

Deric Wu - 21258784 | CP50065E | April 12, 2017

Multi User Dungeon

Functional Programming

Table of Contents

[Introduction 2](#_Toc479865152)

[Commands 2](#_Toc479865153)

[Start Game 3](#_Toc479865154)

[Rooms 3](#_Toc479865155)

[Description 4](#_Toc479865156)

[Connecting Rooms 4](#_Toc479865157)

[Objects 5](#_Toc479865158)

[Rooms with objects 5](#_Toc479865159)

[Rooms without objects 5](#_Toc479865160)

[Inventory 6](#_Toc479865161)

[Inventory with objects 6](#_Toc479865162)

[Inventory without objects 6](#_Toc479865163)

[Win 6](#_Toc479865164)

[Lose 6](#_Toc479865165)

[Problems 7](#_Toc479865166)

[Picking items 7](#_Toc479865167)

[Locked rooms 7](#_Toc479865168)

[High Order Function for removing 7](#_Toc479865169)

[Random Object Placement 8](#_Toc479865170)

[Source Code 8](#_Toc479865171)

[Libraries 8](#_Toc479865172)

[Start Game 8](#_Toc479865173)

[Database 9](#_Toc479865174)

[Items 12](#_Toc479865175)

[Displaying Objects and Rooms 12](#_Toc479865176)

[Actions 13](#_Toc479865177)

[Getting ID 14](#_Toc479865178)

[Help Window 17](#_Toc479865179)

[Reference Association List 17](#_Toc479865180)

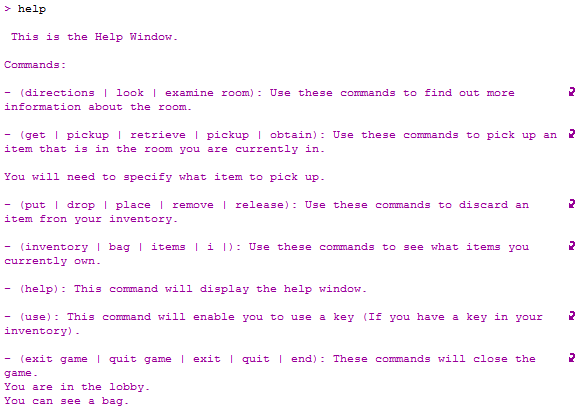
[Git Log 17](#_Toc479865181)

# Introduction

This report will be explaining about the content of the game and what it this game offers. It will also include an insight of how the game functions and the process of how the player will interact with the system. Furthermore, it will be explained about the problems that occurred within the back-end of things and how it goes about solving it.

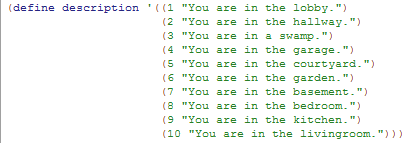
# Commands

When the user types the ‘help’ command, the game will display the help window to assist users when they are stuck.



# Start Game

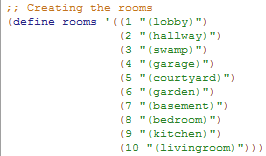
Depending on the id where they wish to start, a player can begin in any room they wish. But the default beginning is the lobby.



Screen Clipping Screen Clipping Screen Clipping

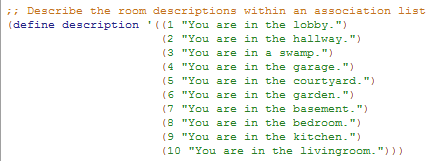
# Rooms

There are ten rooms in this game where the player can get access to. These rooms can hold objects which players and use to access rooms which are locked, that hold objects to win the game, or possibly lose.



## Description

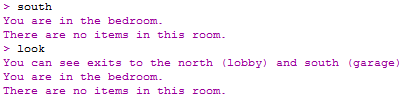
When the player starts the game, they will be shown what room they are currently in. This makes it easier for the player to track their whereabouts.



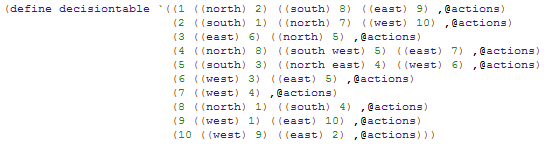
## Connecting Rooms

To access different rooms, the player must choose a direction. Each room has a direction assigned to them, but this is depending on where the player is currently at.

