

知識情報システム実習

鈴木伸崇

2022 年 10 月 10 日

1 概要

この実習では、コンピュータと対戦するオセロゲームを作成し、思考アルゴリズムの基礎を学ぶ。なお、「オセロ」はメガハウス（旧ツクダオリジナル、バンダイナムコグループ系列の 1 社）の登録商標であるため、一般的には「リバーシ」などの名称で呼ばれることが多い。この実習では、特に気にせずに「オセロ」と呼ぶことにする。

オセロは二人零和完全情報ゲームである。ここで、零和は一方のプレイヤーが得た得点と、他方のプレイヤーの損害が一致することである。完全情報とは、全ての情報が両方のプレイヤーに公開されていることである。例えば、すごろくは、自分の得た得点が必ずしも相手の損害と一致するとは限らないので、零和ゲームではない。また、多くのトランプのゲームでは、他人の手札は自分には見えないので完全情報ゲームではない。

今回は準備として、Ruby に慣れつつ盤の定義や石を置く操作などを行うことにする。

1.1 進め方

第 2 回（10/10）と第 3 回（10/17）は対面で、実習室 I(7C102)で行います。第 4 回（10/24）はオンラインです。第 5 回以降は、ご相談の上で決めたいと思います。中間発表会（班内）と、最終発表会（受講生全体で）があります。

基本的には、実習室の PC を使うという想定です。OS は Linux をご利用ください（GUI に Ruby/Tk を使用しますが、実習室の Windows にはインストールされていないため）ご自分の PC を利用することも特に妨げませんが、各自の PC に必要に応じて Ruby と Ruby/Tk をインストールする必要があります。この辺りの情報は追って補足したいと思います。

毎週、授業開始時間までに manaba に資料を掲載しますので、資料に従ってプログラムを作成してください。資料を読み進めていくと「課題」と書かれたところがありますので、そこで課題を完了させて次に進んでください。

出席は respon で取ります。課題が全て終わるか、力が尽きるか、Ruby が嫌いになったら提出してください。

2 盤や定数の定義

まず、ゲームで使う盤や諸々の定数を定義し、初期化を行うプログラムを以下に示す。本プログラムにおける、オセロ盤とその配列による表現を図 1 に示す。オセロ盤は、通常、横軸をアルファベット、縦軸を整数値で数える。プログラムでは、オセロ盤は 2 次元配列 `@rawBoard` で表すことにする。なお、Ruby では、インスタンス変数の先頭には@が付く（後述。ここでは、そういうものだと思っておけばよい）。オセロ盤のサイズは 8x8 であるが、配列では周囲に壁を設けてある（境界の判定を楽にするのと、盤の位置と配列において石が置かれる位置を一致させるため）ため、サイズが 10x10 となっている。プログラム上では、

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

オセロの盤

	0	1	2	3	4	5	6	7	8	9
0	WALL	WALL	WALL	WALL	WALL	WALL	WALL	WALL	WALL	WALL
1	WALL	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	WALL
2	WALL	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	WALL
3	WALL	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	WALL
4	WALL	EMPTY	EMPTY	EMPTY	WHITE	BLACK	EMPTY	EMPTY	EMPTY	WALL
5	WALL	EMPTY	EMPTY	EMPTY	BLACK	WHITE	EMPTY	EMPTY	EMPTY	WALL
6	WALL	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	WALL
7	WALL	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	WALL
8	WALL	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	WALL
9	WALL	WALL	WALL	WALL	WALL	WALL	WALL	WALL	WALL	WALL

配列 @rawBoard による表現

図 1: オセロ盤と配列による表現

BLACK = 1
 WHITE = -1
 EMPTY = 0
 WALL = 2

のように、黒石が BLACK、白石が WHITE、石が置かれてない場合は EMPTY、壁は WALL という定数で表され、その値は上記の通りである。BLACK と WHITE は、-1 を掛けると互いに相手の値になることに注意（その方がプログラムが簡潔になる）では、最初のプログラムを以下に示す。

```

1  # coding: utf-8
2
3  # 盤に配置する石, 壁, 空白
4  BLACK = 1
5  WHITE = -1
6  EMPTY = 0
7  WALL = 2
8
9  # 石を打てる方向 (2進数のビットフラグ)
10 NONE = 0
11 UPPER = 1
12 UPPER_LEFT = 2
13 LEFT = 4
14 LOWER_LEFT = 8
15 LOWER = 16
16 LOWER_RIGHT = 32
17 RIGHT = 64
18 UPPER_RIGHT = 128
19

