Table of Content

Group 58 Testing Documentation 1

Introduction 3

Summary 4

Test Plan 5

Integration Testing 6

System testing 14

Unit Testing 15

Table of Contents 15

1. VirtualPetTest.java 15
2. SaveLoadManger.java 15
3. MusicPlayer.java 16
4. LoadGameScreen.java 16
5. Inventory .java 17
6. Player.java 17

Validation Testing 19

Project Management 23

List of Team Minutes 23

Meeting Minutes - TA-group Meeting Mar. 19th 24

Meeting Minutes - Team Meeting Mar. 14th 25

Meeting Minutes - Team Meeting Mar. 21st 26

Meeting Minutes - Team Meeting Mar. 28th 27

Meeting Minutes - Team Meeting Mar. 29th 28

Meeting Minutes - Team Meeting Mar. 30th 29

Evidence of Issues Using 30

Implementation and Testing

Last edited by **HAOXUAN SUO** 1 hour ago

Group58 Testing Document

Table of Content

- Introduction
- Summary
- Test Plan

Task / Part	Contributors	Description / Notes
The name of the task or part of this milestone that was completed. For example "Introduction" or "Video Demo".	A list of team members who contributed to this component.	A short description of the component if not clear from *name* given in "Taks/Part". Also add notes here to provide more details about who did what (e.g. "Eve edited the video for the Video Demo.").
VirtualPet Class	Jack	Completed VirtualPet Class and Unit testing.
Game Play Screen	Jack	Completed the PlayGameScreen and implemented pet interaction function.
Pet Selection Screen	Jack	Completed the selectPetScreen.
Main Menu Screen	Jack	Completed the main Menu Screen.
Save&Load Manager Class	Yu Li	Completed the Save&Load Manager.
Inventory Class	Jinke Li	Completed the Inventory Class.
Game state Class	Yu Li	Implemented the game state management system to track overall game progress and status.
Load Game Screen	Angel Zhang	Designed and implemented the screen for loading saved game files.
Main Class	Jack, Haoxuan Suo	Created the main application class that initializes and runs the game.
Music Player Class	Angel Zhang	Developed the audio system for background music and sound effects.
Parental Controls Screen	Haoxuan Suo, Jinke Li	Implemented the screen for setting up parental controls and restrictions.
Pet Type Class	Jack	Created the class defining different types of virtual pets and their unique attributes.
Player Class	Haoxuan Suo	Implemented the player Password and progression tracking system.
Runtime Type Adapter Screen	Yu Li	Developed the system for handling different data types during runtime.
Setting Screen	Jinke Li	Created the screen for adjusting game settings and preferences.
Tutorial Screen	Yu Li	Designed and implemented the interactive tutorial for new players.

Introduction	Angel Zhang	Completed the Introduction.
Summary	Jack	Completed the Summary.
Integration Testing	Jinke Li,Yu Li,Haoxuan Suo,Jack,Angel Zhang	Completed the Integration Testing.
Unit Testing	Jinke Li,Yu Li,Haoxuan Suo,Jack,Angel Zhang	Completed the Unit Testing.
Team Minutes	Yu Li	Completed the Team Minutes.
README File	Jinke Li	

Introduction

Last edited by **Sze Wing Angel Zhang** 2 days ago

Overview

The Virtual Pet Game is a simulation game developed in Java where the player takes care of virtual pets by managing their health, happiness, sleep and hunger. The focus of this document is to verify the functionality and reliability of the software system through implementation and testing. It needs to ensure that all project requirements, both functional and non-functional, are realized as expected. This document helps us to identify and resolve potential defects before final delivery, improve the quality of the code, and have confidence in the correctness of the system.

Objectives

This document is a structured plan for the testing phase of the Virtual Pet Game.

The main objectives are:

- Ensure that all implemented functionality conforms to the system specifications.
- Verify that all user interactions and pet-related mechanisms (feeding, playing, sleeping, treating) conform to the design requirements.
- Verify that all game screens respond and function properly under various scenarios.
- Apply a variety of testing techniques including unit testing, integration testing, validation testing, and system testing.

References

Course Instructor. (2025). CS2212B W25 Group Project Specification. Western University.

Group 58. (2025). Group 58 Requirements Documentation. Western University.

Group 58. (2025). Group 58 Design Documentation. Western University.

Summary

Last edited by **Yu Li** 2 days ago

Summary

This document outlines the implementation and testing plan for our virtual pet simulation software. We employed a modular design where different components handle specific tasks, such as gameplay logic, file saving/loading, GUI interactions, and user control. Our testing strategy includes comprehensive unit tests for core logic and selective manual testing for GUI components.

Below is a reference table of key terms, classes, and their descriptions:

Term/Class	Description
`VirtualPet`	Represents the core pet object, including states (hunger, sleep, happiness) and actions.
`Player`	Stores and tracks player-specific data.
`Inventory`	Manages food and gift items the player can use.
`GameState`	Encapsulates the current state of the game, including player, pet, and inventory.
`SaveLoadManager`	Handles saving and loading game data using JSON (via Gson).
`RuntimeTypeAdapterFactory`	Supports polymorphic deserialization of inventory items.
`MusicPlayer`	Controls background music playback and mute functionality.
`MainMenuScreen`	Displays the main menu of the game.
`GamePlayScreen`	Main game interaction screen where the player cares for their pet.
`TutorialScreen`	Shows instructions for how to play the game.
`LoadGameScreen`	Allows the user to load a previously saved game slot.
`SettingScreen`	Contains parental control and sound settings.
`ParentalControlsScreen`	Restricts certain in-game actions for child users.
`SelectPetScreen`	Screen where the user selects their starting pet.
`PetType`	Enum representing different types of pets (Sheep, Duck, Dog).
`InventoryScreen`	GUI screen to view and interact with player's inventory.

Test Plan

Last edited by **HAOXUAN SUO** 59 minutes ago

1. Table of Content

- Integration Testing
- System Testing
- Unit Testing
- Validation Testing

This section discusses the detailed test plan for our software system. This plan indicates both the test cases that we are using to assess the quality of our software.

