Date:2022-02-07

CS2006 Haskell 1 Practical Group Report
Tutor: Gregor Heywood
Group Number: 8A

Matriculation ID's: 200014517
200027532
200019811

Overview
In this practical, we were asked to create a Haskell program which would enable a user to play an interactive adventure game. Our program supports all of the basic functionalities and the following additional requirements:

- **Easy** The game world is currently very small. Extend the world with new rooms and new objects
- **Easy** Introduce new puzzles
- **Easy** Set up a cabal file to describe how to build your program
- **Medium** Refactor your code to use list comprehensions and higher order functions such as map, filter, and foldr where appropriate.
- **Medium** Use QuickCheck to write property based tests for your functions
- **Hard** Use the Haskeline library (available on hackage) for reading user input instead of getLine

Design
The team has decided that the already given code was divided into intuitive files which separated actions from the main game loop, the parser, and the game world. Therefore, we didn't find it necessary to change this file structure.
In Actions.hs file, declarations of some of the functions that had to be implemented were given to us. While implementing these functions, the team found it necessary to use helper methods to make the actual function implementation more readable and less error prone. World.hs was also edited as well to handle the changes made to Actions.hs to make the program function.

We decided to implement all of the actions given at the top of Actions.hs. These all use some of the functions explained below to implement their required functionality.

World.hs
To fulfil the requirements, data types were added for the directions, objects that can be interacted with and the rooms each having predefined value for what they can be. The other changes is mainly just defining information for the newly added rooms. A few of the objects were moved in order to create more puzzles and have them be more interesting (e.g. the mug being moved to the new room: the pantry). This file stayed, mostly the same.

Actions.hs
This is where the main changes occur. Another data type is declared for the structure of the commands that the user can input. The move function uses the findExit function to see if the given direction correlates to an exit.

Find exit returns nothing if not given a list of potential exists. Otherwise it uses a recursive lambda function to check if the given direction is a potential place that can be moved to with the given list of exits. objectHere checks if a given object is in a given room using objectInList. objectInList simply checks that an item is within a list, so the object and list of objects within the room is given to the function.
removeObject takes an object and a room and uses the filter function to get rid of that object from the list of objects in that room. addObject appends the object given to the given rooms object list. findObj uses a recursive lambda function to search for an object in a list of all objects. updateRoom checks if the room being updated exists, if it does, it uses the updateGameWorld function to add the new version of the room. Otherwise a new room will be added to the game state. roomExists checks that a room is in the world, given its name. updateGameWorld filters the old room that is being updated out and then appends the updated version onto that list. addInv calls several previous functions to get the room the objects in and the object being added and then adds it to the inventory and removes it from the room. The state will then be updated. removeInv simply filters out the given object from the inventory list. carrying uses the previous function objectInList to see if the object is in the given inventory list. removeMaybe unwraps a monad and returns the raw value. This is only used when the monad cannot be Nothing.

Testing

The team has decided that the best approach to test the program would be to follow a step by step approach. We ran tests on functions individually to make sure they would not cause problems later on. Moreover, we also prepared a list of test cases which was constructed to make sure every functionality we were asked to implement was in fact supported by our program. Lastly, we utilised QuickCheck to test if certain functions returned the correct value. The addObject, removeObject functions are tested in a file called QuickCheck.hs. (See figure 15 for them all functioning). Most of the test cases we have tested can be found in the table below with the screenshots of these tests attached below the table.

Case No	What case is Being Tested	Pre Conditions	Expected Outcome	Actual Outcome
1	Successful termination	Termination command is given	The “Bye bye” message is displayed	The “Bye bye” message is displayed. (Figure 1)
2	Get An Item	Correct get command is given	The specified item is picked up and added to the inventory. The item is also removed from the room	The specified item is picked up and added to the inventory. The item is also removed from the room. (Figure 2)
3	Drop an Item	Correct drop command is given	The specified item is dropped from the	The specified item is dropped from the inventory

