Virtual Pet Game
Testing Documentation

2.1. Main Page

Task	Contributors	Notes
Introduction	Aryan Baria | Mohammed Bayoumi | Maher Rammal	Mohammed worked on the overview, Aryan worked on the objectives, and Maher formatted the references.
Junit table explanations	Maher Rammal & Dilraj Deogan	Maher created test table explanations, and Dilraj fixed, edited and added some and implemented time and if pass/fail
README FILE	Rayan Amir & Dilraj Deogan	Rayan created the readme file, and Dilraj formatted it.
Summary	Maher Rammal	Wrote the summary and added in new terms
GUI interface designs		
Starting Screen, Tutorial, Password Implementation, Cooldown Feature	Maher Rammal	Created the design for the starting screen, and connected buttons to the respective windows. Also implemented password overlay functionality in the parental settings. Created the design for the full tutorial and connected all tutorial windows. Also implemented cooldown features and food/gift quantity feature
Parental Screen and Settings and Inventory	Dilraj Deogan	I implemented a tab-based inventory system that allows players to view, use, and manage food and gift items. The system supports random item generation, item quantity tracking, and loading inventory from a JSON or fallback method.
Gameplay Screen, Rewards, Saving and Loading system, Inventory	Aryan Baria	Created and implemented the gameplay and rewards screens. Created and implemented the saving to file and loading from file functionalities within the gameplay and starting screens. Implemented the feature where items can be found by completing actions and added to the inventory. Implemented the Gameplay screen where any changes to stats will be displayed using the stat bars and score box.

Aaction Screens, Updating Stats, and Score	Mohammed Bayoumi	Created the action GUI's such as feed, sleep, vet and minigame and implemented get methods in gameplayGUI to be able to get and update the pets' statistics. Also created the game logic to determine how actions impact pet attributes such as health, hunger, etc, when corresponding buttons are clicked.
Saving and Exiting screens	Rayan Amir	Created the Save and Exit GUI screens. Implemented a save functionality using SaveGameManager to write game progress to a file. Added confirmation dialogs and error messages to guide users through saving and exiting.
Implementing data		
Json files	Maher Rammal	Created the json file, and implemented all the json functionality/data to the guis
Junit testing		
JUnit tests	Dilraj Deogan	**Class Tested:** `InventoryScreen.java` Tested the inventory management system used in the game for adding, converting, and validating items. **Class Tested:** `GameplayGUI` Tested the core logic of the main game GUI, including stat tracking, score updates, and action cooldown behavior. **Class Tested:** `ParentalControlScreen` Created a test to ensure key parental control UI methods function and are callable without error.

2.2. Introduction	
Overview	Virtual pets are digital animals that simulate caring for a real pet by assigning users a range of responsibilities such as feeding, grooming, and interacting with their pets. Popular games such as Tamagotchi, Neopets, Nintendogs, and Bitzee are examples of virtual pets. The reason why they were popularized was their ability to engage users through simple, repetitive tasks that grow more rewarding over time as the pet's well-being reflects the user's care. Considering this, with the use of Java and other Computer science tools, our goal is to design and implement this game style. The player will have to take responsibility for the pet while encouraging the users to have constant engagement with the game. The game will also feature a reward system for completing mini-games and an emotional meter that will increase when a mini-game is completed. Although the main aim is to create an engaging and interactive game, there is an educational element where it teaches players simple life skills like taking responsibility and keeping a routine.
Objectives	● Develop a virtual pet game that is both structured and maintainable by applying object-oriented programming strategies. ● Implement requirements such as pet interactions, pet emotions, animations, feeding mechanics, and user experience. ● Implement our design into Java code successfully to capture our decisions into machine code. ● Create a graphical user interface that is both functional and user-friendly. ● Work successfully as a team prioritizing team chemistry and collaboration to push the project success further. ● Write efficient, clean, well-documented Java code that adheres to best practices. ● Reflect and work on decisions made throughout the project to enhance the gameplay experience.
References	● Servos, Daniel. CS2212B Group Project Specification. Version 1.1.0, Winter Session 2025. Owl Brightspace, 2025. ● Draw.io program ● Servos, Daniel. CS2212: Introduction to Software Engineering, Test resources. Quiz 1 resource. Winter Session 2025, Owl. ● Servos, Daniel. CS2212: Introduction to Software Engineering, Week 2 Unified Modeling Language. Winter Session 2025, Owl. ● **"MB."** *Cambridge Dictionary*, Cambridge University Press, https://dictionary.cambridge.org/dictionary/english/mb ● Interaction Design Foundation - IxDF. "What is User Experience (UX) Design?" Interaction Design Foundation - IxDF, 8 Feb. 2025, https://www.interaction-design.org/literature/topics/ux-design.