- **Unit Testing:** Unit testing concentrates on each unit (for example, component, class, etc.) of the software as implemented in source code.
- **Integration Testing:** The focus of integration testing is on the coming together and connecting of the above units, as reflected in the software's overall design and architecture.
- **Validation Testing:** In this testing, requirements established as part of requirements modelling are validated against the software has been constructed.
- **System testing:** Here, the software and other system elements are tested as a whole.

Integration Testing

Last edited by **HAOXUAN SUO** 1 hour ago

Table of Content

- Table of Content
  - Overall Integration Testing Approach
  - Test Case 1: Inventory Feeding Functionality Integration
  - Test Case 2: Parental Controls Time Restriction & Statistics Integration
  - Test Case 3: Pet Revival via Save/Load Integration
  - Test Case 4: Music Playback and Volume Control Integration
  - Test Case 5: New Pet Creation and Game Start Integration
  - Test Case 6: GamePlayScreen VirtualPet Functions Integration

Overall Integration Testing Approach

The project uses a **Top-Down Integration Testing** strategy. This means that we start testing from the top-level GUI screens (e.g., InventoryScreen, ParentalControlsScreen, SelectPetScreen, SettingScreen, etc.) and then validate that these screens correctly invoke the underlying logic modules (such as VirtualPet, Inventory, GameState, SaveLoadManager, and Player).

- For interactions between the GUI and business logic, we use the actual components in our tests.
- If any lower-level components (e.g., those that perform file I/O or time-dependent functions) are not fully reliable, Stubs may be used to simulate fixed responses.
- Most of these tests are conducted manually (GUI-based tests) while some core logic can also be validated via automated JUnit tests.

Test Case 1: Inventory Feeding Functionality Integration

Test Case Name:	**InventoryScreen-Feeding**
Test Case Description:	Test InventoryScreen using the “use” operation to feed the pet using the FoodItem, and verify that the operation updates the fullness attribute of the pet and reduces the quantity of the item in stock accordingly.
Test Steps:	1. Initialize a VirtualPet object and set fullness = 30 (not full). 2. Initialize an Inventory object and add a FoodItem (e.g. name “Bread”, quantity 8, increase fullness by 10). 3. Display the above Inventory and Pet via the InventoryScreen. 4. In the InventoryScreen, locate the FoodItem card and tap the “use” button. 5. In the confirmation dialog that appears, click on the “Confirm” button. 6. Observe the change in the pet status (fullness value) and the inventory list displayed on the screen.
Pre-Requisites:	1. VirtualPet, Inventory, and FoodItem have passed the unit test; 2. The InventoryScreen interface component is normal.
Expected Results:	1. The fullness attribute of Pet is increased from 30 to 40 (30 + 10), but not more than the maximum value; 2. The quantity of FoodItem “Bread” in stock is reduced by 1, and if the quantity is 0, the item is removed; 3. the screen displays the updated values and inventory status, and a success prompt pops up.
	Integration Test

Test Category:	
Requirement:	Make sure that the player's use of items through the inventory interface can correctly affect the status of the pet, and update the inventory information in real time.
Automation:	No (interface test performed manually)
Date Run:	2025-03-29
Pass/Fail:	Pass
Test Results:	The fullness attribute of Pet increased suceesfuly. The quantity of used food decreased by 1, and if the quantity goes to 0, it will be removed from inventory screen.
Remarks:	If the lower-level module is not yet complete, you can use Stub to return fixed data, but currently all modules have been implemented.

Test Case 2: Parental Controls Time Restriction & Statistics Integration

Test Case Name:	**ParentalControls-TimeStats**
Test Case Description:	This test verifies that the Parental Controls screen correctly enables time restrictions and displays/resets gameplay statistics, ensuring proper data exchange between the screen and the Player module.
Test Steps:	1. Access the ParentalControlsScreen and enter or set the parental password to access the control panel. 2. In the control panel, enable “Enable Time Limits” and set an allowed play time range that does not include the current system time (for example, if the current time is 11:00, set allowed range to 12:00–13:00). 3. Click the “Apply Settings” button and observe that the time range label and Allowed Status label update accordingly (e.g., “Play Restricted”). 4. Attempt to start the game (or invoke Player.startPlaying) while the current time is outside the allowed range, and verify that the system prevents gameplay, showing a warning with the allowed time range. 5. Check the “Game Statistics” area to review Total Play Time, Average Play Time, and Game Start Count. 6. Click the “Reset Statistics” button and confirm the reset in the prompt. 7. Verify that all statistics are reset to 0.
Pre-Requisites:	• The Player object has accumulated some gameplay statistics. • A parental password has been set. • The ParentalControlsScreen GUI is operational.
Expected Results:	• When time limits are applied, the Allowed Status label reflects that gameplay is restricted if the current time is outside the allowed window. • Attempts to start gameplay outside the allowed period are blocked and prompt a warning message. • The Game Statistics display shows the correct cumulative play data, and after a reset, all values return to zero.
Test Category:	Integration Test

Requirement:	The system must enforce parental control settings by restricting play outside specified hours and accurately track and reset gameplay statistics.
Automation:	No (manual testing required due to time-based and GUI interactions)
Date Run:	2025-03-29
Pass/Fail:	Pass
Test Results:	Successfully restricted gameplay outside allowed hours. When time limits were enabled and set to 12:00-13:00 while system time was 11:00, the status label correctly displayed "Status: Play Restricted" in red. Attempts to start the game resulted in a warning dialog showing "Current time does not allow gameplay! Allowed time: 12:00 - 13:00". Statistics display showed 0.33 hours total play time, 0.02 hours average play time, and 17 game starts before reset. After reset, all values returned to 0.00 hours and 0 starts.
Remarks:	The integration between Player's time restriction logic and ParentalControlsScreen's UI functions seamlessly. The system correctly loads and applies global settings from the persistence file. One edge case observed: when setting time ranges that cross midnight (e.g., 22:00-06:00), the system correctly identifies and displays "(Crosses midnight)" indicator and properly evaluates current time against this range.

