Requirements Document

2.1 Main Page

Task	Contributors	Description
Team Contract	Medha, Priya, Shikha, Vidhi, Yuchen	Specify the duties and expectations of the team
Domain Analysis	Priya, Shikha	Software domain and target audience research
Functional Requirements	Priya, Shikha	Creating use cases and enumerating necessary features
Scenario Models	Priya	Making UML diagrams (use case diagrams and activity diagrams), actors, and use cases
Non-Functional Requirements	Priya, Shikha	Specifying quality standards and system limitations
Final Review	Medha, Priya, Shikha, Vidhi, Yuchen	Editing and completing the documentation

2.2 Introduction

Overview

Virtual pet games offer players an interactive experience that mirrors the responsibilities of real pet ownership. These digital companions require **feeding, grooming, and interaction**, fostering a sense of care and accountability. Titles like **Tamagotchi, Neopets, Nintendogs, and Bitzee** have demonstrated the appeal of virtual pets by engaging users through repetitive yet rewarding mechanics that enjoyably promote **routine management and responsibility**.

For this project, we are developing a **Java-based virtual pet game** that enables players to nurture a digital pet by managing its **hunger, happiness, and health**. The **Graphical User Interface (GUI)** will facilitate seamless interaction, making it easy for players to monitor and respond to their pet's needs. The game will implement a **time-based system** that gradually decreases the pet's well-being, encouraging players to return regularly and maintain their pet's care.

One of the game's core features is **pet customization**, allowing players to **select a species, name their pet, and interact with its unique traits**. The primary gameplay revolves around maintaining the pet's well-being through **feeding, playing, grooming, and resting**. Additional features like **mini-games, achievements, and unlockable items** enhance player engagement by offering rewards for consistent care.

Beyond entertainment, this project integrates fundamental software development principles by requiring **structured programming, real-time interaction management, and data persistence** through a **save/load feature**. The ultimate goal is to create an engaging, interactive, and educational simulation that **reinforces the importance of responsibility, consistency, and routine care** while demonstrating proficiency in Java programming and software engineering best practices.

Concept and Game Mechanisms

Our virtual pet game will offer an immersive and dynamic pet care experience, focusing on interaction, responsibility, and long-term engagement. The key features include:

1. Pet Customization & Unique Personalities

- Players can choose from various pet species, each with distinct traits, behaviours, and preferences.
- Customization options such as pet names, colours, and accessories will allow players to personalize their companions.
- Pets will have unique personality types (e.g., playful, lazy, energetic, shy) that will influence their reactions and interactions.

2. Dynamic Needs Management System

- The pet's well-being is determined by the following core attributes: **hunger, happiness, and health.**
- Players must regularly feed, groom, and play with their pets to maintain their overall being.
- Neglecting the pet will lead to gradual declines in mood and health, reinforcing responsibility.

3. Interactive Engagement Activities

- Players can engage in various activities such as:
  - **Feeding:** Provide different types of food with unique effects on the pet's health and mood.
  - **Playing:** Mini-games and interactive toys boost happiness and energy levels.
  - **Grooming:** Keeping the pet clean prevents sickness and enhances bonding.
  - **Resting:** Ensuring proper sleep restores the pet's energy and improves well-being.

4. Real-Time Progression & Time-Based Interaction

- The game operates on a real-time system where stats decrease gradually, encouraging regular check-ins.
- Neglecting the pet for extended periods results in visible mood changes and declining health.
- Players are rewarded for consistent care through experience points, levelling up, and unlocking new features.

5. Leveling, Rewards, and Achievements

- A **progression system** allows pets to level up over time, unlocking new items, interactions, and abilities.
- Completing activities earns in-game rewards, such as special food, toys, and accessories.
- Achievement badges recognize milestones like “100 Days of Care” or “Master Groomer,” adding long-term goals.

6. User-Friendly Graphical Interface

- A simple yet visually appealing **Graphical User Interface (GUI)** enhances user experience.
- The interface provides real-time updates on pet’s stats, upcoming needs, and interaction prompts.
- Animated pet reactions make the game feel more engaging and immersive.

7. Save and Load Functionality

- Players can **save their progress** and continue their pet care journey over multiple sessions.
- The pet’s condition stays persistent, reinforcing accountability and a long-term connection with the virtual companion.

By integrating these elements, the game delivers a fun, engaging, and educational experience that teaches responsibility while keeping players entertained.

Objectives

Project Objectives

The primary objective of this project is to develop a functional and engaging **Java virtual pet game**, showcasing **software engineering and game development skills**. This project provides hands-on experience in:

- **Interpreting and implementing** a detailed project specification.
- **Designing and structuring** a Java-based interactive application.
- **Creating a GUI** that facilitates user interaction with the pet.
- **Writing clean, efficient, and well-documented Java code** following best practices.
- **Implementing a progression system** that enhances long-term player engagement.
- **Collaborating in a team environment**, distributing tasks, and solving problems effectively.
- **Reflecting on design decisions**, evaluating their effectiveness, and refining the project as needed.

Game Objectives

The virtual pet game aims to provide players with a **fun and interactive way to experience pet ownership** while reinforcing responsibility and routine management. The core gameplay revolves around **maintaining the pet's well-being** through **feeding, playing, grooming, and resting**. Key features include:

- **Time-based stat management**: The pet's hunger, happiness, and health decrease over time, requiring regular attention.
- **Progression mechanics**: Players are rewarded for consistent care through unlockable items, achievements, and potential pet evolution or skill upgrades.
- **Mini-games and interactions**: Engaging activities allow players to bond with their pets and earn in-game rewards.
- **Save/load functionality**: Players can resume their progress, ensuring continuity in their pet's care.

By integrating these mechanics, the game fosters **long-term engagement** and teaches players that **small, consistent actions can have a meaningful impact** on a pet's well-being. The combination of structured software development and engaging gameplay ensures that the project fulfills both technical and entertainment objectives.

References

“smartdraw.” *smartdraw*,
https://www.smartdraw.com/?id=104640&gad_source=1&gclid=CjwKCAiA2JG9BhAuEiwAH_zf3r5bFapCyI9erKSuoxJ3kB7V122TBf6vKF3EDaLh1FLr7G2TOIhMeBoC_jgQAvD_BwE. Accessed 2 February 2025.

Project specification document

Class notes

2.3 Domain Analysis

Our Primary Domain is Application Software (since it is a user-facing interactive program) and our **subdomain is Game Development Software** (as it involves creating an interactive game specifically **simulation games** that focus on virtual pet care).

Our target audience for the game is primarily **children and young teens (5+ years old)** who enjoy interactive, low-stakes and stress-free games. **Casual gamers and pet lovers** who want a fun and engaging experience. Possibly **parents** looking for a light educational tool for their children. And possibly people who are looking into adopting pets in real life but want to have their first experience in the virtual world. **Individuals interested in digital companionship**, including people who may not be able to own real pets due to restrictions (e.g., allergies, housing rules

The Subdomain is Game Development Software as it involves creating an interactive game and the **Category is Simulation Games / Virtual Pet Games** since it mimics real-life pet care through digital interactions with a focus on **life simulation and pet-care management**. It is also a **time-based simulation** where player actions influence the state of a virtual entity over time.

Virtual pet games have been a popular genre since the late 1990s, with titles like **Tamagotchi and Neopets** leading the way. Initially featuring **simple, monochrome pixel creatures**, these games have evolved to include **intricate animations, artificial intelligence, and physics-based interactions**.

Key aspects of this domain include:

- **Daily Interaction Mechanics** – Players engage in activities such as **feeding, cleaning, and playing**, reinforcing routine management.
- **Progression Systems** – Unlockable **accessories, levels, and achievements** keep players engaged over time.
- **AI-Driven Behavior** – Pets respond dynamically to **care and neglect**, simulating real-life emotional connections.
- **Minimal Time Commitment** – Designed for **quick, periodic check-ins**, making them accessible without requiring extensive playtime.
- **Emotional Bonding** – Players often develop **attachments to their virtual pets**, enhancing long-term engagement.
- **Graphical User Interface (GUI) Importance** – A well-designed GUI is **essential for clear feedback** on the pet's stats, actions, and needs.

Overall, the appeal of virtual pet games lies in their **simple yet rewarding mechanics**, where **consistent care leads to progression and stronger emotional connections**. Their design encourages **frequent, low-effort interactions**, making them ideal for **casual gamers, children, and pet lovers**.

The most common problems/ issues encountered in this domain are:

- **Player Engagement Drops Over Time & Retention** – Due to tedious, Many virtual pet games lose engagement as interactions become repetitive.
- **Balancing Challenge and Fun** – If pets' needs decline too quickly, the game feels punishing; if too slow, it feels unchallenging.
- **Achieving a balance** – between realism and playability is important since too hard of a game might irritate players, while too easy of a game can make it seem uninteresting.
- **User Interface & Accessibility** – Inexperienced players or those who are not familiar with gaming mechanics may find interactions complicated due to poor UI/UX design.

To address the common problems, we come up with these few common solutions:

Dynamic Content: Regularly update the game with new pets, environments, tasks, and rewards to keep it fresh and engaging.

Reward Systems: Implement progressive reward systems to incentivize players with new achievements or items that can be unlocked as they progress.

Social Features: Allow players to interact with each other, like sharing pets or competing in challenges. This social interaction boosts engagement.

Personalization: Let players personalize their pets and environments, which creates a stronger bond and investment in the game.

Adaptive Difficulty: Introduce an algorithm that adjusts the challenge based on the player's skill level or progress. For example, if a player is progressing quickly, the game could increase the challenge or speed up the pets' needs.

Time-Based Scaling: Gradually scale the decline of pet needs over time, allowing for intervals of relaxation in the gameplay.

Player Feedback Loop: Provide players with visual indicators of their pet's needs and moods, so they can better manage the challenge of care while still enjoying the game.

Player Customization: Allow players to adjust settings related to pet care difficulty. For instance, if they want a more relaxed experience, they can lower the frequency of pet needs or make the satisfaction of those needs easier.

Gradual Learning Curve: Introduce pet care mechanics slowly, allowing players to master one aspect of care before introducing more complex needs.

Visual/Audio Feedback: Provide clear visual or audio cues(notifications) that highlight when a task is urgent or can be delayed. This prevents frustration and enhances immersion.

Simplified UI/UX: Ensure that the UI is clean and intuitive. Use simple icons and clear labels for common actions like feeding, playing, or cleaning the pet.

Tutorials and Tooltips: Include tutorials at the start and tooltips or on-screen hints for new players to learn mechanics step-by-step.

Accessibility Options: Offer features like larger text, colorblind modes, and customizable controls to help players who may have different needs or preferences.

Responsive Design: Ensure that the interface adjusts well to different screen sizes, especially for mobile devices, making it accessible for players across various platforms.

