Tic Tac Toe (CUI - Python)

S.Matoike

目次

第 1 章

Python による三目並べ

三目並べは「二人零和有限確定完全情報ゲーム」と呼ばれるゲームに分類されます．簡単に言うと「勝敗は偶然に左右されず，最善手を打てばどちらかが勝つまたは引き分けになるゲーム」のことです．他にもリバーシ，チェッカー，チェス，将棋に囲碁などもこのゲームに分類されます．

初めに三目並べというゲームについて定義をしておきます．

三目並べ: 3 × 3 の格子上に二人のプレイヤーが「O」「X」を配置し、自分のマークを先に 3 つ並べた方が勝ちです．

1.1　太郎さんと花子さんの対戦

プログラムを実行すると盤面が表示され、×の石を置く場所を指定するよう促されます

画面上に示された番号を入力すると、その番号のスロットに×の石が置かれた盤面が表示され、次の手番の○に、石を置く場所を指定するように促されます

手番を交互に変えながらゲームは進み、縦、横、斜めの何れかに、先に一列に自分の石を並べた方が勝ちとなります

既に石の置かれているスロット番号を指定できませんし、スロット番号として 0 〜 8 以外の数値を指定することもできません

```
スタート！ [Tic Tac Toe]
/---|---|---\
| 0 | 1 | 2 |
|---|---|---|
| 3 | 4 | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
太郎 さんの turn です。スロット番号を指定して下さい : 4
/---|---|---\
| 0 | 1 | 2 |
|---|---|---|
| 3 | X | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
花子 さんの turn です。スロット番号を指定して下さい : 2
/---|---|---\
```

```
| 0 | 1 | 0 |
|---|---|---|
| 3 | X | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
太郎 さんの turn です。スロット番号を指定して下さい ：
```

1.1.1　ゲームの進行を司るクラス：Game

Game クラスのオブジェクトである game を生成し、game オブジェクトの中の start() というメソッドを実行します

この後、main.py と Game.py でファイルを別にしていきますので、以下のように、main.py を書いて、その冒頭で、Game.py から Game というクラスを import するという記述が必要になります

ソースコード 1.1　main.py

```
1 from Game import Game
2
3 if __name__ == '__main__':
4     game = Game()
5     game.start()
```

1.1.2　盤面を管理するクラス：Board

ゲームの盤面を保持するクラス Board を作っていきます

Board クラスのオブジェクト board を、Game クラスのコンストラクタで生成することにします

ゲームの進行を行う start() メソッドでは、board オブジェクトを印刷するメソッド draw() を呼び出すようにしています

Game クラスで、Board クラスを使っていますから、そのファイルの冒頭で、Board ファイルから Board クラスを import せよという指示が必要です

ソースコード 1.2　class Game

```
1 from Board import Board
2
3 class Game:
4     def __init__(self):
5         self.board = Board()
6
7     def start(self):
8         self.board.draw()
```

盤面の実体は、各スロットに対応させた 9 個の整数からなるリスト board で保持することとし、Board クラスのコンストラクタで準備を整えます

draw() メソッドは、盤面 board の状態を画面に表示する処理を行っています

9 個の文字からなるリスト bd を用意して、board の値が 0 なら空きスロット、整数の 1 なら一人目の

プレイヤーの石、整数の 2 なら二人目のプレイヤーの石が置かれているとして、それぞれ文字'X' と'O' の文字を表示させています

print() 関数で、「文字列.format」という書き方によって、{} の位置に bd の文字を書いていますが、「文字列.format」という書き方よりも、「f 文字列」という書き方によって {} に値を埋め込む方が Pythonic な書き方だと「Effective Python」の項目 4 には書かれています

