソッドを呼び出した回数だけ、元に戻すことができます。 `put_disc`メソッドを呼び出した回数を超えて、本メソッドを呼び出さないでください。

```
from reversi import BitBoard

board = BitBoard()
board.put_disc('black', 5, 4)
print(board)

board.undo()
print(board)
```

上記の実行結果は下記となります。



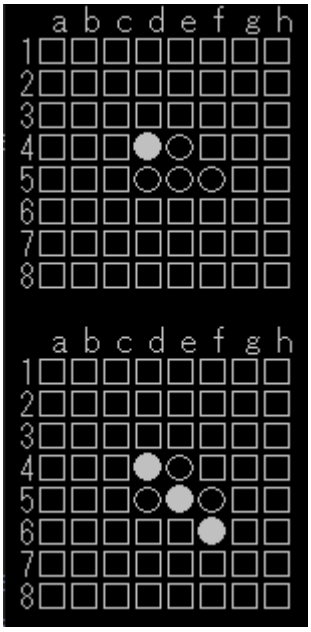
colorオブジェクトの使い方

これまでは、黒番や白番の手番の判別に'black'や'white'の文字列を使う方法を示しましたが
`color`オブジェクトを用いて指定することも可能です。

以下のように`color`オブジェクトである`C`をインポートして、
blackプロパティやwhiteプロパティにてそれぞれ黒番、白番を指定できます。

```
from reversi import BitBoard
from reversi import C as c

board = BitBoard()
board.put_disc(c.black, 5, 4)
print(board)

board.put_disc(c.white, 5, 5)
print(board)
```

上記の実行結果は下記となります。



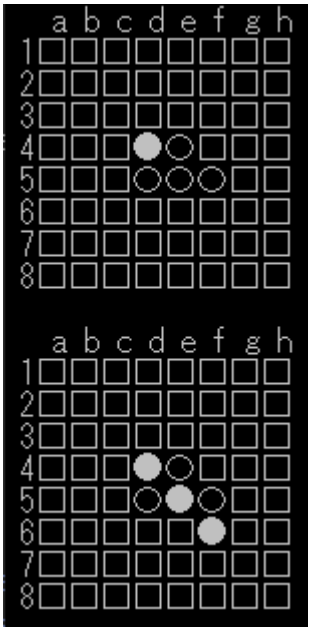
moveオブジェクトの使い方

`move`オブジェクトを使うと、手を打つ座標の指定に、これまでのXY座標形式だけではなく、'a1'・'c3'などのstr形式が使えるようになります。
str形式のアルファベットは大文字・小文字どちらでもOKです。

`Move`クラスをインポートして、下記のように使用してください。

```
from reversi import BitBoard
from reversi import C as c
from reversi import Move as m

board = BitBoard()
board.put_disc(c.black, *m('f5'))
print(board)

board.put_disc(c.white, *m('f6'))
print(board)
```

上記の実行結果は下記となります。



また、moveオブジェクトはXY座標形式でも生成でき、str関数を使ってstr形式へ変換できます。
moveオブジェクトをprint表示した場合も同様にstr形式となります。
caseオプションにupperを指定すると大文字表記になります。

```
from reversi import BitBoard
from reversi import Move as m

move = str(m(5, 4))
print(move)
print(m(5, 5, case='upper'))
```

上記の実行結果は下記となります。

```
f5
F6
```

AIクラス編

本ライブラリに用意されている各種AIクラスについて説明します。

単純思考なAI

Unselfish

取れる石が最も少ない手を選びます。

(使用例)

```
from reversi import Reversi
from reversi.strategies import Unselfish
Reversi({"UNSELFISH": Unselfish()}).start()
```

Random

毎回ランダムな手を選びます。

(使用例)

```
from reversi import Reversi
from reversi.strategies import Random
Reversi({"RANDOM": Random()}).start()
```

Greedy

取れる石が最も多い手を選びます。

(使用例)

```
from reversi import Reversi
from reversi.strategies import Greedy
Reversi({"GREEDY": Greedy()}).start()
```

SlowStarter

序盤(盤面に置かれている石が15%未満の場合)は、取れる石が最も少ない手を選び、
以降は取れる石が最も多い手を選びます。

(使用例)

```
from reversi import Reversi
from reversi.strategies import SlowStarter
Reversi({"SLOWSTARTER": SlowStarter()}).start()
```

マス目の位置に応じて手を選ぶAI

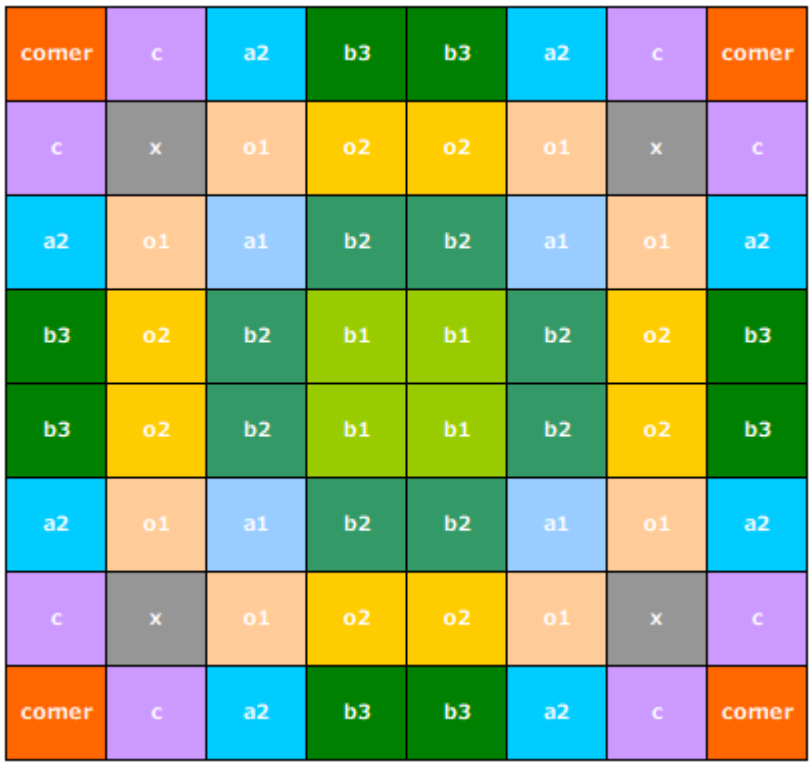
Table

盤面のマス目の位置に重み(重要度)を設定して、
その重みを元に盤面を評価し、スコアが最も高くなる手を選びます。
マス目の重みは使用例の通り、パラメータにて自由に設定が可能です。
ただし、パラメータの値には実数値ではなく整数値(負の値も可)を設定してください。

(使用例)

```
from reversi import Reversi
from reversi.strategies import Table
Reversi(
    {
        'TABLE': Table(
            corner=100,
            c=30,
            a1=50,
            a2=50,
            b1=50,
            b2=50,
            b3=50,
            x=-25,
            o1=45,
            o2=45,
        ),
    }
).start()
```

(パラメータ)
各パラメータに対応するマス目の位置は下図のとおりです。



リバーシでは角を取ると有利になりやすく、Xを取ると不利になりやすいと言われています。
そこで、`corner`パラメータを大きくして角を狙い、`x`パラメータを小さくしてXを避ける、といった調整が可能です。

