Professor Wumpus

Sumanyu Gupta

Scott Crawford

CS 1632 - DELIVERABLE 1: Test Plan and Traceability Matrix

**Introduction**

Professor Wumpus was an interesting piece of software to test. However, our team encountered some initial difficulties in the first stages of developing our test plan. The biggest issue we dealt with was actually reading the requirements extremely closely and paying attention to them word-by-word. For example, the very first time we read requirement 1, we did not relate the necessity for a “6 by 6 matrix” to the existing 5 by 5 matrix that the game displayed. We were not giving the requirements the detailed attention they required. Additionally, writing these test cases in a way that assumed that the first requirement was already being met, despite it obviously not, was an interesting challenge. In particular, every test case with a postcondition that checks the size of the matrix in anyway should be written with the expected behavior of dealing with a 6 by 6 matrix, and not the observed behavior that we had from dealing with a 5 by 5 matrix.

Another issue we faced was the actual testing of the game. Initially, the mindset we had while testing was to make sure that each test we were developed confirmed that the requirements were or were not being met, with no overtesting. Quickly we realized that the more accurate mindset to testing an application was to try to break it, but in a variety of ways. For instance, during our initial tests for requirement 5 (moving into a room that does not exist), we were satisfied by trying to move North from the first row, South from the fifth row, and West from the first column. Our thinking was that since three attempts had shown the requirement to be programmed correctly, there would be no reason to try breaking it through a fourth attempt on the fifth column of the matrix. NonExistentRoom-Movement-Test2, was developed after realizing that our thinking was flawed and that no part of the matrix was to be left untested. In essence, to solve our difficulties, we had to overcome our incorrect mindset of assuming our tests would not pass without even attempting them.

One requirement that proved especially complicated to test was requirement 8, specifically “The TA shall move a random direction at each iteration.” We believe that this requirement was somewhat incorrect because the TA’s movements were not truly random. Of course, this is based on our own definition of random movement and what we believe the seed is meant to do, but since neither of those were specified in the requirements, there was some confusion when it came to testing. Depending on the seed value provided, we found that the TA would move in a set pattern each time the game was played. A seed of X would move the TA West, East, South, North, but a seed of Y would move the TA North, North, South, South. Thus, their movements were “quasi-random.” To test this requirement, however, we first used two different seeds and figured out a set of movements that would lead us to the TA in at least one of those two games. We felt that this test case properly proved that the TA movement was random at least between two different seeds, and also came to the conclusion that the remaining 67% of requirement 8 were satisfied as well through other tests: that we received a message of the TA bumping into the wall and Professor Wumpus did not move once the game had been started.

Since edge cases are elements next to boundary values in equivalence classes, we realized that the simplest method for developing them was to look for literal edges in any sort of input we would provide in the game. The first edge case was for testing seed values and we first looked at the range of values allowed for a signed 32-bit integer. Since the maximum value allowed is 2147483647 and the minimum is -2147483648, an edge case would be to test a seed one value greater than the maximum and/or one value less than the minimum. A second edge case still dealt with a seed value, but we used an alphanumeric value to see if the game would proceed as normal although it was not a valid seed. Combining numbers, letters, and special characters would cover multiple scenarios in which a large number might be accepted but something non-numeric would not. Finally, the last edge case was regarding the boundaries of the matrix. Attempting to move to a nonexistent room should have resulted in the player being notified about running into a wall, and to test this, the player needed to be on the edges of the matrix. The mindset for this edge case was to test movement in the direction where a room would not exist and we did this for all four cardinal directions. The boundary values were the directions that would cause the message to be displayed.

While developing edge cases were an integral part of the testing process, we discovered some missing requirements as well. We believe that the following actions in the game needed to be formalized requirements because they were integral to the functionality of the game itself and could be tested to make sure the game could be played as the designer had intended it to be. There was no requirement that dealt the player actually locating or finding the assignment. Since the entire point of the game was to win by turning in an assignment to Professor Wumpus, we believe it would have been beneficial to have a requirement listed that stated the necessity of finding the assignment and ensuring what would happen once the player found it (in this case, a displayed message to the console), as well as a requirement similar to 10 and 11 regarding informing the player when the assignment is nearby. In addition to this, there was no requirement to test the fact that a student moves to a random room if he/she encounters the TA in a room. Since the encounter affected the immediate location of the player, a requirement that would allow us, as testers, to confirm the player did indeed “Flee into a random room,” would have added to the overall functionality of the game.

Professor Wumpus presented many challenges while we were testing it. However, it improved our testing mindset and helped us get a great introduction to formally writing a software test plan.

