SE 3XA3: Test Report
Lines Per Minute (lpm)

Team #16, Lines Per Minute (lpm)
Jay Mody - modyj - 400195508
Jessica Lim - limj31 - 400173669
Maanav Dalal - dalalm1 - 400178115

March 30, 2021

Table 1: Revision History

Date	**Developer(s)**	**Change**
March 27, 2021	Jay/Jessica/Maanav	Initial document write-up.
March 30, 2021	Jay/Jessica/Maanav	Finish test report

Contents

List of Tables

1 Introduction

This document specifies the results of the testing performed on the lpm package. The document compares the present implementation of lpm to the existing implementation, wpm. It also identifies any changes made to the module as a result of testing, and comments on the testing approaches used. Furthermore, multiple traceability matricies mapping functional requirements to both test cases as well as modules also provided.

2 Requirements Evaluation

2.1 Functional Requirements Evaluation

This section specifies every functional requirement test specified in the Test Plan in a one-to-one mapping. Outputs and test results are documented for every test case. The type of test, initial sate and input and expected output are also included. Further details regarding the individual tests can be found in the Test Plan.

2.1.1 Command Line Interface

Start Typing Interface

1. **test-CLI1**: Start Typing Interface
   Type: Functional, Manual
   Initial State: Newly started terminal session.
   Input: lpm
   Expected output: The lpm typing interface, all languages should be part of the list of available code snippets.
   Actual Output: The lpm typing interface, all languages should be part of the list of available code snippets.
   Result: Passed

2. **test-CLI2:** Start the Typing Interface with Specific Languages
   Type: Functional, Manual
   Initial State: Newly started terminal session.
   Input: lpm python, lpm python java javascript
   Expected output: The lpm typing interface with only python code snippets, the lpm typing interface with python, java and javascript code snippets
   Actual Output: The lpm typing interface with only python code snippets, the lpm typing interface with python, java and javascript code snippets
   Result: Passed

Show Help Menu

1. **test-CLI3**: Show Help Menu
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: lpm –help

Expected output: The text-based help menu.
Actual Output: The text-based help menu.
Result: Passed

Show Typing Statistics

1. **test-CLI4**: Show Statistics
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: A. lpm –stats (with existing history) B. lpm –stats (with no history)
   Expected output: A. The user's lifetime typing statistics B. Display that there is no history
   Actual Output: A. The user's lifetime typing statistics B. Display that there is no history
   Result: Passed

Change Settings

1. **test-CLI5**: Change Settings
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: lpm –settings
   Expected output: Open a vim session with lpmconfig.json file in terminal
   Actual Output: Open a vim session with lpmconfig.json file in terminal
   Result: Passed

2.1.2 Typing Editor

Code Snippet Navigation

1. **test-TE1**: Display Randomized Order Snippet
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: lpm, lpm
   Expected output: Typing interface with random code snippet, Typing interface with random code snippet
   Actual Output: Typing interface with random code snippet, Typing interface with random code snippet
   Result: Passed

2. **test-TE2**: Next/Prev
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: lpm, then the following combination of arrow keys is pressed: →, ←, ←, →
   Expected output: The initial code snippet, the next code snippet, the initial code snippet, the previous code snippet, the initial code snippet
   Actual Output: The initial code snippet, the next code snippet, the initial code snippet, the previous code snippet, the initial code snippet
   Result: Passed

3. **test-TE3**: Start
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: lpm, Type character that is not escape or the arrowkeys
   Expected output: Typing interface with started timer
   Actual Output: Typing interface with started timer
   Result: Passed

4. **test-TE4**: Exit
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: lpm
   Expected output: Terminal screen exited from lpm
   Actual Output: Terminal screen exited from lpm
   Result: Passed

5. **test-TE8**: Stop
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: Escape key
   Expected output: Typing interface with timer ended and cursor at the beginning
   Actual Output: Typing interface with timer ended and cursor at the beginning
   Result: Passed

Editor Information

1. **test-TE5**: Typing Interface Link
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed
   Input: lpm, then the top link is clicked.
   Expected output: Typing interface, as well as a link to the hyperlink opened up in the user's browser
   Actual Output: Typing interface, as well as a link to the hyperlink opened up in the user's browser
   Result: Passed

