**Haskell: Connect 4**

**Nikita Kolotov: nkk31**

**Project Rundown:**

This project is a typical game of Connect 4. When first booted up the Yellow player called “Y” will be informed that it is their turn. Below this notification the game board is displayed with “0” representing every open slot. Below that is a bar with numbers underneath it listing all the possible slot options that the player can choose (0-6). A typical game board is 6 slots tall and 7 slots wide so naturally this board is 6x7. After the player “Y” makes their choice the system will do a sweep and then return to the red player “R”’s turn. If an invalid move is made, then the program will not make the placement and instead return to the original players turn with an error message telling them that the move is invalid. Players will be prompted with an invalid move message if they either do not put the chip into a column between 0 and 6 or if they try to put it into a column that if full. If the user enters a string containing alphabetic characters, this will cause the program to quit due to user error. The win conditions are checked every round as well as for draws so all outcomes have been accounted for. Once there is a winner or a draw the game will immediately announce a message and then quit.

**Motivation:**

My purpose for this project was to further develop my Haskell skills and to compare the programming language with an imperative static typed language, Java. In CS111 the final project was to create a game of Connect 4 and it was surprisingly harder to develop than the Haskell version. Haskell being a functional language only allows for data to be saved in the form of function parameters. Thus, a lot more planning ahead was required. However, once both versions were complete I got to analyze the time complexity for each project.

**Discovery:**

Haskell time outputs:

Test 1.)

real 0m15.745s

user 0m0.193s

sys 0m0.029s

Test 2.)

real 0m5.999s

user 0m0.168s

sys 0m0.021s

I played two games and hit randomly the first time, then tried to win the fastest the second time. This is why for the documentation purposes, the real is not important since it accounts for user input time. The system times and user however, match. This is for consistency sake as I try to make sure the tests are similar. From what I gathered Java does run faster on the system however. This is most likely due to the fact that Java offers much more flexibility and data retention than a pure functional programming language can offer. However, on the other hand, being a function language is very beneficial to the testing cycle as testing the program was easily done. Having the ability to test functions one at a time partnered with Haskell abstraction makes for an easy testing environment. Although a game of Connect 4 is not what Haskell was designed for, it shows me that Haskell is great for evaluating data compared to Java. While the functions are compiled at runtime, we can greatly benefit from the abstraction offered by Haskell. On the other hand a game is much more suitable for Java, which is what I would separate the two as: each one is best for it’s own function.

**Difficulties:**

Initially I was unsatisfied with just having a simple terminal version of Connect 4 so I set out to make a GUI for it. However, the GUI I had settled for was not very well covered online which left a lot to self-teaching. The GUI I had decided upon was Gloss and it was fairly easy to import and set up. However, it was not easy to learn which cost me a lot of time. In the end I decided to scrap the idea but I still have the files to my attempt at a GUI if it is ever needed.

I then tried to see if I could color certain text in the terminal differently but sadly this did not seem possible during my research. So I opted for the simple approach.