**Test Cases**

IDENTIFIER: Matrix-Size-Test  
DESCRIPTION: This test verifies that the game has a 6 by 6 matrix, which means that player should be able to move 5 rooms East, South, West, or North from their initial position, provided they are not interrupted by either the TA or professor. This test will have the player move south five times, at which point they should be in the sixth row of the matrix.  
PRECONDITIONS: Game should be started by executing with “java -jar profwumpus.jar 2032042993”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter:  “S S S S S”.  
POSTCONDITIONS: After the 5th key press, the player, represented by ‘S’, should be standing in the 6th row of the matrix.

IDENTIFIER: Student-Location-Test  
DESCRIPTION: This test verifies that the location of the Student within the matrix, indicated by an ‘S,’ is displayed properly at each iteration. This test will have the player move one room south, confirming that the ‘S’ has moved within the matrix to show their new and correct location.

PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 2032042993”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “S”.

POSTCONDITIONS: The ‘S’ should now appear in the second row of the matrix, still in the first column, one room below their starting position.

IDENTIFIER: Valid-Movement-Test  
DESCRIPTION: This test verifies that the each of the valid inputs will properly move the player in the cardinal direction that was provided. This test will have the player move South, West, North, and East, in that order, using the correct input for each of those directions.

PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 2032042993”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “S E N W”

POSTCONDITIONS: The player should move one room South, then East, then North, then West, in that order after each input, ending up in the starting position at the end of the execution steps.

IDENTIFIER: Invalid-Movement-Test  
DESCRIPTION: This test verifies that an invalid input of direction by the user results in a message telling the user to provide an acceptable input. This test will have the player input “Laboon”, an invalid input, and verify that they have received the invalid input message.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each word: “Laboon”.

POSTCONDITIONS: The console should display “Please enter N, S, E, or W”, directing the user to provide a valid input.

IDENTIFIER: CaseInsensitiveLower-Input-Test  
DESCRIPTION: This test verifies that whenever the player enters lowercase input for direction of movement, the player moves in that direction. This test will have the player input a lowercase ‘E’ and verify that the Student moves one room to the East.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 2032042993”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “e”. The input should not be capitalized.

POSTCONDITIONS: The player should move one room to the East.

IDENTIFIER: CaseInsensitiveUpper-Input-Test  
DESCRIPTION: This test verifies that whenever the player enters uppercase input for direction of movement, the player moves in that direction. This test will have the player input a uppercase ‘E’ and verify that the Student moves one room to the East.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 2032042993”.

EXECUTION STEPS:  Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E”. The input should be capitalized.

POSTCONDITIONS: The player should move one room to the East.

IDENTIFIER: ExistingRoom-Movement-Test  
DESCRIPTION: This test verifies that whenever the user provides valid input, the player’s position in the game changes to move to the room in the direction specified. This test will have the player move one room to the East from the starting position, which should be a valid move.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 2032042993”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E”.

POSTCONDITIONS: The player should move one room to the East from the starting position, which should be an existing room.

IDENTIFIER: NonExistentRoom-Movement-Test1  
DESCRIPTION: This test verifies that whenever a player moves into a nonexistent room, the game will tell them that the room does not exist with a message of “There's a wall there, buddy!”. This test will have the player move one room to the West from the starting position, which should be a nonexistent room.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar”.

EXECUTION STEPS:Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “W”.

POSTCONDITIONS: The player should remain in the starting position. The console should display “There's a wall there, buddy!”, indicating that they have attempted to move into a nonexistent room.

IDENTIFIER: NonExistentRoom-Movement-Test2  
DESCRIPTION: This test verifies that whenever a player moves into a nonexistent room, the game will tell them that the room does not exist with a message of “There's a wall there, buddy!”. This test will have the player move six times to the East from the starting position, which should be an nonexistent room. This test checks an edge case in confirming that all edges of the matrix will return the proper nonexistent room message.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 67891”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E E E E”.

POSTCONDITIONS: The player should be in the sixth column of the matrix, still in the top row.  The console should display “There's a wall there, buddy!”, indicating that they have attempted to move into a nonexistent room.

IDENTIFIER: ValidSeed-Input-Test1  
DESCRIPTION: This test verifies that the game should accept a positive seed value that is a 32-bit signed integer. This test will have the player enter a positive valid seed while running the game and confirm that the seed was accepted.

PRECONDITIONS: User should have the JAR file downloaded.

EXECUTION STEPS: Game should be started by executing “java -jar profwumpus.jar 678908”.

POSTCONDITIONS: The player should see a 6 by 6 grid with an “S” marking their position and game should proceed. On the second line of the output, the console should display “Playing with seed 678908”.

IDENTIFIER: ValidSeed-Input-Test2  
DESCRIPTION: This test verifies that the game should accept a negative seed that is a 32-bit signed integer. This test will have the player enter a negative valid seed while running the game and confirm that the seed was accepted.  
PRECONDITIONS: User should have the JAR file downloaded.

