data = open("test-int.txt").readlines()
game = [i.split() for i in data]
grid = [[0 for col in range(10)] for row in range(10)]
grid[0][4], grid[0][5] = 1, 52
grid[1][4], grid[1][5] = 2, 51
grid[2][0], grid[3][0], grid[4][0], grid[5][0] = 7, 8, 17, 18
grid[2][5], grid[3][5], grid[4][5], grid[5][5] = 50, 41, 40, 31
for row in range(2, 10):
    for col in range(1, 10):
        if col != 5:
            if row in [2, 4]:
                grid[row][col] = grid[row][col - 1] + 1
            if row in [3, 5]:
                grid[row][col] = grid[row][col - 1] - 1
            if col == 4 and row > 5:
                grid[row][col] = grid[row - 1][col] - 1
            if col == 5 and row > 5:
                grid[row][col] = grid[row - 1][col] + 1
occupied = [[' ' for col in range(10)] for row in range(10)]
for row in range(10):
    for col in range(10):
        if grid[row][col] != 0:
            occupied[row][col] = False

h_to_v = [[6, 7, 8], [11, 12, 13], [16, 17, 18], [21, 22, 23], [26, 27, 28], [34, 35, 36], [39, 40, 41], [44, 45, 46], [49, 50, 51]]
def made_H_then_V(small, big):
    for i in range(len(big) - len(small) + 1):
        for j in range(len(small)):
            if big[i + j] != small[j]:
                break
        else:
            return True
    return False
def isPrime(num):
    if num < 2 or (num > 2 and num % 2 == 0):
        return False
    for x in range(3, num, 2):
        if num % x == 0:
            return False
    return True
def newPosition(num, grid, g, occupied):
    location = 0
for i in g:
    if i[1] == num:
        location = i[0]
        x,y = location[0], location[1]
occupied[x][y] = True

def changePosition(num, grid, g, occupied):
    location = 0
    for i in g:
        if i[1] == num:
            location = i[0]
            x,y = location[0], location[1]
occupied[x][y] = False

def isOccupied(num, grid, g, occupied):
    location = 0
    for i in g:
        if i[1] == num:
            location = i[0]
            x,y = location[0], location[1]
    return occupied[x][y]

n = 0
while n < len(game):
    occupied = [["   "] for col in range(10)]
    for row in range(10):
        for col in range(10):
            if grid[row][col] != 0:
                occupied[row][col] = False
    opponent_markers = [int(i) for i in game[n][0:3]]
    player_markers = [int(i) for i in game[n][3:6]]
    dieRolls = [int(i) for i in game[n][7:]]
    for x in opponent_markers:
        newPosition(x, grid, g, occupied)
    for y in player_markers:
        newPosition(y, grid, g, occupied)
    for roll in range(len(dieRolls)):
        a = min(player_markers)
        N = a + dieRolls[roll]
        if isOccupied(N, grid, g, occupied):
            continue
        if N > 52:
            continue
        changePosition(a, grid, g, occupied)
        newPosition(N, grid, g, occupied)
        player_markers.append(N)
        player_markers.remove(a)
    if N == 52:
        changePosition(N, grid, g, occupied)
        player_markers.remove(N)
        clear_path = False
        count = 0
    if isPrime(N):
        for num in range(N+1, (N+6)+1):
            check = isOccupied(num, grid, g, occupied)
            if check:
                changePosition(N, grid, g, occupied)
                newPosition(num-1, grid, g, occupied)
                player_markers.remove(N)
                player_markers.append(num-1)
                break
            else:
                count += 1
if count == 6:
    clear_path = True

if clear_path:
    changePosition(N, grid, g, occupied)
    newPosition(N+6, grid, g, occupied)
    player_markers.remove(N)
    player_markers.append(N+6)

clear_path = False
count = 0

if N in perfsquares:
    for num in range(N-1, (N-6)-1, -1):
        check = isOccupied(num, grid, g, occupied)
        if check:
            changePosition(N, grid, g, occupied)
            newPosition(num+1, grid, g, occupied)
            player_markers.remove(N)
            player_markers.append(num+1)
        break
    else:
        count += 1
if count == 6:
    clear_path = True

if clear_path:
    changePosition(N, grid, g, occupied)
    newPosition(N-6, grid, g, occupied)
    player_markers.remove(N)
    player_markers.append(N-6)

if not isPrime(N) and N not in perfsquares:
    path = []
    for num in range(a, N+1):
        path.append(num)

hor_to_vert = False
for p in h_to_v:
    if made_H_then_V(p, path):
        hor_to_vert = True
    break

candidates = []

if hor_to_vert:
    if N % dieRolls[roll] == 0:
        continue
    for num in path[1:-1]:
        if num % dieRolls[roll] == 0:
            if not isOccupied(num, grid, g, occupied):
                candidates.append(num)

if len(candidates) < 1:
    changePosition(N, grid, g, occupied)
    newPosition(a, grid, g, occupied)
    player_markers.remove(N)
    player_markers.append(a)
else:
    best_option = max(candidates)
    changePosition(N, grid, g, occupied)
    newPosition(best_option, grid, g, occupied)
    player_markers.remove(N)
    player_markers.append(best_option)

n += 1

print(f"n. {sorted(player_markers)}")