SE 3XA3: Test Plan
CraftMaster

Group 307, 3 Craftsmen
Hongqing Cao 400053625
Sida Wang 400072157
Weidong Yang 400065354

February 29, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Feb 13th	1.0	Team info updated
Feb 27th	1.1	General Content added
Feb 28th	1.1	Tables and Figure added
Feb 28th	1.1	Grammar Check

This document describes the scope, approach, resources, and schedule of the testing stage in the development process of CraftMaster.

1 General Information

1.1 Purpose

The main purpose of the testing stage of this project is to verify that the design team correctly transformed the requirements into functionalities of the system and those functionalities perform properly.

1.2 Scope

The functional requirements will be tested with traceability matrices and non-functional requirements will be tested with the fit-criterions defined in the SRS document. Each function will be tested individually by unit testing and the system will be tested as a whole by integration testing and edge case testing. The test cases described in this document will act as a means to check the traceability between the project specifications.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirement Specification
OS	Operating System
GUI	Graphical User Interface

1.4 Overview of Document

This document will act as a guideline for testing activities for CraftMaster.

Table 3: **Table of Definitions**

Term	**Definition**
Python	The programming language that is used for the developement of this project
Pyglet	A Python library for the design of graphical user interface
Pytest	A Python library and framework for unit testing
Sandbox Games	A type of game that allows player to create, modify, and destroy the environment
3D Game	A game in three dimensions

2 Plan

2.1 Software Description

CraftMaster is a 3D sandbox game implemented in Python with **Pyglet** library, which allows users to play by building and destroying.

2.2 Test Team

The individuals responsible for testing are Hongqing Cao, Sida Wang, and Weidong Yang. All testing works are splat evenly to those three testers.

2.3 Automated Testing Approach

The test team will use **Pytest** to perform unit testing. After test cases created, **Pytest** can run automated tests and report code coverage and passes/failures. The time and task division of unit test cases are under the Gantt Chart of 2.5.

2.4 Testing Tools

The testing tool will be used is shown in table below.

2.5 Testing Schedule

See Gantt Chart **HERE**.

Testing Tool	Where to use	remarks
Pytest	Unit Testing	
Python Time Library	System Testing	NFR
Google Form	System Testing	NFR survey

Table 4: Testing Tools

3 System Test Description

3.1 Tests for Functional Requirements

Since the software game highly depends on the mouse and the keyboard inputs, and the game events are all triggered by these input elements, the test for functional requirements will be divided into two sections including testing for mouse inputs and testing for keyboard inputs.

3.1.1 Testing for Mouse Inputs

1. **TFR1: Test Game Start**
   Relevant Functional Requirement id: **FR1, FR2, FR3, FR4**

   Type: Functional, Dynamic, Manual

   Initial State: The software game is installed and ready to execute.

   Input: A cursor placement on the game icon and a double-click on the left mouse key.

   Output: The program opens the GUI frame with game scene and character loaded, and with a crosshair placed at the center of the window.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the GUI is opened, the game scene and character are loaded, and a crosshair is placed at the center of the window.

2. **TFR2: Test Crosshair Position Stability**
   Relevant Functional Requirement id: **FR4, FR4.1**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the crosshair is placed at the center of the GUI window.

Input: A sequence of movements of the character(controlled by keyboard inputs) followed by a sequence of block operations(controlled by keyboard and mouse inputs).

Output: The crosshair keeps in position(center of GUI window).

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the crosshair keeps its position.

3. **TFR3: Test Character Direction**
   Relevant Functional Requirement id: **FR6**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded.

   Input: A movement of the mouse.

   Output: The character changes its view direction.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the character changes its view direction.

4. **TFR4: Test Block Outline**
   Relevant Functional Requirement id: **FR9**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded. A block is near the character.

   Input: A movement of the mouse to aim the crosshair at the block.

   Output: The GUI shows the outline of the block(indicating the block is being aimed at).

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the outline of the block is shown.

5. **TFR5: Test Block Removal**
   Relevant Functional Requirement id: **FR10, FR15**

   Type: Functional, Dynamic, Manual

Initial State: The software game GUI is opened and the game scene and character are loaded. A block is near the character and the player has already aimed the crosshair at the block.

Input: A left-click on the mouse.

Output: The block is being removed from the window and the program plays a sound effect to notify.

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the block is removed and whether the program plays a sound effect to notify this event.

