2212 Virtual Pet Project

Design Documentation

Team 75: Yonas Asmelash, Julia Norrish, Jessica Wang

1. Main Page

Table of Contents

Team Member Contribution Table

Task/Part	Contributors	Description/Notes
1. Main Page	Yonas	Created the title, subtitle, and project title. A subtitle indicates that this document is the design documentation for the project. Moreover, put together a contributor's table outlining which team members contributed to which parts of the milestone. Lastly, outlined a well-structured table of contents for the document.
2. Introduction	Yonas	First, I constructed an overview consisting of a summary of the system requirements of the pet game and the game's purpose, and I assembled a list of references that aid in the understanding of this document. Then, I outlined the general objectives of the design document and described how the project as a whole aligns with these objectives. Lastly, I provided a list of references to other documents that may assist in understanding the design documentation.
3. Class Diagrams	Julia	Created a UML class diagram to represent the classes in the project and the relationships between them.
4. User Interface Mockup	Jessica	Came up with and implemented UI mockup

5. File Formats	Yonas	Researched and specified which file format will be utilized to store data and listed any external libraries required to handle this file format. Provided a detailed structure of the file, including: what data elements will be stored, the order of fields/columns in the file (if applicable).
6. Development Environment	Yonas	Researched and described the IDE, tools, and external libraries we will be using to develop the software, accounting for the fact that the software must be a Java program to run on a Windows system with Javadoc and JUnit to document and test our code, respectively.
7. Patterns (Jessica)	Jessica	
8. Optional Diagrams	N/A	We did not feel it would be beneficial to include additional diagrams and documentation for the design of our software.
9. Summary	Yonas	Provided a brief summary wrapping up our document and included a table of terms, notations, and acronyms you have introduced in your document with concise definitions.

2. Introduction

Overview

This document outlines the design of our Virtual Pet Game, a Java-based application that simulates pet ownership through interactive gameplay. The aim is to provide an engaging experience where a player cares for a virtual pet by controlling its hunger, happiness, and other attributes. The game is aimed at children to help them learn responsibility skills and routine management through gameplay. Furthermore, parents will have access to statistics and parental controls for their child in addition to playing the game as the player would.

Building on the requirements established in our Requirements Documentation, this document details the architectural and technical decisions that will guide the project's implementation. It tackles key design components, including system architecture, UI layout, and interactions between game components. The design decisions follow software engineering principles and best practices in order to present an optimal user experience.

Objectives

The primary objective of this document is to serve as a blueprint for the implementation of our Virtual Pet Game. Specifically, this document aims to:

- Define the structure of the system, detailing key components and their interactions.

- Establish the design of the GUI, ensuring a user-friendly experience that aligns with accessibility standards.
- Outline data management strategies for storing and retrieving game progress, including pet states and player interactions.
- Describe the game mechanics and logic, including pet behavior, command execution, and state transitions.
- Detail the application’s adherence to OOP principles and design patterns for modularity and extensibility.
- Provide a reference for implementation, ensuring consistency across all development team members.

