Automatically Tested With OUnit:

1. Pet Creation and Attribute Initialization:
    - The creation of pets (Camel and Dog) and the initial attributes (health, happiness, energy, nutrition, money) are tested to ensure that pets are initialized correctly.
2. Attribute Getters and Setters:
    - Automated tests cover the functionality of getting and setting individual attributes like health, happiness, energy, and nutrition.
    - These tests also check the system's response to setting attributes within certain boundaries (e.g., health cannot go below 0 or above 10).
3. String Representation Functions:
    - Tests are included to ensure that the string representations of pet attributes (like health_to_string and status_to_string) return the correct format.
4. Microchip, Leash, and Sickness Management:
    - The functionality for managing a pet's microchip, leash, and sickness status, including setting them and retrieving its status, is tested automatically.
5. Boundary Conditions:
    - Several tests focus on how the system handles edge cases, such as trying to reduce a pet's health below zero or increasing it above its maximum limit.

Manually Tested:

1. User Interaction Flow:
    - Starting the Game: Testing the response to initial prompts and user inputs ('Y' or 'N'), ensuring the game reacts appropriately.
    - Selecting an Animal: Validating the animal selection process and correct creation with initial attributes.
2. Gameplay Features
    - Feeding, Walking, and Playing: Comprehensive testing of game mechanics, including user interactions and their impacts on pet health and happiness.
    - Interactive Commands: Testing commands like 'Walk', 'Feed', and 'Play' for proper functionality and updates to the pet's state.
3. Complex Game Mechanics
    - Training Sessions: Verifying that training sessions correctly evaluate responses, accumulate points, and update skills and attributes.

    - Shop Transactions: Ensuring accurate display and handling of items, transactions, and effects on pet's attributes and finances.
4. Special Features
    - Minigame Interaction: Testing the correct functioning of Ctrl+C interrupts and score updates during the minigame.
    - Minigames Mechanics: Ensuring minigames (battle, blackjack, cooking, trivia, etc.) awards points correctly and calculates prizes accurately.
5. Error Handling and Edge Cases
    - Input Validation: Checking that all user inputs are handled correctly, with appropriate feedback or prompts when needed.
    - System Responses: Verifying system actions, such as pet death from zero health or happiness, are triggered and handled correctly.
6. Integration and Flow
    - Overall Game Flow: Testing the integration and interaction of various game parts, ensuring consistent updates and transitions.
    - Consistency in Game State: Making sure that changes in pet attributes are accurately reflected throughout the game and persist over its course.

Black Box Testing:

- Pet Creation and Attribute Initialization: Tests were created without considering the internal workings of the function. The focus was on whether pets (both Camels and Dogs) were initialized correctly with the expected attributes like health, happiness, energy, nutrition, and money. This method checks the output based on the expected state after creation, ensuring that all attributes are set as per the defined rules for pet creation.
- String Representation Functions: Testing these functions involved looking at the output strings for various attributes without considering the internal logic that formats these strings. The correctness of output formats (like health_to_string, status_to_string) was verified against expected string formats, ensuring that user-facing information is displayed correctly.

Glass Box Testing:

- Attribute Getters and Setters: This involved understanding the internal logic and state changes of pets, ensuring that functions like increase_health, decrease_health, set_happiness, etc., correctly modify the pet attributes within defined boundaries (e.g., health not dropping below 0 or above the maximum threshold). The tests check edge cases like boundary conditions specifically.
- Microchip, Leash, and Sickness Management: Tests were designed with knowledge of how the attributes are supposed to work, checking both the setting and getting of the status. The tests ensure that the status can be toggled and

retrieved accurately, reflecting changes immediately as expected by the system's logic.

How our testing demonstrates correctness (conclusion):
The testing approach for the Pet Simulator ensures system correctness through a mix of black box and glass box methodologies, complemented by extensive manual testing. Black box tests validate user-facing functionalities like pet initialization and interface correctness without regard to internal workings, ensuring outputs meet user expectations. Glass box testing delves into the system's logic, verifying that attribute adjustments and management behave as intended across all scenarios, including edge cases. Manual testing further validates the integration of game components and real-world usability, confirming that complex interactions such as shop transactions and training sessions function seamlessly and that the system robustly handles user inputs and game mechanics. This comprehensive testing strategy ensures the game is both functionally sound and engaging for players.