# Virtual Pet Game

## Design Documentation

## 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 did the references. |
| Class Diagrams | Aryan Baria \| Dilraj Deogan | Created the UML Class Diagram that outlines the general flow of the program. |
| User Interface Mockup | Mohammed Bayoumi \| Maher Rammal | Mohammed and Maher worked both on the designing aspect and describing them |
| File formats | Maher Rammal | Wrote file format overview |
| Development Environment | Maher Rammal \| Mohammed Bayoumi | Maher and Mohammed worked on the development environment information. |
| Patterns | Rayan Amir \| Aryan Baria | Rayan and Aryan chose and provided descriptions of the chosen software patterns. |
| Summary | Aryan Baria \| Dilraj Deogan | Wrote a summary wrapping up our document. |

## 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.<br>● Implement requirements such as pet interactions, pet emotions, animations, feeding mechanics, and user experience.<br>● Implement our design into Java code successfully to capture our decisions into machine code.<br>● Create a graphical user interface that is both functional and user-friendly.<br>● Work successfully as a team prioritizing team chemistry and collaboration to push the project success further.<br>● Write efficient, clean, well-documented Java code that adheres to best practices.<br>● 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.<br>● Draw.io program<br>● Servos, Daniel. CS2212: Introduction to Software Engineering, Test resources. Quiz 1 resource. Winter Session 2025, Owl.<br>● Servos, Daniel. CS2212: Introduction to Software Engineering, Week 2 Unified Modeling Language. Winter Session 2025, Owl.<br>● **"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.

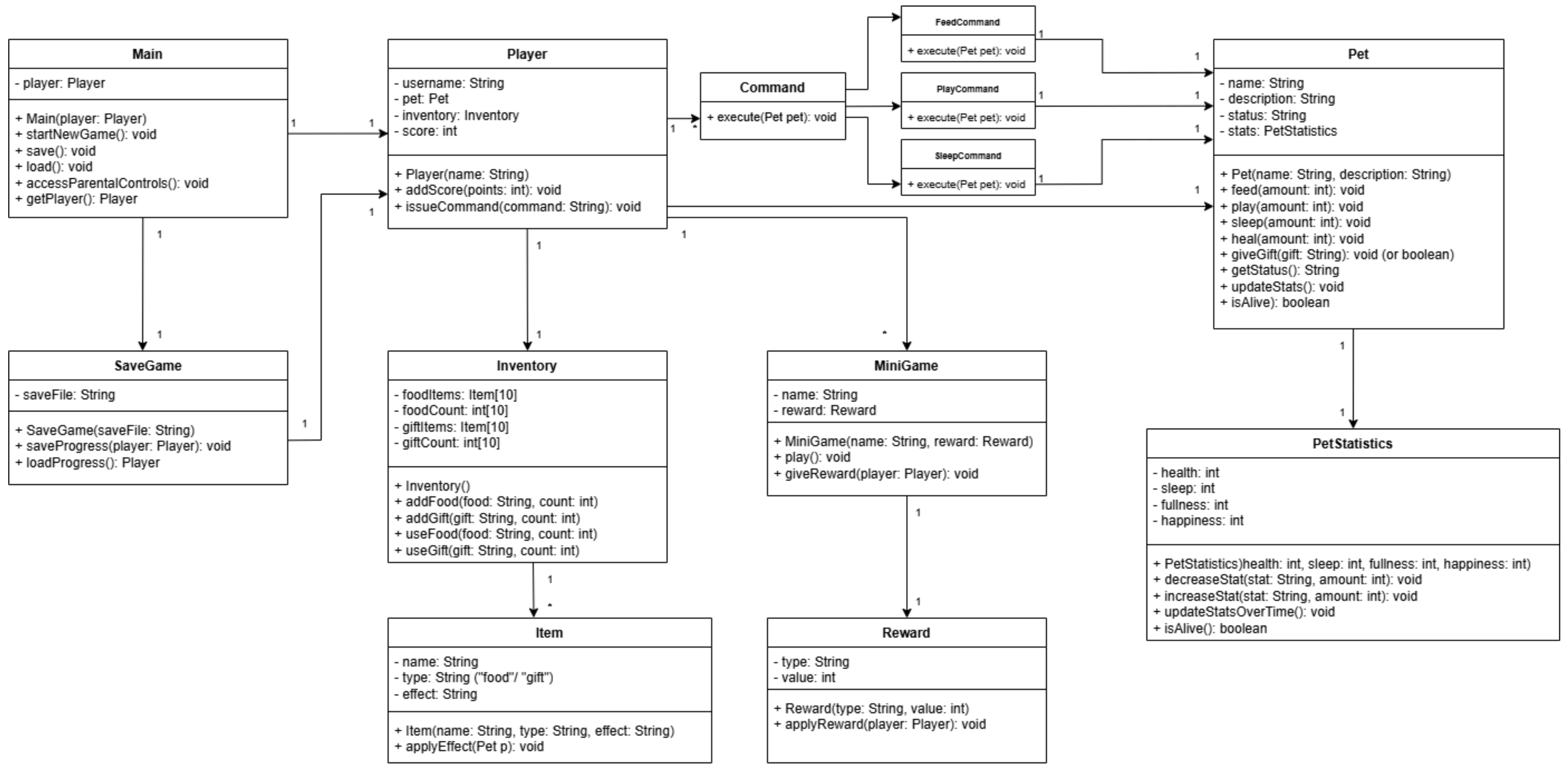
---

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, https://www.khanacademy.org/computing/computer-programming/html-css.
- 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.

| | |
|---|---|
| | ● Servos, Daniel. CS2212: Introduction to Software Engineering, Week 8 Quality concepts. Winter Session 2025, OWL |