			inventory and added back to the room.	and added back to the room. (Figure 2)
4	Illegal move	The user should attempt an illegal move	Display error message, prompt the user again to make a move	Display error message, prompt the user again to make a move (Figure 3)
5	Testing the drop command for an edge case	The item which is picked up should be dropped in another room	The item which is picked up can be dropped to another room	The item which is picked up can be dropped to another room (Figure 4)
6	Successful execution of the go command to move rooms as well as testing the enforcement of the game rules	Correct go command is given	The user should be able to move rooms, however they shouldn't be allowed to go out unless the user meets the game requirements	The user should be able to move rooms, however they shouldn't be allowed to go out unless the user meets the game requirements (Figure 5)
7	Ability to examine objects and list the inventory.	Correct inventory and examine commands are given	The user is presented with the inventory of the items they are carrying and they are presented with the description of the item they want to examine respectively	The user is presented with the inventory of the items they are carrying and they are presented with the description of the item they want to examine respectively (Figure 6)
8	Ability to pour and drink coffee	The user is carrying the items to pour and drink coffee	Appropriate messages are printed and game state is updated after the commands are given	Appropriate messages are printed and game state is updated after the commands are given (Figure 7)
9	Illegal move #2	The user hasn't fulfilled the requirements of the game	Display the correct error message that lists the requirements to be met before the game can be completed	Display the correct error message that lists the requirements to be met before the game can be completed (Figure 8)
10	Successful game end	The user navigates through the requirements to complete the game	The win message is displayed	The win message is displayed (Figure 9)
11	Attempt to pick up an object that isn't in the room	The correct get commands are given	The user is told that the object isn't there.	The user is told that the object isn't there. (Figure 10)

12	Attempt to drop an object that isn’t in the user’s inventory	The correct get and drop commands are given.	The user is told they can’t drop the item here.	The user is told they can’t drop the item here. (Figure 11)
13	Attempt to pick up/drop an item that doesn’t exist	The correct get & drop commands are given.	The user is told they can’t do that.	The user is told they can’t do that. (Figure 12)
14	Cabal file build’s program correctly and it runs	The cabal file is initialised correctly	An executable is built and can be run.	An executable is built and can be run. (Figure 13)

Evaluation

Our submission achieves all the basic requirements and a fair amount of the advanced requirements as well. It is fairly robust as it checks for inputs the user can enter and is able to deal with it well, a potential issue is that when users have to interact with an object it can only take the one specified input from the code i.e. ‘get coffee mug’ being as valid an input as ‘get mug’. Moreover, the program does not successfully implement the following feature:

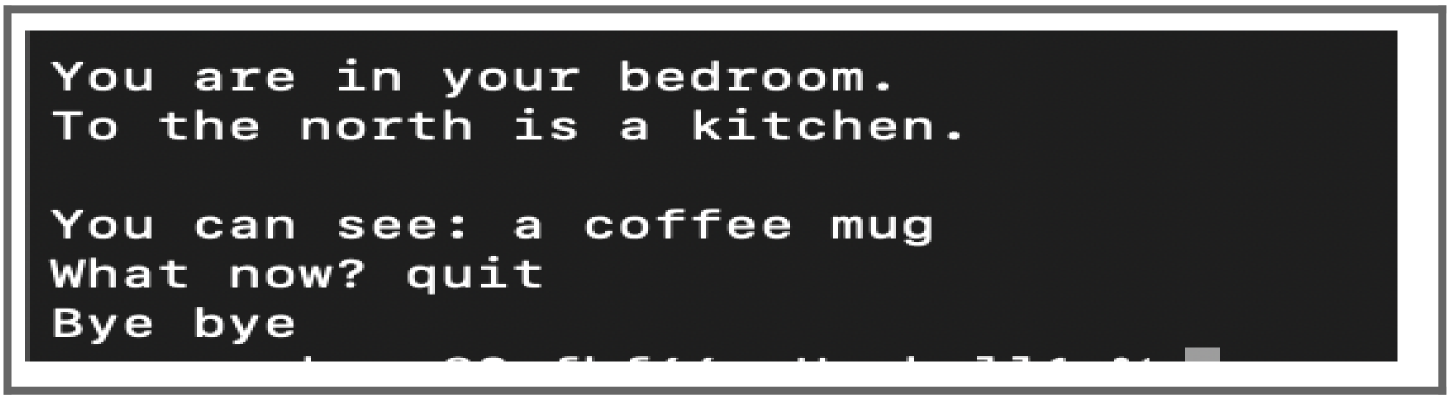
- **Hard** Implement save and load functionality.

Team has attempted to implement this feature, but it is only partially implemented due to time constraint. Furthermore, we would have liked to implement more quickCheck functions if we had more time. Lastly, while we have defined data types to represent objects, directions and commands, they are not thoroughly used within the action of the program, as strings are often still used to identify an instance of an object. No external libraries or code fragments have been used in this program. All of the code in this practical was either written by us, supplied as starter code or mentioned in lectures.