EXECUTION STEPS: Game should be started by executing “java -jar profwumpus.jar -53720462”.

POSTCONDITIONS: Player should see 6 by 6 grid with an “S” marking their position and game should proceed. On the second line of the output, the console should display “Playing with seed -53720462”.

IDENTIFIER: InvalidSeed-Input-Test1  
DESCRIPTION: This test verifies that the game should accept a seed that is not 32-bit signed integer and proceed as if no seed was provided. This test will have the player enter an invalid seed that is exactly one value above the accepted values, and confirm that the game ignores the invalid seed and continues as if no seed was entered. This test is testing an edge case in seeing if only values in the 32-bit signed integer range are considered valid.  
PRECONDITIONS: User should have the JAR file downloaded.

EXECUTION STEPS: Game should be started by executing “java -jar profwumpus.jar 2147483648”.

POSTCONDITIONS: Player should see 6 by 6 grid with an “S” marking their position and game should proceed as normal.

IDENTIFIER: InvalidSeed-Input-Test2  
DESCRIPTION: This test verifies that the game should accept a seed that is not 32-bit signed integer and proceed as if no seed was provided. This test will have the player enter an invalid alphanumeric seed and confirm that the game ignores the invalid seed and continues as if no seed was entered. This test is testing an edge case in seeing if the code parsing the seed is only planning for numeric values or is prepared for any input.  
PRECONDITIONS: User should have the JAR file downloaded.

EXECUTION STEPS: Game should be started by executing “java -jar profwumpus.jar 263knf2e2”.

POSTCONDITIONS: Player should see 6 by 6 grid with an “S” marking their position and game should proceed as normal.

IDENTIFIER: TAMovement-Random-Test  
DESCRIPTION: This test verifies that the TA moves to a random room every iteration that the game progresses. This test will have the player play two games, each with a different seed, going to the same spot in both games, and finding the TA in one game while not finding the TA in the other, confirming that the TA movement is randomized between seeds.  
PRECONDITIONS: User should have the JAR file downloaded.

EXECUTION STEPS: First game should be started by executing “java -jar profwumpus.jar 455019788”. Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E E”. Quit the game by pressing “Control + C” in combination. Another game should be started by executing “java -jar profwumpus.jar 69”. Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E E”.  
POSTCONDITIONS: After the fourth key press in the first game, you will have encountered the TA in the room belonging to the fifth column and first row of the matrix. After the fourth key press in the second game, you will not have encountered the TA in the room belonging to the fifth column and first row of the matrix.

IDENTIFIER: NonExistentRoom-TAMovement-Test  
DESCRIPTION: This test verifies that the user receives a message indicating that the TA has run into a wall if the TA attempts to move to a nonexistent room.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E”.

POSTCONDITIONS:  The console should display “You hear a thud, as if the TA hit into an Eastern wall…”, indicating that the TA attempted to move.

IDENTIFIER: Professor-NoMovement-Test  
DESCRIPTION: This test verifies the professor does move in any situation. This test will have the play find the professor’s location, then find the TA and get teleported away, and return to the previous location of the professor to confirm that he hasn’t moved.

PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “S S S E E N N E S E”.  
POSTCONDITIONS: The first 5 inputs should locate the professor in the matrix. The next four should find the TA and result in you being randomly teleported away. The final input should relocate the professor.

IDENTIFIER: StudentProfessor-HasHomework-Test  
DESCRIPTION: This test verifies the student winning the game by possessing the assignment and encountering Professor Wumpus and thereby turning it in. This test will have the player find the assignment and then turn it into the professor, confirming that they win the game.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E S S S”.  
POSTCONDITIONS: The console should display “You turn in the assignment. YOU WIN!”, and the program should end.

IDENTIFIER: StudentProfessor-DoesNotHaveHomework-Test  
DESCRIPTION: This test verifies the student losing the game by encountering Professor Wumpus without possessing the assignment. This test will have the player run into the professor before finding the assignment, confirming that they lose the game.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E E W”.  
POSTCONDITIONS: The console should display “Prof Wumpus sees you, but you don't have your assignment. YOU LOSE!”, and the program should end.

IDENTIFIER: StudentProfessor-LocatedDirectly-Test1  
DESCRIPTION: This test verifies that the player is notified about Professor Wumpus pontificating when they are one room in a cardinal direction away from the professor. This test will have the player confirm that the console displays “You hear someone pontificating on Computer Science... Professor Wumpus must be nearby!” when approaching the professor from the East.  
PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E E”.  
POSTCONDITIONS: The console should display “You hear someone pontificating on Computer Science... Professor Wumpus must be nearby!”.

IDENTIFIER: StudentProfessor-LocatedDirectly-Test2  
DESCRIPTION: This test verifies that the player is notified about Professor Wumpus pontificating when they are one room in a cardinal direction away from the professor. This test will have the player confirm that the console displays “You hear someone pontificating on Computer Science... Professor Wumpus must be nearby!” when approaching the professor from the North.