ランダムに複数回打ってみて勝率の良い手を選ぶAI

MonteCarlo

モンテカルロ法で手を選びます。
打てる手の候補それぞれについて、ゲーム終了までランダムに打つ事を複数回繰り返し
最も勝率が良かった手を選びます。(持ち時間は1手0.5秒)
ゲーム終了までランダムに打つ回数と、モンテカルロ法を開始する残り手数をパラメータで指定可能です。

(使用例)

```
from reversi import Reversi
from reversi.strategies import MonteCarlo
Reversi(
    {
        'MONTECARLO': MonteCarlo(
            count=100,  # ランダムにゲーム終了まで打つ回数(デフォルト:100)
            remain=60,  # モンテカルロ法を開始する残り手数(デフォルト:60)
        ),
    }
).start()
```

数手先の盤面を読んで手を選ぶAI

盤面の形勢を判断する評価関数を元に、相手も自分同様に最善を尽くすと仮定して、数手先の盤面を読み
その中で評価が最も良くなる手を選びます。
評価関数のカスタマイズ方法はこちらを参照して下さい。

MinMax

MinMax法で手を選びます。読む手数を大きくしすぎると処理が終わらない場合がありますのでご注意下さい。

(使用例)

```
from reversi import Reversi
from reversi.strategies import MinMax
from reversi.strategies.coordinator import Evaluator
Reversi(
    {
        'MINMAX': MinMax(
            depth=2,                # 何手先まで読むかを指定
            evaluator=Evaluator(),  # 評価関数を指定(カスタマイズ方法は後述)
        ),
    }
).start()
```

NegaMax

NegaMax法で手を選びます。MinMax法と性能は同じです。
一手0.5秒の持ち時間の中で手を読みます。

(使用例)

```
from reversi import Reversi
from reversi.strategies import NegaMax
from reversi.strategies.coordinator import Evaluator
Reversi(
    {
        'NEGAMAX': NegaMax(
            depth=2,                # 何手先まで読むかを指定
            evaluator=Evaluator(),  # 評価関数を指定(カスタマイズ方法は後述)
        ),
    }
).start()
```

※持ち時間制限を外したい場合は、`NegaMax`クラスの代わりに`_NegaMax`クラスを使用してください。

AlphaBeta

AlphaBeta法で手を選びます。不要な手(悪手)の読みを枝刈りする(打ち切る)ことでMinMax法より効率よく手を読みます。
一手0.5秒の持ち時間の中で手を読みます。

(使用例)

```
from reversi import Reversi
from reversi.strategies import AlphaBeta
from reversi.strategies.coordinator import Evaluator
Reversi(
    {
        'ALPHABETA': AlphaBeta(
            depth=2,                # 何手先まで読むかを指定
            evaluator=Evaluator(),  # 評価関数を指定(カスタマイズ方法は後述)
        ),
    }
).start()
```

※持ち時間制限を外したい場合は、`AlphaBeta`クラスの代わりに`_AlphaBeta`クラスを使用してください。

NegaScout

NegaScout法で手を選びます。AlphaBeta法の枝刈りに加えて自身の着手可能数がより多くなる手を優先的に読むよう設定しています。
一手0.5秒の持ち時間の中で手を読みます。

(使用例)

```
from reversi import Reversi
from reversi.strategies import NegaScout
from reversi.strategies.coordinator import Evaluator
Reversi(
    {
        'NEGASCOUT': NegaScout(
            depth=2,                # 何手先まで読むかを指定
            evaluator=Evaluator(),  # 評価関数を指定(カスタマイズ方法は後述)
        ),
    }
).start()
```

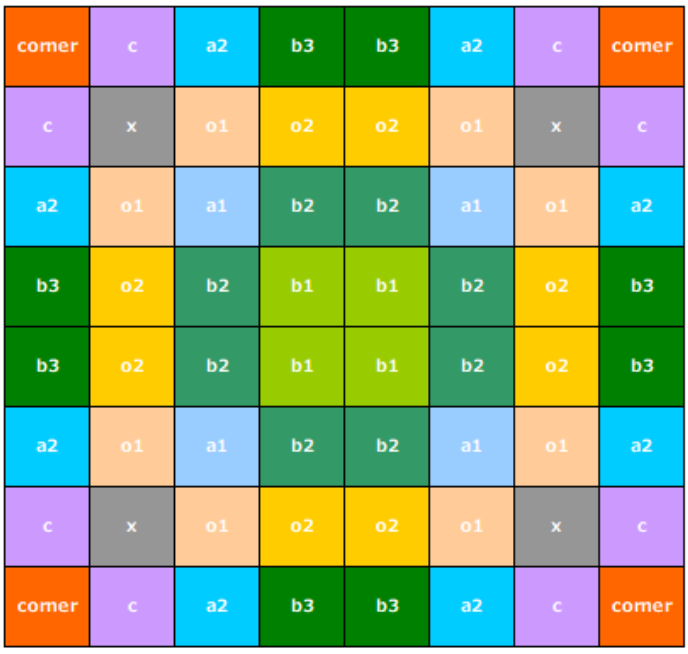
※持ち時間制限を外したい場合は、NegaScoutクラスの代わりに_NegaScoutクラスを使用してください。

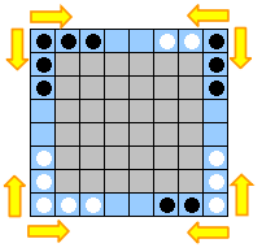
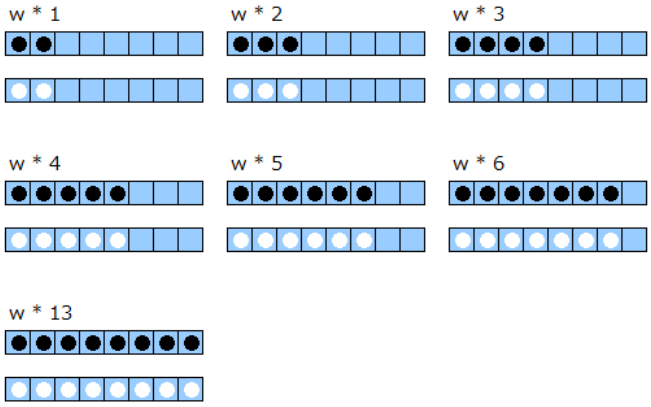
評価関数のカスタマイズ方法

評価関数のカスタマイズにはEvaluatorクラスを使用します。
引数にはseparatedとcombined(それぞれリスト)が指定できます。
各パラメータの要素には盤面のスコアを算出するScorerクラスのオブジェクトを指定する必要があります。

引数	説明
separated	Scorerのスコア算出結果が存在する場合はその結果を評価値とします。リストの先頭の方にあるほど優先度が高くなります。
combined	リストのすべてのScorerのスコア算出結果の和を評価値とします。

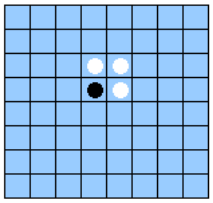
使用可能なScorerクラスは以下になります。

Scorerクラス	説明	パラメータ
TableScorer	盤面の位置に応じた重みづけでスコアを算出します。 	size=盤面のサイズ corner, c, a1, a2, b1, b2, b3, x, o1, o2=マス目の位置に応じて手を選ぶAIと同じ デフォルト:size=8, corner=50, c=-20, a1=0, a2=-1, b1=-1, b2=-1, b3=-1, x=-25, o1=-5, o2=-5