Conclusion

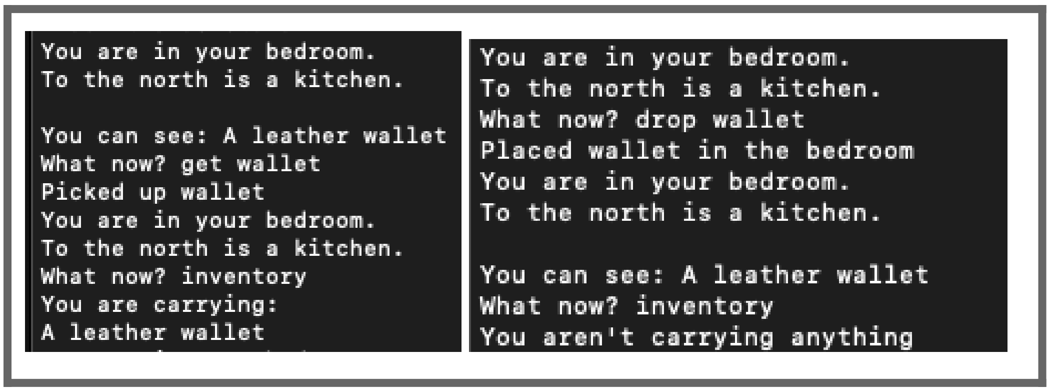
We have completed the basic requirements of the specification, as well as all of the easy and medium additional requirements and a hard requirement. We found it difficult to complete many of the requirements on time, however we managed to complete a significant number of them by the end. If we had more time, we would have extended the QuickCheck file to be more extensive, and used objects more thoroughly throughout our code.

Date:2022-02-07



```
You are in your bedroom.
To the north is a kitchen.

You can see: a coffee mug
What now? quit
Bye bye
```

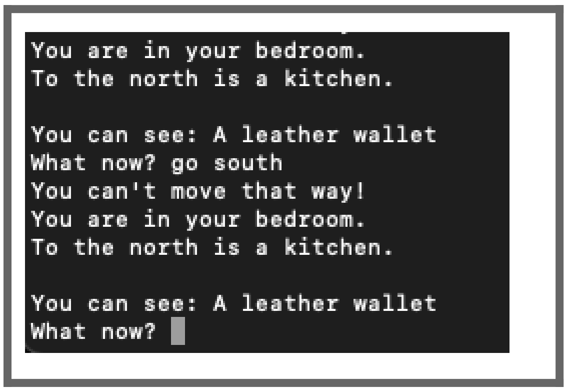
Figure 1: Test case 1, successful termination.



```
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? get wallet
Picked up wallet
You are in your bedroom.
To the north is a kitchen.
What now? inventory
You are carrying:
A leather wallet
```

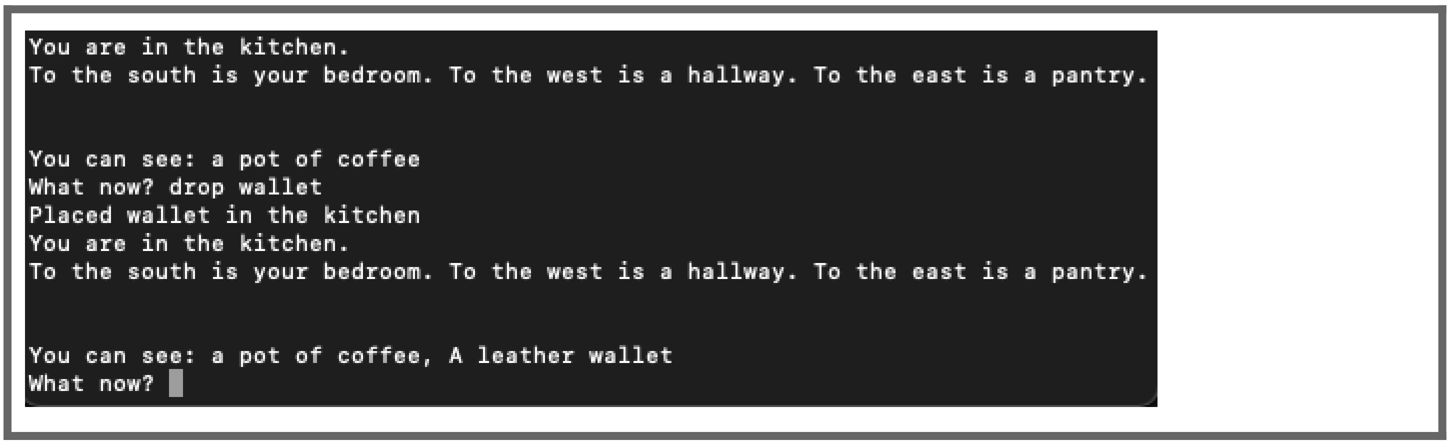
```
You are in your bedroom.
To the north is a kitchen.
What now? drop wallet
Placed wallet in the bedroom
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? inventory
You aren't carrying anything
```