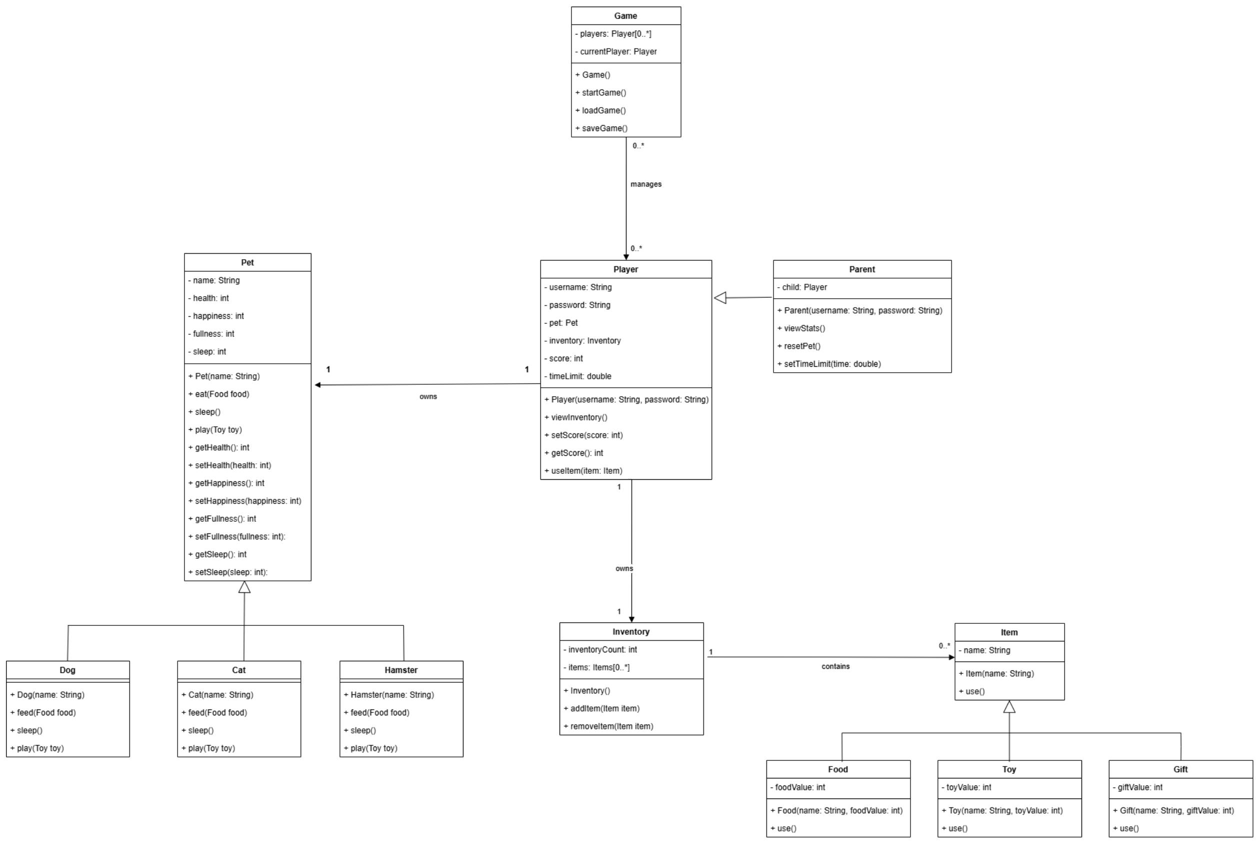
References

The following documents provide additional context and support for understanding this design documentation:

1. CS2212B Group Project Specification - This document outlines the overall expectations, requirements, and assessment criteria for the project.
2. Group 75 Requirements Documentation - This document provides a comprehensive breakdown of the functional and non-functional requirements that guided the creation of this design document.

These references serve as the foundation for our design approach, ensuring alignment with the project’s objectives and the requirements we’ve outlined.

3. Class Diagrams



The Game class controls the instances of the game. It keeps track of all the players, and which one is currently playing, which allows the progress of multiple players to be saved. This class includes the functionality of starting a new game, loading an already-created game, and saving a game. The purpose of this class is to manage the player and ensure the game can be played in different sessions. Having a separate Game class makes it easier to perform global tasks while keeping each player's data separate.

The Player class represents a standard user of the game, and is responsible for holding the player's username, password, pet, inventory, and score. Players can view their inventory to see the items they currently own and use them to interact with their pet. There is a time limit field which stores the player's time limit if one is set by the parent. Storing this data in the Player class makes performing player-related tasks more manageable, as they are self-contained.

The Parent class is a subclass of the Player class meaning it has the same fields and methods as the Player class as well as some additional properties unique to parents. Parents are able to view the game statistics, reset the pet in the event that the game ends, and set a time limit for their child. This design minimizes duplicate code and ensures parents can interact with the game like regular players while also having additional abilities to manage their child's account.

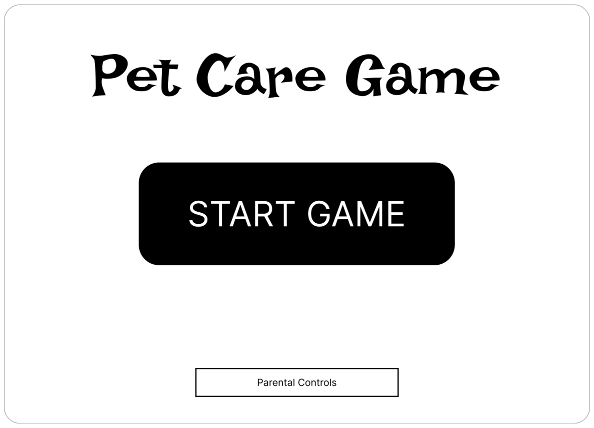
The Inventory class is used to keep track of the items the player owns. Items can be added to and removed from the inventory as players acquire and use them. Making the Inventory a separate class rather than a list of items in the Player class promotes encapsulation, lessens the responsibilities of the Player class, and results in cleaner code.

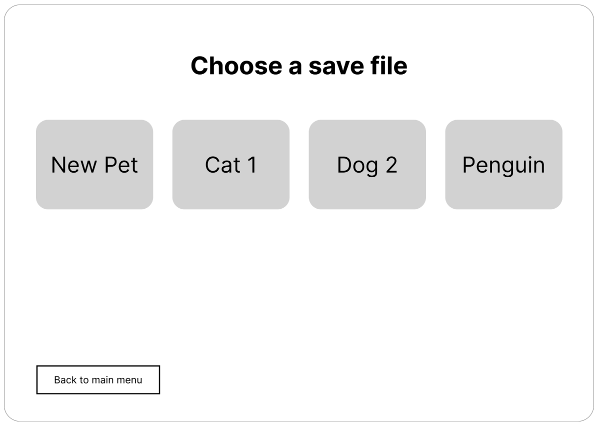
The Item class represents an item that the player can use to interact with their pet. This is a generic class that has the base functionality with the name of the item and a method to use the item. This design allows for each type of item to have common functionality while enabling changes depending on the type of item. The Item class has three subclasses: Food, Toy, and Gift. Each item will have a value associated with it that modifies the pet's statistics. For example, food has a value to increase the pet's fullness, while a toy would increase happiness. Using inheritance ensures the code remains clean and understandable by making the code modular.

The Pet class is used to represent a generic pet in the game. Each pet has a name and an integer value to represent their health, happiness, fullness, and sleep. Pets will be able to eat, play, and sleep. These actions may use items in the game, and they will increase the pet's statistics. This class has three subclasses representing specific pets: Dog, Cat, and Hamster. Each type of pet has differences in their statistics, such as different amounts of sleep needed. Each type of pet will react differently to eating, playing, and sleeping. Some pets may get full faster when eating, while others may require more time playing to be happy. Utilizing this subclass structure ensures each type of pet has the same foundation, while allowing differences to create a more varied and interesting playing experience.