ソースコード 1.3　class Board

```
1  class Board:
2      def __init__(self):
3          self.board = [0,0,0,\
4                        0,0,0,\
5                        0,0,0]
6
7      def draw(self):
8          bd = ['', '', '',\
9                '', '', '',\
10               '', '', '']
11         n = 0
12         for val in self.board:
13             if val==0:
14                 bd[n] = str(n)
15             elif val==1:
16                 bd[n] = 'X'
17             elif val==2:
18                 bd[n] = 'O'
19             n += 1
20         print( '/---|---|---\\')
21         print( '| {} | {} | {} |'.format(bd[0], bd[1], bd[2]) )
22         print( '|---|---|---|')
23         print(f'| {bd[3]} | {bd[4]} | {bd[5]} |')
24         print( '|---|---|---|')
25         print(f'| {bd[6]} | {bd[7]} | {bd[8]} |')
26         print( '\\---|---|---/')
```

1.1.3　プレーヤのクラス：Player

Game クラスのコンストラクタの中で、Player クラスのオブジェクトとして、taro と hanako を生成しています。Player クラスのコンストラクタへの、一つ目の引数で名前を、二つ目の引数で盤面におく石に対応づけた整数を渡しています。

この整数は、taro と hanako の識別子としての役割を持たせますので、区別できる値でなければなりません。

また、game オブジェクトの中で PLAYER というリストを定義して、taro と hanako という Player クラスのオブジェクトのリストを作っています。PLAYER[0] によって taro オブジェクトを、PLAYER[1] によって hanako オブジェクトを取り出せることになります。

ゲームの進行は大まかにいうと、①盤面を表示して、②一方のプレーヤを選んで、③そのプレーヤが盤面上に石を置いて、④手番を交代して、をゲームオーバーまで繰り返すことになります

ソースコード 1.4　class Game

```
from Board import Board
from Player import Player

class Game:
    def __init__(self):
        self.board = Board()
        taro = Player('太郎', 1)
        hanako = Player('花子', 2)
        self.PLAYER = [taro, hanako]

    def start(self):
        turn = 0
        while True:                          # 以下をゲームオーバーまで繰り返す
            self.board.draw()                    # ①盤面を表示して
            player = self.PLAYER[ turn ]         # ②プレーヤを選んで
            player.put_stone( self.board )       # ③そのプレーヤが盤面に石を置いて
            turn = (turn+1) % 2                  # ④手番を交代する
```

Player クラスでは、そのコンストラクタで、引数で受け取った名前と番号を各オブジェクトに保存するようにしています

put_stone() メソッドでは、input() 文でプレーヤにスロット番号の入力を促し、受け取った番号（文字列 s）を int(文字列) 関数によって整数に変換して、盤面 board 上のスロット番号の位置に、このプレーヤオブジェクトが保持している id 番号（1 か 2）を納めています

ソースコード 1.5　class Player

```
from Board import Board

class Player:
    def __init__(self, name, id):
        self.name = name
        self.id = id

    def put_stone(self, board):
        s = input(self.name + "さんの手番です。スロット番号は？ : ")
        n = int(s)
        board.board[ n ] = self.id
```

1.1.4　Board クラスで勝敗の判定

Board クラスの winner() メソッドでは、行方向に 3 つ（row0, row1, row2）、列方向に 3 つ（col0, col1, col2）、斜め方向に 2 つ（crs0, crs1）のそれぞれで、各スロットの中の値が一致しているか否かをチェックしています

行または列、斜めのどれか一つでも一致しているものがあったら、そのときの player の勝ちです

ソースコード 1.6　class Board

```
class Board:
    def __init__(self):
        self.board = [0,0,0,0,0,0,0,0,0,0]

    def draw(self):
        bd = ['', '', '','', '', '','', '', '']
```

```
        for n, val in enumerate( self.board ):
            ・
            変更なし
            ・
        print( '\\---|---|---/')

    # 以下を追加
    def winner(self, player):
        row0 = self.board[0]!=0 and\
               self.board[0]==self.board[1] and self.board[0]==self.board[2]
        row1 = self.board[3]!=0 and\
               self.board[3]==self.board[4] and self.board[3]==self.board[5]
        row2 = self.board[6]!=0 and\
               self.board[6]==self.board[7] and self.board[6]==self.board[8]
        col0 = self.board[0]!=0 and\
               self.board[0]==self.board[3] and self.board[0]==self.board[6]
        col1 = self.board[1]!=0 and\
               self.board[1]==self.board[4] and self.board[1]==self.board[7]
        col2 = self.board[2]!=0 and\
               self.board[2]==self.board[5] and self.board[2]==self.board[8]
        crs0 = self.board[0]!=0 and\
               self.board[0]==self.board[4] and self.board[0]==self.board[8]
        crs1 = self.board[2]!=0 and\
               self.board[2]==self.board[4] and self.board[2]==self.board[6]
        if col0 or col1 or col2 or row0 or row1 or row2 or crs0 or crs1:
            return True
```

