2212 Virtual Pet Project

Testing Documentation

Team 75: Yonas Asmelash, Julia Norrish, Jessica Wang

Table of Contents

Main Page

Task/Part	**Contributors**	**Description/Notes**
Implementation: Documentation	Jessica, Julia, Yonas	Document the testing component of the project
Implementation: Data	Jessica, Julia, Yonas	Created the game assets and background, and found music and SFX online.
Implementation: README	Yonas	Provide those viewing the project/repo with a comprehensive understanding of the project
Implementation: Keyboard	Jessica	Allow user to interact with game

Shortcuts		via keyboard shortcuts
Implementation: UI	Jessica, Julia, Yonas	Create GUI for game
Implementation: Main Menu	Jessica, Julia, Yonas	Present user with a main menu to navigate game
Implementation: Tutorials	Jessica	Provide user with tutorial to understand game
Implementation: New Game & Pet Selection or Load	Jessica, Julia, Yonas	Allow user to create a new save and choose a new pet or load a previous save file
Implementation: Progress bars	Jessica, Julia	Allow users to see pet stats in the form of bars. The percentage font colour changes based on the level of the stat.
Implementation: Commands	Jessica, Julia	Allow user to interact with pet
Implementation: Inventory	Jessica, Julia	Allow user to view inventory
Implementation: Keeping Score	Jessica	Update score when going on space mission, and display score on screen
Implementation: Sprites	Jessica, Yonas	Added sprites to enhance visual experience
Implementation: Parental Controls	Jessica	Allow parent to control downtime, view statistics, and revive pets
Implementation: Housekeeping & Error Handling	Jessica, Julia, Yonas	Implement error handling and debugging messages throughout
Implementation: Sound	Yonas	Added SFX to the game (background music, clicks, etc.)
Testing: Main Page	Jessica	Title and subtitle, contribution table, table of contents, and
Testing: Introduction	Julia	Overview of the testing documentation
Testing: Test Plan	Jessica	Overview of problem being solved, general objectives of

		project, lists references to other documents that provide context in the understanding of this document
Testing: Summary	Yonas	Brief summary of document as well as terms/notations table
Delivery: Video Demo	Yonas	Uploaded game demo video outlining all functional requirements
Project Management	Jessica, Julia, Yonas	How the team contributed to project management (Ex. meeting minutes, gitlab issues, etc.)
Grading	Jessica, Julia, Yonas	Graded, attended meeting, uploaded video, peer reviewed

Introduction

Overview

This document details the implementation and testing for our virtual pet game, AstroPets. This game simulates caring for a pet by encouraging players to meet their pet's needs such as feeding, playing with, and maintaining its health. Through the testing phase of the project, our goal is to ensure that the game meets the functional and nonfunctional requirements, while providing an engaging, user-friendly experience.

Testing will focus on verifying that the gameplay mechanics function as expected. We will conduct testing to identify any errors that need to be resolved.

Objectives

The main objective of the implementation and testing process is to ensure our game meets all the specifications, provides a positive user-experience, and is error-free. Specifically, we aim to:

- Implement all functional and non-functional requirements
- Validate that all core functions, such as feeding, playing, and health management work properly
- Ensure our GUI is intuitive, responsive, and engaging
- Identify and resolve any errors that arise throughout the process
- Create JUnit tests to ensure each method is working correctly
- Test edge cases to ensure stability

References

Group Project Specification - This document outlines the overall expectations, requirements, and assessment criteria for the project.

Requirements Documentation - This document provides a comprehensive breakdown of the functional and non-functional requirements that guided the creation of this design document.

Design Documentation – This document provides details on the design of the game, including the UML class diagrams and UI mock-ups.

Test Plan

Unit Testing

Unit testing for the Virtual Pet Game application will be implemented using JUnit 5 to test individual components and methods in isolation. The focus will be on testing the business logic of the application, particularly the model classes that handle game mechanics, pet statistics, and inventory management.