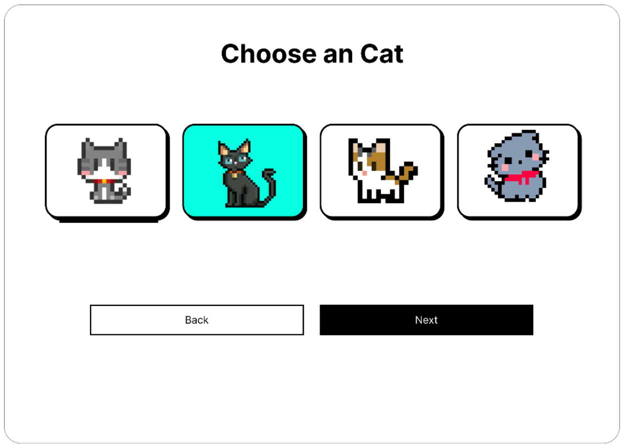
4. User Interface Mockup

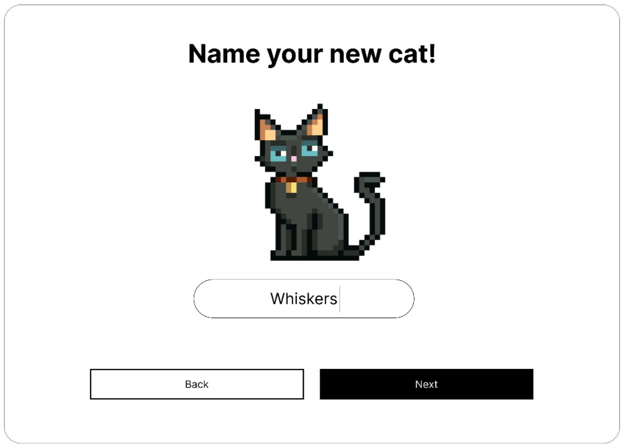
1. Log in and choose save file

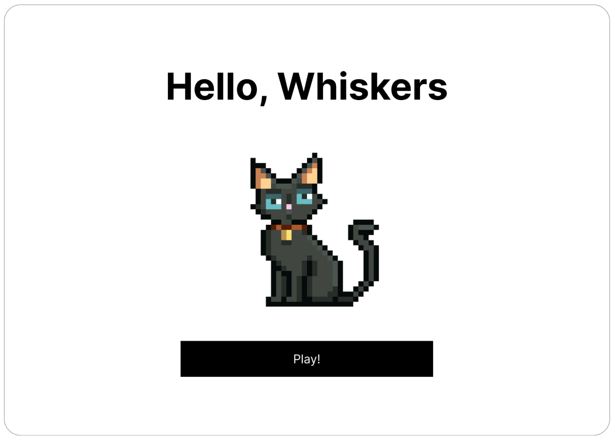




2. Create a new pet

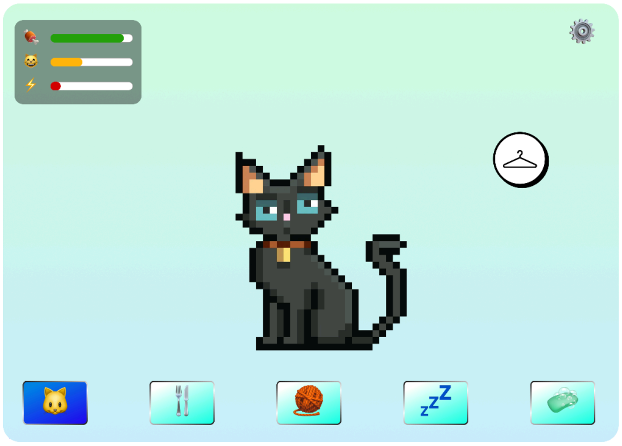


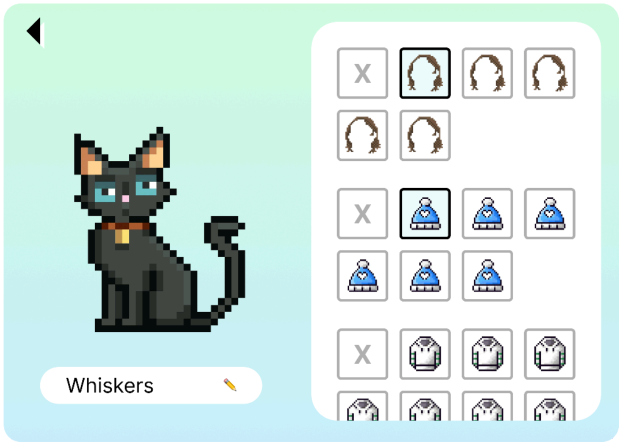




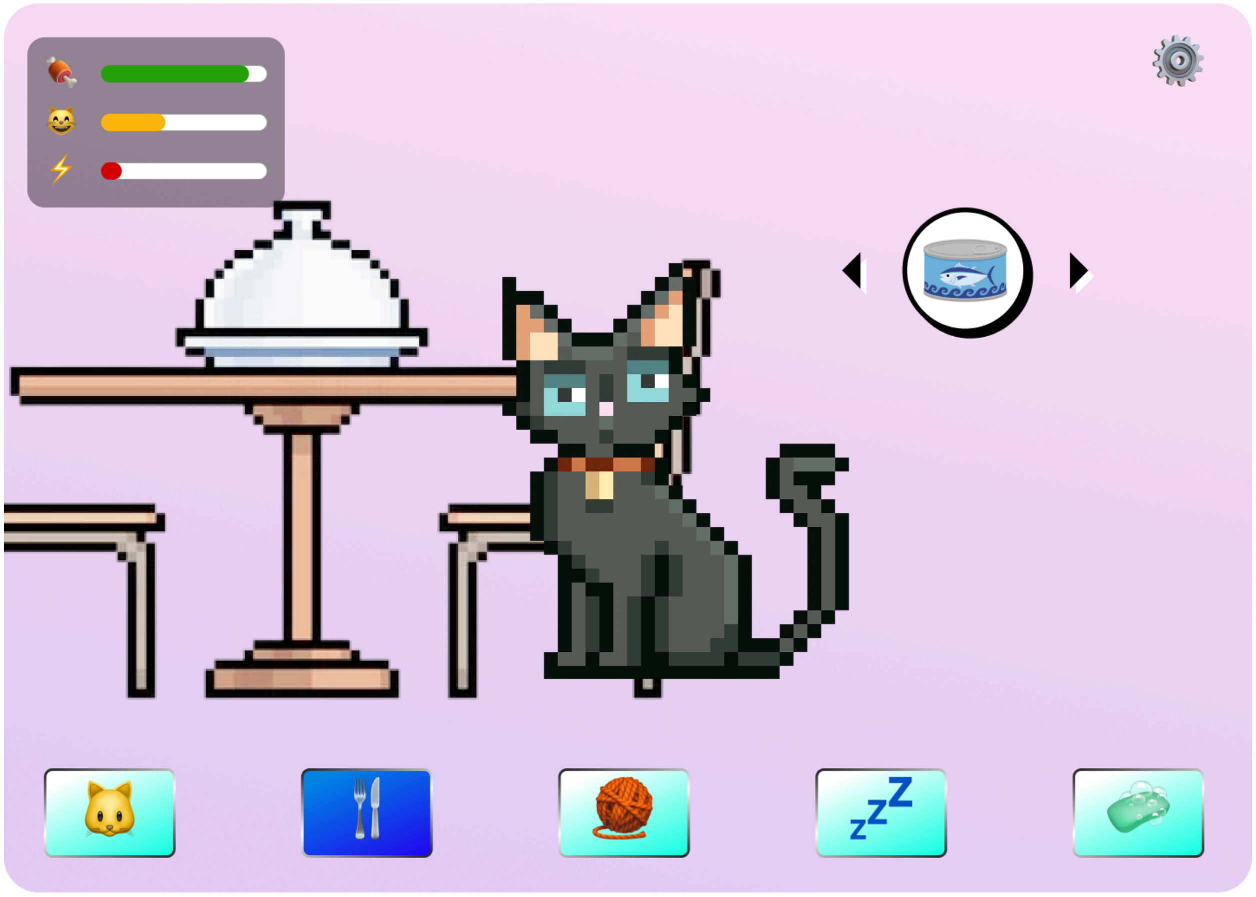
3. Pet Home (Gameplay)

