# Professor Wumpus

Jennifer Patterson and Nick Petro

CS 1632 - DELIVERABLE 1: Test Plan and Traceability Matrix

## Intro

As with any software testing, you are bound to find ways in which it fails, but the goals is to have is it fail gracefully from the viewpoint of the user. In testing *Professor Wumpus* we found a number of defects that did not do so. At first we were unsure if we would be able to find enough defects for this deliverable, but we quickly found two and were eventually able to find two more. The biggest difficulty that we anticipated was being able to develop enough test cases as a number of the deliverables had only one real test case without becoming verbose and testing extensively. After taking a first pass at developing test cases, we had close to twenty, but needed about five more. It was at this point that we took a more strategic approach to developing test cases.

Using concepts from class, we developed edge cases for a couple of the requirements, most notably the requirement involving the seed for the board configuration. This specific requirement involves a signed 8-bit integer which means there is a viable range of accepted values giving very specific edge cases. We developed the edge case for both the largest positive and negative values that a signed 8-bit integer can hold to ensure the software fails with the appropriate message to the player. Due to the fact that it is a signed integer, the range of values is from -2147483648 to 2147483647 which is easily calculated with some binary math. Some other notable test cases are involving the viable inputs for movement of the student around the board, the TA moving randomly each move, and that a particular seed puts the professor in the same spot every time.

In developing test cases for those requirements, we had to take into account all of the things that a player might input, and how those might interact with the given requirement, effectively creating exponentially more test cases (or ways the software can fail). But we aimed to test all of the most relevant cases that cover the most ground, especially checking the user inputs to keep those within the acceptable ranges. We considered different test cases for characters outside of the N, S, E, W range, and also for special characters on the keyboard as they sometimes affect programs differently. We also used prior test cases involving seed inputs to ensure the program responded properly before moving on to test the other two requirements mentioned above: the TA moving randomly, and Professor Wumpus being in the same location for a given seed.

As we began testing, we quickly found that the professor requirement was met, but we discovered something interesting about the TA requirement. We found that for a given seed, the TA moved in the same pattern. We were unsure if that really fulfilled the requirement as it was not truly random for the same seed. This led us to realize how important it is to have the requirements written in a way that makes them clear to both the people writing them, those writing the software, and those testing the software. Otherwise there will be inconsistencies in how the software is expected to work for the user, how it works after development, and how it is tested, resulting in software that might not be what it should be.

In the end this deliverable taught us about writing requirements, developing test cases, and ensuring failures are reproducible to be fixed before release.

# Test Cases

1. **IDENTIFIER: TEST_GAME_BOARD_FORMAT**

   **TEST CASE**: The game shall have a 6 by 6 matrix of rooms displayed every iteration.

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start up, observe that the game consists of 36 “rooms” (indicated using square braces [ ]).

   **POSTCONDITIONS**: Observe the following output:

   ```
   [S] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   (N, S, E, W) > *user input here*
   ```

2. **IDENTIFIER: TEST_GAME_BOARD_FORMAT_STUDENT**

   **TEST CASE**: The game shall have the location of the student indicated by an uppercase S within the board matrix.

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start up, observe the student’s location is somewhere within one of the 36 bracket pairs.

   **POSTCONDITIONS**: Observe the following output:

   ```
   [S] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   [ ] [ ] [ ] [ ] [ ] [ ]
   (N, S, E, W) > *user input here*
   ```

3. **IDENTIFIER: TEST_CORRECT_INPUT_LETTERS**

   **TEST CASE**: Enter any of the letters specified by the program instructions in either upper or lower case (N, S, E, W or n, s, e, w) to specify the preferred direction during game play

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start-up, begin by entering a letter E. See that the student moves one square to the right. Enter a letter S and see that the student moves one square down. Enter a letter W and see that the student moves one square to the left. Enter a letter N and see that the student moves one square up.

   **POSTCONDITIONS**: Observe that the program accepts these inputs.

4. **IDENTIFIER: TEST_INCORRECT_INPUT_LETTERS**

   **TEST CASE**: Enter a letter other than the correct letters specified by the program instructions (any letter other than n, s, e, w) to specify preferred direction during game play

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start-up, begin by entering a letter A. Repeat the former step for every possible incorrect character value.