2.4 Functional Requirements

A list of the Virtual Pet Game's functional requirements is provided below:

- **User authentication:** Users must be able to safely register, log in, and log out of the system.
- **Game Setup:** The ability to launch a new game, choose a level of difficulty, and alter game parameters must be available to users.
- **Game mechanics:** The system should effectively manage player turns, scoring, and game rules. The ability for players to store their progress and reload a previously saved game is known as the "Save & Load" feature.
- **Leaderboard & Scores:** The system ought to keep track of a leaderboard that shows the best players.
- **Settings & Preferences:** The controls, display, and sound must all be customizable by users.
- **Error Handling & Alerts:** In the event of a system failure or invalid input, the game ought to show error messages.
- **Creation and Personalisation of Pets:** Users can design a virtual pet by selecting its species, name, and look. Throughout game sessions, the pet's attributes will be saved and maintained.
- **Pet Status and Requirements:** The pet possesses qualities including health, vitality, happiness, and hunger. As time passes, these qualities deteriorate, necessitating player engagement.
- **Providing the Pet with Food:** The pet can be fed a variety of foods. Distinct food types have distinct effects on a pet's hunger and overall health.
- **Engaging in Play with the Pet:** To make the pet happier, users can participate in a variety of mini-games or activities.
- **Pets Resting and Sleeping:** To recover energy, the pet can sleep. The pet can't do some things while it's sleeping.
- **Evolution and Ageing of Pets:** The pet develops over time and may change depending on the level of care given.
- **System of Health and Illness:** Neglected pets may get sick and need medication or extra attention.
- **Game State Saving and Loading:** Users should be able to store and load their progress in the game.
- **Interface for Graphical Users (GUI):** The game will have an easy-to-use graphical user interface with pet interaction animations.
- **Multiplayer Pet Interaction:** Through a local or internet connection, users can visit and engage with friends' pets. Gamers can compete in minigames, play together, or send presents.

Scenario model

➔ Actors

Table 2.4.1 Actor: Player

Actor	Player
Description	A Human player who uses the game’s interface to communicate with the pet
Aliases	User
Inherits	None
Actor Type	Person
Active/Passive	Active

Table 2.4.2 Actor: Virtual Pet

Actor	Virtual Pet
Description	The user interacts with an AI-controlled pet
Aliases	Pet, Companion
Inherits	None
Actor Type	System
Active/Passive	Passive

Table 2.4.3 Actor: Game System

Actor	Game system
Description	Handles saved data, responds to user interaction, and updates pet status
Aliases	None
Inherits	None
Actor Type	System
Active/Passive	Passive

Table 2.4.4 Actor: Multiplayer

Actor	Multiplayer
Description	Enables users to communicate and engage with the pets of other gamers
Aliases	Online Service
Inherits	None
Actor Type	System
Active/Passive	Passive

Table 2.4.5 Actor: Database

Actor	Database
Description	Stores game stats, scores, and player information
Aliases	None
Inherits	None
Actor Type	External System
Active/Passive	Passive

➔ Use Cases

Table 2.4.6 Use Case 1: User Register

Name	User Register
Primary Actor	User
Secondary Actors	Authentication System
Goal in Context	To play the game, the user must first establish an account.
Preconditions	There must be no existing accounts for the user.
Trigger	‘Register’ is the option the user chooses from the login screen.
Scenario	1. The user inputs their password, email address, and username. 2. The details are verified by the system. 3. The system verifies registration and generates an account.
Alternatives	The user has the option to terminate the registration process.
Exceptions	Invalid email address, weak password, or network outage.
Priority	High

Table 2.4.7 Use Case 2: User Login

Name	User Login
Primary Actor	User
Secondary Actors	Authentication System
Goal in Context	The user opens their account and logs in.
Preconditions	The user needs to already have an account.
Trigger	The user inputs login information.
Scenario	1. The user inputs their password and username. 2. The credentials are validated by the system. 3. The game is made available to the user.
Alternatives	None
Exceptions	A server error, locked account, or incorrect credentials.
Priority	High

Table 2.4.8 Use Case 3: User Logout

Name	User Logout
Primary Actor	User
Secondary Actors	Authentication System
Goal in Context	The user logs out securely
Preconditions	The user must be logged in
Trigger	The user selects ‘Logout’ from the menu
Scenario	1. The user clicks the logout button 2. The system logs out the user and returns them to the login screen
Alternatives	Automatic logout due to inactivity, forced logout, logout with save prompt
Exceptions	Network failure during logout, session already expired, unexpected system crash
Priority	High

Table 2.4.9 Use Case 4: Starting a New Game

Name	Starting a new game
Primary Actor	Player
Secondary Actors	AI Opponent, Game server
Goal in Context	The player chooses the game's parameters and difficulty setting when they begin a new playing session
Preconditions	The user needs to have access to the system and have the game installed
Trigger	The main menu's 'New Game' option is chosen by the player
Scenario	1. The player launches the game 2. The player navigates to the 'New Game' menu 3. The player selects the game mode and difficulty 4. The game initializes the setting and starts the gameplay
Alternatives	None
Exceptions	Error message displayed if game files are missing
Priority	High

Table 2.4.10 Use Case 5: Choosing Difficulty Level

Name	Choosing Difficulty Level
Primary Actor	Player
Secondary Actors	Game setting system
Goal in Context	The player selects the difficulty level
Preconditions	The settings menu or game setup screen must be visible to the player. The game can't be played.
Trigger	The player navigates to the difficulty selection screen
Scenario	1. The user selects a difficulty (Easy, Normal, Hard) 2. The game applies the difficulty settings
Alternatives	If the player does not select a difficulty, the system applies a default difficulty
Exceptions	Invalid selection, Game already in progress, system failure or settings not applied
Priority	Medium

Table 2.4.11 Use Case 6: Player Making Move

Name	Player making move
Primary Actor	Player
Secondary Actors	AI Opponent
Goal in Context	The system verifies and performs the action that the player chooses
Preconditions	The game needs to be operational and capable of receiving user input
Trigger	During their turn, the player chooses a move
Scenario	1. The player inputs a move 2. The system validates the move 3. If valid, the move is executed and the game state updates. 4. If invalid, an error message is displayed
Alternatives	If multiplayer, the move is sent to the game server
Exceptions	The game pauses or switches to offline mode if the network connectivity is lost
Priority	Highest

Table 2.4.12 Use Case 7: Creating Virtual Pet

Name	Create Virtual Pet
Primary Actor	Player
Secondary Actors	The player customizes their pet
Goal in Context	The species, name, and look of the virtual pet are all customizable by the user. For the player to interact with the pet profile while playing, the system saves it.
Preconditions	The player must be signed into their account. Either during a fresh gaming session or as part of the tutorial, the game needs to be in the pet creation phase.
Trigger	By choosing the ‘Create Pet’ option or beginning a new game, the player starts the pet creation procedure.
Scenario	1. The player selects a pet species, name and appearance 2. The system saves the pet's profile
Alternatives	Randomized pet creation, predefined pet selection
Exceptions	Invalid input for name, customization not saved, selection timeout
Priority	High

Table 2.4.13 Use Case 8: Feed the Pet

Name	Feed the pet
Primary Actor	User
Secondary Actors	Virtual Pet
Goal in Context	The pet's hunger and health are impacted by the food the user chooses to feed it
Preconditions	The pet is neither sick nor asleep. Food items are available to the user
Trigger	The user chooses 'feed' from the menu
Scenario	1. The 'feed' option is chosen by the user 2. The system shows the available food items 3. A food item is chosen by the user 4. The system adjusts the pet's health and hunger levels appropriately 5. The status is updated
Alternatives	The system recommends giving medicine if the pet is unwell
Exceptions	If there is no food available, show a message saying so
Priority	High

Table 2.4.14 Use Case 9: Play with the Pet

Name	Play with the pet
Primary Actor	User
Secondary Actors	Virtual pet
Goal in Context	To make the pet happy, the user and the pet participate in an interactive activity
Preconditions	The pet is active and not overly worn out
Trigger	The user chooses the ‘play’ option
Scenario	1. The user chooses ‘play’ from the menu 2. The system displays the available play activities 3. The user chooses a task 4. The pet is animated and its happiness level is updated by the system
Alternatives	The system recommends resting if the pet is exhausted
Exceptions	None
Priority	High

Table 2.4.15 Use Case 10: Pet Needs

Name	Pet Needs
Primary Actor	System
Secondary Actors	The pet’s needs change over time
Goal in Context	Over time, the system regularly updates the pet's status, decreasing its levels of hunger, health, and energy. To keep the pet healthy, the player must take action.
Preconditions	The game requires that a virtual pet be created and active beforehand. You have to have the gaming session open.
Trigger	Based on the evolution of in-game time, the system changes the pet's state regularly.
Scenario	1. Time passes 2. The pet's energy, health, and appetite all decline. 3. The player needs to do something.
Alternatives	Paused or Inactive state, Automatic Basic Care
Exceptions	The game is paused, the pet’s needs reach critical levels, player ignores alerts for too long
Priority	High

Table 2.4.16 Use Case 11: Pet Sleeping

Name	Pet sleeping
Primary Actor	Player
Secondary Actors	System
Goal in Context	The pet regains energy by sleeping.
Preconditions	The pet must be awake, and have an energy level low enough to require rest, and the game must not be in a paused or inactive state
Trigger	The player selects the ‘sleep’ option for the pet
Scenario	1. The player makes the pet sleep. 2. The pet cannot interact during sleep.
Alternatives	Early wake-up, auto-wake
Exceptions	Pet refuses to sleep, interrupted sleep
Priority	Medium

Table 2.4.17 Use Case 12: Pet Evolution

Name	Pet Evolution
Primary Actor	System
Secondary Actors	Player
Goal in Context	Care determines the pet's growth.
Preconditions	The pet needs to have been made and kept up in the game. There must have been some time spent in-game. The interaction, feeding, and care history of the pet must be tracked by the system.
Trigger	The pet has reached particular care milestones or a predetermined time has elapsed.
Scenario	1. Over time, the pet ages. 2. It might look different.
Alternatives	Multiple evolution paths delayed evolution
Exceptions	Pet does not evolve, Interrupted evolution
Priority	Medium

Table 2.4.18 Use Case 13: Save Game Progress

Name	Save Game Progress
Primary Actor	Player
Secondary Actors	Game engine, file system
Goal in Context	To continue the game later, the player saves their current state.
Preconditions	The game has to be active
Trigger	From the menu, the player chooses ‘Save’
Scenario	1. The game menu is displayed to the player 2. ‘Save Game’ is chosen by the player 3. The player’s progress and the current board condition are recorded by the game engine 4. The game state is written by the system to a database or file 5. The successful saving appears to be indicated by a confirmation message
Alternatives	Before the save is confirmed, the player cancels
Exceptions	Problems with the file system or not enough storage
Priority	Medium

Table 2.4.19 Use Case 14: Load Saved Game

Name	Load saved game
Primary Actor	Player
Secondary Actors	Game Engine, File System
Goal in Context	A previously saved game session is resumed by the player.
Preconditions	There must be a saved game file.
Trigger	The ‘Load Game’ option is chosen by the player.
Scenario	1. The game menu is displayed to the player. 2. ‘Load Game’ is chosen by the player. 3. A list of stored games is displayed by the system. 4. A saved game is chosen by the player. 5. Player progress and the board state are restored by the game engine. 6. The last stored state is used when the game resumes.
Alternatives	Corrupt save file or file system error
Exceptions	An error notice shows if there is no saved game
Priority	Medium

Table 2.4.20 Use Case 15: Display Game Rules

Name	Display Game Rules
Primary Actor	Player
Secondary Actors	Game system
Goal in Context	The player sees the instructions and game rules.
Preconditions	The player is in either the main menu or the pause menu while the game is underway.
Trigger	‘Game Rules’ is the menu item that the player chooses.
Scenario	1. The menu is displayed to the player. 2. "Game Rules" is chosen by the player. 3. The game instructions are shown by the system.
Alternatives	The player may choose to view a particular section of the game rules rather than the entire document if there are distinct sections. Instead of reading every regulation, the player may choose to browse through or look for particular ones.
Exceptions	The player is taken back to the menu and an error message is shown if there is a system error or a missing game rules file.
Priority	Low