Customize Pet Screens

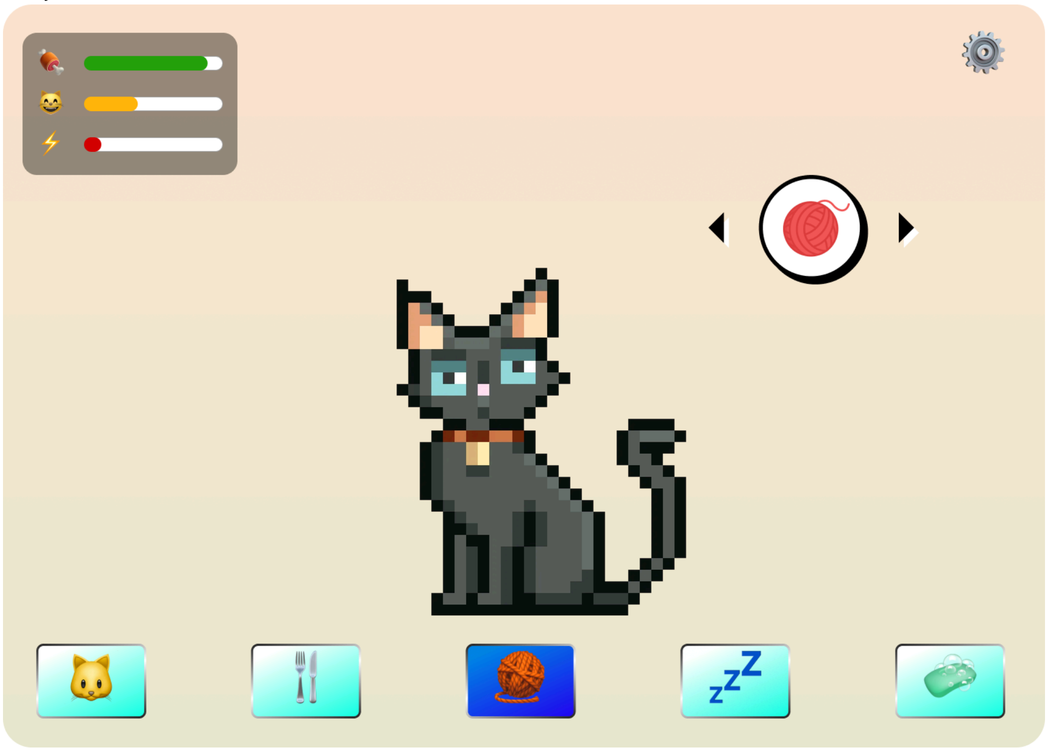




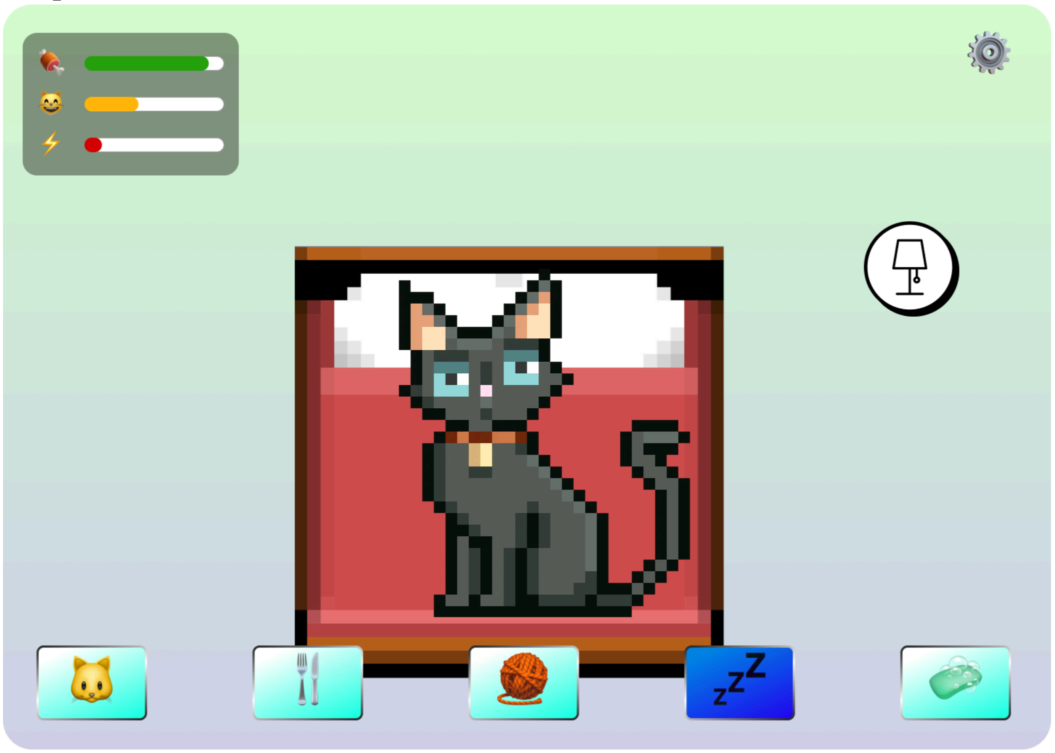
Feed Pet



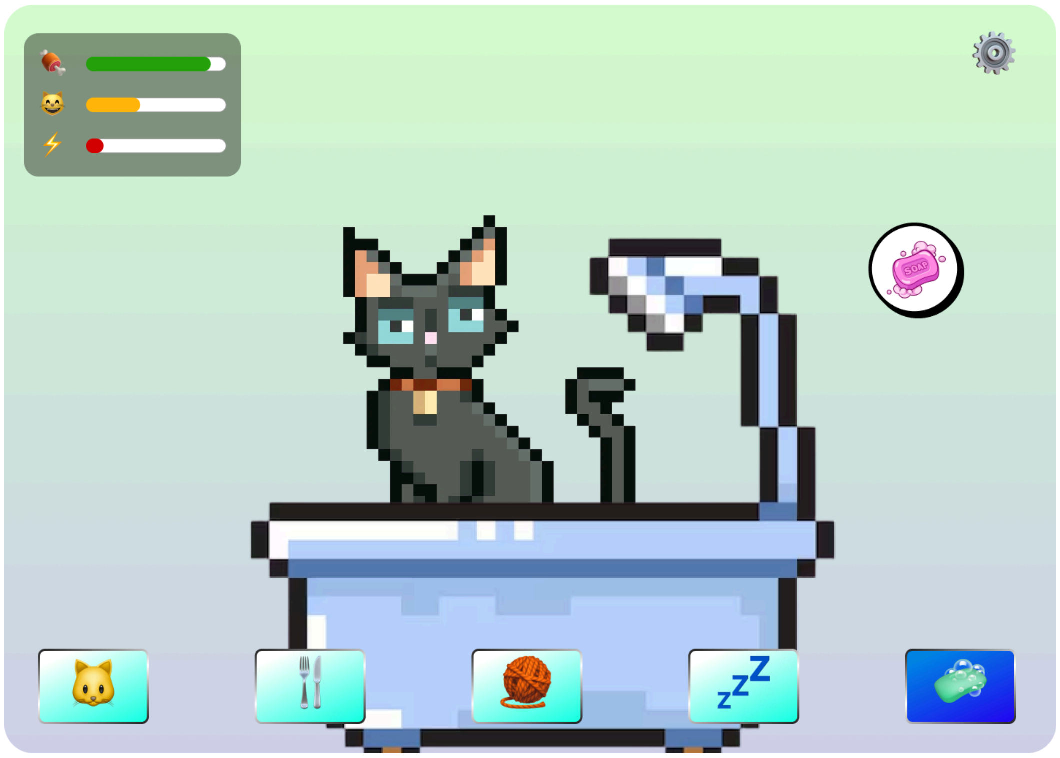
Play with Pet



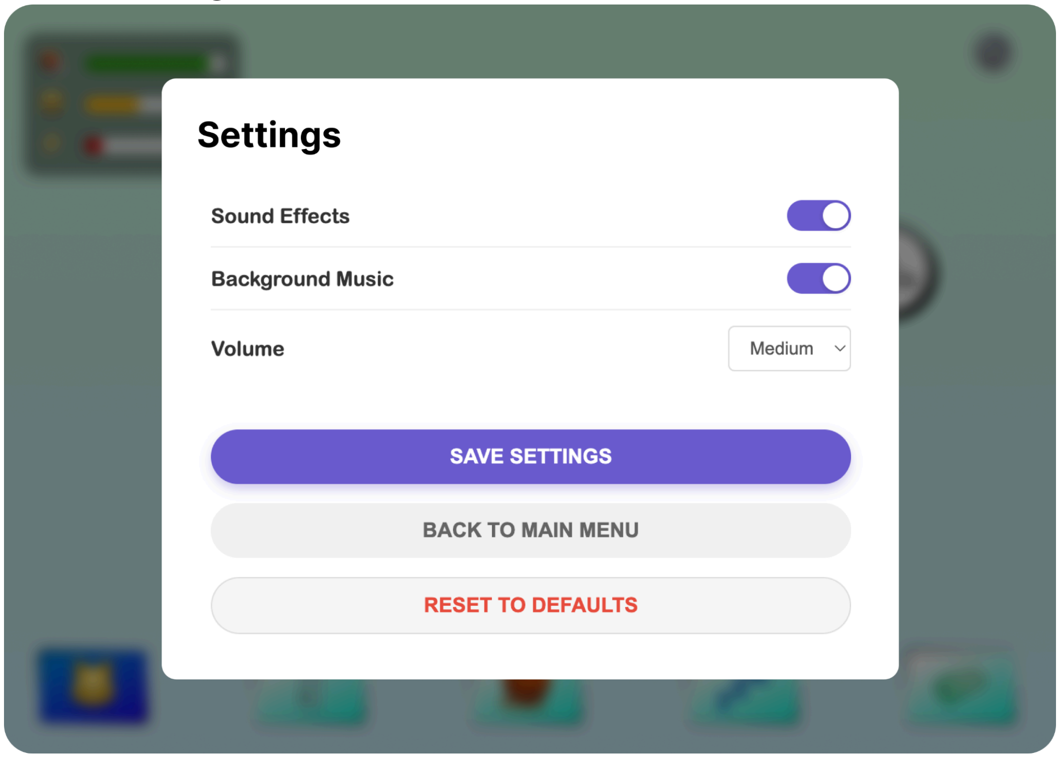
Sleep



Clean Pet



In-Game Settings



Settings

Sound Effects: On

Background Music: On