   **POSTCONDITIONS**: Observe that the program correctly displays "Please enter N, S, E, or W" to the program user.

5. **IDENTIFIER: TEST_INCORRECT_INPUT_SPECIAL_CHARACTERS**

   **TEST CASE**: Enter a keyboard character other than the correct letters specified by the program instructions (any letter other than n, s, e, w) to specify preferred direction during game play

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start up, begin by entering a letter A. Repeat the former step for every possible incorrect character value.

   **POSTCONDITIONS**: Observe that the program correctly displays "Please enter N, S, E, or W" to the program user.

6. **IDENTIFIER: TEST_UPPER_CASE**

   **TEST CASE**: Enter a uppercase letter (N, S, E, W) to specify preferred direction during game play

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start up, begin by entering a uppercase S. Repeat the former step for every possible directional case (N, E, W).

   **POSTCONDITIONS**: Observe that the student is moved in the proper direction (or the program accounts for running into walls).

7. **IDENTIFIER: TEST_LOWER_CASE**

   **TEST CASE**: Enter a lowercase letter (n, s, e, w) to specify preferred direction during game play

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start up, begin by entering a lowercase s. Repeat the former step for every possible directional case (n, e, w).

   **POSTCONDITIONS**: Observe that the student is moved in the proper direction (or the program accounts for running into walls) and acts as if an uppercase letter was entered.

8. **IDENTIFIER: TEST_MOVE_STUDENT_ROOM_EXISTS**

   **TEST CASE**: Enter a correct letter (N, S, E, W) to specify preferred direction during game play

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**: Upon program start up, begin by entering a uppercase S. Repeat the former step for every possible directional case (N, E, W).

   **POSTCONDITIONS**: Observe that the student is moved in the proper direction (or the program accounts for running into walls), each time a letter is entered, the game board is updated with the new student location, and a new input line is generated.

9. **IDENTIFIER: TEST_MOVE_STUDENT_ROOM_DOESNT_EXIST**

   **TEST CASE**: Attempting to go past a wall in the matrix

   **PRECONDITIONS**: Begin the program execution by running the jar file.

   **EXECUTION STEPS**:

1. Start out in the top left corner of the game board.
2. Choose to go to the left or up by entering W or N respectively.
3. Go south by entering S. Enter W again.

**POSTCONDITIONS**: Observe the following output:

“There’s a wall there, buddy!” Followed by the game board and the program’s prompt for user input.

## 10. IDENTIFIER: TEST_STUDENT_MOVING_ALL_ROOMS

**TEST CASE**: Attempting to enter a new room using the directional inputs (N, S, E, and W)

**PRECONDITIONS**: Begin the program by executing the jar file provided.Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar -1
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Enter E for the next 4 iterations to move the student to the top right corner of the game board.
3. Enter S to move down one square.
4. Enter W on the next 4 iterations to move to the left-most room in the current row of the game board.
5. Enter S to move down one square.
6. Enter E on the next 4 iterations to move to the right-most room in the current row of the game board.
7. Enter S to move down one square.
8. Enter W on the next 3 iterations to move to the left 3 spaces (we don’t want to go over the full way because of Professor Wumpus).
9. Enter S to move down one square.
10. Enter E on the next 2 iterations and observe that you find the assignment.
11. Enter E on the next iteration to move into the bottom right corner of the game board.
12. Enter W on the next 4 iterations to move into the bottom left corner of the game board.
13. Enter N to move north one space and hand in your assignment to Professor Wumpus.

**POSTCONDITIONS**: Observe the following output:

“You turn in your assignment. YOU WIN!”

## 11. IDENTIFIER: TEST_GAME_START_WITHOUT_SEED

**TEST CASE**: If an invalid value is entered for the seed, the system shall ignore it and assume that no argument was passed in (that is, it will act as though no seed were entered).

