

SE 3XA3: Test Plan Namcap

Team 2, VPB Game Studio
Prajvin Jalan (jalanp)
Vatsal Shukla (shuklv2)
Baltej Toor (toorbs)

October 31, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2016-10-30	1.0	Addition of content to sections 1.1, 1.2 and 1.4
2016-10-30	1.1	Addition of content to sections 2.1 and 2.2
2016-10-30	1.2	Addition of content to sections 2.3 and 2.4
2016-10-30	1.3	Addition of content to section 3.2
2016-10-30	1.4	Updates to content of subsections of 3.2 and 7.1
2016-10-30	1.5	Completion of content for 3.2 and update to 7.1
2016-10-31	1.6	Completion of content for 3.1

This document is the test plan for the Pacman redevelopment project, Namcap. The test plan outlines the testin methodologies and techniques to be used when testing the functionalities and characteristics of the system and its component parts.

1 General Information

1.1 Purpose

The purpose of testing is to ensure that the developed implementation functions correctly and to address any areas where the system is vulnerable. Through the formal specification of the testing methods and verification techniques, testing the implementation becomes mroe reliable.

1.2 Scope

As Namcap is a redevelopment of a classic arcade game, the test plan will aim to formalize the various functionality testing techniques as well as the usability tests utilized in order to ensure that the implementation meets the given requirement. This document will explicitly detail the different methods and testing tools to be utilized for this project.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	Definition
Abbreviation1	Definition1
Abbreviation2	Definition2

Table 3: **Table of Definitions**

Term	Definition
Term1	Definition1
Term2	Definition2

1.4 Overview of Document

This document will describe the testing methodologies to be utilized to verify Namcap as an implementation and development. The test plan will outline all testing tools, schedules, automated and manual tests, tests to address the requirements for the application, unit tests, and any additional testing performed on the PoC and existing implementation.

2 Plan

2.1 Software Description

Namcap is a redevelopment of the classic 2D arcade game Pacman. The gameplay involves the player sprite moving through a 2D level attempting to acquire (collide with) as many dots as possible to increase the score. Enemy sprites (ghosts in the original) will move throughout the level and upon collision with the player will cause the player to lose a life and/or end the game. The player can however consume (collide with) a power up to send the enemies back to the center of the level (when collision occurs). The implementation covers these aspects of the core gameplay (scoring, collision, movement, and enemy mechanics).

2.2 Test Team

The test team for Namcap is comprised of Prajvin Jalan, Vatsal Shukla, and Baltej Toor.

2.3 Automated Testing Approach

For the purposes of automated testing, the test team will use both JUnit and the built-in Robot class library. JUnit is a unit testing framework that will run automated tests for most logical components and any GUI components where applicable. The Robot class library will be used to automate testing that simulates user input. Based on the simulated user input, logical and GUI components will be tested to ensure that the appropriate collision and scoring responses occur.

2.4 Testing Tools

The automated unit testing tool JUnit and the Robot class library are the only testing tools that the team will use to verify the implementation.

2.5 Testing Schedule

Follow this link to the 'Namcap Gantt Project' (must have project file structure).

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Game Functionality Testing

A robot (automated) unit testing class will be implemented and used to test the mechanics of the game.

1. GFT1

Type: Functional, Dynamic, Automated

Initial State: Application is displaying the main menu page

Input: Cursor clicked on Start Game button

Output: New game is started and window is changed to reflect a new game state

How test will be performed: The robot class will place the cursor within the coordinates of the Start Game button and the robot will perform a left-click

2. GFT2

Type: Functional, Dynamic, Automated

Initial State: Within the game state

Input: Arrow keys

Output: Player moves in the respective direction (if path is clear)

How test will be performed: The robot class will virtually press the left/right/up/down arrow

3. GFT3

Type: Functional, Dynamic, Automated

Initial State: Player comes in contact with wall

Input: No input

Output: Player stops moving when coming in contact with the wall

How test will be performed: The robot class will output a line of text to the console indicating that the player has stopped due to collision with a wall

4. GFT4

Type: Functional, Dynamic, Automated

Initial State: Player comes in contact with enemy

Input: No input

Output: If player has more than 1 life, decrement lives. If player has one life, end game.