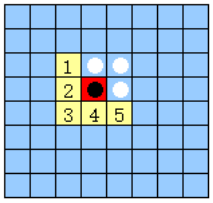
Scorerクラス	説明	パラメータ
PossibilityScorer	着手可能数に基づいてスコアを算出します。	w=(自分の着手可能数 - 相手の着手可能数)の重み デフォルト:5
WinLoseScorer	勝敗に基づいてスコアを算出します。勝敗が決まっていない場合はスコアなしとします。	w=自分が勝ちの場合のスコア デフォルト:10000
NumberScorer	石差(自分の石数 - 相手の石数)をスコアとして算出します。	パラメータなし
EdgeScorer	辺の確定石の数に基づいてスコアを算出します。下図の4隅から8方向を探索し、同じ石が連続する数に応じてスコアを決定します。相手のスコアは合計値からマイナスされます。  	w=確定石一つあたりの重み デフォルト:100
BlankScorer	石が空きマスに接するパターンに基づいてスコアを算出します。算出するスコアは以下の3種類。 1. 置かれた石の周囲8方向の空きマスに接する数	w1=左記1 w2=左記2 w3=左記3 デフォルト:w1から順に-1、-4、-2

Scorerクラス	説明	パラメータ

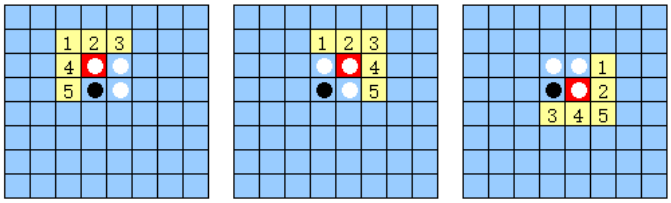
算出例



● … w1 * 5



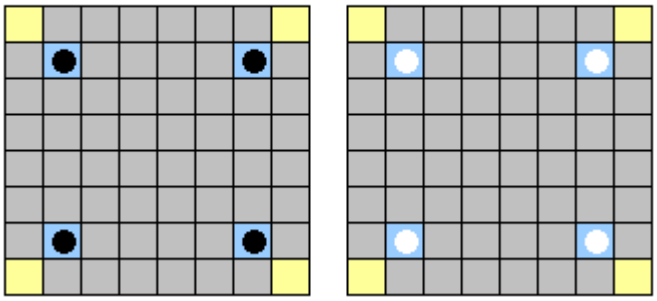
○ … w1 * 15



●手番のスコア = (w1 * 5) - (w1 * 15) = w1 * (-10)

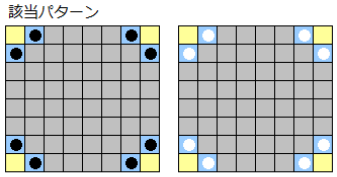
2. 空いた隅に接するX打ち

該当パターン（いずれもw2 * 1）

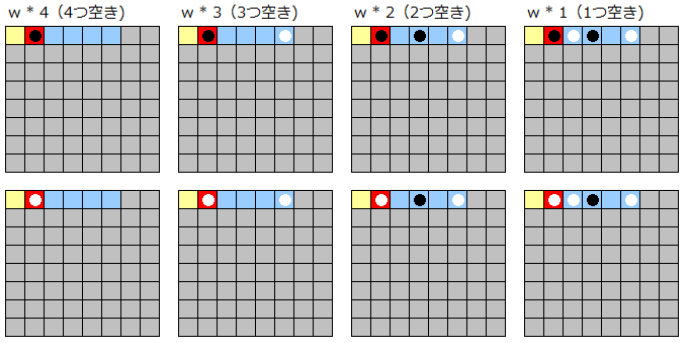


3. 空いた隅に接するC打ち時の辺の空きマスの数

該当パターン



w * 4（4つ空き） w * 3（3つ空き） w * 2（2つ空き） w * 1（1つ空き）



※上記3種類ともに相手のスコアはマイナスします。

(使用例)
以下に、評価関数をカスタマイズする例を示します。

- 勝ちが見える手は優先的に選ぶ
- 勝敗が確定していない場合は以下の指標をすべて加味して手を選ぶ
    - マス目の位置に応じた重みで形勢を判断し、評価値が大きくなるよう手を選ぶ
    - 自身の着手可能数が相手よりも多くなるように手を選ぶ

```
from reversi import Reversi
from reversi.strategies import AlphaBeta
from reversi.strategies.coordinator import Evaluator, TableScorer,
PossibilityScorer, WinLoseScorer
Reversi(
    {
        'CUSTOMIZED': AlphaBeta(
            depth=4,
            evaluator=Evaluator(
                separated=[
                    WinLoseScorer(
                        w=10000,
                    ),
                ],
                combined=[
                    TableScorer(
                        corner=50,
                        c=-20,
                        a1=0,
                        a2=-1,
                        b1=-1,
                        b2=-1,
                        b3=-1,
                        x=-25,
                        o1=-5,
                        o2=-5,
                    ),
                    PossibilityScorer(
                        w=5,
                    ),
                ],
            ),
        ),
    }
).start()
```

評価関数の自作方法

Evaluatorクラスを自作することで、より自由度の高い評価関数を用意することもできます。
以下に、評価関数を自作したAIを作るためのひな形を示します。

(前提)

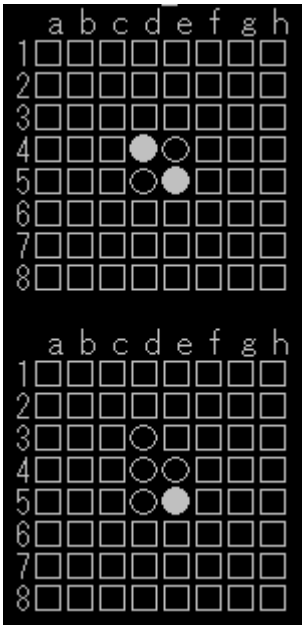
- 探索方法にはAlphaBeta法を用いる(数手先の盤面を読んで手を選ぶAIならいずれでも可)
- 6手先まで読み、探索時間の制限はなしとする
- 評価関数には自作した`MyEvaluator`を用いる
- 評価関数のスコアは高いほど、自身の形勢が良いことを示し、もっともスコアの高い手が選ばれる

(処理内容)

- 初期盤面を表示する
- 自作したAIに、黒の手番での次の一手を求めさせ、盤面に打つ
- 盤面を表示する

```
from reversi import BitBoard
from reversi import C as c
from reversi.strategies.common import AbstractEvaluator
from reversi.strategies import _AlphaBeta

class MyEvaluator(AbstractEvaluator):
    def evaluate(self, color, board, possibility_b, possibility_w):
        score = 0
        #
        # 現在の盤面のスコア(評価値)を算出するロジックをコーディングして下さい。
        #
        return score

my_ai = _AlphaBeta(depth=6, evaluator=MyEvaluator())
board = BitBoard()
print(board)
x, y = my_ai.next_move(c.black, board)
board.put_disc(c.black, x, y)
print(board)
```

