SE 3XA3: Test Plan
Dragon Age

Group 8: Team Eight
Stanley Liu (MacID: liuz23)
Toni Miharja (MacID: miharjat)
Zhi Zhang (MacID: zhangz1)

October 27 2017

Contents

1 Revision History **3**

2 General Information **3**
- 2.1 Purpose . . . 3
- 2.2 Test Objectives . . . 3
- 2.3 Acronyms, Abbreviations, and Symbols . . . 4
- 2.4 References . . . 4

3 Plan **4**
- 3.1 Software Description . . . 4
- 3.2 Test Team . . . 4
- 3.3 Automated Testing Approach . . . 4
- 3.4 Dynamic Testing . . . 5
- 3.5 Tools Used for Testing . . . 5
- 3.6 Testing Schedule . . . 5

4 System Test Description **5**
- 4.1 Tests for Functional Requirements . . . 5
  - 4.1.1 User Input . . . 5
  - 4.1.2 Enemy Attack . . . 8
  - 4.1.3 User Resources . . . 9
  - 4.1.4 Game Over Detection . . . 9
- 4.2 Tests for Nonfunctional Requirements . . . 10
  - 4.2.1 Usability . . . 10
  - 4.2.2 Performance . . . 11
- 4.3 Traceability Between Test Cases and Requirements . . . 12

5 Testing for Proof of Concept **12**
- 5.1 Dragon Tower Test . . . 12
  - 5.1.1 Tower Build Location . . . 12
  - 5.1.2 Tower Range . . . 13
  - 5.1.3 Tower Upgrade . . . 13
- 5.2 Bullet Test . . . 14
  - 5.2.1 Bullet Remove . . . 14
  - 5.2.2 Bullet Damage . . . 14

6 Comparison to Existing Implementation **14**

7 Unit Testing Plan **14**
- 7.1 Unit Testing for Internal Functions . . . 14
- 7.2 Unit Testing for Output Files . . . 15

8 Appendix **15**
- 8.1 Symbolic Parameters . . . 15
- 8.2 Usability Survey Questions . . . 15

1 Revision History

Date	Developer	Change	Revision
October 22, 2017	Zhi	Added introduction.	1.0
October 23, 2017	Stanley, Toni	Introduction, Plan, Testing for POC, Testing for functional requirements	1.5
October 24, 2017	Stanley, Zhi	User inputs testing, Testing for functional requirements, Testing for non-functional requirements	1.8
October 25, 2017	Stanley, Zhi	Testing for functional requirements, Testing for non-functional requirements	2.0
October 27, 2017	Stanley, Toni, Zhi	Testing for POC, Unit Testing Plan, Appendix	2.2

Table 1: Revision History: Requirements Documentation

2 General Information

2.1 Purpose

The purpose for testing this project is to find defects which may get created by the programmer while developing the software and to gain confidence that the software was implemented correctly with all the desired functions working.

2.2 Test Objectives

The objectives for testing this project is to make sure the full functionality of our game is being met and to fix any defects found during testing process. Before releasing our software to the users, all our tests must be thoroughly completed and passed to show robustness and readiness of the program.

2.3 Acronyms, Abbreviations, and Symbols

Abbreviation	Definition
POC	Proof of Concept
SRS	Software Requirement Specification
DA	Dragon Age

Table 2: Table of Abbreviation

Term	Definition
Static Testing	Testing that does not involve program execution
Dynamic Testing	Testing that runs the test cases which requires program execution
Unit Testing	Finds difference between the design model of a unit and its corresponding implementation
Automated Testing	Testing that runs automatically which does not has to be run by people
Boundary Testing	Testing that focuses on the boundary values of input parameters
Structural Testing	Testing that not designed to ensure correct function, but to verify structurally sound
Functional System Testing	Testing that ensures the software conforms with all requirements

Table 3: Table of Definitions

2.4 References

The game is a redevelopment of an existing project on GitHub named "Pokemon Tower Defense". Here is the link to this project: *https://github.com/kristing400/Pokemon-Tower-Defense*

3 Plan

3.1 Software Description

"Dragon Age" is a tower defense game running on personal computer. Basically, the user will send out three types of dragons to locate alongside a designed route to defend the homeland. As the enemy wave increases, the dragons can be upgraded to do more damages.

