Coffee Maker Quest

Austin Choi, Charlotte Chen

CS 1632 – DELIVERABLE 1: Test Plan and Traceability

Some of our concerns and difficulties arose from the ambiguity of the requirements. For example, the requirement FUN-MOVE states that player should not move north or south if there is no door leading in each respective direction, but it does not specify if the act of teleporting into a magical land and back to the beginning is considered ‘moving’. In this case, we assumed that player did not ‘move’ unless he was transported to a different room. Consequently, when the player was teleported to the beginning from the northernmost room after trying to move north, we considered the behavior a defect. When the player tried to move south in the starting room, he remained in the same spot despite being teleported to a magical land; we considered this behavior normal.

We also noticed that there was an overlap in some of the requirements. The requirement, FUN-ITERATION, specifies that the player should be able to enter six different commands, but the other requirements, FUN-INVENTORY, FUN-LOOK, and FUN-HELP, all specified functionality for a specific key. Obviously, if one of those requirements were satisfied, then FUN-ITERATION would also be satisfied for its respective command. In this case, we only made test cases for FUN-ITERATION that examined the movement commands, since these keys did not have their own separate requirements.

In terms of edge cases, we considered all abnormal inputs that a user might enter (intentionally or accidentally) on a command line program: special characters, blank values, very long values, etc.

—TEST CASES—

IDENTIFIER: 1

TEST CASE: Ensure that the directional command ‘N’ causes the player to move north.

PRE-CONDITIONS: Player is in a room.

INPUT VALUES: N

EXECUTION STEPS: Input the value on the command line and hit enter

OUTPUT VALUES: N/A

POST-CONDITIONS: If door exists north, player should end up in a new room. If no north door exits, player should remain in existing room

-------------------------------------------------------

IDENTIFIER: 2

TEST CASE: Ensure that upon entering ‘H’, the player is shown a list of commands

PRE-CONDITIONS: None

INPUT VALUES: H

EXECUTION STEPS: Input each value in order.

OUTPUT VALUES: N/A

POST-CONDITIONS: For each of the input value, program displays a list of commands and their corresponding effects

-------------------------------------------------------

IDENTIFIER: 3

TEST CASE: Ensure that upon entering a numerical value, the program does not execute any command.

PRE-CONDITIONS: None

INPUT VALUES: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

EXECUTION STEPS: Input each value in order.

OUTPUT VALUES: N/A

POST-CONDITIONS: For each of the input value, program displays a “What?” message. Player remains in the same room

-------------------------------------------------------

IDENTIFIER: 4

TEST CASE: Ensure that upon entering a letter not corresponding to the keys in the requirement “FUN-ITERATION”, the program does not execute any command.

INPUT VALUES: q, Q, w, W, e, E, r, R, t, T, y, Y, u, U, o, O, p, P

PRE-CONDITIONS: None

EXECUTION STEPS: Input each value in order.

OUTPUT VALUES: N/A

POST-CONDITIONS: For each of the input value, program displays a “What?” message.

-------------------------------------------------------

IDENTIFIER: 5

TEST CASE: Ensure that as the player moves north, all rooms have unique adjectives attached to them

INPUT VALUES: N

PRE-CONDITIONS: Player is in the “Small room”

EXECUTION STEPS: Run the input value “N” in the command line until the player reaches the “Rough room” with no doors leading north

OUTPUT VALUES: N/A

POST-CONDITIONS: Player ends up in “Rough room” and no two rooms they passed on the way have had the same adjective describing them

-------------------------------------------------------

IDENTIFIER: 6

TEST CASE: Ensure that as the player moves south, each room has one and only one unique object

INPUT VALUES: S

PRE-CONDITIONS: Player is in the “Rough room”

EXECUTION STEPS: Run the input value “S” in the command line until the player reaches the “Small room” with no doors leading south

OUTPUT VALUES: N/A

POST-CONDITIONS: Player ends up in “Small room” and every room they passed on the way have had one and only one unique object

-------------------------------------------------------

IDENTIFIER: 7

TEST CASE: Ensure that the player cannot move north after reaching the northernmost room

INPUT VALUES: N

PRE-CONDITIONS: Player is in the “Rough room” with no door leading north

EXECUTION STEPS: Run the input value “N” at the command line and hit enter

OUTPUT VALUES: N/A

POST-CONDITIONS: Player remains in the “Rough room”

-------------------------------------------------------

IDENTIFIER: 8

TEST CASE: Ensure that the player cannot move south after reaching the southernmost room

INPUT VALUES: S

PRE-CONDITIONS: Player is in the “Small room” with no door leading south

EXECUTION STEPS: Run the input value “S” at the command line and hit enter

OUTPUT VALUES: N/A

POST-CONDITIONS: Player remains in the “Small room”

-------------------------------------------------------

IDENTIFIER: 9

TEST CASE: Ensure that the program will display a success/win message when user collected all three of coffee, sugar, and cream and entered the “drink” command