上記の実行結果は下記となります。



なお、Evaluatorクラスのevaluate関数の引数には以下が渡されます。
必要に応じて使用してください。

引数	説明
color変数	blackかwhiteのstr型の文字列が渡され、それぞれ黒番か白番かを判別することができます。
boardオブジェクト	リバーシの盤面情報を持ったオブジェクトが渡されます。黒と白の石の配置情報のほか、石が置ける位置の取得などゲームを進行するために必要となる、パラメータやメソッドを持っています。
possibilitiy_b変数	黒番の着手可能数が格納されています。
possibilitiy_w変数	白番の着手可能数が格納されています。

序盤の定石打ちを追加する方法

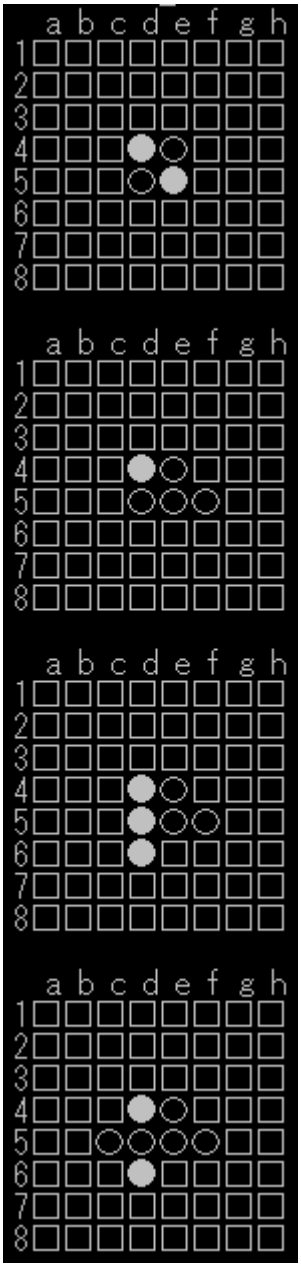
Josekiクラスを活用すると、AIに序盤は定石どおりに手を選ばせることができます。

以下に、自作したAIに定石打ちを追加する例を示します。
(前提)

- 自作したAI(MyAI)に兎進行の定石打ちを追加する

```
import random

from reversi import BitBoard
from reversi import C as c
from reversi.strategies import AbstractStrategy, Usagi

class MyAI(AbstractStrategy):
    """自作AI(ランダムに打つ)"""
    def next_move(self, color, board):
        legal_moves = board.get_legal_moves(color)
        return random.choice(legal_moves)

my_ai = Usagi(base=MyAI())  # base引数に自作AIを与える
board = BitBoard()
print(board)
for color in [c.black, c.white, c.black]:  # 3手進める
    x, y = my_ai.next_move(color, board)
    board.put_disc(color, x, y)
    print(board)
```

上記の実行結果は下記となります。



使用可能な`Joseki`クラスの一覧は以下になります。

Josekiクラス	説明
Usagi	兎進行を選びます。
Tora	虎進行を選びます。
Ushi	牛進行を選びます。
Nezumi	鼠進行を選びます。
Neko	猫進行を選びます。
Hitsuji	羊進行を選びます。

上記いずれにも同じ定石が搭載されており、それぞれの進行を外れても打てる定石に差異はありません。

手の進行に応じてAIを切り替える方法

Switch

`Switch`クラスを活用すると、現在の手数に応じて、AIを切り替えることができます。

以下に、使い方の例を示します。
(前提)

- 盤面のサイズは8x8
- 1～30手目までは`Random`
- 31～50手目までは`Table`
- 51～60手目までは`MonteCarlo`

```
from reversi import Reversi
from reversi.strategies import Random, Table, MonteCarlo, Switch

Reversi(
    {
        'SWITCH': Switch(      # 戦略切り替え
            turns=[
                29,            # 1～30手目まではRandom              (30手目-1を設定)
                49,            # 21～50手目まではTable              (50手目-1を設定)
                60             # それ以降(残り10手)からはMobteCarlo (最後は60を設定)
            ],
            strategies=[
                Random(),      # Random AI
                Table(),       # Table AI
                MonteCarlo(),  # MonteCarlo AI
            ],
        ),
    }
).start()
```

上記の例のように、序盤は適当に打ちハンデを与え、
中盤以降から徐々に強くするといった、ゲーム性を調整する場合や
序盤、中盤、終盤それぞれで最適な戦略を切り替えて、
より強いAIを作成する場合などにも活用することができます。

終盤に完全読みを追加する方法

完全読みとは、互いに最善を尽くす前提でゲーム終了まで打ち
最終の石数の差が、より自身にとって多くなる手を選ぶという方法です。
決着まで読み切るため、手数によっては大きく時間がかかる場合もありますが、
手を読んだ時点で勝てる手が残っていれば、必ず相手に勝つことができます。

EndGame

探索手法にAlphaBeta法を用いて完全読みを行います。
処理時間の目安として、残り14手以下の盤面の手を概ね0.5秒以内に読みます。

ただしこれは努力目標であり、盤面によっては著しく時間がかかる場合がございます。
`EndGame`クラスは制限時間あり、`_EndGame`クラスは制限時間なしとなります。

以下に、自作したAIに終盤の完全読みを追加する例を示します。

(前提)

- 盤面のサイズは8x8
- 残り10手から完全読みを開始する

(使用例)

```
import random

from reversi import Reversi
from reversi.strategies import Switch, _EndGame

class MyAI(AbstractStrategy):
    """自作AI(ランダムに打つ)"""
    def next_move(self, color, board):
        legal_moves = board.get_legal_moves(color)
        return random.choice(legal_moves)

Reversi(
    {
        'ENDGAME': Switch(    # 戦略切り替え
            turns=[
                49,           # 残り11(= 60 - 49)手まではMyAI()
                60            # それ以降(残り10手)からは完全読み
            ],
            strategies=[
                MyAI(),       # 自作AI
                _EndGame(),   # 完全読み(制限時間なし)
            ],
        ),
    }
).start()
```

---

tkinterアプリケーションの遊び方

ゲーム紹介

盤面のサイズや対戦プレイヤーをいろいろ選べるリバーシです。 難易度の異なる多種多様なAIがお相手いたします。
おまけ要素として、お好きなプログラミング言語で作ったAIをゲームに追加して遊べる機能もございます。

tkinterアプリケーションで遊ぶには、以下の2通りの方法がございます。

1. サンプルをインストールする
2. Windows版のアプリケーションをダウンロードする(インストール不要)

ダウンロード

Windows版のアプリケーションで遊ぶ場合は下記リンクをクリックし、
"reversi.zip"をダウンロードしてください(無料)。

- reversi.zip

"reversi.zip"を解凍後、reversi.exeをダブルクリックするとアプリケーションで遊ぶ事ができます。

メニュー一覧

選択可能なメニューの一覧です。

名前	内容
Size	盤面のサイズ(4～26までの偶数)を選択します。
Black	黒(先手)のプレイヤーを選択します。
White	白(後手)のプレイヤーを選択します。
Cputime	CPUの持ち時間を設定します。デフォルトは0.5秒となっております。
Extra	外部プログラムのAIを追加します。Cputimeの持ち時間の設定は適用されません。
Assist	打てる手の候補をハイライト表示するかどうか選びます。
Language	言語設定(English or 日本語)を選びます。
Cancel	ゲームを中断します。