3.2 Test Team

The individuals responsible for testing are Stanley Liu, Toni Miharja, and Zhi Zhang.

3.3 Automated Testing Approach

As the testing framework is *pytest*, all the testings done automatically. The main idea to approach testing for the game is to write python files which contain all the functions for testing. Each function represents the testing for a particular function in the game. When running the testing file from command line, all the testing functions inside should be ran, then the testing result should be displayed including how many tests passed and in totally how many seconds.

3.4 Dynamic Testing

For dynamic testing, our team plan on to test the game while the game is running. During the running process, printing statements of corresponding tests will be printed onto the python IDLE.

3.5 Tools Used for Testing

The testing framework that will be utilized for the testing is *pytest*.

3.6 Testing Schedule

Team Member	Task	Date
Zhi Zhang	Intro	November 3, 2017
Stanley Liu	Enemy	November 3, 2017
Zhi Zhang	Path	November 10, 2017
Stanley Liu	Draw	November 10, 2017
Toni Miharja	Dragon Tower	November 10, 2017
Toni Miharja	Bullet	November 17, 2017
Toni Miharja	TimerFired	November 17, 2017
Stanley Liu	DragonAge(include mouse cursor)	November 24, 2017
Zhi Zhang	Non-functional Requirements	November 24, 2017

Table 4: Table of Definitions

Note: The first eight tasks are python classes with *.py* extension. Please see "src" folder in repository for more information about classes of the game. Each team member tests different classes of the game.

4 System Test Description

4.1 Tests for Functional Requirements

4.1.1 User Input

4.1.1.1 Select Dragon Tower

1. FR-SDT-1

    - Type: Functional, Dynamic, Manual
    - Initial State: No dragon has been selected
    - Input:User selects one dragon type
    - Output: The dragon is available for being built on the board
    - How test will be performed: The function that determines whether or not a dragon is being selected will be called. Following that the user will be able to drag and move the dragon on the board.

4.1.1.2 Place Dragon Tower

1. FR-PDT-1

    - Type: Functional, Dynamic, Manual
    - Initial State: No dragon has been placed on the board

- Input: User drags the dragon to invalid location coordinates(along the enemy path)
- Output: The program prompts the user saying that the current location is invalid
- How test will be performed: The function that determines whether or not the location coordinate is valid will be called with a predetermined invalid location coordinates as an input. If the output is false, the dragon tower will not be placed on the board.

2. FR-PDT-2

   - Type: Functional, Dynamic, Manual
   - Initial State: No dragon has been placed on the board
   - Input: User drags the dragon to valid location coordinates(along the enemy path)
   - Output: The game placed the dragon on this location when the user releases the mouse
   - How test will be performed: The function that determines whether or not the location coordinate is valid will be called with a predetermined invalid location coordinates as an input. If the output is true, the dragon tower will be placed on the board.

4.1.1.3 Upgrade Dragon Tower

1. FR-UDT-1

   - Type: Functional, Dynamic, Manual
   - Initial State: The dragon is not upgraded
   - Input: User selects the upgrade button with insufficient coins
   - Output: The program prompts the user saying that the coins are insufficient
   - How test will be performed: The function that determines whether or not the user has sufficient coins will be called. If the output is false, the dragon will not be upgraded.

2. FR-UDT-2

   - Type: Functional, Dynamic, Manual
   - Initial State: The dragon is not upgraded
   - Input: User selects the update button with sufficient coins
   - Output: The dragon is being upgraded 1 level
   - How test will be performed: The function that determines whether or not the user has sufficient coins will be called. If the output is true, the dragon will be upgraded.

