Downloads

- Executable JAR of the visualizer
- Source code of the visualizer

In order to use the offline tester / visualizer tool for testing your solution locally, you'll have to modify your solution by adding the main method that interacts with the tester / visualizer via reading data from standard input and writing data to standard output. As long as you do not change the implementation of method *init* and *placeTowers*, this doesn't affect the way your solution works when being submitted to our server.

To simulate a single test case, your program should implement the following protocol (each integer is to be read from / printed in a separate line):

- Read integer **N**.
- Read integer **money**.
- Read **N** strings **board**[0], **board**[1], ..., **board**[**N**-1].
- Read integer **creepHealth**.
- Read integer **creepMoney**.
- Read integer **NT**. Note that (NT=T*3).
- Read **NT** integers **towerType**[0], **towerType**[1], ..., **towerType**[**NT**-1].
- Call *init*(**board**, **money**, **creepHealth**, **creepMoney**, **towerType**).
- Simulate **2000** steps by performing the following **2000** times:
- Read integer **money**.
- Read integer **NC**.
- Read **NC** integers **creep**[0], **creep**[1], ..., **creep**[**NC**-1].
- Read integer **B**.
- Read **B** integers **baseHealth**[0], **baseHealth**[1], ..., **baseHealth**[**B**-1].
- Call *placeTowers*(**creep**, **money**, **baseHealth**). Let *ret[]* be the return value.
- Print the length of *ret[]* on a separate line.
- Print the integers in *ret[]* each on a separate line. Flush standard output stream.

In other words, you should implement the following pseudocode in the main method of your solution:

```
N = parseInt(readLine())
money = parseInt(readLine())
for (i=0; i < N; i++)
    board[i] = readLine()
creepHealth = parseInt(readLine())
creepMoney = parseInt(readLine())
NT = parseInt(readLine())
for (i=0; i < NT; i++)
    towerType[i] = parseInt(readLine())
init(board, money, creepHealth, creepMoney, towerType)

for (t=0; t < 2000; t++)
{
    money = parseInt(readLine())
    NC = parseInt(readLine())
    for (i=0; i < NC; i++)
        creep[i] = parseInt(readLine())
    B = parseInt(readLine())
```

```
        for (i=0; i < B; i++)
            baseHealth[i] = parseInt(readLine())
        ret = placeTowers(creep, money, baseHealth)
        printLine(ret.length)
        for (i=0; i < ret.length; i++)
        printLine(ret[i])
        flush(stdout)
    }
```

In order to run the tester / visualizer, you should use the following command:

```
java -jar tester.jar -exec "<command>"
```

`<command>` is the command you would use to execute your solution. If your compiled solution is an executable file, the command will just be the full path to it, for example, "C:\TopCoder\solution.exe" or "~/topcoder/solution". 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 Solution".

Additionally you can use the following parameters (all are optional):

- `-seed <seed>`. Sets the seed used for test case generation. Default value is 1.
- `-novis`. Switches the visualization off, leaving only text output.
- `-sz <cell size>`. Sets the size of one cell of the board, in pixels. The default value is 12.
- `-delay <delay>`. Sets the delay (in milliseconds) between visualizing consecutive simulation steps. The default value is 100.
- `-pause`. Starts visualizer in paused mode. See more information below.
- `-delay`. Render more information in the visualizer.

Visualizer works in two modes. In *regular* mode, steps are visualized one after another with a delay specified with `-delay` parameter. In *paused* mode, the next turn will be visualized only when you press any key (except space). The space key can be used to switch between regular and paused modes. The default starting mode is regular. You can use `-pause` parameter to start in paused mode. Pressing the **d** character will toggle debug mode on and off.

You can print any debug information of your solution to the standard error stream and it will be forwarded to the standard output of the tester.

For more information on using visualizers, please check the following recipe draft from TopCoder Cookbook. Note that this is not a troubleshooting thread, please use the match forum for questions instead.