```

```

20 # 盤のサイズと手数の最大数
21 BOARD_SIZE = 8
22 MAX_TURNS = 60
23
24 # 盤を表すクラスの定義
25 class Board
26
27   # 盤を表す配列
28   @rawBoard = nil
29   # 石を打てる場所を格納する配列
30   @movableDir = nil
31
32   # 盤を（再）初期化
33   def init
34     @turns = 0
35     @current_color = BLACK
36
37     # 配列が未作成であれば作成する
38     if @rawBoard == nil
39       @rawBoard = Array.new(BOARD_SIZE + 2).map{Array.new(BOARD_SIZE + 2,EMPTY)}
40     end
41     if @movebleDir == nil
42       @movableDir = Array.new(BOARD_SIZE + 2).map{Array.new(BOARD_SIZE + 2,NONE)}
43     end
44
45     # @rawBoardを初期化, 周囲を壁 (WALL) で囲む
46     for x in 0..BOARD_SIZE + 1 do
47       for y in 0..BOARD_SIZE + 1 do
48         @rawBoard[x][y] = EMPTY
49         if y == 0 or y == BOARD_SIZE + 1 or x == 0 or x == BOARD_SIZE + 1
50           @rawBoard[x][y] = WALL
51         end
52       end
53     end
54
55     # 石を配置
56     @rawBoard[4][4] = WHITE
57     @rawBoard[5][5] = WHITE
58     @rawBoard[4][5] = BLACK
59     @rawBoard[5][4] = BLACK
60
61     # self.initMovable
62   end
63   # ここに initMovable と checkmobility の定義を追加
64
65   # ここに move と loop の定義を追加

```

```

66 end
67
68 # Board インスタンスの生成
69 board = Board.new
70
71 # 盤を初期化
72 board.init
73 # loop の実行（コメントは後で外す）
74 # board.loop

```

上記プログラムについて説明する．まず大前提として，Python ではインデント（字下げ）が意味をもっており，字下げを間違えるとプログラムが動作しなくなることがあった．Ruby や他のプログラミング言語では，インデントはあくまで見た目を整えて可読性を向上させるためのものであり，インデントの有無や量によってプログラムの動作が変わることはない．

また，1 行のコメントは Python と同様 `#` を用いる．

- 3～22 行目：定数の定義．9～18 行目の定数については後述．
- 25～66 行目：Board クラスの定義．このクラスの中で，オセロ盤を実現するための変数や，盤を操作するためのメソッド（関数）等をまとめて定義する．

クラスはいわば「雛形」であり，そこからオブジェクトを生成する．クラス内で定義される，オブジェクトが使用する変数をインスタンス変数といい，先頭に`@`を付与する．

- 27～30 行目：`@rawBoard` は盤を表す 2 次元配列（図 1 右）で，`@movableDir` は「石を打てる場所」を格納する配列（後述）である．
- 33～62 行目：初期化を行うメソッド `init` の定義．Ruby では，`def` を使ってメソッドを定義する．なお，`def` の定義は 62 行目までであるが，定義の終わりに `end` を記述する．Python ではインデントによって範囲を表していたが，Ruby では `end` を使うことによって終了位置を明示する．`def` 以外にも，`if` 文や `for` 文など，終了位置を示す必要のあるものは `end` を用いる．Board などのクラス定義も同様である．

* `@turns` は手数（盤に置かれた石の総数，何手まで進んだか）を表す変数．`@current_color` は現在の手番を表す変数で，最初は黒の番なので `BLACK` に初期化されている．

* 38～43 行目：2 次元配列 `@rawBoard` と `@movableDir` を生成する．39, 42 行目は分かり難いが，あまり気にしなくてよい．なお，38 行目の `if` 文について，Python では `if` 文の最後にコロンの `(:)` を付けていたが，Ruby では不要である．その代わり，`if` が終了する箇所に `end` を書く必要がある（例えば，38 行目の `if` と 40 行目の `end` が対応している）．`for` 文などでも同様である．

* 46～59 行目：`@rawBoard` を初期化し，図 1 右のような状態にしている．46 行目の `for` 文は，Python の

```
for x in range(0, BOARD_SIZE+1):
```