3. FR-UDT-3

   - Type: Functional, Dynamic, Manual
   - Initial State: The dragon is at its highest possible level
   - Input: User selects the upgrade button
   - Output: The program prompts the user saying that the dragon can not being upgraded anymore
   - How test will be performed: The function that determines whether or not the dragon is at highest level will be called. If the output is true, the dragon can not be upgraded any more.

4.1.1.4 Buy New Dragon Tower

1. FR-BNDT-1
    - Type: Functional, Dynamic, Manual
    - Initial State: No new dragon tower has been bought
    - Input: User selects the tower type with insufficient coins
    - Output: The program prompts the user saying that the coins are not sufficient
    - How test will be performed: The function that determines whether or not the user has sufficient coins will be called. If false, the new dragon tower will not be purchased by the user.
2. FR-BNDT-2
    - Type: Functional, Dynamic, Manual
    - Initial State: No new dragon tower has been bought
    - Input: User selects the tower type with sufficient coins
    - Output: The user gets the new dragon tower
    - How test will be performed: The function that determines whether or not the user has sufficient coins will be called. If true, following that the user will be able to place the dragon tower.
3. FR-BNDT-3
    - Type: Functional, Dynamic, Manual
    - Initial State: No new dragon tower has been bought
    - Input: User selects the tower type with not sufficient space available on the board
    - Output: The program prompts the user saying that the the user can not have more dragons since there is no more available space to build on the board.
    - How test will be performed: The function that determines whether or not the user has more space to build a new dragon tower will be called. If false, the user will not be able to purchases more tower units.

4.1.1.5 Start/Pause State Selection

1. FR-SPSS-1
    - Type: Functional, Dynamic, Manual
    - Initial State: The game is running
    - Input: User presses start/stop button
    - Output: All the movements stop(the enemy and the dragon)
    - How test will be performed: The function that determines whether or not the user pressed the start/stop button will be called, if true, the game stops.
2. FR-SPSS-2
    - Type: Functional, Dynamic, Manual
    - Initial State: The game is stopped
    - Input: User presses start/stop button
    - Output: The game continues with moving units
    - How test will be performed: The function that determines whether or not the user pressed the start/stop button will be called, if true, the game continues.

4.1.2 Enemy Attack

4.1.2.1 Enemy Wave

1. FR-EW-1
    - Type: Functional, Dynamic, Manual
    - Initial State: No enemy enters the board
    - Input: User presses start/stop button at stop state
    - Output: One wave of enemy enters the designed path
    - How test will be performed: The function that determines whether or not the user starts the game will be called, if true, the enemy wave enters the board.

4.1.2.2 Enemy Wave Upgrade

1. FR-EWU-1
    - Type: Functional, Dynamic
    - Initial State: A small enemy wave at level 1 with slow movement
    - Input: User is leveled up
    - Output: The number of enemies per wave increases and moves faster each level
    - How test will be performed: The function that determines whether or not the user's level is upgraded will be called, if true, the enemy wave upgrade automatically.

4.1.2.3 Towers Defend Enemies

1. FR-TDE-1
    - Type: Functional, Dynamic
    - Initial State: Tower is static
    - Input: Enemy enters the launch range of the tower
    - Output: Tower launches bullets and hits the enemy, each hit does an amount of damage to the enemy's health
    - How test will be performed: The functions that determines whether or not the enemy is in launch range of the tower and whether or not the bullets his in the safety range of the enemy will be called, if true, the enemy's health should reduce a certain amount.
2. FR-TDE-2
    - Type: Functional, Dynamic
    - Initial State: Enemy moves and the tower launches bullets
    - Input: A bullet hit the enemy and the damage of the bullet is greater than the health of the enemy
    - Output: The enemy is killed and removed from the board
    - How test will be performed: The function that determines whether or not the enemy is killed will be called, if true, the enemy should be removed from the board.
3. ~~FR-TDE-3~~
    - ~~Type: Functional, Dynamic~~
    - ~~Initial State: Enemy moves along the path~~
    - ~~Input: The tower launches bullet but does not hit the enemy~~
    - ~~Output: Enemy keeps moving~~
    - ~~How test will be performed: The function that determines whether or not the hit is in the safety range of the enemy will be called, if false, no damage will be done to the enemy.~~