6. **TFR6: Test Block Build**
   Relevant Functional Requirement id: **FR11**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded. A block is near the character and the player has already aimed the crosshair at the block(the crosshair could be pointed to top, or bottom, or any side).

   Input: A right-click on the mouse.

   Output: The new block is being built at the position that is next to the surface of the existing block that the crosshair was pointed to and the program plays a sound effect to notify.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the new block is built in the right place. The tester will also check whether the program plays a sound effect to notify this event as well.

7. **TFR7: Test Game Close**

   Relevant Functional Requirement id: **FR13**

   Type: Functional, Dynamic, Manual

   Initial State: Based on the successful post condition of **TFR19 Test Cursor release**

   Input: A cursor placement on the close button(“X”) on the GUI followed by a left-click on the mouse.

Output: The GUI is closed and the game terminates.

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the GUI closes and the game terminates successfully.

8. **TFR8: Test Background Music**
   Relevant Functional Requirement id: **FR14**

   Type: Functional, Dynamic, Manual

   Initial State: The software game is installed and ready to execute.

   Input: A cursor placement on the game icon and a double-click on the left mouse key.

   Output: The program plays the background music of the game.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the program plays the background music.

9. **TFR9: Test Stone Block Operations**
   Relevant Functional Requirement id: **FR17**

   Type: Functional, Dynamic, Manual

   Initial State: A cursor is placed on a stone block

   Input: A left-click on the mouse.

   Output: The stone is not removed.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the stone is removed.

3.1.2 Testing for Keyboard Inputs

1. **TFR10: Test Forward Movement**
   Relevant Functional Requirement id: **FR5.1**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded. The character is free to move forward(no blocks are close to and at the front of the character).

   Input: A click on the “W” key on the keyboard.

Output: The character moves forward.

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the character moves forward.

2. **TFR11: Test Left Movement**
   Relevant Functional Requirement id: **FR5.2**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded. The character is free to move to the left(no blocks are close to and on the left of the character).

   Input: A click on the "A" key on the keyboard.

   Output: The character moves to the left.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the character moves to the left.

3. **TFR12: Test Backward Movement**
   Relevant Functional Requirement id: **FR5.3**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded. The character is free to move backward(no blocks are close to and at the back of the character).

   Input: A click on the "S" key on the keyboard.

   Output: The character moves backward

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the character moves backward.

4. **TFR13: Test Right Movement**
   Relevant Functional Requirement id: **FR5.4**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded. The character is free to move to the right(no blocks are close to and on the right of the character).

Input: A click on the “D” key on the keyboard.

Output: The character moves to the right

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the character moves to the right.

5. **TFR14: Test Jump Action**
   Relevant Functional Requirement id: **FR7**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded.

   Input: A click on the space key on the keyboard.

   Output: The character jumps once.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the character jumps.

6. **TFR15: Test Flying Mode**
   Relevant Functional Requirement id: **FR8, FR8.1, FR8.2, FR8.3**

   Type: Functional, Dynamic, Manual

   Initial State: The software game GUI is opened and the game scene and character are loaded.

   Input: A click on the tab key, followed a sequence of clicks on the “W” and “S” keys, and followed by a click on the tab key on the keyboard.

   Output: The character jumps once.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the tab key enables or disables the flying mode, and to check whether the “W” and “S” keys would control the character to fly under flying mode.

7. **TFR16: Test Block Type Change - Brick**
   Relevant Functional Requirement id: **FR11.1**

   Type: Functional, Dynamic, Manual

   Initial State: Based on the successful post condition of **TFR6 Test Block Build**. A block is near the character and the player has already aimed the crosshair at the block(the crosshair could be pointed to top, or bottom, or any side).

Input: A click on the “1” key on the keyboard, followed by a right-click on the mouse.

Output: A brick block is built.

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the block type is changed to brick.

8. **TFR17: Test Block Type Change - Grass**
   Relevant Functional Requirement id: **FR11.2**

   Type: Functional, Dynamic, Manual

   Initial State: Based on the successful post condition of **TFR6 Test Block Build**. A block is near the character and the player has already aimed the crosshair at the block(the crosshair could be pointed to top, or bottom, or any side).

   Input: A click on the “2” key on the keyboard, followed by a right-click on the mouse.

   Output: A grass block is built.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the block type is changed to grass.