Table 2.4.21 Use Case 16: Check Win Condition

Name	Check Win Condition
Primary Actor	Game Engine
Secondary Actors	Player, AI Opponent
Goal in Context	A player's victory is decided by the game itself.
Preconditions	Just now, an AI or player made a move.
Trigger	After each move, the system looks for win conditions.
Scenario	1. The move is finished. 2. The board state is assessed by the game engine. 3. The game announces the winner if a winning condition is satisfied. 4. The game goes on if no win condition is satisfied.
Alternatives	A draw is announced if there is no winner and the board is full
Exceptions	None
Priority	High

Table 2.4.22 Use Case 17: End Game and Show Results

Name	End Game and Show Results
Primary Actor	Game Engine
Secondary Actors	Player, AI Opponent
Goal in Context	Show the game's final results and give the player the option to quit or resume.
Preconditions	Either a player has won or there has been a draw.
Trigger	The game recognizes a draw or a win condition.
Scenario	1. A win or tie is detected by the game. 2. The results screen is shown by the system. 3. The player has the option to quit or restart the game.
Alternatives	None
Exceptions	System crash or unexpected error
Priority	High

Table 2.4.23 Use Case 18: Change Game Settings

Name	Change Game Settings
Primary Actor	Player
Secondary Actors	Game Engine
Goal in Context	The player can alter gameplay parameters like sound preferences, board size, and difficulty.
Preconditions	The player is in the settings menu while the game is going.
Trigger	From the menu, the player chooses the "Settings" option.
Scenario	1. The settings menu is displayed by the player. 2. The player can change several parameters, including board size, sound, and difficulty. 3. The modifications are saved or applied by the player. 4. The updated settings become active.
Alternatives	Invalid inputs or corrupted settings files
Exceptions	The player can restore settings to their original state
Priority	Medium

Table 2.4.24 Use Case 19: View Player Statistics

Name	View Player Statistics
Primary Actor	Player
Secondary Actors	Game engine
Goal in Context	The player sees their previous results, including their average game length and win/loss ratio.
Preconditions	The player has at least one game under their belt.
Trigger	The ‘Statistics’ option is chosen by the player from the menu.
Scenario	1. To access the statistics screen, the player navigates. 2. Relevant player statistics are retrieved by the system and displayed. 3. If necessary, the player can reset or filter statistics.
Alternatives	The player has the option to reset statistics.
Exceptions	Inaccurately recorded statistics or corrupted data
Priority	Medium

Table 2.4.25 Use Case 20: Debug Mode

Name	Debug Mode
Primary Actor	Developer
Secondary Actors	Game Engine
Goal in Context	Debugging tools for performance monitoring, motion validation, and AI behaviour testing can be enabled by developers.
Preconditions	Developer settings must have debug mode activated.
Trigger	Debugging mode is activated by the developer.
Scenario	1. Debugging options are accessed by the developer. 2. The debug features (such as move tracking and AI decision logs) are activated. 3. The game records essential debugging data. 4. The data is used by developers for testing and debugging.
Alternatives	The developer has the option to turn off debug mode.
Exceptions	If debug logs are overloaded, performance may be slowed.
Priority	High (for developers), Low (for players)

Table 2.4.26 Use Case 21: Undo Last Move

Name	Undo Last Move
Primary Actor	Player
Secondary Actors	Game Engine
Goal in Context	The player has the option to reverse their previous action and try a new tactic.
Preconditions	Undo functionality must be enabled and the game must be running.
Trigger	The ‘Undo’ option is chosen by the player.
Scenario	1. A move is made by the player. 2. The player chooses to reverse the action. 3. The prior board state is restored in the system. 4. A new move is made by the player.
Alternatives	Undo may not be possible in multiplayer mode.
Exceptions	The player may only have a certain number of updos.
Priority	Medium

Table 2.4.27 Use Case 22: Display Error Messages

Name	Display Error Messages
Primary Actor	System
Secondary Actors	Game system
Goal in Context	The system alerts the user of issues
Preconditions	While the game is underway, the player must be in either the main menu or the pause menu. There must be a "Game Rules" option in the menu.
Trigger	The "Game Rules" option is chosen by the player from the menu.
Scenario	1. The player inputs invalid data 2. The system displays an error message
Alternatives	The player may choose to view a particular section of the game rules rather than the entire document if there are distinct sections. Instead of reading every regulation, the player may choose to browse through or look for particular ones.
Exceptions	The player is taken back to the menu and an error message is shown if there is a system error or a missing game rules file.
Priority	High

➔ Activity Diagrams

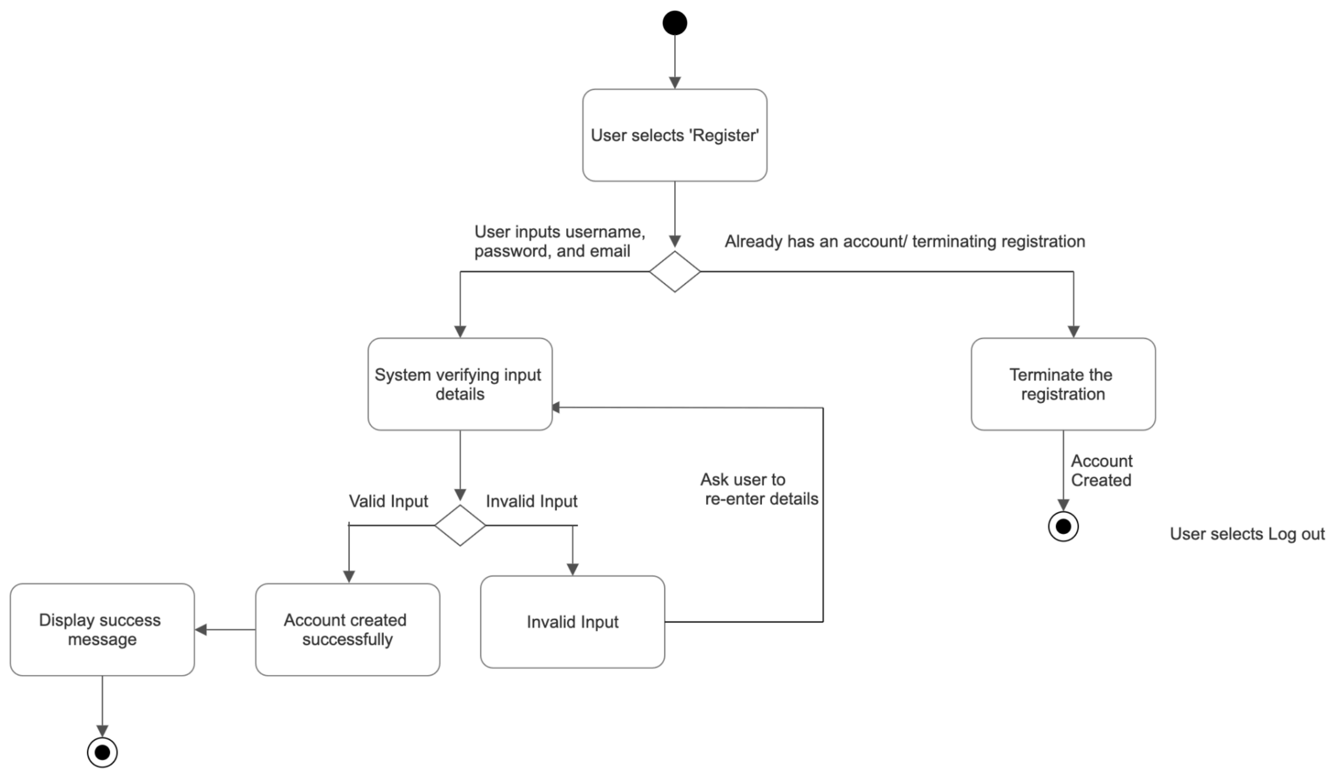

User selects 'Register'
User inputs username, password, and email
Already has an account/ terminating registration
System verifying input details
Terminate the registration
Account Created
User selects Log out
Ask user to re-enter details
Valid Input
Invalid Input
Display success message
Account created successfully
Invalid Input

Figure 2.4.1 Activity Diagram 1: User Register

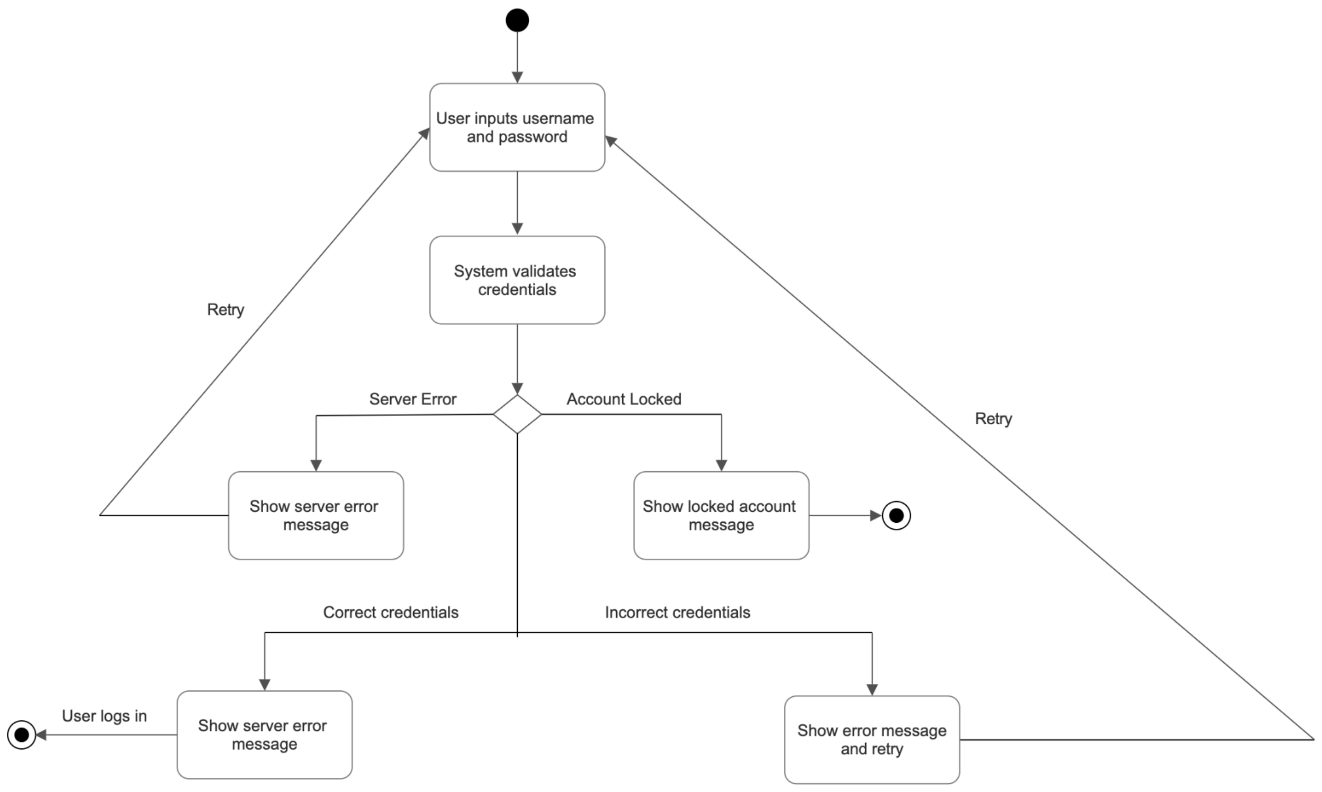

User inputs username and password
System validates credentials
Retry
Server Error
Account Locked
Retry
Show server error message
Show locked account message
Correct credentials
Incorrect credentials
User logs in
Show server error message
Show error message and retry

