Requirements Documentation

Team 81

Andrew Durnford, Eric Jiang, Calvin Luo, Edison Qu, Surajj Vinodh Kumar

1.0 Team 81 Contract

1.1 Meetings

1.1.1 Schedule and Format

- Regular team meetings will be held every Saturday at 7:30 PM EST via Google Meet
- Duration: 1 hour (may extend to 1.5 hours if needed)
- Additional meetings may be scheduled as needed for urgent matters or approaching deadlines
- Members who cannot attend must provide at least 24 hours notice

1.1.2 Meeting Roles and Responsibilities

- A designated note-taker will record minutes for each meeting
- Note-taking responsibility will rotate alphabetically by last name each week
- Minutes will include:
    - Attendance
    - Topics discussed
    - Decisions made
    - Action items assigned
    - Next steps
- Meeting minutes will be posted to the team's GitLab wiki within 24 hours

1.2 Work Norms

1.2.1 Time Commitment

- Team members are expected to contribute approximately 5 hours per week to the project
- This includes:
    - Meeting attendance (1-1.5 hours)
    - Individual work (3-4 hours)
    - Code review and documentation
- Additional time may be required as deadlines approach

1.2.2 Deadlines and Milestones

- Internal deadlines will be set at least one week before actual due dates
- Work will be broken down into weekly sprints
- Progress updates are required at each weekly meeting
- All code must be committed at least 48 hours before major project milestones

1.2.3 Work Review Process

- All code changes require review by at least one other team member
- Pull requests must be reviewed within 24 hours of submission
- Documentation changes must be reviewed by all team members
- Merge conflicts will be resolved during team meetings

1.2.4 Decision Making

- Technical decisions will be made by team consensus where possible
- If consensus cannot be reached, we will:
    1. Have each side present their argument
    2. Take a team vote
    3. In case of a tie, consult with the TA for guidance
- All major technical decisions must be documented in the GitLab wiki

1.3 Work Division

1.3.1 High-Level Division

- Frontend Development (GUI)
    - Java Swing/JavaFX implementation
    - UI/UX design
    - Sprite integration
- Backend Development
    - Game logic
    - Save/load functionality
    - Pet state management
- Testing and Documentation
    - Unit testing
    - Documentation
    - GitLab wiki maintenance

1.3.2 Task Assignment Process

- Tasks will be created and assigned during weekly meetings
- Team members can volunteer for tasks based on their strengths
- Workload will be balanced among team members
- Each task will have a primary owner and a secondary reviewer
- Task progress will be tracked using GitLab issues

1.4 Communication

1.4.1 Response Times

- Team members should respond to urgent messages within 4 hours
- Non-urgent messages should receive a response within 24 hours
- Weekend response times may be longer but should not exceed 48 hours

1.4.2 Tools and Platforms

- Primary Communication: Google Meet for meetings
- Code Repository: GitLab
- Documentation: GitLab Wiki
- Task Tracking: GitLab Issues
- Group Chat: On Discord

1.5 Amendments

This contract may be amended by unanimous team agreement during regular team meetings. All changes must be documented in the GitLab wiki.

1.6 Agreement

By contributing to the team's GitLab repository, all team members agree to abide by this team contract.

Last Updated: Feb 3, 2025

Signed by:
- Calvin
- Edison
- Andrew
- Surajj
- Eric

2.1 Main Page

Task / Part	Contributors	Description / Notes
Main page and Introduction	Calvin and Edison	Created the project overview page including title, subtitle, and table of contents. Set up the initial document structure and formatting.
Domain Analysis	Surajj, Andrew, Eric	Answered all the questions about the software domain of the product. Did properly analysis and answered all 9 questions.
Use Case Documentation	Eric	Created detailed use case diagrams and descriptions.
Use Case Diagrams	Andrew	Created and made use of case diagrams using the documentation.
Activity Diagrams	Edison And Calvin	Created detailed activity diagrams using the documentation provided.
Non-functional requirements	Edison And Calvin	Drafted all the non-functional requirements for the game to be developed.
Summary	Calvin and Edison	Created a summary alongside a list of acronyms to use.