Screen Clipping

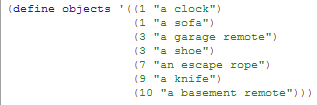


This is done by using decision table to read what room the user is in, and their input of direction they wish to go to.



# Objects

There are objects in the game the user can pick up. This is done by typing ‘pick’. Followed by their object of choice.



## Rooms with objects

There will be instances where a room has objects or does not. This is when the room does contain objects.

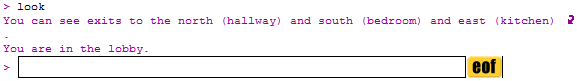
Screen Clipping

As mentioned above, to pick an object the user must type ‘pick’ followed by the item of choice.

Screen Clipping

Screen Clipping

Since both objects are now picked up, there should not be any more objects left in the room.



## Rooms without objects

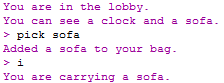
Here is what happens when a user enters the room where no objects are present.

Screen Clipping

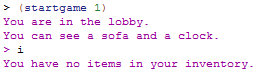
# Inventory

The inventory is where the objects the user picks up get put into. Users can easily access the inventory by simply typing ‘i’.

## Inventory with objects

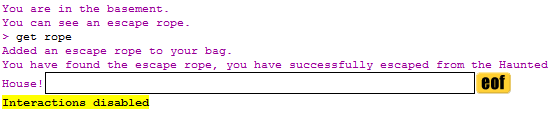


## Inventory without objects



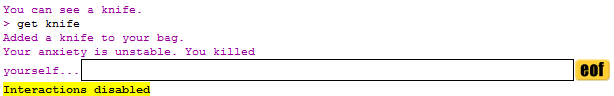
# Win

There will be an item in the game where if the user acquires it, they will win the game.



# Lose

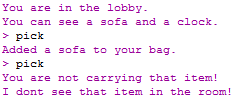
If the user gets picks up the knife, their anxiety will rise and will lose the game.



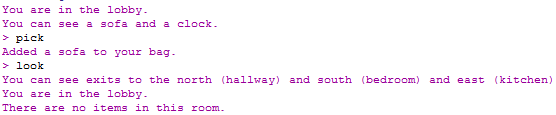
# Problems

## Picking items

When the user types in ‘pick’ without an object after it, the game picks up the first object and nothing else. Without the object, the game should notify the user that they must choose an object in the room. This makes it so that the game runs more professionally and without ambiguity.



Another issue is when if there is more than one object in the room, typing ‘pick’ will pick up the first object, and then remove the second object from the room. In this instance, the clock should still be present in the room.



## Locked rooms

Another issue was trying to implement a feature where a room is locked and the only way to access it is to have the key for it. It was attempted by making another association list but with the locked rooms. This will be worked on in the future and implemented properly.

Screen Clipping

Screen Clipping

## High Order Function for removing

Creating a high order function to handle the removing objects from the room and the inventory took a lot of time. Removing the items from the room worked properly but removing the items from the inventory did not go as planned.

All these problems and bugs present in the game will be solved in future versions where new possible implementations will be included.

# Random Object Placement

This is a future implementation to be added to the game. As of current, if the user has gone through the game once and completed, they would be able to easily beat the game again by remembering the location of each item. By having a random placement of objects, objects will be shuffled every time a user starts a new game. This is make the game more challenging to solve.

# Source Code

## Libraries

;;Scheme Request for Implementation

(**require** srfi/1) ;; This srfi works with pairs and lists

(**require** srfi/13) ;; This srfi works with String Libraries

(**require** srfi/48) ;; Intermediate Format Strings

## Start Game

;; Describe the room descriptions within an association list

(**define** description '((**1** "You are in the lobby.")

(**2** "You are in the hallway.")

(**3** "You are in a swamp.")

(**4** "You are in the garage.")

(**5** "You are in the courtyard.")

(**6** "You are in the garden.")

(**7** "You are in the basement.")

(**8** "You are in the bedroom.")

(**9** "You are in the kitchen.")

(**10** "You are in the livingroom.")))

;; The game loop, starting with the id the user chooses (room they begin in).

(**define** (**startgame** initial-id)

(**let** loop ((**id** initial-id) (**description** **#t**))

(**when** description

;; When there is description available to show to the player

(**display-description** id)

(**display-objects** objectdb id))

;; If there is no description nor objects to display

(printf "> ")

;; Read users input

(**let\*** ((**input** (**read-line**))

;; This is from the srfi/13 library. Each string in the users input will be split into substrings

(**string-tokens** (**string-tokenize** input))

;; Creates a list of symbols.