## 3. Class Diagrams



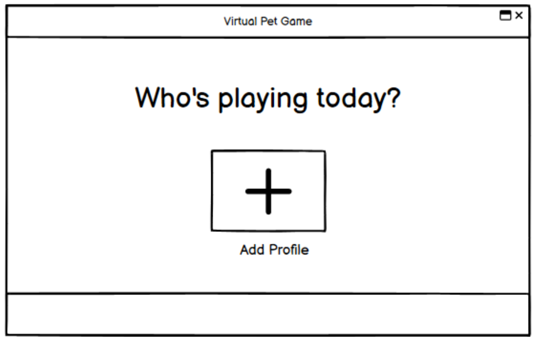
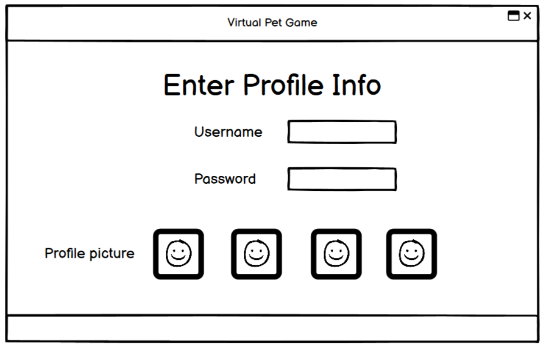
Main
- player: Player
+ Main(player: Player)
+ startNewGame(): void
+ save(): void
+ load(): void
+ accessParentalControls(): void
+ getPlayer(): Player
SaveGame
- saveFile: String
+ SaveGame(saveFile: String)
+ saveProgress(player: Player): void
+ loadProgress(): Player
Player
- username: String
- pet: Pet
- inventory: Inventory
- score: int
+ Player(name: String)
+ addScore(points: int): void
+ issueCommand(command: String): void
Command
+ execute(Pet pet): void
FeedCommand
+ execute(Pet pet): void
PlayCommand
+ execute(Pet pet): void
SleepCommand
+ execute(Pet pet): void
Pet
- name: String
- description: String
- status: String
- stats: PetStatistics
+ Pet(name: String, description: String)
+ feed(amount: int): void
+ play(amount: int): void
+ sleep(amount: int): void
+ heal(amount: int): void
+ giveGift(gift: String): void (or boolean)
+ getStatus(): String
+ updateStats(): void
+ isAlive): boolean
Inventory
- foodItems: Item[10]
- foodCount: int[10]
- giftItems: Item[10]
- giftCount: int[10]
+ Inventory()
+ addFood(food: String, count: int)
+ addGift(gift: String, count: int)
+ useFood(food: String, count: int)
+ useGift(gift: String, count: int)
MiniGame
- name: String
- reward: Reward
+ MiniGame(name: String, reward: Reward)
+ play(): void
+ giveReward(player: Player): void
PetStatistics
- health: int
- sleep: int
- fullness: int
- happiness: int
+ PetStatistics)health: int, sleep: int, fullness: int, happiness: int)
+ decreaseStat(stat: String, amount: int): void
+ increaseStat(stat: String, amount: int): void
+ updateStatsOverTime(): void
+ isAlive(): boolean
Item
- name: String
- type: String ("food"/ "gift")
- effect: String
+ Item(name: String, type: String, effect: String)
+ applyEffect(Pet p): void
Reward
- type: String
- value: int
+ Reward(type: String, value: int)
+ applyReward(player: Player): void
1
*

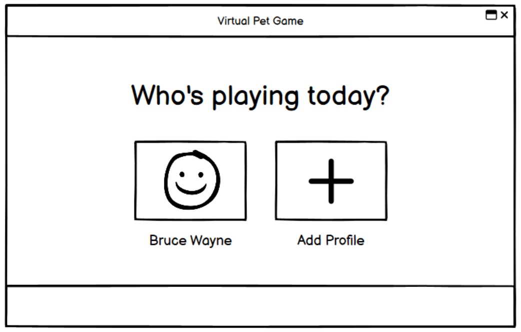
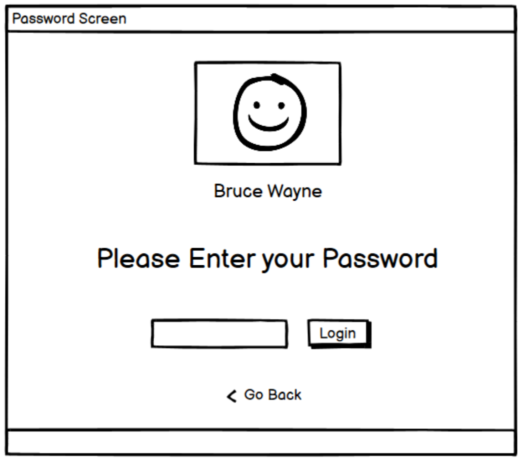
| **Class Name** | **Description** |
|---|---|
| Main | The Main class acts as the core of the program. This class holds the player and uses the SaveGame class to save/load the game. This class allows for new games to be created as well and allows for parental controls to be accessed. This is the main class that will allow the user to interact will all other components of the program. |
| Player | The Player class will represent an object for the player of the game. They will have attributes that include their username, the pet they own, their inventory, and their score. This means they must use the Pet and Inventory class. To interact with the game, this class will use the Command class allowing functionality as well as the MiniGame class that allows for mini-games to be played. |
| SaveGame | The SaveGame class will allow for the game to be saved by interacting with the Player class. It will have a saveFile where the data of the player will be stored. This will be used by the main class to save data for the player. |
| Pet | The Pet class will represent an object for a virtual pet that the game can be played with. This pet will have a name, description, status, and stats as its attributes as well as many methods to interact with the pet (such as feeding and sleeping). This class will use the PetStatistics class to keep track of the pet's stats such as health, sleep levels, fullness, and happiness. |
| Inventory | The inventory class acts as an object for the player's inventory. This allows for the player to hold amounts of different items that they can use in the game. There are a maximum of 10 different items which can have a quantity of up to 10. There are 2 different types of items which are classified as food items, and gift items. They are of type Item which is a class that this class uses. This class has methods to both add and use items for when the player gets more items or uses them on their pet. |
| Item | The item class acts as one of the items used by the inventory class. Items have a name, a type (either food or gift) as well as the effect that that item will have. This class has a method called applyEffect() which applies the effect specific to this item to the given pet. |
| MiniGame | The MiniGame class acts as a mini-game that the player can play. This will have a name and a reward that the player will get for completing the mini-game. There will be methods to both play the mini-game and give the reward to the player. The reward will be a Reward object that this class uses. |

