Programming challenge: Connect Z

Release

Yobota

Sep 13, 2019

CONTENTS:

CHAPTER

ONE

PROGRAMMING CHALLENGE - CONNECT Z

1.1 Background

This programming challenge is based on the classic game of Connect Four. In a game of Connect Four, two players take turns to drop coloured counters into a vertical frame. The first player to achieve a horizontal, vertical or diagonal line of four or more counters of their own colour wins the game. Games can also result in a draw if the frame fills up with counters before either player can complete a line of four or more counters.

The following resources provide more detailed background information on how Connect Four works:

- http://www.wikihow.com/Play-Connect-4
- https://en.wikipedia.org/wiki/Connect_Four

1.2 Challenge

Your goal is to implement a *game checker* program for Connect Z. In Connect Z the concept of the traditional game of Connect Four is generalized to include playing frames of any size and a target lines of any length. When provided with a data file that describes a game of Connect Z your checker program should determine if that game was won, drawn or contains an error of some kind. The format of the data files and expected output is described in detail below.

1.3 Program specification

- This challenge should be written in Python 3. The reference implementation that will be used to assess the submission is Python 3.6.
- Third party libraries are not allowed. The challenge should be completed using only features provided by the standard libraries. https://docs.python.org/3/library/index.html
- The game checker should be submitted as a Python script called 'connectz.py'.
- The script will read input from a datafile which is a pain ASCII textfile.
- This script takes a single argument 'inputfilename' which is the name of the data file containing game information to be checked.
- The script will be called as:

```
python connectz.py inputfilename
```

- If the game is run with no arguments or more than one argument it should print the following message as a single line to standard out.

```
connectz.py: Provide one input file
```

1.4 Input file specification

Your program will be supplied data files that describe games of Connect Z. The data file includes both the dimensions of the game being played and the moves that were made. The specifics are as follows:

- The input should be a plain text file consisting of one or more lines of ASCII characters.
- The first line of the file is required and describes the dimensions of the game in the format:

```
X Y Z
```

- X, Y and Z are each positive integers separated by whitespace:

X represents the width of the frame in columns;

Y represents the height of the frame in rows;

Z represents the minimum number of counters in a straight line required to win the game.

For example the classic game of Connect 4 is played on a 7 by 6 frame with 4 counters in a line required to win. This would be represented in the input format as:

```
7 6 4
```

After the game dimensions line, each subsequent line of the file represents a single move in the game. Starting with player one, lines alternate between player one and player two indicating the moves that they made. A move is described by a single positive integer which represents the column the player chose to drop their counter in. The file should end after the game is won or drawn. Additional moves should be detected as an invalid game.

For example in a classic Connect 4 game, if the first player has a strategy to always select column one and the second player always selects column two, the the first player will win on the seventh move, with a vertical line in column one. The complete game file would be as follows:

```
7 6 4
1
2
1
2
1
2
1
```

1.5 Impossible games

Note that using the input format it is easy to describe games that would be impossible to win. For example:

```
2 4 6
```

represents a game that can never be won, since a line of length 6 on a 2 by 4 frame could never be achieved. Situations like this should be detected as invalid game. See the output specification.

1.6 Output specification

- The output of the program should be a single integer printed to standard out.
- The integer is a code which describes the input.
- Output codes 0 - 3 are for valid game files.
- Output codes 4 - 9 represent errors.

The codes are defined as follows:

Code	Reason	Description
0	Draw	This happens when every possible space in the frame was filled with a counter, but neither player achieved a line of the required length.
1	Win for player 1	The first player achieved a line of the required length.
2	Win for player 2	The second player achieved a line of the required length.
3	Incomplete	The file conforms to the format and contains only legal moves, but the game is neither won nor drawn by either player and there are remaining available moves in the frame. Note that a file with only a dimensions line constitues an incomplete game.
4	Illegal continue	All moves are valid in all other respects but the game has already been won on a previous turn so continued play is considered an illegal move.
5	Illegal row	The file conforms to the format and all moves are for legal columns but the move is for a column that is already full due to previous moves.
6	Illegal column	The file conforms to the format but contains a move for a column that is out side the dimensions of the board. i.e. the column selected is greater than X
7	Illegal game	The file conforms to the format but the dimensions describe a game that can never be won.
8	Invalid file	The file is opened but does not conform the format.
9	File error	The file can not be found, opened or read for some reason.

1.7 Assessment

We will judge your submission on a number of criteria. Firstly we will run your program against our set of test input cases. These will test all of the output conditions we have defined. Note that also some of our input files may be *very* large so we will be looking at the efficiency and optimisations made in your solutions. Finally we will also look at your coding style. We pay particular attention to clear and sensible naming of variables and functions. Bonus points for well commented code!

1.8 Submission

Please submit your solution by email to case.study@yobota.xyz with *Connect Z* as the subject line.

As a minumum you should attach your *connectz.py* file. This is the only file that will be run and should work standalone.

You may additionally submit a readme.txt file containing a summary of your solution, how you approached the problem, what you found challenging / easy. What you would like to improve and any other part of your solution you would like to draw our attention too.

If you create an additional files as part of implementing your solution feel free to include these as well and explain what they are for in your readme.txt

CHAPTER

TWO

APPENDIX - SAMPLE TEST CASES

This appendix contains sample test cases that you can use as a starting point to check your program is logically correct.

Please note that these test cases are not exhaustive and we will run your program additional test cases to those described here.

We strongly recommend supplementing the test cases provided here with some of your own.

2.1 Draw

2.1.1 input

```
3 3 3
1
2
3
1
3
2
1
3
2
```

2.1.2 output

```
0
```

2.2 Win for player 1

2.2.1 input

```
3 3 3
1
2
1
2
1
```

2.2.2 output

```
1
```

2.3 Win for player 2

2.3.1 input

```
3 3 3
1
2
3
2
1
2
```

2.3.2 output

```
2
```

2.4 Incomplete

2.4.1 input

```
3 3 3
1
2
3
1
```

2.4.2 output

```
3
```

2.5 Illegal continue

2.5.1 input

```
3 3 3
1
2
1
2
```

```
1
2
```

2.5.2 output

```
4
```

2.6 Illegal row

2.6.1 input

```
3 3 3
1
2
2
1
1
2
2
```

2.6.2 output

```
5
```

2.7 Illegal column

2.7.1 input

```
3 3 3
1
2
3
4
```

2.7.2 output

```
6
```

2.8 Illegal game

2.8.1 input

```
3 3 4
1
2
3
1
3
2
1
3
2
2
```

2.8.2 output

```
7
```

2.9 Invalid file

2.9.1 input

```
banana
```

2.9.2 output

```
8
```