**PRECONDITIONS**: Open the command line and cd to the proper folder that holds the profwumpus.jar file

**EXECUTION STEPS**: Begin the program execution by running the jar file with an integer seed entered as command line input. The input should be formatted as follows:

```
Java -jar profwumpus.jar
```

**POSTCONDITIONS**: Observe the following output:

```
Welcome to Professor Wumpus
Playing with seed *random integer*
[S] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
(N, S, E, W) > *user input here*
```

12. **IDENTIFIER: TEST_GAME_START_WITH_SEED_EDGE**

**TEST CASE**: The game shall accept a 32-bit signed integer seed for the random number generator. This should be entered as an argument for the program on the command line.

**PRECONDITIONS**: Open the command line and cd to the proper folder that holds the profwumpus.jar file

**EXECUTION STEPS**: Begin the program execution by running the jar file with an integer seed entered as command line input. The input should be formatted as follows:

```
Java -jar profwumpus.jar 2147483647 / -2147483648
```

**POSTCONDITIONS**: Observe the following output:

```
Welcome to Professor Wumpus
Playing with seed 2147483647
[S] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
(N, S, E, W) > *user input here*
```

13. **IDENTIFIER: TEST_GAME_START_WITH_SEED_ZERO**

**TEST CASE**: The game shall accept a 32-bit signed integer seed for the random number generator. This should be entered as an argument for the program on the command line.

**PRECONDITIONS**: Open the command line and cd to the proper folder that holds the profwumpus.jar file

**EXECUTION STEPS**: Begin the program execution by running the jar file with an integer seed entered as command line input. The input should be formatted as follows:

```
Java -jar profwumpus.jar 0
```

**POSTCONDITIONS**: Observe the following output:

```
Welcome to Professor Wumpus
Playing with seed 0
[S] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
(N, S, E, W) > *user input here*
```

14. **IDENTIFIER: TEST_GAME_START_WITH_INCORRECT_SEED**

**TEST CASE**: If an invalid value is entered for the seed, the system shall ignore it and assume that no argument was passed in (that is, it will act as though no seed were entered).

**PRECONDITIONS**: Open the command line and cd to the proper folder that holds the profwumpus.jar file

**EXECUTION STEPS**: Begin the program execution by running the jar file with a floating point/decimal number, string or special character value entered as command line input. The input should be formatted as follows:

```
Java -jar profwumpus.jar ponies
```

**POSTCONDITIONS**: Observe the following output:

```
Welcome to Professor Wumpus
Playing with seed *random integer*
[S] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
```

```
(N, S, E, W) > *user input here*
```

## 15. IDENTIFIER: TEST_TA_MOVES_RANDOMLY

**TEST CASE**: The TA shall move a random direction at each iteration.

**PRECONDITIONS**: Begin the program by executing the jar file provided.Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar 350
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Enter S on the next 4 iterations to move the student south 4 squares (he should be in the bottom left corner of the game board).
3. Enter an E and observe the following output: "You hear a thud, as if the TA hit a Northern wall..."
4. Enter N on the next 2 iterations to move the student north 2 squares, and observe the following output: "You here the shuffling of graded papers...the TA must be nearby!"
5. Enter E on the next 4 iterations to move the student east 4 squares (he/she should be in the bottom right corner of the game board).
6. Enter a W to move the TA one square back to the left and observe the following output: "You hear a thud, as if the TA hit into a Western wall…"
7. Enter a W on the next 3 iterations to move the student into the bottom left corner of the game board.

**POSTCONDITIONS**: Observe the following output:

"You hear a thud, as if the TA hit into a Western wall…"

Continue moving the student, starting with the northern direction and observe that the TA bumps into various walls around the game board.

## 16. IDENTIFIER: TEST_TA_BUMP_WALLS

**TEST CASE**: If the TA attempts to move to a room which does not exist, the user shall be informed that they hear the TA bump into a wall.

**PRECONDITIONS**: Begin the program by executing the jar file provided.Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar 300
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Enter E two turns in a row to move two squares to the right.

3. Upon reaching the 3rd square to the right, observe the output message:

   “You hear a thud, as if the TA hit into a Northern wall…”

4. This indicates that the TA in against one of the walls and tried to move into a non-existing room

5. Continue entering E to move the student to the right until you reach the right-most room on the top row.

**POSTCONDITIONS**: Observe the following output:

> “You see the TA and flee to a random room!”

Observe the student has been moved to a random location on the game board. .