Inventory.java Test Plan

Test Case Name:	Test Add Item
Test Case Description:	Verifies that an item can be successfully added to the inventory
Test Prerequisites:	1. A newly initialized empty Inventory object
Test Steps	1. Create a new Food object "Kibble" 2. Add the Food object to the inventory 3. Get the list of inventory items 4. Verify the item was added correctly
Expected Results:	- The addItem method returns true - The inventory contains exactly one item - The item's name matches "Kibble"
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit test

Test Case Name:	Test Case Name: Test Remove Item
Test Case Description:	Test Case Description: Verifies that an item can be successfully removed from the inventory
Test Prerequisites:	An Inventory object containing a Food item
Test Steps	Test Steps:

	1. Create a new Food object "Kibble" 2. Add the Food object to the inventory 3. Remove the Food object from the inventory 4. Get the list of inventory items 5. Verify the item was removed correctly
Expected Results:	Expected Results: - The removeItem method returns true - The inventory contains zero items after removal
Test Category:	Test Category: Unit Test
Requirement:	Requirement: 3.1.8 Player Inventory
Automation	Automation: Yes - JUnit test

Test Case Name:	Test Inventory Limit
Test Case Description:	Verifies that the inventory has a maximum capacity limit
Test Prerequisites:	A newly initialized empty Inventory object
Test Steps	1. Add 20 Food items to the inventory (one at a time) 2. Attempt to add one more Food item beyond the limit 3. Get the list of inventory items 4. Verify the inventory respects its capacity limit
Expected Results:	- The addItem method returns true for the first 20 items - The addItem method returns false for the 21st item - The inventory contains exactly 20 items
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit test

Test Case Name:	Test Contains
Test Case Description:	Verifies that the inventory can correctly identify if it contains a specific item
Test Prerequisites:	An Inventory object containing a Food item
Test Steps	1. Create a Food object "Kibble" and add it to the inventory 2. Create another Food object with the same name "Kibble" 3. Create a different Food object "Apple" 4. Test if the inventory contains both items
Expected Results:	- The contains method returns true for an item with the same name as an item in inventory - The contains method returns false for an item with a different name
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit Test

Player.java Test Plan

Test Case Name:	Test Player Initialization
Test Case Description:	Verifies that a Player object is correctly initialized with a pet and an inventory
Test Prerequisites:	None
Test Steps	1. Create a new Dog object named "Rex" 2. Create a new Player object with the Dog object 3. Verify the Player has the correct pet 4. Verify the Player has a non-null inventory
Expected Results:	- The player's pet matches the Dog object - The player has a valid inventory object
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory

Automation	Yes - JUnit test

Test Case Name:	Test Feed Pet
Test Case Description:	Verifies that feeding a pet increases its hunger level and removes the food from inventory
Test Prerequisites:	Player object with a Dog pet and an inventory containing food
Test Steps	1. Create a Food object "Kibble" 2. Add the Food object to the player's inventory 3. Set the dog's initial hunger to 80 4. Feed the pet with the Food object 5. Verify the pet's hunger has increased 6. Verify the Food object has been removed from inventory
Expected Results:	- The pet's hunger level increases above the initial value of 80 - The food item is no longer in the player's inventory
Test Category:	Unit Test
Requirement:	3.1.7 Commands - Feed
Automation	Yes - JUnit Test

Test Case Name:	Test Gift to Pet
Test Case Description:	Verifies that giving a gift to a pet increases its happiness and removes the gift from inventory
Test Prerequisites:	A Player object with a Dog pet and an inventory containing a toy
Test Steps	1. Create a Toy object "Ball" 2. Add the Toy object to the player's inventory 3. Set the dog's initial happiness to 80 4. Give the gift to the pet 5. Verify the pet's happiness has increased

	6. Verify the Toy object has been removed from inventory
Expected Results:	- The pet's happiness level increases above the initial value of 80 - The toy item is no longer in the player's inventory
Test Category:	Unit Test
Requirement:	3.1.7 Commands - Give Gift
Automation	Yes - JUnit test

