NCursesConnectN

Matthew's Stats

Time Taken: 3 hours
Files: 12
Lines: 651
Structs: 5

What to submit

- A CMakeLists.txt file to create a program named NCursesConnectN.out
- The necessary source files to compile your project

Problem Description

We will be creating an extension to the classic children's game Connect4 called ConnectN. In Connect4 players take turns dropping pieces into a 6 X 7 board trying to line up 4 or more pieces in a row either vertically, horizontally, or diagonally. You can play a game of Connect4 by following one of the links.

The differences between Connect4 and ConnectN is that in ConnectN we get to specify how big the board is and how many pieces in a row we need to win.

Problem Details

Input

Command Line Arguments

The user will enter the following command line parameters in this order

1. The number of rows of the board
   a. This is guaranteed to be an integer number >= 1
2. The number of columns of the board
   a. This is guaranteed to be an integer number >= 1
3. The number of pieces in a row needed to win
   a. This is guaranteed to be an integer number >= 1
4. The path to a file to log your board state into

    a. This is guaranteed to be a valid path

Standard Input

- An **‘a’** or **‘j’** means to move the play piece 1 space to the left
    - If the play piece is currently at the left edge of the board it should wrap around to the right edge
- An **‘s’** or **‘k’** means to move the play piece 1 space to the right
    - If the play piece is currently at the right edge of the board it should wrap around to the left edge
- A space or enter means to drop the piece into the column that the play piece is on assuming it is not full
    - If the column is full the input should be ignored
- All other input should be ignored

Gameplay

Players take turns dropping pieces into one of the non-full columns on the board until either

- A player wins by connecting at least N of their pieces in a row either vertically, horizontally, or diagonally
- Or the board becomes full without either player winning, resulting in a tie

After the game is over your game should declare one of

- Player 1 Won!
- Player 2 Won!
- Tie Game

based on how the game ended

Output

The Screen

Your screen should be implemented using curses and should contain 3 main areas

- Play Area
    - This is the area that shows where the piece that the user is about to play will be dropped
- Board
    - This shows the board and the pieces that have been played so far
    - Player 1 plays ‘X’, Player 2 plays ‘O’ and spaces that haven’t been played in are filled with ‘*’
- Announcement
    - This is where you will announce who won the game or if there was a tie
    - If the game is in progress it should be blank

An example of what the board should look like on a 6 X 7 board is bellow. The play area is highlighted in yellow, the board in green, and the announcement in purple. The underscores should not actually be displayed but are only there so that I could highlight the area.

At start of game

```
X______
*******
*******
*******
*******
*******
*******
_______
```

In the middle of a game

```
     X_
*******
*******
****X**
***XO**
**XOXO*
*OXOXO*
_______
```

At the end of a game

```
     X_
*******
*****X*
****XO*
***XOX*
**XOXOO
*OXOXOX
Player 1 won!
```

The Log File

In order to test your program and see if it is working correctly, you will need to log your screen to a file. Every time a character is entered and after the game ends you will need to write the state of the screen to the log. Unfortunately, we can't just redirect your output to a file and check that as the cursors implementation doesn't work well for this. While this is what Kodethon will be checking for the automated testing, the TAs will be playing your game to see if it does the curses portion correctly. Those that don't get the curses portion working correctly will only receive 70% of their Kodethon score.

Examples

In the Kodethon for this problem, you should find 2 executable files.

- KodethonNCursesConnectN.out will run on Kodethon
  - If you are going to run it first create a new terminal by typing terminal in the shell or clicking the terminal button
- CSIFNCursesConnectN.out will run on the CSIF
  - If you are remotely accessing the CSIF through ssh you will need to do TERM=vt100 on the command line before running CSIFNCursesConnectN.out so that it will display correctly
- You will need to run chmod on the executable to add back in execute permissions to be able to run the program: `chmod u+x exeName`

Hints

- Good usage of structs makes this problem much easier to manage
- I had structs for
  - The Game
  - The Board
  - A Move
  - The View (Screen)
  - A Window that tracked where the cursor is at
- I highly recommend having a struct that tracks where you want the cursor to be at. This way you can more easily control when the "cursor" moves and makes wrapping around the screen much easier
- The tic tac toe example we did in class is a good reference point