```
//TylerKim
//Intermediate -Patolli

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class TylerKim_INTc4Patolli {

    public static void main(String args[]) throws FileNotFoundException {
        Scanner input = new Scanner(new
        File("C:\\Users\\tyler\\dev\\TylerKimJavaHomework\\Competition\\src\\TylerKim_INTc4Patoll
        iInput"));

        while(input.hasNext()) {
            String inputValue = input.nextLine();

            //get starting location
            int[][] board = createBoard();
            ArrayList<int[]> inputValues = findInput(inputValue);
            int[] playerMarkerLocations = inputValues.get(0);
            int[] capturedMarkerLocations = inputValues.get(1);
            int numberOfDiceMoves = inputValues.get(2)[0];
            int[] diceResults = inputValues.get(3);

            ArrayList<Integer> output = new ArrayList<>();

            //run until there are no more moves left
            for(int i = 0; i < numberOfDiceMoves; i++) {
                int lowestValueIndex = indexOfLowestValue(playerMarkerLocations);
                int lowestValue = playerMarkerLocations[lowestValueIndex];

                //negate turn and move if destination is unavailable
                if(!hitsOccupiedPosition(lowestValue, playerMarkerLocations,
                capturedMarkerLocations, diceResults[i], board)) {
                    //get destination after move
                    int destination = lowestValue + diceResults[i];

                    if(isPrimeNumber(destination)) {
                        destination = primeRule(playerMarkerLocations, capturedMarkerLocations,
                        destination);

                    } else if(isPerfectSquareNumber(destination)) {
                        destination = perfectSquareRule(playerMarkerLocations,
                        capturedMarkerLocations, destination);

                    } else {
                        if(movedHorizontalToVertical(lowestValue, destination, board,
                        diceResults[i])) {
                            destination = nonSquareRootOrPrimeRule(destination, diceResults[i],
                            playerMarkerLocations, capturedMarkerLocations);
                        }
                    }

                    playerMarkerLocations[lowestValueIndex] = destination;
                    //negation

                } else {

                }
            }
            ArrayList<Integer> markerLocation = convertArrToArrayList(playerMarkerLocations);

            markerLocation = purgeEndGame(markerLocation);

            Collections.sort(markerLocation);
```

```
        System.out.println(markerLocation);
    }


}



//setup methods
    //initialization
public static ArrayList<int[]> findInput(String inputValue) {
    String[] inputArray = inputValue.split(" ");

    int[] player1MarkerLocation = new int[3];
    int[] capturedMarkerLocation = new int[3];
    int[] diceResults = new int[inputArray.length - 7];
    int[] numberOfDiceMoves = new int[1];

    ArrayList<int[]> finalArray = new ArrayList<>();

    //opponent marker
    for (int i = 0; i < 3; i++) {
        capturedMarkerLocation[i] = Integer.parseInt(inputArray[i]);
    }

    //player markers
    for (int i = 0; i < 3; i++) {
        player1MarkerLocation[i] = Integer.parseInt(inputArray[i + 3]);
    }

    numberOfDiceMoves[0] = Integer.parseInt(inputArray[6]);

    //moveList
    for (int i = 0; i < inputArray.length - 7; i++) {
        diceResults[i] = Integer.parseInt(inputArray[i + 7]);
    }


    finalArray.add(player1MarkerLocation);
    finalArray.add(capturedMarkerLocation);
    finalArray.add(numberOfDiceMoves);
    finalArray.add(diceResults);

    return finalArray;

}

    //create board
public static int[][] createBoard() {
    int[][] board = {
            {0, 0, 0, 0, 1, 52, 0, 0, 0, 0},
            {0, 0, 0, 0, 2, 51, 0, 0, 0, 0},
            {7, 6, 5, 4, 3, 50, 49, 48, 47, 46},
            {8, 9, 10, 11, 12, 41, 42, 43, 44, 45},
            {17, 16, 15, 14, 13, 40, 39, 38, 37, 36},
            {18, 19, 20, 21, 22, 31, 32, 33, 34, 35},
            {0, 0, 0, 0, 23, 30, 0, 0, 0, 0},
            {0, 0, 0, 0, 24, 29, 0, 0, 0, 0},
            {0, 0, 0, 0, 25, 28, 0, 0, 0, 0,},
            {0, 0, 0, 0, 26, 27, 0, 0, 0, 0}};

    return board;
}



//game mechanics
    //test if occupied location is true
```

```
public static boolean hitsOccupiedPosition(int lowestValue, int[] playerMarkers, int[]
capturedMarkers, int diceRoll, int[][] board) {
   for(int i = 0; i < playerMarkers.length; i++) {
       if((lowestValue + diceRoll) == playerMarkers[i]) {
           return true;
       }
   }

   for(int i = 0; i < capturedMarkers.length; i++) {
       if((lowestValue+diceRoll) == capturedMarkers[i]) {
           return true;
       }
   }

    return false;
}


    //if it is prime move 6 forward unless unable to
public static int primeRule(int[] playerPositions, int[] capturedPositions, int
destinationValue) {

        //run 6 times
    for(int x = 0; x < 6; x++) {

            //player position
        for(int i = 0; i < playerPositions.length; i++) {

            if((destinationValue + 1) == playerPositions[i]) {
                return destinationValue;

