SE 3XA3: Test Report
BlockBuilder

Team 28, OAC
Andrew Lucentini, lucenta
Owen McNeil, mcneilo
Christopher DiBussolo dibussoc

December 5, 2018

Contents

1 Functional Requirements Evaluation **1**

2 Nonfunctional Requirements Evaluation **4**

2.1 Look and Feel 4
2.2 Usability and Humanity 4
2.3 Performance 5
2.4 Operational and Environmental 6

3 Comparison to Existing Implementation **6**

4 Unit Testing **6**

5 Changes Due to Testing **7**

5.1 Functional Requirements Evaluation 7
5.2 Look and Feel Evaluation 7
5.3 Usability and Humanity Evaluation 8
5.4 Performance Evaluation 8
5.5 Operational and Environmental Evaluation 8

6 Automated Testing **8**

7 Trace to Requirements **9**

8 Trace to Modules **11**

9 Code Coverage Metrics **13**

List of Tables

1 **Revision History** i
2 Trace Between Test Cases and Requirements 10
3 Trace Between Test Cases and Modules 12

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Dec 2, 2018	1.0	Creation of Test Report
Dec 5, 2018	1.1	Final changes made to Test Report

This document elaborates on the BlockBuilder testing results. Any changes made to BlockBuilder due to testing are identified in this document. In addition, this document provides a traceability matrix from test results to the BlockBuilder requirements and BlockBuilder Modules.

1 Functional Requirements Evaluation

Test Name: UI1

Results: The user was successfully able to move their player to the right with respect to the player's line of sight while the 'D' key was pressed.

Test Name: UI2

Results: The user was successfully able to move their player to the left with respect to the player's line of sight while the 'A' key was pressed.

Test Name: UI3

Results: The user was successfully able to move their player in the direction of the player's line of sight while the 'W' key was pressed.

Test Name: UI4

Results: The user was successfully able to move their player in the opposite direction of the player's line of sight while the 'S' key was pressed.

Test Name: UI5

Results: The user was successfully able to make their player simulate a jumping motion when the space bar was pressed while the user was "on top" of a block.

Test Name: UI6

Results: The user was able to successfully rotate their screen to the right when they moved the mouse in the right direction.

Test Name: UI7

Results: The user was able to successfully rotate their screen to the left when they moved the mouse in the left direction.

Test Name: UI8

Results: The user was able to successfully rotate their screen in the upward direction when they moved the mouse in the upward direction.

Test Name: UI9

Results: The user was able to successfully rotate their screen in the downward direction when they moved the mouse in the downward direction.

Test Name: UI10

Results: The user was able to successfully add blocks in the world in the desired locations with specified block types.

Test Name: UI11

Results: The user was able to successfully remove blocks in world at the desired locations provided the blocks were not stone.

Test Name: UI12

Results: The user was able to successfully change the block type that is to be added when the user adds a block. After the user pressed '1' on the keyboard, they placed a grass block in the world when adding a block. After the user pressed '2' on the keyboard, they placed a brick block when adding a block. After the user pressed '3' on the keyboard, they placed a brick block in the world when adding a block. After the user pressed '4' on the keyboard, they placed a stone block in the world when adding a block. Each test confirmed that the block to be added was successfully changed.

Test Name: UI13

Results: The user was able to successfully engage and disengage "flying mode". While in "flying mode", the user was able to move around freely as if the gravity in the world was turned off.

Test Name: UI14

Results: For of 9/10 instances, the user was able to successfully play the game while administering a variety of inputs into the system as all predicted outputs were displayed on the GUI correctly. In one of the instances, the test member noticed a bug in the player's movement as the player was always moving left. In addition, the test member could not exit flying mode in this instance. Overall, satisfaction was not high as the team considered this bug unacceptable and further investigated into the issue.

Test Name: UI15

Results: In every instance of attempting to place a block at the players feet, the tester found that the block was successfully placed despite the fact that one should logically not be able to, as it causes a collision with the player model. The issue was looked into and corrected to users can no longer place block inside the player position.

Test Name: VA1

Results: Worlds with 5, 10, and 30 blocks of selected types were individually and successfully generated after the code was run for each world.

Test Name: VA2

Results: The test code successfully generated the four types of block textures.

