Professor Wumpus

Edmund Anderson and Alec Brumbaugh

CS 1632 – DELIVERABLE 1: Test Plan and Traceability Matrix

Introduction

Professor Wumpus is the newest Java-gaming sensation from Bill Laboon that is taking the Pitt CS Department by storm. The game is played in a 6 by 6 matrix of *rooms*, beautifully crafted from ASCII brackets in the state-of-the-art console. The user will work their way through this grid of *rooms* by entering the characters *N,S,E*, or *W*, or their respective lowercase counterparts, corresponding to North, South, East, or West. Besides the user, there are two other characters in the game, *Professor Wumpus,* and a *TA.* While the *TA* moves, *Professor Wumpus* is stationary. There is also an *assignment*. The goal is to, whilst avoiding the *TA*, first find the *assignment*, and only after this is found, find *Professor Wumpus*. When the user is occupying a room that is directly North, South, East, or West from either the *TA* or *Professor Wumpus,* they each give indications (console output alerts) that they are near. Once the game is either won or lost, the program will end.

While the game is simple enough to understand in a paragraph, the problems that can occur are plentiful. The difficulties that are expected will be to map a random number seed to the resulting gameplay setup, i.e. the initial layout of the TA and Professor Wumpus on the room matrix. The location of the *Professor* and the *assignment* are easier to place, since they do not move (hopefully). This is how we will be able to test cases where we locate the *Professor* with or without the assignment. The TA will require a more thoughtful approach, as TA can spawn anywhere, and will move about. Because we want the test cases to be replicable, they will be more unique when dealing with the TA. Other than that, we anticipate that with a relatively small number of tests, we will be able to find a high percentage of all actual errors that could be encountered during normal gameplay.

We will consider a handful of specific test cases. First, we will want to test that the program displays the Matrix correctly. Next, all of the “correct” inputs should be tested, as well as some erroneous input. The walls will also need tested. A corner case will be multiple consecutive run ins with the walls, as we are not sure how they are represented in the program. We also want to check for case-sensitivity, to ensure ‘N’ and ‘n’ give the same intended function, for all of the possible directions. Additionally, we will check for some corner and edge cases, like input of the incorrect datatype, as well as a seed that is outside the valid range for the datatype requested, since functionality is given in the requirements for such cases.

In some test cases, the position of a character or item in the matrix is given in (x,y) form, x (column) and y (row) are measured in rooms from the top left position (0x0), and the bottom right position would be (6,6). Although it was unnecessary to number the test cases as well as list separate identifiers, we did so for tracking purposes, and felt it stupid to remove the numbering.

Test Case 1.

IDENTIFIER: VALID-PARAMETER

DESCRIPTION: Run the program with a valid parameter

INPUT VALUES: 20

PRECONDITIONS: None

EXECUTION STEPS:

At command line, enter:

*java -jar profwumpus.jar 20*

POSTCONDITIONS: Program is running

Test Case 2.

IDENTIFIER: VALID-PARAMETER-NONE

DESCRIPTION: Run the program with no parameter

INPUT VALUES:

PRECONDITIONS: None

EXECUTION STEPS:

At command line, enter:

*java -jar profwumpus.jar*

POSTCONDITIONS: Program is running

Test Case 3.

IDENTIFIER: VALID-PARAMETER-HUGE

DESCRIPTION: Run the program with an invalid seed \*\*Corner Case\*\*

INPUT VALUES: 3000000000

PRECONDITIONS: None

EXECUTION STEPS:

At command line, enter:

*java -jar profwumpus.jar 3000000000*

POSTCONDITIONS: Program is running as if no seed was given

Test Case 4.

IDENTIFIER: INPUT-S

DESCRIPTION: Input southward moving input

INPUT VALUES: S

PRECONDITIONS: The character should be anywhere but the bottom row.

EXECUTION STEPS:

1. *Type “S”*
2. *Press enter*

POSTCONDITIONS: The program should move the player South one room.

Test Case 5.