- W3Schools. “What is JSON?” W3Schools, https://www.w3schools.com/whatis/whatis_json.asp.
- GeeksforGeeks. “Introduction to JUnit 5.” GeeksforGeeks, 4 Jan. 2025, https://www.geeksforgeeks.org/introduction-to-junit-5/.
- GeeksforGeeks. “What is Graphical User Interface?” GeeksforGeeks, 29 Jan. 2024, https://www.geeksforgeeks.org/what-is-graphical-user-interface/.
- **"GitLab."** *TechTarget*, October 2020, https://www.techtarget.com/whatis/definition/GitLab.

New References being added:

- Balsamiq. Balsamiq Wireframes. Balsamiq, https://balsamiq.com/company/news/balsamiq-wireframes/.
- "Pet Sprite." The Spriters Resource, https://www.spriters-resource.com/.
- Granholm, Alva. Inspiration for Inventory design in mockup. ArtStation, https://alvagranholm.artstation.com/projects/PenDqZ.
- Netflix. Inspiration for Making Account design in mockup. Netflix, https://www.netflix.com/ca/.
- Jenkov. "Jackson ObjectMapper Tutorial." Jenkov Tutorials, https://jenkov.com/tutorials/java-json/jackson-objectmapper.html.
- "MVC Design Pattern." GeeksforGeeks, https://www.geeksforgeeks.org/mvc-design-pattern/.
- "Command Pattern." OODesign, https://www.oodesign.com/command-pattern.
- "Observer Pattern." SourceMaking, https://sourcemaking.com/design_patterns/observer.
- Servos, Daniel. CS2212: Introduction to Software Engineering, Test Resources. Quiz 2 Resource. Winter Session 2025, OWL.
- Servos, Daniel. CS2212: Introduction to Software Engineering, Week 4 UML. Winter Session 2025, OWL.
- Servos, Daniel. CS2212: Introduction to Software Engineering, Week 6 UX. Winter Session 2025, OWL.
- "Introduction to Java Swing." GeeksforGeeks, https://www.geeksforgeeks.org/introduction-to-java-swing/ .
- "Apache Maven Project." Maven, https://maven.apache.org/.
- "HTML/CSS Tutorial." Khan Academy,
- Servos, Daniel. CS2212: Introduction to Software Engineering, Week 8 Quality concepts. Winter Session 2025, OWL
- https://www.khanacademy.org/computing/computer-programming/html-css.

New References being added:

- Java Classes and Objects. GeeksforGeeks, https://www.geekster.in/articles/java-classes-and-object/#:~:text=A%20Java%20object%20is%20an,to%20initialize%20the%20object's%20state.
- JPanel | Java Swing Tutorial https://www.youtube.com/watch?v=4PfDdJ8GFHI&ab_channel=JavaCodeJunkie
- Adding images to JPanel https://www.youtube.com/watch?v=1OyYyVlziRM&ab_channel=BrianSeaver
- Java panels https://www.youtube.com/watch?v=dvzAuq-YDpM&ab_channel=BroCode
- Java actionListener https://www.geeksforgeeks.org/java-actionlistener-in-awt/
- Java MouseListener. Bro Code. https://www.youtube.com/watch?v=jptf1Wd_omw
- JButton default cursor. Stackoverflow. https://stackoverflow.com/questions/27194858/jbutton-default-cursor
- Freecodecamp, author tapas Adhikary, November 29, 2021, https://www.freecodecamp.org/news/what-is-json-a-json-file-example/
- Learn JSON in 10 Minutes, Web Dev Simplified, https://www.youtube.com/watch?v=iiADhChRriM
- Java: Reading From and Writing JSON Data to File, ChargeAhead, https://www.youtube.com/watch?v=9Xr-o_QWMeE
- Stackoverflow, Dynamically creating Swing GUI from JSON Schema, https://stackoverflow.com/questions/57989453/dynamically-creating-swing-gui-from-json-schema-using-metawidget
- How to create a hovering button in java or changing the color of the button in jav. Glensourcecode. https://www.youtube.com/watch?v=uOdqVJX3tcg
- GridLayout | Java Swing Tutorial for Beginners. Java Code Junkie. https://www.youtube.com/watch?v=impJtkTcQ94