と同等である．

* 61 行目：`initMovable` は `@movableDir` を初期化するメソッド．ここではコメントにしておくこと（後で使う）

- 69 行目：Board オブジェクト `board` を生成する．
- 72 行目：生成した `board` を `init` で初期化．74 行目は現時点ではコメントのままとする（後で使う）．

課題

上記のプログラムをよく読んで、Python との違いを噛み締めなさい。噛みしめることができたか、あるいは飽きてきたら、このプログラムが reversi.rb として manaba に掲載されているので、ダウンロードして実行しなさい。プログラムは、ターミナル等から、

```
ruby reversi.rb
```

などとすれば実行できる。なお、この時点では実行しても何も出力されないの、エラーが出なければ問題ないと解釈すればよい。

また、Ruby ではインデントを増やしたり削除しもプログラムの動作に影響がない。インデントを修正して実行してみなさい。

3 石をひっくり返せる方向を求める

オセロで石を打つ処理を行う場合、「その位置に石が打てるかどうか」「どの方向にひっくり返せるか」といった情報が必要になる。これを求めるメソッドを定義する。なお、ここでは方向を求めるのみで、実際にひっくり返すには別のメソッドを定義して行う（次回）。

この情報は何度もアクセスされることになるため、配列を用意してその情報を格納しておくことにする。図 2 にその配列の様子を示す。@movableDir という配列を用意し、その各要素に対して、その位置で石をひっくり返せるならその方向を、ひっくり返せないなら NONE を格納する。例えば、黒の番であるとして、オセロ盤で d3 の位置を考えると「下」の石はひっくり返すことができるので、@movableDir の値は「下」を表す LOWER となっている。同様に、オセロ盤で c4 の位置を考えると、「右」の石をひっくり返すことができるので、@movableDir の値は「右」を表す RIGHT となっている。このような値を計算し、@movableDir に格納することを考える。

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

	0	1	2	3	4	5	6	7	8	9
0	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用
1	未使用	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	未使用
2	未使用	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	未使用
3	未使用	NONE	NONE	NONE	LOWER	NONE	NONE	NONE	NONE	未使用
4	未使用	NONE	NONE	RIGHT	NONE	NONE	NONE	NONE	NONE	未使用
5	未使用	NONE	NONE	NONE	NONE	NONE	LEFT	NONE	NONE	未使用
6	未使用	NONE	NONE	NONE	NONE	UPPER	NONE	NONE	NONE	未使用
7	未使用	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	未使用
8	未使用	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	未使用
9	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用

オセロの盤

配列 @movableDir

図 2: 打てる方向を格納する配列 @movableDir

ひっくり返す方向は、前述の p.2 のプログラムにあるように、以下のように定義されている。

NONE = 0

```

UPPER = 1
UPPER_LEFT = 2
LEFT = 4
LOWER_LEFT = 8
LOWER = 16
LOWER_RIGHT = 32
RIGHT = 64
UPPER_RIGHT = 128

```

これは2進数のビットフラグとなっており、2進数で表すと以下のようになる。

```

NONE          = 00000000
UPPER          = 00000001
UPPER_LEFT    = 00000010
LEFT          = 00000100
LOWER_LEFT    = 00001000
LOWER         = 00010000
LOWER_RIGHT   = 00100000
RIGHT         = 01000000
UPPER_RIGHT   = 10000000

```

例えば、@movebleDir のある位置の値が 01000000 であった場合、それは RIGHT を意味するので、その位置からみて右の石をひっくり返すことができることを表す。また、00000101 のように複数の位置が1となる場合もある。この場合、UPPER と LEFT の位置が1であるので、上と左の石をひっくり返すことができるという意味になる。このように、それぞれの方向のビットは位置が重ならないように定義されているため、複数の方向を1つの変数で表すことができる。

以下に、プログラムの一部を示す。1～9行目は、それぞれの座標 (x,y) に対して checkMobility でひっくり返せる方向 dir を計算し、それを@movableDir[x][y] に格納している。dir には、上述の 00000101 のような値が格納されている。11行目以降が checkMobility の定義であり、ここでひっくり返せる方向を求めている。例えば、21～32行目では、(x,y) の「上」の石がひっくり返せるかどうかを調べ、ひっくり返せる場合は dir に UPPER をセットしている。より具体的には、「上方向にひっくり返せるかどうか」は、自分の真上に自分と反対の色の石が連続して1個以上並んでおり（24～28行目）、その直後に自分と同じ色の石が置かれている場合（29行目）に、上にひっくり返せると判断できるので、dir に UPPER をセットする（30行目）。30行目の

```
dir |= UPPER
```