INPUT VALUES: D

PRE-CONDITIONS: User has found all 3 items: coffee, sugar, and cream

EXECUTION STEPS: Run input value “D” and hit enter

OUTPUT VALUES: N/A

POST-CONDITIONS: Program displays “You win!” and exits without error message

-------------------------------------------------------

IDENTIFIER: 10

TEST CASE: Ensure that the program will display a losing message when user collected less than three of the required items and entered the “drink” command

INPUT VALUES: D

PRE-CONDITIONS: Users has less than three of the required items: coffee, sugar, and cream

EXECUTION STEPS: Run input value “D” in the command line and hit enter

OUTPUT VALUES: N/A

POST-CONDITIONS: Program displays “You lose!” and exits without error message

-------------------------------------------------------

IDENTIFIER: 11

TEST CASE: Ensure that when user does not input any value and directly hit enter, the program does not execute any command, crash, or exit

INPUT VALUES: None

PRE-CONDITIONS: None

EXECUTION STEPS: Directly hits enter without inputting any value

OUTPUT VALUES: N/A

POST-CONDITIONS: Program displays a “What?” message and remains in the same room

-------------------------------------------------------

IDENTIFIER: 12

TEST CASE: Ensure that when the user inputs a special character and hits enter, the program does not execute any command

INPUT VALUES: !, @, #, $, %, ^, &, \*, (, ), +, =

PRE-CONDITIONS: None

EXECUTION STEPS: Input each value in order

OUTPUT VALUES: N/A

POST-CONDITIONS: For each of the input value, program displays a “What?” message and remains in the same room

-------------------------------------------------------

IDENTIFIER: 13

TEST CASE: Ensure that lower case commands will generate the same response as their upper case commands

PRE-CONDITIONS: None

INPUT VALUES: n, s, l, h, d

EXECUTION STEPS: Enter each lower case input value on the command line

OUTPUT VALUES: N/A

POST-CONDITIONS: For each of the input value, the program will generate the same response as their upper-case correspondents (‘N’, ‘S’, ‘L’, ‘H’, and ‘D’) do in the same situation

-------------------------------------------------------

IDENTIFIER: 14

TEST CASE: Ensure that when user type “I”, the program shows the items he/she has collected

INPUT VALUES: I

PRE-CONDITIONS: None

EXECUTION STEPS: Run input value on the command line and hit enter

OUTPUT VALUES: N/A

POST-CONDITIONS: The program displays the items collected and inform of the items not yet collected; player remains in the same room

-------------------------------------------------------

IDENTIFIER: 15

TEST CASE: Ensure program updates inventory immediately without needing to enter a new room. If user collected a new item in the current room and immediately checked their inventory, the program shows all items collected (including the item just collected in the current room)

INPUT VALUES: ‘L’ and ‘I’

PRE-CONDITIONS: (1) The current room contains an item (2) User has not “looked” yet in the current room (3) User input “I” to check inventory before proceeding to a new room

EXECUTION STEPS: Upon entering a room containing an item, enter the input values ‘L’ and ‘I’ in order

OUTPUT VALUES: N/A

POST-CONDITIONS: Program displays the list of the items collected, which must include the item just collected in the current room

-------------------------------------------------------

IDENTIFIER: 16

TEST CASE: Ensure that when user enters “L” for look, any items in the room will be added to the user’s inventory

INPUT VALUES: ‘L’

PRE-CONDITIONS: There exists an item in the current room

EXECUTION STEPS: Input value ‘L’ in the command line

OUTPUT VALUES: N/A

POST-CONDITIONS: Program shows message informing the identity of the item found and added to the user’s inventory

-------------------------------------------------------

IDENTIFIER: 17

TEST CASE: Ensure collecting the same item twice does not cause the program to crash or exit

INPUT VALUES: ‘L’

PRE-CONDITIONS: Item in the current room has already been collected and added to inventory

EXECUTION STEPS: Input value ‘L’ in the command line

OUTPUT VALUES: N/A

POST-CONDITIONS: Program shows message informing the identity of the item found. User remains in the same room and can proceed to the next room without error

-------------------------------------------------------

IDENTIFIER: 18

TEST CASE: Ensure that if a room does not contain any item and user type ‘L’ to look, program should show “you don’t see anything out of the ordinary” and do nothing else

INPUT VALUES: ‘L’

PRE-CONDITIONS: Current room does not contain any of the 3 items

EXECUTION STEPS: Input value ‘L’ in the command line

OUTPUT VALUES: N/A

POST-CONDITIONS: Program displays message “you don’t see anything out of the ordinary”. Nothing is added to the inventory and user remains in the same room

-------------------------------------------------------

IDENTIFIER: 19

TEST CASE: Ensure that collecting duplication of a single item is equivalent to have collected just one of that item, and does not affect the inventory list when user input ‘I’

INPUT VALUES: ‘L’, ‘I’

PRE-CONDITIONS: User is in a room that contains an item that he/she has already “looked” and included in his/her inventory

EXECUTION STEPS: Type the input values on the command line in order

OUTPUT VALUES: N/A

POST-CONDITIONS: Inventory list shows all items collected (including in the item in the current room). Collecting duplicates of that same item does not add its numerical quantity nor eliminate it from the inventory list. Program proceeds without error and player remains in the same room

-------------------------------------------------------

IDENTIFIER: 20

TEST CASE: Ensure that the directional command ‘S’ causes the player to move south.