プレイヤー紹介

選択可能なプレイヤーの一覧です。
難易度は8x8サイズの場合の目安となっております。

名前	特徴	難易度
User1, User2	人が操作します。	?
Unselfish	なるべく少なく取ろうとします。	★
Random	ランダムに手を選びます。	★
Greedy	なるべく多く取ろうとします。	★
SlowStarter	序盤はUnselfish、それ以降はGreedyになります。	★
Table	マス目の位置に重みをつけたテーブルで盤面を評価し、自身の形勢が良くなるよう手を選びます。なるべく少なく取り、角を狙い、角のそばは避けるよう心掛けます。	★★
MonteCarlo	モンテカルロ法で手を選びます。持ち時間の限り、すべての手の候補についてゲーム終了までランダムな手を打ちあうプレイアウトを繰り返し、最も勝率が高かった手を選びます。	★★

名前	特徴	難易度
MinMax	ミニマックス法で2手先を読んで手を選びます。Tableの盤面評価に加えて、着手可能数と勝敗を考慮します。自身の置ける場所は増やし、相手の置ける場所は減らし、勝ちが見えた手を優先するよう手を読みます。	★★
NegaMax	MinMaxの探索手法をネガマックス法に替えて、持ち時間の限り3手先を読んで手を選びます。手を読む探索効率はミニマックス法と同じです。	★★★
AlphaBeta	NegaMaxの探索手法をアルファベータ法(ネガアルファ法)に替えて、持ち時間の限り4手先を読んで手を選びます。αβ値の枝刈りにより、ネガマックス法より効率良く手を読みます。	★★★
Joseki	AlphaBetaに加えて、序盤は定石通りに手を選びます。	★★★
FullReading	Josekiに加えて、終盤残り9手からは最終局面までの石差を読んで手を選びます。	★★★
Iterative	FullReadingに反復深化法を適用して持ち時間の限り徐々に深く手を読みます。読む手の深さを増やす際は前回の深さで最も評価が高かった手を最初に調べます。それにより、不要な探索を枝刈りしやすくし、4手よりも深く手を読む場合があります。	★★★★
Edge	Iterativeの盤面評価に加えて、4辺のパターンを考慮し確定石を増やすよう手を選びます。	★★★★
Switch	Edgeの各パラメータを遺伝的アルゴリズムを使って強化し、手数に応じて5段階にパラメータを切り替えることで、よりゲームの進行に応じた手を選びます。また、探索手法をアルファベータ法からネガスカウト法に変更し、自身の着手可能数が相手より多くなる手を優先的に探索するよう候補を並び替え、探索効率を上げています。加えて終盤残り10手から石差を読みます。	★★★★
Blank8_16	Edgeのパラメータに加え、自身の石がなるべく空きマスに接しないよう考慮して手を選びます。制限時間を考慮せず必ず毎回8手先を読みます。加えて終盤残り16手(制限時間なし)から石差を読みます。	★★★★★
BlankI_16	Blank8_16と同じ評価関数で手を読みますが、こちらは制限時間0.5s以内で見つけた最善手を選びます。加えて終盤残り16手(制限時間なし)から石差を読みます。	★★★★★

プレイヤー追加機能

概要

本アプリケーションはお好きなプログラミング言語で作成したAI(追加プレイヤー)を
ゲームに参加させて遊ぶことができます。
また、あらかじめ用意された追加プレイヤーについても動作環境を準備する事で遊ぶ事ができます。
なお、追加プレイヤーのプログラムを作成する際は入出力を後述のフォーマットに準拠させて下さい。

追加プレイヤー紹介

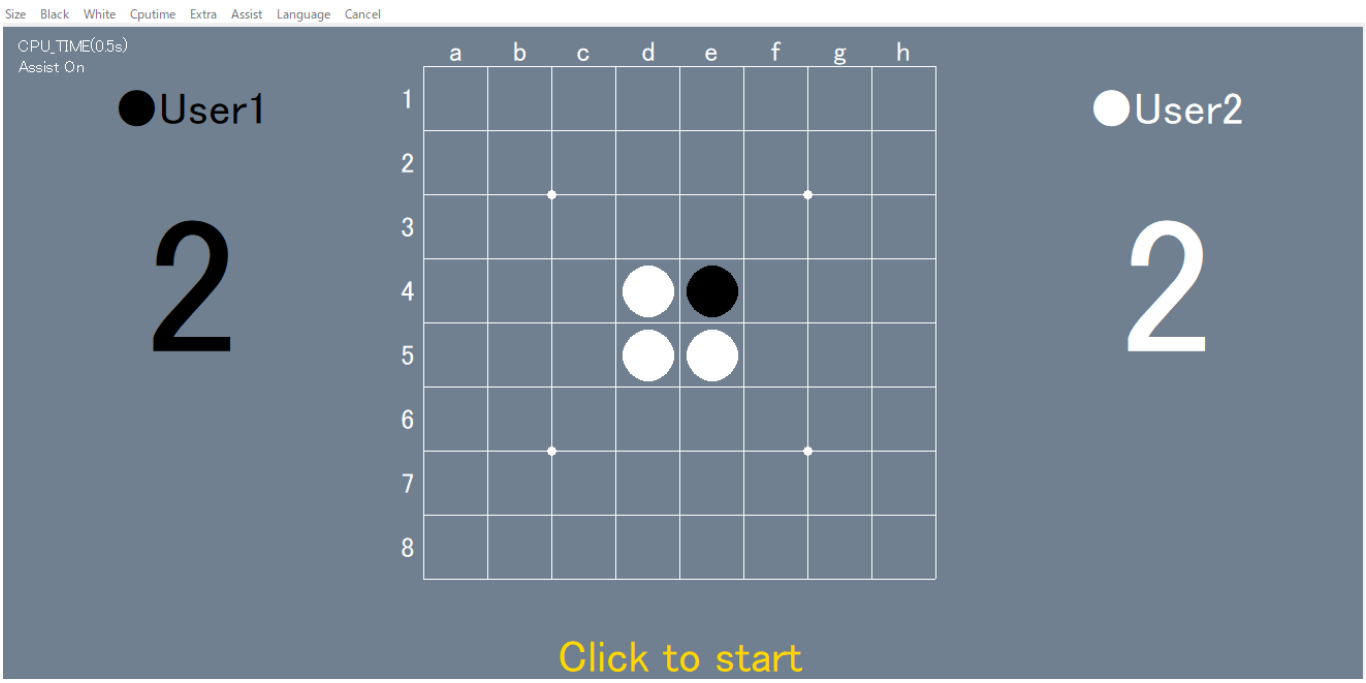
あらかじめ用意された追加プレイヤーの一覧です。
動作環境を準備し、Extraメニューより登録ファイルを読み込ませると遊べるようになります。

名前	特徴	難易度	登録ファイル	開発言語	動作確認環境
TopLeft	打てる手の中から一番上の左端を選びます。	★	topleft.json	Python	Windows10 64bit Python 3.7.6
BottomRight	打てる手の中から一番下の右端を選びます。	★	bottomright.json	Perl	Windows10 64bit Strawberry Perl 5.30.1.1
RandomCorner	角が取れる時は必ず取ります。 それ以外はランダムに手を選びます。	★	randomcorner.json	VBScript	Windows10 64bit