Test Case 3: Pet Revival via Save/Load Integration

Test Case Name:	**RevivePet-SaveLoad**
Test Case Description:	This test verifies that the parental revival feature correctly restores a dead pet’s attributes via Player.revivePet(), using SaveLoadManager to update the save file and reflecting the changes in the pet’s state.
Test Steps:	1. Ensure one save slot (e.g., Save Slot 2) contains a pet in a dead state (e.g., Health = 0). 2. On the ParentalControlsScreen, in the “Pet Revival” section, select the corresponding save slot from the drop-down list. 3. Click the “Revive Pet” button and confirm the action in the prompt. 4. Observe the on-screen message (e.g., “Pet has been revived!”). 5. Load the game from the revived slot (using LoadGameScreen or directly starting GamePlayScreen) to verify that the pet’s attributes (Health, Sleep, Fullness, Happiness) are now restored (e.g., set to 50).
Pre-Requisites:	• There exists a save slot with a dead pet’s GameState. • The Player is in parent mode (isParent == true). • The SaveLoadManager functions correctly.
Expected Results:	• After revival, the GameState in the selected slot is updated: Health, Sleep, Fullness, and Happiness are set to maximum. • A success message confirms the revival operation. • When the revived save is loaded, the GamePlayScreen displays a pet that is no longer dead and shows the updated stats.
	Integration Test

Test Category:	
Requirement:	Parent users must be able to revive a dead pet, with the system updating the corresponding save file and restoring the pet's vital stats.
Automation:	No (manual confirmation through GUI and data verification
Date Run:	2025-03-29
Pass/Fail:	Pass
Test Results:	Successfully revived pet in Save Slot 2. After selecting the slot containing a dead pet (Health=0, Sleep=0, Fullness=0, Happiness=0) and clicking "Revive Pet", the success message "Pet has been revived!" was displayed. When the save was subsequently loaded, the GamePlayScreen showed all pet attributes restored to 50 points. The pet was visually responsive and interactive again, confirming the revival was complete.
Remarks:	The integration between Player.revivePet() and SaveLoadManager functions correctly. The GameState object is properly loaded, modified, and saved back to disk. We observed that the process maintains all other save data (such as pet name, type, score, etc.) while only updating the vital statistics. One recommendation for future improvement: add a visual confirmation of the pet's state in the ParentalControlsScreen before revival to verify which pets need revival.

Test Case 4: Music Playback and Volume Control Integration

Test Case Name:	**SettingScreen-Music**
Test Case Description:	This test verifies that the music settings in the SettingScreen correctly interact with MusicPlayer. It checks that switching music tracks, adjusting volume, and toggling mute affect the background music as expected.
Test Steps:	1. Open the SettingScreen and click the "Music" button to navigate to the music settings panel. 2. In the music settings panel, select a music track (e.g., "Music2") from the drop-down menu. 3. Adjust the volume slider to 50% and ensure the music toggle checkbox is checked (unmuted). 4. Click the "Save" button; verify that the selected track starts playing at approximately 50% volume. 5. Re-open the music settings panel, uncheck the music toggle checkbox (to mute), and click "Save." 6. Confirm that the background music becomes silent (no audible output).
Pre-Requisites:	• MusicPlayer and SettingScreen are fully implemented; • Audio files exist in the specified paths; • The system's sound output is working.
Expected Results:	• After saving, MusicPlayer.play() is called with the correct track (e.g., resources/music2.wav), and the volume is set to 50%. • When unmuted, the music plays continuously; when muted, the music becomes silent while still running in the background. • The SettingScreen reflects the current music settings without errors.

Test Category:	Integration Test
Requirement:	Users must be able to control background music playback, volume, and mute settings from the SettingScreen, and these controls should directly affect MusicPlayer's behavior.
Automation:	No (requires manual observation of audio output and GUI interaction)
Date Run:	2025-03-29
Pass/Fail:	Pass
Test Results:	• The selected track (resources/music2.wav) played successfully at approximately 50% volume. • Muting via the checkbox effectively silenced the background audio without interrupting playback. • Unmuting restored the volume as expected. • No runtime errors or audio glitches were encountered during the test. • UI components (volume slider, checkbox, and track selector) responded accurately and updated their state correctly after saving.
Remarks:	Test was performed on macOS with functional sound output. The audio behavior matched expectations during both mute and unmute actions. All SettingScreen interactions were smooth and visually stable.

Test Case 5: New Pet Creation and Game Start Integration

Test Case Name:	**SelectPet-NewGame**
Test Case Description:	This test validates that creating a new pet via the SelectPetScreen results in the pet being saved (using SaveLoadManager) and then loaded into the GamePlayScreen. It covers the entire flow from pet selection and naming to saving and launching gameplay.
Test Steps:	1. From the main menu, choose "Start a new game" to open the SelectPetScreen. 2. On the SelectPetScreen, click the "Select" button under a pet option (e.g., Dog). 3. In the "Give Your Pet a Name" dialog, enter a name (e.g., "Buddy") and click "Confirm." 4. The system determines whether to use a new save slot or replace the oldest save (using getSaveFileCounts() and findOldestSlot()) and saves the new pet's GameState. 5. The SelectPetScreen and naming dialog close, and the GamePlayScreen launches automatically. 6. Verify in the GamePlayScreen that the pet's name (e.g., "Buddy"), type, and initial stats (health, fullness, happiness, sleep, etc.) are correctly displayed.
Pre-Requisites:	• There is at least one available save slot, or if all three are in use, the system will overwrite the oldest slot; • The SelectPetScreen, SaveLoadManager, and GamePlayScreen are fully implemented and working. • VirtualPet objects initialize their stats using default values defined by their PetType.
Expected Results:	• A new GameState is created and saved to the appropriate slot (new or overwritten as needed). • The GamePlayScreen launches, displaying the new pet's information (name "Buddy," correct pet image, and full initial stats). • The entire transition from pet selection to game start occurs without error or unexpected pop-ups.

Test Category:	Integration Test
Requirement:	The game must support starting a new adventure by allowing the user to select and name a pet. The new pet's data should be saved (managing save slots automatically) and then loaded correctly into the gameplay screen.
Automation:	No (manual verification of the interactive flow)
Date Run:	2025-03-29
Pass/Fail:	Pass
Test Results:	New data file successfully saved to the appropriate slot. The state and image of new pet displays on the screen.
Remarks:	To verify slot replacement, repeat the test when three saves exist and check that the oldest save is overwritten.