2.2 Introduction

2.2.1 Overview

This project is about designing and simulating a virtual pet in Java, in which players care for a digital pet by managing four key aspects: hunger, happiness, health, and sleep. The simulation will be a simple game in which users can interact with their pets using commands such as feeding, playing, and providing medical care. Each pet will have unique characteristics and behaviors, making player decisions important in ensuring the pet's well-being. The game will include progressive mechanics, which means that the pet's needs will change over time, necessitating constant attention and management from the player. Neglecting the pet will result in decreased happiness, increased hunger, and eventual health deterioration.

The application will include a scoring system that rewards players for consistent and responsible pet care, as well as a gamification component to encourage regular participation. In-game mechanics allow players to earn items such as food and toys, which they can then use to improve their pet's happiness and health. A save/load feature will allow users to resume gameplay from their previous saved state, ensuring that progress is not lost when the app is closed. Furthermore, parental controls will allow guardians to set playtime limits and track gameplay statistics, making the game appropriate for younger players.

The application will be built with Java Swing or JavaFX to provide a visually appealing and user-friendly experience. The Graphical User Interface (GUI) will have several interactive screens, including a main menu, pet selection screen, gameplay interface, and parental control dashboard.

Aside from entertainment, this project will serve as an educational tool, assisting players in developing time management skills, responsibility, and an understanding of care-based interactions by allowing them to maintain their virtual pet. We hope to create a well-structured and maintainable Java application that combines fun and learning by following best practices in software engineering.

The core problem our software addresses is: How can we create an immersive, interactive experience that feels dynamic to the user while maintaining a simple and natural GUI? This problem can be divided into two parts: (1) delivering an immersive experience and (2) designing a natural and easy-to-navigate interface.

For the immersive experience, we aim to ensure every component of the project prioritizes the player’s engagement. We believe players want a virtual pet that feels “alive,” so we will emphasize regular interaction to create a sense of realism and uniqueness. This approach not only makes the pet feel like a real companion with evolving needs but also serves as an educational tool, teaching players responsibility and routine management.

To design a natural and intuitive GUI, we will draw inspiration from successful games like Tamagotchi, Neopets, Nintendogs, and Bitzee. By leveraging proven design practices from these games, we can implement a user interface that feels familiar and easy to navigate, ensuring a seamless and enjoyable experience for players.

Ultimately, this project will serve as an educational tool in addition to being an entertaining game. Players will develop time management skills, responsibility, and an understanding of care-based interactions through maintaining their virtual pet. By implementing software engineering best practices, we aim to create a well-structured and maintainable Java application that provides both entertainment and learning opportunities.

2.2.2 Objectives

Our main objectives of this project are to:

- Implement a functional, interactive virtual pet game with meaningful gameplay mechanics.
- Adhere to software engineering principles and Java best practices.
- Develop a user-friendly graphical interface using Java Swing or JavaFX.
- Include a save/load feature, allowing players to retain progress.
- Implement parental controls for setting restrictions and tracking playtime.
- Enhance project documentation with UML diagrams (use case and activity diagrams).
- Utilize GitLab for collaboration, ensuring version control and proper documentation.
- Create an interactive pet that can rest, eat, and display its happiness of when it needs more attention/care
- Create achievements to incentivize players to keep playing and reach the next goal/achievement

2.2.3 References

- CS2212B Group Project Specification, Winter 2025.
- Java Swing Documentation: https://docs.oracle.com/javase/tutorial/uiswing/
- JavaFX Official Site: https://openjfx.io/
- UML Use Case Diagrams: https://en.wikipedia.org/wiki/Use_case_diagram
- Activity Diagrams in UML: https://en.wikipedia.org/wiki/Activity_diagram
- GitLab Documentation: https://docs.gitlab.com/

2.3 Domain Analysis

The virtual pet project falls under the game development domain, specifically within the casual simulation game category. As a virtual pet simulation, it shares characteristics with games like Tamagotchi, Neopets, and Nintendogs, where players engage in light, repetitive gameplay to care for a digital pet. The core mechanics involve feeding, playing, and monitoring the pet’s health and happiness, creating a low-stakes but engaging experience.

The primary target audience includes casual gamers, particularly children aged 7 and up. While the primary demographic is a younger audience, casual older audiences should also be able to enjoy the game. The game is also designed for parents who may wish to monitor or limit playtime. Given these demographics, the game must maintain an intuitive and accessible interface, ensuring that players of all ages can easily interact with the pet and understand its needs.

