220005149, 220017161, 220010065 Tutor: Olexandr Konovalov 7 February 2024

CS2006 Haskell 1 Practical

Overview:

For this practical, we were asked to create an interactive, text-based game in Haskell. The starting point and basic requirements involved implementing various undefined functions to facilitate the basic operations of the game. From here we designed and implemented various additional features. We have produced an implementation which meets all basic requirements, includes a majority of the additional suggestions given, and extended the codebase with some of our own ideas for extra features. Since our project allows the player to move between rooms, interact with objects and eventually leave the house, it successfully meets the basic criteria given, while the optional features we have included provide the user with a wider range of possible outcomes when playing the game. As such, the finished program is a successful submission at constructing the game described in the specification.

Functionality:

The implementation of our practical contains all of the Basic functionality from the specification along with several of the suggested additional requirements. Furthermore we added several other objectives and aspects to increase the complexity and enjoinment of playing. To complete the basic requirements of the specification several steps were taken. We quickly realised that the provided undefined functions in Actions.hs all were implemented using strings for the commands. To simplify the code and streamline progress, we introduced data types such as objects, directions, actions, and rules. Objects were quite simple to extend from the provided World.hs file. Directions were implemented to decrease string complexity. This was done with a new data type in World.hs which contained aspects representing the various possible movements. Actions are also a newish data type, as they work slightly differently than the previous

implemented string variant. In Actions.hs, action, action2 and completeAction act as our version of a parser which process user provided strings into the command data type. These commands are passed to completeAction which calls the various according functions to enable functionality. Finally the last type is rules which are very similar to actions or commands. This type is defined for functions the user calls which do not update game state but just provide the player with additional information. We decided to separate commands into actions and rules to provide simpler parsing complexity. All of the undefined functions provided in Actions.hs were also completed. This functionality was achieved by splitting the tasks into smaller helper functions as well as larger command functions to perform specific tasks. While writing this functionality we ensured to use higher order haskell functions instead of recursion. While this not only simplified the process it resulted in a cleaner and more efficient code base. At this point in the project we has completed all of the basic requirements in the specification. We knew this was not enough to satisfy a strong implementation so we began work on the suggested additional requirements. Starting with the easy requirements we added several more rooms and objects to liven up the world. A bathroom was created and several other objects were added such as a toothbrush, torch, eggs to name a few. Then we decided to add more puzzles into the game to increase complexity and user playability. We made it so the entire game/world is dark until the lights are switched on. Furthermore while the game can be beat by doing the bare minimum of drinking coffee, we have implemented an achievement system which enables the player to achieve unique end game messages for completing different tasks. After this, we added cabal to our program for simple building, compiling and running of the code. A .cabal file is included with the submission which contains all of the required dependencies for functionality. From the start of our implementation we used higher order functions such as filter, map, foldr and others. While coding we bore in mind Haskell's preference for pattern matching over “if” statements due to the former being more explicit and exhaustive, but in some scenarios where we noted HLint’s suggestion to use an if statement for simplicity and readability. This enabled us to complete yet another medium requirement. Moving forward to the hard requirements two/three were implemented. Haskeline was implemented into the Adventure.hs file to enable a simple game loop for grabbing user commands as well as printing output to the terminal. Along with this we added save and load

features to the game. The player can type SAVE or LOAD followed by a save file name to save and load the current game state from a saves directory. The program uses show and read from the prelude to write and read the GameData to a file. Lastly, and this is only barely included as it is only a partial completion, we implemented a more complex version of our parser which accepts multi word responses. We use this in order to enable save/load as well as the combine command we created which combines two objects. On top of the suggestions we added more game functions like new commands and game types. There is a cupboard box type which is located in the kitchen. This is a type of object which can store values and can be opened/closed to reveal objects to the user. Additionally new commands were created to eat, combine and use objects. These all add into the game play of our implementation add give more options to the player. Lastly a rule type, help, was created to let the user see all of the available commands.

Problems:

Of the medium requirements, all but one was fully implemented. We attempted to add QuickCheck to test properties of our functions. Several issues arose with this implementation as both cabal and ghc did not play nicely with the version we created. After attempting several different revision and different computers, neither ghc nor cabal new how to run the files. When attmpeting to run we would get the same error occurrence repeatedly:

ActionsTest.hs:12:11: error:

- No instance for (Arbitrary GameData)

  arising from a use of ‘quickCheck’

This was perplexing as the proper import was included at the top of the file, import Test.QuickCheck as well as downloading and updating the dependencies in cabal. We attempted to look up this error online to no helpful suggestions. Along with this we looked at the provided examples on studres which did exactly what we were doing. We also consulted other groups who reached similar conclusions regarding the implementation. It is still unclear to us whether this is an issue due to the latest version of ghc and cabal or an entirely different issue from something

we did. Unfortunately while we wish we could have gotten quick check to work we ran out of time and could not complete that one aspect of the practical.