Figure 2: Test cases 2 and 3, successful execution of get and drop commands with game state being updated



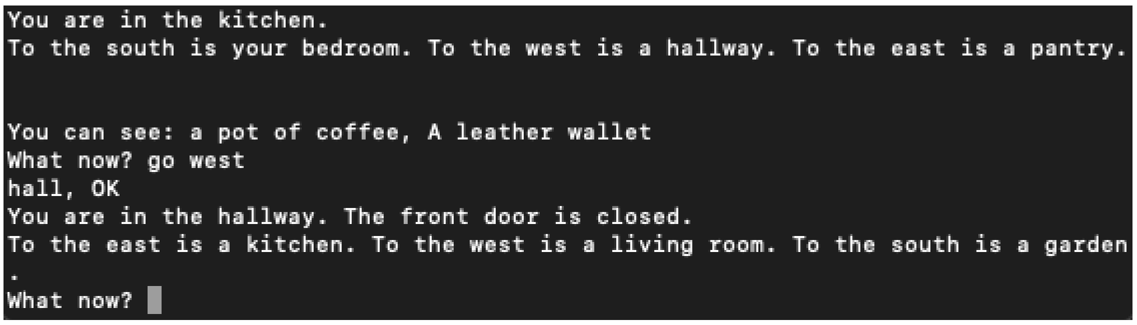
```
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? go south
You can't move that way!
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now?
```

Figure 3: Test case 4, successful handling of an illegal move.



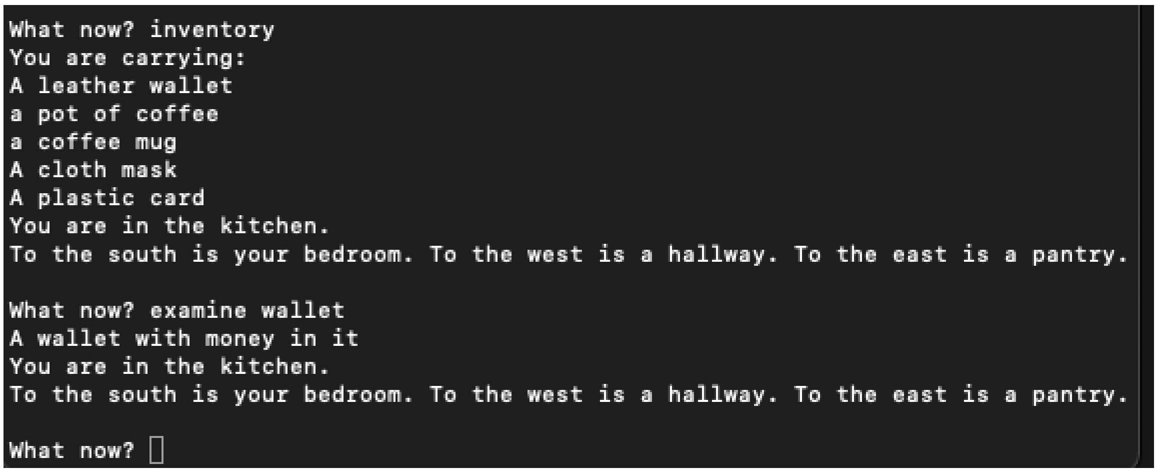
```
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.


You can see: a pot of coffee
What now? drop wallet
Placed wallet in the kitchen
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.


You can see: a pot of coffee, A leather wallet
What now?
```

Figure 4: Test case 5, successful execution of an edge case drop command with game state being updated