プレイヤー作成手順

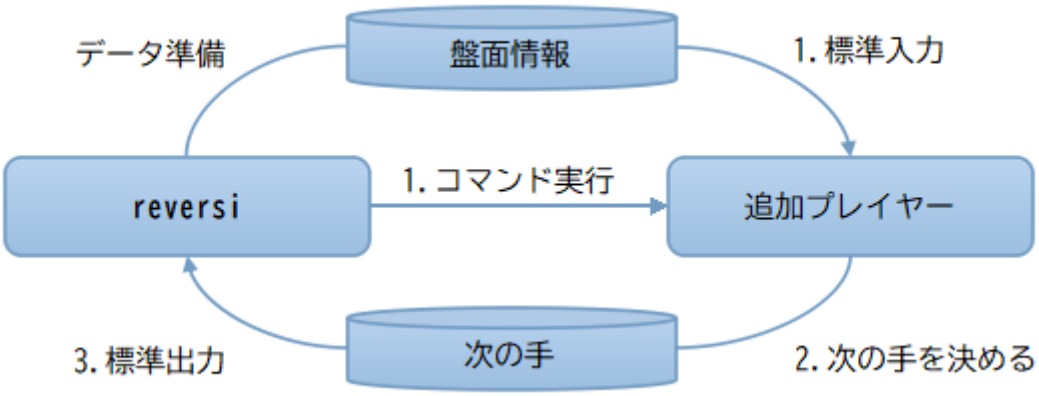
プレイヤーを自作して遊ぶには、下記の手順でプレイヤーの作成と登録を行って下さい。

1. お好きなプログラミング言語の実行環境を準備する
2. 追加プレイヤーのプログラムを書く
3. 登録ファイルを作成する
4. アプリケーションを起動する
5. Extraメニューより登録ファイルを読み込ませる



追加プレイヤーの実行

追加プレイヤーをアプリケーションに登録すると外部プログラムとして実行されるようになります。
以下に処理の流れを示します。



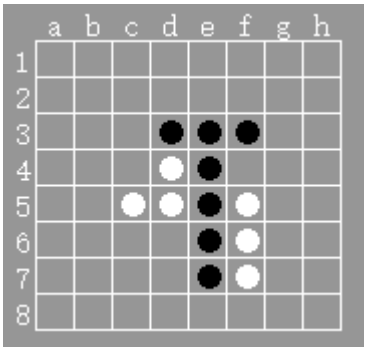
1. ゲーム開始後、追加プレイヤーの手番になるとアプリケーションは対応するプログラムのコマンドを実行します。
   その際、標準入力に盤面情報を渡し、追加プレイヤーのプログラムの応答を待ちます。
2. 追加プレイヤーは標準入力から盤面情報を受け取り、次の手を決め、その結果を標準出力します。
   (そのようなプログラムを書いて下さい)
3. アプリケーションは追加プレイヤーの標準出力(次の手)を受け取るとゲームを再開します。
   一定時間応答がない場合は追加プレイヤーのプログラムを強制終了し、反則負けとして扱います。

標準入力フォーマット

追加プレイヤーが受け取る標準入力の盤面の情報です。

```
手番の色(黒:1、白:-1)
盤面のサイズ(4～26までの偶数)
盤面の石の2次元配置(空き:0、黒:1、白:-1をスペース区切り)
```

下記に白の手番、盤面サイズ8x8の例を示します。



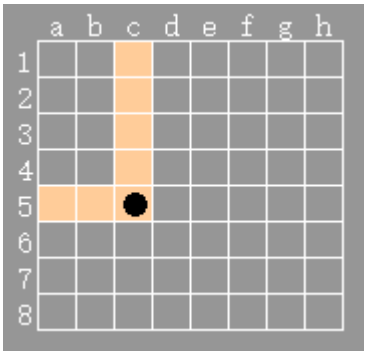
```
-1
8
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0
0 0 0 -1 1 0 0 0
0 0 -1 -1 1 -1 0 0
0 0 0 0 1 -1 0 0
0 0 0 0 1 -1 0 0
0 0 0 0 0 0 0 0
```

標準出力フォーマット

追加プレイヤーが標準出力する次の手の情報です。

```
盤面の座標(左上を起点(0,0)としてxとyをスペース区切り)
```

下記にc5へ打つ場合の例を示します。




```
2 4
```

登録ファイル

追加プレイヤーをアプリケーションに登録するために本ファイルを作成する必要があります。
登録ファイルは下記のフォーマット(JSON形式)に従って`extra/`以下に作成して下さい。
作成後、Extraメニューより読み込む事でプレイヤーが追加されます。

```
{
    "name": "追加プレイヤーの名前",
    "cmd": "追加プレイヤー実行用のコマンド",
    "timeouttime": 追加プレイヤーからの応答待ち時間(秒) ※起動に時間がかかる場合があるため、余裕を持った設定を推奨します
}
```

下記に、Windows10上のPythonで動作するTopLeft(あらかじめ用意されたプレイヤー)の例を示します。

```
{
    "name": "TopLeft",
    "cmd": "py -3.7 ./extra/python/topleft/topleft.py",
    "timeouttime": 60
}
```

コンソールアプリケーションの遊び方

ゲーム紹介

コマンドプロンプト(Windowsの場合)のようなコンソール上で遊べるリバーシです。 通常と異なる多種多様な20種類にも及ぶ盤面で遊ぶ事ができます。

サンプルをインストールすると遊べます。

メニュー画面

アプリケーションを起動すると、以下の画面が表示されます。

メニューの項目	内容
BoardType	現在選択中のボードの種類
BlackPlayer	現在選択中の黒のプレイヤー名
WhitePlayer	現在選択中の白のプレイヤー名

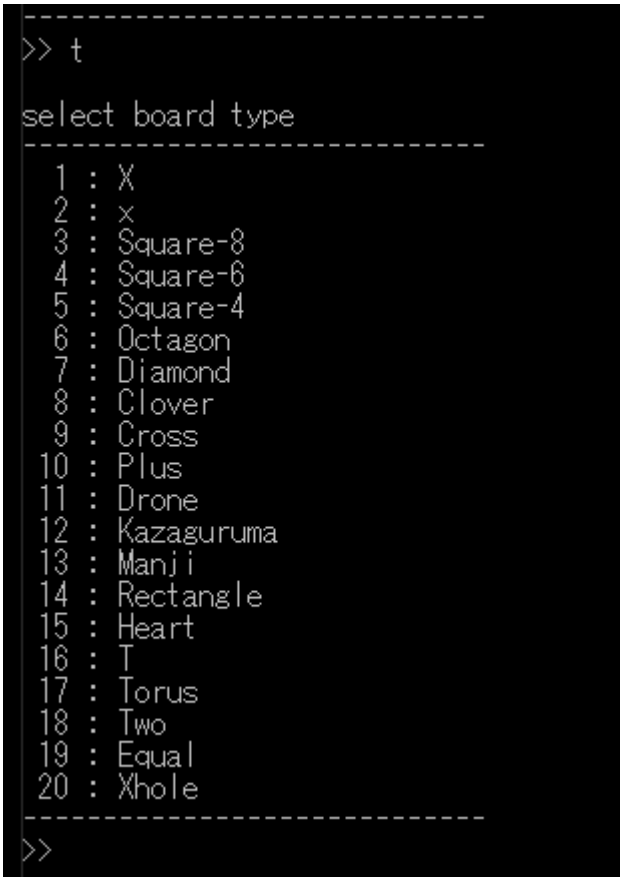
キー入力により、設定変更やゲームの開始および終了を行えます。

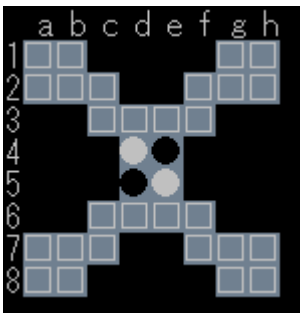
キー入力	内容
enter	Enterキー入力でゲームを開始します。
t + enter	ボードの種類を変更します。
b + enter	黒のプレイヤーを変更します。