Volume: Medium

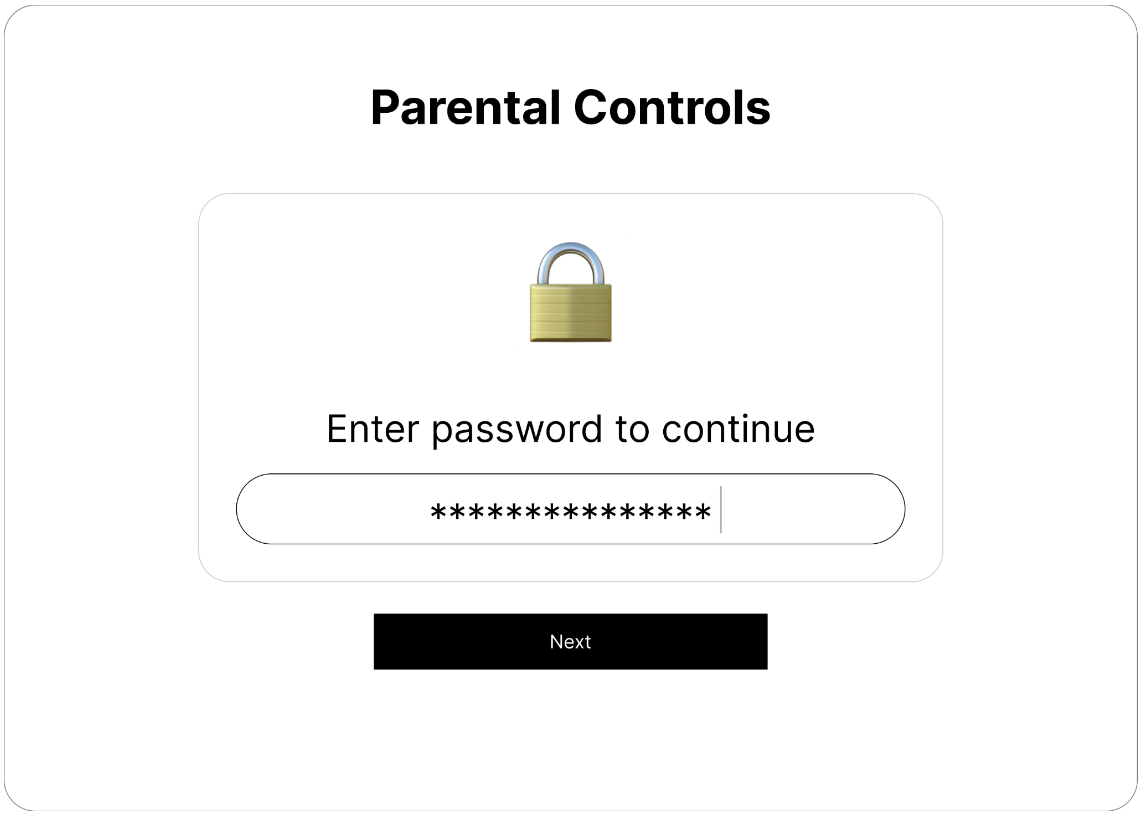
SAVE SETTINGS

BACK TO MAIN MENU

RESET TO DEFAULTS

4. Parental Controls

Password Protection



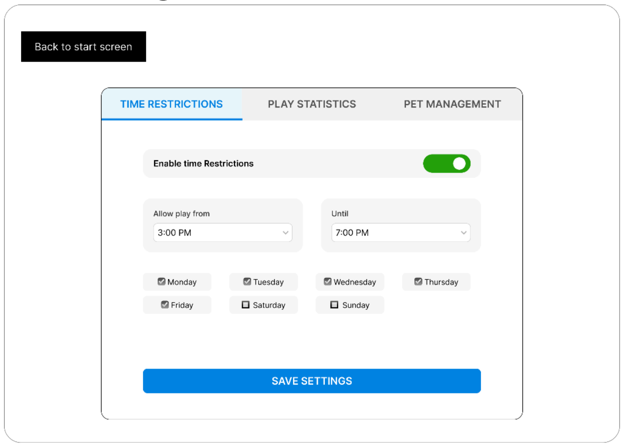
Parental Controls

Enter password to continue

****

Next

Time Management Controls



Back to start screen