Within the game development domain, this project falls into the virtual pet simulation subdomain, a subset of life simulation games. This category revolves around digital caretaking mechanics, where player engagement depends on the emotional connection formed with their virtual pet. The game must incorporate time-based mechanics, ensuring that player actions impact the pet’s well-being over time, encouraging repeated interactions.

Common challenges in this domain include player engagement and retention, balancing simplicity with depth, and ensuring robust data persistence. Virtual pet games often struggle to retain players beyond the initial novelty phase. Without engaging mechanics or a reason to return, players may lose interest. Another challenge touched on earlier is balancing simplicity and depth. Too simple and the game may be deemed boring. Too complex and the target audience will be confused. Additionally, data persistence and state management are critical, as the game must reliably save and restore pet progress, inventory, and player statistics.

The industry has a couple ways to address these issues. For player retention, games in the virtual pet simulation domain often include features such as daily rewards and notifications. While these methods are outside of the scope of the project, player retention can be remedied by varied gameplay. Balancing simplicity and complexity is achieved through careful planning, as well as introducing features gradually. Data persistence is solved through structured data formats, as well as manual save features and robust error handling to prevent save file corruption.

With this knowledge in mind, we can improve the efficiency of development. Focusing on the mechanics can both improve player retention and the complexity balance. Using Java frameworks like Swing can avoid lengthy custom solutions. Similarly, this market heavily benefits from object-oriented approaches due to the natural modularity of the systems. This allows for swifter debugging and better maintainability.

2.4 Functional Requirements

2.4.1 Functionality to be Delivered

User Interface
The game features a graphical user interface that allows players to interact with the pet and navigate menus. The interface includes mouse-based interactions to perform key actions, such as feeding the pet, playing, and cleaning. Keyboard shortcuts are available for frequently used actions, such as pressing 'P' to pause or ‘F’ to feed the pet. The UI must provide visual and auditory feedback to indicate player actions and pet responses.
At least five distinct screens or pages must be implemented such as:
Main menu screen, Gameplay screen, Instructional or tutorial screen, Load game or new game screen, Parental controls screen

Main Menu
The main menu provides options to start a new game, load a saved game, access settings, and exit the application. Menu navigation must be clear, with tooltips or labels to assist users.

Instructional and Tutorial System
The game includes an interactive tutorial to guide new players on feeding, playing, and managing the pet’s health.

Save and Load System
Players must be able to save their game progress and load previous sessions. Saved data must include pet status, game time, and user settings.

Pet Needs and Status Management
The pet must have three core needs that the player manages: hunger, happiness, and health. These attributes must decline over time, requiring player interaction to maintain them.

Interaction and Activities
Players must be able to interact with the pet through actions such as feeding, which improves hunger and health, playing, which improves happiness and reduces stress, and grooming, which maintains hygiene and health. The pet must visibly react to player actions.

Aging and Growth System
The pet must grow over time, changing in appearance and behavior based on its needs and interactions.

Events and Random Occurrences
Random events must occur, such as sickness or surprises, requiring the player’s attention.

Customization Features
Players should be able to customize their pet's appearance, such as fur color and accessories. Customization options should be available at the start of the game and accessible later.

Economy System
An in-game currency system must be implemented, allowing players to earn and spend currency on pet care items.

Parental Controls
A parental control screen must allow guardians to restrict in-game purchases and limit playtime.

Multiplayer Mode
Players can interact with each other's pets in a shared environment. This feature includes a friends list to invite and connect with other players, the ability to visit a friend's pet and perform limited interactions such as gifting items or petting, and a chat function for basic text-based communication.