IDENTIFIER: INPUT-E

DESCRIPTION: Input eastward moving input

INPUT VALUES: E

PRECONDITIONS: The character should be in any room that isn’t in the Easternmost column

EXECUTION STEPS:

1. *Type “E”*
2. *Press enter*

POSTCONDITIONS: The program should move the player East one room

Test Case 6.

IDENTIFIER: INPUT-W

DESCRIPTION: Input westward moving input

INPUT VALUES: W

PRECONDITIONS: The character should be in any room that isn’t in the Westernmost column

EXECUTION STEPS:

1. *Type “W”*
2. *Press enter*
3. *Type “w”*
4. *Press enter*

POSTCONDITIONS: The program should move the player West one rooms.

Test Case 7.

IDENTIFIER: INPUT-N

DESCRIPTION: Input northward moving input

INPUT VALUES: N

PRECONDITIONS: The character should be in any room but not in the top row.

EXECUTION STEPS:

1. *Type “N”*
2. *Press enter*

POSTCONDITIONS: The program should move the player North one room.

Test Case 8.

IDENTIFIER: INPUT-CASE-SAME

DESCRIPTION: Input both northward moving inputs

INPUT VALUES: N, n

PRECONDITIONS: The character should be in any room, but not in the top two rows.

EXECUTION STEPS:

1. *Type “N”*
2. *Press enter*
3. *Type “n”*
4. *Press Enter*

POSTCONDITIONS: The program should move the player North two rooms

Test Case 9.

IDENTIFIER: INPUT-CASE-SAME-2

DESCRIPTION: Input both eastward moving inputs

INPUT VALUES: E, e

PRECONDITIONS: The character should be in any room, but not in the Easternmost two rows.

EXECUTION STEPS:

1. *Type “E”*
2. *Press enter*
3. *Type “e”*
4. *Press Enter*

POSTCONDITIONS: The program should move the player East two rooms

Test Case 10.

IDENTIFIER: INPUT-INVALID-UPPER

DESCRIPTION: Test an invalid input \*\*Edge Case\*\*

INPUT VALUES: X

PRECONDITIONS: The program is running

EXECUTION STEPS:

1. *Type “X”*
2. *Press enter*

POSTCONDITIONS: The program should output “Please enter N, S, E, or W”.

Test Case 11.

IDENTIFIER: INPUT-INVALID-LOWER

DESCRIPTION: Test an invalid input in lower case \*\*Edge Case\*\*

INPUT VALUES: a

PRECONDITIONS: The program is running

EXECUTION STEPS:

1. *Type “a”*
2. *Press enter*

POSTCONDITIONS: The program should output “Please enter N, S, E, or W”.

Test Case 12.

IDENTIFIER: INPUT-INVALID-INT

DESCRIPTION: Test an invalid input as an integer \*\*Edge Case\*\*

INPUT VALUES: 78 (char ‘N’ in Decimal)

PRECONDITIONS: The program is running

EXECUTION STEPS:

1. *Type “78”*
2. *Press enter*

POSTCONDITIONS: The program should output “Please enter N, S, E, or W”.

Test Case 13.

IDENTIFIER: INPUT-INVALID-NONE

DESCRIPTION: Test an invalid input, no input given \*\*Edge case\*\*\*

INPUT VALUES:

PRECONDITIONS: The program is running

No input is typed in console

EXECUTION STEPS:

1. *Press enter*

POSTCONDITIONS: The program should output “Please enter N, S, E, or W”.

Test Case 14.

IDENTIFIER: WALL-EAST

DESCRIPTION: Test a collision with the user and the east wall

INPUT VALUES: E

PRECONDITIONS: The character should be in the sixth column of the matrix (farthest East)

EXECUTION STEPS:

1. *Type “E”*
2. *Press enter*

POSTCONDITIONS: The program should output “There’s a wall there, buddy!”

Test Case 15.