Test Name: VA3

Results: The python dictionary that contains all of the blocks in the world was successfully updated after blocks were added and removed in the world.

Test Name: GP1

Results: The user verified that they successfully collided with all blocks and they were not able to pass through any blocks.

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

Test Name: LF1

Results: 95% of the testers concluded that the overall resemblance of the original Minecraft (in terms of textures and game play) is satisfactory.

Test Name: LF2

Results: The testing individual confirmed that for each block type placed in the world, the respective textures for that block were successfully mapped to the cube.

2.2 Usability and Humanity

Test Name: UH1

Results: One of the team members asked their father, and younger sibling to play the game. Both users were able to successfully play the game and reported no difficulties. However, the younger sibling fell off the generated plane at one point and did not know how to stop falling. Overall, satisfaction was high as there was an easy solution to this problem.

Test Name: UH2

Results: Two users who have previously played Minecraft were provided BlockBuilder and were both successfully able to learn and play the game with no prior knowledge of the game or provided instructions.

2.3 Performance

Test Name: PP1

Results: Out of 1000 inputs, 98% of the inputs generated results in less than maximum 30ms. The average response time was 15ms.

Test Name: PP2

Results: The average FPS time for 10 instances of BlockBuilder, each running for 10 minutes, was 61.8 fps, which was better than the goal of 60 fps.

Test Name: PP3

Results: N/A as the test was removed from the test plan.

Test Name: PP4

Results: After attempting to run the game for 20 hours with a large number of automated inputs (block additions, removals and user movement), the team found that the game had run for the full 20 hours. However, it was found upon further inspection that the inputs were not producing correct outputs as the player was always moving to the left. It was estimated that this error occurred at approximately 15-18 hours into the test. Overall, satisfaction was high due to the fact that users will not likely play the game for periods this long.

Test Name: PP5

Results: N/A as this test was not performed as the test is very similar to the test UI15.

2.4 Operational and Environmental

Test Name: OE1

Results: The testing individual confirmed that BlockBuilder was able to run on the latest version of Mac OS and Windows. The tester played the game 5 times on each operating system, for a duration of 20 minutes each time.

3 Comparison to Existing Implementation

Aside from efficiency changes as well as coding style changes, the primary difference between OAC's implementation and the original implementation lies in the modularization and overall design of the project. The original project featured a single, 900 line main.py file that one could call ”spaghetti code”. The main drawback of having no structure to one's code is that it drastically reduces maintainability of the project, as well as making it significantly more difficult to implement new features in the future. The team realized this immediately and decided that instead of the main focus being adding new visual and game play features, the team would instead direct their efforts to completely redesigning the structure of the project to focus on maintainability and robustness. The team made this decision based on the original mission of the open source project, to create a simple game that could one day be used as a learning tool for other programmers looking to dip their feet into the world of game design and 3D rendering. The other reason for changing the implementation in this way was to make it as easy as possible to add new features in the future. To see a breakdown of the new design, see the Module Interface Specification.

4 Unit Testing

Only the functions in the module Functions.py were tested (See MIS for reference). Due to the nature of 3D gaming, unit testing provides little to no additional information as manual testing successfully tests the code in all other BlockBuilder test cases. Section 6 of this document provides more additional reasoning for the lack of unit/automated testing.

- Description of tests: Unit testing was performed on all functions contained in the Functions.py Module These tests checked the functions desired output against its actual output, given an input. Unit test were performed with pytest, having 5 test cases for each function. All test cases ensured that the functions were tested for regular and boundary conditions.
  Results: All functions passed all 5 test cases. All of the functions produced the correct outputs to their respective inputs.

5 Changes Due to Testing

5.1 Functional Requirements Evaluation

Minor changes were made to BlockBuilder due to tests corresponding to the functional requirements category. More specifically, changes to BlockBuilder were made due to test UI14. Out of 10 test cases for this test, one of the tests failed as user input was not behaving as expected in this test case. Further inspection into the code reveled that the reason due to the test only passing for 9/10 cases lied in game running at the desired 80fps. It was found that when multiple inputs were pressed repeatedly and simultaneously, the function that controls user input would not complete in time (i.e it would take longer than 1/80 seconds for the function to perform its task) for the next function call as more code within the function is being executed (due to multiple inputs). This caused some inputs to not be detected which caused the user to be stuck moving in certain directions. As a result, BlockBuilder was set to 60 fps to ensure there is enough time between each function call. The change was successful as test UI14 passed in 10 instances out of 10 tests.