Test Case 6: GamePlayScreen VirtualPet Functions Integration

Test Case Name:	**GamePlayScreen-VirtualPetFunctions**
Test Case Description:	This test verifies that the GamePlayScreen correctly triggers the underlying VirtualPet functions—such as sleep, play, feeding, taking to the vet, and exercise—when the corresponding buttons are pressed. It checks that each action appropriately updates the pet's state (e.g., health, sleep, fullness, happiness) and score, and that the GUI reflects these changes.
Test Steps:	1. **Initialize Test Environment:** · Launch the GamePlayScreen with a VirtualPet instance initialized in a valid (alive) state. · Ensure the pet's current state allows function calls (e.g., not dead or sleeping if testing play or feeding). 2. **Sleep Function Test:** · Click the "Sleep" button on the GamePlayScreen. · Observe that the VirtualPet.goSleep() function is invoked. · Verify that the pet's state changes to SLEEPING and that its sleep value begins increasing toward its maximum (or reaches the expected value after the sleep duration). 3. **Play Function Test:** · After the pet wakes up (or ensure it is not sleeping), click the "Play" button. · Confirm that VirtualPet.play(playValue) is called. · Verify that the pet's happiness increases appropriately and that the score is increased by the expected amount (subject to cooldown conditions). 4. **Take to Vet Function Test:**

	· Click the "Take to Vet" button. · Confirm that VirtualPet.takeToVet() is executed. · Verify that the pet's health is reset to its maximum value and that the score is decreased by 100. 5. **Exercise Function Test:** · Click the "Exercise" button. · Verify that VirtualPet.exercise(healthBoost) is invoked. · Check that the pet's health increases (up to its max) while its sleep and fullness decrease accordingly. 6. **Feeding or Gift Function Test** · Verify that Feed button only shows invetory with food, and Gift button only shows inventory with gifts. · Verify that VirtualPet.feed() or VirtualPet.giveGift() is invoked respectively, and that the corresponding attribute (fullness for food, happiness for gift) is updated as expected.
Pre-Requisites:	• The GamePlayScreen must be fully operational with an initialized VirtualPet instance in a non-dead state. • The VirtualPet's attributes (health, sleep, fullness, happiness) are set to known baseline values that allow for measurable changes. • All relevant buttons (Sleep, Play, Take to Vet, Exercise, Inventory) must be visible and enabled.
Expected Results:	• **Sleep:** Pressing "Sleep" triggers VirtualPet.goSleep(), changes the pet's state to SLEEPING, and increases the sleep attribute over time. • **Play:** Pressing "Play" invokes VirtualPet.play(), increasing the pet's happiness (subject to cooldown) and increasing the score accordingly. • **Take to Vet:** Pressing "Take to Vet" calls VirtualPet.takeToVet(), resetting the pet's health to maximum and deducting 100 points from the score. • **Exercise:** Pressing "Exercise" calls VirtualPet.exercise(), which should increase health (up to the maximum) while decreasing sleep and fullness as defined. • **Feeding/Gifting (if applicable):** Using a food or gift item results in a corresponding increase in fullness or happiness, and the inventory count is updated. • The GUI (progress bars, labels, score display) refreshes to reflect all changes in real time.
Test Category:	Integration Test
Requirement:	The GamePlayScreen must correctly invoke all VirtualPet methods when the corresponding actions are taken by the user, and the pet's status (including health, sleep, fullness, happiness, and score) must update to reflect these actions.
Automation:	No (manual testing is required to simulate button presses and observe GUI updates)
Date Run:	2025-03-29
Pass/Fail:	Pass
Test Results:	The GamePlayScreen correctly calls all the key VirtualPet functions when the corresponding buttons are pressed. The pet's attributes and score updated in real time on the screen, and all GUI components (progress bars, labels, score

	display) reflected the expected state changes. No errors or unexpected behavior were observed during the test.
Remarks:	Ensure that tests account for any cooldowns (e.g., play or vet actions) that might prevent immediate repeated invocation. If any VirtualPet method has conditions (e.g., pet must not be sleeping when playing), prepare the test environment accordingly..

System testing

Last edited by **Jinke Li** 1 day ago

Test Case: Full Game Cycle on Windows

Field	Description
Test Case Name	Full Game Cycle on Windows
Test Case Description	Ensure the complete application runs smoothly on Windows, including saving and resuming a pet's progress.
Test Steps	1. Launch the app 2. Select and name a pet 3. interact with pet 4. Press keyboard shortcuts and verify behavior 5. Adjust music volume 6. Save and exit 7. Reopen and load slot 8. Simulate pet dying and save 9. Open Parental Controls, enter password, and click "Revive" to restore the pet
Pre-Requisites	- Windows 11 - Java 24 installed - `gson-2.10.1.jar` in `src/` - Resources in `src/resources/`
Expected Results	- No crashes/errors - UI renders correctly - Music plays - Pet data restores from save - Keyboard shortcuts perform expected actions - Revive pet successfully
Test Category	System Test
Requirement	All core gameplay, UI, and save/load functionalities
Automation	No
Date Run	2025-03-31
Pass/Fail	Pass
Test Results	Game launched successfully. All features worked as intended.
Remarks	Tested on Windows 11. No bugs found.

Unit Testing

Last edited by **HAOXUAN SUO** 1 hour ago

Table of Contents

- Table of Contents
  - 1. VirtualPetTest.java
  - 2. SaveLoadManger.java
  - 3. MusicPlayer.java
  - 4. LoadGameScreen.java
  - 5. Inventory .java
  - 6. Player.java

1. VirtualPetTest.java

We follow a **white-box testing** approach, leveraging knowledge of the internal logic and state transitions of the pet object. Edge cases such as "health drops to zero", "feeding while sleeping", and "invalid/null input objects" are explicitly tested to ensure resilience and correctness of the system.