4.1.3 User Resources

4.1.3.1 Gain Coins

1. FR-GC-1
    - Type: Functional, Dynamic
    - Initial State: The user has certain amount of coins
    - Input: An enemy is being killed
    - Output: The user gains certain amount of coins
    - How test will be performed: The function that determines whether or not an enemy is killed will be called, if true, the amount of user's coins should increase.
2. FR-GC-2
    - Type: Functional, Dynamic
    - Initial State: User gains certain amount of coins when an enemy is being killed
    - Input: The user levels up
    - Output: The user gains more coins for each enemy being killed at each higher level
    - How test will be performed: The function that determines whether or not the user is leveled up will be called, if true, the value of the enemy will increase.

4.1.3.2 Use Coins

1. FR-UC-1
    - Type: Functional, Dynamic, Manual
    - Initial State: The user has certain amount of coins
    - Input: The user selects a product with insufficient coins
    - Output: The program prompts the user saying that his/her coins is not enough for purchasing this item
    - How test will be performed: The function that determines whether or not the user has sufficient coins will be called, if false, the purchase is failed.
2. FR-UC-2
    - Type: Functional, Dynamic, Manual
    - Initial State: The user selects a product with sufficient coins
    - Input: The user levels up
    - Output: The product is added to the user's bag and ready to use
    - How test will be performed: The function that determines whether or not the user has sufficient coins will be called, if true, the purchases is successed.

4.1.4 Game Over Detection

4.1.4.1 The User Exits Game

1. FR-GOD-1
    - Type: Functional, Dynamic, Manual
    - Initial State: The user is in the game
    - Input: The user quits the game
    - Output: The program ends and exits
    - How test will be performed: The function that determines whether or not the user closes the game program will be called, if true, the game exits.

4.1.4.2 The maximum number of enemies enter home base

1. FR-GOD-2

   - Type: Functional, Dynamic
   - Initial State:The user is in the game
   - Input: Maximum allowed enemy enters home base
   - Output: The game ends and displays "Game Over"
   - How test will be performed: The function that determines whether or not the number of enemy enters home base is at maximum will be called, if true, the game stops.

4.2 Tests for Nonfunctional Requirements

4.2.1 Usability

1. NFR-UB-1

   - Type: Structural, Static, Manual
   - Initial State: The game file is downloaded onto personal computers
   - Input: Launch the game on personal computers that run operating systems Windows, Mac OS and Linux
   - Output: The game should be opened and run successfully on operating systems.
   - How Test will be performed: Game will be run on three operating systems. The tester will make sure the game opens correctly on different operating systems.

1. NFR-UB-2

   - Type: Structural, Static, Manual
   - Initial State: The game is opened on personal computers
   - Input: Users play the game, use mouse to click on game buttons(start, menu, dragons, etc.)
   - Output: All the buttons works correctly, right outcome generated from clicking the button
   - How Test will be performed: Tester will play the game at least 5 times after each update of the game to make sure the mouse cursor and all buttons work correctly.

1. NFR-UB-3

   - Type: Structural, Static, Manual
   - Initial State: The testing team has already tested the previous two tests by downloading and playing the game
   - Input: Members in testing group are asked to rate the ease of installation, effectiveness of button clicks and overall satisfaction, etc of DA from 1 to 5
   - Output: The average rating on all categories is greater than 3
   - How Test will be performed: Each member in testing group is provided a questionnaire to complete based on the usability testing with a scale for each question where 1 represents very poor, 2 represents below expectations, 3 represents satisfactory, 4 represents above expectations, and 5 represents excellent. The average rating is calculated.