**17. IDENTIFIER: TEST_PROF_WUMPUS_STAY_STILL**

**TEST CASE**: When Professor Wumpus is planted in a room (based on the seed entered in the command line), he will not move at all during game play.

**PRECONDITIONS**: Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar 2147483647
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. The statement “You hear someone pontificating on Computer Science… Professor Wumpus must be nearby!”should be outputted to the screen immediately.
3. Enter an S to move down a square.
4. Observe the output: “Prof Wumpus sees you but you don’t have your assignment…YOU LOSE!” - (if you had not previously encountered the assignment)
5. Run the program again, using the same seed as before:
   ```
   java -jar profwumpus.jar 2147483647
   ```
   and repeat steps 1 through 4.

**POSTCONDITIONS**: Observe the following output:

> “Prof Wumpus sees you but you don’t have your assignment…YOU LOSE!”

This indicates that Professor Wumpus does not move from his designated spot that is based on the seed entered on program start-up.

**18. IDENTIFIER: TEST_PROF_WUMPUS_W_ASSIGNMENT**

**TEST CASE**: If the Student has found the Assignment and encountered Professor Wumpus, the player shall win.

**PRECONDITIONS**: Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar 300
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Enter the letter S, to move down the game board.
3. When you see the statement "You smell the ink of a printed assignment!" outputted to the screen, begin by moving to the square directly south of the student's current location.
4. Continue moving the student 2 squares east of the student's current location (enter E on the next two turns).
5. Move the student exactly one square south of his or her current location.

**POSTCONDITIONS**: Observe the following output:

"You turn in your assignment. YOU WIN!"

19. **IDENTIFIER: TEST_PROF_WUMPUS_W/O_ASSIGNMENT**

**TEST CASE**: If the Student has encountered Professor Wumpus but has not found the Assignment, the player shall lose. In either case, after the scenario occurs, the program shall end.

**PRECONDITIONS**: Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar 300
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Enter the letter S, to move down the game board a single space.
3. Move the student two squares to the east of his or her current location (enter E on the next two turns).
4. Move the student two squares down from his or her current location (enter S on the next two turns).

**POSTCONDITIONS**: Observe the following output:

"Prof Wumpus sees you, but you don't have your assignment…YOU LOSE!"

20. **IDENTIFIER: TEST_NEARBY_PROF_WUMPUS**

**TEST CASE**: If the Student is in a room directly to the North, South, East or West of Professor Wumpus, the program shall indicate that they hear somebody pontificating on Computer Science.

**PRECONDITIONS**: Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar 300
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Enter the letters N, S, E, and W to move around the game board.
3. When you see the statement "You hear someone pontificating on Computer Science… Professor Wumpus must be nearby!" outputted to the screen, begin by moving to the square directly next to where the student currently is in the same direction you were going before (i.e. if you had requested to go east on the previous turn and there is another open room to the right, go East again).
4. Continue moving in different directions until you encounter Professor Wumpus in one of the rooms that was directly next to the one the student was in when the statement was initially outputted.

**POSTCONDITIONS**: Observe the following output:

"Prof Wumpus sees you but you don't have your assignment…YOU LOSE!" - (if you had not previously encountered the assignment)

OR

"You turn in your assignment. YOU WIN!" - (if you HAD previously found the assignment)

## 21. IDENTIFIER: TEST_NEARBY_TA

**TEST CASE**: If the Student is in a room directly to the North, South, East or West of the TA, the program shall indicate that they hear the rustling of graded papers.

**PRECONDITIONS**: Begin the program by executing the jar file provided.

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Only choose to go to the up by entering S.
3. Enter direction keys (n, s, e, w) to move around the game board.

**POSTCONDITIONS**: Repeat the above procedures until the following message is displayed to the screen:

"You hear the shuffling of papers…the TA must be nearby!"

Once you encounter this message, the TA should be within one room of you. If you see the TA, the program will cause you to "flee in terror" to a random room of the game's choosing.