Figure 2.4.2 Activity Diagram 2: User Login

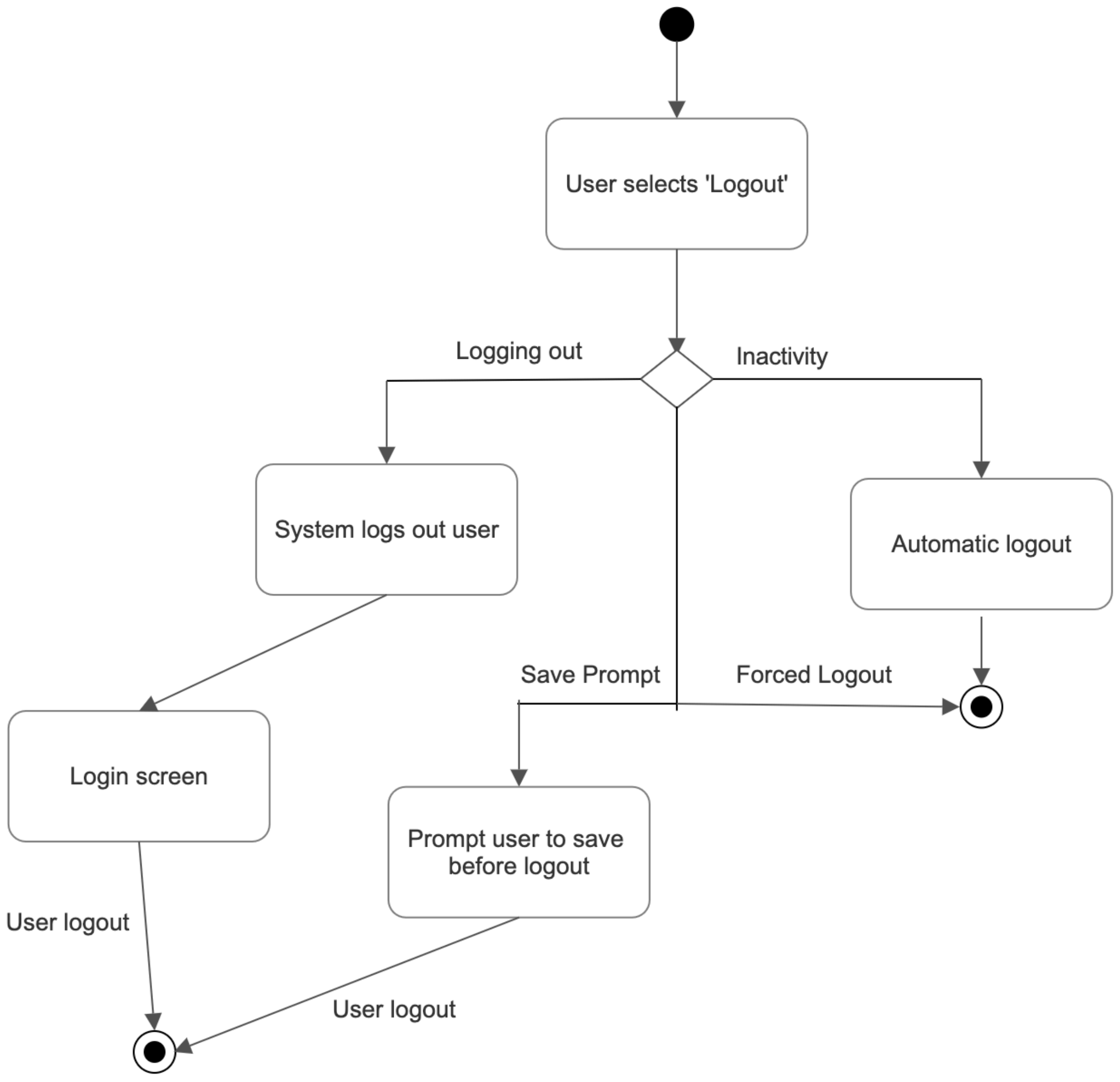


Figure 2.4.3 Activity Diagram 3: User Logout

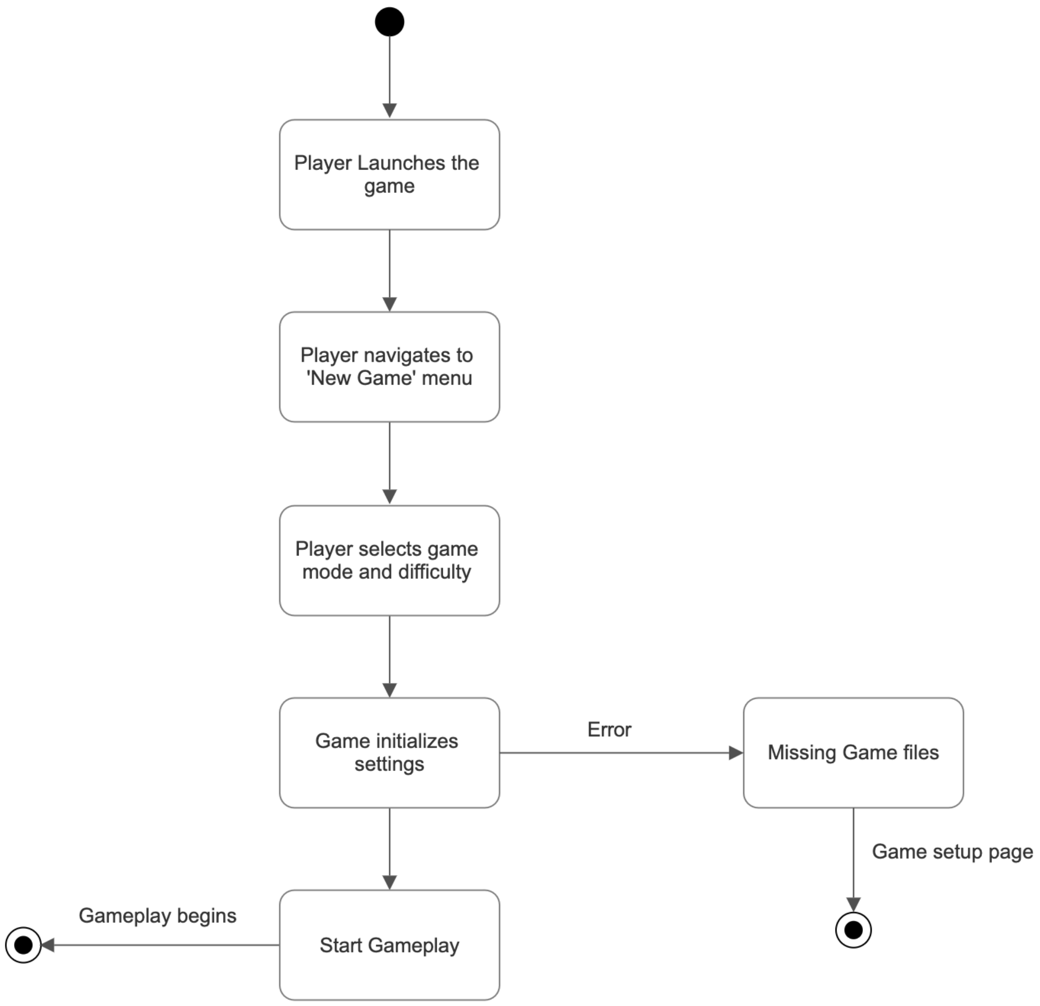

Player Launches the game
Player navigates to 'New Game' menu
Player selects game mode and difficulty
Game initializes settings
Error
Missing Game files
Game setup page
Start Gameplay
Gameplay begins

Figure 2.4.4 Activity Diagram 4: Starting a New Game

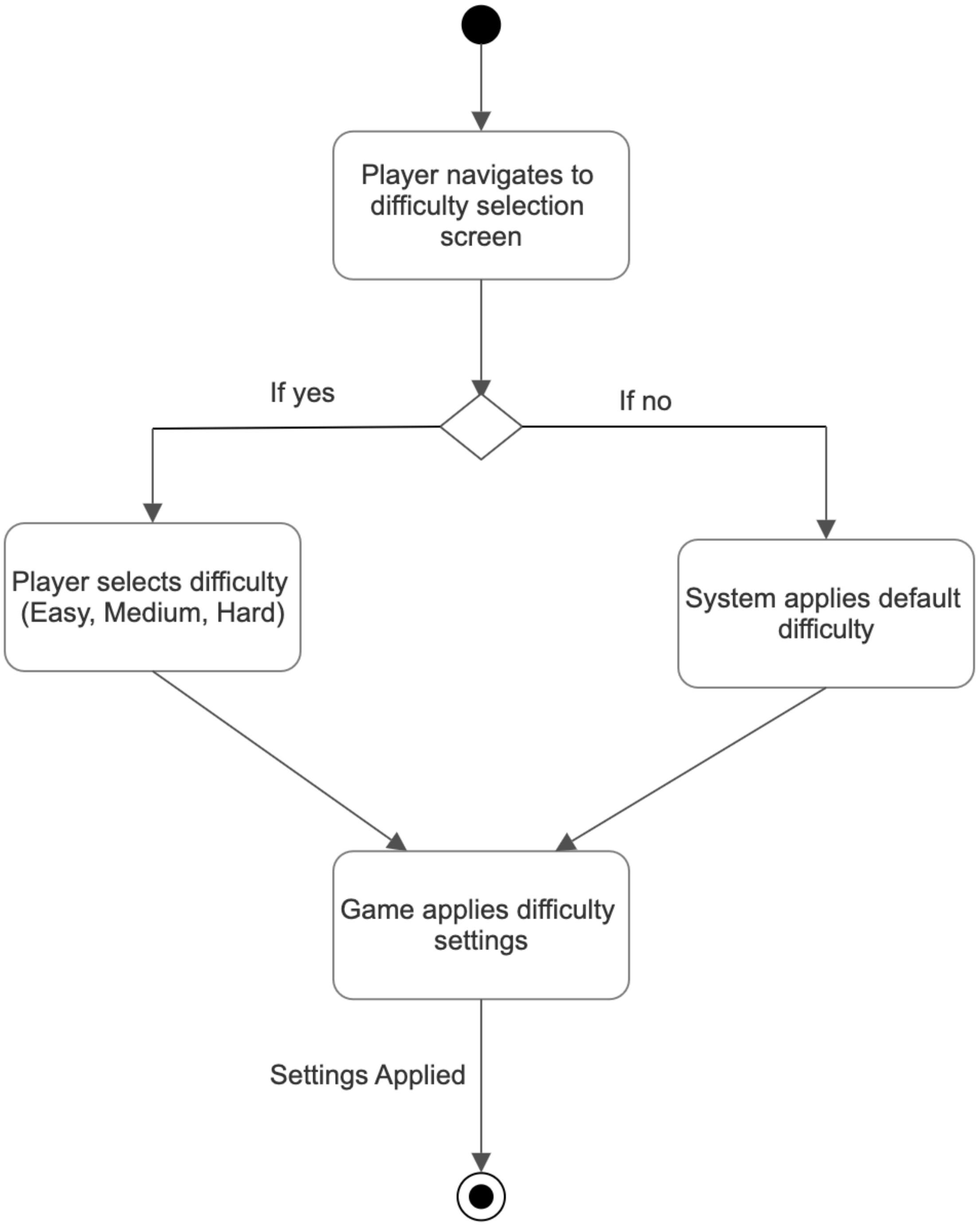

Player navigates to difficulty selection screen
If yes
If no
Player selects difficulty (Easy, Medium, Hard)
System applies default difficulty
Game applies difficulty settings
Settings Applied