We have tested the following areas:

- **Basic attributes and setup**: We verify that the pet's name is correctly stored and returned, and that its core attributes (health, sleep, fullness, happiness) are properly initialized and can be updated.
- **State transitions**: The `updateState()` method is tested extensively to ensure the pet properly transitions between NORMAL, HUNGRY, SLEEPING, ANGRY, and DEAD states depending on its internal values. We verify these states are triggered at appropriate thresholds, such as when hunger reaches zero or when happiness falls too low.
- **Warnings and feedback**: Whenever an attribute reaches a critical level (e.g., health below 25%), the system is expected to issue a warning. Tests confirm that the correct warning messages are printed to the console.
- **Feeding behavior**: Feeding the pet using a `FoodItem` is tested under several conditions:
  - Fullness increases normally
  - Fullness does not exceed the maximum value
  - Feeding fails if the pet is DEAD or SLEEPING
  - Feeding with null food has no effect
  - The quantity of food items decreases as expected
- **Gift-giving behavior**: We ensure that giving gifts increases happiness, respects the maximum cap, reduces item quantity, and does not work when the pet is DEAD or SLEEPING. We also test the behavior when a gift is null or when the quantity is zero.
- **Exercise behavior**: The pet's `exercise()` function is tested to confirm that it increases health and decreases both sleep and fullness. It is also verified that exercise does not work when the pet is in invalid states (like DEAD or SLEEPING), and that attribute boundaries are respected.
- **Playing with the pet**: We test the `play()` method to ensure it increases happiness correctly, observes cooldown restrictions, and is blocked during sleep.
- **Vet visits**: The `takeToVet()` method is tested to confirm it restores health, respects cooldown timing, and does not work when the pet is DEAD, SLEEPING, or ANGRY.
- **Edge cases**: Additional tests handle edge conditions such as feeding with null items, issuing commands with invalid quantities, or interacting with the pet in inappropriate states.

2. SaveLoadManger.java

We adopt a white-box testing approach for `SaveLoadManager`, leveraging our understanding of how the system stores and retrieves serialized game data using JSON. The focus is on validating that data integrity is preserved across save and load operations, that file management functions behave as expected, and that corner cases such as missing or empty save slots are handled gracefully.

We have tested the following areas:

- **Saving and loading game state:**
  - We verify that a `GameState` object, once saved to a slot, can be correctly reloaded with all its attributes intact. This includes checking that the `VirtualPet` and `Inventory` data remain unchanged after serialization and deserialization.
    - Saving to slot 1, then loading and comparing fields
    - Ensuring `creationTime` is only set on the first save
    - Ensuring `lastSavedTime` updates on every save
- **Save file count logic:**
  - We confirm that `getSaveFileCounts()` accurately counts the number of existing save files in the `"saves/"` directory.
    - Count is 0 before any save
    - Count increases after each unique save
    - Count does not exceed 3 (based on max slot limit)
- **Oldest slot detection:**
  - We test the `findOldestSlot()` method to ensure it correctly identifies the slot with the earliest `creationTime`, which is used to replace old pets when all slots are full.
    - Three slots with manually assigned creation times
    - The method returns the correct slot index
    - Works even when one or more slots are empty
- **File edge cases:**
  - We simulate scenarios where `.json` files might be missing, empty, or partially corrupted.
    - `loadGame()` returns null for non-existing slot
    - `findOldestSlot()` skips invalid or unreadable files
    - `saveGame()` still creates a valid file even if directory was initially empty
- **Directory behavior:**
  - We verify that the `"saves/"` directory is created if missing, ensuring no file-related exceptions are thrown during save/load operations.

3. MusicPlayer.java

We used a white-box testing approach for the MusicPlayer class, focusing on verifying the correctness of its core audio control logic, including volume adjustment, mute/unmute functions, and default track loading.

We have tested the following areas:

- **Initial state validation:**
  The newly instantiated `MusicPlayer` was initialized with default values — volume set to 100, unmuted, and no track loaded (`getCurrentTrack()` returns null). These values were confirmed using the provided getter methods. The `getInstance()` method was used to verify singleton behavior.
- **Volume control:**
  The `setVolume()` method was tested with edge cases: 0, 100, and values below 0 and above 100. We confirmed the method correctly clamps inputs within the allowed range of 0–100.
- **Mute and unmute behavior:**
  We tested that `mute()` correctly sets the internal mute state and silences the player, while `unmute()` restores the previous volume. The `isMuted()` method correctly reflects the player's state after each operation.
- **Manual playback and runtime behaviour:**
  A manual test was included to simulate real-world usage. It starts music playback, then performs mute and unmute operations with short pauses in between. The goal is to verify the player responds correctly during runtime .
- **Stability on repeated calls to mute/unmute:**
  We tested repeated calls to `mute()` and `unmute()` to confirm the internal state remains consistent. This ensures that the volume state is preserved, not corrupted, and that muting/unmuting works reliably even under frequent toggling. The `stop()` method was also called before each test to ensure test isolation and reset playback state.

4. LoadGameScreen.java

- We adopted a white-box testing approach for the `LoadGameScreen` class, focusing on verifying save file loading, slot selection, and UI interaction behaviour. The goal was to ensure users can reliably view save slot information and launch gameplay from a chosen slot without error.
- **Displaying slot data:**
  - For each of the three slots, data is loaded using `SaveLoadManager.loadGame(i + 1)`.
    - If a slot contains valid data, the pet name, type, health, happiness, sleep, fullness, score, and last saved time are displayed.
    - Missing or zero values are replaced with `"--"`, and null pet types do not break image loading.
    - The image path is derived from the pet type: e.g., `"resources/dog/normal.png"` for a Dog pet.
- **Stat formatting:**
  - All numeric stats (health, happiness, sleep, fullness) are stored in the range 0–50.
    - A helper method converts these to percentage format using `(value / 50.0) * 100`, rounded with `Math.round()`.
    - Values like 25 become `50%`, 50 becomes `100%`, and 0/null becomes `"--"`.
- **Slot selection behavior:**
  - Each slot card contains a "Select" button.
    - When clicked, the `selectedSlot` is updated to that slot's index.
    - The selected card is highlighted with a thick pink border (`5px`), while other cards revert to a thin grey border.
    - We tested selecting each slot to ensure correct visual feedback and internal state update.
- **Confirm button logic:**
  - If no slot is selected (`selectedSlot == -1`), clicking "Confirm" shows a dialog prompting the user to choose a slot.
  - If a valid slot is selected:
    - A new `GamePlayScreen` is launched using the path `"saves/slotX.json"`.
    - A success dialog is displayed, and the `LoadGameScreen` window is closed via `dispose()`.
- **Home button behaviour:**
  - Clicking "Home" shows a confirmation message and opens `MainMenuScreen`, disposing the current frame.
- **Handling of missing or empty saves:**
  - When no data is found for a slot:
    - The panel shows only the slot number (e.g., "Slot 3") and blank space for layout consistency.
    - No crash or exception occurs; the game gracefully skips empty or unreadable slots.
- **Image loading and rendering:**
  - Pet images are loaded from disk using `loadAndResizeIcon()` and scaled to 64×64 with `Image.SCALE_SMOOTH`.
    - If the image file is missing, no icon is shown, but the program remains functional.
- **UI consistency:**
  - All card elements are center-aligned for uniform appearance.
- Vertical spacing is controlled using `Box.createVerticalStrut()` to ensure visual balance.