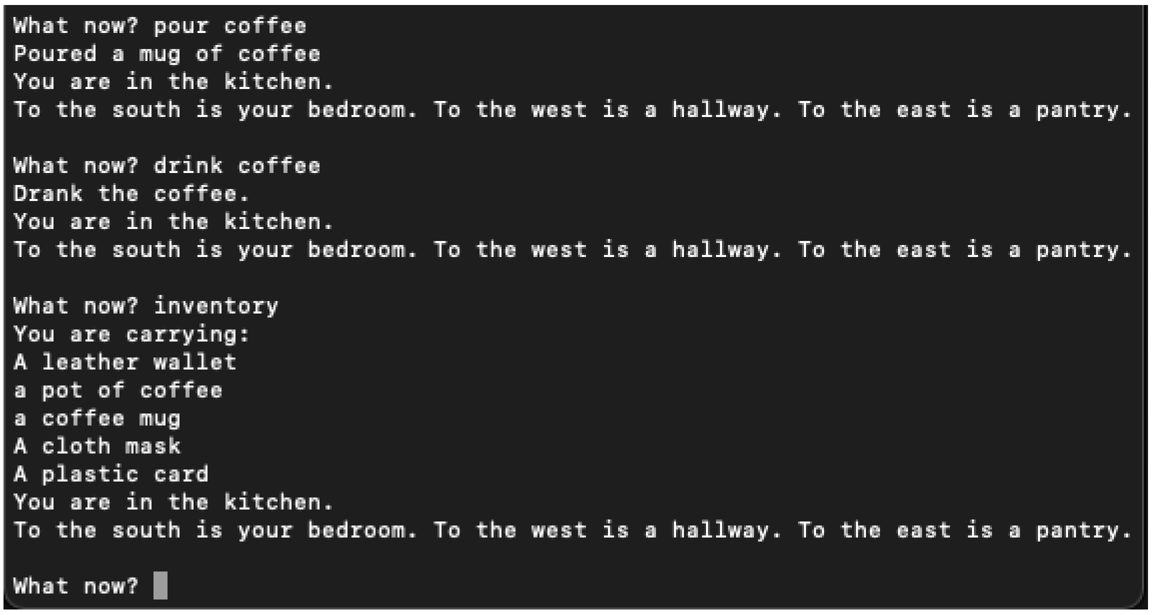
```
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.


You can see: a pot of coffee, A leather wallet
What now? go west
hall, OK
You are in the hallway. The front door is closed.
To the east is a kitchen. To the west is a living room. To the south is a garden
.
What now?
```

Figure 5: Test case 6, successful execution of the go command which also displays the enforcement of the game rules which prevents the user from going out unless the game conditions are met.



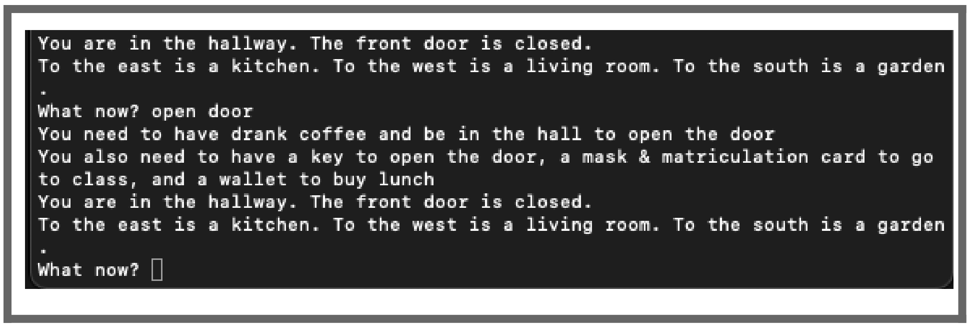
```
What now? inventory
You are carrying:
A leather wallet
a pot of coffee
a coffee mug
A cloth mask
A plastic card
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.

What now? examine wallet
A wallet with money in it
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.

What now?
```

Figure 6: Test case 7, successful implementation of inventory and examine commands



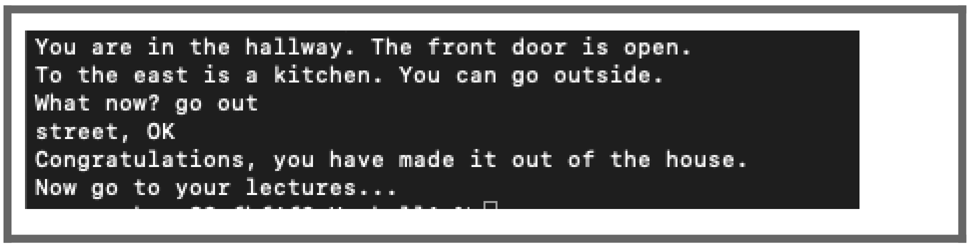
```
What now? pour coffee
Poured a mug of coffee
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.

What now? drink coffee
Drank the coffee.
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.

What now? inventory
You are carrying:
A leather wallet
a pot of coffee
a coffee mug
A cloth mask
A plastic card
You are in the kitchen.
To the south is your bedroom. To the west is a hallway. To the east is a pantry.

What now?
```