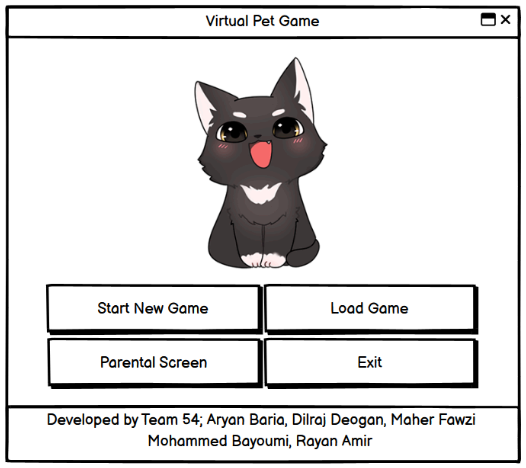
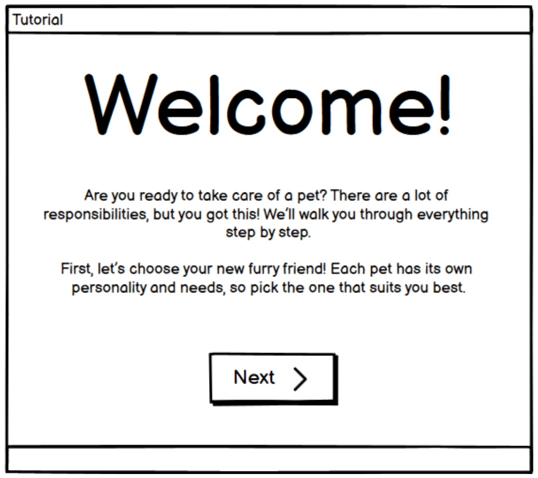
| | |
|---|---|
| Reward | The reward class acts as a reward that can be acquired for completing a mini-game. It will have a type which describes the type of reward as well as a value which shows the effect the reward will have. This class has a method that allows for the reward to be used, granting the value. |
| Command | The command class acts as a command that the player can use to interact with their pet. This class has the subclasses FeedCommand, SleepCommand, and playCommand that will respectively feed the pet, put the pet to sleep and play with the pet. |
| PetStatistics | The PetStatistics class holds the stats for the pet. These include health, sleep, fullness, and happiness. This class have methods to manage stats as well as an isDead() method to check if the pet is dead or not. This is used by the pet class to manage the stats for a pet. |

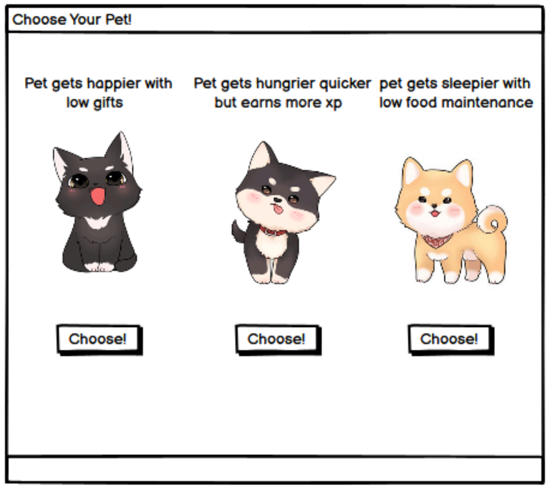
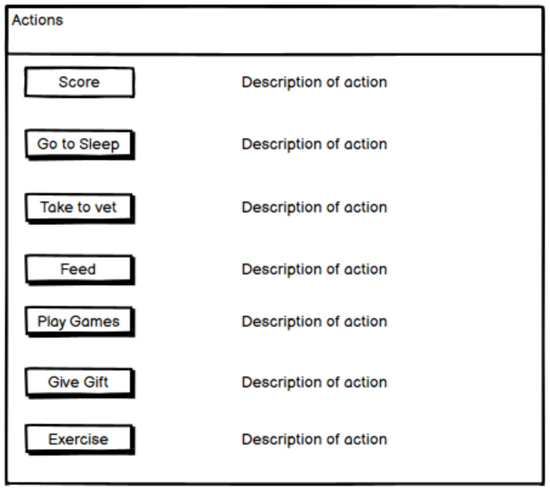
## 4.User Interface Mockup

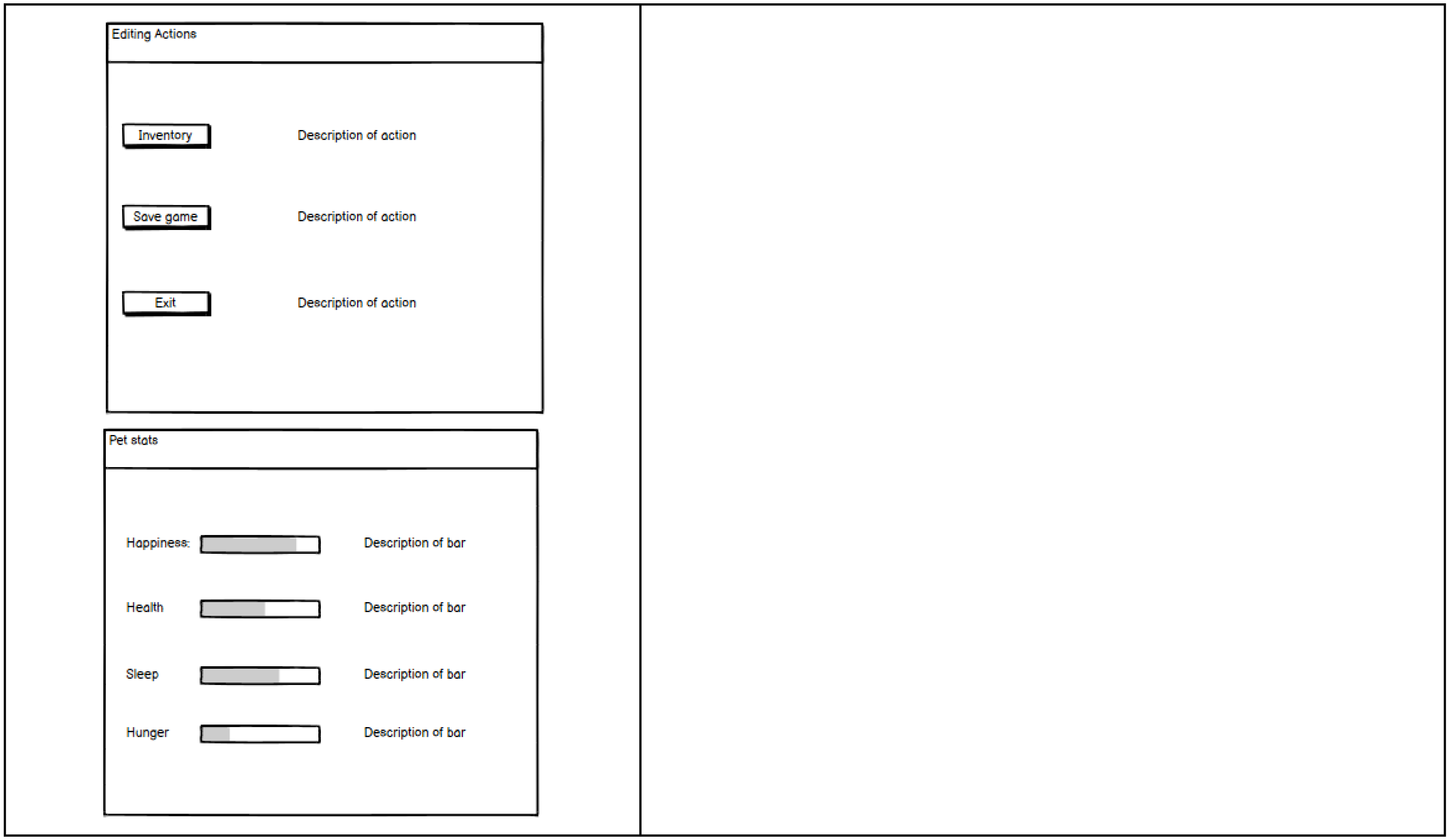
**Initial Startup** **(Purposed idea on having multiple users NOTE: this might be added. We have designed it and made a mock up if we plan to do it. If we do not plan to use it, the user will just be prompted main menu screen - start, load, parental, exit)**