TIME RESTRICTIONS | PLAY STATISTICS | PET MANAGEMENT

Enable time Restrictions

Allow play from: 3:00 PM

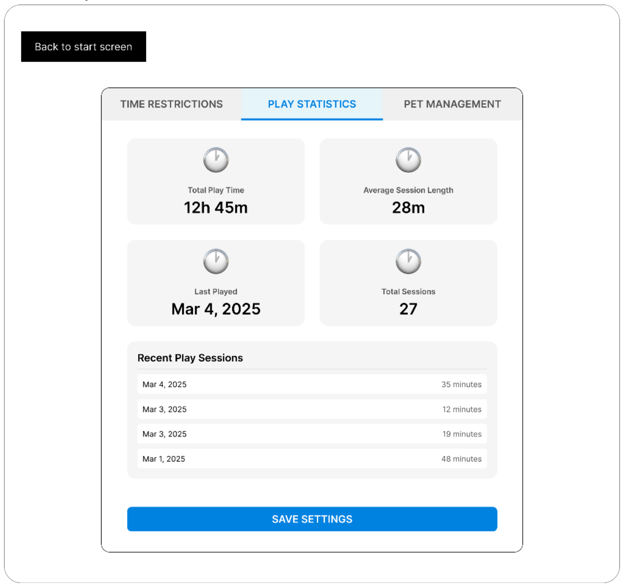
Until: 7:00 PM

☑ Monday ☑ Tuesday ☑ Wednesday ☑ Thursday

☑ Friday ☐ Saturday ☐ Sunday

SAVE SETTINGS

Play Statistics



Back to start screen