	● JLabel Example Java | Java Programming Tutorial 2020. Professor Saad. https://www.youtube.com/watch?v=hyWNd1Y912s ● JLabel | Java Swing Tutorial for Beginners. Java Code Junkie. https://www.youtube.com/watch?v=PLvIyoWPfmM ● How to create an overlay window in Java? https://stackoverflow.com/questions/1768062/how-to-create-an-overlay-window-in-java ● How to link one jframe to another frame in Java Swing-Windows Builder - Intact Abode. Intact Abode. https://www.youtube.com/watch?v=5m0zzr98k50 ● JPasswordField | Java Swing Tutorial for Beginners. Java Code Junkie. https://www.youtube.com/watch?v=bMBsqqFHvJo ● Java Tutorial 35 GUI Set Background Color on JPanel. learn Debugtime. https://www.youtube.com/watch?v=FCCPtI_s-OI ● Java GUI: Full Course (FREE). Bro Code. https://www.youtube.com/watch?v=Kmgo00avvEw (Since this is a long video, more specifically, (00:12:20) labels, (00:29:45) panels, (01:08:27) FlowLayout, (01:15:53) GridLayout, (01:27:57) open new window, (01:50:03) textfields, (02:07:58) radio buttons, (02:18:20) combo boxes, (02:40:49) progress bar, (02:51:43) menubars, (03:06:10) select a file, (03:21:58), MouseListener, (04:11:58) 2D graphics) ● Servos, Daniel. CS2212: Introduction to Software Engineering, Test resources. Quiz 3 resource. Winter Session 2025, Owl. ● Fonts used by Google Fonts. ● Icons from Falticon. https://www.flaticon.com/free-icons/getter

2.3. Test plan

For integration testing, we are using Bottom-Up Integration. So in this case, we test the smaller components first, like updating stats or parsing cooldowns, and then connect them into larger screens like feed, sleep, and vet. We use mock versions of GameplayGUI as drivers to simulate interaction (actual components are already built).

For system testing, we made sure that at least one test runs the full game on Windows, including saving and loading files, opening all screens. So in this case, we're making sure all main features work from start to finish, feeding, sleeping, playing, using items, and saving/loading the game.

Unit Testing & Validation Testing

Test Case Name	**Test Password Overlay Unlock**
Test Case Description	This test ensures that entering the correct password into the overlay removes the blocking panel, and entering the wrong password keeps it visible.
Test Steps	1. Setting a known value for parentPassword ("2212"). 2. Simulating clicking the "PASSWORD" button. 3. Input the correct password and make sure that the overlay becomes invisible. 4. Repeat with an incorrect password and make sure that the overlay remains visible and an error dialog appears.
Pre-Requisites	Main game must be opened, and parental settings is clicked
Expected Results	Overlay is removed when the correct password is entered, and overlay remains and error dialog appears when the password is incorrect.