PRE-CONDITIONS: Player is in a room.

INPUT VALUES: S

EXECUTION STEPS: Input the value on the command line and hit enter

OUTPUT VALUES: N/A

POST-CONDITIONS: If door exists south, player should end up in a new room. If no south door exits, player should remain in existing room

-------------------------------------------------------

IDENTIFIER: 21

TEST CASE: Ensure that the program does not crash or behave abnormally when accepting a large input.

INPUT VALUES: N/A

PRE-CONDITIONS: Game is running and accepting input.

EXECUTION STEPS: Copy and paste a paragraph that is at least 200 characters long into the command line. Hit enter. Quit the program.

OUTPUT VALUES: N/A

POST-CONDITIONS: The program should respond with the phrase “What?” to the input. The program should exit without any errors.

—TRACEABILITY MATRIX—

FUN-ITERATION: 1, 20

FUN-UNKNOWN-COMMAND: 3, 4, 11, 12, 21

FUN-INPUT-CAPS: 13

FUN-MOVE: 7, 8

FUN-WIN: 9

FUN-LOSE: 10

FUN-INVENTORY: 14, 15, 19

FUN-LOOK: 16, 17, 18

FUN-HELP: 2

FUN-UNIQ-ROOM: 5

FUN-UNIQ-ROOM-FURNISHING: 6

— DEFECT 1 —

DESCRIPTION: Lowercase n is not a valid command

SUMMARY: When lowercase n is input, the program responds with ‘What?’ instead of moving the player north. This behavior conflicts with the requirement FUN-INPUT-CAPS and is found when implementing test case #13.

REPRODUCTION STEPS: Input n and hit enter

EXPECTED BEHAVIOR: Player will move north into a new room.

OBSERVED BEHAVIOR: Program responds with “What?”. Player remains in the same room.

— DEFECT 2 —

DESCRIPTION: ‘H’ is not a valid command.

SUMMARY: When the user input “H” or “h”, the program responds with “What?” instead of showing the user a help page consisting the list of possible commands and their effects. This behavior conflicts with the requirement FUN-HELP and is found when implementing test case #2.

REPRODUCTION STEPS: User input “H” or “h” and hit enter.

EXPECTED BEHAVIOR: Program should show a list of possible commands and their effects.

OBSERVED BEHAVIOR: Program does not recognize command and displays “What?”. Player remains in the same room.

— DEFECT 3 —

DESCRIPTION: User exits with error code “Exiting with error code 0” when won

SUMMARY: When the user wins the game, upon showing the user “you win!” the program always displays afterward: “Exiting with error code 0”. This behavior conflicts with the requirement FUN-WIN and is found when implementing test case #9.

REPRODUCTION STEPS: After user collected all 3 items, he/she types “D” and hit enter

EXPECTED BEHAVIOR: Program should display “You drink the beverage and are ready to study!”, “You win!”, and automatically exit the program.

OBSERVED BEHAVIOR: Program displays “You drink the beverage and are ready to study!” and “You win!”, but it also displays “Exiting with error code 0” before quitting the program.

— DEFECT 4 —

DESCRIPTION: User is able to move north in a room without a north door.

SUMMARY: In the rough room with no north door, if the player inputs the command to move north, he is transported to a ‘magical land’ and is returned to the beginning of the game. This behavior conflicts with the requirement FUN-MOVE and is found when implementing test case #7.

REPRODUCTION STEPS: Reach the rough room. Input N.

EXPECTED BEHAVIOR: The player should remain in the rough room.

OBSERVED BEHAVIOR: The program displays “You are in a magical land! But you are returned to the beginning!” The player is teleported back to the small room.

— DEFECT 5 —

DESCRIPTION: The program displays an error message when a large input has been entered.

SUMMARY: After inputting a paragraph of about 250 characters into the command line, the program responds normally with a ‘What?’ message. However, after exiting the program, the error message ‘ Exception in thread main” ’ appears.

REPRODUCTION STEPS: Copy and paste a paragraph into the command line. Make sure it contains around 250 characters. Hit enter. Quit the program.

EXPECTED BEHAVIOR: The program should respond with “What?” to the input. The program should exit with no errors.

OBSERVED BEHAVIOR: The program responds with “What?” to the input. The program exits with ‘ Exception in thread “main” ’.