Figure 2.4.5 Activity Diagram 5: Choosing Difficulty Level

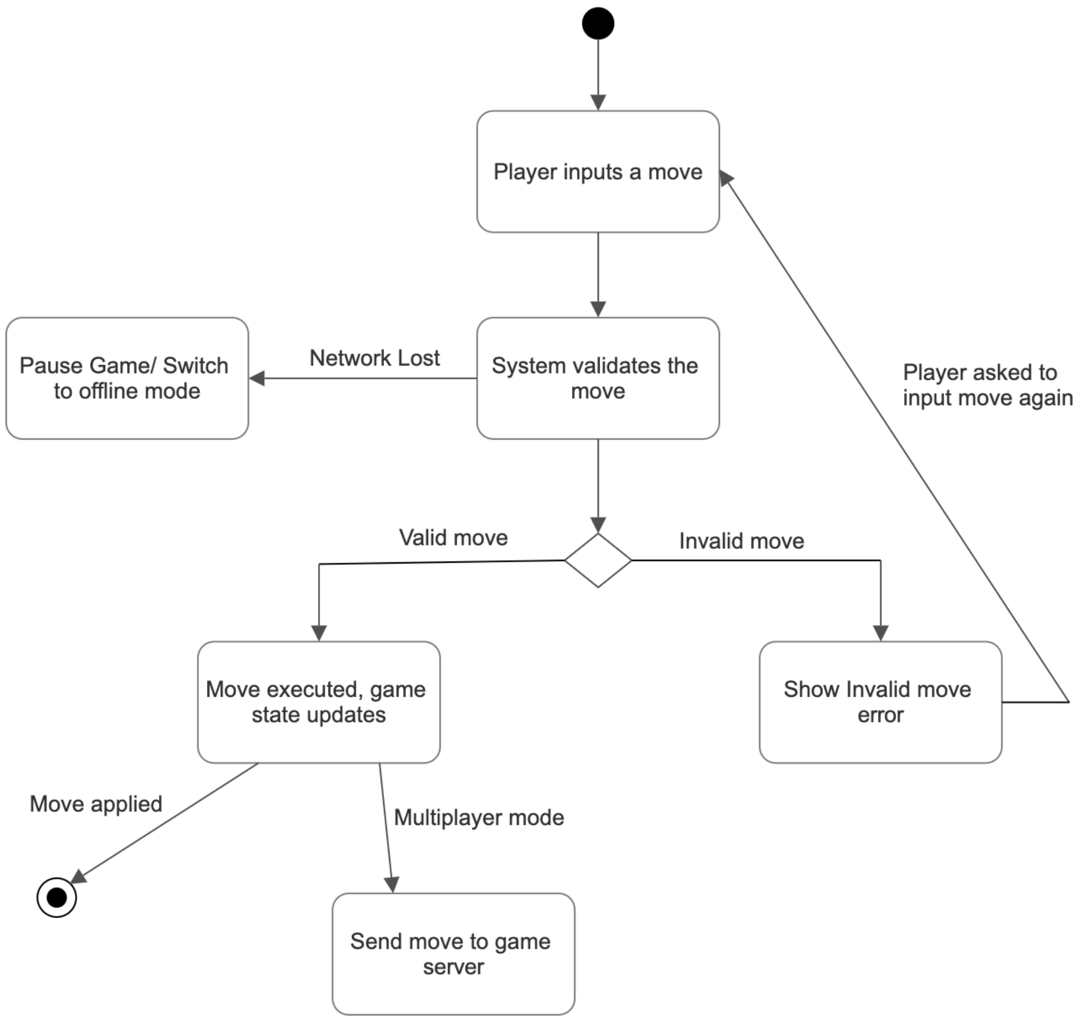
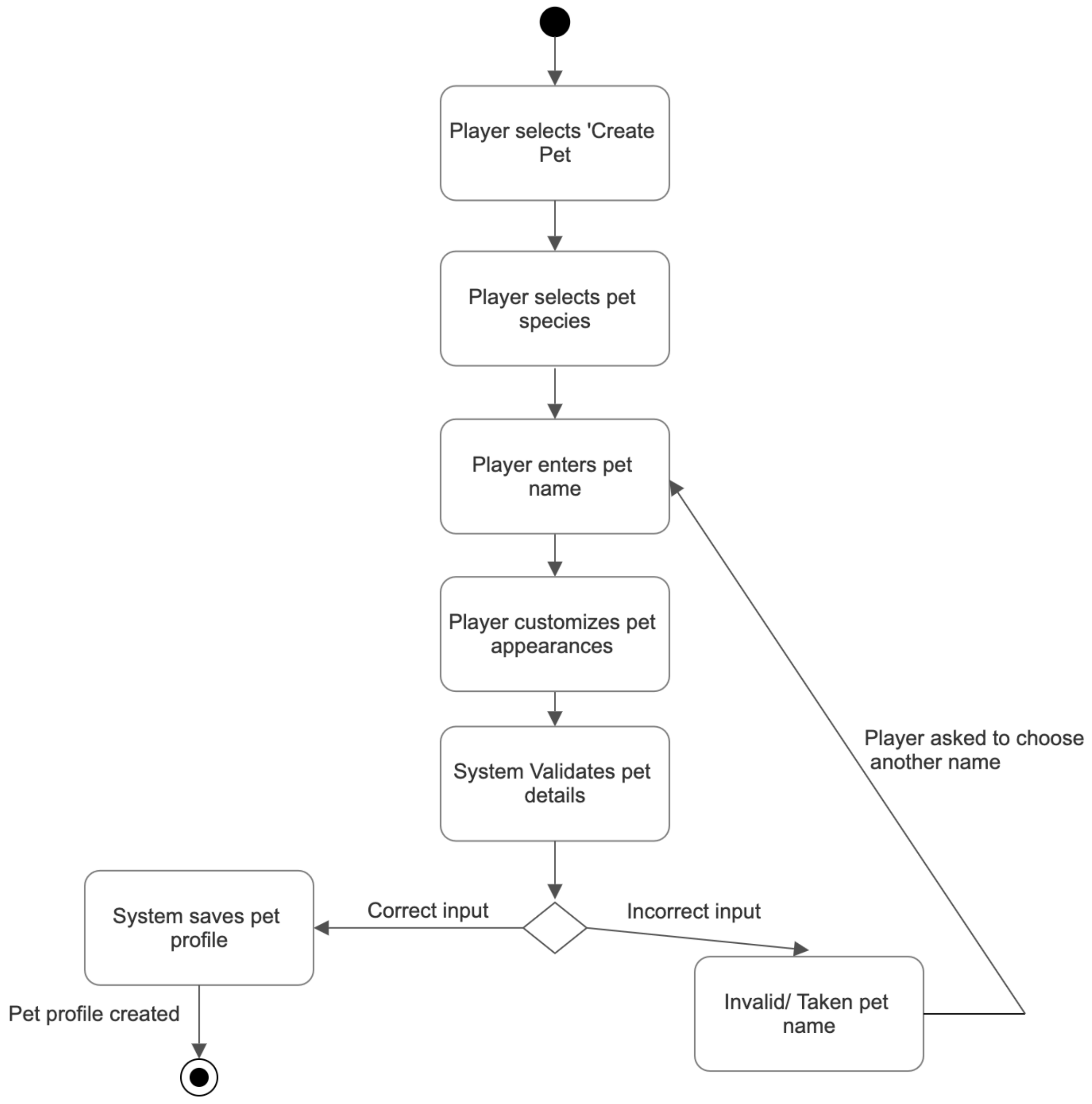


Figure 2.4.6 Activity Diagram 6: Player Making Move



Player selects 'Create Pet
Player selects pet species
Player enters pet name
Player customizes pet appearances
System Validates pet details
Correct input
Incorrect input
System saves pet profile
Invalid/ Taken pet name
Player asked to choose another name
Pet profile created

Figure 2.4.7 Activity Diagram 7: Creating Virtual Pet

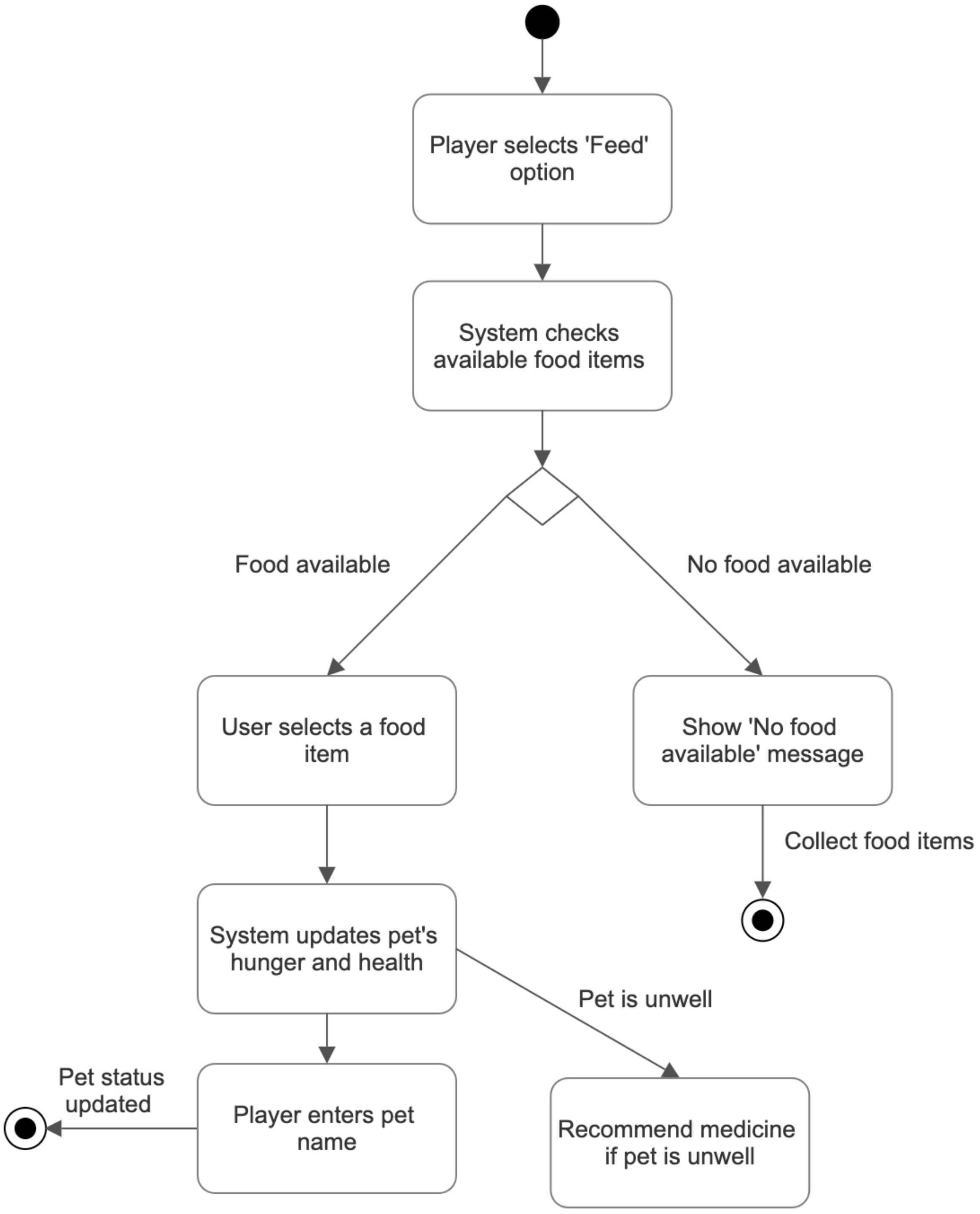

Player selects 'Feed' option
System checks available food items
Food available
No food available
User selects a food item
Show 'No food available' message
Collect food items
System updates pet's hunger and health
Pet is unwell
Pet status updated
Player enters pet name
Recommend medicine if pet is unwell