1. NFR-UB-4

   - Type: Structural, Static, Manual
   - Initial State: The testing team has already tested NFR-UB-1 and NFR-UB-2 on the existing game and has already played the existing game(Pokemon Tower Defense)

- Input: Members in testing group are asked to rate the ease of installation, effectiveness of button clicks and overall satisfaction, etc of existing game from 1 to 5
- Output: The average rating on all categories is greater than 3
- How Test will be performed: Each member in testing group is provided a questionnaire to complete based on the usability testing with a scale for each question where 1 represents very poor, 2 represents below expectations, 3 represents satisfactory, 4 represents above expectations, and 5 represents excellent. The average rating is calculated and should be lower than the average rating of DA implementation.

4.2.2 Performance

1. NFR-P-1

   - Type: Structural, Static, Manual
   - Initial State: Game is launched with default settings
   - Input: Users click on the game file and open the game
   - Output: The game is launched and loaded into the introduction screen under 2 seconds
   - How test will be performed: Members of the testing group or a group of randomly chosen users will be asked to launch the game. The time elapsed from users clicking the game file to the game is loaded into the introduction screen should be less or equal to 2 seconds.

1. NFR-P-2

   - Type: Structural, Static, Manual
   - Initial State: The game is launched and loaded to the game screen where users can start to play
   - Input: Users click on buttons in game including “menu”, “Start”, “Restart”, etc
   - Output: The reaction time of button clicks to achieve each button's function should be under 0.5 seconds
   - How test will be performed: Members of the testing group or a group of randomly chosen users will be asked to click on all buttons included in the game while playing the game, and make sure the time elapsed from users clicking the button to the function of button is achieved is less or equal to 0.5 seconds.

4.3 Traceability Between Test Cases and Requirements

Test Cases											
Requirements	FR-EW-1	FR-EWU-1	FR-TDE-1	FR-TDE-2	FR-TDE-3	FR-GC-1	FR-GC-2	FR-UC-1	FR-UC-2	FR-GOD-1	FR-GOD-2
4.1.2.1	x										
4.1.2.2		x									
4.1.2.3			x	x	x						
4.1.3.1						x	x				
4.1.3.2								x	x		
4.1.4.1										x	
4.1.4.2											x

Table 5: Traceability Matrix(1)

Test Cases						
Requirements	NFR-UB-1	NFR-UB-2	NFR-UB-3	NFR-UB-4	NFR-P-1	NFR-P-2
4.2.1	x	x	x	x		
4.2.2					x	x

Table 6: Traceability Matrix(2)

5 Testing for Proof of Concept

Proof of Concept testing is mainly focused on validating and verifying the means by which the team will perform automated testing as well as ensuring that the program that the team created for the Proof of Concept Demonstration is working correctly as expected.

5.1 Dragon Tower Test

5.1.1 Tower Build Location

1. DT-LOC-1

   - Type: Structural, Static, Manual
   - Initial State: Tower has not been built on the board.
   - Input: Invalid location coordinates (along the enemy path)
   - Output: The program prompts the user saying that the current location is invalid.
   - How test will be performed: The function that determines whether or not the location coordinate is valid will be called with a predetermined invalid location coordinates as an input. If the output is false, the user will be prompted with a message stating that the location is invalid.

1. DT-LOC-2

   - Type: Structural, Static, Manual
   - Initial State: Tower has not been built on the board.
   - Input: Valid location coordinates
   - Output: The tower will be built successfully at the location coordinates.

- How test will be performed: The function that determines whether or not the location coordinate is valid will be called with a predetermined valid location coordinates as an input. If the output is true, the tower will be built successfully.

1. DT-LOC-3

   - Type: Structural, Static, Manual
   - Initial State: Tower has not been built on the board.
   - Input: Invalid location coordinates (on top of another existing tower)
   - Output: The program prompts the user saying that the current location is invalid.
   - How test will be performed: The function that determines whether or not the location coordinate is valid will be called with a predetermined invalid location coordinates as an input. If the output is false, the user will be prompted with a message stating that the location is invalid.