PRECONDITIONS: Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS: Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E S S”.  
POSTCONDITIONS: The console should display “You hear someone pontificating on Computer Science... Professor Wumpus must be nearby!”.

IDENTIFIER: StudentTA-LocatedDirectly-Test1  
DESCRIPTION: This test verifies that the player is notified about rustling papers when they are one room in a cardinal direction away from the TA. This test will have the player confirm that the console displays “You hear the shuffling of graded papers... the TA must be nearby!” when approaching the TA from the West.  
PRECONDITIONS:  Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS:  Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E”.  
POSTCONDITIONS: The console should display “You hear the shuffling of graded papers... the TA must be nearby!”.

IDENTIFIER: StudentTA-LocatedDirectly-Test2

DESCRIPTION: This test verifies that the player is notified about rustling papers when they are one room in a cardinal direction away from the TA. This test will have the player confirm that the console displays “You hear the shuffling of graded papers... the TA must be nearby!” when approaching the TA from the South.  
PRECONDITIONS:  Game should be started by executing “java -jar profwumpus.jar 455019788”.

EXECUTION STEPS:  Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E E N”.  
POSTCONDITIONS: The console should display “You hear the shuffling of graded papers... the TA must be nearby!”.

**Traceability Matrix**

Requirement 1: Matrix-Size-Test, Student-Location-Test

Requirement 2: Invalid-Movement-Test, Valid-Movement-Test

Requirement 3: CaseInsensitiveLower-Input-Test, CaseInsensitiveUpper-Input-Test

Requirement 4: ExistingRoom-Movement-Test

Requirement 5: NonExistentRoom-Movement-Test1, NonExistentRoom-Movement-Test2

Requirement 6: ValidSeed-Input-Test1, ValidSeed-Input-Test2

Requirement 7: InvalidSeed-Input-Test1, InvalidSeed-Input-Test2

Requirement 8: TAMovement-Random-Test, NonExistentRoom-TAMovement-Test, Professor-NoMovement-Test

Requirement 9: StudentProfessor-HasHomework-Test, StudentProfessor-DoesNotHaveHomework-Test

Requirement 10: StudentProfessor-LocatedDirectly-Test1, StudentProfessor-LocatedDirectly-Test2

Requirement 11: StudentTA-LocatedDirectly-Test1, StudentTA-LocatedDirectly-Test2

**Defects**

SUMMARY: The game is only a 5 by 5 matrix instead of 6 by 6.  
DESCRIPTION: This defect was discovered during test case Matrix-Size-Test. According to requirement 1, the matrix should have 6 rows and 6 columns, but only has 5 rows and 5 columns instead.

REPRODUCTION STEPS: The game should be started by executing the JAR file, with the provided seed of “2032042993”. Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “S S S S S”.  
EXPECTED BEHAVIOR: The player should now be in the first column and the sixth row.  
OBSERVED BEHAVIOR: From the time that the game has started running, only a 5 by 5 matrix is shown. On the fifth key press, when they should arrive in the sixth row, the console instead displays the nonexistent room message, “There's a wall there, buddy!”.

SUMMARY: Attempting to go East from the fifth column of the matrix raises an exception and exits the game.

DESCRIPTION: This defect was discovered in NonExistentRoom-Movement-Test2. According to requirement 5, when a player tries to move into a nonexistent room, the message “There's a wall there, buddy!” should be displayed by the console. However, when the player tries to move East from the fifth column into a nonexistent room, the game instead raises an out of bounds exception and ends.

REPRODUCTION STEPS: The game should be started by executing the JAR file, with the provided seed of “203204299”. Enter the following input in order, with an ‘Enter’ or ‘Return’ press between each letter: “E E E E E”.

EXPECTED BEHAVIOR: Player should receive a message displaying “There’s a wall there, buddy!” when he/she attempts to move to a room that does not exist.  
OBSERVED BEHAVIOR: The game ends unexpectedly due to an ArrayIndexOutOfBoundsException.

SUMMARY: An invalid input seed raises an exception and exits the game.  
DESCRIPTION: This defect was discovered during test case InvalidSeed-Input-Test1.

Requirement 7 states that if an input seed that is not a 32-bit signed integer is entered, the system should ignore it and act as if not seed value was entered by the user. However, the system fails to ignore an invalid seed and instead causes an ungraceful exit of the game.

REPRODUCTION STEPS: The player should attempt to start the game with the command “java -jar profwumpus.jar 2147483648”.  
EXPECTED BEHAVIOR: The system should ignore invalid seed and proceed as if no seed was entered.  
OBSERVED BEHAVIOR: The game ends unexpectedly due to NumberFormatException.