Figure 2.4.8 Activity Diagram 8: Feed the Pet

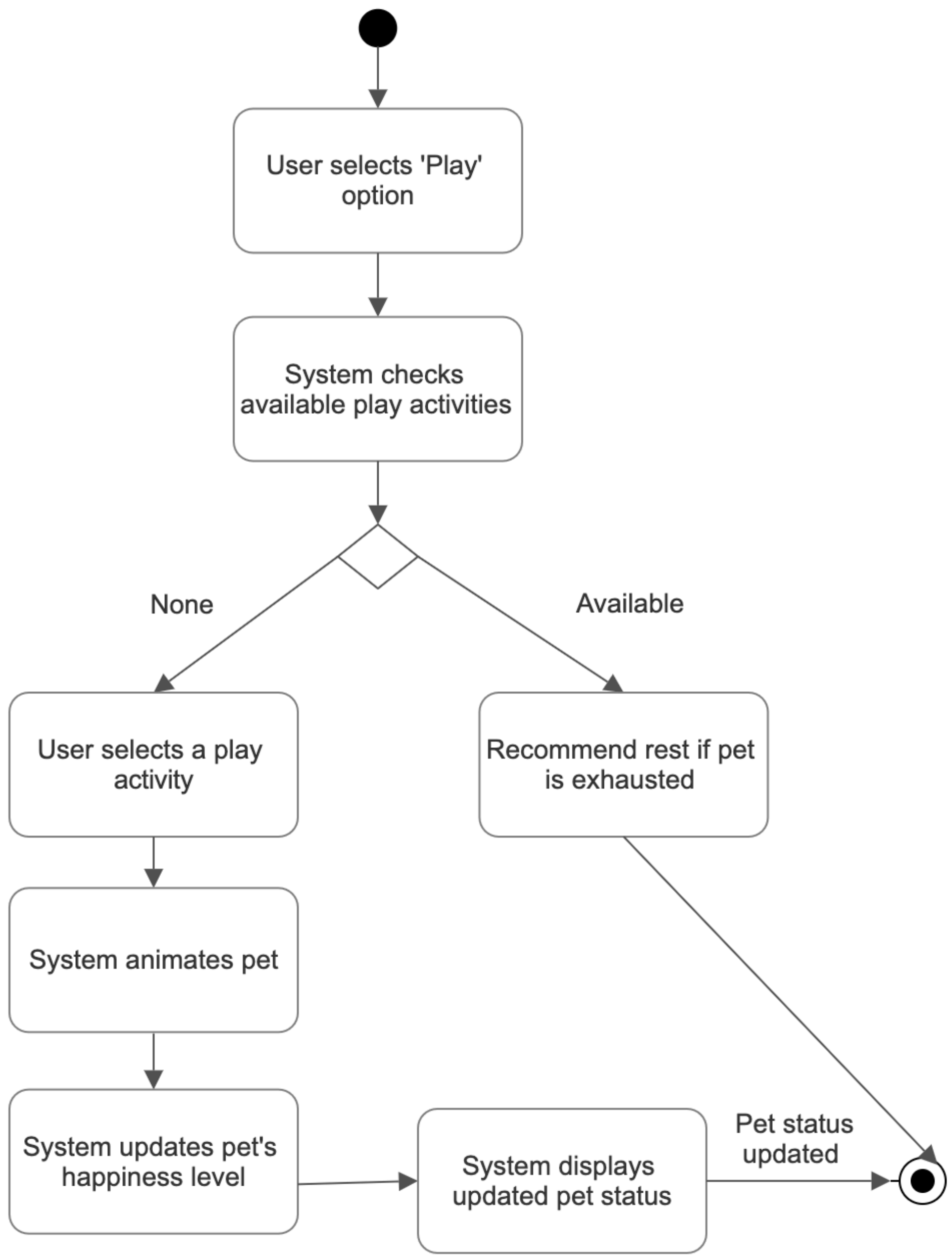

User selects 'Play' option
System checks available play activities
None
Available
User selects a play activity
Recommend rest if pet is exhausted
System animates pet
System updates pet's happiness level
System displays updated pet status
Pet status updated

Figure 2.4.9 Activity Diagram 9: Play with the Pet

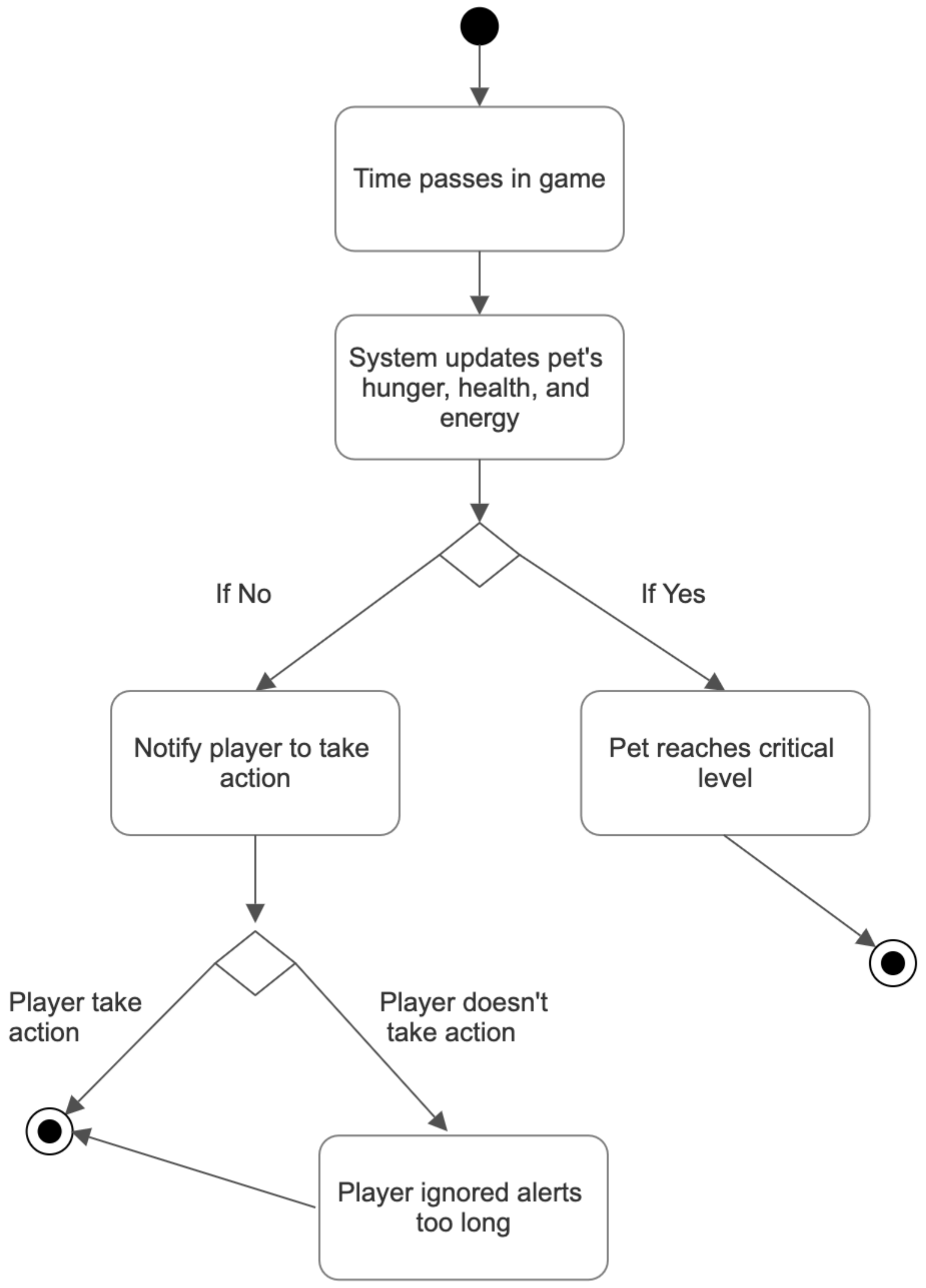

Time passes in game
System updates pet's hunger, health, and energy
If No
If Yes
Notify player to take action
Pet reaches critical level
Player take action
Player doesn't take action
Player ignored alerts too long