Test Case Name:	Test Send To Bed
Test Case Description:	Verifies that sending a pet to bed changes its state to SLEEPING
Test Prerequisites:	A Player object with a Dog pet
Test Steps	1. Send the pet to bed 2. Verify the pet's state has changed to SLEEPING
Expected Results:	- The pet's current state is set to SLEEPING
Test Category:	Unit Test
Requirement:	3.1.7 Commands - Go to bed
Automation	Yes - JUnit Test

Test Case Name:	Test Play With Pet
Test Case Description:	Verifies that playing with a pet increases its happiness
Test Prerequisites:	A Player object with a Dog pet
Test Steps	1. Set the dog's initial happiness to 80 2. Play with the pet 3. Verify the pet's happiness has increased
Expected Results:	- The pet's happiness level increases above the

	initial value of 80
Test Category:	Unit Test
Requirement:	3.1.7 Commands - Play
Automation	Yes - JUnit Test

Test Case Name:	Test Take To Vet
Test Case Description:	Verifies that taking a pet to the vet increases its health
Test Prerequisites:	A Player object with a Dog pet
Test Steps	1. Set the dog's initial health to 50 2. Take the pet to the vet 3. Verify the pet's health has increased
Expected Results:	- The pet's health level increases above the initial value of 50
Test Category:	Unit Test
Requirement:	3.1.7 Commands - Take to the Vet
Automation	Yes - JUnit Test

Test Case Name:	Test Pet Exercise
Test Case Description:	Verifies that exercising a pet increases its health and decreases energy and hunger
Test Prerequisites:	A Player object with a Dog pet
Test Steps	1. Set the dog's initial health to 50 2. Exercise the pet 3. Verify the pet's health has increased
Expected Results:	- The pet's health level increases above the initial value of 50
Test Category:	Unit Test
Requirement:	3.1.7 Commands - Exercise

Automation	Yes - JUnit Test

Game.java Test Plan

Test Case Name:	Test Create New Pet
Test Case Description:	Verifies that a new pet can be created with the specified name and type
Test Prerequisites:	A newly initialized Game object and a temporary directory for save files
Test Steps	1. Create a new pet named "Rex" of type "dog" 2. Get the player object 3. Verify the player has a pet 4. Verify the pet's name is "Rex" 5. Verify the pet is an instance of Dog
Expected Results:	- The createNewPet method returns true - The player has a non-null pet - The pet's name is "Rex" - The pet is an instance of Dog
Test Category:	Unit Test
Requirement:	3.1.4 New Game & Pet Selection
Automation	Yes - JUnit Test

Test Case Name:	Test Start And Stop Game
Test Case Description:	Verifies that the game can be started and stopped correctly
Test Prerequisites:	A Game object with a created pet
Test Steps	1. Create a new pet 2. Start the game 3. Wait for a short period 4. Stop the game 5. Verify pet IDs were saved
Expected Results:	- The list of existing pet IDs is not empty after stopping the game

Test Category:	Unit Test
Requirement:	3.1.12 Housekeeping & Error Handling
Automation	Yes - JUnit Test

Test Case Name:	Test Save And Load Pet
Test Case Description:	Verifies that a pet can be saved and loaded correctly
Test Prerequisites:	A Game object
Test Steps	1. Create a new pet named "Rex" of type "dog" 2. Modify the pet's stats (health=75, hunger=80) 3. Save the pet 4. Disable parental controls 5. Clear the current pet 6. Load the pet back 7. Verify the loaded pet has the correct properties
Expected Results:	- The loadPet method returns true - The loaded pet is not null - The loaded pet's name is "Rex" - The loaded pet's health is 75 - The loaded pet's hunger is 80
Test Category:	Unit Test
Requirement:	3.1.5 Save/Load Game State
Automation	Yes - JUnit Test

Test Case Name:	Test Get Available Pets
Test Case Description:	Verifies that all available pets can be retrieved
Test Prerequisites:	A Game object
Test Steps	1. Create a pet named "Rex" of type "dog" 2. Get the pet's ID

	3. Create a pet named "Bubbles" of type "fish" 4. Get the pet's ID 5. Get available pets 6. Verify both pets are available
Expected Results:	- The available pets map contains both pet IDs - The names associated with the pet IDs are correct
Test Category:	Unit Test
Requirement:	3.1.5 Save/Load Game State
Automation	Yes - JUnit Test