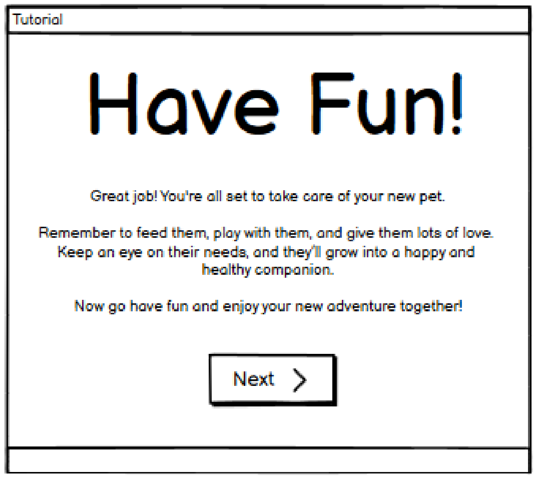
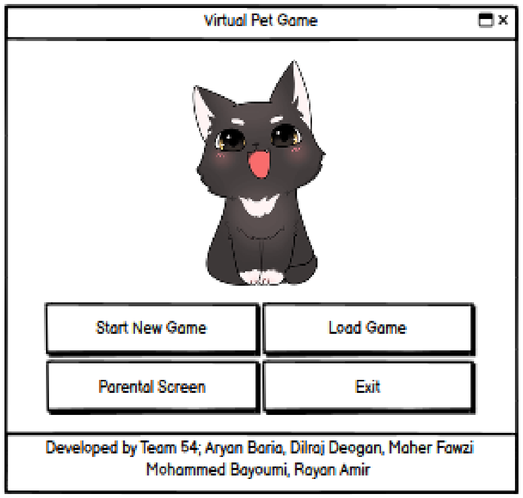
| | |
|---|---|
|  | When the program is first run (in this case the program is run for the first time, clean state) the user will be prompted to add a profile. Adding in this profile will save all their progress, states, customization on that profile only. The user will click the add profile button (this is similar to how netflix does it with how they allow the user to add a profile) after doing so the user will be taken to the next interface, to enter their username, select a password, and select a profile picture. |
|   Virtual Pet Game<br>Enter Profile Info<br>Username<br>Password<br>Profile picture | After the user selects, adds a profile, they will then be prompted to enter a username, password, and select a profile picture so that their profile is unique to themselves. |

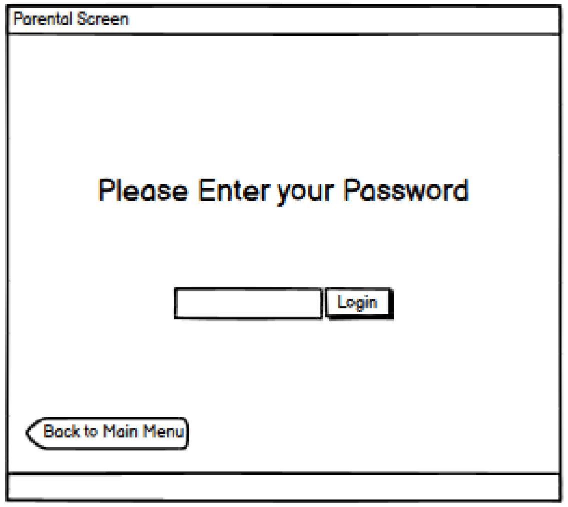
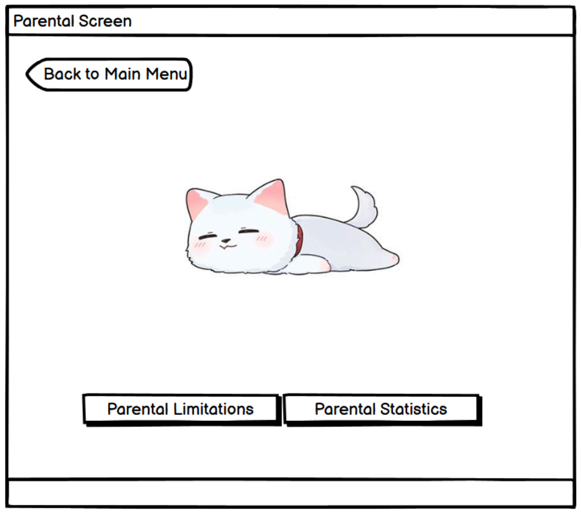
| | |
|---|---|
|  Virtual Pet Game<br>Who's playing today?<br>Bruce Wayne<br>Add Profile  Password Screen<br>Bruce Wayne<br>Please Enter your Password<br>Login<br>Go Back | This is how the interface will look like with an already existing profile. For example if a profile was made, and the user played a bit, or just closed the program, when they open it back up, they will say their profile, in which they can select. When they select, of course, they will be prompted to enter their password, as of course, in this case, it is done for security measures. If the user pressed the wrong profile, then we can allow them to go back to the 'Who's playing today" interface |