How test will be performed: The robot class will navigate the payer towards an enemy until they collide.

5. GFT5

Type: Functional, Dynamic, Automated

Initial State: Within the game state

Input: No input

Output: Enemies move on a valid path

How test will be performed: A JUnit test will be implemented to fail whenever the enemy moves through an obstacle.

6. GFT6

Type: Functional, Dynamic, Automated

Initial State: Player comes in contact with dots

Input: Arrow keys

Output: Dot disappears after collection

How test will be performed: The Robot class will navigate the player through the map and collect the dots. A function will be implemented to check if the dots disappear upon collection

7. GFT7

Type: Functional, Dynamic, Automated

Initial State: Player collects the big dot

Input: Arrow keys

Output: Big dot disappears after collection

How test will be performed: The robot class will navigate the player to the nearest big dot and collect it. A function will be implemented to check if the big dot disappears upon collection

8. GFT8

Type: Functional, Dynamic, Automated

Initial State: Player collects the big dot

Input: Arrow keys

Output: Player is able to collide with enemies

How test will be performed: The robot class will navigate the player to the nearest enemy, after collecting the big dot. A function will be implemented to check if the collision doesn't decrement the player's lives

9. GFT9

Type: Functional, Dynamic, Automated

Initial State: Player collects the big dot

Input: No input

Output: Enemies change colour

How test will be performed: A JUnit test will be implemented to check if the asset used to view the enemies has changed.

10. GFT10

Type: Functional, Dynamic, Automated

Initial State: Player collects all dots

Input: Arrow keys

Output: Game over screen is activated

How test will be performed: A JUnit test will be implemented to check which JFrame is currently active.

11. GFT11

Type: Functional, Dynamic, Automated

Initial State: Within game state

Input: Cursor clicked on exit game

Output: Application must terminate

How test will be performed: The robot class will move the cursor to the exit game button and perform a left-click. If the application closes, the test is successful.

12. GFT12

Type: Functional, Dynamic, Automated

Initial State: Player collects dot

Input: Arrow keys

Output: The points are increased

How test will be performed: A JUnit test will be implemented to check if the player's points were increased

13. GFT13

Type: Functional, Dynamic, Automated

Initial State: Player collects big dot

Input: Arrow keys

Output: The points are increased at twice the rate

How test will be performed: A JUnit test will be implemented to check if the player's points were increased by twice the standard rate

14. GFT14

Type: Functional, Dynamic, Automated

Initial State: Player collides with enemy after collection of big dot

Input: Arrow keys

Output: Enemy is removed from game and respawned back to their original cell

How test will be performed: A JUnit test will be implemented to check if the enemy's coordinates were reset to their original spot.

3.2 Tests for Non-functional Requirements

This section of System Testing will consist of manual tests since it would be infeasible to implement automated tests for a majority of the subsections of Non-functional Requirements. Manual tests will be performed by users, therefore many of the tests will correspond with the Usability Survey so testers can ensure that these requirements are met.

3.2.1 Look and Feel Requirements

Application Layout

1. AL1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the instructions page

Input: No user input

Output: Namcap instructions page

How test will be performed: User will look at the Namcap instructions page and ensure that the layout and color scheme of the display is similar (but not identical) to that of the original arcade Pacman

2. AL2

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the game board with game entities

Input: No user input

Output: Namcap game board with game entities

How test will be performed: User will look at the Namcap game board with its game entities and ensure that the layout and color scheme of the display is similar (but not identical) to that of the original arcade Pacman

3.2.2 Usability and Humanity Requirements

Understandable Objectives

1. UO1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the instructions page

Input: No user input

Output: Namcap instructions page in English

How test will be performed: User will look at the Namcap instructions page and ensure that the instructions are in English

2. UO2

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the instructions page

Input: No user input

Output: Namcap instructions page clearly defining objectives

How test will be performed: User will look at the Namcap instructions page and ensure that the objective of the game is clear before playing the game