Scenario Model

2.4.2 Actors:

Actor	Player
Description	The User who interacts with the virtual pet
Aliases	None
Inherits	None
Actor Type	Person
Active/Passive	Active

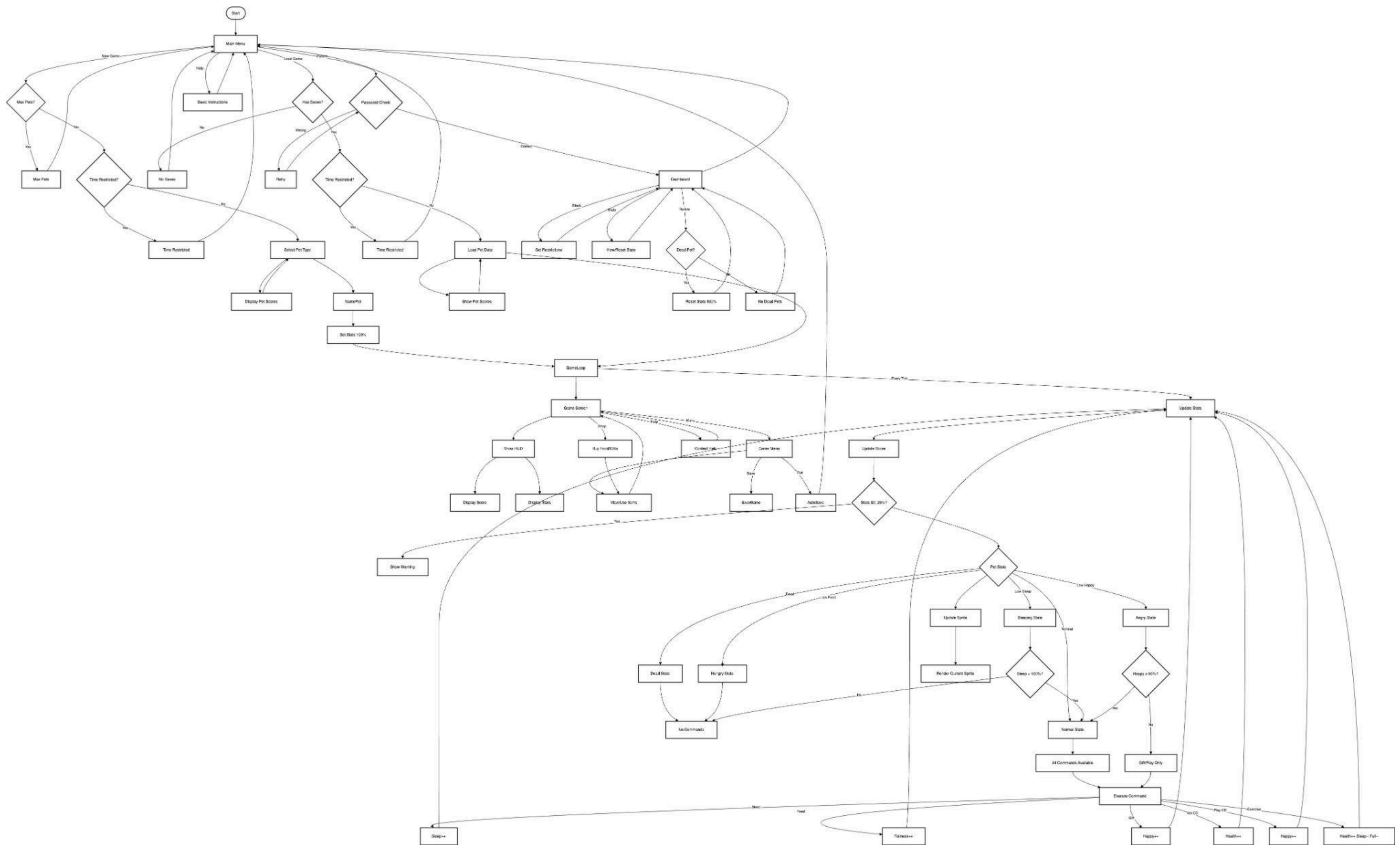
Actor	Pet
Description	The virtual entity that the player cares for
Aliases	None
Inherits	None
Actor Type	System

