#Thanh Vu #17627579 #UCnetid: thanhhv ICS 32 Spring 2014 Lab 4

```
Black_player= 'B'
White_player= 'W'
none=' '

direction_list=[[-1,0],[-1,-1],[0,-1],[1,-1],[1,0],[1,1],[0,1],[-1,1]]

class Invalidothello_move(Exception):
pass

class gameover_othello_error(Exception):
pass


class Othello_game_state:

def __init__(self,column,row,turn:'black or white'):
# Initilize an empty column * row board game
self.game_board_column = column
self.game_board_row = row

self.game_turn = turn
self.valid_move=[]

self.game_board = []
for i in range(row):
self.game_board.append([])
for i in range(column):
self.game_board[-1].append(none)




def pieces_at_game_begin(self,player,other_player):
self.game_board[int(self.game_board_row/2)-1][int(self.game_board_column/2)-1]= player
self.game_board[int(self.game_board_row/2)-1][int(self.game_board_column/2)]= other_player
self.game_board[int(self.game_board_row/2)][int(self.game_board_column/2)-1]= other_player
self.game_board[int(self.game_board_row/2)][int(self.game_board_column/2)]= player
return self.game_board

def copy_board(self, current_board):
#representation of the game with score (better table format)
board_copy=[]
```

```
for row in range(self.game_board_row):
board_copy.append([])
for column in range(self.game_board_column):
board_copy[-1].append(current_board[row][column])
self.current_score_black,self.current_score_white = self.total_score()
self.print_board()

return board_copy

def print_board(self):
print("CURRENT SCORE")
print(' BLACK: {} points'.format(self.current_score_black))
print(' WHITE: {} points'.format(self.current_score_white))
print('\n')
print(' {} player turn '.format(self.game_turn))
print('\n')
for i in range(1, self.game_board_column + 1):
print(' ', i, end= '')
print()
for i in range(0, self.game_board_row):
print(i+1 ,"", self.game_board[i])

def opposite_turn(self):
# switch to other player's turn after making a move and update the board.
if self.game_turn == Black_player:
self.game_turn = White_player
elif self.game_turn == White_player:
self.game_turn = Black_player

return self.game_turn

def convert_two_player(self):
self.other_piece= White_player
if self.game_turn == Black_player:
self.other_piece = White_player
elif self.game_turn == White_player:
self.other_piece = Black_player

return self.other_piece
def location_same_color_piece(self):
self.location_list=[]
for row in range(self.game_board_row):
for column in range(self.game_board_column):
if self.game_board[row][column]== self.game_turn:
self.location_list.append([row,column])
```

```
def adjacent_spot_flip_list_move(self):
self.other_piece = self.convert_two_player()
#return a list with coordinate adjacent of a particular spot on board_game
adjacent_list =[]
self.piece_flip_list=[]
self.valid_move=[]
for x_dir, y_dir in direction_list:
for each_spot in self.location_list:
self.adj_row = each_spot[0] + x_dir
self.adj_column = each_spot[1] + y_dir
if self.valid_board_position(self.adj_row,self.adj_column) and
self.game_board[self.adj_row][self.adj_column] == self.other_piece:
# check if the adjacent piece is on board and different color
adjacent_list.append([self.adj_row,self.adj_column])


# if next piece is that direction is that same different color, check the next adjacent spot in same
direction if they are same
while self.game_board[self.adj_row][self.adj_column] == self.other_piece:
self.adj_row = self.adj_row + x_dir
self.adj_column = self.adj_column + y_dir
if not self.valid_board_position(self.adj_row,self.adj_column): # check if it reach the edge of the
board
break

# if the next piece in that direction is a blank, append it to the valid_move_list
if self.valid_board_position(self.adj_row,self.adj_column):
if self.game_board[self.adj_row][self.adj_column] == none:
self.valid_move.append([self.adj_row,self.adj_column])

elif self.game_board[self.adj_row][self.adj_column] == self.game_turn:
# two pieces cap at two end, there are pieces to flip over in between
# need to go in reverse from x,y till we get to the other end of the same piece
while True:
self.adj_row = self.adj_row - x_dir
self.adj_column = self.adj_column - y_dir

if self.game_board[self.adj_row][self.adj_column] == self.game_turn:
break
self.piece_flip_list.append([self.adj_row,self.adj_column])

#uncomment to see the make_move list , flip_list
print ('Here is the pieces to flip:', [[j + 1 for j in i] for i in self.piece_flip_list])

print('Here is the valid spot to make a move:', [[j + 1 for j in i] for i in self.valid_move])
```

```
def valid_board_position(self,row_select,column_select):
return 0 <= row_select < self.game_board_row and 0 <= column_select <
self.game_board_column

def make_a_move(self,row_select,column_select):
# check the user_select and add a piece with same color to the new valid move spot
# flip the pieces to the same color in between & update the board with pieces flipped & return
updated board