2. **test-TE6**: Keystroke Information
   Type: Functional, Manual
   Initial State: lpm code snippet in session
   Input: A. Key matching next character on screen, B. Key not matching next character on screen
   Expected output: A. Character in CORRECT_COLOR B. Character in INCORRECT_COLOR, All other characters in TEXT_COLOR
   Actual Output: A. Character in CORRECT_COLOR B. Character in INCORRECT_COLOR, All other characters in TEXT_COLOR
   Result: Passed

3. **test-TE7**: Typing Interface Statistics
   Type: Functional, Manual
   Initial State: Terminal session with lpm installed

Input: lpm, then the user presses a character
Expected output: Typing interface, with stats information showing
Actual Output: Typing interface
Result: Passed

4. **test-TE9**: Keystroke Information Correction
Type: Functional, Manual
Initial State: lpm session with previously typed characters
Input: Backspace key
Expected output: Character returned to TEXT_COLOR and cursor moved back one spot
Actual Output: Character returned to TEXT_COLOR and cursor moved back one spot
Result: Passed

2.1.3 Code Snippets

Code snippet lengths of valid character and line length

1. **test-CS1**: Snippet Line Length Test
Type: Functional, Automated
Initial State: Snippets object is fully loaded with all lpm code snippets from pickle
Input: Every single Snippet stored within the Snippets object
Expected Output: Pass or assertion error if any snippet length is larger than MAX_LENGTH
Actual Output: No assertion error raised. Asserts that every snippet length is smaller than MAX_LENGTH.
Result: Passed

2. **test-CS2:** Snippet Character Length Test
Type: Functional, Automated
Initial State: Snippets object is fully loaded with all lpm code snippets from pickle
Input: Every single Snippet stored within the Snippets object
Expected Output: Pass or assertion error if any snippet length is larger than MAX_COLS
Actual Output: No assertion error raised. Assertions pass for every individual snippet
Result: Passed

Sufficient Code snippets available in Python, Java and Javascript

1. **test-CS3**: Snippets Per Language
Type: Functional, Automated
Initial State: Snippets object is fully loaded with all lpm code snippets from pickle
Input: Every single Snippet stored within the Snippets object
Expected Output: Pass or assertion error if any language has less than MIN_SNIPPETS snippets
Actual Output: No assertion error raised. Assert all three languages have more than MIX_SNIPPETS snippets
Result: Passed

2.1.4 Statistics

Track statistics per individual code-snippet

1. **test-S1**: Test lines per minute speed
   Type: Functional, Dynamic, Automated
   Initial State: lpm environment properly setup, Empty stats History
   Input: Type one line in 10 seconds, Type 2 lines in 10 seconds, Type zero lines in 10 seconds
   Expected Output: 6 lines/min, 12 lines/min, 0 lines/min
   Actual Output: For given input, pytest tests assert: 6 lines/min, 12 lines/min, 0 lines/min
   Result: Passed

2. **test-S2**: Test characters per minute speed
   Type: Functional, Dynamic, Automated
   Initial State: lpm environment properly setup, Empty stats History
   Input: Type 20 characters 5 seconds, Type zero characters in 10 seconds
   Expected Output: 240 char/min, 0 char/min
   Actual Output: For given input, pytest tests assert: 240 char/min, 0 char/min
   Result: Passed

3. **test-S3**: Test words per minute speed
   Type: Functional, Dynamic, Automated
   Initial State: lpm environment properly setup, Empty stats History
   Input: Type 10 words in 10 seconds, Type zero lines in 10 seconds
   Expected Output: 60 words/min, 0 words/min
   Actual Output: For given input, pytest tests assert: 60 words/min, 0 words/min
   Result: Passed

4. **test-S4**: Test errors rate
   Type: Functional, Dynamic, Automated
   Initial State: lpm environment properly setup, Empty stats History
   Input: Match code statement perfectly, Make one char mistake when typing a 100 char line, Make a mistake on every character
   Expected Output: 100%, 99%, 0%
   Actual Output: For given input, pytest tests assert: 100%, 99%, 0%
   Result: Passed

Track statistics per program lifetime

1. **test-S5: Test lines per minute lifetime speed**
   Type: Functional, Dynamic, Automated
   Initial State: lpm environment properly setup, clean slate with newly installed program
   Input: Type 3 lines in 10 seconds, close session, type 4 lines in 10 seconds
   Expected Output: 21 lines per minute
   Actual Output: For given input across multiple sessions, pytest tests assert: 12 lines per minute
   Result: Passed

2.2 Nonfunctional Requirements Evaluation

This section specifies every non-functional requirement test specified in the Test Plan in a one-to-one mapping. Outputs and test results are documented for every test case. The type of test, initial sate and input and expected output are also included. Further details regarding the individual tests can be found in the Test Plan.