IDENTIFIER: WALL-WEST

DESCRIPTION: Test a collision with the user and the west wall

INPUT VALUES: W

PRECONDITIONS: The character should be in the first column of the matrix (farthest West)

EXECUTION STEPS:

1. *Type “W”*
2. *Press enter*

POSTCONDITIONS: The program should output “There’s a wall there, buddy!”

Test Case 16.

IDENTIFIER: DISPLAY-CHECK

DESCRIPTION: Test that startup displays 6x6 matrix of rooms

INPUT VALUES: None

PRECONDITIONS: None

EXECUTION STEPS:

1. At command line, enter: *java -jar profwumpus.jar 20*
2. Observe ASCII output

POSTCONDITIONS: Program is running and displaying ASCII rooms in a 6x6 grid, one of the 36 rooms in the grid containing an “S”

Test Case 17.

IDENTIFIER: WUMPUS-PROXIMITY-SOUTH

DESCRIPTION: Test that Professor Wumpus alerts you when he is to the South of user

INPUT VALUES: “S”

PRECONDITIONS: Program is running with seed value of 20.

Player is in position (1,2)

EXECUTION STEPS:

1. Type “S”
2. Press Enter

POSTCONDITIONS: Program outputs “You hear someone pontificating on Computer Science… Professor Wumpus must be nearby!”

Test Case 18.

IDENTIFIER: WUMPUS-PROXIMITY-NORTH

DESCRIPTION: Test that Professor Wumpus alerts you when he is to the North of user

INPUT VALUES: “W”

PRECONDITIONS: Program is running with seed value of 20.

Player is in position (0,5)

EXECUTION STEPS:

1. Type “E”
2. Press Enter

POSTCONDITIONS: Program outputs “You hear someone pontificating on Computer Science… Professor Wumpus must be nearby!”

Test Case 19.

IDENTIFIER: WUMPUS-PROXIMITY-EAST

DESCRIPTION: Test that Professor Wumpus alerts you when he is to the East of user

INPUT VALUES: “S”

PRECONDITIONS: Program is running with seed value of 20.

Player is in position (0,3)

EXECUTION STEPS:

1. Type “S”
2. Press Enter

POSTCONDITIONS: Program outputs “You hear someone pontificating on Computer Science… Professor Wumpus must be nearby!”

Test Case 20.

IDENTIFIER: WUMPUS-PROXIMITY-WEST

DESCRIPTION: Test that Professor Wumpus alerts you when he is to the West of user

INPUT VALUES: “S”

PRECONDITIONS: Program is running with seed value of 20.

Player is in position (3,3)

EXECUTION STEPS:

1. Type “S”
2. Press Enter

POSTCONDITIONS: Program outputs “You hear someone pontificating on Computer Science… Professor Wumpus must be nearby!”

Test Case 21.

IDENTIFIER: MOVE-ROOM-EXISTS

DESCRIPTION: Test that is the character moves into a valid room, the “S” in the matrix moves and a new iteration begins

INPUT VALUES: “S”

PRECONDITIONS: Program has just started running with no further input

EXECUTION STEPS:

1. Type “S”
2. Press Enter

POSTCONDITIONS: Program reprints the matrix, and the “S” character has moved down one from its previous location. Program awaits next input character.

Test Case 22.

IDENTIFIER: WUMPUS-WITH-ASSIGNMENT

DESCRIPTION: Test that game is won after getting assignment and landing on WUMPUS

INPUT VALUES: “E”

PRECONDITIONS: Program is running with seed value of 20.

Player has obtained the assignment

Player is in position (0,4)

EXECUTION STEPS:

1. Type “E”
2. Press Enter

POSTCONDITIONS: Program outputs “You turn in your assignment. YOU WIN!”

Program ends

Test Case 23.

IDENTIFIER: WUMPUS-WITHOUT-ASSIGNMENT

DESCRIPTION: Test that game is lost after landing on WUMPUS without first

getting the assignment

INPUT VALUES: “S”

PRECONDITIONS: Program is running with seed value of 20.