Test Category	Unit test
Requirement	The overlay panel in ParentalControlScreen is only dismissed when the correct password is provided
Automation	Junit
Date Run	2025-04-03
Pass/fail	Pass
Test result	Overlay removed successfully
Remarks	N/A

Test Case Name	**Test Inventory Quantity and Removal on Use**
Test Case Description	Validating that the quantity of an item is correctly tracked, decremented upon use, and that items with quantity 0 are removed from the list
Test Steps	1. Adding a food item for example "Fish" with quantity 1 to inventory 2. Checking that quantity decrements after each use: a. After first: quantity = 1 b. After second: item removed 3. Confirming correct UI update (item disappears after 2nd use). 4. Repeating the same steps for a gift item for example "Frisbee" with quantity 1
Pre-Requisites	Items must have quantity = 1 initially, and of course InventoryScreen must be running

Expected Results	Each use reduces quantity by 1, and when quantity reaches 0, item is removed from list, an UI is updated
Test Category	Unit testing
Requirement	Quantity tracking and removal logic works for all inventory items
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	Quantity for item is updated successfully after use
Remarks	N/A

Test Case Name	**Test Cooldown Timer After Using Item**
Test Case Description	Cooldown values for actions like "feed", "play", "sleep", and "takeToVet" are properly loaded from jsonFile.json and stored in the actionCooldowns map
Test Steps	1. Getting data from json file 2. Initializing GameplayGUI 3. Calling getActionCooldown("feed"), getActionCooldown("play"), etc. 4. Making sure the correct values are returned 5. Modifying the file or simulating an invalid file to ensure fallback values (default = 5) are used.
Pre-Requisites	The JSON file must be simulated and the class must be initialized with access to json/jsonFile.json.

Expected Results	Correct cooldown values returned from JSON, and the fallback values are returned when JSON is missing or malformed.
Test Category	Unit testing
Requirement	Cooldown values are parsed accurately and stored correctly in memory
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	Cooldown is applied when an action is done
Remarks	N/A

Test Case Name	**Test Time Restriction Selection and Confirmation**
Test Case Description	Ensuring that the user can select start and end times from the combo boxes and that clicking the "Confirm Restrictions" button displays a confirmation message in the parent screen.
Test Steps	1. Creating a mock ParentalControlScreen instance. 2. Opening a TimeRestrictionWindow with this parent. 3. Programmatically setting start time to for example "10:00 AM" and end time to "2:00 PM". 4. Simulating clicking the "Confirm Restrictions" button. 5. Displaying that the confirmation message on the parent screen is for example "Time Restrictions Saved: 10:00 AM - 2:00 PM". 6. Verifying the window closing and parent screen is re-enabled.
Pre-Requisites	The combo boxes should be populated with valid times.
Expected Results	Time selection is registered correctly and confirmation message is displayed on the parent screen. After that, TimeRestrictionWindow is disposed of, and the parent screen is re-enabled.
Test Category	Unit Test
Requirement	Time restrictions selected by the user are passed correctly to the parent screen and confirmed.
Automation	JUnit
Date Run	2025-04-03
Pass/fail	Pass
Test result	Time selection was registered correctly and displayed on the parental controls screen.

Remarks	N/A

Test Case Name	**Test Average Playtime Selection and Confirmation**
Test Case Description	Selecting a playtime value from the combo box and confirming it triggers the correct message on the parent screen and closes the window.
Test Steps	1. Creating a mock ParentalControlScreen instance. 2. Opening an AveragePlaytimeWindow with this parent. 3. Setting the combo box 4. Simulating clicking the "Confirm Playtime" button. 5. Displaying the confirmation message on the parent screen, for example something like"Average Playtime Saved: 3 hours". 6. Allowing the window to be disposed of and the parent screen is re-enabled.
Pre-Requisites	The combo box must contain valid playtime options.
Expected Results	The selected playtime value is retrieved correctly, and the parent screen shows a confirmation message with selected value. The window should also close and the parent screen becomes active again.
Test Category	Unit testing
Requirement	Selected average playtime is correctly passed and acknowledged by the parent screen.
Automation	Junit

Date Run	2025-04-03
Pass/Fail	Pass
Test result	Average playtime is set and displayed for the user
Remarks	N/A

Integration Testing & System Testing

Test Case Name	**Test Use of Food or Gift Item**
Test Case Description	Clicking the "Use" or "Give" button in the inventory properly applies the item effect, updates pet stats, removes the item if quantity reaches zero, and updates the UI accordingly.
Test Steps	1. Initializing InventoryScreen with sample items and a mock GameplayGUI. 2. Adding a food item ("Apple" with quantity 1) and a gift item ("Toy" with quantity 1). 3. Simulating clicking the "Use Food" or "Use Gift" button. 4. Selecting the available item and confirming usage. 5. Items are removed from inventory. 6. Updating the appropriate stats(hunger or happiness) increased on GameplayGUI.
Pre-Requisites	At least one food and one gift item must exist in inventory, and also GameplayGUI must be initialized and track stats.