2.2.1 Look and Feel

The typing interface shall respond to a user input within 10ms.

1. **test-LF1: User input latency $\leq$ 10ms testing**
   Type: Manual
   Initial State: lpm is opened and a code snippet is loaded.
   Input: User will type out the code snippet
   Expected output: The game should perform all functions properly with a max of 10ms latency.
   Actual Output: No noticeable latency.
   Result: Passed

The user interface should be visible in both light and dark terminal backgrounds.

1. **test-LF2: Light theme test**
   Type: Manual
   Initial State: Command line is open
   Input: The user launches lpm in light mode
   Expected output: The lpm package opens in light mode
   Actual Output: The lpm package launched in light mode.
   Result: Passed

2. **test-LF3: Dark theme test**
   Type: Manual
   Initial State: Command line is open
   Input: The user launches lpm in dark mode
   Expected output: The lpm package opens in dark mode
   Actual Output: The lpm package launched in dark mode.
   Result: Passed

The provided theme shall be easy on the eyes and follow the WCAG AA or AAA specification in terms of colour choice.

1. **test-LF4: WCAG AA testing**
   Type: Manual
   Initial State: lpm is opened and a few characters are typed, such that all potential colours are displayed (i.e. some correct characters as well as some incorrect characters.)
   Input: The hex value of all the colours for each theme of the lpm package
   Expected output: A WCAG number above 4.5

Actual Output: A WCAG number of 4.7 was achieved.
Result: Passed

The code snippets chosen should be diverse, and representative of the languages' syntax

1. **test-LF5: Code snippet diversity testing**
   Type: Manual
   Initial State: lpm is loaded
   Input: The user will use skipping functions to view multiple code snippets.
   Expected output: The user will have seen various code snippets and be able to comment on their diversity / uniqueness.
   Actual Output: The user commented that there was a variety of logic, functions, imports, lengths, and in general code snippets across languages were unique.
   Result: Passed

The entirety of the user interface should fit within a terminal window sized MAX_COLS (width) x MAX_LINES (length) or greater

1. **test-LF6: lpm resoultion testing**
   Type: Manual
   Initial State: lpm is loaded in a terminal of size MAX_COLS (width) x MAX_LINES
   Input: The user will type a code snippet
   Expected output: The user will have seen the entirety of the lpm screen, including their code snippet and statistics. Through this process, they will be able to inform us if the package was usable in a window with a resolution of MAX_COLS (width) x MAX_LINES
   Actual Output: The user commented that the package was usable at resolutions above MAX_COLS X MAX_LINES.
   Result: Passed

2.2.2 Usability and Humanity

The system's typing interface, as well as its cursor indicator, should be intuitive.
The system shall be easy to use for anyone with basic knowledge of the console.
The instructions will be easily comprehensible by anyone with basic understanding of English.

1. **test-UH1: General use testing**
   Type: Manual
   Initial State: Command line is open
   Input: User will open lpm and interact with the software
   Expected output: The user's overall impressions on the usability, typing interface, and comprehensibility of the lpm package
   Actual Output: The user commented that the package was overall straightforward and worked as expected. They further commented that instructions were

easily comprehended and the package was intuitive to use.
Result: Passed

The user will only require the keys on a typical 60% keyboard to correctly type all given code.

1. **test-UH2: 60% keyboard testing**
   Type: Manual
   Initial State: lpm is opened and a code snippet is loaded. The user is using a 60% keyboard
   Input: The user types through two code snippets.
   Expected output: The user is either successful or unsuccessful in typing the required code snippets
   Actual Output: The user ran into no issues while typing through multiple snippets using lpm, as their 60% keyboard had all required keys to type snippets contained within lpm.
   Result: Passed

2.2.3 Performance

When the user loads the package, excluding the first time loading, the time it takes for the package to be ready to accept user input shall not exceed 1 second.