Test Case Name:	Test Get Alive Pets
Test Case Description:	Verifies that only alive pets are retrieved
Test Prerequisites:	A Game object
Test Steps	1. Create a pet named "Rex" of type "dog" 2. Get the pet's ID 3. Create a pet named "Bubbles" of type "fish" 4. Get the pet's ID 5. Kill the fish pet by setting its health to 0 6. Update the pet's state 7. Save the pet 8. Get alive pets 9. Verify only the dog pet is alive
Expected Results:	- The alive pets map contains only the dog's ID - The alive pets map does not contain the fish's ID
Test Category:	Unit Test
Requirement:	3.1.5 Save/Load Game State, 3.1.6 Vital Statistics & Rules
Automation	Yes - JUnit Test

Test Case Name:	Test Revive Pet
Test Case Description:	Verifies that a dead pet can be revived
Test Prerequisites:	A Game object
Test Steps	1. Create a pet named "Rex" of type "dog" 2. Get the pet's ID 3. Disable parental controls 4. Kill the pet by setting its health to 0 5. Update the pet's state 6. Verify the pet is in the DEAD state 7. Save the pet 8. Revive the pet 9. Verify the pet's stats are reset to maximum 10. Verify the pet's state is NORMAL
Expected Results:	- The revivePet method returns true - The pet's health is 100 - The pet's happiness is 100 - The pet's hunger is 100 - The pet's sleep is 100 - The pet's state is NORMAL
Test Category:	Unit Test
Requirement:	3.1.11.3 Revive Pet
Automation	Yes - JUnit Test

Test Case Name:	Test Parental Controls
Test Case Description:	Verifies that parental controls can be set and verified
Test Prerequisites:	A Game object
Test Steps	1. Set parental controls with time restriction enabled, start hour = 8, end hour = 20, password = "testpass" 2. Verify time restriction is enabled 3. Verify start hour is 8

	4. Verify end hour is 20 5. Verify correct password passes verification 6. Verify incorrect password fails verification 7. Reset play time stats 8. Verify total play time is 0 9. Verify last session time is 0 10. Verify average session time is 0
Expected Results:	- Time restriction is enabled - Start hour is 8 - End hour is 20 - Password verification passes with "testpass" - Password verification fails with "wrongpass" - Total play time is 0 after reset - Last session time is 0 after reset - Average session time is 0 after reset
Test Category:	Unit Test
Requirement:	3.1.11 Parental Controls
Automation	Yes - JUnit Test

Test Case Name:	Test Action Cooldowns
Test Case Description:	Verifies that action cooldowns work correctly
Test Prerequisites:	A Game object
Test Steps	1. Set cooldown for "vet" action 2. Verify "vet" action is on cooldown 3. Verify remaining cooldown time for "vet" is greater than 0 4. Set cooldown for "play" action 5. Verify "play" action is on cooldown 6. Verify remaining cooldown time for "play" is greater than 0
Expected Results:	- "vet" action is on cooldown after setting cooldown - Remaining cooldown time for "vet" is greater than 0 - "play" action is on cooldown after setting cooldown - Remaining cooldown time for "play" is greater than 0

Test Category:	Unit Test
Requirement:	3.1.7 Commands
Automation	Yes - JUnit Test

Pet.java Test Plan

Test Case Name:	Test Pet Initialization
Test Case Description:	Verifies that a pet is initialized with the correct default values
Test Prerequisites:	None
Test Steps	1. Create a new Dog pet named "Rex" 2. Verify the pet's name is "Rex" 3. Verify the pet's initial health is 100 4. Verify the pet's initial sleep is 100 5. Verify the pet's initial hunger is 100 6. Verify the pet's initial happiness is 100 7. Verify the pet's initial space readiness is 0 8. Verify the pet's initial state is NORMAL
Expected Results:	- The pet's name is "Rex" - The pet's health is 100 - The pet's sleep is 100 - The pet's hunger is 100 - The pet's happiness is 100 - The pet's space readiness is 0 - The pet's state is NORMAL
Test Category:	Unit Test
Requirement:	3.1.6 Vital Statistics & Rules
Automation	Yes - JUnit Test

