```
#Initialize board

board = []

for i in range(64):
    board.append(0)

class Pawn:
    def __init__(self, colour, moves, moveCount) -> None:
        self.colour = colour
        self.moves = moves
        self.moveCount = moveCount

    def __repr__(self) -> str:
        if self.colour == 0:
            return "P"
        else:
            return "p"

    def setPos(self, position):
        self.position = position

    def addMove(self, move):
        self.moves.append(move)

    def isMoveLegal(self, destination):
        if self.colour == 0:
            if board[destination] == 0:
                if (destination - self.position)/8 == 1:
                    return True
                elif (destination - self.position)/8 == 2 and
self.moveCount == 0:
                    return True
            elif board[destination].colour == 1:
                if (destination - self.position) == 7:
                    return True
                elif (destination - self.position) == 9:
                    return True
        elif self.colour == 1:
            if board[destination] == 0:
                if (destination - self.position)/8 == -1:
                    return True
                elif (destination - self.position)/8 == -2 and
self.moveCount == 0:
                    return True
            elif board[destination].colour == 0:
                if (destination - self.position) == -7:
```

```
                    return True
                elif (destination - self.position) == -9:
                    return True

    #def canMove():

class Knight:
    def __init__(self, colour, moves, moveCount) -> None:
        self.colour = colour
        self.moves = moves
        self.moveCount = moveCount

    def __repr__(self) -> str:
        if self.colour == 0:
            return "N"
        else:
            return "n"

    def setPos(self, position):
        self.position = position

    def addMove(self, move):
        self.moves.append(move)

    def isMoveLegal(self, destination):
        if self.colour == 0:
            if board[destination] == 0 or board[destination].colour == 1:
                if (destination - self.position)/15 == 1 or (destination -
self.position)/15 == -1:
                    return True
                elif (destination - self.position)/17 == 1 or (destination
- self.position)/17 == -1:
                    return True
                elif (destination - self.position)/10 == 1 or (destination
- self.position)/10 == -1:
                    return True
                elif (destination - self.position)/6 == 1 or (destination
- self.position)/6 == -1:
                    return True
        elif self.colour == 1:
            if board[destination] == 0 or board[destination].colour == 0:
                if (destination - self.position)/15 == 1 or (destination -
self.position)/15 == -1:
                    return True
                elif (destination - self.position)/17 == 1 or (destination
- self.position)/17 == -1:
                    return True
```

```
                elif (destination - self.position)/10 == 1 or (destination
- self.position)/10 == -1:
                    return True
                elif (destination - self.position)/6 == 1 or (destination
- self.position)/6 == -1:
                    return True

    #def canMove():

class Bishop:
    def __init__(self, colour, moves, moveCount) -> None:
        self.colour = colour
        self.moves = moves
        self.moveCount = moveCount

    def __repr__(self) -> str:
        if self.colour == 0:
            return "B"
        else:
            return "b"

    def setPos(self, position):
        self.position = position

    def addMove(self, move):
        self.moves.append(move)

    def isMoveLegal(self, destination):
        if self.colour == 0:
            if board[destination] == 0 or board[destination].colour == 1:
                if (destination - self.position)%7 == 0:
                    return True
                elif (destination - self.position)%9 == 0:
                    return True
        elif self.colour == 1:
            if board[destination] == 0 or board[destination].colour == 0:
                if (destination - self.position)%7 == 0:
                    return True
                elif (destination - self.position)%9 == 0:
                    return True

    #def canMove():

class Rook:
    def __init__(self, colour, moves, moveCount) -> None:
        self.colour = colour
        self.moves = moves
```

```
        self.moveCount = moveCount

    def __repr__(self) -> str:
        if self.colour == 0:
            return "R"
        else:
            return "r"

    def setPos(self, position):
        self.position = position

    def addMove(self, move):
        self.moves.append(move)

    def isMoveLegal(self, destination):
        if self.colour == 0:
            if board[destination] == 0 or board[destination].colour == 1:
                if ((self.position)//8)*8 <= destination <=
((((self.position)//8)*8) + 7):
                    return True
                elif (destination - self.position)%8 == 0:
                    return True
        elif self.colour == 1:
            if board[destination] == 0 or board[destination].colour == 0:
                if ((self.position)//8)*8 <= destination <=
((((self.position)//8)*8) + 7):
                    return True
                elif (destination - self.position)%8 == 0:
                    return True

    #def canMove():

class Queen:
    def __init__(self, colour, moves, moveCount) -> None:
        self.colour = colour
        self.moves = moves
        self.moveCount = moveCount

    def __repr__(self) -> str:
        if self.colour == 0:
            return "Q"
        else:
            return "q"

    def setPos(self, position):
        self.position = position
```

```