1. **test-PF1**
   Type: Manual
   Initial State: Command line is open, lpm package previous loaded
   Input: lpm command is entered
   Expected output: the lpm package is interactive
   Actual Output: the lpm package is interactive
   Result: Passed

Application should be available 99.999% (5 nines) of the time. This translates to 5.26 minutes of downtime in a given year, afforded by user updates of python or lpm, as well as unforeseen circumstances in the CI/CD Pipeline.

1. **test-PF2**
   Type: Manual
   Initial State: PyPI page opened
   Input: python package name, 'lpm'
   Expected output: Uptime based on pypi statistics and versioning
   Actual Output: lpm was always available to the user unless they were updating the package.
   Result: Passed

2.2.4 Operational and Environmental

The system shall work on Python 3.6 and up

1. **test-OE1**
   Type: Manual
   Initial State: Python 3.6 or above, lpm repository installed
   Input: run lpm
   Expected output: Package is successfully run on multiple versions of Python.
   Actual Output: Package was tested on Python 3.6.0, 3.7.5, and 3.8.1 and no errors or differences were recorded between versions.
   Result: Passed

The system shall work on Linux, macOS, and Windows operating systems.

1. **test-OE2**
   Type: Manual
   Initial State: lpm repository installed
   Input: run lpm, run CI/CD test suite
   Expected output: lpm can be installed and run on all 3 platforms.
   Actual Output: lpm was installed and successfully run on Ubuntu 18.04, Windows 10, and MacOS 11.
   Result: Passed

In terms of computer specs, the package shall run on any computer that is able to run Python 3 based on their respective minimum requirements (i.e. if running on Python 3, the user's computer shall at least have Python 3's minimum requirements to be supported officially by the lpm package).

1. **test-OE3**
   Type: Manual
   Initial State: A third party who uses a laptop that is 10 years old does not currently have lpm installed opens their terminal.
   Input: The third party types pip install lpm, runs lpm, then types out one code snippet
   Expected output: Qualitative information on the performance of lpm, and information about whether it matched their standards.
   Actual Output: lpm was run and performed very similarly to on a newer device - latency was low, and performance did not suffer.
   Result: Passed

Application should be installable via the pip package manager.

1. **test-OE4**
   Type: Manual
   Initial State: Environment without lpm package installed
   Input: pip install –upgrade lpm
   Expected output: lpm package successfully updated.
   Actual Output: lpm package successfully updated.
   Result: Passed

2.2.5 Maintainability and Support

Application should be easy to update (via pip)

1. **test-MS1**
   Type: Manual
   Initial State: Environment with non-current version of lpm installed
   Input: Changes are committed, and a pull request is made for an issue.
   Expected output: Pull request is accepted, and changes are merged into the master branch. The lpm package now has the changes
   Actual Output: Pull request is accepted, and changes are merged into the master branch. The lpm package now has the changes
   Result: Passed

It should be easy to add additional code snippets.

1. **test-MS2**
   Type: Manual
   Initial State: lpm installed
   Input: New code snippet can be added with a permalink. A pull request is made requesting addition of the snippet.
   Expected output: The code snippet is added to the lpm repository and shipped to users in the next release of lpm. Users will get new snippets when they reset snippets
   Actual Output: The code snippet is added to the lpm repository and shipped to users in the next release of lpm. Users will get new snippets when they reset snippets
   Result: Passed

2.2.6 Security

External systems shall not have access to the system.

1. **test-SC1**
   Type: Manual
   Initial State: lpm is installed on the system
   Input: A third party tries to access the system without access to it.
   Expected output: The party is unable to access it since there are 0 states in which someone external can access
   Actual Output: The party is unable to access it since there are 0 states in which someone external can access
   Result: Passed

2.2.7 Cultural

Code snippets that include harmful, vulgar, controversial, political, or offensive content shall not be displayed in lpm

1. **test-CT1**
   Type: Manual
   Initial State: lpm is installed and running.