Expected Results	Item’s effect is applied (hunger +10), item quantity decreases, if quantity reaches 0, item is removed, and UI updates accurately.
Test Category	Unit testing
Requirement	Using an item updates pet stats, updates inventory, and behaves correctly on the UI
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	Food or gift is applied successfully
Remarks	N/A

Test Case Name	**Test Feeding Action Updates Pet Stats and Score**
Test Case Description	Clicking the "Feed!" button correctly updates the pet’s stats, increases the score, and applies logic for stats boundaries and penalties
Test Steps	1. Initializing a GameplayGUI with values 2. Launching the feed window. 3. Simulating clicking the "Feed!" button. 4. Making sure the following status changes
Pre-Requisites	A GameplayGUI instance initialized with predictable values

Expected Results	Pet stats update as per the feeding logic, and core increases by 20.
Test Category	Unit testing
Requirement	Feeding behavior updates all pet values correctly and respects limits
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	File saves to the associated file type
Remarks	N/A

Test Case Name	**Test Playing Action Updates Pet Stats and Score**
Test Case Description	Clicking the "Play!" button updates the pet's stats as expected: increases happiness, decreases health, hunger, and sleep, and score should increase by 20.
Test Steps	1. Initializing a GameplayGUI mock with with values 2. Launching miniGame with this mocked object. 3. Clicking the "Play!" button. 4. Making sure the following status changes
Pre-Requisites	A GameplayGUI instance initialized with predictable values

Expected Results	Pet stats update as per the playing logic
Test Category	Unit testing
Requirement	Feeding behavior updates all pet values correctly and respects limits
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	Pet stats and score is updated when playing with pet
Remarks	N/A

Test Case Name	**Test sleeping Action Updates Pet Stats and Score**
Test Case Description	Clicking the "Go to bed!" button adjusts the pet’s stats properly, increasing sleep and reducing hunger, happiness, and health, and score should increase by 20.
Test Steps	1. Initializing a GameplayGUI mock with with values 2. Launching the sleep screen 3. Clicking the "Go to bed" button. 4. Making sure stats changes
Pre-Requisites	A GameplayGUI instance initialized with predictable values

Expected Results	Pet stats update as per sleeping logic
Test Category	Unit testing
Requirement	Verifying stats impact of sleeping
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	Pet stats and score is updated when putting pet to sleep
Remarks	N/A

Test Case Name	**Test Vet Visit Updates Health and Other Stats**
Test Case Description	When the "Heal!" button is pressed, the pet's health increases, while sleep, hunger, and happiness decrease slightly and score should also increase by 20.
Test Steps	5. Initializing a GameplayGUI mock with with values 6. Launching the going to vet screen 7. Clicking the "Heal" button. 8. Making sure stats changes
Pre-Requisites	A GameplayGUI instance initialized with predictable values
Expected Results	Pet stats update as per healing logic

Test Category	Unit testing
Requirement	Verifying stats impact of healing
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	Pet stats and score is updated when taking pet to the vet
Remarks	N/A

Test Case Name	**Rewards System**
Test Case Description	Tested that the RewardsGUI opens correctly with various item strings (e.g., “Bone”, “Toy”) Confirmed that the image path is built dynamically using the reward string and that no crashes occur when given expected values.
Test Steps	1. Confirmed that the image path is built dynamically using the reward string 2. Confirm that no crashes occur when given expected values.
Pre-Requisites	A GameplayGUI instance initialized with predictable values
Expected Results	Reward is given to inventory
Test Category	Unit testing

Requirement	Verifying rewards
Automation	Junit
Date Run	2025-04-03
Pass/Fail	Pass
Test result	Inventory is updated once reward is given
Remarks	N/A