(**tokens** (map string->symbol string-tokens)))

;; Checks the id of the matching action. The greatest action is called to the code

(**let** ((**response** (**lookup** id tokens)))

(**cond** ((number? response)

(**loop** response **#t**))

;; If the user inputs a token that the game does not recall

((eq? **#f** response)

(printf "huh? I didn't understand that!\n")

(**loop** id **#f**))

;; To look for directions to other rooms

((eq? response 'look)

(**get-directions** id)

(**loop** id **#t**))

;; Token containing a string to pick up an item

((eq? response 'pick)

;; Picks up the item the user chooses

(**pick-item** id input)

(**loop** id **#f**))

;; Token containing a string to drop an item

((eq? response 'drop)

;; Drops the corresponding item

(**put-item** id input)

(**loop** id **#f**))

;; Token containing a string to show user inventory

((eq? response 'bag)

;; Displays user inventory

(**display-inventory**)

(**loop** id **#f**))

;; Token containing a string to show help window

((eq? response 'help)

;; Displays help window

(**help-window**)

(**loop** id **#t**))

;; Token containing a string to quit the game

((eq? response 'quit)

;; Quits the game

(printf "So Long.\n")

(**exit**)))))))

## Database

;; Creating the objects

(**define** objects '((**1** "a clock")

(**1** "a sofa")

(**3** "a garage remote")

(**3** "a shoe")

(**7** "an escape rope")

(**9** "a knife")

(**10** "a basement remote")))

;; A number of hash tables are defined in order to support the mutable data structure

;; that will contain our reference to the objects available in the rooms and the user inventory

(**define** objectdb (**make-hash**))

(**define** inventorydb (**make-hash**))

;; This function adds an object to the database

(**define** (**add-object** db id object)

;; Checks if hash key exists

(**if** (hash-has-key? db id)

;; If exists, program assigns to record the content of the key id

(**let** ((**record** (**hash-ref** db id)))

;; The new item is added, followed by the old items stored in the database.

(**hash-set!** db id (cons object record)))

;; if the key is not used, then the item is added directly.

(**hash-set!** db id (cons object empty))))

;; Use the for-each function to add every single item in the assoc list to the objectdb using

;; the add-object function.

;; The (first r) returns the index of the objects assoc list room

;; The (second r) returns the item of the objects assoc list room

(**define** (**add-objects** db)

(**for-each**

(λ (**r**)

;; Each of the list is added to the object db in order.

(**add-object** db (**first** r) (**second** r))) objects))

;; Populate objectdb with the items provided in the object

(**add-objects** objectdb)

;; Displays the objects that are in the room and/or in our inventory

(**define** (**display-objects** db id)

;; If hash key exist

(**if** (hash-has-key? db id)

(**begin**

;; Join string if there are multiple items

(**let\*** ((**record** (**hash-ref** db id))

(**output** (**string-join** record " and ")))

;; Depending on the parameters, provide different answers

(**if** (not (equal? output ""))

;; If the id is bag

(**if** (eq? id 'bag)

;; Print items in inventory

(printf "You are carrying ~a.\n" output)

;; Print items in room

(printf "You can see ~a.\n" output))

(**empty-room-bag** id))))

(**empty-room-bag** id)))

;; Remove object from database

(**define** (**remove-object-from-room-and-inventory** db id str)

;; Checks if hash key exists

(**when** (hash-has-key? db id)

;; If exists,

(**let\*** ((**record** (**hash-ref** db id))

;; result is the list of items in the room

(**result** (**remove** (λ (**x**) (**string-suffix-ci?** str x)) record))

;; item is the difference with the previous list e.g. the item collected

(**item** (**lset-difference** equal? record result)))

;; Checks if the item is not null

(**cond** ((null? item)

;; If the id matches the id in the inventory

;;(if (eq? id 'bag)

(printf "You are not carrying that item!\n")

(printf "I dont see that item in the room!\n"))

(**else**

(**cond**

((eq? id 'bag)

;; Game notifies user which item has been removed

(printf "Removed ~a from your bag.\n" (**first** item))

;; A removed item from inventory is put back into the room

(**add-object** objectdb id (**first** item))

;; Updates inventory

(**hash-set!** db 'bag result))

;; If the item id matches the item in the room

(**else**

;; When the user picks up an item from a room

(printf "Added ~a to your bag.\n" (**first** item))

;; The item s added to the inventory

(**add-object** inventorydb 'bag (**first** item))

;; New object is added

(**hash-set!** db id result)

;; A condition to check if the user has picked up a specific item.