Figure 2.4.10 Activity Diagram 10: Pet Needs

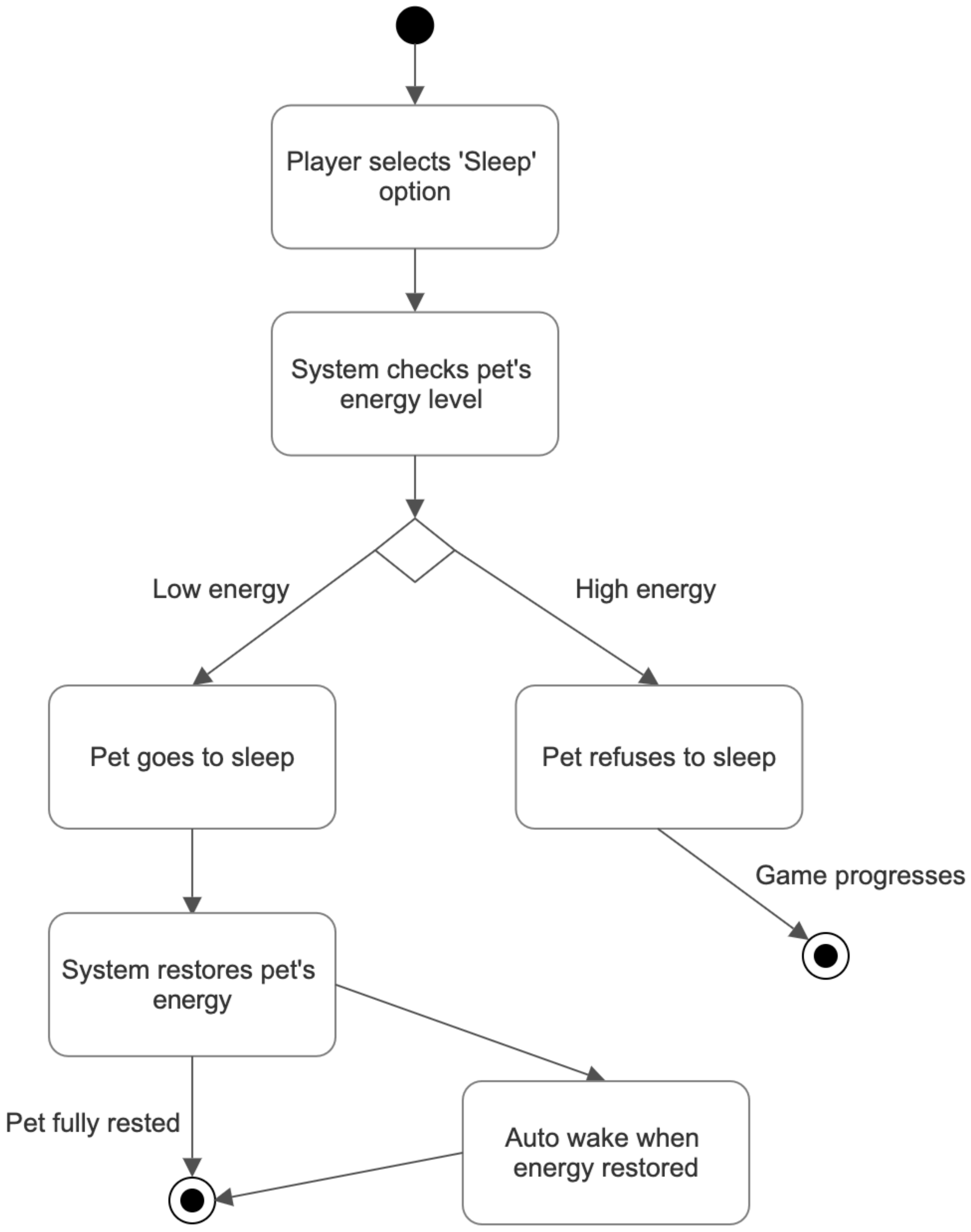

Player selects 'Sleep' option
System checks pet's energy level
Low energy
High energy
Pet goes to sleep
Pet refuses to sleep
Game progresses
System restores pet's energy
Pet fully rested
Auto wake when energy restored

Figure 2.4.11 Activity Diagram 11: Pet Sleeping

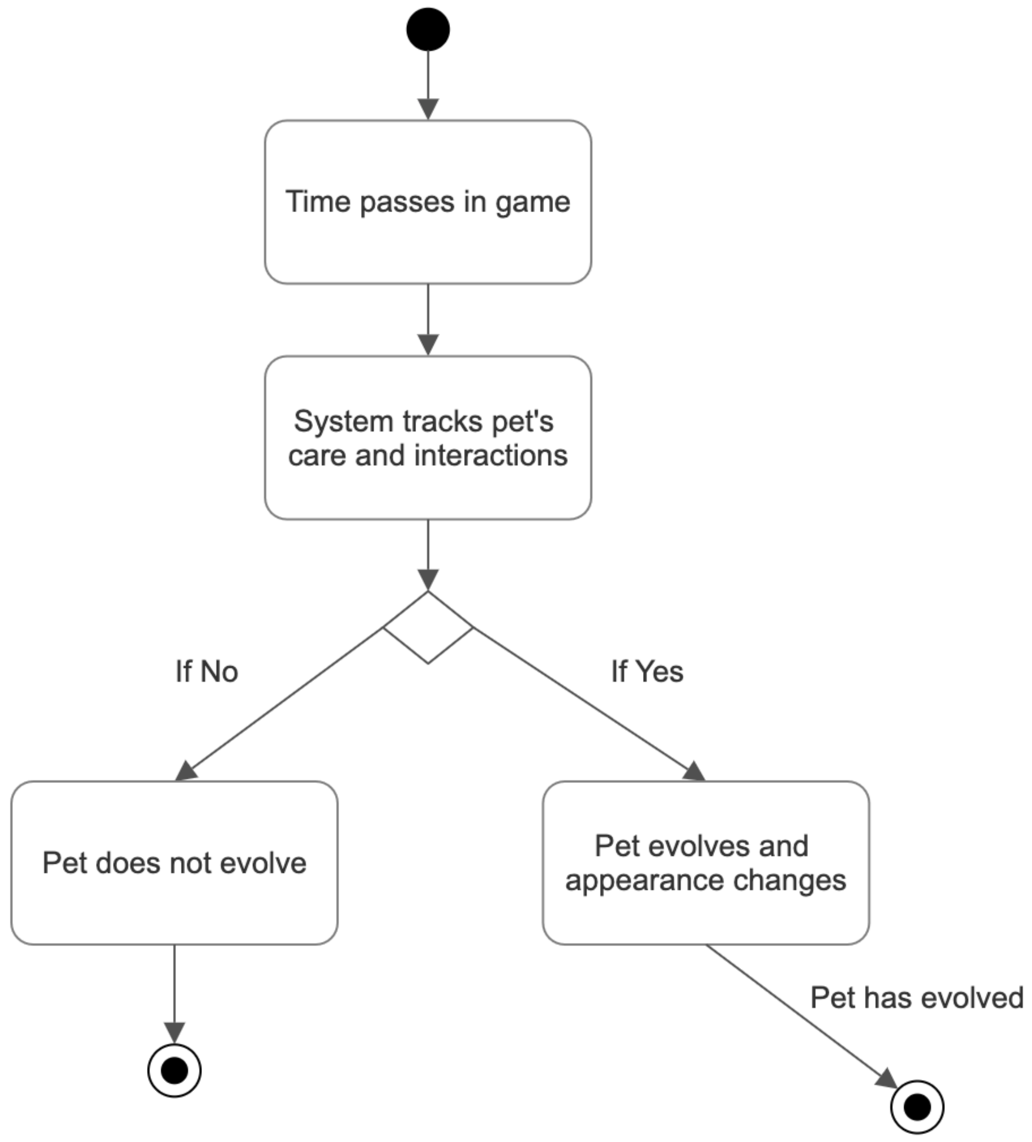

Time passes in game
System tracks pet's care and interactions
If No
If Yes
Pet does not evolve
Pet evolves and appearance changes
Pet has evolved

Figure 2.4.12 Activity Diagram 12: Pet Evolution

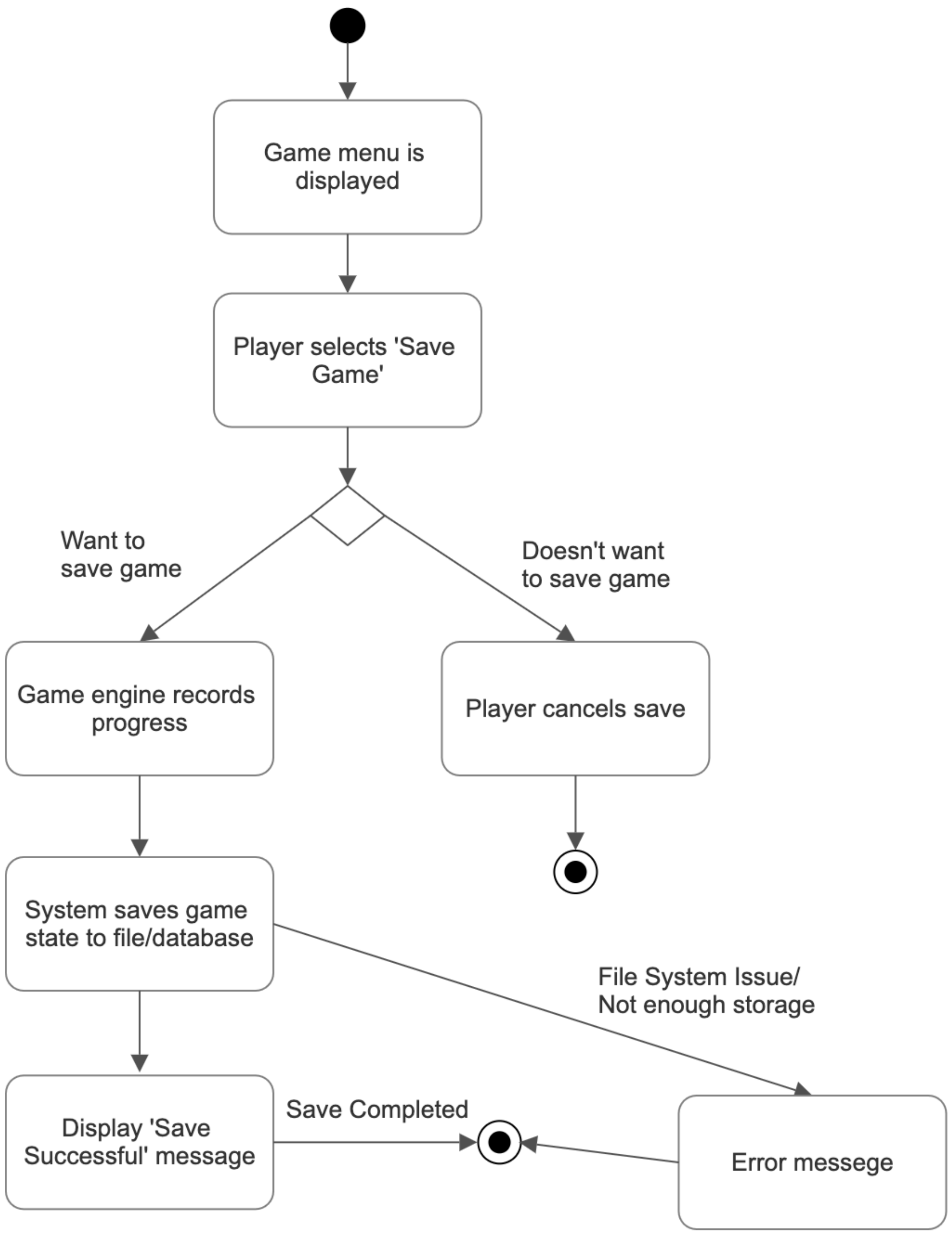
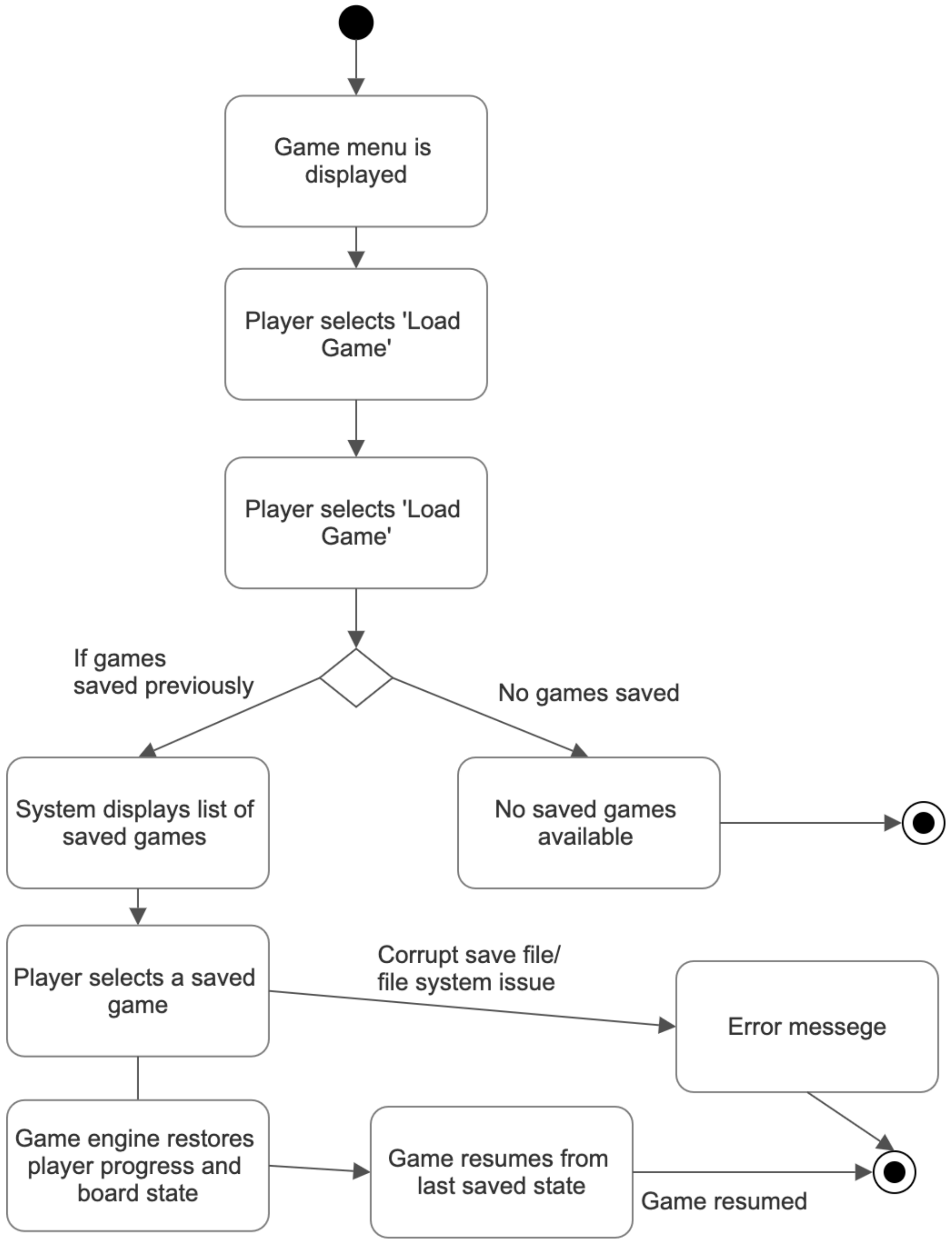

