

## 🎯 The Goal

Compute all possible future states of an ongoing game of *Mark Steere's Cephalopods* in the **shortest time possible**.

## ✓ Rules

Cephalopods is an abstract two player game played with dice on a grid.

Your objective is to find **all the possible board states** after a given number of turns by **simulating all possible moves** from a given board state.

This version of Cephalopods is played on a **3x3** board, on which players take turns adding dice of their own color, one die per turn.

A newly placed die must show a **one**, unless it's a **capturing placement**, more information below.

Once the board is full, the game **ends**, and the winner is the player with the most dice showing a six.

*Note:* In this exercise, you will **not** be required to compute the winner of the game, nor will you need to track whose dice is whose.

## Capturing placement

If a die is placed adjacent to two or more dice, and any combination of these adjacent dice has a sum of displayed values **less than or equal to six**, then the player must remove that combination from the board. The played die must then display the sum of the values of the removed dice.

If **multiple combinations** are possible, the player **chooses** which one to apply.

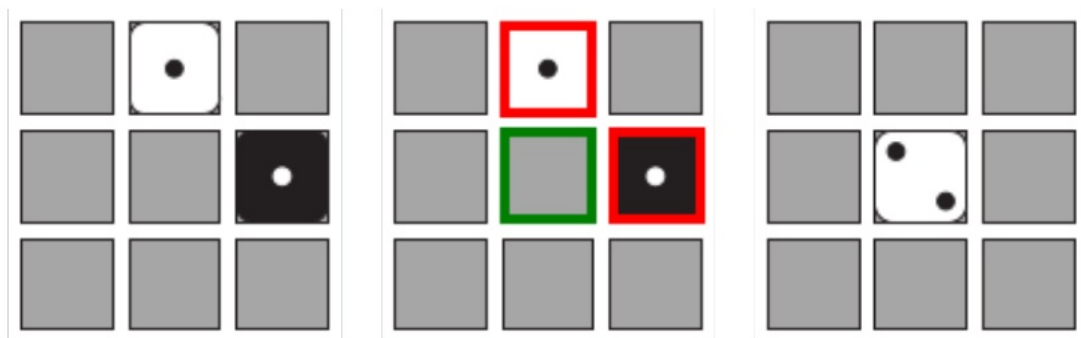
Captures are **mandatory** when placing a die where a capture is possible.

## Non-capturing placement

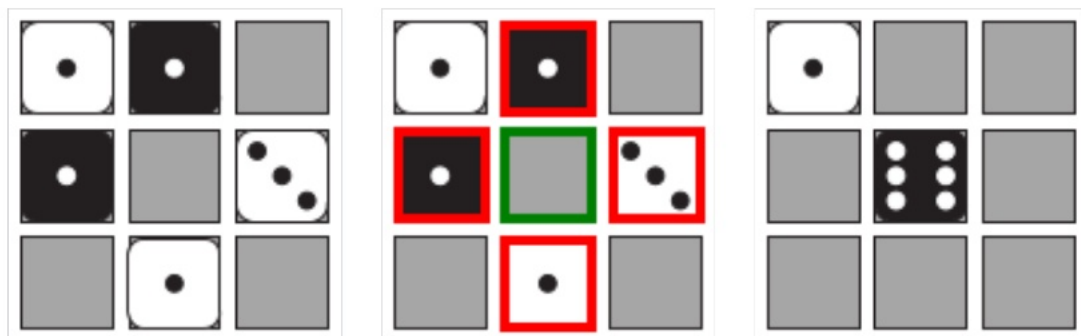
If a die is placed on a space where no captures are possible, its value will be **one**.