Player has NOT obtained the assignment

Player is in position (2,3)

EXECUTION STEPS:

1. Type “S”
2. Press Enter

POSTCONDITIONS: Program outputs “Prof Wumpus sees you, but you don’t have your assignment. YOU LOSE!”

Program ends.

Test Case 24.

IDENTIFIER: TA-WALL

DESCRIPTION: Test that the when the TA hits a wall, program gives alert

INPUT VALUES: “W” repeatedly

PRECONDITIONS: Program has just begun running with seed value of 22.

Player has not entered any other input

EXECUTION STEPS:

1. Repeat the below process 100 times

1. Type “W”
2. Press Enter

POSTCONDITIONS: Program iterates normally with at least one output “You hear a thud, as if the TA hit into a ……” alert.

Test Case 25.

IDENTIFIER: TA-PROXIMITY

DESCRIPTION: Test that when the TA is within one room of proximity, alert is given

INPUT VALUES: E, E, S, S, W, W, N, N

PRECONDITIONS: Program is running with seed value of 22.

Player is in position (1,1).

EXECUTION STEPS:

1. Repeat the below process 100 times

1. Type “E”
2. Press Enter
3. Type “E”
4. Press Enter
5. Type “S”
6. Press Enter
7. Type “S”
8. Press Enter
9. Type “W”
10. Press Enter
11. Type “W”
12. Press Enter
13. Type “N”
14. Press Enter
15. Type “N”
16. Press Enter

POSTCONDITIONS: Program iterates normally with at least one output “You hear a the shuffling of graded papers… the TA must be nearby!” alert.

Traceability Matrix

REQ-1: DISPLAY-CHECK

REQ-2: INTPUT-S, INPUT-N, INPUT-W, INPUT-E, INPUT-INVALID-LOWER, INPUT-INVALID-NONE, INPUT-INVALID-UPPER, INPUT-INVALID-INT

REQ-3: INPUT-CASE-SAME, INPUT-CASE-SAME-2

REQ-4: MOVE-ROOM-EXISTS

REQ-5: WALL-EAST, WALL-WEST

REQ-6: VALID-PARAMETER, VALID-PARAMETER-NONE

REQ-7: VALID-PARAMETER-HUGE

REQ-8: TA-WALL

REQ-9: WUMPUS-WITH-ASSIGNMENT, WUMPUS-WITHOUT-ASSIGNMENT

REQ-10: WUMPUS-PROXIMITY-NORTH, WUMPUS-PROXIMITY-SOUTH, WUMPUS- PROXIMITY-EAST, WUMPUS-PROXIMITY-WEST

REQ-11: TA-PROXIMITY

Defects

Defect 1.

SUMMARY: The room matrix is not 6x6.

DESCRIPTION: The matrix is listed in the requirements as needing to be 6x6, but when the game is run, only a 5x5 matrix is given viewable in ASCII.

REPRODUCTION STEPS:

Run the program with a valid seed or no seed at all.

EXPECTED BEHAVIOR:

The program displays a 6x6 ASCII room matrix.

OBSERVED BEHAVIOR:

The program displays a 5x5 ASCII room matrix.

Defect 2.

SUMMARY: Easternmost wall throws exception

DESCRIPTION: A collision between the user “Student” and the easternmost wall throws an exception.

REPRODUCTION STEPS:

Launch program with seed 20.

Press E and hit enter 5 times.

On the 5th press, observe behavior.

EXPECTED BEHAVIOR: Program should alert you that a wall is present, cannot move to room

OBSERVED BEHAVIOR: Java Array Index Exception is thrown, program crashes.

Defect 3.

SUMMARY: Invalid seed

DESCRIPTION: On an invalid seed, program should proceed as if no seed is given

REPRODUCTION STEPS:

Run program with command line:

*java -jar profwumpus.jar 3000000000*

EXPECTED BEHAVIOR: Program begins as if no seed is gven

OBSERVED BEHAVIOR: Java number format exception thrown. Program does not start up.