5. Inventory .java

- Item Addition: Verifies that new items can be added correctly, and that adding an existing item increases its quantity.
- Item Removal: Tests both partial and complete removal of items by name, ensuring quantity updates or item deletion behave as expected.
- Item Retrieval: Includes tests for retrieving items by name, checking item quantities, and fetching the full list of inventory items.
- Item Usage Effects: Confirms that FoodItem and GiftItem correctly apply their effects to the virtual pet (e.g., increasing fullness or happiness) and that item quantities decrease upon use.
- Empty Inventory Handling: Ensures proper behavior when the inventory is empty, such as returning null or an empty list where appropriate.

6. Player.java

The Player class is a key component of our parental control system, responsible for managing game time restrictions, tracking play sessions, and enforcing parental control settings. Our unit testing approach for this class focuses on verifying its core functionality through a series of targeted test cases.

The PlayerTest class implements multiple test methods, each designed to validate a specific aspect of the Player class. These tests systematically verify password management, time restriction controls, play session tracking, and various edge cases to ensure proper functionality in all scenarios.

Our testing strategy for the Player class includes:

- Testing password initialization, verification, and access control mechanisms
- Validating parental control settings activation and configuration
- Verifying time restriction enforcement both for standard time ranges and ranges that cross midnight
- Confirming accurate play time tracking and statistics generation
- Testing edge cases such as invalid time formats and boundary time values

The tests utilize JUnit's assertion methods to verify expected outcomes against actual results from the Player class. By isolating the Player class from other components, we can thoroughly test its internal logic and ensure it correctly implements all required functionality before integration with the broader system.

Validation Testing

Last edited by **Sze Wing Angel Zhang** 19 minutes ago

Table of Contents

- Overall Validation Testing Approach
- Test Case 1: Save & Load Game Consistency
- Test Case 2: Background Music Control via Settings
- Test Case 3: LoadGameScreen UI Data Display
- Test Case 4: Tutorial Screen Functionality
- Test Case 5: Gameplay Action Button Reactions
- Test Case 6: Main Menu Navigation

Overall Validation Testing Approach

To ensure that the final application aligns with the original system requirements, we adopted a black-box validation testing approach. The focus was on validating user-facing behaviors such as save/load functionality, background music controls, UI interactions, and pet behavior in response to various actions.

These validation tests were carried out manually through the GUI, with testers simulating typical gameplay scenarios and confirming expected outcomes through direct observation.

Since many of the requirements involve user interaction, visual/audio feedback, or in-game state transitions, most tests required human observation. The validation cases presented below are scenario-based, designed to reflect real user experiences. Each one ties directly to system requirements and uses actual game elements such as save data, pet attributes, and audio files.

Test Case 1: Save & Load Game Consistency

Test Case Name:	**Validation–SaveLoadConsistency**
Test Case Description:	Verifies that saved game data can be accurately loaded.
Test Steps:	1. Launch game and save to a slot. 2. Re-launch and load the slot. 3. Compare pet stats, score, etc.
Pre–Requisites:	• Game runs without error. • SaveLoadManager functional.
Expected Results:	• All pet attributes persist correctly after loading.
Test Category:	Validation Test
Requirement:	Game must allow accurate game state persistence.
Automation:	No – manual visual confirmation
Date Run:	2025–03–29
Pass/Fail:	Pass
Test Results:	Game state correctly restored from save file.
Remarks:	Consistent behavior across all save slots.

Test Case 2: Background Music Control via Settings

Test Case Name:	**Validation–MusicControlFromSettings**
Test Case Description:	Checks that SettingScreen controls MusicPlayer volume and mute states.
Test Steps:	1. Open SettingScreen. 2. Select a track and adjust volume. 3. Toggle mute checkbox. 4. Click Save.
Pre–Requisites:	• MusicPlayer properly linked to UI. • Sound files exist.
Expected Results:	• Correct track plays. • Mute/unmute reflects user input.
Test Category:	Validation Test
Requirement:	Music settings must apply in real time.
Automation:	No – requires audio confirmation
Date Run:	2025–03–29
Pass/Fail:	Pass
Test Results:	Volume and mute correctly updated through UI.
Remarks:	SettingScreen and MusicPlayer integration confirmed.

Test Case 3: LoadGameScreen UI Data Display

Test Case Name:	**Validation–LoadGameScreenDisplay**
Test Case Description:	Verifies correct slot information is shown on LoadGameScreen.
Test Steps:	1. Open LoadGameScreen. 2. Observe pet name, stats, time. 3. Select and confirm a slot.
Pre–Requisites:	• Valid save files exist.
Expected Results:	• Correct values displayed per slot. • Confirmation loads correct data.
Test Category:	Validation Test
Requirement:	Game must support choosing and loading saves.
Automation:	No – GUI interaction

Date Run:	2025–03–29
Pass/Fail:	Pass
Test Results:	Slot data displayed and loaded correctly.
Remarks:	Graceful handling of empty/missing save files.