は、dir と UPPER のビットごとの OR をとり、その結果を dir に代入することを表す。同様に、34～45行目では、「下がひっくりかえせるかどうか」を判定し、ひっくり返せる場合は dir に LOWER をセットしている。なお、このプログラムでは、残る4方向、

- 右上 (UPPER_RIGHT)
- 左上 (UPPER_LEFT)
- 左下 (LOWER_LEFT)
- 右下 (LOWER_RIGHT)

の判定は省略されている（ので、ここは後ほど課題で作成する）。

```

1  # @movableDir の値を設定
2  def initMovable
3      for x in 1..BOARDSIZE do
4          for y in 1..BOARDSIZE do
5              dir = self.checkMobility(x,y,@current_color)
6              @movableDir[x][y] = dir
7          end
8      end
9  end
10
11 # 石を打てる方向を調べる
12 def checkMobility(x1,y1,color)
13     # 石が置いてあれば打てない
14     if @rawBoard[x1][y1] != EMPTY
15         return NONE
16     end
17
18     # 打てる方向 dir を初期化
19     dir = NONE
20
21     # 上
22     x = x1
23     y = y1
24     if @rawBoard[x][y-1] == -color
25         y = y - 1
26         while (@rawBoard[x][y] == -color)
27             y = y - 1
28         end
29         if @rawBoard[x][y] == color
30             dir |= UPPER
31         end
32     end
33
34     # 下
35     x = x1
36     y = y1
37     if @rawBoard[x][y+1] == -color
38         y = y + 1
39         while (@rawBoard[x][y] == -color)
40             y = y + 1
41         end
42         if @rawBoard[x][y] == color
43             dir |= LOWER
44         end
45     end
46

```

```

47     # 左
48     x = x1
49     y = y1
50     if @rawBoard[x-1][y] == -color
51         x = x - 1
52         while (@rawBoard[x][y] == -color)
53             x = x - 1
54         end
55         if @rawBoard[x][y] == color
56             dir |= LEFT
57         end
58     end
59
60     # 右
61     x = x1
62     y = y1
63     if @rawBoard[x+1][y] == -color
64         x = x + 1
65         while (@rawBoard[x][y] == -color)
66             x = x + 1
67         end
68         if @rawBoard[x][y] == color
69             dir |= RIGHT
70         end
71     end
72
73     # 以下同様に、右上 (UPPER_RIGHT), 左上 (UPPER_LEFT), 左下 (LOWER_LEFT),
74     # 右下 (LOWER_RIGHT) の判定を行うコードを書く
75
76     return dir
77 end

```

課題

- 上のプログラムを読み、大体意味がわかったら p.2~3 のプログラムに追加しなさい。manaba に kadail.txt があるのでそれを使えば良い。追加する位置は、p.3 のプログラムの 63 行目（コメントがあるところ）である。すなわち、initMovable と checkMobility を Board クラスのメソッドとして追加する。

加えて、checkMobility 内にコードを追加して、残る 4 方向（右上、左上、左下、右下）のチェックができるようにしなさい。「上」や「下」などを参考にして同様に作成すればできると思われる。

- プログラムができたら、p.3 のプログラムの 61 行目のコメントを外し、init メソッドから self.initMovable が実行されるようにしなさい。ここで、self は自分自身、すなわち、init が呼び出されたオブジェクト自身を指している。

この状態で実行してもまだ何も表示されないなので、実行して特にエラーがでなければ OK である。

4 最低限の表示

この状態では、盤の状態が全くわからないので、最低限の表示を行うことにする。ここでは最低限のテキスト表示をすることとし、GUIでの表示は別途扱う予定である。

次ページ以降に示すプログラムは、次のように動作する。まず、

```
abcdefgh  
1  
2  
3  
4   ox  
5   xo  
6  
7  
8
```

次は黒です。石を置く座標を入力してください（例：a1）->

のように、盤の内容を表示し、石を置く座標を聞いてくる。例えば、ここで

次は黒です。石を置く座標を入力してください（例：a1）-> c4

と座標 c4 を入力したとする。すると、c4 に黒石が置かれ、手番が入れ替わって以下のような表示になる。現時点では、ただ石を置くだけであり、挟んだ石をひっくり返すことはしていない（これは次回）。

```
abcdefgh  
1  
2  
3  
4  xox  
5   xo  
6  
7  
8
```

次は白です。石を置く座標を入力してください（例：a1）->

もしひっくり返す石がない座標を指定すると、エラーメッセージが表示され、再度座標の入力が促される。

abcdefgh

```
1
2
3
4   ox
5   xo
6
7
8
```

次は黒です. 石を置く座標を入力してください (例: a1) -> a1

そこには打てません. 知らんけど.