勝敗を判定して結果を表示し、break によってゲームのループを抜けます

ソースコード 1.7　class Game

```
from Board import Board
from Player import Player

class Game:
    def __init__(self):
        self.board = Board()
        taro = Player('太郎', 1)
        hanako = Player('花子', 2)
        self.PLAYER = [taro, hanako]

    def start(self):
        turn = 0
        while True:
            self.board.draw()
            player = self.PLAYER[turn]

            b1 = self.board.winner(player)
            if b1:
                print( player.name + 'さん　の勝ち')
                break

            turn = (turn+1) % 2
```

1.1.5　引き分けの判定

board オブジェクトが持っている board プロパティは、9 個の各スロットの状態（空かマルかバツか→ 0 か 1 か 2 か）を保持するリストでした。スロットが空き（スロットの値がゼロ）であれば、石を置くことができます。

値がゼロのスロット番号（n）を集めたリストを vacant に作り、その長さは len() 関数で調べることができます。リストの長さがゼロであるということは、空のスロット（値がゼロのスロット）は無くなったということですから、もう石を置くことはできません（このとき引き分けです）。

```
        # 引き分けの判定はここから
        empty = []
        n = 0
        for slot in board.board:
            if slot==0:
                empty.append( n )
            n += 1
        if len( empty )==0:
            return False
        # ここまで
```

```
    # 引き分けの判定はここから
    empty = []
    for n in range( len(board.board) ):
        slot = board.board[n]
        if slot==0:
            empty.append( n )

    if len( empty )==0:
        return False
    # ここまで
```

引き分けの判定部分は、上のように書くこともできますが、ここでは以下のような enumerate() 関数の使い方を学習しましょう。「Effective Python」には項目 7 で、range ではなく enumerate を使うのが Pythonic だとされています。

また、if len(empty)==0: という記述は、if not empty: と書いた方が Pythonic だと同じ本の第一章には書かれています。

ソースコード 1.8　class Board

```
class Board:
    def __init__(self):
        self.board = [0,0,0,0,0,0,0,0,0]

    def draw(self):
        bd = ['', '', '','', '', '','', '', '']
        for n, val in enumerate( self.board ):
            ・
            変更なし
            ・
        print( '\\---|---|---/')

    def winner(self, player):
            ・
            変更なし
            ・
        if col0 or col1 or col2 or row0 or row1 or row2 or crs0 or crs1:
            return True

    # 以下を追加
    def vacant(self):
        empty = []
        for n, slot in enumerate(self.board):
```

```
24              if slot == 0:
25                  empty.append(n)
26          return empty
```

入力されたスロット番号 n が、vacant リストの中にあることを確認すること（if n in vacant:）によって、既に置かれた石の上に上書きすることを防いでいます。

今の段階で、put_stone() メソッドが返す値は、False ならば引き分け、True ならばボード上に石を置きましたという意味になります

ソースコード 1.9　class Player

```
1  from Board import Board
2
3  class Player:
4      def __init__(self, name, id):
5          self.name = name
6          self.id = id
7
8      def put_stone(self, board):
9          # 引き分けの判定はここから
10         vacant = board.vacant()
11         if not vacant:
12           return False
13         # ここまで
14         while True:
15             s = input(self.name + "さんの手番です。スロット番号は？ : ")
16             n = int(s)
17             if n in vacant:       # 空きスロットのリストに番号 n は含まれているか？
18                 board.board[ n ] = self.id
19                 break
20         return True
```