Active/Passive	Passive

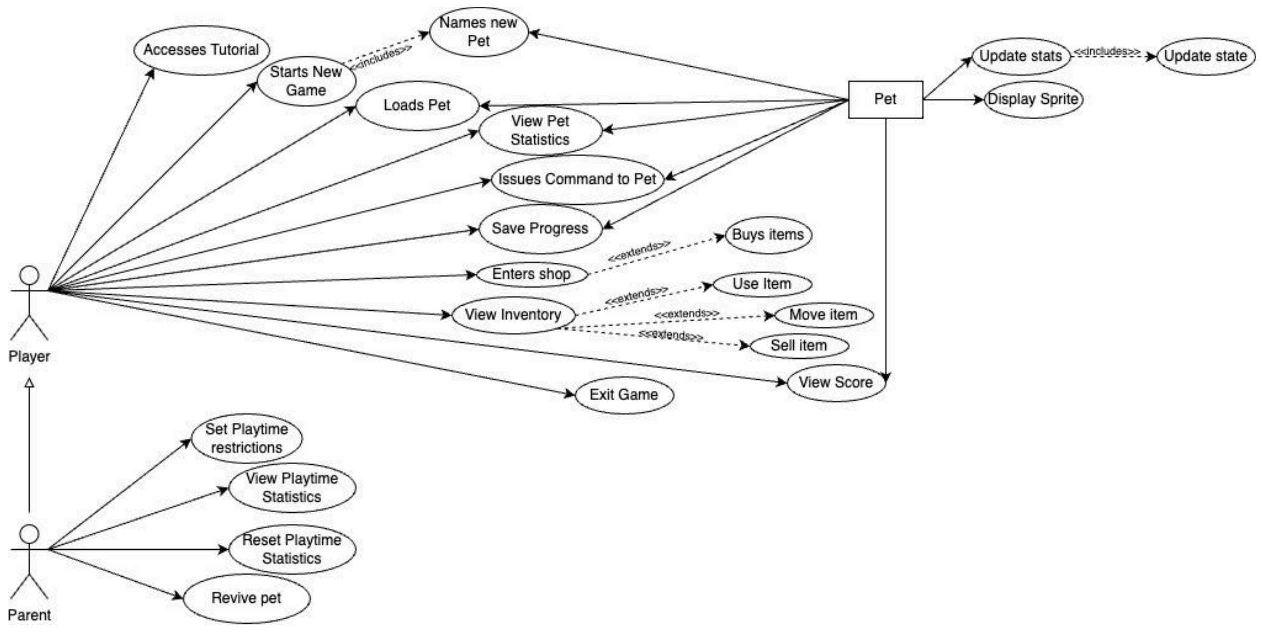
Actor	Guardian
Description	A parent or guardian who sets parents regulations
Aliases	None
Inherits	None
Actor Type	Person
Active/Passive	Active

2.4.3 Use cases & Diagrams

Activity Diagram Zoomed In



Use Case Diagram: Use case Diagram



Player starts a new game
User loads an existing pet
Player selects and names pet
View pet statistics
Instruction/tutorial screen
Issue commands to pet
Obtain items (food and gift)
Save progress
View & manage inventory
View score
Exit game
Update pet stats
Change state based on statistics
Display sprite
Access parental controls
Set playtime restrictions
View total and average playtime statistics
Reset playtime statistics
Revive a dead pet

Name	*Player starts a new game*
Primary actor	Player
Secondary actors	Pet
Goal in context	To load the screen to select a new pet
Preconditions	The user may or may not have pets already
Trigger	User accesses the pet screen
Scenario	1. The user accesses the pet selection screen from the main many 2. The user goes to the pet selection screen
Alternatives	1. The user accesses the pet selection screen within an existing game 2. The user goes to the pet selection screen
Exceptions	1. The player already has the maximum amount of pets 2. The player is unable to play due to parental restrictions
Priority	Highest

Name	*User loads an existing pet*
Primary actor	Player
Secondary actors	Pet
Goal in context	The user attempts to load an existing game
Preconditions	The user has an existing game
Trigger	User accesses the pet screen
Scenario	1. The user accesses the pet selection screen from the main screen 2. The user selects a pet, and that pet is loaded
Alternatives	1. The user accesses the pet selection screen within an existing game 2. The user selects a pet, and that pet is loaded
Exceptions	1. The player already has no pets 2. The player is unable to play due to parental restrictions
Priority	Highest

Name	*Player selects and names pet*
Primary actor	Player
Secondary actors	Pet
Goal in context	The player creates a new pet
Preconditions	Player has started a new game
Trigger	Player starts a new game
Scenario	1. Player goes through the start game use case 2. The player chooses the type of pet from a few options. Each option has a different tradeoff 3. The player gets to give the pet a name
Alternatives	1. User goes back to the pet screen
Exceptions	None
Priority	High