石を置く座標を入力してください (例: a1) ->

以下がプログラムである.

- 1~15 行目の move メソッドは, 指定された座標に石が置けるかどうかをチェックして置けないのであれば false を返す (3~5 行目). そうでない場合, 指定された座標に石を置き (8 行目), 手数を 1 加算し (10 行目), 手番を入れ替え (11 行目), @movableDir を更新する (12 行目).

なお, 7 行目のコメントはそのまま付けておくこと.

- 18 行目以降の loop メソッドは, 盤の表示を行い (20~38 行目), 次の手番を示し (40~46 行目), 座標の入力とその座標に石が打てるかのチェックを行い (48~69 行目), 入力された座標に石を配置する (72 行目). なお, 61 行目の if 文は本来 1 行であるが, スペースの都合で 2 行で記載しているので注意.

Ruby についての補足: 25 行目の "1".ord は, "1" という文字列 (の最初の文字) の文字コードを返す. よって, s には, "1" の文字コードに y-1 が加算された文字コードが s に格納される. 26 行目の s.chr("utf-8") は, s に格納されている文字コードに対応する文字を返す. また, 54 行目の gets.chomp について, gets が標準入力から文字列を読み込むもので, ここではコマンドプロンプトやターミナルから入力された文字列を得る. chomp は文字列の末尾の改行を削除する.

```
1  # 石を置き, ひっくり返す
2  def move(x,y)
3    if @movableDir[x][y] == NONE
4      return false
5    end
6
7  #    self.flipDisks(x,y)
8    @rawBoard[x][y] = @current_color
9
10   @turns += 1
11   @current_color = -1 * @current_color
12   self.initMovable
13
14   return true
```

```

15     end
16
17     # 「盤を描画して、手を入力をしてもらう」のを繰り返す（暫定テキスト版）
18     def loop()
19         while true do
20             print(" abdceefgh\n")
21
22             for y in 1..BOARDSIZE do
23                 for x in 1..BOARDSIZE do
24                     if x == 1
25                         s = "l".ord + y - 1
26                         print(s.chr("utf-8"))
27                     end
28                     if @rawBoard[x][y] == EMPTY
29                         print(" ")
30                     elsif @rawBoard[x][y] == BLACK
31                         print("x")
32                     elsif @rawBoard[x][y] == WHITE
33                         print("o")
34                     end
35                 end
36                 print("\n")
37             end
38             print("\n")
39
40             print("次は")
41             if @current_color == BLACK
42                 print("黒")
43             elsif
44                 print("白")
45             end
46             print("です. ")
47
48             # 入力された座標が正しいかどうかを表す変数
49             isvalid = false
50
51             # 正しい入力 that 得られるまで座標を入力してもらう
52             while !isvalid do
53                 print("石を置く座標を入力してください（例：a1）-> ")
54                 input = gets.chomp
55
56                 if input.length == 2
57                     x = input[0].ord - "a".ord + 1
58                     y = input[1].ord - "1".ord + 1
59
60                     # もし入力された座標が石を打てる場所であれば, isvalid を true にする

```

```

61         if x.between?(1,BOARDSIZE) and y.between?(1,BOARDSIZE)
62             and @movableDir[x][y] != NONE
63             isvalid = true
64         end
65     end
66     if !isvalid
67         print("そこには打てまへんで。知らんけど。 \n")
68     end
69 end
70
71 # 石を打ち、(ひっくり返して) 手番を入れ替える。ただし今回は石を置くだけで、
72 # ひっくり返すのは次回
73 move(x,y)
74 end

```

課題

上のプログラムを追加しなさい。追加する位置は、p.3のプログラムの65行目（コメントがあるところ）である。プログラムはmanabaのkadai2.txtからコピーすればよい。

また、プログラムの末尾（p.4のプログラムの74行目）にある

```
# board.loop
```

のコメント（先頭の#）を外して、loopメソッドが実行されるようにせよ。

プログラムの動作を試し、「石を打てる/打てない」の判定が正しく行われていることを確認せよ。

本日の内容は以上です。お疲れさまでした。出席はresponからご登録ください。