Figure 7: Test case 8, successful implementations of pouring and drinking coffee



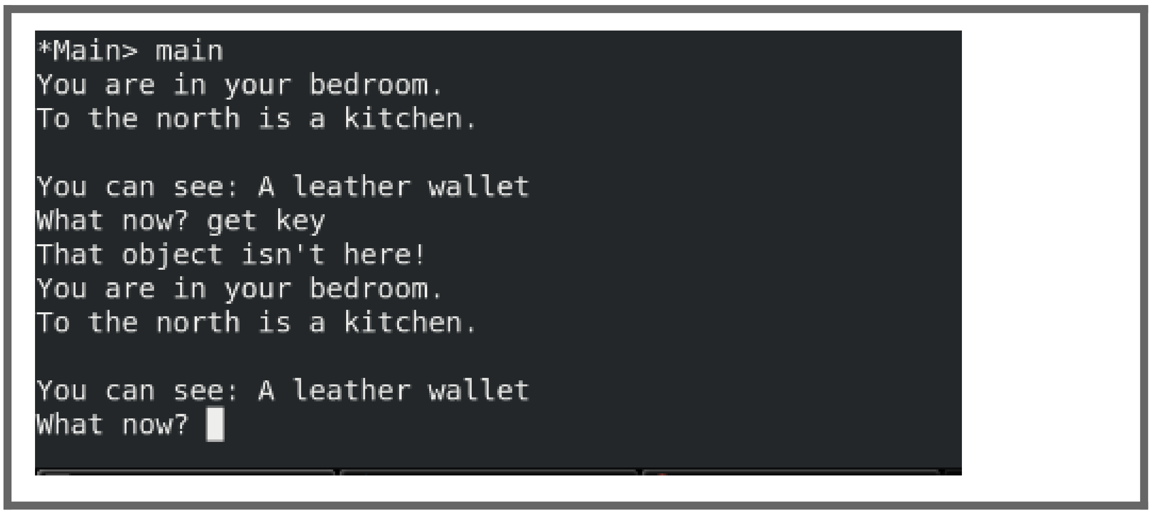
```
You are in the hallway. The front door is closed.
To the east is a kitchen. To the west is a living room. To the south is a garden
.
What now? open door
You need to have drank coffee and be in the hall to open the door
You also need to have a key to open the door, a mask & matriculation card to go
to class, and a wallet to buy lunch
You are in the hallway. The front door is closed.
To the east is a kitchen. To the west is a living room. To the south is a garden
.
What now? 
```

Figure 8: Test case 9, displaying correct error message which displays the requirements to be met before completion of the game



```
You are in the hallway. The front door is open.
To the east is a kitchen. You can go outside.
What now? go out
street, OK
Congratulations, you have made it out of the house.
Now go to your lectures...
```

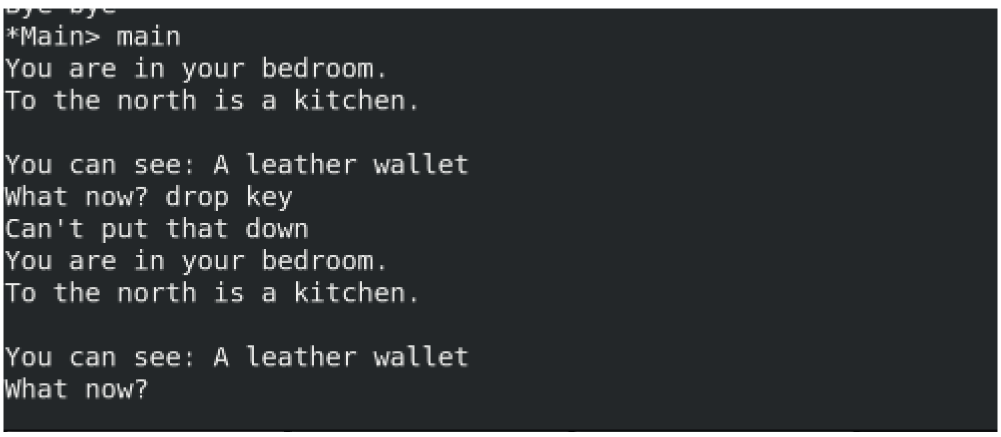
Figure 9: Test case 10, displaying successful completion of the game



```
*Main> main
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? get key
That object isn't here!
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? 
```