Name	*View pet statistics*
Primary actor	Player
Secondary actors	Pet
Goal in context	The user is shown the current statistics of the pet
Preconditions	The user has a pet
Trigger	1. The user is in the pet selection screen and has at least one pet 2. The user is in game with a pet currently loaded
Scenario	1. The player is in the pet selection screen or is playing the game with a pet loaded 2. The information about the pet shows up: a. Health b. Sleep c. Fullness d. Happiness e. Status with priority in the order given: i. Dead ii. Sleeping iii. Angry iv. Hungry v. Normal
Alternatives	None
Exceptions	None

Priority	High

Name	*Instruction/tutorial screen*
Primary actor	Player
Secondary actors	None
Goal in context	The player is shown the basic instructions
Preconditions	The user may be unfamiliar with games. The instructions need to be clear to everyone, inducing children.
Trigger	The user opens the tutorial on the home screen, or in game (unsure of which of these yet)
Scenario	1. The user presses the tutorial/help button 2. An instruction screen/overlay shows up 3. The user is given an option to exit the instruction screen/overlay
Alternatives	None
Exceptions	None
Priority	Low

Other	The tutorial screen won't show all the information - only what's needed to play, due to the target audience

Name	*Issue commands to pet*
Primary actor	Player
Secondary actors	Pet
Goal in context	The player interacts with the pet
Preconditions	The player currently has a pet, and is currently playing with it
Trigger	The player issues one of the possible commands

Scenario	1. The user loads a pet
	2. The user selects a command from the following (not finalized)
	    a. Go to bed
	    b. Feed
	    c. Give gift
	    d. Take to the vet. Has cooldown
	    e. Play. Has cooldown
	    f. Exercise
	3. The corresponding actions of the pet are taken

Alternatives	If alternative behaviour is possible at any of the steps outlined in the Scenario, it should be described here in a similar fashion. Say 'None' if there is no such alternative behaviour.
Exceptions	1. If the pet is asleep or dead, reject all commands 2. If the pet is angry, only give gift and play are available

Priority	High

Name	*Obtain items (food and gift)*
Primary actor	Player
Secondary actors	None
Goal in context	The player obtains food and gifts (and possibly other items)
Preconditions	The user is currently playing with a pet
Trigger	The user enters a shop menu
Scenario	1. The user enters a shop menu 2. The user purchases food, gifts, and other items if applicable THE ABOVE IS NOT FINALIZED, BUT THEY WILL HAVE SOME SIMILAR WAY TO OBTAIN FOOD AND GIFTS.
Alternatives	None
Exceptions	1. The player is unable to play due to parental restrictions
Priority	Low

Name	*Save progress*
Primary actor	User
Secondary actors	Pet
Goal in context	The user saves the current state of the pet to play later
Preconditions	The user is playing with a pet
Trigger	1. The user presses the save button 2. The user exits to the home screen
Scenario	1. The user plays with the pet 2. The user presses the save button 3. The game is saved into a file
Alternatives	1. The user plays with the pet 2. The user goes to the home screen 3. The game is saved into a file
Exceptions	Identify potential issues or situations that may arise from the various steps of this use case.
Priority	Highest

Name	*View & manage inventory*
Primary actor	Player

Secondary actors	None
Goal in context	The player views and manages the inventory
Preconditions	The user is currently playing with a pet
Trigger	The user presses the inventory button
Scenario	1. The user presses the inventory button 2. The user sees the current items and their quantities 3. The user can interact with the items: a. Use item b. Move item (in the inventory) c. Sell item (if currency is added)
Alternatives	None
Exceptions	None
Priority	Low

Name	*View score*
Primary actor	Player

Secondary actors	Pet
Goal in context	The player views the score
Preconditions	The user has pet(s)
Trigger	The user is in the pet selection screen or the user is playing as a pet
Scenario	1. The user enters the pet selection screen 2. The score for each pet is shown
Alternatives	1. The user is playing as a pet 2. The score is shown on the HUD
Exceptions	None
Priority	High

Name	*Exit game*
Primary actor	Player
Secondary actors	None

Goal in context	The user exits the game
Preconditions	None
Trigger	The user presses the close button in the menu or the window
Scenario	1. The user is in the game, opens a menu, then presses the exit button 2. The game closes. The game may be saved right before exit (not finalized)
Alternatives	1. The user presses the close button in the window 2. The game closes. The game may be saved right before exit (not finalized)
Exceptions	None
Priority	Highest