Understandable Gameplay

1. UG1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the game board with game entities

Input: Custom user input for gameplay

Output: Game display updates accounting for movement, collision, scoring, and enemy mechanics

How test will be performed: User will play Namcap (one playthrough - 3 lives) and be able to successfully maneuver the Player through dots and enemies, reinforcing their understanding of the game's controls and objectives

3.2.3 Performance Requirements

Response Time

1. RT1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the game board with game entities

Input: Custom user input for gameplay

Output: Game display updates player movement based on keyboard input with a delay less than Θ

How test will be performed: User will play Namcap and note down any instances where their player does not respond to their keyboard inputs within a delay of Θ

Unexpected Failures

1. UF1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the game board with game entities

Input: Custom user input for gameplay testing

Output: Game display updates for user input and application does not crash

How test will be performed: User will play Namcap and note down any instances where during their testing, the application unexpectedly crashes (the number of crashes will be within Ω of total user tests)

3.2.4 Maintainability and Support Requirements

Operating System Support

The primary operating systems that Namcap is being tested on currently are Windows and Mac OS since these are the most readily available for testers.

1. OSS1

Type: Functional, Dynamic, Manual

Initial State: A Namcap jar file is available on a machine running Windows

Input: User runs the jar file

Output: Namcap starts up and game can be played

How test will be performed: User will run the jar file on their computer (any Windows version with a JVM) and should be able to successfully play the game (one playthrough)

2. OSS2

Type: Functional, Dynamic, Manual

Initial State: A Namcap jar file is available on a machine running Mac OS

Input: User runs the jar file

Output: Namcap starts up and game can be played

How test will be performed: User will run the jar file on their computer (any Mac OS version with a JVM) and should be able to successfully play the game (one playthrough)

3.2.5 Security Requirements

Open-Source Code

1. OSC1

Type: Structural, Static, Manual

Initial State: Open-source repository is publicly accessible

Input: User accesses the open-source repository (visits the repository link)

Output: User is able to view the application source code

How test will be performed: User will visit the repository link and be able to view the application source code

3.2.6 Cultural Requirements

Offensive Symbols or Text

1. OST1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the instructions page

Input: No user input

Output: Namcap instructions page

How test will be performed: User will look at the Namcap instructions page and ensure that it contains no offensive symbols or text

2. OST2

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the game board with game entities

Input: No user input

Output: Namcap game board with game entities

How test will be performed: User will look at the Namcap game board with its game entities and ensure that it contains no offensive symbols or text

3.2.7 Legal Requirements

Legal Violation

1. LV1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the instructions page

Input: No user input

Output: Namcap instructions page

How test will be performed: User will look at the Namcap instructions page and ensure that literary work and listed game entities have been altered such that the redevelopment is not too similar to the original Pacman

2. LV2

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the game board with game entities

Input: No user input

Output: Namcap game board with game entities

How test will be performed: User will look at the Namcap game board with its game entities and ensure that literary work and game entities have been altered such that the redevelopment is not too similar to the original Pacman

3.2.8 Health and Safety Requirements

Break Prompt

1. BP1

Type: Functional, Dynamic, Manual

Initial State: Application is displaying the instructions page

Input: No user input, application runs for Φ

Output: After Φ , application pauses and prompts the user to take a break

How test will be performed: User will let the application run for Φ and ensure that when the time passes the application pauses appropriately and prompts them to take a break

4 Tests for Proof of Concept

4.1 Area of Testing1

Title for Test

1. test-id1

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

2. test-id2

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

4.2 Area of Testing2

...

5 Comparison to Existing Implementation

6 Unit Testing Plan

6.1 Unit testing of internal functions

6.2 Unit testing of output files

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS; their values are defined in this section for easy maintenance.

Table 4: **Symbolic Constants**

Constant	Value	Description
Θ	0.5 seconds	Response time between user actions and in-game operations
Ω	1 %	Percent of application tests that will cause unexpected failures in the application
Φ	2 hours	Duration that application runs for before prompting user to take a break

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.