Pet Haskell Assignment Report

by Adnan Abdulhussein (aa1462)

Overview

I have grown to really enjoy programming in Haskell, and this is one of the reasons I decided to do *Pet* over *Robocode*. However, when I first looked at the *Pet* assignment I wasn't sure what to do -- the examples didn't *really* appeal to me.

After a bit of thought I decided that I wanted to build an example of the difference between *object-oriented/imperative* and *functional* programming languages (in that I wanted to see how easy it would be to *port* programs between the types of languages). I also wanted to have a go at using a GUI library in Haskell.

Overall I enjoyed the assignment, however if given more time I could have produced a better end result.

Initial idea

My intial idea was to reproduce the *Db* assignment from Java in Haskell. This went pretty well, until I *stupidly* deleted the files accidently (command line *fun*). I had not yet implemented a GUI yet so I decided to look into that before I started the assignment again.

Choosing a GUI library

It seemed there were two popular GUI libraries for Haskell, *WxHaskell* and *Gtk2Hs*. I initially went for *WxHaskell* however it didn't work properly on my machine. So I went for *Gtk2Hs* instead.

Oxo

Given the time I had used up starting the assignment (messing up on the first idea and spending about two days installing the GUI library), it seemed *Db* would take too long to redo. So I decided to do *Oxo* (again from Java), as it would only require one window to operate.

The logic was straight forward, and generally easier than in Java, thanks to lists. The interesting part was the *GUI*. I needed the *game state* to be known and updated each time a cell was clicked. I started out changing all the buttons' actions each time a cell was clicked, but that would only add a new action to the stack (so it would eventually be executing functions multiple times) and there was no way to disconnect previous actions because all the newly generated actions for all the buttons would need to have been known beforehand -- this couldn't be done because the actions had to be built button by button (i.e. traversing through the list in Haskell).

Then it hit me. I could just use a label to hold the game state (who's turn it was, whether the game was over and who the winner is). I then just pass this label as a parameter to the actions and read and update it as necessary -- thus determining the game state without any messy action changing. The label needed to be shown anyway so that the users could have an idea of what was going on in the game.

There was one issue with *Gtk2Hs*, I was unable to close the window properly with the `New Game` button. This was because, bizarrely enough, there is no way to close/hide a window in *Gtk2Hs*, well according to it's documentation anyway.

Unfortunately, I was unable to get round to implementing the AI I used in the Java assignment due to running out of time, and had to leave it as a multiplayer game.