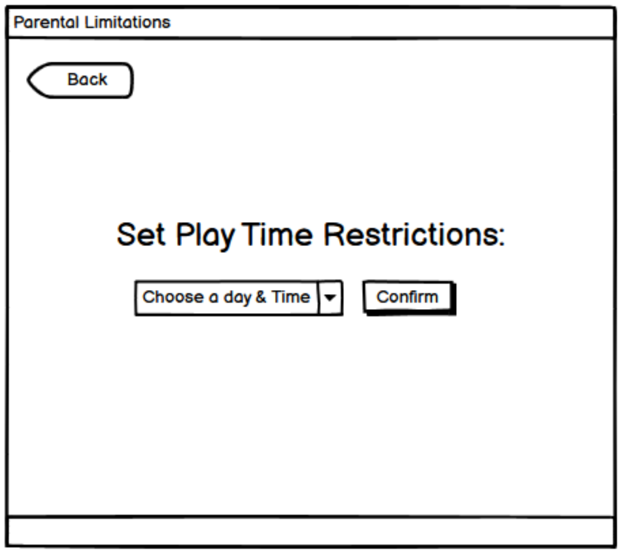
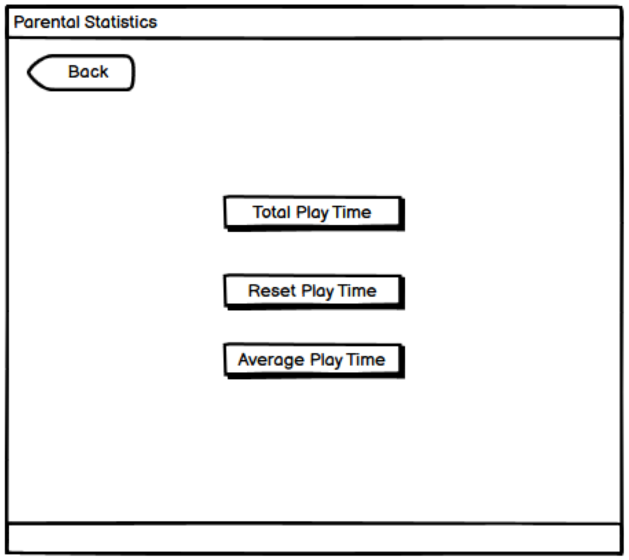
| | |
|---|---|
|  Virtual Pet Game<br>Start New Game<br>Load Game<br>Parental Screen<br>ExitDeveloped by Team 54; Aryan Baria, Dilraj Deogan, Maher Fawzi Mohammed Bayoumi, Rayan Amir | After the user selects the profile, they will be prompted with the main menu screen. Various buttons are presented, so that the user can start a new game, load a new game, access the parental screen, and exit. |
| **Start new Game** | |
|  Welcome!<br>Are you ready to take care of a pet? There are a lot of responsibilities, but you got this! We'll walk you through everything step by step.<br>First, let's choose your new furry friend! Each pet has its own personality and needs, so pick the one that suits you best. | Clicking start a new game, users are welcomed with a tutorial screen designed to introduce them to the game's mechanics so they wouldn't be lost. It will show an indepth step by step description of the game, so that it is easy to understand. |

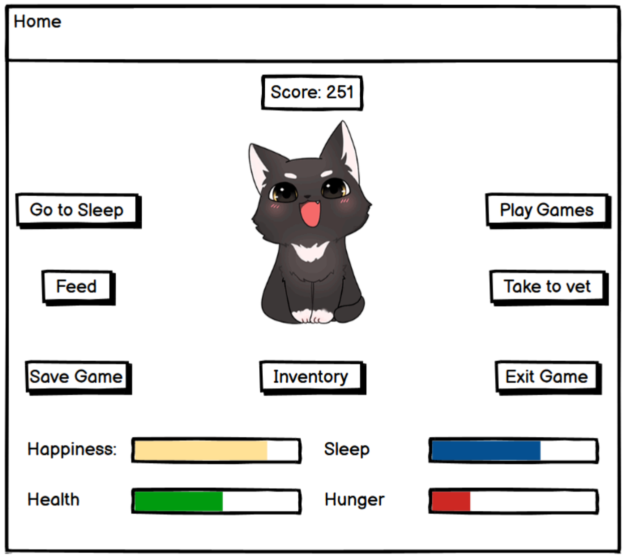
| | |
|---|---|
|  Choose Your Pet!<br>Pet gets happier with low gifts<br>Pet gets hungrier quicker but earns more xp<br>pet gets sleepier with low food maintenance<br>Choose!<br>Choose!<br>Choose! | Users will then be led to a screen where they can pick out their ideal pet. Each pet comes with different statistics. For example one may feel hunger quicker than another or another one may feel happier with less care than another one. |
|  Actions<br>Score<br>Description of action<br>Go to Sleep<br>Description of action<br>Take to vet<br>Description of action<br>Feed<br>Description of action<br>Play Games<br>Description of action<br>Give Gift<br>Description of action<br>Exercise<br>Description of action | The next screen of the tutorial will be shown. It gives a brief description of each action button that could be found on the game's interface making it easier for the player to understand the game mechanics. There are also editing actions, explaining what the user can do in terms of saving the game, in their inventory, and exiting. The user will also be taught on the stats of the pet through our health bar system. Once the user learns, about everything, (in this case the functional aspects) then they can start playing |

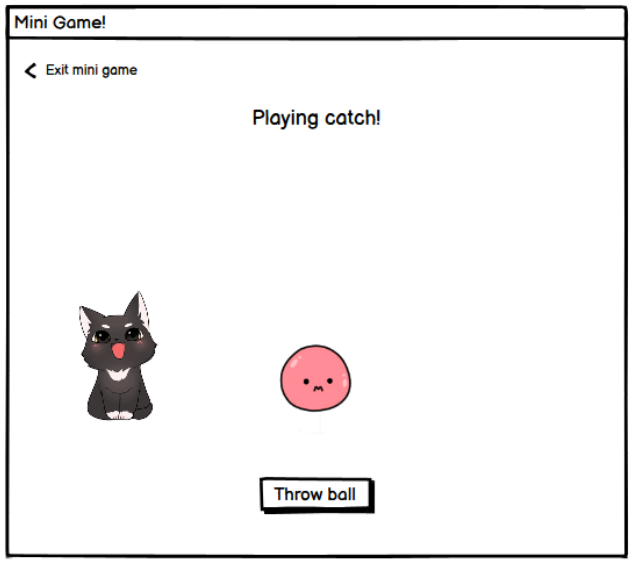
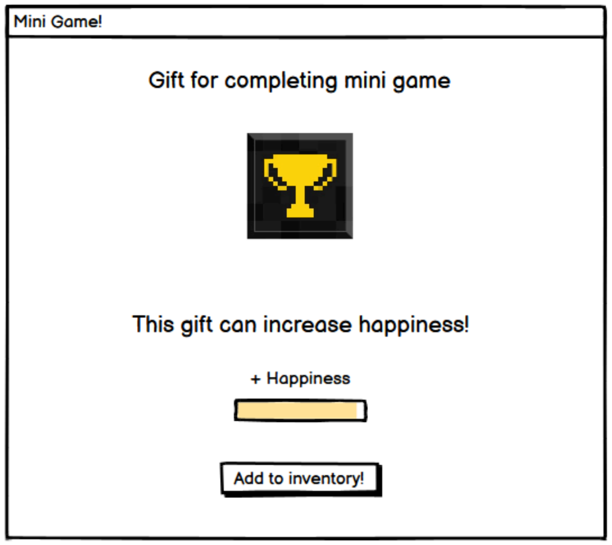