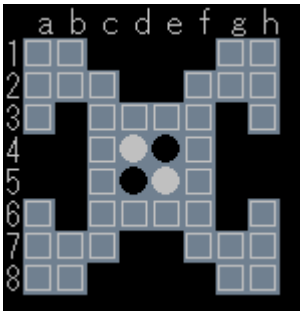
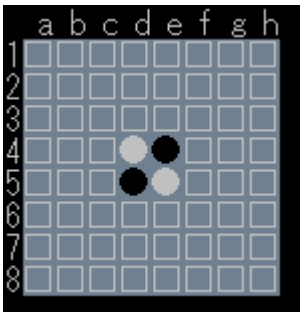
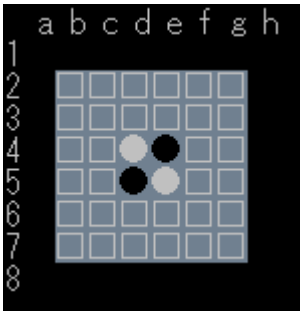
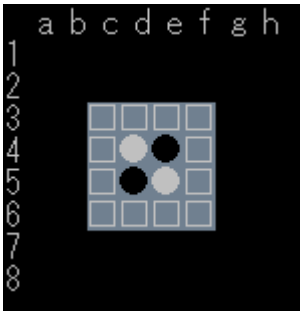
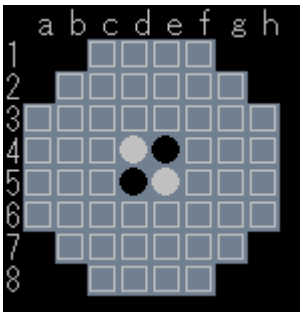
キー入力	内容
w + enter	白のプレイヤーを変更します。
q + enter	ゲームを終了します。
ctrl + c	Ctrl+cでゲームを強制終了します。(ゲーム途中で終了する場合)

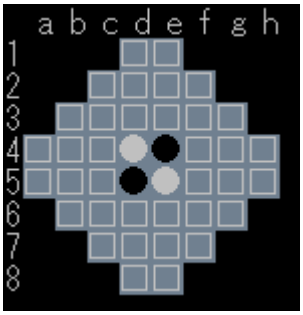
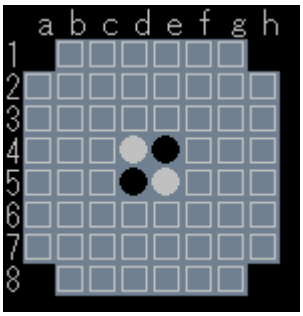
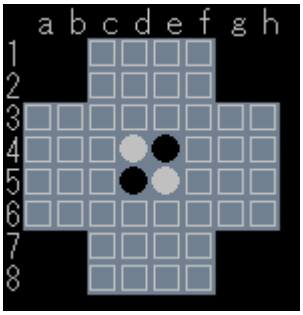
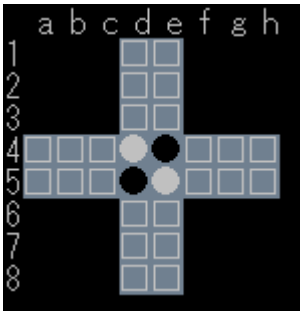
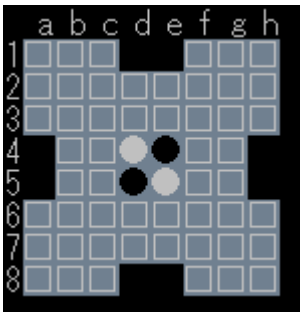
ボードを変更する

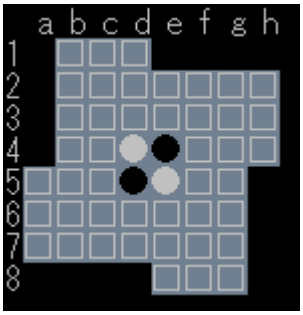
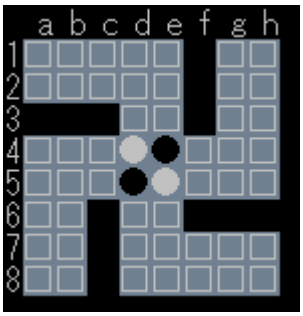
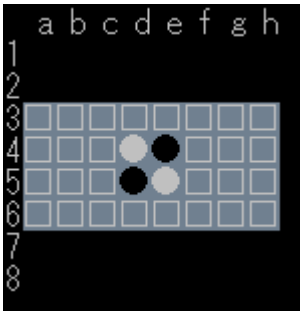
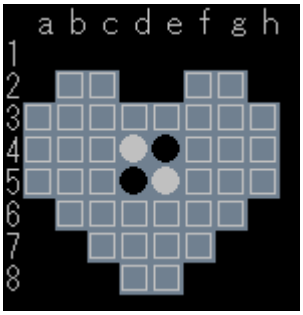
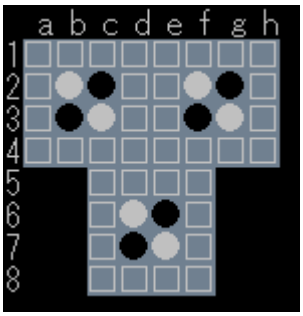
ボードの種類に対応する番号を入力すると、ボードを変更できます。

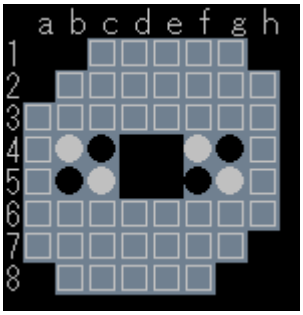
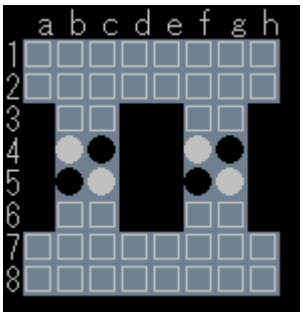
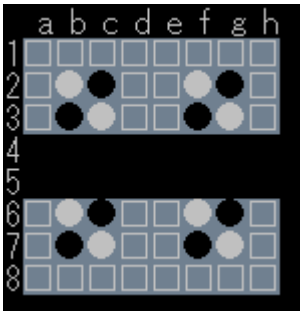
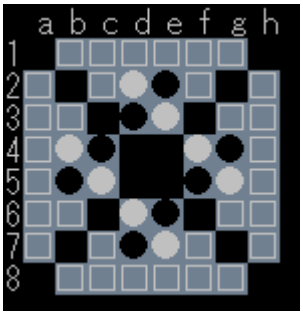