    def addMove(self, move):
        self.moves.append(move)

    def isMoveLegal(self, destination):
        if self.colour == 0:
            if board[destination] == 0 or board[destination].colour == 1:
                if ((self.position)//8)*8 <= destination <=
((((self.position)//8)*8) + 7):
                    return True
                elif (destination - self.position)%8 == 0:
                    return True
                elif (destination - self.position)%7 == 0:
                    return True
                elif (destination - self.position)%9 == 0:
                    return True

        elif self.colour == 1:
            if board[destination] == 0 or board[destination].colour == 0:
                if ((self.position)//8)*8 <= destination <=
((((self.position)//8)*8) + 7):
                    return True
                elif (destination - self.position)%8 == 0:
                    return True
                elif (destination - self.position)%7 == 0:
                    return True
                elif (destination - self.position)%9 == 0:
                    return True


    #def canMove():

class King:
    def __init__(self, colour, moves, moveCount) -> None:
        self.colour = colour
        self.moves = moves
        self.moveCount = moveCount

    def __repr__(self) -> str:
        if self.colour == 0:
            return "K"
        else:
            return "k"

    def setPos(self, position):
        self.position = position

    def isMoveLegal(self, destination):
```

```
        if self.colour == 0:
            if board[destination] == 0 or board[destination].colour == 1:
                if (destination - self.position)/8 == 1:
                    return True
                if (destination - self.position)/8 == -1:
                    return True
                if (destination - self.position)/7 == 1:
                    return True
                if (destination - self.position)/7 == -1:
                    return True
                if (destination - self.position)/9 == 1:
                    return True
                if (destination - self.position)/9 == -1:
                    return True
                if (destination - self.position) == 1:
                    return True
                if (destination - self.position) == -1:
                    return True

        elif self.colour == 1:
            if board[destination] == 0 or board[destination].colour == 1:
                if (destination - self.position)/8 == 1:
                    return True
                if (destination - self.position)/8 == -1:
                    return True
                if (destination - self.position)/7 == 1:
                    return True
                if (destination - self.position)/7 == -1:
                    return True
                if (destination - self.position)/9 == 1:
                    return True
                if (destination - self.position)/9 == -1:
                    return True
                if (destination - self.position) == 1:
                    return True
                if (destination - self.position) == -1:
                    return True

    #def canMove():

Wpawn1 = Pawn(0, [], 0)
Wpawn2 = Pawn(0, [], 0)
Wpawn3 = Pawn(0, [], 0)
Wpawn4 = Pawn(0, [], 0)
Wpawn5 = Pawn(0, [], 0)
Wpawn6 = Pawn(0, [], 0)
Wpawn7 = Pawn(0, [], 0)
```

```
Wpawn8 = Pawn(0, [], 0)
Wknight1 = Knight(0, [], 0)
Wknight2 = Knight(0, [], 0)
Wbishop1 = Bishop(0, [], 0)
Wbishop2 = Bishop(0, [], 0)
Wrook1 = Rook(0, [], 0)
Wrook2 = Rook(0, [], 0)
Wqueen = Queen(0, [], 0)
Wking = King(0, [], 0)

Bpawn1 = Pawn(1, [], 0)
Bpawn2 = Pawn(1, [], 0)
Bpawn3 = Pawn(1, [], 0)
Bpawn4 = Pawn(1, [], 0)
Bpawn5 = Pawn(1, [], 0)
Bpawn6 = Pawn(1, [], 0)
Bpawn7 = Pawn(1, [], 0)
Bpawn8 = Pawn(1, [], 0)
Bknight1 = Knight(1, [], 0)
Bknight2 = Knight(1, [], 0)
Bbishop1 = Bishop(1, [], 0)
Bbishop2 = Bishop(1, [], 0)
Brook1 = Rook(1, [], 0)
Brook2 = Rook(1, [], 0)
Bqueen = Queen(1, [], 0)
Bking = King(1, [], 0)

def startingPos():
    board[8] = Wpawn1
    board[9] = Wpawn2
    board[10] = Wpawn3
    board[11] = Wpawn4
    board[12] = Wpawn5
    board[13] = Wpawn6
    board[14] = Wpawn7
    board[15] = Wpawn8
    board[0] = Wrook1
    board[1] = Wknight1
    board[2] = Wbishop1
    board[3] = Wqueen
    board[4] = Wking
    board[5] = Wknight2
    board[6] = Wbishop2
    board[7] = Wrook2

    board[48] = Bpawn1
    board[49] = Bpawn2
```

```
board[50] = Bpawn3
board[51] = Bpawn4
board[52] = Bpawn5
board[53] = Bpawn6
board[54] = Bpawn7
board[55] = Bpawn8
board[56] = Brook1
board[57] = Bknight1
board[58] = Bbishop1
board[59] = Bqueen
board[60] = Bking
board[61] = Bknight2
board[62] = Bbishop2
board[63] = Brook2

Wpawn1.setPos(8)
Wpawn2.setPos(9)
Wpawn3.setPos(10)
Wpawn4.setPos(11)
Wpawn5.setPos(12)
Wpawn6.setPos(13)
Wpawn7.setPos(14)
Wpawn8.setPos(15)
Wrook1.setPos(0)
Wknight1.setPos(1)
Wbishop1.setPos(2)
Wqueen.setPos(3)
Wking.setPos(4)
Wbishop2.setPos(5)
Wknight2.setPos(6)
Wrook2.setPos(7)

Bpawn1.setPos(48)
Bpawn2.setPos(49)
Bpawn3.setPos(50)
Bpawn4.setPos(51)
Bpawn5.setPos(52)
Bpawn6.setPos(53)
Bpawn7.setPos(54)
Bpawn8.setPos(55)
Brook1.setPos(56)
Bknight1.setPos(57)
Bbishop1.setPos(58)
Bqueen.setPos(59)
Bking.setPos(60)
Bbishop2.setPos(61)
Bknight2.setPos(62)
```

```
    Brook2.setPos(63)

WattackList = []
BattackList = []

def makeWattackList(attack):
    attack.append()

def move(piece, dest):
    if piece.isMoveLegal(dest):
        if board[dest] != 0 and board[dest].colour != piece.colour:
            board[dest].position = -983
        board[piece.position] = 0
        board[dest] = piece
        piece.position = dest
        piece.addMove(dest)
        piece.moveCount+=1

#Testing
startingPos()
move(Wpawn5, 28)
move(Bpawn4, 35)
move(Wpawn5, 35)
move(Bpawn4, 27)
print(board, Wpawn5.position, Bpawn4.position)
```