Editing Actions
Inventory
Description of action
Save game
Description of action
Exit
Description of action
Pet stats
Happiness:
Description of bar
Health
Description of bar
Sleep
Description of bar
Hunger
Description of bar

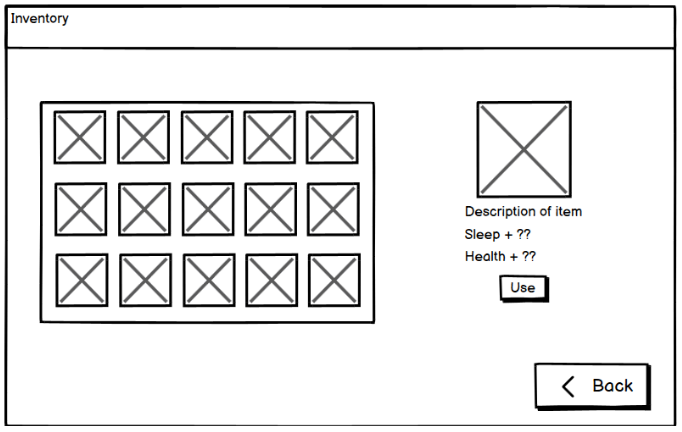
| | |
|---|---|
| Tutorial<br>**Have Fun!**<br>Great job! You're all set to take care of your new pet.<br>Remember to feed them, play with them, and give them lots of love. Keep an eye on their needs, and they'll grow into a happy and healthy companion.<br>Now go have fun and enjoy your new adventure together!<br> | |
| **Parental Settings** | |
|  Virtual Pet Game<br>Start New Game<br>Load Game<br>Parental Screen<br>Exit<br>Developed by Team 54; Aryan Baria, Dilraj Deogan, Maher Fawzi Mohammed Bayoumi, Rayan Amir | From the main menu, parents will be able to access the parental screen panel which would allow them to set certain times of the day where the player is allowed to play the game, view play time statistics, revive pets, etc. If the user has already gone through the tutorial and wants to resume they just go into load game and will take them straight to the main game |

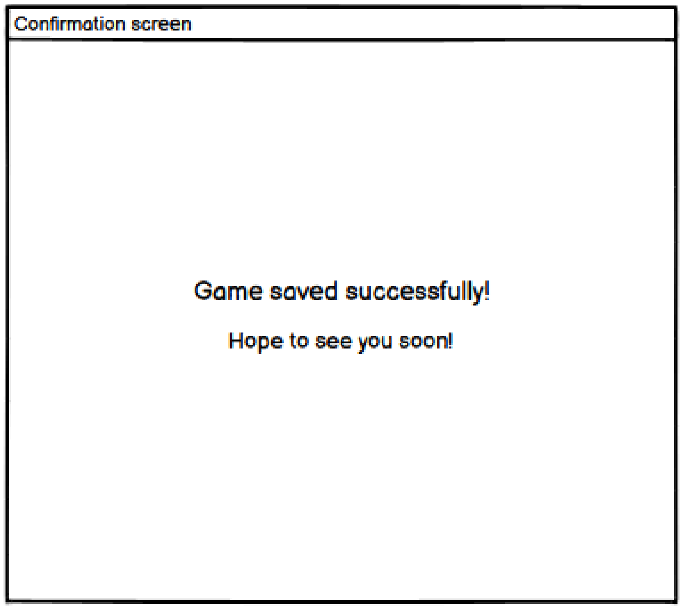
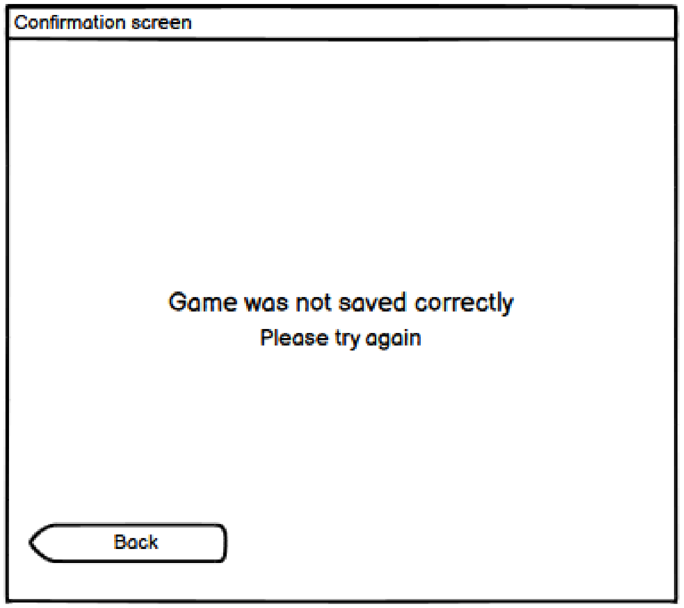
|  |  |
|---|---|
|   Parental Screen<br>Please Enter your Password<br>Login<br>Back to Main Menu | Before being able to access the parental controls, Parents must enter their passwords. |
|   Parental Screen<br>Back to Main Menu<br>Parental Limitations<br>Parental Statistics | Upon entering their passwords, the parents are left with a couple options as previously mentioned such as limitations, statistics, resetting play time, total play time or average play time and once they are done they would be able to return back to the main menu |

