## Overview

Lines is a game played on a **N**x**N** grid. Each cell of the grid is either empty or contains a ball with one of **C** colours. A line is a set of **5 or more** adjacent (horizontally, vertically or diagonally) balls of the same colour. In each move you can select a ball and send it to another empty location, provided that the ball can reach it via a 4-connected path of empty cells. If a move creates one or more lines then the balls comprising those lines will be removed giving you points. In particular, removing **n** balls gives you **n*n - 7n + 20** points. If a move does not create any lines then no points are scored and **3** new balls are added to the grid at random empty locations. The game ends when there are no more moves to be made (grid is full) or when you reach **1000** moves. Your task is to score as many points as possible.

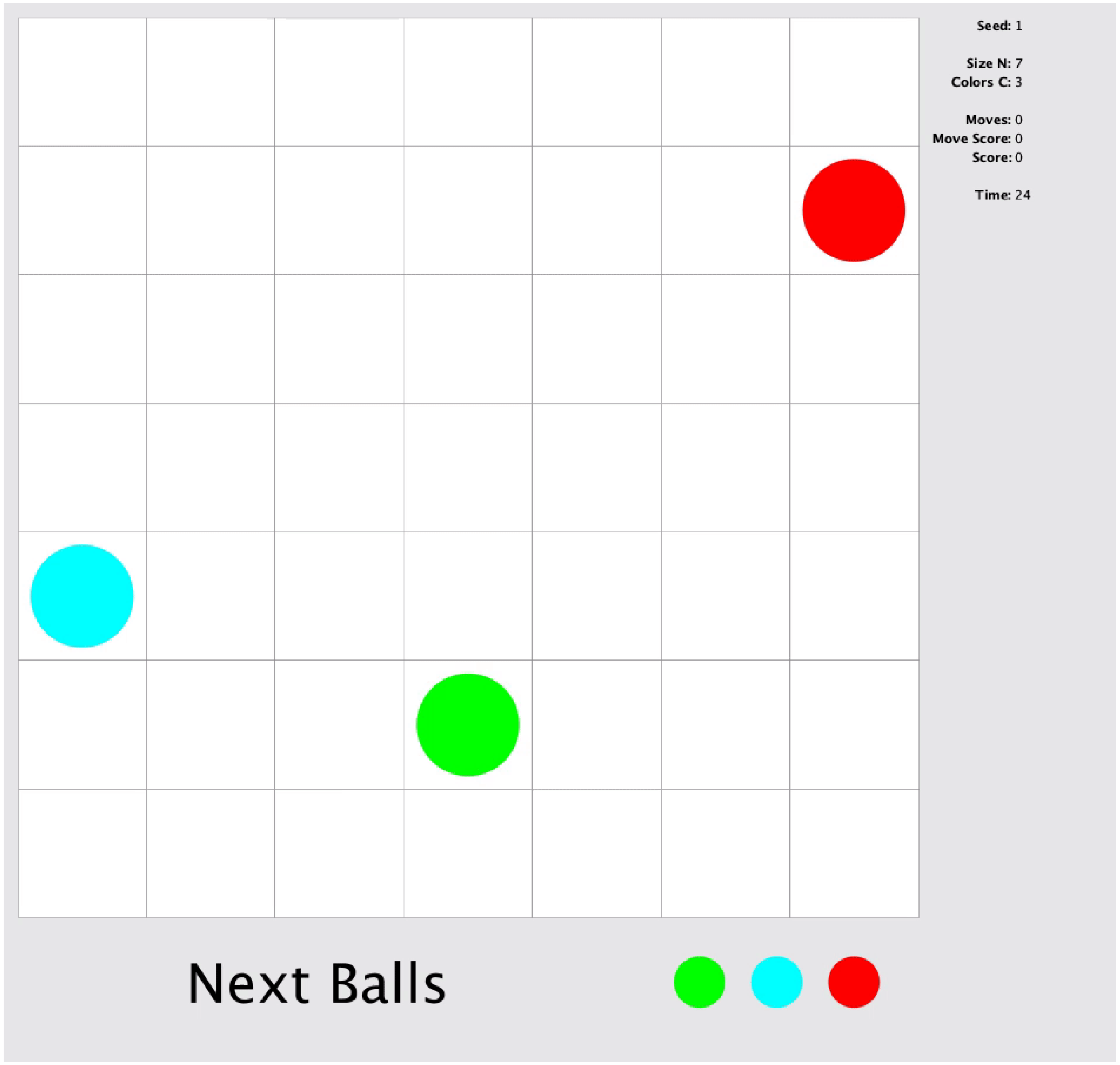
Here is an animation of a solution for seed=1

**TOOLBOX:**

Topcoder Extension for VSCode ?

Topcoder Templates repository

*ID: 30175218*



## Input and Output

## Input and Output

This is an interactive problem, so your code needs to interact with the tester for each move. Initially your code will receive as input the following values, each on a separate line:

- **N**, the size of the grid.
- **C**, the number of ball colours.