Name	*Update pet stats*
Primary actor	Pet
Secondary actors	None
Goal in context	The statistics of the pet gets updated
Preconditions	The pet is currently being played

Trigger	1. Some stats go down over time periodically 2. A command is issued
Scenario	1. An internal timer ticks 2. When it reaches a threshold, reduce all stats except health a. An indicator appears when any stat is below 25%
Alternatives	1. Health and happiness is decreased through hunger 2. The player issues a command: a. Go to sleep: sleep value gradually goes up b. Feed: Fullness increases c. Give gift: Happiness increases, varies by type of gift d. Take to the vet: health increases e. Play: happiness increases f. Exercise: sleep and fullness decrease, but health increases
Exceptions	1. Pet is dead, then only health can be increased by the revive
Priority	Highest

Name	*Change state based on statistics*
Primary actor	Pet
Secondary actors	None
Goal in context	The pet changes state based on its stats
Preconditions	The pet is currently being played

Trigger	The stats reach a threshold
Scenario	1. After the update pet stats use case runs, check the stats and run through the list. Priority is given based on the order a. If health is 0, pet is dead b. If sleep is 0, pet is sleeping until it is full c. If hunger is 0, pet is hungry d. If happiness is 0, pet is angry until it reaches half
Alternatives	None
Exceptions	None
Priority	High

Name	*Display sprite*
Primary actor	Pet
Secondary actors	None
Goal in context	Display the correct sprite
Preconditions	The pet is currently being played
Trigger	Change state based on statistics use case has been run

Scenario	1. The state of the pet is calculated 2. The corresponding sprite is displayed
Alternatives	None
Exceptions	None
Priority	High

Name	*Access parental controls*
Primary actor	Parent
Secondary actors	None
Goal in context	The parent accesses the parental controls
Preconditions	None
Trigger	The parental control menu is selected
Scenario	1. The parental control menu is selected 2. A password is prompted 3. If the password matches, the user is granted access to parental controls 4. Otherwise, the user is re-prompted to input the password

Alternatives	None
Exceptions	None
Priority	High

Name	*Set playtime restrictions*
Primary actor	Parent
Secondary actors	None
Goal in context	The parent sets time restrictions
Preconditions	The parent has entered the correct password
Trigger	The parent presses the time restriction button
Scenario	1. The parent presses the time restriction button 2. The parent is then shown options where the game cannot be played 3. Then, the player cannot play the game in the designated times without entering the parental password
Alternatives	None

Exceptions	None
Priority	Medium

Name	*View total and average playtime statistics*
Primary actor	Parent
Secondary actors	None
Goal in context	The parent views the playtime statistics
Preconditions	The parent has entered the correct password
Trigger	The parent presses the view playtime statistics button
Scenario	1. The parent presses the time restriction button 2. The parent is then shown the playtime statistics
Alternatives	None
Exceptions	None

Priority	Medium

Name	*Reset playtime statistics*
Primary actor	Parent
Secondary actors	None
Goal in context	The parent sets time restrictions
Preconditions	The parent has entered the correct password
Trigger	The parent views the playtime statistics
Scenario	1. The view playtime statistics use case goes through 2. Within the view stats screen, the parent is given the option to reset the playtime statistics
Alternatives	None
Exceptions	None
Priority	Medium

Name	*Revive a dead pet*
Primary actor	Parent
Secondary actors	None
Goal in context	The parent sets time restrictions
Preconditions	The parent has entered the correct password
Trigger	The parent presses the revive pet button
Scenario	1. The parent goes to the pet selection screen as a normal player 2. As a parent, they also see a button to revive a pet if the state is "dead" 3. When pressed, the pet then has all of the statistics maximized
Alternatives	None
Exceptions	None
Priority	Medium

2.5 Non-Functional Requirements

2.5.1 Technical Requirements

2.5.1.1 Development Environment

- The application must be developed using Java 21 or newer
- The application must be executable on Windows and Mac systems with standard Java installation
- All team members must use the same development environment and tools as specified in the team contract