Input: A third party will page through all of the quotes in the lpm tool.
Expected output: The third party will inform us if they find any of the snippets harmful, vulgar, controversial, political, or offensive, and if so, corrective measures will be taken in the form of removing said quotes from the document.
Actual Output: No harmful, vulgar, controversial, political, or offensive code snippets were found throughout the entirety of our code snippets database.
Result: Passed

2.2.8 Legal

Code snippets shall have a valid open source license or the developers have given explicit permission to be displayed in lpm

1. **test-LG1**
   Type: Manual
   Initial State: lpm repository
   Input: A code snippet is added to the repository with link to code snippet
   Expected Output: Each code snippet provides a link to a source repository. The link either comes from a repository with an open source license, or lpm has explicit permissions from the developer.
   Actual Output: Each code snippet provides a link to a source repository. The link either comes from a repository with an open source license, or lpm has explicit permissions from the developer.
   Result: Passed

Tests results	
Test ID #	Pass-Fail
test-CLI1	passed
test-CLI2	passed
test-CLI3	passed
test-CLI4	passed
test-CLI5	passed
test-TE1	passed
test-TE2	passed
test-TE3	passed
test-TE4	passed
test-TE5	passed
test-TE6	passed
test-TE7	passed
test-TE8	passed
test-TE9	passed
test-CS1	passed
test-CS2	passed
test-CS3	passed
test-S1	passed
test-S2	passed
test-S3	passed
test-S4	passed
test-S5	passed
test-LF1	passed
test-LF2	passed
test-LF3	passed
test-LF4	passed
test-LF5	passed
test-LF6	passed
test-UH1	passed
test-UH2	passed
test-PF1	passed
test-PF2	passed
test-OE1	passed
test-OE2	passed
test-OE3	passed
test-OE4	passed
test-MS1	passed
test-MS2	passed
test-SC1	passed
test-CT1	passed
test-LG1	passed

Table 2: Tests results

3 Comparison to Existing Implementation

Our implementation adds many changes to the existing implementation (wpm). For instance, the way that modules interacted with eachother - namely between the Game, Screen, and Stats modules. These changes helped to improve the property of maintainability, as we followed the separation of concerns principle in a better fashion than the initial implementation did. This change also improved the ease of testing, as in the initial implementation, the Game module and the Stats module both calculated user typing statistics, which would have required us to validate both calculations in our testing. In contrast, we ensured that only the Stats module was doing any calculations, and the Game Module acted solely as a controller. These changes also resulted in improvements to the readability of the code for future changes. Other things that have changed in contrast with the existing implementation consist of: Implementing the ability to interpret code snippets, an 'LPM' metric, as well as the ability to choose between different types of snippets (in this case, we segmented using language).

In terms of testing, the wpm testing framework (based on our review of the repository) only has test for importing of quotes (the equivalent of code snippets on lpm). In comparison, our test suite is much more comprehensive. Lpm's testing framework covers more modules - including tests for the game, screen and stats modules, which are not tested in wpm. The lpm testing plan also includes manual testing. As a result, it has a code coverage of over 90%, which is a far greater amount of testing than the wpm package's 6 unit tests.

4 Unit Testing

TODO Due to the nature of the Scrabble program, where the bag and rack of the player are random and the types of input are non-deterministic, it was infeasible to conduct unit testing for the majority of the modules. Because the majority of the program was tested through the front-end interface in Tkinter, most testing was exploratory testing where the aim was to cover the majority of situations that would occur in regular Scrabble gameplay (for example, placing a word that created more words). For more information regarding the back-end during execution of the moves during a playthrough, print statements to the terminal were used and evaluated after each test. During development, unit testing was conducted on modules in the Model classes, so Bag.py, Rack.py, Board.py, Player.py and Tiles.py. This was conducted with PyTest and was prior to developing the back-end for the Scrabble game to ensure it was correct before connecting it with the front-end. After the front-end to back-end implementation was complete the majority of testing was manual testing through the front-end.

5 Changes Due to Testing

5.1 Functional Requirements Changes Evaluation

Success and failures traced to Functional Requirements incredibly important when evaluating lpm, and changes were made to the code base when functional requirements.

Initially, the automated test-CS3 was failing due to an insufficient amount of Java Python and Javascript snippets. This test brought attention to the fact that requirements were unsatisfied, and prompted the addition of further snippets.

The importance of differentiating the function of the ESCAPE key when typing a snippet, compared to when browsing through the snippets, became clear when performing tests test-TE4 and test-TE8. Using the expected inputs and outputs of the test, lpm's game logic was altered so that the functionality matched the requirements.