The following game loop repeats for **1000** moves or until the grid is full, whichever occurs first:

- Your code receives the following values:
    - **N*N** lines describing the grid in row-major order. Each cell is either **0** (empty) or a number from **1** to **C**, representing the colour of the ball in that cell. The first row is at the top of the grid.
    - **3** lines representing the colour of the next **3** balls.
    - A number representing the total time (in milliseconds) spent in your solution.
- Your solution outputs a move command formatted as "r1 c1 r2 c2". This means that you want to move the ball from (r1, c1) to an empty cell at (r2, c2). These coordinates are 0-based and there must be a 4-connected path of empty cells between them. Row **0** is at the top and row **N-1** is at the bottom. Column **0** is left.
- The tester will then:
    - Move the ball from (r1, c1) to (r2, c2).
    - If the move created any new lines of **5 or more** balls of the same colour then these balls will be removed and your score incremented.
    - Otherwise **3** new balls will be added at random empty locations.
    - The new balls can potentially make new lines of **5 or more** balls of the same colour. In that case, the balls from these lines will be removed and your score incremented.

## Scoring

The raw score is the total score you achieved over all moves. Removing **n** balls in one move gives you **n*n - 7n + 20** points.

If your return was invalid, then your raw score on this test case will be -1. Possible reasons include:

- Not formatting the moves correctly.

- Using coordinates that are out of bounds.
- Trying to move the ball to an unreachable empty cell.

If your raw score for a test case is negative then your normalized score for that test case is 0. Otherwise, your normalized score for each test case is YOUR/MAX, where YOUR is your raw score and MAX is the largest positive raw score currently obtained on this test case (considering only the last submission from each competitor). Finally, the sum of all your test scores is normalized to 100.

## Test Case Generation

Please look at the generate() method in visualizer's source code for the exact details about test case generation. Each test case is generated as follows:

- The grid size **N** is chosen between **7** and **11**, inclusive.
- The number of ball colours **C** is chosen between **3** and **9**, inclusive.
- The grid is initialized with **3** randomly placed balls.
- The colour of the next **3*1000** balls is chosen between **1** and **C**, inclusive.
- All values are chosen uniformly at random.

## Notes

- If all balls have been removed (empty grid) then new balls will be added.
- It is possible to remove multiple lines in one move. In a rare case these lines can be of different colour.
- Two locations are 4-connected if they differ by one row or by one column.
- The time limit is 10 seconds per test case (this includes only the time spent in your code). The memory limit is 1024 megabytes.
- The compilation time limit is 30 seconds.
- There are 10 example test cases and **100** full submission (provisional) test cases. There will be 2000 test cases in the final testing.
- The match is rated.

## Languages Supported

C#, Java, C++ and Python

## Submission Format

Your submission must be a single ZIP file not larger than 500 MB, with your source code only.
Please Note: Please zip only the file. Do not put it inside a folder before zipping, you should directly zip the file.

Make sure you name your Source Code file as Lines.*<appropriate extension>*

#### SAMPLE SUBMISSIONS

Here are example solutions for different languages, modified to be executed with the visualizer. You may modify and submit these example solutions:

- Java Source Code - Lines.java
- C++ Source Code - Lines.cpp
- Python3.6 Source Code - Lines.py
- C# Source Code - Lines.cs

## Tools

An offline tester is available below. You can use it to test/debug your solution locally. You can also check its source code for an exact implementation of test case generation and score calculation. You can also find links to useful information and sample solutions in several languages.

#### Downloads

- Visualizer Source - Line_Source.zip
- Local Tester - tester.jar

#### Offline Tester / Visualizer

In order to use the offline tester/visualizer tool for testing your solution locally, you'll have to include in

In order to use the offline tester/visualizer tool for testing your solution locally, you'll have to include in your solution the main method that interacts with the tester/visualizer via reading data from standard input and writing data to standard output.

To run the tester with your solution, you should run:

**java -jar tester.jar -exec "<command>" -seed <seed>**

Here, <command> is the command to execute your program, and <seed> is seed for test case generation.
If your compiled solution is an executable file, the command will be the full path to it, for example, "C:\TopCoder\Lines.exe" or "~/topcoder/Lines".
In case your compiled solution is to be run with the help of an interpreter, for example, if you program in Java, the command will be something like "java -cp C:\TopCoder Lines".

Additionally you can use the following options:

- **-seed** <seed> Sets the seed used for test case generation, default is seed 1.
- **-debug**. Print debug information.
- **-novis**. Turns off visualisation.
- **-delay** <delay>. Sets the delay (in milliseconds) between visualizing consecutive simulation steps, default is 200.
- **-fastMoves**. Does not show intermediate steps when moving balls, so makes the animation much faster.
- **-pause**. Starts visualizer in paused mode. See more information below.
- **-manual**. Play the game manually using the mouse. Left click to select a ball, and left click again on an empty cell to move the ball there. Clicking the same ball again deselects it.
- **-N** <N> Sets a custom grid size.
- **-C** <M> Sets a custom number of ball colours.

The visualizer works in two modes. In *regular* mode, steps are visualized one after another with a delay specified with the **-delay** parameter. In *paused* mode, the next move will be visualized only when you press any key. The space key can be used to switch between regular and paused modes. The default starting mode is regular. You can use the **-pause** parameter to start in paused mode. You can also play