Examples:

Included below is one of the many possibilities for playing the game. It tests and shows examples of all the major functionality of our implementation except save and load. That is easiest tested and run individually. There are instructions in the README file on how to load and run the game.

[redacted]@[redacted]-MBP InteractiveFiction % cabal run

[Game start! Type help for list of commands and quit to exit.]

You cannot see anything, the lights are off.

What now?

>> use lightswitch

Lights turned on/off.

You are in your bedroom.

To the north is a kitchen. To the south is a bathroom.

You can see: a coffee mug.

What now?

>> get mug

Item added to inventory.

You are in your bedroom.

To the north is a kitchen. To the south is a bathroom.

What now?

>> go south

Moved successfully.

You are in the bathroom. There is a shower and sink.

To the north is your bedroom.

You can see: a blue and white toothbrush, a shower.

What now?

>> use toothbrush

What are you gonna brush your teeth with, your fingers?

You are in the bathroom. There is a shower and sink.

To the north is your bedroom.

You can see: a blue and white toothbrush, a shower.

What now?

>> get toothbrush

Item added to inventory.

You are in the bathroom. There is a shower and sink.

To the north is your bedroom.

You can see: a shower.

What now?

>> use toothbrush

Brushed teeth.

You are in the bathroom. There is a shower and sink.

To the north is your bedroom.

You can see: a shower.

What now?

>> use shower

Showered successfully. Ignore the fact that the water turned black beneath you, I'm sure it's not important.

You are in the bathroom. There is a shower and sink.

To the north is your bedroom.

You can see: a shower.

What now?

>> go north

Moved successfully.

You are in your bedroom.

To the north is a kitchen. To the south is a bathroom.

What now?

>> go north

Moved successfully.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: a pot of coffee, a black torch, a jug of milk, an oven. There is also: a cupboard (closed)

What now?

>> get pot

Item added to inventory.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: a black torch, a jug of milk, an oven. There is also: a cupboard (closed)

What now?

>> get milk

Item added to inventory.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: a black torch, an oven. There is also: a cupboard (closed)

What now?

>> get torch

Item added to inventory.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (closed)

What now?

>> open cupboard

Opened successfully!

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (containing eggs, bread, batteries).

What now?

>> get eggs

Item added to inventory.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (containing bread, batteries).

What now?

>> get bread

Item added to inventory.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (containing batteries).

What now?

>> get batteries

Item added to inventory.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> combine torch batteries

Combined successfully! Check inventory for result.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> use oven

Make some eggy bread to cook first.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> combine eggs bread

Combined successfully! Check inventory for result.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> use oven

Cooked French toast! Added to inventory.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> eat french toast

Ate some delicious French toast.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> pour coffee

Poured successfully.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> combine milk coffee

Combined successfully! Check inventory for result.

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> drink coffee

Drank successfully - refreshing!

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> inventory

You are carrying:

a blue and white toothbrush

a pot of coffee

a black torch

a sense of satisfaction

an empty jug of milk

a coffee mug

You are in the kitchen.

To the south is your bedroom. To the west is a hallway.

You can see: an oven. There is also: a cupboard (empty)

What now?

>> go west

Moved successfully.

You are in the hallway. The front door is closed.

To the east is a kitchen.

What now?

>> open door

Opened front door.

You are in the hallway. The front door is open.

To the east is a kitchen. You can go outside.

What now?

>> go out

Moved successfully.

Congratulations, you have made it out of the house.

You showered, brushed your teeth and ate breakfast! I'm very proud of you for doing the bare minimum of human function.

In fact, you made a fantastic meal. You should be proud of yourself! [Achievement: Michelin Star]

You even made a "fancy" coffee while you're at it. Well done. [Achievement: Barista]

Now go to your lectures...

[redacted]@[redacted]-MBP InteractiveFiction %

Code Sources:

For this practical nearly all of the code was written by ourselves or modified in the provided source files. Actions.hs is a mixture of both our written code as well as modified specification code. Adventure.hs was heavily modified to implement our additional specifications. This file does still retain some of the original code but most of it has been rewritten. Parsing.hs was never touched in out implementation and remains the same as the provided file. World.hs is similar to Actions.hs as it contains both original code and modified versions of the original. ActionsTest.hs which contains all of the QuickCheck code is all written by us with some minor inspiration from the provided examples on studres. We did not source code from anywhere other than lectures, studres examples and the provided files. Several of the games functions required looking up on websites like Hoogle and Hackage which provide documantation of haskell prelude and packages.

Conclusion:

Upon the completion of this project we have provided a specification accurate implementation. All of the basic requirements were implemented. Nearly all of the additional specification were added to our game along with some of our own ideas. The program has been well and rigorously tested to remove all errors and bugs. While quick check was unable to run,

each of the functions has been manually tested using ghci testing/debugging. All of the code has been commented and is easily readable for inspection.

Version Control Repo:

via GitHub at: https://github.com/FChamb/InteractiveFiction

Word Count: 1464