## 📱 Move examples

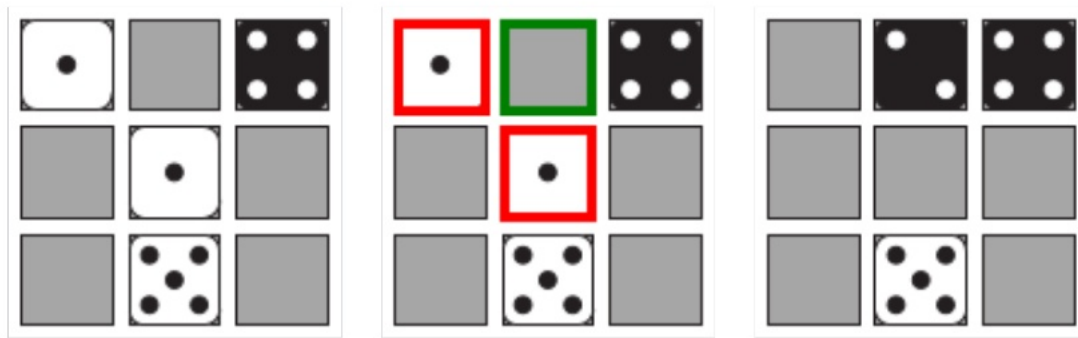
### Capture



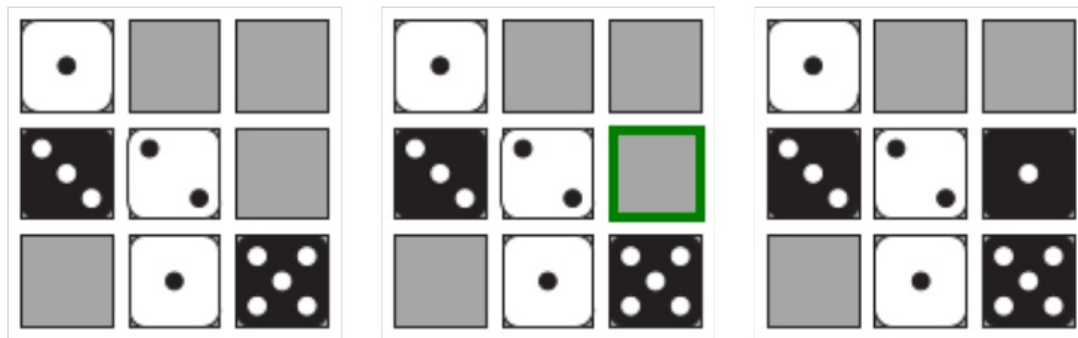
### Capture



## Capture with multiple possible dice combinations



## Non-capture



## Your mission

Your program must read from the **standard input** the initial state of the board and **maximum number of turns** to simulate, then compute all possible board states **after** the given number of turns, including games that have ended before the maximum depth was reached.

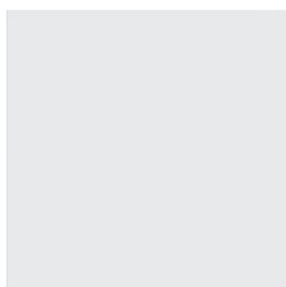
See input protocol for more details.

## Board state hashes

Each board state is represented as **32-bit** integer using a simple algorithm. Here's how the hashing works:

- The board is treated as a grid where each square is encoded with a single piece of information: the **value of the die** on that square (**1-6** or **0** for empty).
- The hash is built by iterating over all squares from **left to right** then **top to bottom**. The integer is **shifted left** by 1 digits and the die value is added. Essentially, **each square is a digit** in the integer.

As an example, here is the encoding for the following board.



|   |   |   |
|---|---|---|
| 1 | 0 | 0 |
| 3 | 2 | 1 |
| 0 | 1 | 5 |

= 100321015

= 100321015

### Expected output

Simulate all possible games and calculate the hash of the final board state at the end of the given number of turns for each game.

Then, **sum all these hashes**.

To avoid overflow, the sum is calculated **modulo  $2^{30}$** .

For instance, if you have the following hashes:

656010333

4022100

551401012

...

Then the final sum should be calculated like this:

```
final_sum = 0
```

```
final_sum = (final_sum + 656010333) % 230
```

```
final_sum = (final_sum + 4022100) % 230
```

```
final_sum = (final_sum + 551401012) % 230
```

etc...

### How is my score calculated?



We will run your code **ten** times per validation test. After removing the **two** best and worst times, we will calculate the average of the remaining **6**. This average will be your score for that test, and the total sum of each will be your final score, given in **milliseconds**.

### ! Note

**You can run the tests by launching them from the “Test cases” window.** Submit your code to enter the leaderboard. **Each validation tests you pass will earn you some points.**

**Warning:** the tests provided are similar to the validation tests used to compute the final score **but remain different**. This is a “hardcoding” prevention mechanism. Hardcoded solutions will not get any points.

**Moreover,** the validation tests **will change again** after the event ends for the ranking computation. Your last submission and the submission with the best score during the event will be compared for this final run.

### Game Input

#### Input

**First line:** one integer **depth** for the max number of player moves to simulate.

**Next 3 lines:** 3 **die\_value** integers representing each space of one row of the board.

- **die\_value**: the value of the die on this space or **0** if this space is empty

#### Output

**One integer:** **final\_sum**, the sum of all board hashes for possible game states after at most **depth** turns.

## Constraints

$$1 \leq \text{depth} \leq 40$$

Response time to output: **10 seconds** for most test cases.

The last two validators have a max response time of **30** and **40** seconds respectively.