1. DT-LOC-4

   - Type: Structural, Static, Manual
   - Initial State: Tower has not been built on the board.
   - Input: Invalid location coordinates (on the menu section, not on the playing field)
   - Output: The program prompts the user saying that the current location is invalid.
   - How test will be performed: The function that determines whether or not the location coordinate is valid will be called with a predetermined invalid location coordinates as an input. If the output is false, the user will be prompted with a message stating that the location is invalid because they cannot build tower on the menu window.

5.1.2 Tower Range

1. DT-RNG-1

   - Type: Structural, Static, Manual
   - Initial State: Tower has been built and is idle.
   - Input: Enemy spawns on location coordinate within range of the tower
   - Output: The tower will attack and fire bullet at the enemy unit.
   - How test will be performed: The function that determines whether or not the enemy is within range is invoked. The function calculates the distance between the tower and the enemy and if the distance is smaller or equal to the range, the function will return true and the tower will fire at the enemy.

5.1.3 Tower Upgrade

1. DT-UPG-1

   - Type: Structural, Static, Manual
   - Initial State: Tower has been built and there is no sufficient coins to upgrade tower by player
   - Input: Tower ID and current coin count of the player
   - Output: The tower will not be upgraded to the next level.
   - How test will be performed: The function that determines whether or not the player has sufficient coins to upgrade the existing tower on the map is invoked. The function will check and compare the current coins and the coins need to upgrade existing tower, if coins are not enough, return true.

5.2 Bullet Test

5.2.1 Bullet Remove

1. DT-BR-1

   - Type: Structural, Static, Manual
   - Initial State: Towers are placed, enemy wave is coming into the map
   - Input: Bullets are shot from towers at enemies within range
   - Output: Bullets should removed at the moment they touch enemies
   - How test will be performed: The function that determines whether the bullets are removed when hitting enemies is invoked. The function will check if the bullet list is empty, if it is, return true.

1. DT-BR-2

   - Type: Structural, Static, Manual
   - Initial State: Towers are placed, enemy wave is coming into the map
   - Input: Bullet does not hit the enemy then exit the map and hence must be removed
   - Output: Bullets should removed at the moment they touch enemies
   - How test will be performed: The function that determines whether the bullets are removed when they exit the map is invoked. The function check if the position of bullet is in board bound, if not, check if it is removed from bullet list, if it is removed, return true.

5.2.2 Bullet Damage

1. DT-BD-1

   - Type: Structural, Static, Manual
   - Initial State: Tower shoot bullets at enemies
   - Input: Bullets hit enemies
   - Output: The health of enemies are decreased
   - How test will be performed: The function that determines whether the health of enemy is decreased is invoked. The function check the enemies health before and after the bullet hits the enemy, if health decreases, return true.

6 Comparison to Existing Implementation

There are two tests that compare the existing implementation to the DA. Please refer to:

- Test NFR-UB-3, and NFR-UB-4 in Test for Non-Functional Requirements - Usability

7 Unit Testing Plan

The *pytest* framework will be utilized to do unit testing for this project.

7.1 Unit Testing for Internal Functions

Internal functions will also be tested whenever possible so as to ensure there is no bug that may affect the program. Unit testing for internal functions will be conducted by providing these methods with input values and comparing the generated output with the correct results. Unit tests hence can be created in this case and the team believe that they can be done fairly easily given the logical nature of the internal functions. This project will not need any drivers or stubs to be imported specifically for the testing as the individual classes import the necessary classes. For the purpose of this, our team will be using coverage metrics and determine how much of our code has been covered. We aim for a high code coverage that test all functions adequately by at least 85 percent coverage.

7.2 Unit Testing for Output Files

As for output files, DA does not generate any output file. Hence, unit testing for output files is not necessary at this point.

8 Appendix

8.1 Symbolic Parameters

Symbolic parameters are not used the test cases documentation.

8.2 Usability Survey Questions

Rate the following regarding to the game:
1: poor, 2: below expectations, 3: satisfactory, 4: above expectations, 5: excellent

- Ease of installation
- Effectiveness of button clicks
- Understandability of text
- Difficulty level
- Game interactivity
- Satisfaction of entertainment
- Overall satisfaction