CS 2006 Haskell Practical 1

Distributed: 11.00, Tuesday October 19th 2010
Deadline: 10.00, Monday November October 15th 2010

Title: Text Adventure
Short description: Your task is to build a simple interactive text adventure game, in Haskell.

Technical Note

You should ensure that any code you produce and submit can be compiled work on the machines in the Mac Lab. (I will need to be able to run the code you submit so that I can test that it works.)

Preparation

Some code is given as a starting point. You can download the code at `https://studres.cs.st-andrews.ac.uk/2010_2011/Teaching/Teaching_UG/SubHons/Level_2/CS2006-APP/Practicals/p2/code/`. There are three files:

- `Adventure.hs`, the main program
- `World.hs`, containing information about the world.
- `Actions.hs`, which implements user commands (and is unfinished).

To run the game, you can either:

- Load `Adventure.hs` into `ghci` and run the function `main`. You can also test individual functions this way.
- Compile to an executable with `ghc --make Adventure.hs -o adventure` at the terminal prompt, then run `./adventure`.

You will find that in its current form, the only command which works is `quit`.

Review the lecture notes for Lecture 6, which give an overview and history of text adventure games, and an overview of the architecture of these games in Haskell.

Plot

The plot of the adventure game as it stands is very simple, and the game world small. You start in your bedroom, and the goal is get out of your house so that you can go to your lectures. However, you will need to have some coffee first.

Basic Requirements

In `Action.hs` you will find incomplete definitions of a number of functions which implement the game commands, along with descriptions of what these commands should do. The basic requirements are therefore to:

- Implement "movement" so that the player can move around the game world with commands such as
  - `go north`
  - `go west`
- Implement "get" and "drop" commands so that the player can pick up and drop objects. This involves removing the object from the current room and adding it to the player's inventory (and vice versa).

- Implement specific commands for pouring and drinking coffee, and opening the door. These involve updating the game state.
- Implement commands to help the user, to allow them to examine objects and list their inventory.

In `Actions.hs` you will find stubs for functions to achieve the above, as well as stubs for a number of "helper" functions to break the tasks down.

General hints: Many of the functions involve using record access and record update syntax to extract the relevant fields before processing the data. You may find `where` clauses useful for writing list processing functions. Remember also that you can test functions individually at the `ghci` prompt. Finally, think carefull about how to divide the work between you — for example, each of the above requirements can be tackled independently.

Submission

1. **Haskell Code for upload on to MMS:**
   There will be a group submission for your code – i.e. everyone in the group should submit the same code. Your MMS submission should be a single `.zip`, `.tar.gz` or `.tar.bz2` file containing a single directory (which may have subdirectories if you wish) with all the source code (well-commented `.hs` files) needed to run your application, i.e. everything should be in that directory or part of the Haskell standard library – there should be no dependencies on external libraries. I should be able to run your program form the command line of a bash shell.
2. **Report for upload on to MMS:** There will be an individual entry on MMS for upload of individual reports. A single PDF file of around 6–8 pages in length, which includes the information listed below. Please submit PDF documents only. In your report, where appropriate, try to concentrate on why you did something the way that you did in your code, rather than just explaining what the code does:
   (a) A summary of the functionality of your program indicating the level of completeness with respect to the Basic Requirements listed above, and any Additional Requirements listed below. Include a description of:
      i. Any known problems with your code, where you do not meet the Basic Requirements, or the Additional Requirements that you have attempted.
      ii. Any specific problems you encountered and how you esolved them.
   (b) An accurate summary stating which files or code fragments you have written and any code you have modified from that which was given out in class, or which you have sourced from elsewhere.
   (c) A description of how you tested your application.
   (d) Screenshots or output text of the working code, as appropriate.
   (e) Your own contribution to the group work.

Grading

A well-commented/documented, fully-working solution that meets the Basic Requirements, accompanied by a clear and informative report, should get you a grade of 13. The report should include all the items listed above in the Basic Requirements, and should make clear the work you have completed.

It is recommended most strongly that you ensure you have completed the Basic Requirements and have something to submit before you attempt the Additional Requirements listed below!

Additional Requirements

To gain a higher level grade, you need to complete the Basic Requirements and attempt one or more of the following items. The level of difficulty is given in brackets *(like this)*. For successful completion of individual items that are marked *Easy* or *Medium* you are unlikely to gain a grade above 16.

Many of these will involve updating `World.hs` and `Adventure.hs` as well as `Actions.hs`.

1. The game world is currently very small. Extend the world with new rooms and new objects *(Easy)*.
2. Where possible, replace recursive definitions with `foldr`, `map` and `filter`, or list comprehensions. *(Medium)*.
3. Introduce new puzzles, for example:
   (a) At the beginning of the game, you cannot see anything until you put the light on.
   (b) You need to find a key to unlock the front door.

   (Medium)
4. Save and Load functionality. To do this you will need functions `save :: GameData -> String -> IO ()` and `load :: String -> IO GameData` where the `String` is a file name. Look at the documentation for the `IO` library (`http://www.haskell.org/ghc/docs/6.12.2/html/libraries/haskell98-1.0.1.1/IO.html`) in particular the `readFile` and `writeFile` functions. You will need to modify the `repl` function in `Adventure.hs`.

   (Difficult)
5. Objects and directions are represented as strings. It would be better if they were represented as data types, e.g. `data Direction = North | South | East | West`. Introduce data types for directions and objects and update the game to use them. *(Difficult)*
6. Similarly, commands are represented as strings. Introduce a data type for commands such as `data Command = Go Direction | Get Object | ...` and update the parser (the `process` function in `Adventure.hs`) and game to use it. *(Difficult)*
7. The parser is very simple, only able to understand two word phrases. Extend it so that it can understand phrases such as:
   (a) `open door with key`
   (b) `open door then go out`

   (Difficult)

Attempting one or more items from the list of Additional Requirements above does not guarantee you a higher level grade automatically – it will depend on the overall quality of your submission, both code and report, of course!

In your report, highlight where you believe you have done something "extra", for example one of the items listed above, or something else you believe is novel or innovative with respect to the work that has been done in class so far. Include in your report screenshots or output text, as appropriate, showing any additional functionality.

Academic Fraud

If you present someone elses work as your own, that is plagiarism, and you are leaving yourself open to being accused of Academic Fraud. This happens when you take the work or ideas of someone else (whether from your class; from someone else in the University; from a text book; or from a WWW site found through a Google search), and present them as if you had produced them yourself.

Plagiarism is cheating and can carry severe penalties: do not plagiarise.