Game クラスでは、put_stone() メソッドを呼び出して判定する必要があります

引き分けの判定（b2）で False が返された（not b2 == True）なら、’引き分け’と表示してゲームの反復を抜けます（break）

b2 が True を返してきた場合は、順当にその時の手番（turn）の player オブジェクトが、盤面オブジェクト（self.board）に石を置いた（put_stone() した）場合になりますから、そのままゲームを続行します

引き分けと勝敗の判定を、ゲーム進行の途中に入れています

引き分けた場合、あるいは勝敗のついた場合のいずれの場合でも、break 文によってゲームの進行（反復）を終えています

ソースコード 1.10　class Game

```
1  from Board import Game
2  from Player import Player
3
4  class Game:
5      def __init__(self):
6          self.board = Board()
7          taro = Player('太郎', 1)
8          hanako = Player('花子', 2)
9          self.PLAYER = [taro, hanako]
10
```

```
11      def start(self):
12          turn = 0
13          while True:
14              self.board.draw()
15              player = self.PLAYER[turn]
16
17              b2 = player.put_stone( self.board )
18              if not b2:
19                  print('引き分け')
20                  break
21
22              b1 = self.board.winner(player)
23              if b1:
24                  print( player.name + 'さん　の勝ち')
25                  break
26
27              turn = (turn+1) % 2
```

1.2　太郎さんとコンピュータの対戦

花子さんをコンピュータに変更します

1.2.1　Game クラスの書き換え

Game クラスの最初、コンストラクタで Player クラスのオブジェクト生成を変更します

ソースコード 1.11　class Game

```
1  from Board import Game
2  from Player import Player
3
4  class Game:
5      def __init__(self):
6          self.board = Board()
7          human = Player('Taro', 1)          # 変更点はここから3行
8          machine = Player('computer', 2)
9          self.PLAYER = [human, machine]
10
11     def start(self):
12         turn = 0
13         while True:
14             self.board.draw()
15                 ・
16                 ・    この部分に変更なし
17                 ・
```

1.2.2　Player クラスの書き換え

Player クラスでは、computer オブジェクトの場合には乱数によって、 vacant リストの中から石をおくスロット番号を選ぶようにします

human オブジェクトの場合は、スロット番号の入力を促しています

Player クラスのオブジェクトは、human の場合と machine の場合がありますから、Player クラスの

それぞれのオブジェクトが持っている self.id の値を見て、それが 1 なら human オブジェクト、2 なら machine オブジェクトだと判定している部分に注目しましょう

ソースコード 1.12　class Player

```
1  from Board import Board
2  from random import randint
3
4  class Player:
5      def __init__(self, name, id):
6          self.name = name
7          self.id = id
8
9      def put_stone(self, board):
10         # 引き分けの判定はここから
11         vacant = board.vacant()
12         if not vacant:
13             return False
14         # 変更点は以下の部分
15         if self.id == 2:       # Computerの場合は乱数で
16             n = randint( 0, len(vacant)-1 )
17             board.board[ vacant[n] ] = self.id
18         else:                  #
               Taroの場合はキーボード入力でスロット番号を指定させる
19             while True:
20                 s = input(self.name + "さんの手番です。スロット番号は？ : ")
21                 n = int(s)
22                 if n in vacant:
23                     board.board[ n ] = self.id
24                     break
25         return True
```

1.2.3　Board クラスの書き換え

winner() メソッドをもう少し簡潔に記述し直すことにしましょう

ソースコード 1.13　class Board

```
1  class Board:
2          .
3          . この部分に変更なし
4          .
5      def winner(self, player):
6          def scan(n1, n2, n3):
7              return self.board[n1]!=0 and\
8                     self.board[n1]==self.board[n2] and\
9                     self.board[n1]==self.board[n3]
10         return scan(0,1,2) or scan(3,4,5) or scan(6,7,8) or\
11                scan(0,3,6) or scan(1,4,7) or scan(2,5,8) or\
12                scan(0,4,8) or scan(2,4,6)
```

参考文献

[1]