SE 3XA3: Test Plan
CraftMaster

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

April 7, 2020

Contents

List of Tables

List of Figures

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
Apr 5th	2.0	Completed for Rev1

Table 1: **Revision History**

This document describes the scope, approach, resources, and schedule of the testing stage in the development process of CraftMaster. The format of this document follows the template provided by Dr.Bokhari and Thien Trandinh.

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

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

Table 2: **Table of Abbreviations**

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

Table 3: **Table of Definitions**

1.4 Overview of Document

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

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.

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**

2.5 Testing Schedule

See Gantt Chart **HERE**.

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**

   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 main menu rendered.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the GUI is opened and the main menu is rendered.

2. **TFR2: Test Main Menu and Sub-Menus directions**
   Relevant Functional Requirement id: **FR2.1**

   Type: Functional, Dynamic, Manual

   Initial State: The GUI is opened and the main menu is rendered.

   Input: Mouse clicks on the buttons on the main menu that direct to sub-menus and buttons on the sub-menus that direct to the main menu.

   Output: The program will go to sub-menus from the main menu and go to the main menu from sub-menus.

   How test will be performed: The tester will perform the input action and conduct a visual test to check whether the GUI will go to sub-menus from the main menu and go to the main menu from sub-menus.

3. **TFR3: Test New Game**
   Relevant Functional Requirement id: **FR2.1.1, FR3, FR4**

   Type: Functional, Dynamic, Manual

   Initial State: The GUI is opened and the game menu is rendered.

   Input: Mouse clicks on the ”start new game” button.

   Output: The program will load a new game scene, initialize the character, and place the crosshair 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 program will load a new game scene, initialize the character, and place the crosshair at the center of the window.

4. **TFR4: Test Saved Game**
   Relevant Functional Requirement id: **FR2.1.1, FR3, FR4**

   Type: Functional, Dynamic, Manual

   Initial State: Based on the successful post condition of **TFR14 Test Game Save**.The GUI is opened and the game menu is rendered. There is a saved game scene.

   Input: A mouse clicks on the ”load game” button and a mouse clicks on a game saving spot(Game One or Game Two).

   Output: The program will load a saved game scene, initialize the character, and place the crosshair 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 program will load a saved game scene, initialize the character, and place the crosshair at the center of the window.

5. **TFR5: Test Crosshair Position Stability**
   Relevant Functional Requirement id: **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.

6. **TFR6: 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.

7. **TFR7: 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.

8. **TFR8: 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.

9. **TFR9: Test Block Build**
   Relevant Functional Requirement id: **FR11, 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(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.

10. **TFR10: 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 test to check whether the program plays the background music.

11. **TFR11: Test Marble Block Operations**
    Relevant Functional Requirement id: **FR17**

    Type: Functional, Dynamic, Manual

    Initial State: A crosshair is placed on a marble block.

    Input: A left-click on the mouse.

    Output: The marble is not removed.

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

12. **TFR12: Test Day-Night Mode**
    Relevant Functional Requirement id: **FR2.1.2, FR 12.2**

    Type: Functional, Dynamic, Manual

    Initial State: The GUI is at the setting menu or at the ESC menu.

    Input: A left-click on the Day-Night Mode switch.

    Output: The Day-Night mode changes.

    How test will be performed: The tester will perform the input action and conduct a visual test to check whether the Day-Night mode changes.

13. **TFR13: Test Game Quit**
    Relevant Functional Requirement id: **FR2.2**

    Type: Functional, Dynamic, Manual

Initial State: The GUI is at the main menu.

Input: A left-click on "quit" button on the main menu.

Output: The program terminates and the GUI closes.

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

14. **TFR14: Test Game Save**
    Relevant Functional Requirement id: **FR12.3, FR13**

    Type: Functional, Dynamic, Manual

    Initial State: The GUI is at ESC menu.

    Input: A left-click on game saving and saving spot selection button.

    Output: The json file appears in the game folder.

    How test will be performed: The tester will perform the input action and conduct a visual test to check whether the json file appears in the game folder.

15. **TFR15: Test ESC menu to Game menu**
    Relevant Functional Requirement id: **FR12.1**

    Type: Functional, Dynamic, Manual

    Initial State: The GUI is at ESC menu.

    Input: A left-click on game saving and saving spot selection button.

    Output: The GUI returns to the game menu.

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

3.1.2 Testing for Keyboard Inputs

1. **TFR16: 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. **TFR17: 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. **TFR18: 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. **TFR19: 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. **TFR20: 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. **TFR21: 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 flying mode is activated, the character flies and then the flying mode is deactivated.