TIME RESTRICTIONS | PLAY STATISTICS | PET MANAGEMENT

Total Play Time: 12h 45m

Average Session Length: 28m

Last Played: Mar 4, 2025

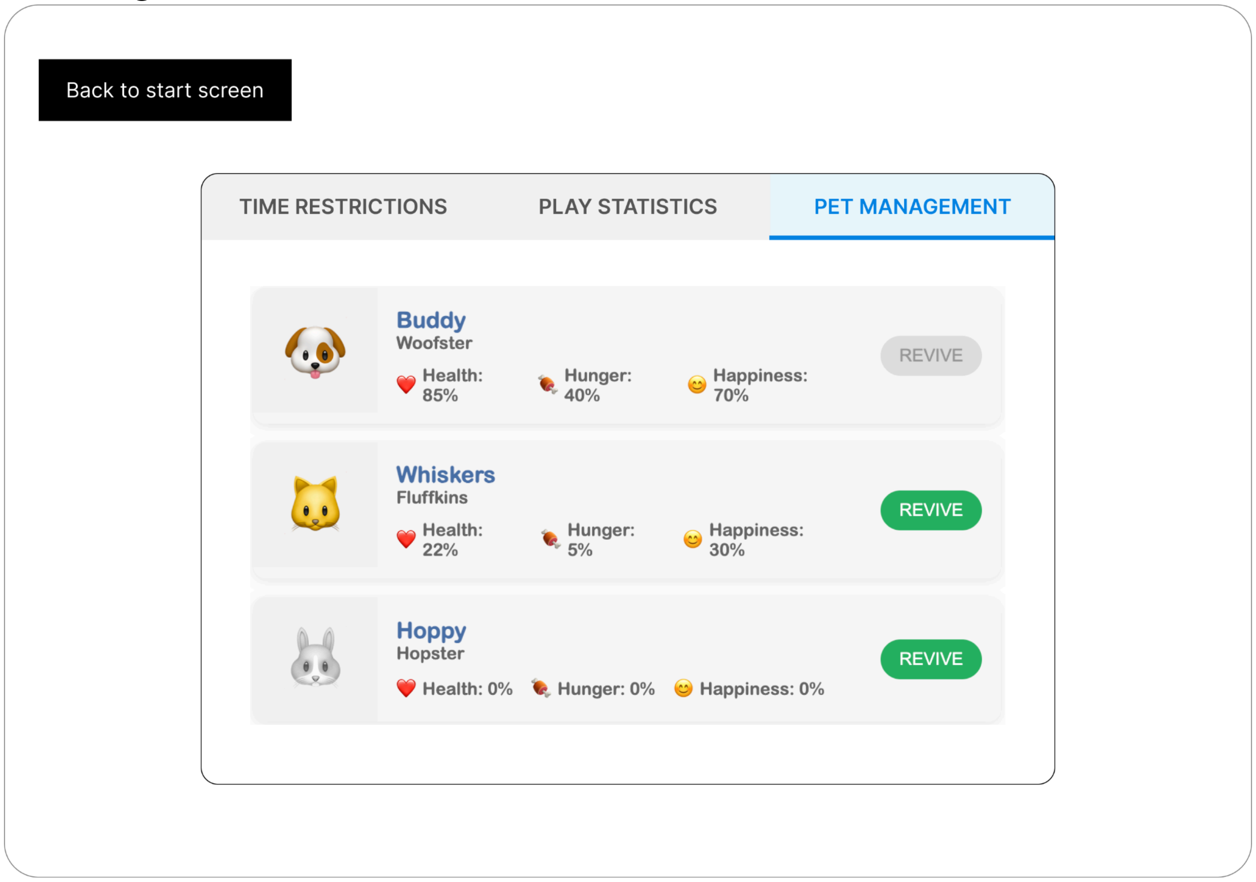
Total Sessions: 27

Recent Play Sessions

Mar 4, 2025	35 minutes
Mar 3, 2025	12 minutes
Mar 3, 2025	19 minutes
Mar 1, 2025	48 minutes