            }
        }

            //captured positions
        for(int i = 0; i < capturedPositions.length; i++) {

            if((destinationValue + 1) == capturedPositions[i]) {
                return destinationValue;

            }
        }

        destinationValue++;
    }

    return destinationValue;
}

    //if it is perfect square move 6 back unless unable to
public static int perfectSquareRule(int[] playerPositions, int[] capturedPositions, int
destinationValue) {


    for(int i = 0; i < 6; i++) {

        for(int x = 0; x < playerPositions.length; x++) {

            if((destinationValue - 1) == playerPositions[x]) {
                return destinationValue;
            }
        }

        for(int y = 0; y < capturedPositions.length; y++) {

            if((destinationValue - 1) == capturedPositions[y]) {
                return destinationValue;
            }
        }
```

```
        destinationValue--;

    }
    return destinationValue;
}

    //find next location
public static int nonSquareRootOrPrimeRule(int destination, int diceRoll, int[]
playerLocations, int[] capturedLocations) {

    outerLoop:
    for(int i = destination; i > (destination - diceRoll); i--) {
        if(findMultipleOfNumber(i, diceRoll)) {


            for(int x = 0; x < playerLocations.length; x++) {
                if(i == playerLocations[x]) {
                    continue outerLoop;
                }
            }


            for(int y = 0; y < capturedLocations.length; y++) {
                if(i == capturedLocations[y]) {
                    continue outerLoop;
                }
            }

            destination = i;
            return destination;
        }

    }

    return destination - diceRoll;
}



//supplementary
    //find multiple of number
public static boolean findMultipleOfNumber(int testedMultiple, int multiplierValue) {

    if(testedMultiple % multiplierValue == 0) {
        return true;
    }

    return false;
}


    //find lowest value
public static int indexOfLowestValue(int[] initialLocations) {

    int minValue = initialLocations[0];

    for (int i = 1; i < initialLocations.length; i++) {
        if (minValue > initialLocations[i]) {
            minValue = initialLocations[i];
        }
    }

    return indexOfNumberInArr(minValue, initialLocations);
}

    //find index of a number in an array
public static int indexOfNumberInArr(int x, int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == x) {
```

```
            return i;
        }
    }

    return 0;
}

    //prime number tester
public static boolean isPrimeNumber(int x) {
    for (int i = 2; i < x; i++) {
        if (x % i == 0) {
            return false;
        }
    }
    return true;
}

    //perfect square number tester
public static boolean isPerfectSquareNumber(int x) {
    if ((x % Math.sqrt(x) == 0) && x > 6) {
        return true;
    }
    return false;
}

    //get location of number
public static int[] indexOfNumber(int number, int[][] board) {
    int[] output = new int[2];

    outerloop:
    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board.length; j++) {
            if (board[i][j] == number) {
                output[0] = i;
                output[1] = j;
                break outerloop;
            }
        }
    }

    return output;
}

    //convert array to arraylist
public static ArrayList<Integer> convertArrToArrayList(int[] arr) {
    ArrayList<Integer> output = new ArrayList<>();

    for(int i = 0; i < arr.length; i++) {
        output.add(arr[i]);
    }

    return output;
}

//clean any arraylist greater than 52
public static ArrayList<Integer> purgeEndGame(ArrayList<Integer> playerMarkerLocations) {

    for(int i = 0; i < playerMarkerLocations.size(); i++) {
        if(playerMarkerLocations.get(i) >= 52) {
            playerMarkerLocations.remove(i);
        }
    }

    return playerMarkerLocations;
}

//detect if one horizontal followed by a vertical move has occurred
public static boolean movedHorizontalToVertical(int previousValue, int finalValue, int[][]
board, int diceValue) {
```

```
        int changeInI = Math.abs(indexOfNumber(previousValue, board)[0] -
        indexOfNumber(finalValue, board)[0]);
        int changeInJ = Math.abs(indexOfNumber(previousValue, board)[1] -
        indexOfNumber(finalValue, board)[1]);


        if (diceValue > 1) {

            if(changeInI == 0 || changeInJ == 0) {
                return false;
            }

            for(int i = previousValue; i < finalValue; i++) {
                changeInI = Math.abs(indexOfNumber(i, board)[0] - indexOfNumber(i+1, board)[0]);
                changeInJ = Math.abs(indexOfNumber(i, board)[1] - indexOfNumber(i+1, board)[1]);

                //horizontal first
                if((changeInJ == 1 && changeInI == 0) && (i+2) <= finalValue) {
                    //update next move
                    changeInI = Math.abs(indexOfNumber(i+1, board)[0] - indexOfNumber(i+2,
                    board)[0]);
                    //test if the very next is vertical
                    if(changeInI == 1) {
                        return true;
                    }

                } else {

                }
            }

        }
        return false;
    }
}
```