## 22. TEST_TA_ENCOUNTERED

**TEST CASE**: If the Student encounters the TA, the student shall flee to a random room.

**PRECONDITIONS**: Begin the program by executing the jar file provided.Begin the program by executing the jar file provided with the following command line input:

```
java -jar profwumpus.jar 350
```

**EXECUTION STEPS**:

1. Start in the top left corner of the game board.
2. Enter S on the next 4 iterations to move the student south 4 squares (he should be in the bottom left corner of the game board).
3. Enter an E and observe the following output: “You hear a thud, as if the TA hit a northern wall.”
4. Enter N on the next 2 iterations to move the student north 2 squares, and observe the following output: “You here the shuffling of graded papers...the TA must be nearby!”
5. Enter N again to attempt to move the student north a single square.

**POSTCONDITIONS**: Observe the following output:

```
You see the TA and flee in terror to a random room!
```

And observe the student has been moved to a random location on the board.

# Traceability Matrix

| **Functional Requirements** | **Test Cases** |
|---|---|
| The game shall consist of a 6 by 6 matrix of rooms, which shall be displayed to the player at each iteration, along with the location of the Student (indicated as an S). | TEST_GAME_BOARD_FORMAT, TEST_GAME_BOARD_FORMAT_STUDENT |
| At each iteration, the player shall be able to input the direction they wish the Student to move (N for North, S for South, E for East or W for West). There are no other options. If a player enters any command other than N, S, E, or W, or their lowercase equivalents, the game shall display "Please enter N, S, E, or W". | TEST_CORRECT_INPUT_LETTERS, TEST_INCORRECT_INPUT_LETTERS, TEST_INCORRECT_INPUT_SPECIAL_CHARACTERS |
| All user input shall be case-insensitive. For example, typing either "N" or "n" shall take the Student to the room to the North (if it exists). | TEST_UPPER_CASE, TEST_LOWER_CASE |
| If a room exists in the direction they have indicated, the Student shall move to that room and a new iteration shall commence. | TEST_MOVE_STUDENT_ROOM_EXISTS, TEST_STUDENT_MOVING_ALL_ROOMS |
| If a room does not exist, the game shall indicate to the user that they cannot move in that direction. | **TEST_MOVE_STUDENT_ROOM_DOESNT_EXIST** |
| The game shall accept a 32-bit signed integer seed for the random number generator. This should be entered as an argument for the program on the command line. | TEST_GAME_START_WITHOUT_SEED, TEST_GAME_START_WITH_SEED_EDGE, TEST_GAME_START_WITH_SEED_ZERO |
| If an invalid value is entered for the seed, the system shall ignore it and assume that no argument was passed in (that is, it will act as though no seed were entered). | TEST_GAME_START_WITH_INCORRECT_SEED |
| The TA shall move a random direction at each iteration. If the TA attempts to move to a room which does not exist, the user shall be informed that they hear the TA bump into a wall. Professor Wumpus shall not move. | TEST_TA_MOVES_RANDOMLY, TEST_TA_BUMP_WALLS, TEST_PROF_WUMPUS_STAY_STILL |
| If the Student has found the Assignment and encountered Professor Wumpus, the player shall win. If the Student has encountered Professor Wumpus but has not found the Assignment, the player shall lose. In either case, after the scenario occurs, the program shall end. | TEST_PROF_WUMPUS_W_ASSIGNMENT, TEST_PROF_WUMPUS_W/O_ASSIGNMENT |
| If the Student is in a room directly to the North, South, East or West of Professor Wumpus, the program shall indicate that they hear somebody pontificating on Computer Science. | TEST_NEARBY_PROF_WUMPUS |

| | |
|---|---|
| If the Student is in a room directly to the North, South, East or West of the TA, the program shall indicate that they hear the rustling of graded papers. | TEST_TA_NEARBY, TEST_TA_ENCOUNTERED |