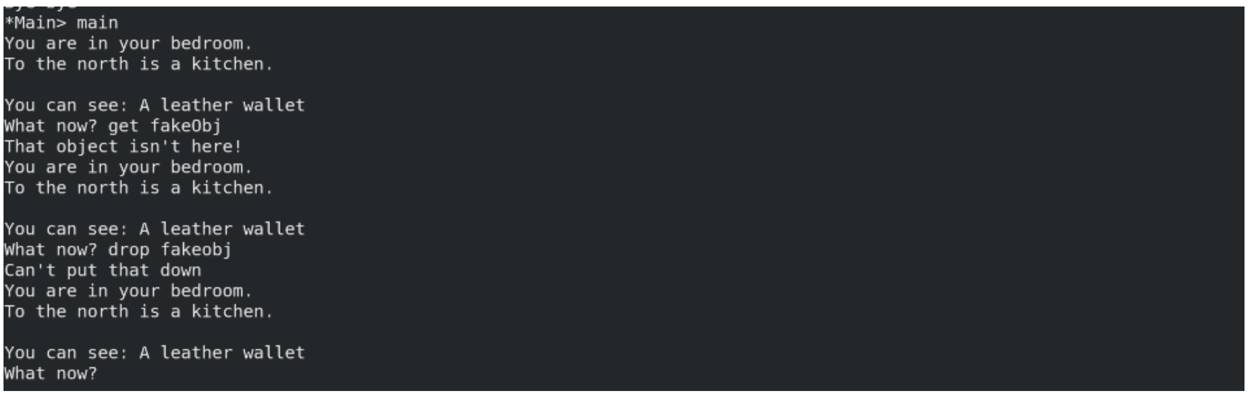
Figure 10: Test case 11, user fails to pick up object that isn't in that room

Date:2022-02-07



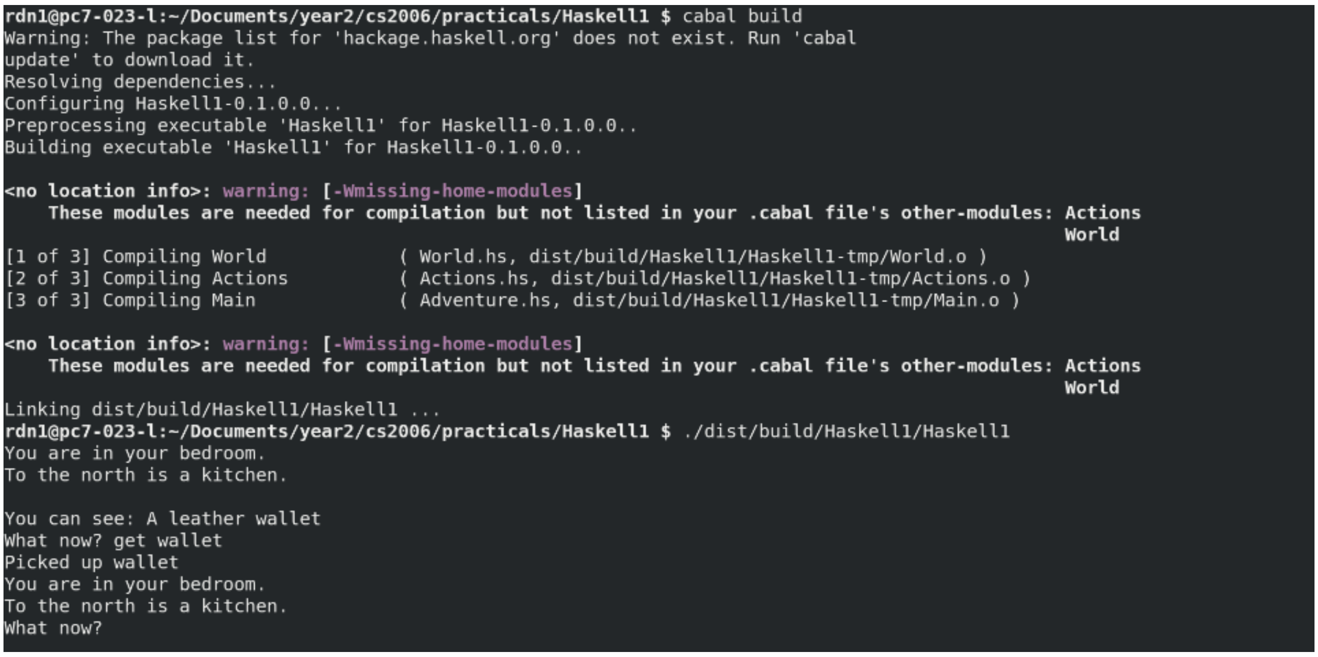
```
*Main> main
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? drop key
Can't put that down
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now?
```

Figure 11: Test case 12, user fails to drop object that isn't in their inventory



```
*Main> main
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? get fakeObj
That object isn't here!
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? drop fakeobj
Can't put that down
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now?
```