9. **TFR18: Test Block Type Change - Sand**
   Relevant Functional Requirement id: **FR11.3**

   Type: Functional, Dynamic, Manual

   Initial State: Based on the successful post condition of **TFR6 Test Block Build**. A block is near the character and the player has already aimed the crosshair at the block(the crosshair could be pointed to top, or bottom, or any side).

   Input: A click on the “3” key on the keyboard, followed by a right-click on the mouse.

   Output: A sand block is built.

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the block type is changed to sand.

10. **TFR19: Test Cursor Release**
    Relevant Functional Requirement id: **FR12**

    Type: Functional, Dynamic, Manual

    Initial State: The software game GUI is opened.

    Input: A click on the "ESC" key on the keyboard

    Output: The cursor is released from the GUI, which means it does not control the direction change and block operations of the character.

    How test will be performed: The tester will perform the input action and conduct a visual test to check whether the cursor is released.

11. **TFR20: Test Block-Through Movement**
    Relevant Functional Requirement id: **FR16**

    Type: Functional, Dynamic, Manual

    Initial State: Based on the successful post condition of **Test Movements TFR10, TFR11, TFR12, TFR13**. The character is in a position where is surrounded by blocks.

    Input: A sequence of clicks on "W", "A", "S", "D".

    Output: The character does not move through the surrounding blocks.

    How test will be performed: The tester will perform the input action and conduct a visual test to check whether the character can move through the surrounding blocks.

3.2 Tests for Nonfunctional Requirements

3.2.1 Testing for Look and Feel Requirements

1. **TNFR1 Test Look and Feel**
   Relevant Nonfunctional Requirement id: **NFR1, NFR2**

   Type: Dynamic, Manual

   Initial State: The game is installed and is given to a group of teenagers with experience of playing Minecraft to play.

Condition: The group should be satisfied with the attractiveness and style of the game.

Result: The group is satisfied with the attractiveness and style of the game.

How test will be performed: One member of the testing team will assign a group of teenagers with experience of playing Minecraft, give them the game to play and hand out a survey after they play the game. The survey will ask for the attractiveness(a rank between 1-10) and "Minecraft-like Style"(basically saying how similar it is compared to Minecraft, a rank between 1-10) of the game, the average of the result should be above **AVG1** to pass the test.

3.2.2 Testing for Usability and Humanity Requirement

1. **TNFR2: Test Game Difficulty**
   Relevant Nonfunctional Requirement id: **NFR3**

   Type: Dynamic, Manual

   Initial State: The game is installed and is given to a group of random aged people to play.

   Condition: 95% of the players in the group should be able to play the game with no difficulty.

   Result: 95% of the players in the group play the game with no difficulty.

   How test will be performed: One member of the testing team will assign a group of ten people from different age groups, give them the game to play and hand out a survey after they play the game. The survey will ask for the difficulty of the game(a rank between 1-10), the average of the result should be below **AVG2** to pass this test.

2. **TNFR3: Test Learning Curve**
   Relevant Nonfunctional Requirement id: **NFR4**

   Type: Dynamic, Manual

   Initial State: The game is installed and is given to a group of random aged people to play.

   Condition: 95% of the players in the group should be able to learn the game within a time between **LEARN_MIN** to **LEARN_MAX** minutes.

Result: 95% of the players in the group successfully learn the game within a time between **LEARN_MIN** to **LEARN_MAX** minutes.

How test will be performed: One member of the testing team will assign a group of ten people from different age groups, give them the game to play and hand out a survey after they play the game. The survey will ask for the learning time of the game, the average of the result should be between **LEARN_MIN** to **LEARN_MAX** minutes. to pass this test.

3.2.3 Testing for Performance Requirements

1. **TNFR4: Test Speed**
   Relevant Nonfunctional Requirement id: **NFR5**

   Type: Dynamic, Manual

   Initial State: The game is installed.

   Condition: The software game should respond to game events in less than **RESPONSE** second for 99% of the interrogations and no response should take longer than **FALSE_RESPONSE** second.

   Result: The software game responds to game events in less than **RESPONSE** second for 99% of the interrogations and no response should take longer than **FALSE_RESPONSE** second.

   How test will be performed: One member of the testing team will use the Python build-in Time library to count the execution time of **OPERATION_NUM** operations in the program. All operations should take less than **RESPONSE** second.

2. **TNFR5: Test Availability**
   Relevant Nonfunctional Requirement id: **NFR6**

   Type: Dynamic, Manual

   Initial State: The game is installed.