SAVE SETTINGS

Pet Management



Buddy
Woofster
Health: 85% Hunger: 40% Happiness: 70%

Whiskers
Fluffkins
Health: 22% Hunger: 5% Happiness: 30%

Hoppy
Hopster
Health: 0% Hunger: 0% Happiness: 0%

5. File Formats

To ensure efficient and structured storage of player and virtual pet information, we will use the JSON (JavaScript Object Notation) file format. JSON provides a lightweight and human-readable means of storing structured data, making it ideal for managing game state persistence. Moreover, JSON is easily compatible with Java and provides ease of integration with external libraries.

In regards to external libraries, we will utilize the Jackson library (com.fasterxml.jackson.core) to handle JSON parsing and serialization in Java as it's widely used for converting Java objects to JSON and vice versa, offering robust support for nested structures and data type conversions.

The following data elements will be stored in the JSON file:

- Player name
- Player score
- Player inventory (list of items and quantities)

- Pet name
- Pet type
- Pet health
- Pet happiness
- Pet hunger
- Pet thirst
- Pet sleep level
- Pet state (ex. happy, angry, etc.)
- Game settings (ex. sound, difficulty etc.)

For data handling and file organization, the player's progress will be stored in a uniquely named JSON file. When a game is saved, the current state of the player and pet will be written to the corresponding JSON file. Then, when the game is reloaded, the data from the JSON can be used to reconstruct the player and pet objects. The software will handle error detection to prevent data corruption, including validation of missing or malformed JSON files. In doing so, we will be able to ensure that game progress can be easily saved, loaded, and modified safely.

6. Development Environment

Our team will be using VScode for developing the Virtual Pet Game. VScode's UI is very intuitive, and our members all have extensive experience utilizing the software to build projects in the past. Moreover, it has support for many extensions that allow it to be as good as a dedicated Java IDe whilst maintaining flexibility.

We will be using GitLab for version control so that we can ensure collaborative development. Documenting our code will be done via Javadoc. Lastly, we will use JUnit 5 for testing.

7. Patterns

a) Observer Pattern

Why it's appropriate: Looking at our class diagram, we need a mechanism for our Pet's state changes (health, happiness, fullness, sleep) to trigger updates in the UI. The Observer pattern allows for this loose coupling between the Pet object (as the subject) and various UI components (as observers).

Implementation: While not explicitly shown in our current class diagram, we can implement this by having our Game class act as an observer of Pet state changes. The Pet class would notify the Game when significant state changes occur (like stats falling below 25%), and the Game would then update the appropriate UI elements. This maintains our current class structure while adding the pattern's benefits without significant restructuring.

b) Strategy Pattern

Why it's appropriate: Our class diagram shows we have different pet types (Dog, Cat, Hamster) that inherit from the Pet class but implement certain behaviors differently. The Strategy pattern will help us manage these varying behaviors without extensive conditional logic.

Implementation: We're already using this pattern in our class hierarchy. The Pet class defines the interface for behaviors like feed(), sleep(), and play(), while the concrete subclasses (Dog, Cat, Hamster) provide specific implementations. This follows the Strategy pattern where algorithms (pet behaviors) are encapsulated into separate, interchangeable classes.

c) Singleton Pattern

Why it's appropriate: Our class diagram shows a Game class that manages players and game state. This is a perfect candidate for the Singleton pattern, as we only need one instance of the Game class throughout the application's lifecycle.

Implementation: The Game class in our diagram already has the structure to be implemented as a Singleton. It contains global state (players and currentPlayer) and methods that operate on this state (startGame(), loadGame(), saveGame()). We can enhance it by ensuring only one instance exists by implementing a private constructor and a static getInstance() method. This ensures consistent game state management across the application.

These patterns fit naturally with our existing class diagram and will help us create a robust, maintainable system. The Observer pattern will improve communication between components, the Strategy pattern is already reflected in our inheritance structure, and the Singleton pattern will ensure proper state management through our Game class. These implementations require minimal changes to our current class structure while providing significant architectural benefits.

8. Summary

The **Virtual Pet Project Design Documentation** explains how Team 75 plans to build a Java-based pet simulation game. It includes details on how the game is structured, such as class diagrams, how data will be stored using JSON, and how the user interface will look. The document also describes the tools the team will use, like GitLab for collaboration and JUnit for testing. Overall, it serves as a guide to help the team develop the game efficiently and keep everything well-organized.

Terms/Notations/Acronyms	**Definition**
IDE	Integrated Development Environment
GUI	Graphical User Interface
JSON	JavaScript Object Notation, a lightweight data format used for storing and exchanging dat
JavaDoc	A documentation tool used in Java
JUnit	A testing framework for Java used to write and run unit tests.
Gitlab	A version control platform

Jackson	A Java library used for parsing and serializing JSON data
OOP	Object-Oriented Programming, a programming practice that organizes software design around objects and classes.