SE 3XA3: Test Report
Tetrileet

Team #15, AAA Solutions
Ali Tabar and sahraeia
Abdallah Taha and tahaa8
Andrew Carvalino and carvalia

April 12, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
April 12 2021	1.0	Added the introduction section as well as section 1 Functional Requirement Evaluation. Also added section 5: Changes due to testing. Added Section 6: Automated Testing.
April 12 2021	1.1	Added Sections 2, 3, and 4 - Evaluation of Nonfunctional Requirements, Comparison to Existing Implementation, and Unit Testing sections respectively.
April 12 2021	1.2	Added Sections 7, 8, and 9 - Trace to Requirements, Trace to Modules, and Code Coverage Metrics respectively.

This document will include a summary of the testing that was done for Tetrileet and the results of that testing. This includes going over the Functional and Non functional Requirement testing evaluation. Furthermore, this document will also cover the performance evaluation of the program, as well as a comparison of our testing to the original implementation, as well as a comparison of both implementations. In addition, we will also cover the changes that were made due to the testing and our reasoning for not using automated testing. Finally, this document will end with two traceability tables that will trace the testing to the requirements, and to the modules, as well as code coverage metrics.

1 Functional Requirements Evaluation

All of the target functional requirements were satisfied in the implementation of Terileet. Furthermore, all of the functional requirements were successfully tested through the use of manual unit tests.

Functional Tests for the Backend and the User Interface:

- Functional Requirement F1 tested by Test EH-1

  - Manual Test

  - HTML file was opened with Google Chrome, Microsoft Edge, and Firefox and game was played to ensure that it had compatibility with the prior specified web browsers. The UI was also manually checked for consistency throughout.

  - Test was successful .

- Functional Requirement F2 tested by Test IC-1

  - Manual Test

  - Test was run on multiple web browsers and each time the tester launched the software, and visually confirmed that it went to a standby state at the start menu. Different inputs were given to the program to ensure that it would only leave this state once the start button was clicked and all other inputs were ignored.

  - Test was successful.

- Functional Requirement F3 tested by Test IGC-1 and IGC-

  - Manual Test

  - This functional requirement states that the game will have no blocks on the grid at the start of the game and that the score will be set to 0. In order to do this, Test IGC-1 and IGC-2 were done manually. The tester would launch the game and press the start game button and ensure that the scoreboard read zero and that the grid did not have any blocks present excluding the first block that begins falling at the beginning of the game.

  - Test was successful.

- Functional Requirement F4 tested by Test IGC-2

  - Manual Test

  - This test was done to ensure that the game responds to the start game button and begins the game by allowing the WASD inputs to be applicable and function correctly. The game was run and the tester manually clicked the start button and inputted the WASD keys and visually checked that their inputs were being received by the software.

  - Test was successful.

- Functional Requirement F5 tested by Test GO-1

  - Manual Test

  - Game was played by the tester and blocks were stacked up until they reached the top of the grid. The tester then visually confirmed that the game goes to its end state and restricts user input of the WASD keys while displaying an end game screen.

  - Test was successful.

- Functional Requirement F6 tested by Tests BM-1, BM-2, BM-3, BM-4, BM-5

  - Manual Test

  - Game was played by the tester and the corresponding inputs of WASD, were inputted into the game. The tester then visually confirmed that the blocks had rotation functionality, left, down, and right movement.

  - Test was successful.

- Functional Requirement F7 tested by Tests CR-1

  - Manual Test

  - Game was played by the tester and blocks were stacked in a way that a full row was filled. The tester then confirmed that the blocks were cleared and above blocks were moved down by one row. Finally, they also checked that the score was incremented by +10.

  - Test was successful.

All of the tests for the functional requirements passed, leading the developers to conclude that the game satisfies the functional requirements within the documentation.

2 Nonfunctional Requirements Evaluation

2.1 Appearance

2.2 Appearance Requirements

1. AR-1

   Visual testing was done on different types of browsers (Microsoft Edge, Google Chrome, Mozilla FireFox) to ensure that all elements of the game were placed in the right positions. Users were also able to visually verify the width and colours of the grid and scoreboard were up the standards desired.

2. AR-2

Programmers ran a visual test to ensure that the company logo was on the main screen, and distinguishable. In this test, both of these factors were verified.

3. AR-3

   Through manually running the game and playing it for a few minutes, the programmers visually verified that the different Tetris shapes in the game were all a unique colour, and easily distinguishable from one another.

2.2.1 Style Requirements

1. SR-1

   A visual test was done, ensuring that all of the fonts throughout the program are consistent - which they were.

2. SR-2

   The game was manually run by both programmers and test users alike, both of which were able to verify that the blocks were falling smoothly, and that the screen changes the colors within the grid seamlessly.

2.3 Usability

1. UH-1

   Tetrileet was played by multiple users, who then filled out a questionnaire that asked them for feedback about if the game was understandable on launch. The feedback was very positive, and all users rated the game either a 4 or 5 on a scale of 1 - 5 in regards to this aspect.