while True:
if [row_select ,column_select ] in self.valid_move:
self.game_board[row_select][column_select] = self.game_turn
self.adjacent_spot_flip_list_move()
current_board = self.flip_pieces()
return current_board
else:
raise Invalidothello_move

def flip_pieces (self):
# flip all the pieces in the list
for pieces in self.piece_flip_list:
self.flip_error()
self.game_board[pieces[0]][pieces[1]] = self.game_turn
return self.game_board

def total_score(self):
total_score_black= 0
total_score_white= 0
for row in range(self.game_board_row):
for column in range(self.game_board_column):
if self.game_board[row][column] == Black_player:
total_score_black += 1
elif self.game_board[row][column] == White_player:
total_score_white += 1
return (total_score_black,total_score_white)

def winner_check_most_points(self):
winner= Black_player
if self.game_over_check():
if self.current_score_black > self.current_score_white:
winner = Black_player
elif self.current_score_white > self.current_score_black:
winner = White_player
return winner

def winner_check_least_points(self):
winner=Black_player
```

```
if self.game_over_check():
if self.current_score_black > self.current_score_white:
winner = White_player
elif self.current_score_white > self.current_score_black:
winner = Black_player
return winner



def game_over_check(self):
# check if valid move list is empty
if len(self.valid_move) == 0:
return True
else:
return False

def flip_error(self):
if len(self.piece_flip_list)== 0:
raise Invalidothello_move
```

```
#Thanh Vu #17627579 #UCnetid: thanhhv ICS 32 Spring 2014 Lab 4
import othello_logic


def main():
column_board, row_board = user_input_column_row()
turn = first_player_move_turn()

game_state = othello_logic.Othello_game_state(column_board,row_board,turn)

# default 4 pieces of the game at the beginning
player, other_player = top_left_player_piece()
win_method = method_to_win_rule()
current_board = game_state.pieces_at_game_begin(player,other_player)
user_interface(game_state,current_board,win_method,row_board,column_board,turn)

def user_interface(game_state,current_board,win_method,row_board, column_board,turn):
while True:
try:
game_state.copy_board(current_board)
game_state.location_same_color_piece()


# game_state.selected_one_grid(3,3)
game_state.adjacent_spot_flip_list_move()
if win_method == 'A':
if game_state.game_over_check() == True:
game_state.copy_board(current_board)
winner = game_state.winner_check_most_points()
print(winner,'is the winner')
quit()
elif win_method == 'B':
if game_state.game_over_check() == True:
game_state.copy_board(current_board)
winner = game_state.winner_check_least_points()
print(winner,'is the winner')
quit()

row_select, column_select = user_input(row_board,column_board)

current_board = game_state.make_a_move(row_select,column_select)
game_state.opposite_turn()
except:
print('Please Try again')
```

```
def user_input (board_row, board_col):
while True:
try:
row_input = int(input("Select the row: "))-1
column_input = int(input("Select the column: "))-1
if 0 <= row_input < board_row and 0 <= column_input < board_col:
return (row_input,column_input)
except:
print("Not a valid column/row number. Please try again")
def user_input_column_row():
# row and column is even integer from 4-16, ask user again if invalid input
while True:
column_input = int(input("Please enter even interger from 4-16 for column of the board game:
"))
row_input = int(input("Please enter even interger from 4-16 for row of the board game: "))
if 4 <= column_input <= 16 \
and 4 <= row_input <= 16\
and column_input % 2 == 0 \
and row_input % 2 == 0:

return (column_input,row_input)
else:
print("nope")

def first_player_move_turn():
while True:
player_move_first = input("Please indicate which player will move first (B or W): ").strip()
if player_move_first == 'B' or player_move_first == 'W':
return (player_move_first)
else:
print("not a valid command for player's turn.")

def top_left_player_piece():
while True:
piece_input = input("Please indicate the color of the piece at the top left of the center of the
board game (B or W): ")
if piece_input == 'B':
player = piece_input
other_player = 'W'
return (player,other_player)
elif piece_input == 'W':
player = piece_input
other_player = 'B'
return (player,other_player)
else:
print("Try a valid command for color of the piece")
```

```
def method_to_win_rule():
while True:
win_input= input('''
Please Select A: player with most pieces on the board at the end win
B: player with least pieces on the board at the end win
''')
if win_input == 'A': # access the rule with score counts the most pieces win
return win_input
elif win_input =='B': # access the rule with score counts least pieces win
return win_input


if __name__ == '__main__':
main()
```