Test Case 4: Tutorial Screen Functionality

Test Case Name:	**Validation–TutorialScreenNavigation**
Test Case Description:	Ensures tutorial content shows and allows return to menu.
Test Steps:	1. Open TutorialScreen. 2. Read content. 3. Click return/back.
Pre–Requisites:	• Screen accessible via menu.
Expected Results:	• Tutorial instructions displayed. • User can return without error.
Test Category:	Validation Test
Requirement:	Tutorial must be accessible.
Automation:	No
Date Run:	2025–03–29
Pass/Fail:	Pass
Test Results:	All instructions visible, return button works.
Remarks:	Tested for layout and content clarity.

Test Case 5: Gameplay Action Button Reactions

Test Case Name:	**Validation–GameplayInteractionButtons**
Test Case Description:	Validates that gameplay actions like feed/sleep/play work as expected.
Test Steps:	1. Enter GamePlayScreen. 2. Click action buttons. 3. Observe stat changes.
Pre–Requisites:	• Pet is loaded.
Expected Results:	• Pet stats update based on actions.

Test Category:	Validation Test
Requirement:	Game must support pet interaction.
Automation:	No
Date Run:	2025–03–29
Pass/Fail:	Pass
Test Results:	All buttons affect pet state as intended.
Remarks:	Confirmed stat change logic for each action.

Test Case 6: Main Menu Navigation

Test Case Name:	**Validation–MainMenuNavigation**
Test Case Description:	Confirms buttons on main menu lead to correct screens.
Test Steps:	1. Launch app. 2. Click every menu option.
Pre–Requisites:	• All pages are implemented.
Expected Results:	• Each button transitions to correct page.
Test Category:	Validation Test
Requirement:	User must access all pages from menu.
Automation:	No
Date Run:	2025–03–29
Pass/Fail:	Pass
Test Results:	Menu links functional and intuitive.
Remarks:	Screens load without lag or crash.

Team Minutes

Last edited by **Yu Li** 1 minute ago

List of Team Minutes

TA-Group Meetings

- Jan. 29th
- Feb. 12nd
- Mar. 5th
- Mar. 19th

Just Team Meetings

- Jan. 26th
- Feb. 2nd
- Feb. 6th
- Feb. 14th
- Feb. 28th
- Mar. 2nd
- Mar. 4th
- Mar. 6th
- Mar. 14th
- Mar. 21st
- Mar. 28th
- Mar. 29th
- Mar. 30th

TA Group Meeting Mar 19th

Last edited by **Yu Li** just now

Meeting Minutes - TA-Group Meeting Mar 19th

Attendance

- [x] Zhenkang Xu
- [x] Yu Li
- [x] Sze Wing Angel Zhang
- [x] Jinke Li
- [x] Haoxuan Suo

Time & Location

Location: Online

Date: Mar. 19th

Time: 18:30 - 18:50

Items Discussed

- Reviewed expectations and structure for the Implementation Documentation.
- Clarified which components of the project require detailed method-level explanations.
- Discussed examples of good documentation practices, including formatting and use of technical terms.
- Addressed common issues found in previous years' implementation documents, such as lack of clarity or missing diagrams.

Action Items

- None

Team Meeting Mar. 14th

Last edited by **Yu Li** 2 days ago

Meeting Minutes - Mar. 14th

Attendance

- [x] Zhenkang Xu
- [x] Yu Li
- [x] Sze Wing Angel Zhang
- [x] Jinke Li
- [x] Haoxuan Suo

Time & Locations

Location: Online
Date: Mar. 14th
Time: 20:00 -- 22:00

Items Discussed

- Discussed the expectations and structure of the upcoming assignment: **Implementation and Testing**.
- Clarified deliverables including coding and screen development for each major component of the system.
- Assigned each team member specific Java classes and UI screens to implement based on their strengths and availability.
- Agreed on following a modular development approach so that integration and testing can proceed smoothly.
- Emphasized the importance of writing clean, testable code and preparing for JUnit-based unit testing.

Action Items

- [x] **Bella** – Implement `saveGame` logic; build the **Tutorial** screen
- [x] **Jinke** – Implement `Inventory` logic; build the **Parental Control** screen
- [x] **Angel** – Implement `MusicPlayer` logic; build the **Load Game** screen
- [x] **Bruno** – Implement `Player` class logic; build the **GamePlay** screen
- [x] **Jack** – Implement `VirtualPet` logic; build the **Main Page** screen

Team Meeting Mar. 21st

Last edited by **Yu Li** 2 days ago

Meeting Minutes - Mar. 21st

Attendance

- [x] Zhenkang Xu
- [x] Yu Li
- [x] Sze Wing Angel Zhang
- [x] Jinke Li
- [x] Haoxuan Suo

Time & Locations

Location: Online
Date: Mar. 21st
Time: 20:00 -- 22:00

Items Discussed

- Reviewed and checked each other's implementation of the assigned Java classes and UI screens.
- Gave feedback and suggestions for improvement where needed (e.g., code clarity, naming conventions, missing methods).
- Made sure core functionality was working as expected and saved files behaved correctly.
- Confirmed all members had completed their assigned coding tasks.
- Started writing **JUnit unit tests** for key classes, such as `SaveLoadManager`, `Inventory`, and `VirtualPet`.
- Clarified how to isolate logic for testing and how to avoid testing UI directly.
- Decided to document unit testing coverage in the shared test plan.

Action Items

- [x] Each member continues writing **unit tests** for their own assigned classes.
- [x] Group members review and push test cases to Bitbucket.
- [x] Finalize unit testing section in the project documentation.

Team Meeting Mar. 28th

Last edited by **Yu Li** 2 days ago

Meeting Minutes - Mar. 28th

Attendance

- [x] Zhenkang Xu
- [x] Yu Li
- [x] Sze Wing Angel Zhang
- [x] Jinke Li
- [x] Haoxuan Suo

Time & Locations

Location: Online
Date: Mar. 28th
Time: 20:00 -- 22:00

Items Discussed

- Everyone continued progressing on their assigned **coding tasks** and **unit tests**.
- Completed writing **JavaDoc** comments for all major classes and public methods.
- Reviewed and cleaned up documentation for better readability and consistency.
- Identified and **fixed several minor bugs**, including logic errors and file-saving issues.
- Ensured proper version control updates (push/pull) were done for all changes.
- Discussed remaining polishing tasks and final testing steps before submission.