   Condition: The software game should allow access to the game at different times.

   Result: The software game allows access to the game at different times.

   How test will be performed: One member of the testing team will write a driver to randomly access the program in **TIME** minutes for **TRIES** times. The driver will be used two times on two different dates. The result should be all successful accesses to pass this test.

3. **TNFR6: Test Reliability**
   Relevant Nonfunctional Requirement id: **NFR7**

   Type: Dynamic, Manual

   Initial State: The game is installed.

   Condition: The software game should run for five hours.

   Result: The software game successfully runs for five hours.

   How test will be performed: One member of the testing team will start the game and keep it for five hours. The game should run with no errors and failures during that time period to pass this test.

3.2.4 Testing for Operational and Environmental Requirements

1. **TNFR7: Test Adjacent System Effect**
   Relevant Nonfunctional Requirement id: **NFR8**

   Type: Dynamic, Manual

   Initial State: The game is installed.

   Condition: The software game should not produce any negative effects on adjacent system.

   Result: The software game does not produce any negative effects on adjacent system.

   How test will be performed: One member of the testing team will execute the game and monitor the activities in the process monitor on the computer. The game should not cause other programs to terminate to pass this test.

3.2.5 Testing for Maintainability and Support Requirements

1. **TNFR8: Test Adaptability**
   Relevant Nonfunctional Requirement id: **NFR10**

   Type: Dynamic, Manual

   Initial State: The game is available to be downloaded from internet.

Condition: The software game should be easily downloaded, installed, and opened onto both Windows and Linux OS.

Result: The software game can be easily downloaded, installed, and opened onto both a Windows OS and a Linux OS.

How test will be performed: One member of the testing team will download the game from the game website and install it onto both a Windows OS and a Linux OS and then open the game on both OS. There should be no unexpected issue happening during this process to pass the test.

3.2.6 Testing for Security Requirements

1. **TNFR9: Test Integrity**
   Relevant Nonfunctional Requirement id: **NFR11**

   Type: Dynamic, Manual

   Initial State: The game is installed.

   Condition: The software game should prevent low level threats.

   Result: The software game prevents low level threats and does not crash due to intentional abuse.

   How test will be performed: One member of testing team will write a specific threat test case for intentional abuse to the game. There should be no errors or failures happening to pass the test.

2. **TNFR10: Test Privacy**
   Relevant Nonfunctional Requirement id: **NFR12**

   Type: Static, Manuel

   Initial State: The game is installed.

   Condition: There should be no external user data generated during the game execution.

   Result: There is no external user data generated during the game execution.

   How test will be performed: The testing team will do a static analysis on the code and go through each module to check whether the program potentially produces external data. The result should be "No" to pass the test.

3.2.7 Testing for Cultural Requirements

1. **TNFR11: Test Cultural Politeness**
   Relevant Nonfunctional Requirement id: **NFR13**

   Type: Dynamic, Manual

   Initial State: The game is installed and is given to a group of people from different cultural groups to play.

   Condition: The group should have satisfaction with the cultural politeness of the game.

   Result: The group has satisfaction with the cultural politeness of the game.

   How test will be performed: One member of the testing team will assign a group of ten people from different cultural groups, give them the game to play and hand out a survey after they play the game. The survey will ask for the satisfaction of the cultural politeness of the game(a rank between 1-10), the average of the result should be above **AVG1** to pass this test.

3.2.8 Testing for Legal Requirements

1. **TNFR12: Test Compliance**
   Relevant Nonfunctional Requirement id: **NFR14**

   Type: combined with Dynamic and Static, Manual

   Initial State: The game is installed and the documentation is complete.

   Condition: The software product should not violate the Digital Millennuim Copy-right Act[1].

   Result: The software product does not violate the Digital Millennuim Copy-right Act[1].

   How test will be performed: One member of the testing team will show the game and the documentation to a legal expert and get feedback. The feedback should say the software product does not violate the Digital Millennuim Copy-right Act[1].

3.2.9 Testing for Health and Safety Requirements

1. **TNFR13: Test Safety**
   Relevant Nonfunctional Requirement id: **NFR15**

   Type: combined with Dynamic and Static, Manual

   Initial State: The game is installed and the documentation is complete.

   Condition: The software product should not generate any mental or physical threat to the players.

   Result: The software product does not generate any mental or physical threat to the players.