| | |
|---|---|
|   Parental Limitations<br>Back<br>Set Play Time Restrictions:<br>Choose a day & Time<br>Confirm | Through the parental screen, the ability to change the play time can be adjusted. The user will select the drop down, and it will be like a scroll type of drop down similar to in iphones. They will also select multiple day(s) and multiple time(s) slots, and once confirmed. The game can not be applied during the selected day(s) and time(s) |
|   Parental Statistics<br>Back<br>Total Play Time<br>Reset Play Time<br>Average Play Time | This window allows the parents to be able to view the player's total, average and are able to reset the play time for the players. |

| **Main game interface** | |
|---|---|
|  Home<br>Score: 251<br>Go to Sleep<br>Play Games<br>Feed<br>Take to vet<br>Save Game<br>Inventory<br>Exit Game<br>Happiness:<br>Sleep<br>Health<br>Hunger | After that user has gone through the tutorial, they will be able to see the main game interface where the game goes on. They can interact with the pet by feeding it, making it go to sleep, playing a mini game,etc. |

| | |
|---|---|
|  Mini Game!<br>Exit mini game<br>Playing catch!<br>Throw ball<br> Mini Game!<br>Gift for completing mini game<br>This gift can increase happiness!<br>+ Happiness<br>Add to inventory! | This would be a game idea where users would be able to play it to receive gifts which would make the pet happier. (exact game has not been set in stone yet). In this case, the user can play catch, for example they would drag the ball somewhere, then press the throw ball, and possibly show an animation of the pet to the ball. After doing so, they will be prompted to the next screen, notifying them that they received a gift/reward for completing the game and after adding a gift to inventory the game will bring them to the main interface screen. We will also show that the pet has increased in some stat, for example happiness. The same mechanics shown here for the game, will be applied to take to vet, feed, go to sleep. |

| | |
|---|---|
|  Inventory<br>Description of item<br>Sleep + ??<br>Health + ??<br>Use<br>Back | The user can view the items they have in their inventory. In the inventory, once they click on an item it will show to the side the description, the stats that it can improve, and a button if the user wants to use it or not. If they use it, then the item would be removed from the inventory creating more space. |
| **Exit** | |

| | |
|---|---|
|  Confirmation screen<br>Game saved successfully!<br>Hope to see you soon!<br> Confirmation screen<br>Game was not saved correctly<br>Please try again<br> | When a user decides to exit the game, they will be shown a confirmation screen explaining whether or not tha game had been saved or not. If the game was successfully saved, the user will be kicked out of the game but if the game was not saved correctly, they will have to go back to the main game interface to manually save the game and try exiting again. If that works, the confirmation screen will show and the user is out of the program. |

## 5. File Formats

| | |
|---|---|
| **Which file format?** | Json as defined in our terms from our requirement documentation, is a Short for JavaScript Object Notation, is a self-describing format of storing and transporting data. So in this case it can work well with object oriented programming and allow us to store structured data. Structured data in that we have player and pet names, scores and stats. JSON will feel more familiar as it will just feel like creating variables and assigning values. |
| **External Libraries** | In terms of which library to use, Jackson looks like to be the most popular. So in this case it will probably fit with our system as well considering all the features and game modes we want to add so it can support more data, as that is what it is known for.<br><br>The Jackson ObjectMapper can also create JSON from Java objects, (explained from: https://jenkov.com/tutorials/java-json/jackson-objectmapper.html) this is important because we can can convert Java objects to JSON and store them in files for saving player progress, pet states, or game settings. We can also convert JSON back into Java objects when loading saved game data. So this will be convenient for us.<br><br>In this case, we can define a more general note:<br>- When saving game progress (convert Java object → JSON)<br>- When loading saved progress (convert JSON → Java object) |
| **Example cases** | We will most likely have 4 json files organizing the flow of the game. Everything in the start will be in setupFile, everything in parental settings will be in parentalSettings, everything in gameplay will be in mainGame, and everything in terms of exiting or saving, will be in exitOrSave |

| Data type | Field Definitions (data types, note, the names are not set in stone, these are just examples of how we can name them): | Example name (camelCase for now, snake_case if we plan to use a database) |
|---|---|---|
| Setting up and | Field Definitions: | setupFile.json |

| | | | |
|---|---|---|---|
| | initial start | profiles: An array containing individual user profiles which will include the following:<br>- id: Unique identifier for the profile.<br>- username: The user's chosen name.<br>- password: An encrypted string.<br>- profilePicture: A URL or file path to the profile image.<br><br>- pet: An object with initial pet details.<br>    - petId: Unique pet identifier.<br>    - petName: Name of the pet.<br>    - petType: Type (description).<br>    - stats: Object containing pet's attributes (starting health, happiness, hunger).<br><br>- progress: Data tracking current game progress (tutorial status, current screen). | |
| | Parental settings | parentalControls: Main object for parental settings, which will include the following:<br><br>- parentPassword: Encrypted parental password<br><br>- playTimeRestrictions: An array where each entry specifies:<br>    - day: Day of the week (Monday)<br>    - Time: Start time (format "HH:MM").<br><br>- statistics: Object containing play time:<br>    - totalPlayTime: Cumulative play time.<br>    - averagePlayTime: Average session duration<br>    - resetPlayTime: reset play time<br><br>- actions: Additional settings like pet revival | parentalSettingsFile.json |

| | | |
|---|---|---|
| | option | |
| Main game data | gameSession: Object for the ongoing game state, which includes the following:<br>- currentPetState: Object with the pet’s current attributes.<br>  - health: Current health level.<br>  - happiness: Current happiness level.<br>  - sleep: Current energy or sleep level.<br>  - hunger: Current hunger level.<br><br>- actionHistory: Array of objects logging actions taken by the user.<br>  - actionName: Name of the action (feed, goToSleep, takeToVet).<br>  - timestamp: The time when the action was performed.<br>  - effects: Object detailing the changes to pet stats (hunger -20, happiness +5).<br><br>- availableActions: Object listing the available actions and their parameters.<br>  - feed: Object describing the feed action.<br>    - description: Brief description of what feeding does.<br>    - cooldown: Time (in seconds or minutes) before the action can be used again.<br>    - effects: Object specifying how feeding affects pet stats (hunger -20, happiness +5).<br>  - goToSleep: Object describing the sleep action.<br>    - description: Brief description of | mainGame.json |