(**cond**

((eq? (**first** item) "an escape rope")

;; Shows message that the user has escaped the haunted house.

(printf "You have found the escape rope, you have successfully escaped from the Haunted House!")

;; User completes game, game terminates.

(**exit**))

;; If the item is picked up is a garage remote

((eq? (**first** item) "a garage remote")

(printf "You have found the garage remote. The garage is now unlocked!"))

;; If the item is picked up is a basement remote.

((eq? (**first** item) "a basement remote")

(printf "You have found the basement remote. The basement is now unlocked!"))

((eq? (**first** item) "a knife")

(printf "Your anxiety is unstable. You killed yourself...")

(**exit**))

;; The new object is saved

(**hash-set!** db id result)))))))))

;; Remove an object from your inventory

(**define** (**remove-object-from-inventory** db id str)

;; Checks if hash key exists

(**when** (hash-has-key? db 'bag)

;; If exists

(**let\*** ((**record** (**hash-ref** db 'bag))

(**result** (**remove** (λ (**x**) (**string-suffix-ci?** str x)) record))

(**item** (**lset-difference** equal? record result)))

(**cond** ((null? item)

(printf "You are not carrying that item !\n"))

(**else**

(printf "Removed ~a from your bag.\n" (**first** item))

(**add-object** objectdb id (**first** item))

(**hash-set!** db 'bag result))))))

## Items

;; Creating the objects

(**define** objects '((**1** "a clock")

(**1** "a sofa")

(**3** "a garage remote")

(**3** "a shoe")

(**7** "an escape rope")

(**9** "a knife")

(**10** "a basement remote")))

;; Maps the paramter to the list of atoms then joins it.

(**define** (**slist->string** l)

(**string-join** (map symbol->string l)))

;; Picking up objects

(**define** (**pick-item** id input)

;; Only the name of the item is retrieved

(**let** ((**item** (**string-join** (cdr (**string-split** input)))))

(**remove-object-from-room-and-inventory** objectdb id item)))

;; Removing objects from inventory and adding it to the room

(**define** (**put-item** id input)

;; Only the name of the item is retrieved.

(**let** ((**item** (**string-join** (cdr (**string-split** input)))))

(**remove-object-from-inventory** inventorydb id item)))

## Displaying Objects and Rooms

;; Displays to user if there are no items in the room or in their inventory

(**define** (**empty-room-bag** id)

;; If the user wants to check their inventory

(**if** (eq? id 'bag)

;; Displays message notifying users they have no items

(printf "You have no items in your inventory.\n")

;; If user does not check their inventory, check the room.

(printf "There are no items in this room.\n")))

;; Displays the objects that are in the room and/or in our inventory

(**define** (**display-objects** db id)

;; If hash key exist

(**if** (hash-has-key? db id)

(**begin**

;; Join string if there are multiple items

(**let\*** ((**record** (**hash-ref** db id))

(**output** (**string-join** record " and ")))

;; Depending on the parameters, provide different answers

(**if** (not (equal? output ""))

;; If the id is bag

(**if** (eq? id 'bag)

;; Print items in inventory

(printf "You are carrying ~a.\n" output)

;; Print items in room

(printf "You can see ~a.\n" output))

(**empty-room-bag** id))))

(**empty-room-bag** id)))

;; Displays what is inside your inventory

(**define** (**display-inventory**)

(**display-objects** inventorydb 'bag))

;; Displays the description

(**define** (**display-description** id)

;; Outputs the text from the matching id

(printf "~a\n" (**get-description** id)))

## Actions

; Define some actions which will include into our decisiontable structure.

; The first is the user's input in the command line and the second is for the software to understand

; what the user's input is associated with.

(**define** look '(((**directions**) look) ((**look**) look) ((**examine** room) look)))

(**define** quit '(((exit game) quit) ((**quit** game) quit) ((**exit**) quit) ((**quit**) quit) ((**end**) quit)))

(**define** pick '(((**get**) pick) ((**pickup**) pick) ((**retrieve**) pick) ((**pick**) pick) ((**obtain**) pick)))

(**define** drop '(((**put**) drop) ((**drop**) drop) ((**place**) drop) ((**remove**) drop) ((**release**) drop)))

(**define** bag '(((**inventory**) bag) ((**bag**) bag) ((**items**) bag) ((**i**) bag)))

(**define** help '(((**help**) help)))

(**define** use '(((**use**) use)))

; Quasiquote is used to apply special properties.