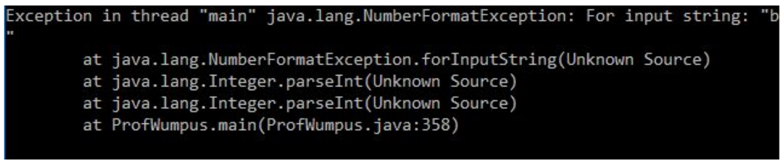
# Defects Found Through Testing

1) **SUMMARY**: When hitting the right-most wall, program crashes with an ArrayIndexOutOfBoundsException
**DESCRIPTION**: According to the requirements, if a room does not exist, the game shall indicate to the user that they cannot move in that direction. With other cases in west, south and north directions, the game outputs the message “There’s a wall there, buddy!” If the user tries to move in a direction where no room exists. For any of the east walls in any row, an exception occurs when trying to test if the same message is outputted when moving too far east. The test case used to uncover this defect was the TEST_EAST_WALLS test case.
**REPRODUCTION STEPS**: Start the program by running the jar file (via the command line, java -jar profwumpus). When the game is launched, the student will begin in the top left corner of the game board. To make the student go to the right, enter an E or e. Continue doing so until the student reaches the right most top corner. Enter another E to attempt to go past the wall.
**EXPECTED BEHAVIOR**: Program shall display message to the user indicating that there is a wall preventing them from going in the direction specified.
**OBSERVED BEHAVIOR**: Program crashes and generates an ArrayIndexOutOfBoundsException

2) **SUMMARY**: The program does not account for user input error for the seed upon program start up.
**DESCRIPTION**: According to the requirements, the game shall accept a 32-bit signed integer seed for the random number generator and ignore invalid input. This should be entered as an argument for the program on the command line. If the user inputs an incorrect value (i.e. a character or string), the program outputs an exception. The test case used to uncover this defect was the TEST_GAME_START_WITH_INCORRECT_SEED test case
**REPRODUCTION STEPS**: Enter the following line into the command line:

```
java -jar profwumpus.jar b
```

Hit enter to run the program.
**EXPECTED BEHAVIOR**: The program shall ignore the invalid input and run with a randomly generated seed value.
**OBSERVED BEHAVIOR**: The program produces the following exception:



```
Exception in thread "main" java.lang.NumberFormatException: For input string: "b
"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at ProfWumpus.main(ProfWumpus.java:358)
```

3) **SUMMARY**: The game board generated by the program is 5-by-5 instead of 6-by-6
**DESCRIPTION**: According to the requirements, the game shall consist of a 6 by 6 matrix of rooms, which shall be displayed to the player at each iteration, along with the location of the Student (indicated as an S). When the program is run, a 5-by-5 matrix of squares is generated instead of the expected 6-by-6 matrix.
**REPRODUCTION STEPS**: Start up the program by running the jar file provided via the command line. Input the following command after entering the correct folder where the jar file is located:

```
java -jar profwumpus.jar 100
```

You can also run the program without entering a seed and it will return the same output:

```
java -jar profwumpus.jar
```

**EXPECTED BEHAVIOR**: Game board is a 6-by-6 matrix of rooms
**OBSERVED BEHAVIOR**: Game board is a 5-by-5 matrix of rooms

4) **SUMMARY**: TA does not move "randomly" at each iteration of the game.
**DESCRIPTION**: According to the requirements, the TA shall move a random direction at each iteration. When the same seed is entered for two separate runs of the program, the TA's movements are the same and not "randomly" select at runtime.
**REPRODUCTION STEPS**:

1. Begin the program by executing the jar file provided with the following command line input: `java -jar profwumpus.jar 350`
2. Start in the top left corner of the game board.
3. Enter S on the first iteration of game play and observe the following output: "You hear a thud, as if the TA hit into an Eastern wall…"
4. Enter S on the next 3 iterations to move the student south 4 squares (he should be in the bottom left corner of the game board).
5. Enter an E on the next 2 iterations and observe the following output: "You hear a thud, as if the TA hit a Northern wall."
6. Enter E on the next 2 iterations to move the student north 2 squares, and observe the following output: "You smell the ink of a printed assignment!"
6. Enter W on the next two iterations to move the student two squares to the left, and observe the following output: "You hear a thud, as if the TA hit into a Western wall…"
7. Enter W on the next two iterations to move the student into the bottom left corner.
8. Enter N on the next iteration and observe the following output: "You hear a thud, as if the TA hit into a Western wall…"
9. Enter N on the next 3 iterations to move the TA to the top left corner.
10. Quit the program and repeat steps 1 through 9.

**EXPECTED BEHAVIOR**: The TA's movements will be randomized on each run of the program, even if the same seed is used.
**OBSERVED BEHAVIOR**: The TA's movements can be predicted when running the program with the same seed and going through the same execution steps on successive runs.