Furthermore, the typeface used for the INCORRRECT_COLOR was changes due to a test failure in test-TE6. Initially, when character spaces were typed incorrectly, this did not register on the screen. To improve upon the requirement that correct, incorrect, and untyped characters must appear uniquely to the user, a change was made to create a red background for incorrectly typed characters, allowing test-TE6 to pass.

5.2 Look and Feel Changes Evaluation

Due to test-LF6, a change was made to the resolution size requirements. The MAX_COLS and MAX_LINES went outside of the allocated terminal dimensions, so after testing, the resolution sizes were altered appropriately to allow for universal scalability.

5.3 Usability and Humanity Changes Evaluation

There were no overall changes made to the code due to tests in the Usability category.

5.4 Performance Changes Evaluation

There were no overall changes made to the code due to tests in the Performance category.

5.5 Operational and Environmental Changes Evaluation

Due to unit testing, it was decided that Python2 compatibility would not be included for the initial version of lpm as the environments resulted in too many complications. No other changes were made to the user interface due to tests in the Operational and Environmental category.

5.6 Maintainability and Support Changes Evaluation

To satisfy test-MS2, changes were made to the code base to simplify the process of adding code snippets. While the original plan was to create JSON objects that would house the properties and links for all open-source code snippets, this proved to be a burden, making it difficult to satisfy test-MS3. To make the process simpler, the data was stored using pickle, which allows users to simply add the url link of a github snippet, making it easier to add additional code snippets.

5.7 Security Changes Evaluation

There were no overall changes made to the code due to tests in the Security category.

5.8 Cultural Changes Evaluation

There were no overall changes made to the code due to tests in the Cultural category.

5.9 Legal Changes Evaluation

There were no overall changes made to the code due to tests in the Legal category.

6 Automated Testing

Automated Testing for the lpm package was completed using the pytest testing framework. Given that this is the industry standard for testing in the Python language, it was the ideal choice for testing our program. Automated testing was performed on a per-module basis for the Snippets, Config, and Stats modules. These modules were perfect candidates for automated testing, as they had very low coupling with the other modules, and could be used independently from the application as a whole. However, modules like Screen and Game had higher coupling and proved difficult to unit test. As such, we utilized manual testing in place of automated testing for many of our tests where it made the most sense. Since these manual tests would be utilizing the system as a whole, the automated testing allowed us to be confident that if an error was revealed during testing, it was not a result of the modules that were tested with pytest.

The following tests were performed in an automated fashion: test-CS1, test-CS2, test-CS3, test-S1, test-S2, test-S3, test-S4, test-S5. In total, our test plan includes 8 automated tests, however, we ended up implementing an additional 4 tests (giving us a total of 12 automated tests). All 12 tests passed. A picture of the test results is attached below:


testResults.png

7 Trace to Requirements

The following section displays updated traceability matrices between both functional and nonfunctional requirements to test cases. These were originally outlined in the SRS.

7.1 Traceability Between Test Cases and Requirements

Functional Requirement Traceability Matrix	
Functional Requirement #	Test ID
FR1	test-CLI1
FR2	test-CLI1
FR3	test-CLI2
FR4	test-CLI3
FR5	test-CLI4
FR6	test-CLI5
FR7	test-TE1, test-CLI2
FR8	test-TE1
FR9	test-TE5
FR11	test-TE7
FR12	test-TE2
FR13	test-TE2
FR14	test-TE3
FR15	test-TE8
FR16	test-TE6
FR17	test-TE6
FR18	test-TE6
FR19	test-TE6, test-TE9
FR20	test-TE4
FR21	test-CS1
FR22	test-CS2
FR23	test-CS3
FR24	test-CS3
FR25	test-S1, test-S5
FR26	test-S3, test-S5
FR27	test-S2, test-S5
FR28	test-S4, test-S5

Table 3: Functional Requirements Traceability Matrix

Non-Functional Requirements Traceability Matrix	
Non-Functional Requirement #	Test ID
NFR1	test-LF1
NFR2	test-LF2, test-LF3
NFR3	test-LF4
NFR4	test-LF5
NFR5	test-LF6
NFR6	test-UH1
NFR7	test-UH1
NFR8	test-UH1
NFR9	test-UH2
NFR10	test-PF1
NFR11	test-PF2
NFR12	test-OE1
NFR13	test-OE2
NFR14	test-OE3
NFR15	test-OE4
NFR16	test-MS1
NFR17	test-MS2
NFR18	test-SC1
NFR19	test-CT1
NFR20	test-LG1

Table 4: Non-Functional Requirements Traceability Matrix

8 Trace to Modules

This section displays the traceability matrix between the test cases and the modules. While all function requirements trace to a module, some NFRs are system wide and do not directly trace to a module.