2.5.1.2 Performance

- The application must run efficiently without unnecessary resource consumption
- The interface must be responsive with no noticeable lag during gameplay
- The application should have a total file size under 500 megabytes
- All user interactions must receive immediate visual feedback

2.5.2 Design and Architecture

2.5.2.1 Code Structure

- The application must follow object-oriented principles
- Must implement appropriate design patterns with documented justification
- Code must be modular and maintainable for easy updates
- Must adhere to consistent coding conventions across all files

2.5.2.2 Documentation

- All code must include Javadoc comments for methods and files
- External code usage must be properly cited in both documentation and comments
- All documentation must be written in clear, comprehensible English
- The project must maintain comprehensive documentation in the GitLab wiki

2.5.3 User Interface

2.5.3.1 Accessibility

- Must support both keyboard and mouse interactions
- UI elements must follow a logical tab order
- Color schemes must consider colorblind users

- Interface elements must scale appropriately if window resizing is allowed

2.5.3.2 User Experience

- Must follow best UX practices for intuitive navigation
- Must maintain consistent visual language throughout
- Must be appropriate for users aged 7 and up
- Must provide clear error messages in simple English

2.5.4 Data Management

2.5.4.1 Storage

- All data must be stored locally without internet dependency
- Must use standard file formats (JSON, XML, CSV, or TSV)
- Must not modify files outside the application directory
- Must protect save data integrity during read/write operations

2.5.4.2 Data Security

- Must not expose sensitive user data
- Must validate all user inputs
- Must handle errors gracefully without crashing
- Must maintain data consistency across game sessions

2.5.5 Development Process

2.5.5.1 Version Control

- All code must be maintained in GitLab repository
- Regular commits must be made throughout development
- Must maintain proper documentation of changes
- Must include appropriate .gitignore files

2.5.5.2 Testing

- Must include comprehensive JUnit 5 tests
- GUI elements exempt from unit testing requirement
- Must include error handling for all user inputs
- Must undergo thorough acceptance testing

2.5.6 Additional Requirements

2.5.6.1 Dependencies

- Must only use freely available third-party libraries
- Must document all external dependencies
- Must include setup instructions for required libraries
- Must minimize external dependencies

2.5.6.2 Compliance

- Must adhere to academic integrity guidelines
- Must properly credit all external resources
- Must comply with software licensing requirements
- Must follow course-specific guidelines and requirements

These requirements are designed to ensure the development of a robust, maintainable, and user-friendly virtual pet application that meets both technical and educational objectives.

2.6 Summary

The ultimate purpose of this document is to provide a detailed outline of the project's specifications and requirements. Before doing so, our team has signed a contract which all team members will follow. The contract provides colour around expectations surrounding deadlines, schedules, and communication that we, as a team, believe we must adhere towards to ensure efficient collaboration and accountability.

Other key parts of this document include an executive summary of the project's specifications, which is within the introduction. At a high-level, our project is a game centred around taking care of a pet's happiness, hunger, sleep, and health - taking inspiration from popular successes such as Tamagotchi. Aside from the core functionality, our team has taken the liberty to include many ancillary features that support the overall user experience such as parental controls and an auto-save feature. Supporting the development of the project are non-functional requirements, which our team similarly outlines. Our team uses these requirements to protect the integrity of the game, such as local storage (without internet), version control, and compliance with all regulatory restrictions when using third-party and course-specific architecture.

Our team also describes the software domain for the project, which is primarily targeting casual gamers within the virtual pet simulation subdomain**.** We believe our game is best positioned to benefit from these domains, and we plan on mitigating the challenges associated with them through incorporating daily rewards and notifications, to name a few.

Deeper into the document contains parts that go more in depth into how the project is coded. Our team details actors, specific use cases, and a comprehensive activity diagram that shows the flow of the entire software and how all the actors and use cases tie together. Readers interested should take the time to analyze how each use case impacts the overall game, and please don't hesitate to email cluo86@uwo.ca, cqu43@uwo.ca**,** ejiang26@uwo.ca**,** adurnfor@uwo.ca**,** or svinodh@uwo.ca if you have any questions.

2.6.1 Terms and Definitions

Term	**Definition**
GUI	Graphical User Interface
UI	User Interface
UX	User Experience
UML	Unified Modeling Language
HUD	Heads-Up Display