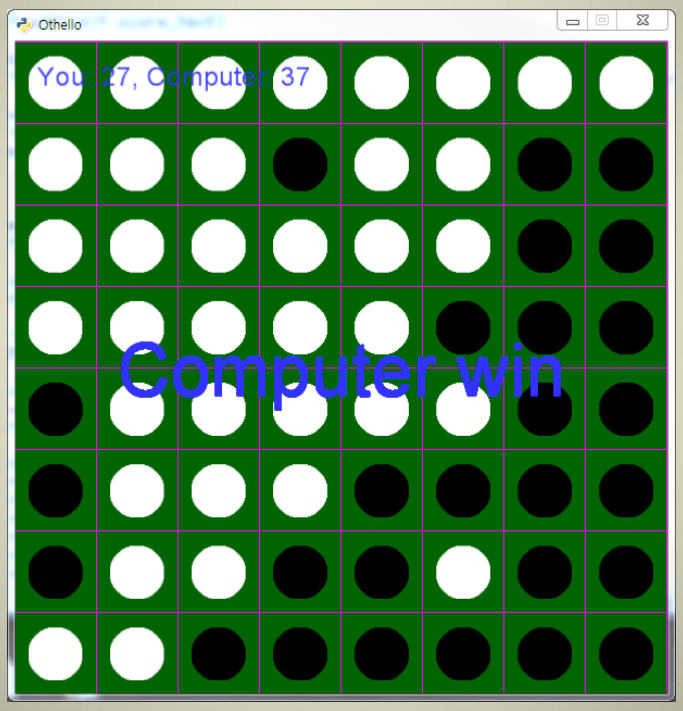
Practice 05 (1)



Practice 05 (2)

```
import numpy as np
class GameLayer(cocos.layer.Layer):
    is_event_handler = True
    PERSON = -1
    COMPUTER = 1
    def __init__(self, difficulty, hud_layer):
        super(GameLayer, self).__init__()
        self.difficulty = difficulty

        self.levelDepth = self.difficulty*2+2
        self.hud = hud_layer
        self.square = 75;          self.row = 8
        self.column = 8;      self.height = 8*self.square
        self.width = 8*self.square
        self.table = \    # board, PERSON, COMPUTER, 0
np.arange(self.row*self.column).reshape(self.row, self.column)
```

Practice 05 (3)

```
        for x in range  (0, self.column+1) :
            line = cocos.draw.Line((x*self.square, 0),
             (x*self.square, self.height), (255, 0, 255, 255))
            self.add(line)
        for y in range  (0, self.row+1) :
            line = cocos.draw.Line((0, y*self.square),
             (self.width, y*self.square), (255, 0, 255, 255))
            self.add(line)

        self.disk = \ # disk sprite
[[None for i in range(self.column)] for j in range(self.row)]
```

Practice 05 (4)

```
        for y in range (0, self.row) :
            for x in range (0, self.column) :
                centerPt = eu.Vector2(x*self.square + \
              self.square/2,y*self.square + self.square/2)
                self.disk[y][x] = \
                    cocos.sprite.Sprite('ball.png', \
                    position = centerPt, color = (255, 255, 255))
                self.add(self.disk[y][x])

        self.setup()
        self.turn = GameLayer.PERSON
        self.schedule(self.update)
```

Practice 05 (5)

```
def setup(self):
    for y in range (0, self.row) :
        for x in range (0, self.column) :
            self.table[y][x] = 0

    self.table[3][3] = GameLayer.PERSON
    self.table[3][4] = GameLayer.COMPUTER
    self.table[4][3] = GameLayer.COMPUTER
    self.table[4][4] = GameLayer.PERSON
```

Practice 05 (6)

```
def update(self, dt):
    computer = 0;        person = 0
    for y in range  (0, self.row) :
        for x in range  (0, self.column) :
            if self.table[y][x] == GameLayer.COMPUTER:
                self.disk[y][x].color = (255, 255, 255)
                self.disk[y][x].visible = True
                computer += 1
            elif self.table[y][x] == GameLayer.PERSON:
                self.disk[y][x].color = (0, 0, 0)
                self.disk[y][x].visible = True
                person += 1
            else:
                self.disk[y][x].visible = False
    # decision on outcome
```

Practice 05 (7)

```
def isPossiable(self, x, y, turn, board): # check position
    …
def computer(self): # computer turn
    …
def minimax(self, player): # minimax
    …
def maxMove(self, board, depth, alpha, beta):
    …
def minMove(self, board, depth, alpha, beta):
    …
def boardScore(self, board):
    …
def getMoves(self, turn, board): # move list
    …
```

Practice 05 (8)

```
def isPossiable(self, x, y, turn, board):
     rtnList = list()
     if board[y][x] != 0: return rtnList # table
     for dirX in range(-1, 2):
         for dirY in range(-1,2):
             if dirX == 0 and dirY == 0: continue
             if(x+dirX < 0 or x+dirX >= self.column): continue
             if(y+dirY < 0 or y+dirY >= self.row): continue
             xList = list();    yList = list()
             if dirX == 0:
                 for yy in range(y+dirY*2, self.row*dirY,dirY):
                     if(yy < 0 or yy >= self.row): break
                     xList.append(x)
                     yList.append(yy)
             …
```

Practice 05 (9)

```
def isPossiable(self, x, y, turn, board):
    …
    for dirX in range(-1, 2):
        for dirY in range(-1,2):
            …
            bDetected = False
            revList = []
            if board[y+dirY][x+dirX] == turn*-1:
                revList.append((x+dirX, y+dirY))
                for xx, yy in zip(xList, yList):
                    # check xx, yy
                    # check board[yy][xx]
                if(bDetected == False): revList = []
            rtnList += revList;
    return rtnList
```


Practice 05 (10)

```
x = np.array([1, 2, 3])
y = x
z = np.copy(x)
x[0] = 10
x[0] == y[0] # True, shallow copy
x[0] == z[0] # False, deep copy
```


Practice 05 (11)

```
def minimax(self, player):
    … # get move list
    alpha = float("-inf")
    beta = float("inf")
    for i, move in enumerate(moves): # move list
        boardCopy = self.Move(…, GameLayer.COMPUTER,
          np.copy(self.table))
        scores[i] = self.maxMove(boardCopy, 1, alpha, beta)

    maxIndex = np.argmax(scores)
    return moves[maxIndex]
```

Practice 05 (12)

```
def maxMove(self, board, depth, alpha, beta):
    … # get move list, scores=np.zeros(length of list)
    if len(moves)==0:
        if depth<=self.levelDepth:
            return self.minMove(board, depth+1, alpha, beta)
        else:  # return score
    for i, move in enumerate(moves):
        boardCopy = self.Move(…, turn, np.copy(board))
        if depth>=self.levelDepth:     # scores[i] = …
        else:
            scores[i] = self.minMove(boardCopy, depth+1,
                alpha, beta)
            if scores[i] > alpha:    alpha = scores[i]
            if beta <= alpha:        return scores[i]
    return max(scores)
```

Practice 05 (13)

```
def minMove(self, board, depth, alpha, beta):
    … # get move list, scores=np.zeros(length of list)
    if len(moves)==0:
        if depth<=self.levelDepth:
            return self.maxMove(board, depth+1, alpha, beta)
        else:  # return score
    for i, move in enumerate(moves):
        boardCopy = self.Move(…)
        if depth>=self.levelDepth:     # scores[i] = …
        else:
            scores[i] = self.maxMove(boardCopy, depth+1,
                alpha, beta)
            if beta > scores[i]:     beta = scores[i]
            if beta <= alpha:        return scores[i]
    return min(scores)
```