Action Items

- [x] Continue writing and reviewing unit tests to ensure full coverage.
- [x] Make final adjustments to code and UI as needed.
- [x] Review and finalize project documentation and JavaDocs.

Team Meeting Mar. 29th

Last edited by **Yu Li** 2 days ago

Meeting Minutes - Mar. 29th

Attendance

- [x] Zhenkang Xu
- [x] Yu Li
- [x] Sze Wing Angel Zhang
- [x] Jinke Li
- [x] Haoxuan Suo

Time & Locations

Location: Online
Date: Mar. 29th
Time: 20:00 -- 22:00

Items Discussed

- Debugged and fixed **logic issues in the music playing feature**, ensuring proper start/stop behavior.
- Clarified and finalized the implementation logic for **three save slot management**:
  - Save files are assigned dynamically.
  - The oldest save slot will be replaced when all three slots are taken.
- Tested the game flow involving selecting pets, naming them, saving, and replacing slots.
- Continued verifying code stability and maintaining consistent JavaDoc style.

Action Items

- [x] Finalize save slot replacement logic and validate through UI testing.
- [x] Test music feature across different screens to ensure smooth playback control.
- [x] Confirm all core features are integrated and working as expected.

Team Meeting Mar. 30th

Last edited by **Yu Li** 2 days ago

Meeting Minutes - Mar. 30th

Attendance

- [x] Zhenkang Xu
- [x] Yu Li
- [x] Sze Wing Angel Zhang
- [x] Jinke Li
- [x] Haoxuan Suo

Time & Locations

Location: Online
Date: Mar. 30th
Time: 20:00 -- 22:00

Items Discussed

- Final review of all implementation components and documentation.
- Ensured all **unit tests** and **JavaDocs** are complete and up to standard.
- Finalized the **README.md** with installation, feature, and usage instructions.
- Completed and reviewed the **demo video** walkthrough of gameplay and features.
- Checked all required files are organized and ready for **submission**, including:
  - Source code
  - Unit test results
  - Final report and test plan
  - Video demo
  - Executable

Action Items

- [x] Verify the completeness of JavaDocs across all classes
- [x] Finalize and proofread the README file
- [x] Review and upload the demo video
- [x] Organize and double-check submission folder
- [x] Make sure to submit all required components on time

Implementation and Testing

Edit

Open Issue created 1 week ago by **Yu Li**

👍 0 👎 0

Create merge request

Drag your designs here or click to upload.

Child items 12 Add

- Read Me File #28 — Closed
- Main Page Doc #27 — Closed
- Parental Control #25 `screen` — Closed
- Game Play #29 `screen` — Closed
- Load Game #24 `screen` — Closed
- Tutorial #23 `screen` — Closed
- main page #22 `screen` — Closed
- Virtual pet #15 `coding` — Closed
- player #14 `coding` — Closed
- music player #13 `coding` — Closed
- inventory #12 `coding` — Closed
- save game #30 `coding` — Closed

Linked items 0 Add

Link issues together to show that they're related. Learn more.

Activity

Sort or filter

- **Yu Li** changed due date to March 31, 2025 1 week ago
- **Yu Li** assigned to @yli3894 1 week ago
- **Yu Li** assigned to @gzhou54 and unassigned @yli3894 1 week ago
- **Yu Li** assigned to @yli3894 and unassigned @gzhou54 1 week ago
- **Yu Li** unassigned @yli3894 1 week ago
- **Yu Li** added `coding` `screen` labels 1 week ago
- **Yu Li** added #12 (closed) as child task 1 week ago
- **Yu Li** added #13 (closed) as child task 1 week ago
- **Yu Li** added #14 (closed) as child task 1 week ago
- **Yu Li** added #15 (closed) as child task 1 week ago
- **Yu Li** created branch `10-implementation-and-testing` to address this issue 1 week ago
- **Yu Li** added #22 (closed) as child task 1 week ago
- **Yu Li** added #23 (closed) as child task 1 week ago
- **Yu Li** added #24 (closed) as child task 1 week ago
- **Yu Li** added #25 (closed) as child task 1 week ago
- **Yu Li** created branch `10-implementation-and-testing-2` to address this issue 1 week ago
- **Yu Li** added #27 (closed) as child task 1 week ago
- **Yu Li** added #28 (closed) as child task 1 week ago
- **Yu Li** added #29 (closed) as child task 1 week ago
- **Yu Li** added #30 (closed) as child task 1 week ago

Normal text

Write a comment or drag your files here...

Switch to plain text editing

☐ Make this an internal note

Comment Close issue

Add a to-do item

0 Assignees Edit
None - assign yourself

Labels Edit
`coding` `screen`

Milestone Edit
None

Due date Edit
Mar 31, 2025 - remove due date

Time tracking
No estimate or time spent

Confidentiality
Confidentiality controls have moved to the issue actions menu (⋮) at the top of the page.

2 Participants

0 Assignees Edit
None - assign yourself

Labels Edit
`coding` `screen`

Milestone Edit
None

Due date Edit
Mar 31, 2025 - remove due date

Time tracking
No estimate or time spent

Confidentiality
Confidentiality controls have moved to the issue actions menu (⋮) at the top of the page.

2 Participants

0 Assignees Edit
None - assign yourself

Labels Edit
`coding` `screen`

Milestone Edit
None

Due date Edit
Mar 31, 2025 - remove due date

Time tracking
No estimate or time spent

Confidentiality
Confidentiality controls have moved to the issue actions menu (⋮) at the top of the page.

2 Participants

0 Assignees Edit
None - assign yourself

Labels Edit
`coding` `screen`

Milestone Edit
None

Due date Edit
Mar 31, 2025 - remove due date

Time tracking
No estimate or time spent

Confidentiality
Confidentiality controls have moved to the issue actions menu (⋮) at the top of the page.

0 Assignees Edit
None - assign yourself

Labels Edit
`coding` `screen`

Milestone Edit
None

Due date Edit
Mar 31, 2025 - remove due date

Time tracking
No estimate or time spent

Confidentiality
Confidentiality controls have moved to the issue actions menu (⋮) at the top of the page.

2 Participants