Test Case Name:	Test Pet State Changes
Test Case Description:	Verifies that a pet's state changes correctly based on its vital statistics

Test Prerequisites:	None
Test Steps	1. Set the dog's health to 0 and verify its state changes to DEAD 2. Create a new dog to reset state 3. Set the dog's sleep to 0 and verify its state changes to SLEEPING 4. Set the dog's sleep to 100 and update state, verify it returns to NORMAL 5. Set the dog's happiness to 0 and verify its state changes to ANGRY 6. Create a new dog to reset state 7. Set the dog's hunger to 0 and verify its state changes to HUNGRY
Expected Results:	- When health is 0, state is DEAD - When sleep is 0, state is SLEEPING - When sleep returns to 100, state returns to NORMAL - When happiness is 0, state is ANGRY - When hunger is 0, state is HUNGRY
Test Category:	Unit Test
Requirement:	3.1.6 Vital Statistics & Rules
Automation	Yes - JUnit Test

Test Case Name:	Test Pet Actions
Test Case Description:	Verifies that pet actions correctly affect the pet's vital statistics
Test Prerequisites:	None
Test Steps	1. Set the dog's hunger to 50 and feed it kibble, verify hunger increases 2. Set the dog's happiness to 50 and play with it, verify happiness increases 3. Set the dog's happiness to 50 and give it a ball gift, verify happiness increases

	4. Set the dog's health to 50 and exercise it, verify health increases 5. Set the dog's health to 20 and take it to the vet, verify health increases
Expected Results:	- After eating kibble, hunger is greater than 50 - After playing, happiness is greater than 50 - After receiving a ball gift, happiness is greater than 50 - After exercising, health is greater than 50 - After visiting the vet, health is greater than 20
Test Category:	Unit Test
Requirement:	3.1.7 Commands
Automation	Yes - JUnit Test

Test Case Name:	Test Progress Bar Update
Test Case Description:	Verifies that the pet's vital statistics decrease over time
Test Prerequisites:	None
Test Steps	1. Set the dog's hunger, happiness, and sleep to 100 2. Update the progress bars 3. Verify that hunger, happiness, and sleep all decrease
Expected Results:	- After updating progress bars, hunger is less than 100 - After updating progress bars, happiness is less than 100 - After updating progress bars, sleep is less than 100
Test Category:	Unit Test
Requirement:	3.1.6 Vital Statistics & Rules

Automation	Yes - JUnit Test

Test Case Name:	Test Space Mission
Test Case Description:	Verifies that the pet can go on a space mission and it affects its statistics
Test Prerequisites:	None
Test Steps	1. Set the dog's space readiness to 100 2. Record initial health, hunger, happiness, and sleep 3. Send the dog on a space mission 4. Verify the mission was successful 5. Verify space readiness is reset to 0 6. Verify health, hunger, happiness, and sleep all decrease 7. Verify the mission count increases
Expected Results:	- The goOnSpaceMission method returns true - Space readiness is reset to 0 - Health, hunger, happiness, and sleep all decrease from their initial values - Total mission count is 1
Test Category:	Unit Test
Requirement:	3.1.13 One Extra Functional Requirement
Automation	Yes - JUnit Test

Food.java Test Plan

Test Case Name:	Test Food Creation
Test Case Description:	Verifies that food items are created with the correct type and fullness value
Test Prerequisites:	None
Test Steps	1. Create a new Food object "Kibble" 2. Verify its type is KIBBLE

	3. Verify its fullness value is 10 4. Create a new Food object "Apple" 5. Verify its type is APPLE 6. Verify its fullness value is 20
Expected Results:	- Kibble food has type KIBBLE - Kibble food has fullness value 10 - Apple food has type APPLE - Apple food has fullness value 20
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit Test