; Unquote-splicing is used because we need to remove the extra list that would

; be generated had we just used unquote.

(**define** actions `(,@look ,@quit ,@pick ,@drop ,@bag ,@use ,@help))

(**define** decisiontable `((**1** ((**north**) **2**) ((**south**) **8**) ((**east**) **9**) ,@actions)

(**2** ((**south**) **1**) ((**north**) **7**) ((**west**) **10**) ,@actions)

(**3** ((**east**) **6**) ((**north**) **5**) ,@actions)

(**4** ((**north**) **8**) ((**south** west) **5**) ((**east**) **7**) ,@actions)

(**5** ((**south**) **3**) ((**north** east) **4**) ((**west**) **6**) ,@actions)

(**6** ((**west**) **3**) ((**east**) **5**) ,@actions)

(**7** ((**west**) **4**) ,@actions)

(**8** ((**north**) **1**) ((**south**) **4**) ,@actions)

(**9** ((**west**) **1**) ((**east**) **10**) ,@actions)

(**10** ((**west**) **9**) ((**east**) **2**) ,@actions)))

## Getting ID

;; This function examines the data for a single room and looks for direction entries, e.g ((north) 2).

;; It does this by checking for a number at the second position of each list

(**define** (**get-directions** id)

;; List decisiontable assigns the id of the decisiontable searched to record

(**let\*** ((**record** (assq id decisiontable))

(**assign** (assq id rooms)))

;; The cdr of record goes through filter. If the second value is a number (a room), it is assigned to result.

;; Length n is the possible rooms the user can access in their current position.

(**let\*** ((**result** (**filter** (λ (**n**) (number? (**second** n))) (cdr record)))

(**n** (length result)))

;; Conditions to check the directions

(**cond** ((= **0** n)

;; If there are no directions

(printf "You appear to have entered a room with no exits. \n"))

;; Get the direction and extract them to string using the slist->string function.

((= **1** n)

(printf "You can see an exit to the ~a.\n" (**slist->string** (caar result))))

(**else**

(**let\*** (;; losym in this instant creates a list by extracting the id of the rooms

;; and consing it with the direction to make a list

(**losym** (map (λ (**x**) (**begin** (cons (car x) (cdr (assv (**second** x) rooms))))) result))

;; lostr gets the list from losym and converts every list into a string using symbol->string.

(**lostr** (map (λ (**x**) (**string-join** (list (symbol->string (caar x))

(**second** x) ""))) losym)))

;;(lostr (map (λ (x) (slist->string x)) losym)))

(printf "You can see exits to the ~a.\n" (**string-join** lostr "and "))))))))

;; Generates a keyword list based on the id given

(**define** (**get-keywords** id)

;; Lets the assq-ref of the id we choose in decision table hold into 'keys'

(**let** ((**keys** (**assq-ref** decisiontable id)))

;; Maps the car of the keys into key. The car of ((north) 1) is (north).

(map (λ (**key**) (car key)) keys)))

;; A wrapper function which allows us to supply an id and list of tokens

;; Returns the matching id of the room

(**define** (**lookup** id tokens)

;; Let the record hold the tokens in decisiontable with a id you provide

(**let\*** ((**record** (**assq-ref** decisiontable id))

;; Keylist holds the keywords from given id

(**keylist** (**get-keywords** id))

;; Holds the index largest number from list-of-lengths into index

(**index** (**index-of-largest-number** (**list-of-lengths** keylist tokens))))

(**if** index

;; Lists the cadr of the id of the decisiontable and its index given.