   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. **TFR22: Test Block Type Change - Brick**
   Relevant Functional Requirement id: **FR11.1**

   Type: Functional, Dynamic, Manual

Initial State: Based on the successful post condition of **TFR9 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. **TFR23: Test Block Type Change - Grass**
   Relevant Functional Requirement id: **FR11.2**

   Type: Functional, Dynamic, Manual

   Initial State: Based on the successful post condition of **TFR9 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. **TFR24: Test Block Type Change - Stone**
   Relevant Functional Requirement id: **FR11.3**

   Type: Functional, Dynamic, Manual

   Initial State: Based on the successful post condition of **TFR9 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 stone 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 stone.

10. **TFR25: Test ESC menu Open**
    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 and the ESC menu opens.

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

11. **TFR26: 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.

12. **TFR27: Test Trivial Keys**
    Relevant Functional Requirement id: **FR18**

    Type: Functional, Dynamic, Manual

    Initial State: The GUI is loaded with a game scene.

    Input: A sequence of mouse clicks on keys that are not assigned with any task, in this case use 'Z', 'P', '0', ';', 'CTRL'.

    Output: The system does nothing to respond and also does not crash.

How test will be performed: The tester will perform the input action and conduct a visual test to check whether the system does nothing to respond and does not crash.

3.1.3 Tests for Exception or Edge Case

- **TFR4.1: Test Saved Game Exception**
  Relevant Functional Requirement id: **FR2.1.1, FR3, FR4**

  Type: Functional, Dynamic, Manual

  Consideration: This case is to test for the exception that loading the saved game when there is no saved game. It is associated with **TFR4 Test Saved Game**.

  Initial State: Based on the successful post condition of **TFR14 Test Game Save**. The GUI is opened and the game menu is rendered. There is no saved game scene.

  Input: A mouse clicks on the ”load game” button and a mouse clicks on a game saving spot(Game One or Game Two).

  Output: The buttons that control the saving spot selection are not clickable and the system will not load a game scene.

  How test will be performed: The tester will perform the input action and conduct a visual test to check whether the button controls the saving spot selection is not clickable and the system will not load a game scene.

- **TFR20.1: Test Jump Action Edge**
  Relevant Functional Requirement id: **FR7**

  Type: Functional, Dynamic, Manual

  Consideration: This case is to test for the exception that the player controls the character to jump when the character is in a narrow space(height is not enough to jump). It is associated with **TFR20 Test Jump Action**.

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

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

  Output: The character does not jump.

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

- **TFR7.1: Test Block Outline Edge**
  Relevant Functional Requirement id: **FR9**

  Type: Functional, Dynamic, Manual

  Consideration: This case is to test for the edge case that the player points the crosshair to the middle of two blocks that are in contact with each other. It is associated with **TFR7 Test Block Outline**.

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

  Input: A movement of the mouse to aim the crosshair at the middle of two blocks.

  Output: The GUI shows the outline of the block that is closer to the crosshair.

  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 that is closer to the crosshair is shown.

- **TFR28: Test Menu Click Exception**
  Relevant Functional Requirement id: **FR2, FR2.1, FR2.1.1, FR2.1.2, FR2.2, FR 12, FR 12.1, FR 12.2, FR 12.3**

  Type: Functional, Dynamic, Manual

  Consideration: This case is to test for the exception that the player clicks on a empty space on any menu, which is associated with test cases on all menus. This test case will be used once for each menu.

  Initial State: The software game GUI is rendered with any menu.

  Input: A sequence of clicks on the empty spaces on the menu.

  Output: The software game does nothing to respond and does not crash.

  How test will be performed: The tester will perform the input action and conduct a visual test to check whether the software game does nothing to respond and does not crash.

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**

   Type: Dynamic, Manual

   Initial State: The game is installed and is given to a group of teenage players.

   Condition: The group should be 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, give them the game to play and hand out **Survey 1** after they play the game.

3.2.2 Testing for Usability and Humanity Requirement

1. **TNFR2: Test Game Difficulty**
   Relevant Nonfunctional Requirement id: **NFR2**

   Type: Dynamic, Manual

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

   Condition: The group should be able to 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 **Survey 2** after they play the game.

2. **TNFR3: Test Learning Curve**
   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: The group should be able to 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 **Survey 3** after they play the game.

3.2.3 Testing for Performance Requirements

1. **TNFR4: Test Speed**
   Relevant Nonfunctional Requirement id: **NFR4**

   Type: Dynamic, Automated

   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.

   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: **NFR5**

   Type: Dynamic, Automated

   Initial State: The game is installed.