Test Case Name:	Test All Food Types
Test Case Description:	Verifies that all food types are created with the correct fullness values
Test Prerequisites:	None
Test Steps	1. Create each type of food (Kibble, Apple, Cheese, Bread, Icecream, Chicken) 2. Verify the fullness value for each food type
Expected Results:	- Kibble has fullness value 10 - Apple has fullness value 20 - Cheese has fullness value 15 - Bread has fullness value 30 - Icecream has fullness value 5 - Chicken has fullness value 30
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit Test

Item.java Test Plan

Test Case Name:	Test Item Basic Properties

Test Case Description:	Verifies that items have the correct basic properties (name and description)
Test Prerequisites:	None
Test Steps	1. Create a new Food object "Kibble" 2. Verify its name is "Kibble" 3. Verify its description is appropriate 4. Create a new Toy object "Ball" 5. Verify its name is "Ball" 6. Verify its description is appropriate
Expected Results:	- Food item has name "Kibble" - Food item has appropriate description - Toy item has name "Ball" - Toy item has appropriate description
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit Test

Toy.java Test Plan

Test Case Name:	Test Toy Creation
Test Case Description:	Verifies that toy items are created with the correct type and fun value
Test Prerequisites:	None
Test Steps	1. Create a new Toy object "Ball" 2. Verify its type is BALL 3. Verify its fun value is 30 4. Create a new Toy object "Rocket" 5. Verify its type is ROCKETTOY
Expected Results:	- Ball toy has type BALL - Ball toy has fun value 30 - Rocket toy has type ROCKETTOY - Rocket toy has fun value 25
Test Category:	Unit Test

Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit Test

Test Case Name:	Test All Toy Types
Test Case Description:	Verifies that all toy types are created with the correct fun values
Test Prerequisites:	None
Test Steps	1. Create each type of toy (Ball, Rocket, Frisbee, Alien, StarPlush) 2. Verify the fun value for each toy type
Expected Results:	- Ball has fun value 30 - Rocket has fun value 25 - Frisbee has fun value 10 - Alien has fun value 15 - StarPlush has fun value 20
Test Category:	Unit Test
Requirement:	3.1.8 Player Inventory
Automation	Yes - JUnit Test

Integration Testing

Integration Strategy: Bottom-Up Integration
The bottom-up integration approach will be used, where we first test the lowest level components (e.g., items, pets) in isolation, and then progressively integrate and test higher-level components (e.g., inventory, player, game) that depend on them.

Test Case Name:	**Pet and Item Integration**
Test Case Description:	Verify that Pet class correctly interacts with Food and Toy items
Test Steps:	1. Create a Pet object 2. Create a Food item 3.

	Create a Toy item 4. Have the Pet use the Food item 5. Have the Pet use the Toy item
Pre-Requisites:	Java development environment with all required classes compiled
Expected Results:	1. Pet's hunger increases when food is used 2. Pet's happiness increases when toy is used
Test Category:	Integration Test
Requirement:	3.1.7 Commands
Automation:	Yes - JUnit
Date Run:	March 20th
Pass/Fail:	Pass
Test Results:	Passed case
Remarks:	Might need to modify description fetching

Test Case Name:	**INT-02: Player and Pet Integration**
Test Case Description:	Verify that Player can correctly control a Pet
Test Steps:	1. Create a Player with a Pet 2. Add items to Player's inventory 3. Have Player feed the Pet 4. Have Player give a gift to the Pet 5. Have Player send Pet to bed
Pre-Requisites:	Java development environment with all required classes compiled
Expected Results:	1. Pet's hunger increases when fed 2. Pet's happiness increases when given gift 3. Pet enters sleeping state when sent to bed
Test Category:	Integration Test
Requirement:	3.1.7 Commands
Automation:	Yes - JUnit
Date Run:	March 23rd
Pass/Fail:	Pass

Test Results:	All cases passed
Remarks:	Looks good for now

Test Case Name:	**INT-03: Game and Player Integration**
Test Case Description:	Verify that Game class correctly manages Player and Pet objects
Test Steps:	1. Create a Game object 2. Create a new Pet through the Game 3. Issue commands through the Game to affect the Pet 4. Save and load the Pet
Pre-Requisites:	Temporary directory for save files
Expected Results:	1. Game creates Pet correctly 2. Commands affect Pet through Game 3. Pet state is preserved through save/load
Test Category:	Integration Test
Requirement:	3.1.5 Save/Load Game State
Automation:	Yes - JUnit
Date Run:	March 26th
Pass/Fail:	Pass
Test Results:	Game and Player working for now
Remarks:	will need to integrate with parental controls