The modules of our system are as follows:

Module Traceability Matrix		
Module #	FR Test ID	NFR Test ID
M1	test-CLI2, test-CLI3, test-CLI4, test-CLI5	
M2	test-CLI5	
M3	test-TE1, test-TE2, test-TE3, test-TE4, test-TE8, test-TE6, test-TE9	test-LF1, test-UH1, test-UH2
M4	test-TE1, test-TE2, test-TE3, test-TE4, test-TE8, test-TE5, test-TE6, test-TE7, test-TE9	test-LF1, test-LF2, test-LF3, test-LF4, test-LF6
M5	test-CLI2, test-TE1, test-TE2, test-TE3, test-TE4, test-TE8, test-TE5, test-CS1, test-CS2, test-CS3	test-LF5, test-MS2, test-CT1, test-LG1
M6	test-CLI4, test-TE7, test-S1, test-S2, test-S3, test-S4, test-S5	
M7	test-CLI1	test-PF1

Table 5: Module to Test Case Traceability Matrix

9 Code Coverage Metrics

Using coverage.py, we found that our combined code coverage across both manual and automated testing was 92% (see Figure **??**). Automated testing by itself was only 23% (see Figure **??**). This is expected, as only the Config, Snippets, and Stats modules are included in automated testing. During testing, the code coverage reports helped us uncover several areas that lacked testing, for which we went back and added tests fo. However, there were some tests that we didn't end up implementing due to time constraints.

For example, our final code coverage report reveals that a piece of logic in game.py that accounts for snippets with empty lines was never run during testing (see Figure **??**). This was just unlucky, as lpm gives code snippets at random, and the testers happened to have tested snippets that did not contain empty lines. To combat this, we would want to add more test cases which should include a more diverse range of code snippets.

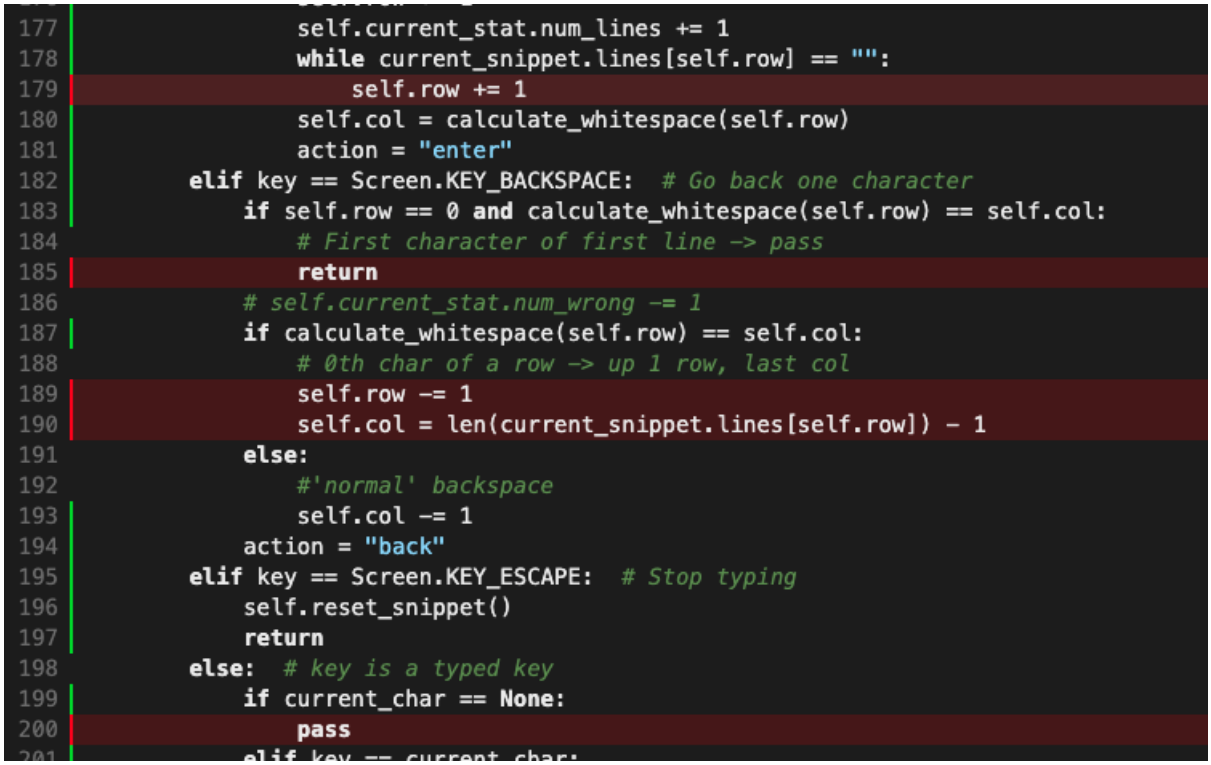
Another example would be in the Screen module, for which we did not have test cases for small terminal sizes (see Figure **??**). To account for this, we would add additional test cases to test-LF6 where the terminal size is insufficient and we expect the program to throw an error informing the user that their terminal is too small.