   Condition: The software game should allow 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: **NFR6**

   Type: Dynamic, Automated

   Initial State: The game is installed.

   Condition: The software game should run 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: **NFR7**

   Type: Dynamic, Manual

   Initial State: The game is installed.

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

   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: **NFR9**

   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.

   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: **NFR10**

   Type: Dynamic, Manual

Initial State: The game is installed.

Condition: The software game should prevent low level threats.

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.

3.2.7 Testing for Cultural Requirements

1. **TNFR10: Test Cultural Politeness**
   Relevant Nonfunctional Requirement id: **NFR11**

   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.

   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 Survey 4 after they play the game.

3.2.8 Testing for Legal Requirements

1. **TNFR11: Test Compliance**
   Relevant Nonfunctional Requirement id: **NFR12**

   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].

   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. **TNFR12: Test Safety**
   Relevant Nonfunctional Requirement id: **NFR13**

   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.

   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.

3.2.10 Testing for Robustness Requirements

1. **TNFR13: Test Robustness**
   Relevant Nonfunctional Requirement id: **NFR14**

   Type: Dynamic, Manuel

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

   Condition: The software program does not crash or generate unstable behaviours when receiving unexpected input.

   How test will be performed: One member of the testing team will open the game and start a new game. When the game scene is loaded, the tester will click on a key on the keyboard that is not assigned any task and observe the behaviour of the program.

3.3 Traceability Between Test Cases and Requirements

FR	TFR1	TFR2	TFR3	TFR4	TFR4.1	TFR5	TFR6	TFR7
FR1	✓							
FR2	✓							
FR2.1		✓						
FR2.1.1			✓	✓	✓			
FR3			✓	✓	✓			
FR4			✓	✓	✓			
FR4.1						✓		
FR6							✓	
FR9								✓

FR	TRF7.1	TFR8	TFR9	TFR10	TFR11	TFR12	TFR13	TFR14
FR2.1.2						✓		
FR2.2							✓	
FR5.1								✓
FR9	✓							
FR10		✓						
FR11			✓					
FR12.1								✓
FR12.2						✓		
FR12.3								✓
FR13								✓
FR14				✓				
FR15		✓	✓					
FR17					✓			

FR	TFR15	TFR16	TFR17	TFR18	TFR19	TFR20	TRF20.1	TFR21
FR5.1		✓						
FR5.2			✓					
FR5.3				✓				
FR5.4					✓			
FR7						✓	✓	
FR8								✓
FR8.1								✓
FR8.2								✓
FR8.3								✓
FR12.1	✓							

FR	TFR22	TFR23	TFR24	TFR25	TFR26	TFR27	TFR28
FR2							✓
FR2.1							✓
FR2.1.1							✓
FR2.1.2							✓
FR2.2							✓
FR11.1	✓						
FR11.2		✓					
FR11.3			✓				
FR12				✓			✓
FR12.1							✓
FR12.2							✓
FR12.3							✓
FR16					✓		
FR18						✓	

Table 6: **Traceability Matrix for FRs**

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

NFR	TNFR8	TNFR9	TNFR10	TNFR11	TNFR12	TNFR13
NFR9	✓					
NFR10		✓				
NFR11			✓			
NFR12				✓		
NFR13					✓	
NFR14						✓

Note: NFR8 is in terms of software updates and maintenance that is not testable.

Table 7: **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, three test cases will be made, including one boundary value test case, one equivalence test case, and one exception test case. The unit testing stage will also check for the cases where nothing would happen to the system to ensure the robustness of the program. 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 90%.

6.2 Unit testing of output files

The output files of CraftMaster include two types of files, one is the game saving output file(in the format of json) produced by the game module, and another is the texture image produced by the Image Processor. To complete the unit testing for these output files, the testing team will mock up expected output files and check the actual output files against them automatically.

7 Appendix

7.1 NFR Surveys

- **Survey 1:** 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.
- **Survey 2:** 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.
- **Survey 3:** 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.
- **Survey 4:** 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.

7.2 Survey Questions

Relevant Survey id	Survey Question
1	How do you like this game, rate it 1-10? How does this game remind you of Minecraft, rate it 1-10?
2	How difficult is this game, rate it 1-10?
3	How long does it take you to learn this game?
4	What is the level of cultural politeness of this game, rate it 1-10?

7.3 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**

References

[1] ”Video Games and the law: Copyright, Trademark and Intellectual Property”, NewMediaRights, 2020. [Online]. Available: https://www.newmediarights.org/guide/legal/Video_Games_law_ Copyright_Trademark_Intellectual_Property. [Accessed: 06- Feb- 2020].