Test Case Name:	**INT-04: Game Timer Integration**
Test Case Description:	Verify that game timer correctly updates Pet statistics over time
Test Steps:	1. Create a Game object 2. Create a new Pet 3. Start the Game timer 4. Wait for a short period 5. Stop the Game timer 6. Check if Pet statistics have decreased
Pre-Requisites:	Java development environment with all required classes compiled

Expected Results:	Pet statistics (hunger, sleep, happiness) decrease over time
Test Category:	Integration Test
Requirement:	3.1.6 Vital Statistics & Rules
Automation:	Yes - JUnit
Date Run:	March 26th
Pass/Fail:	Pass
Test Results:	Timer successfully updates stats every 30 seconds
Remarks:	need to integrate with parental controls

Test Case Name:	**INT-05: Save/Load with File System**
Test Case Description:	Verify that Pet data can be saved to and loaded from the file system
Test Steps:	1. Create a Pet with specific statistics 2. Save the Pet to a file 3. Clear the Pet object 4. Load the Pet from the file
Pre-Requisites:	Temporary directory with write permissions
Expected Results:	1. Pet data is correctly saved to file 2. Pet data is correctly loaded from file 3. Pet statistics match pre-save values
Test Category:	Integration Test
Requirement:	3.1.5 Save/Load Game State
Automation:	Yes - JUnit
Date Run:	March 26th
Pass/Fail:	Pass
Test Results:	Persistence system working
Remarks:	Had to switch from JSON file system

Validation Testing

Test Case Name:	**VAL-01: Pet Type Selection**
Test Case Description:	Verify that the game allows selection of at least 3 different pet types
Test Steps:	1. Launch the application 2. Navigate to New Game 3. Check available pet types
Pre-Requisites:	Application installed and running
Expected Results:	At least 3 different pet types are available for selection
Test Category:	Validation Test
Requirement:	3.1.4 New Game & Pet Selection
Automation:	No - Manual
Date Run:	March 22nd
Pass/Fail:	Pass
Test Results:	Able to select different pet types
Remarks:	Choose specific animals later

Test Case Name:	**VAL-02: Pet States Visualization**
Test Case Description:	Verify that pet states are visually indicated to the player
Test Steps:	1. Create a pet 2. Manipulate stats to change pet states (hungry, angry, etc.) 3. Observe visual indicators for each state
Pre-Requisites:	Application installed and running
Expected Results:	Visual indicators (sprites, icons, text) clearly show the pet's current state
Test Category:	Validation Test
Requirement:	3.1.6 Vital Statistics & Rules
Automation:	No - Manual

Date Run:	March 23rd
Pass/Fail:	Pass
Test Results:	Sprite imagine routing successfully changing based on pet state
Remarks:	Replace placeholder PNGs with actual sprites later

Test Case Name:	**VAL-03: Command Availability**
Test Case Description:	Verify that commands are only available based on pet state
Test Steps:	1. Create a pet 2. Change pet to various states (normal, angry, sleeping, etc.) 3. Check which commands are available in each state
Pre-Requisites:	Application installed and running
Expected Results:	Commands follow availability rules in section 3.1.7
Test Category:	Validation Test
Requirement:	3.1.7 Commands
Automation:	No - Manual
Date Run:	March 29th
Pass/Fail:	Fail
Test Results:	Pet state is bugging, need to fix
Remarks:	There is either a mismatch in logic or responsibility that is not getting detected my the GamePanel.java GUI file.