2. UH-2

   Tetrileet was played by multiple users under a five minute timer. In the questionnaire given to them after the session, the team asked them if the game was fully understandable within 5 minutes of playing it. Feedback was positive, with most users rating this quality of the program a 5/5 on a scale of 1 - 5.

3. UH-3

   In the questionnaire given to the test group after playing the game, one question asked if the application is completely navigable and playable for a person with prior

knowledge only on how to use a mouse or keyboard. All users answered "yes" to this question.

4. UH-3

   The game was run and then played until a game over happened, upon which a programmer visually verified that the "play again" button appeared onscreen.

2.4 Performance

1. Speed and Latency Test - 1

   After selecting a difficulty, the game started, and a programmer manually started a timer to verify that the first block fell onto the grid within 5 seconds of one of the three difficulty buttons being pressed to start the game. The test gave the expected time, which was under 5 seconds.

2. Speed and Latency Test - 2
   A programmer placed the final block in order to end a running game, and then manually timed that the "game over" screen appears with the final score within three seconds of the game ending. This test was successful, as the "game over" screen with the final score appeared almost instantaneously upon the game ending.

3 Comparison to Existing Implementation

LoveDaisy Tetris did not have any testing that was found on their git repository. Therefore, we could not make a comparison based on our testing.

When comparing both implementations, the first thing to note is the usability factor. Our game is much easier to launch and navigate, and has more options than the original implementation. LoveDaisy Tetris could only be launched through a console, and was a raw Python file. This also decreases its portability, as it cannot be run on a computer that does not have Python and Ubuntu installed. Tetrileet is launched through a single click on the HTML file, can be run on any web browser that supports JavaScript - no prior installation of anything is required, as most people who use a computer will have a web browser on it. Another factor that differentiates the two is that Tetrileet implemented different difficulties to increase user engagement and increase playtime, as well just adding a better user experience overall. Skilled players can select the "hard" difficulty, while players who want an easier or standard game can pick "easy" or "medium". Lastly, it can be argued that Tetrileet looks more visually appealing than LoveDaisy Tetris - the original is locked to a small window, but Tetrileet has a full-screen UI with a background picture.

4 Unit Testing

Due to the fact that we chose not to use automated testing, no unit tests were performed in the overall testing of Tetrileet. The choice of not including automated testing is further expanded upon in section 6, "Automated Testing".

5 Changes Due to Testing

Throughout our manual testing, we found that our game had a bug, when it came to the rotation of shapes along the edge of the grid. In order to fix this bug we restricted the rotation of blocks along the edge of the grid. Furthermore, user testing for the UI found styling improvements that could be made which were also fixed for the final product. Some of these UI changes included changing font colors, adding company logos, increasing size of buttons, and changing locations of game components such as the upcoming block grid.

6 Automated Testing

Due to the nature of the software project, automated testing was found to not be an optimized way to test. We rather found that Manual testing, through the experience of actually playing the game and identifying bugs, was more effective. That is why we made the testing decision to not use Automated testing but rather focus on User Testing and Manual Testing. Additionally, our program only allows user inputs that it can recognize, so we were able to effectively verify every aspect of the game by manually recreating scenarios we wanted to test and noting the outcomes.

7 Trace to Requirements

Test Case	Requirement
TC-HD	F3, F5
TC-EH	F1
TC-IC	F2
TC-IGC	F3, F4
TC-BM	F6, F7
TC-CR	F7
TC-GO	F5
TC-AR	NF1.1
TC-SR	NF1.2
TC-UH	NF2.1, NF2.2, NF2.3, NF2.4, NF2.5, NF2.6, F3, F4, F5
TC-PR	NF3.1, NF3.3, NF3.4, NF3.6, NF3.8, F5

Table 2: Trace Between Test Cases and Requirement

8 Trace to Modules

Test Case	Module
TC-HD	M5, M6, M7, M8, M10
TC-EH	M2
TC-IC	M2
TC-IGC	M2, M5, M10
TC-BM	M1, M4, M5, M6, M7, M10
TC-CR	M1, M5, M6, M9, M10
TC-GO	M1, M4, M8, M10
TC-AR	M1, M2, M5, M7, M10
TC-SR	M2, M5, M7, M10
TC-UH	M1, M2, M4, M5, M6, M7, M8, M10
TC-PR	M1, M2, M5, M7, M10

Table 3: Trace Between Test Cases and Module

9 Code Coverage Metrics

All code testing was done manually by the members of AAA Solutions, with no third party testing resources being used to assess the code coverage of Tetrileet. This choice was made because all the features of the edge cases of the program were much easier to test by having

a user play the game in such a way that might cause errors in the current state, such as rotating a block at the edge of the grid. This and other edge cases were tested through both black-box and white-box methods, with the former involving a standard play-through of the game, while the latter involved thorough knowledge of the code and testing each individual function or module that might fail to function as required in a given state. The choice to rely solely on manual testing meant more time could be spent on discovering and fixing errors in the code, rather than developing automated test cases for a game so heavily reliant upon its user interface.