   How test will be performed: One member of the testing team will show the game and the documentation to a safety expert and get feedback. The feedback should say the software product does not generate any mental or physical threat to the players.

3.3 Traceability Between Test Cases and Requirements

FR	TFR1	TFR2	TFR3	TFR4	TFR5	TFR6	TFR7
FR1	✓						
FR2	✓						
FR3	✓						
FR4	✓	✓					
FR6			✓				
FR9				✓			
FR10					✓		
FR11						✓	
FR13							✓
FR15					✓		

FR	TFR8	TFR9	TFR10	TFR11	TFR12	TFR13	TFR14
FR5			✓	✓	✓	✓	
FR7							✓
FR14	✓						
FR17		✓					

FR	TRF15	TFR16	TFR17	TFR18	TFR19	TFR20
FR8	✓					
FR11		✓	✓	✓		
FR12					✓	
FR16						✓

Table 5: **Traceability Matrix for FRs**

NFR	TNFR1	TNFR2	TNFR3	TNFR4	TNFR5	TNFR6	TNFR7
NFR1	✓						
NFR2	✓						
NFR3		✓					
NFR4			✓				
NFR5				✓			
NFR6					✓		
NFR7						✓	
NFR8							✓

NFR	TNFR8	TNFR9	TNFR10	TNFR11	TNFR12	TNFR13
NFR10	✓					
NFR11		✓				
NFR12			✓			
NFR13				✓		
NFR14					✓	
NFR15						✓

Note: NFR9 is in terms of software updates and maintenance that is not testable.

Table 6: **Traceability Matrix for NFRs**

4 Tests for Proof of Concept

There were not many issues or conflicts happened to the project's first Proof of Concept Demonstration. This section will focus on the area of testings including the product delivery style and 3D game perspectives. The testing plan described in this section is used for future Proof of Concept Demonstration preparations.

Testing for Product Delivery Style

- **Test Executable Files**
  How test will be performed:

  The development team will generate Windows version and Linux version executable files according to each update of the source code. The test team will ensure that each generated executable file will automatically open the game, has its unique icon, and not invoke the terminal window.

- **Test Game Download**
  How test will be performed:

  The development team will upload the executable files on the game website for each release. The test team will ensure that the game download is stable, which means the game can be successfully downloaded and played.

Testing for 3D Game Perspectives

- **Test 3D Game Bugs**
  How test will be performed:

  By following the principle of edge case testing, the test team will play the game and try to discover as many 3D game bugs as possible to keep the system stable and reliable.

5 Comparison to Existing Implementation

6 Unit Testing Plan

The **Pytest** framework and library will be used to accomplish unit testing for CraftMaster. The **Pytest** library provides sufficient testing functionalities and also supports automated testing approach as described in section 2.3.

6.1 Unit testing of internal functions

There will be one test file for each module of the implementation. For each method within the module, two test cases will be made, including one boundary value test case and one equivalence test case. All mutator methods will be implicitly tested by performing test cases on accessor methods. With **Pytest** framework, each test case can be done by taking an individual method with inputs and giving it an expected return value. There will not be any drivers and stubs to be imported or implemented for the unit testing stage since it is supported by **Pytest** framework. Since **Pytest** supports coverage message, the test coverage will be checked according to the test report generated by **Pytest** instead of manual coverage metrics. For each test file, the coverage needs to exceed 95%.

6.2 Unit testing of output files

Since there is no data output from the system to the external environment, unit testing will not be performed on any output data file. Hence, this section does give any specification of unit testing.

7 Appendix

7.1 Symbolic Parameters

- **AVG1: 7**
- **AVG2: 3**
- **LEARN_MIN: 2**
- **LEARN_MAX: 30**
- **RESPONSE: 0.1**
- **FALSE_RESPONSE: 0.5**
- **OPERATION_NUM: 10**
- **TIME: 60**
- **TRIES: 100**

7.2 Survey Questions

Relevant Test case id	**Survey Question**
TNFR1	How do you like this game, rate it 1-10? How does this game remind you of Minecraft, rate it 1-10?
TNFR2	How difficult is this game, rate it 1-10?
TNFR3	How long does it take you to learn this game?
TNFR11	What is the level of cultural politeness of this game, rate it 1-10?

Bibliography

[1] New Media Rights. *Video Games and the law: Copyright, Trademark and Intellectual Property* [*https://www.newmediarights.org/guide/legal/Video_Games_law_Copyright_Trademark_Intellectual_Property*]. 2018.