Test Case Name:	**VAL-04: Parental Controls**
Test Case Description:	Verify that parental controls restrict gameplay during unauthorized hours
Test Steps:	1. Set parental controls to restrict play between specific hours 2. Attempt to play

	during restricted hours 3. Attempt to play during allowed hours
Pre-Requisites:	Application installed and running
Expected Results:	1. Game prevents play during restricted hours 2. Game allows play during allowed hours
Test Category:	Validation Test
Requirement:	3.1.11.1 Parental Limitations
Automation:	No - Manual
Date Run:	March 25th
Pass/Fail:	Pass
Test Results:	Time restriction is working
Remarks:	Uses built-in Game Timer system

Test Case Name:	**VAL-05: Pet Revival**
Test Case Description:	Verify that parents can revive dead pets
Test Steps:	1. Create a pet 2. Let the pet die 3. Access parental controls 4. Enter password 5. Revive the pet
Pre-Requisites:	Application installed with parent password set
Expected Results:	1. Pet is successfully revived 2. Pet statistics are reset to maximum values
Test Category:	Validation Test
Requirement:	3.1.11.3 Revive Pet
Automation:	No - Manual
Date Run:	March 26th
Pass/Fail:	Fail
Test Results:	Game shows “Pet revived” message but pet stats do not change in the saved file

Remarks:	Come back to this later - not a priority

System Testing

Test Case Name:	**SYS-01: Windows Compatibility**
Test Case Description:	Verify that the application runs correctly on Windows 11
Test Steps:	1. Install the application on a Windows 11 system 2. Launch the application 3. Test all major functions
Pre-Requisites:	Windows 11 system with Java 23+ installed
Expected Results:	Application runs without errors and all features work correctly
Test Category:	System Test
Requirement:	3.2.13
Automation:	No - Manual
Date Run:	March 30th
Pass/Fail:	Pass
Test Results:	Works on Windows and MacOS

Test Case Name:	**SYS-02: Complete Game Lifecycle**
Test Case Description:	Verify a complete game lifecycle from creation to pet death
Test Steps:	1. Create a new pet 2. Play with the pet through multiple sessions 3. Save and load the game between sessions 4. Allow pet to die through neglect 5. Revive the pet through parental controls
Pre-Requisites:	Application installed and running
Expected Results:	Each stage of the pet lifecycle functions correctly from creation to death and revival

Test Category:	System Test
Requirement:	Multiple (3.1.4, 3.1.5, 3.1.6, 3.1.11.3)
Automation:	No - Manual
Date Run:	April 2nd
Pass/Fail:	Pass
Test Results:	Everything is working in the pet life cycle!

Test Case Name:	**SYS-03: GUI Responsiveness**
Test Case Description:	Verify that the GUI remains responsive during gameplay
Test Steps:	1. Launch the application 2. Navigate through all screens 3. Perform rapid interactions with pet 4. Run game for extended period
Pre-Requisites:	Application installed and running
Expected Results:	GUI remains responsive with no noticeable lag or freezing
Test Category:	System Test
Requirement:	3.2.17
Automation:	No - Manual
Date Run:	April 1st
Pass/Fail:	Pass
Test Results:	All GUI elements are responsive
Remarks:	May need to update once layout is fixed

Test Case Name:	**SYS-04: Keyboard and Mouse Controls**
Test Case Description:	Verify that both keyboard shortcuts and mouse controls work
Test Steps:	1. Launch the application 2. Test all functions

	using mouse 3. Test all functions using keyboard shortcuts
Pre-Requisites:	Application installed and running
Expected Results:	Both input methods work correctly for all functions
Test Category:	System Test
Requirement:	3.1.1 User Interface
Automation:	No - Manual
Date Run:	April 2nd
Pass/Fail:	Pass
Test Results:	Keyboard shortcuts for commands all working

Summary

This document outlines the testing procedures for AstroPets, ensuring the game meets all requirements. The test plan encompasses unit, integration, validation, and system testing with structured test cases. Future updates may include refinements based on test outcomes.

By maintaining a rigorous testing methodology, we ensure that AstroPets delivers a seamless and engaging experience for players. The structured test cases and thorough validation process allow us to detect and resolve potential issues early in the development cycle, leading to a high-quality final product. Moving forward, continuous testing and iterative refinements will help improve the game's performance, stability, and user satisfaction.