5.2 Look and Feel Evaluation

There have been no changes made due to tests corresponding to the look and feel category.

5.3 Usability and Humanity Evaluation

Minor changes were made to BlockBuilder due to test corresponding to the usability and humanity evaluation. More Specifically, changes to BlockBuilder were made due to test UH1. One of the test users had difficulty in recovering after falling off of the 2D generated world. A simple fix to this issue was creating an unbreakable barrier that surrounds the 2D playing world to ensure users do not fall off of the map.

5.4 Performance Evaluation

There have been no changes made due to tests corresponding to the performance category.

5.5 Operational and Environmental Evaluation

There have been no changes made due to tests corresponding to the operational and environmental category.

6 Automated Testing

Initially, the team had planned to perform automated testing on functions that modified class state variables. However, when the time came to do so, the team immediately realized it was unfeasible and not useful to perform such automated tests. Due to the fact that BlockBuilder is a game designed to be used by a player using visual interaction, the team found that automated testing presented little to no benefit to validate the project. The only way to properly test BlockBuilder was to run the game and interact with it. Since the game state in the code is reflected visually, and errors in the implementation of the code would be visually reflected. For instance, adding a block in the world can be tested via unit testing, but proves to be useless as the success of adding a block into the world can be determined visually, which reflects the success of the implementation in the code. In the end, the team opted not to implement automated testing and instead focus more on manual, dynamic testing.

7 Trace to Requirements

This section shows the Traceability Matrix between the test cases and the requirements to which the tests apply.

Test Case	**Req.**
UI1	REQ8, REQ5
UI2	REQ7, REQ5
UI3	REQ6, REQ5
UI4	REQ9, REQ5
UI5	REQ11, REQ5
UI6	REQ10
UI7	REQ10
UI8	REQ10
UI9	REQ10
UI10	REQ13
UI11	REQ14, REQ19
UI12	REQ15, REQ16, REQ17, REQ18
UI13	REQ12
UI14	REQ20
UI15	REQ21, PP6, PP7, PP8
VA1	REQ13, REQ14
VA2	REQ13, REQ14
VA3	REQ13, REQ14
GP1	REQ22, REQ21
LF1	LF1, LF2
LF2	LF1, LF2
UH1	UH1, UH3, UH7, UH9
UH2	UH2, UH5, UH6, UH7, UH8
PP1	PP1
PP2	PP2
PP4	PP5
OE1	OE2, OE3

Table 2: Trace Between Test Cases and Requirements

8 Trace to Modules

This section shows the Traceability Matrix between the test cases and the modules to which the tests apply. As always, breakdowns of the modules can be seen in the MIS and Module Guide. For Reference, the modules are Window (M1), World (M2), Block (M3), Constants (M4) and Function (M5).

Test Case	**Req.**
UI1	M1, M2, M5
UI2	M1, M2, M5
UI3	M1, M2, M5
UI4	M1, M2, M5
UI5	M1, M2, M5
UI6	M1, M2, M5
UI7	M1, M2, M5
UI8	M1, M2, M5
UI9	M1, M2, M5
UI10	M1, M2, M3, M4
UI11	M1, M2, M3, M4
UI12	M2, M3, M4
UI13	M2, M4
UI14	M1, M2, M3, M4, M5
UI15	M2, M3
VA1	M1, M2, M3, M4, M5
VA2	M4
VA3	M2
GP1	M1, M2, M3
LF1	M1, M3
LF2	M1, M2, M3
UH1	M1
UH2	M1
PP1	M1, M2, M5
PP2	M1, M2, M5
PP4	M1, M2, M5
OE1	M1, M2, M3, M5

Table 3: Trace Between Test Cases and Modules

9 Code Coverage Metrics

The OAC team has managed to produce roughly 95%-98% code coverage through testing. This prediction was derived from inserting print statements in the code to cover all possible branches the code can take. The number of possible branches depends heavily on the inputs administered by the user. Provided that the user administers complex combinations of various inputs, almost all of the possible branches in the code are taken. In addition, the traceability matrix from test case to modules identifies that each module was tested multiple times, implying that each module was covered thoroughly.