Name	Stmts	Miss	Cover
lpm/__init__.py	2	0	100%
lpm/__main__.py	3	0	100%
lpm/commandline.py	90	12	87%
lpm/config.py	28	3	89%
lpm/game.py	134	15	89%
lpm/screen.py	146	8	95%
lpm/snippets.py	63	2	97%
lpm/stats.py	73	5	93%
TOTAL	539	45	92%

Figure 1: Coverage for Automated + Manual Testing

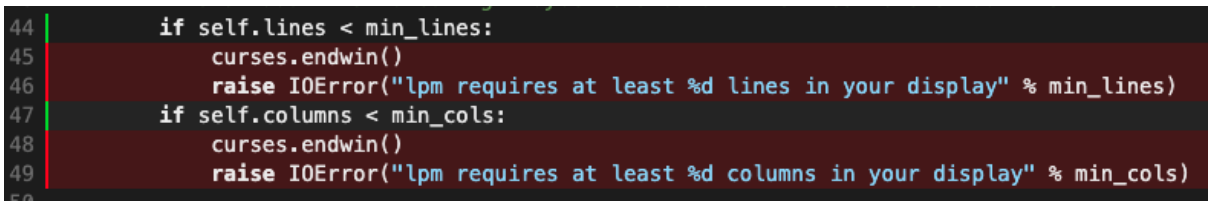
Name	Stmts	Miss	Cover
lpm/__init__.py	2	0	100%
lpm/__main__.py	3	3	0%
lpm/commandline.py	90	90	0%
lpm/config.py	28	1	96%
lpm/game.py	134	134	0%
lpm/screen.py	146	146	0%
lpm/snippets.py	63	33	48%
lpm/stats.py	73	9	88%
TOTAL	539	416	23%

Figure 2: Coverage for Automated Testing



```
177                 self.current_stat.num_lines += 1
178                 while current_snippet.lines[self.row] == "":
179                     self.row += 1
180                 self.col = calculate_whitespace(self.row)
181                 action = "enter"
182         elif key == Screen.KEY_BACKSPACE:  # Go back one character
183             if self.row == 0 and calculate_whitespace(self.row) == self.col:
184                 # First character of first line -> pass
185                 return
186             # self.current_stat.num_wrong -= 1
187             if calculate_whitespace(self.row) == self.col:
188                 # 0th char of a row -> up 1 row, last col
189                 self.row -= 1
190                 self.col = len(current_snippet.lines[self.row]) - 1
191             else:
192                 #'normal' backspace
193                 self.col -= 1
194             action = "back"
195         elif key == Screen.KEY_ESCAPE:  # Stop typing
196             self.reset_snippet()
197             return
198         else:  # key is a typed key
199             if current_char == None:
200                 pass
```

Figure 3: Missed coverage for game.py



```
44         if self.lines < min_lines:
45             curses.endwin()
46             raise IOError("lpm requires at least %d lines in your display" % min_lines)
47         if self.columns < min_cols:
48             curses.endwin()
49             raise IOError("lpm requires at least %d columns in your display" % min_cols)
```

Figure 4: Missed coverage for screen.py