Test Case Name	**Full Gameplay and Windows Compatibility**
Test Case Description	Tested to make sure that the game ran the **entire game start-to-finish** on a Windows machine, which included: Starting new games / Feeding, sleeping, playing, and vet interactions / Using items from the inventory / Saving and loading from files / Accessing Parental Controls / Navigating tutorial and dialog boxes
Test Steps	1. Verified compatibility on Windows, including fonts, file writing/reading 2. Confirmed image rendering went smooth on windows
Pre-Requisites	Any Windows machine
Expected Results	Full game runs on machine
Test Category	Unit testing
Requirement	All functionality passes tests
Automation	Junit

Date Run	2025-04-03
Pass/Fail	Pass
Test result	Full game runs with perfect functionality

2.4. Summary

In the largest stage of the project, we have designed, created, and implemented most ideas from the requirements and design documentation. Our game functions well from the starting screen to settings, to teaching the user how to play, to actual gameplay, saving, and exiting. Our tests also helped us find what we did well and what we need to improve on across Unit, Integration, Validation, and System Testing. We've tried our best to organize everything clearly, such as keeping image files, JSON files, and screens in their own folders, and connecting all parts smoothly. We have also kept in mind our objectives throughout the process, the importance of OOP design, gameplay mechanics like feeding and emotion tracking, converting our design into Java code, building a clean and usable GUI, and working efficiently as a team. Lastly, we focused on making the game not just structured, but fun, educational, and inspiring for players to virtually care for a pet.

As a group, we have also learned to work together by connecting through regular meetings, using GitLab to stay organized, and assigning tasks based on each member's strengths. We made sure to communicate often, whether it was to solve bugs, review each other's code, or make design decisions. In this case, GitLab helped us track our progress, commit changes safely, and avoid overwriting each other's work. We divided responsibilities clearly, some focused on gameplay logic, others on UI design or JSON integration, but always came together to test and polish the final product. Overall, we've improved our collaboration, time management, and ability to adapt quickly as a team.

Table of Terms, Notations, Acronyms (Citations are found in references section of Introduction table)

Terms, Notations, Acronyms	**Description**
JSON	Short for JavaScript Object Notation, is a self-describing format for storing and transporting data
JUnit 5	Is a testing framework
GUI	Short for Graphical unit interface, displays a combined system of visual components for s system software
UX	Short for User experience design, is used by design teams to create products that the user can learn and benefit from

GitLab	Collaborative software development software that helps teams organise their projects.
IDE	integrated development environment which is the application that allows for software coding.
Java swing	Java Swing is a part of Java Foundation Classes and is used to create various applications.
Apache Maven	Apache Maven is a project management tool that leverages a centralized Project Object Model (POM) to streamline building, reporting, and documentation
HTML/CSS	HTML (markup language) structures content with elements like headings, lists, and tables, while CSS (stylesheet language) defines the visual appearance of that content, such as color, fonts, and layout.
Visual studio Code	An IDE studio/software to write/test code
Wireframe	Visual representation of interface (A rough draft of how it will) but not provide any functionality
Object	An object is a concrete instance of a class blueprint that holds its own unique set of data values for the defined fields.
Live Server	Live Server is a development tool that automatically refreshes your web browser whenever you make changes to your HTML, CSS, or JavaScript files
JavaDoc	A tool that creates HTML-based documentation from comments in Java source code
JPanel	A container that holds and organizes components like buttons or labels in a GUI
MouseListener	An interface that lets you react to mouse actions like clicks or entering a component
ActionListener	An interface that detects when a button or similar component is clicked
Unit testing	Testing small parts of your code (like a method or class) in isolation
Radio buttons	A group of options where only one can be selected at a time
Combo boxes	A dropdown menu that lets users pick one item from a list
README file	A text file that explains how to run or use a project

Executable File	A file that runs a program when you double-click it (mainly on Windows)
ZIP Archive	A compressed folder that stores one or more files in a smaller size
PDF	A file format used to share documents that look the same on any device
Unit Testing	Testing one small piece of code, a single method or class, by itself
Integration Testing	Testing how different parts of the program work together when connected
Validation Testing	Making sure the program does what the user or requirements expect
System Testing	Testing the entire program as a whole