(cadr (list-ref record index))

**#f**)))

;; Returning the position of the largest value in our list

(**define** (**index-of-largest-number** list-of-numbers)

;; Sorts the list of numbers from the greatest value

(**let** ((**n** (car (**sort** list-of-numbers >))))

;; Checks if list is empty

(**if** (zero? n)

**#f**

;; Returns the index of the greatest value

(**list-index** (λ (**x**) (eq? x n)) list-of-numbers))))

;; Accepts a keyword list and does a match against a list of tokens

;; Outputs the list in the form: (0 0 0 2 0)

(**define** (**list-of-lengths** keylist tokens)

(**map**

(λ (**x**)

(**let** ((**set** (**lset-intersection** eq? tokens x)))

;; apply some weighting to the result

(\* (/ (length set) (length x)) (length set))))

keylist))

;; Retrieves description of the location

(**define** (**get-description** id)

(car (**assq-ref** description id)))

## Help Window

(**define** (**help-window**)

(printf "\n This is the Help Window.\n

Commands:\n

- (directions | look | examine room): Use these commands to find out more information about the room.\n

- (get | pickup | retrieve | pickup | obtain): Use these commands to pick up an item that is in the room you are currently in.\n

You will need to specify what item to pick up.\n

- (put | drop | place | remove | release): Use these commands to discard an item fron your inventory.\n

- (inventory | bag | items | i |): Use these commands to see what items you currently own.\n

- (help): This command will display the help window.\n

- (use): This command will enable you to use a key (If you have a key in your inventory).\n

- (exit game | quit game | exit | quit | end): These commands will close the game.\n"))

## Reference Association List

;; This function retrieves the cdr of the assqlist pair where the id is equal to.

(**define** (**assq-ref** assqlist id)

(cdr (assq id assqlist)))

# Git Log

commit 5dad2e8a488751e4bb0cffbe27364098b488eb55

Author: D WU <D WU>

Date: Thu Apr 13 16:40:39 2017 +0100

Final Code

commit 9f81313fb1d8ffa25d1bd2caaa1e04a56d7569cf

Author: D WU <D WU>

Date: Wed Apr 12 21:33:42 2017 +0100

Created a high order function to handle removing objects from room and inven tory. Also implemented previous code into the new function.

commit 6437f268e7f18b0e54d5d4d4a09c7bf18b727354

Author: D WU <D WU>

Date: Tue Apr 11 20:16:50 2017 +0100

Added a 'locked' room. Updated comments and more

commit e54988aab1c1386bc19493e3c08f4134650f00a2

Author: Deric Wu <21258784@student.uwl.ac.uk>

Date: Mon Mar 20 19:50:03 2017 +0000

:...skipping...

commit 5dad2e8a488751e4bb0cffbe27364098b488eb55

Author: D WU <D WU>

Date: Thu Apr 13 16:40:39 2017 +0100

Final Code

commit 9f81313fb1d8ffa25d1bd2caaa1e04a56d7569cf

Author: D WU <D WU>

Date: Wed Apr 12 21:33:42 2017 +0100

Created a high order function to handle removing objects from room and inventory. Also implemented previous code into the new function.

commit 6437f268e7f18b0e54d5d4d4a09c7bf18b727354

Author: D WU <D WU>

Date: Tue Apr 11 20:16:50 2017 +0100

Added a 'locked' room. Updated comments and more

commit e54988aab1c1386bc19493e3c08f4134650f00a2

Author: Deric Wu <21258784@student.uwl.ac.uk>

Date: Mon Mar 20 19:50:03 2017 +0000

Implemented a new lostr and losym code to handle the concatination of directions and rooms.

Also added a games idea file.

commit 7a8c851339552fdb3e9246da7427145d3d3dcbff

Author: Deric Wu <21258784@student.uwl.ac.uk>

Date: Mon Mar 20 16:19:20 2017 +0000

Added a function to display objects in the room and our inventory

commit 195fc40580c2c2f191f8123768d968577a9fc901

Author: Deric Wu <21258784@student.uwl.ac.uk>

Date: Mon Mar 13 18:33:16 2017 +0000

Added more action such as pick, drop and inventory. Also added a hash database for objects

commit c8fef8e8ee0ac52e5656db6dc18897ffd877a7c9

Author: Deric <21258784@student.uwl.ac.uk>

Date: Mon Mar 13 17:38:48 2017 +0000

Set theme jekyll-theme-cayman

commit 066dd088756a34023940efff3ea708e53eea6ce8

Author: Deric Wu <21258784@student.uwl.ac.uk>

Date: Mon Mar 13 17:33:44 2017 +0000

New code has been added for the assignment based on the MUD

commit f119160b1205e55cfd3d4916930c575d6413f58f

Author: Deric Wu <21258784@student.uwl.ac.uk>

Date: Mon Mar 6 17:54:31 2017 +0000

Updated the definitions for FunctionalProgramming assignment 2