| | | | |
|---|---|---|---|
| | | what going to sleep does.<br>- cooldown: Cooldown period for the sleep action.<br>- effects: Object specifying how sleep improves pet stats (energy +30).<br>- takeToVet: Object describing the vet visit action.<br>  - description: Brief description of how a vet visit improves pet health.<br>  - cooldown: Cooldown period for vet visits.<br>  - effects: Object specifying the impact on pet stats (health +15).<br><br>- NOTE: Reward/food may be earned<br>  - item: Food or reward<br><br>- inventory: An array of items.<br>  - itemId: Unique identifier for the item.<br>  - itemName: Name of the item.<br>  - description: Details about the item.<br>  - effects: Object listing how the item affects pet stats (health +10).<br><br>- play: Object that tracks game progress (scores, last played times).<br>- gameProgress: Object with overall progress such as current level or total play time | |
| | Exit/Saving Data | - saveStatus: Indicates whether the save was successful ("success", "failure").<br>- lastSaveTime: Timestamp of the last successful save. | exitOrSaving.json |

| | |
|---|---|
| | <table><tr><td></td><td>- exitConfirmation: Boolean indicating if exit was confirmed.<br>- errorMessage: Optional field for error details if saving failed.</td><td></td></tr></table> |

| 6. Development Environment | |
|---|---|
| **IDE** | The main IDE that will be used for this project will be Visual Studio Code. Virtual Studio code allows users to download extensions from its marketplace type of section. So in this case we can download java compilers and java libraries from there. We will use JavaDoc to document all of our code. The same goes with JavaDoc in terms of downloading an extension from the built in marketplace. |
| **Tools** | All of our code will be uploaded on the GitLab, while also doing push and pull changes. We will be using JUnit 5 to test our code and make sure there are no errors and try to use Apache Maven as well. Apache Maven is a build automation tool for Java, similar to how live server automates refreshing web pages for HTML/CSS. While Live Server reloads the website in real time, Maven automates compiling, testing, and packaging Java code, so in this case we can be more efficient. We will also be using the GitLab issues to help us stay organized and on top of each task we do as a team. |
| **External Libraries** | Java swing will be used of course to design and build the GUI, it already comes standard in the java standard library. More specifically it will help build the windows, buttons, text fields, functionality, themes (background, colour etc.). Jackson will also be used to help us map java objects to Json. |

## 7. Patterns

| Pattern | Discussion | Reference (URL) | Why is it appropriate? | How will it be applied? |
|---|---|---|---|---|
| Observer Pattern (Object-Oriented Pattern) | The Observer pattern allows for objects to automatically update when another object changes. In our virtual pet game, the UI should update whenever the pet's stats change (e.g. hunger, happiness, energy, sleep) | https://sourcemaking.com/design_patterns/observer | ● The game UI should automatically reflect the changes in pet status without manually checking<br>● If the pet's hunger reaches 0, the UI should show an alert immediately | ● The pet class will act as the Observable and store the pet's stats (hunger, happiness, energy, sleep)<br>● The GameUI class implements the Observer interface and updates the UI automatically when the pets stats change<br>● When Pet.setHunger() or Pet.setHappiness() is called, all observers notified (including GameUI) to refresh the UI |
| Command Pattern | The command pattern allows for the details of a command to be encapsulated as an object. This allows for parameterizing objects that have different requests, queueing, and allowing for | https://www.oodesign.com/command-pattern | ● New actions for the pet (such as giving gifts or taking it to the vet) can be implemented easily without modifying previous code.<br>● Allows for better control | ● Each command that can be issued will be encapsulated in its own command object that will be invoked by the player and Received by the pet. |

| | | | | |
|---|---|---|---|---|
| | undoing of requests. | | over tha game state and interactions as commands can be queued/undone. | <ul><li>This allows the player to used commands that will then be received by the pet, allowing for it to respond according.</li><li>The system can view a history of the commands allowing for actions to be undone.</li></ul> |
| Model-View-Controller (MVC) (Architectural Pattern) | Divides the application into Model (Data), View (UI), and Controller (Logic). This is done in order to make it easier to modify and expand game components | https://www.geeksforgeeks.org/mvc-design-pattern/ | <ul><li>Helps to maintain clear separation between the UI and logic</li><li>Allows for future improvements to the UI without changing the game logic</li></ul> | <ul><li>Pet stores stats like hunger, happiness, and energy (Model)</li><li>GameUI displays the pet animations and menus (View)</li><li>GameController handles the player interactions (Controller)</li></ul> |

## 9. Summary

Our Virtual Pet Game is a project designed with extreme thought, from technical aspects to user friendly interfaces, we have covered it all! This document specifically provides a structured and detailed overview of the architecture and class relationships within the Virtual Pet Game. We start off with an introduction outlining the inspiration behind virtual pet games, while referencing well known titles. The project objectives foys on implementing pet interaction mechanics, developing a maintainable game using Java, designing an intuitive graphical user interface, and adhering to best practices in software engineering to ensure efficiency and scalability. A key component of the documentation is the UML class diagram, this models the systems core relationships, which include Player → Pet → PetStats for managing pet attributes, Inventory → Item for a structured item storage, MiniGame → Reward for enhancing gameplay engagement, and the Command Pattern for ensuring player actions such as feeding, playing, and sleeping. Additionally, the SaveGame class ensures data persistence, allowing players to resume progress seamlessly. These class diagrams and structures object-oriented design enhance modularity, allowing future development much easier to implement.

Throughout this document, we have also included User Interface Mockups and Wireframes allowing us to illustrate the game's flow. The UI design ensures an engaging aspect, making players want to keep playing. We cover the startup screen, where users create and manage profiles, the main menu which gives access to game features, and the main gameplay interface, where users interact with their pet. Other UI elements include the parental control panel, which allows parents to set playtime restrictions and track game statistics, and the inventory system allowing users to manage food and gift items. The mockups and interface plans prioritize usability and accessibility, ensuring a smooth player experience.

As for data storage and file handling the documentation defines JSON as the primary file format, and this allows structured and accessible game data. Separate JSON files store player data, pet data, game progress, and mini game settings, this makes sure that player progress and pet status are consistently saved and loaded.

All in all, this document provides a clear and detailed blueprint for the development of the game, covering the system architecture, interface design, data handling, development tools and key software principles. The design ensures a scalable and interactive structure, allowing the best results possible This documentation serves as an essential reference for developers, ensuring the game is built with efficiency, structure and scalability.

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

| Terms, Notations, Acronyms | Description |
|---|---|
| UML | Unified Modeling Language is a way to visually depict various aspects of the overall design of a |

| | |
|---|---|
| | solution. |
| 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. |
| Jackson library | Jackson can parse JSON from a string, stream or file, and create a Java object or object graph representing the parsed JSON |
| 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 |