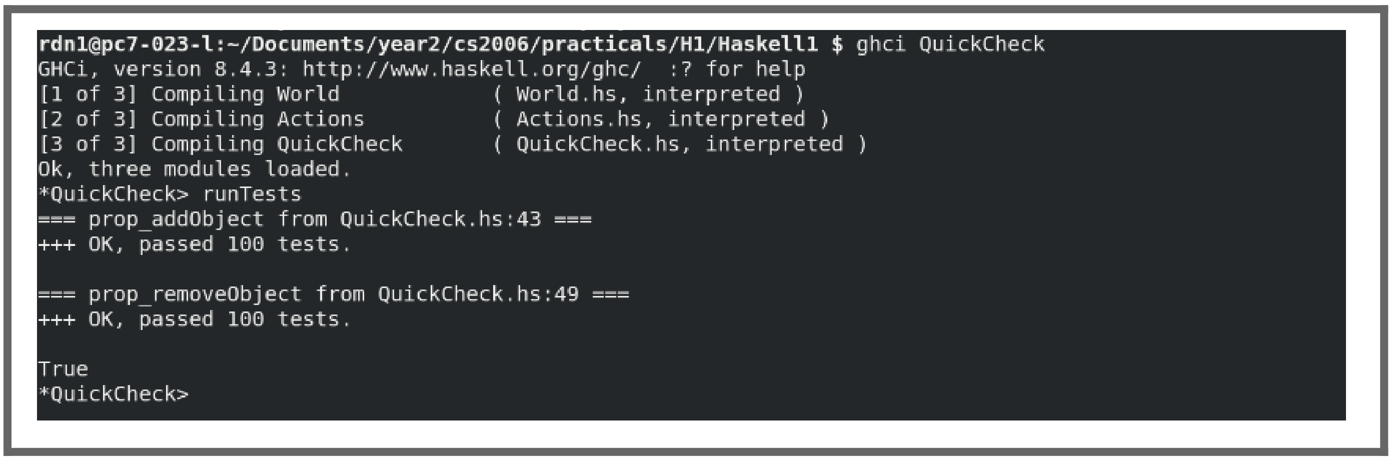
Figure 12: Test case 13, user fails to pick up & drop an object that doesn't exist



```
rdn1@pc7-023-l:~/Documents/year2/cs2006/practicals/Haskell1 $ cabal build
Warning: The package list for 'hackage.haskell.org' does not exist. Run 'cabal
update' to download it.
Resolving dependencies...
Configuring Haskell1-0.1.0.0...
Preprocessing executable 'Haskell1' for Haskell1-0.1.0.0..
Building executable 'Haskell1' for Haskell1-0.1.0.0..

<no location info>: warning: [-Wmissing-home-modules]
    These modules are needed for compilation but not listed in your .cabal file's other-modules: Actions
                                                                                                  World
[1 of 3] Compiling World            ( World.hs, dist/build/Haskell1/Haskell1-tmp/World.o )
[2 of 3] Compiling Actions          ( Actions.hs, dist/build/Haskell1/Haskell1-tmp/Actions.o )
[3 of 3] Compiling Main             ( Adventure.hs, dist/build/Haskell1/Haskell1-tmp/Main.o )

<no location info>: warning: [-Wmissing-home-modules]
    These modules are needed for compilation but not listed in your .cabal file's other-modules: Actions
                                                                                                  World
Linking dist/build/Haskell1/Haskell1 ...
rdn1@pc7-023-l:~/Documents/year2/cs2006/practicals/Haskell1 $ ./dist/build/Haskell1/Haskell1
You are in your bedroom.
To the north is a kitchen.

You can see: A leather wallet
What now? get wallet
Picked up wallet
You are in your bedroom.
To the north is a kitchen.
What now?
```

Figure 14: Test case 13, user fails to pick up & drop an object that doesn't exist

Date:2022-02-07



```
rdn1@pc7-023-l:~/Documents/year2/cs2006/practicals/H1/Haskell1 $ ghci QuickCheck
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
[1 of 3] Compiling World            ( World.hs, interpreted )
[2 of 3] Compiling Actions          ( Actions.hs, interpreted )
[3 of 3] Compiling QuickCheck       ( QuickCheck.hs, interpreted )
Ok, three modules loaded.
*QuickCheck> runTests
=== prop_addObject from QuickCheck.hs:43 ===
+++ OK, passed 100 tests.

=== prop_removeObject from QuickCheck.hs:49 ===
+++ OK, passed 100 tests.

True
*QuickCheck>
```

Figure 15: QuickCheck tests working