キー入力	ボード名	形状
1 + enter	X	

キー入力	ボード名	形状
2 + enter	x	
3 + enter	Square-8	
4 + enter	Square-6	
5 + enter	Square-4	
6 + enter	Octagon	

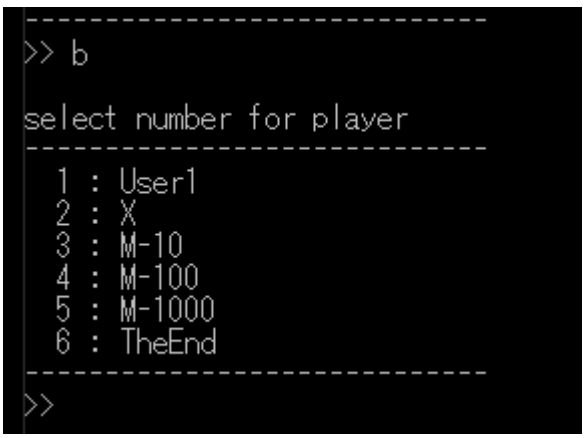
キー入力	ボード名	形状
7 + enter	Diamond	
8 + enter	Clover	
9 + enter	Cross	
10 + enter	Plus	
11 + enter	Drone	

キー入力	ボード名	形状
12 + enter	Kazaguruma	
13 + enter	Manji	
14 + enter	Rectangle	
15 + enter	Heart	
16 + enter	T	

キー入力	ボード名	形状
17 + enter	Torus	
18 + enter	Two	
19 + enter	Equal	
20 + enter	Xhole	

プレイヤーを変更する

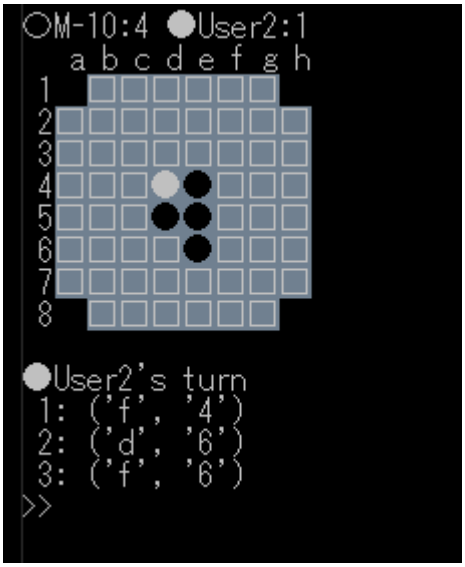
黒と白ともにプレイヤー名に対応する番号を入力すると、プレイヤー(AI)を変更できます。



キー入力	名前	難易度	特徴
1 + enter	User1, User2	人が操作	-
2 + enter	X	★	ランダム
3 + enter	M-10	★★	モンテカルロ10回
4 + enter	M-100	★★★	モンテカルロ100回
5 + enter	M-1000	★★★★	モンテカルロ1000回
6 + enter	TheEnd	★★★★★	モンテカルロ10000回 + 終盤14手完全読み

手を打つ

ゲーム開始後、打てる手(座標)に対応する番号(+ enter)を入力すると、手を打つことができます。



インストールがうまくいかない場合

reversiのインストールがうまくいかない場合は 下記の手順(1～5)に従って環境を準備して下さい。

1. Pythonのインストール

下記よりPythonの64bit版インストーラのexeをダウンロード後、インストールして下さい。
Python 3.7.6

インストール後、コマンドプロンプトを立ち上げて下記の'$'以降を入力してEnterを押し、同じ結果が出ればOKです。

```
$ py -3.7 --version
Python 3.7.6
```

2. pipの更新

reversiをPythonから実行するためにはいくつかの外部パッケージが必要となります。
正しくインストールできるようにするために下記を実行してpipをアップデートして下さい。

```
$ py -3.7 -m pip install --upgrade pip
 :
Successfully installed pip-20.0.2
```

※バージョンが異なる場合は上位であれば問題ないはずです

3. 関連パッケージのインストール

reversiの実行に必要なPythonのパッケージのインストールは下記で一括して行えます。
事前にコマンドプロンプトにてreversiフォルダ以下に移動しておいてください。

```
$ py -3.7 -m pip install -r requirements.txt
```

もしうまくいかない場合は、以降の"(パッケージインストールの補足)"を個別に実行してください。

4. Visual C++のインストール

reversiの実行にはC言語のコンパイル環境が必要となります。
下記よりVisual C++をダウンロードして下さい。
Microsoft Visual C++ 2019

5. 動作確認

サンプルを参照して、サンプルが動作するか確認してください。

(パッケージインストールの補足)

cythonパッケージのインストール

reversiを実行するためにはcythonという外部パッケージが必要となります。
下記を実行してインストールして下さい。

```
$ py -3.7 -m pip install cython
 :
Successfully installed cython-0.29.15
```

pyinstallerパッケージのインストール

reversiのexeを生成するためにはpyinstallerという外部パッケージが必要となります。
下記を実行してインストールして下さい。不要な場合は省略しても構いません。

```
```

```
$ py -3.7 -m pip install pyinstaller
 :
Successfully installed altgraph-0.17 future-0.18.2 pefile-2019.4.18 pyinstaller-
3.6 pywin32-ctypes-0.2.0
```

うまくいかない場合は下記を実行後に、再度上記を試してみて下さい。

```
$ py -3.7 -m pip install wheel
```

インストール完了後、pyinstallerを実行できるようにするために環境変数に下記を追加して下さい。

```
C:\Users\{あなたのユーザ名}\AppData\Local\Programs\Python\Python37\Scripts
```

参考書籍

- 「実践Python3」 Mark Summerfield著 斎藤 康毅訳 株式会社オライリー・ジャパン ISBN978-4-87311-739-3
- 「Java言語で学ぶデザインパターン入門」 結城 浩著 ソフトバンククリエイティブ株式会社 ISBN4-7973-2703-0
- 「日経ソフトウェア2019年11月号」 日経BP ISSN1347-4685
- 「Python計算機科学新教本」 David Kopec著 黒川 利明訳 株式会社オライリー・ジャパン ISBN978-4-87311-881-9
- 「Cython Cとの融合によるPythonの高速化」 Krurt W. Smith著 中田 秀基監訳 長尾 高弘訳 株式会社オライリー・ジャパン ISBN978-4-87311-727-0
- 「Fluent Python ----Pythonicな思考とコーディング手法」 Luciano Ramalho著 豊沢 聡、桑井 博之監訳 梶原 玲子訳 株式会社オライリー・ジャパン ISBN978-4-87311-817-8

参考サイト

- 「オセロ・リバーシプログラミング講座 ～勝ち方・考え方～」https://uguisu.skr.jp/othello/
- 「オセロ･リバーシの勝ち方、必勝法」https://bassy84.net/
- 「強いオセロプログラムの内部動作」http://www.amy.hi-ho.ne.jp/okuhara/howtoj.htm
- 「オセロAI入門」https://qiita.com/na-o-ys/items/10d894635c2a6c07ac70
- 「Thellのアルゴリズムについて」http://sealsoft.jp/thell/algorithm.html
- 「ブラウザで打てる自分より強いオセロAIを作る」https://github.com/trineutron/othello

脚注

[1]: 一部でCythonを使用しています。↑

ライセンス

本リポジトリのソースコードはMITライセンスです。 商用・非商用問わず、ご自由にお使い下さい。