Game menu is displayed
Player selects 'Save Game'
Want to save game
Doesn't want to save game
Game engine records progress
Player cancels save
System saves game state to file/database
File System Issue/ Not enough storage
Display 'Save Successful' message
Save Completed
Error messege

Figure 2.4.13 Activity Diagram 13: Save Game Progress



Game menu is displayed
Player selects 'Load Game'
Player selects 'Load Game'
If games saved previously
No games saved
System displays list of saved games
No saved games available
Player selects a saved game
Corrupt save file/ file system issue
Error messege
Game engine restores player progress and board state
Game resumes from last saved state
Game resumed

Figure 2.4.14 Activity Diagram 14: Load Saved Game

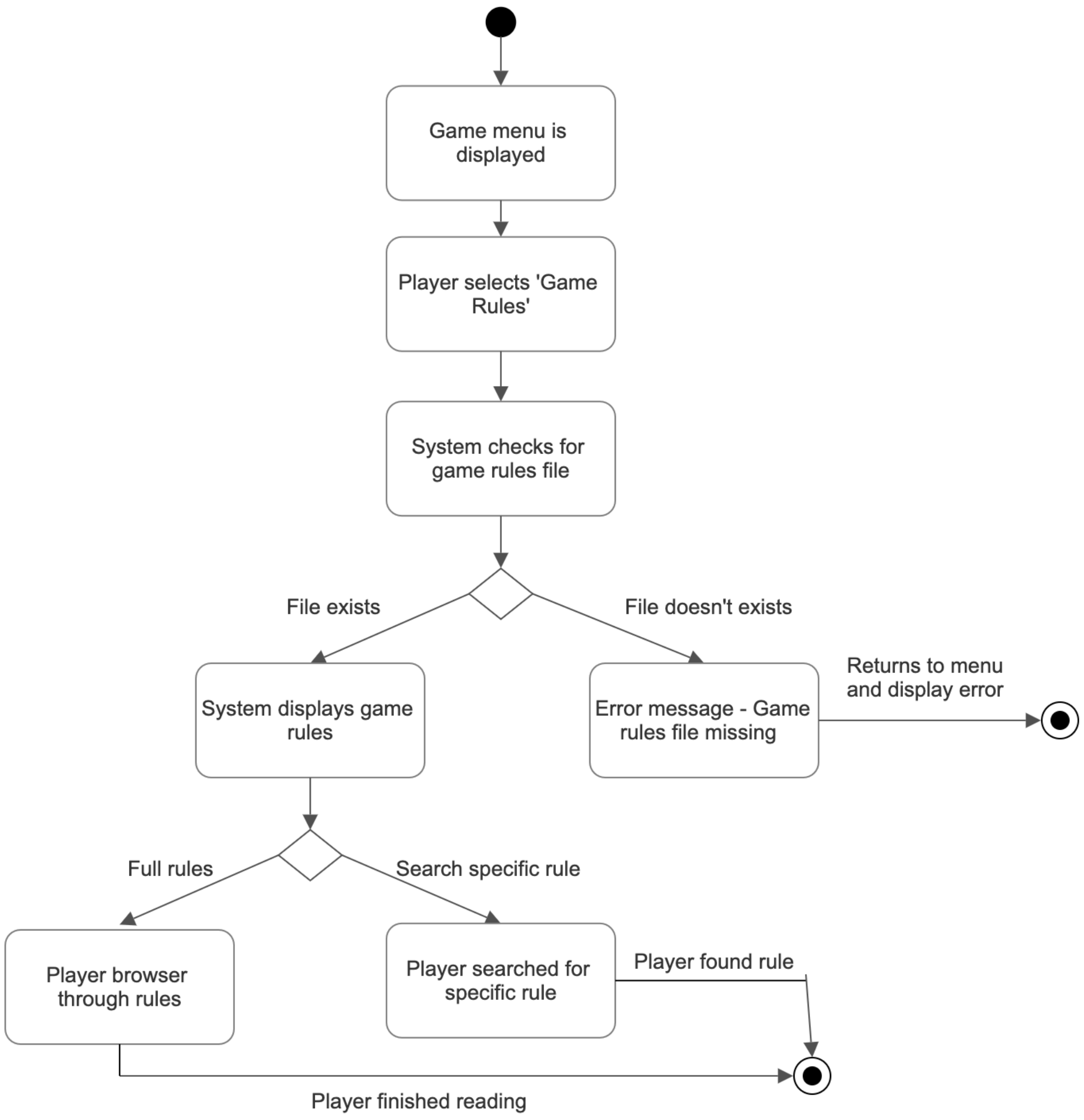

Game menu is displayed
Player selects 'Game Rules'
System checks for game rules file
File exists
File doesn't exists
System displays game rules
Error message - Game rules file missing
Returns to menu and display error
Full rules
Search specific rule
Player browser through rules
Player searched for specific rule
Player found rule
Player finished reading

Figure 2.4.15 Activity Diagram 15: Display Game Rules

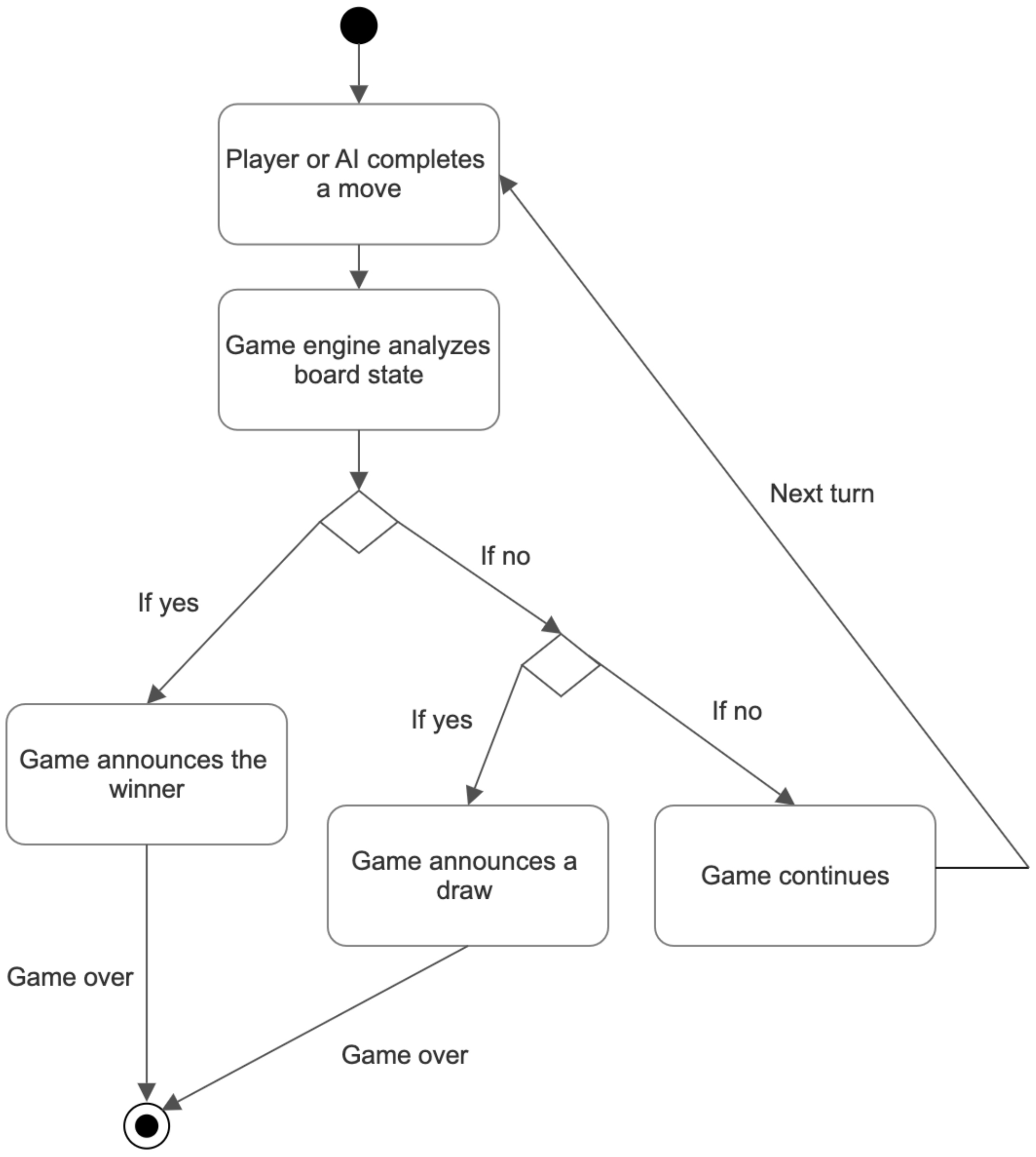

Player or AI completes a move
Game engine analyzes board state
If yes
If no
Next turn
If yes
If no
Game announces the winner
Game announces a draw
Game continues
Game over
Game over

Figure 2.4.16 Activity Diagram 16: Check Win Condition

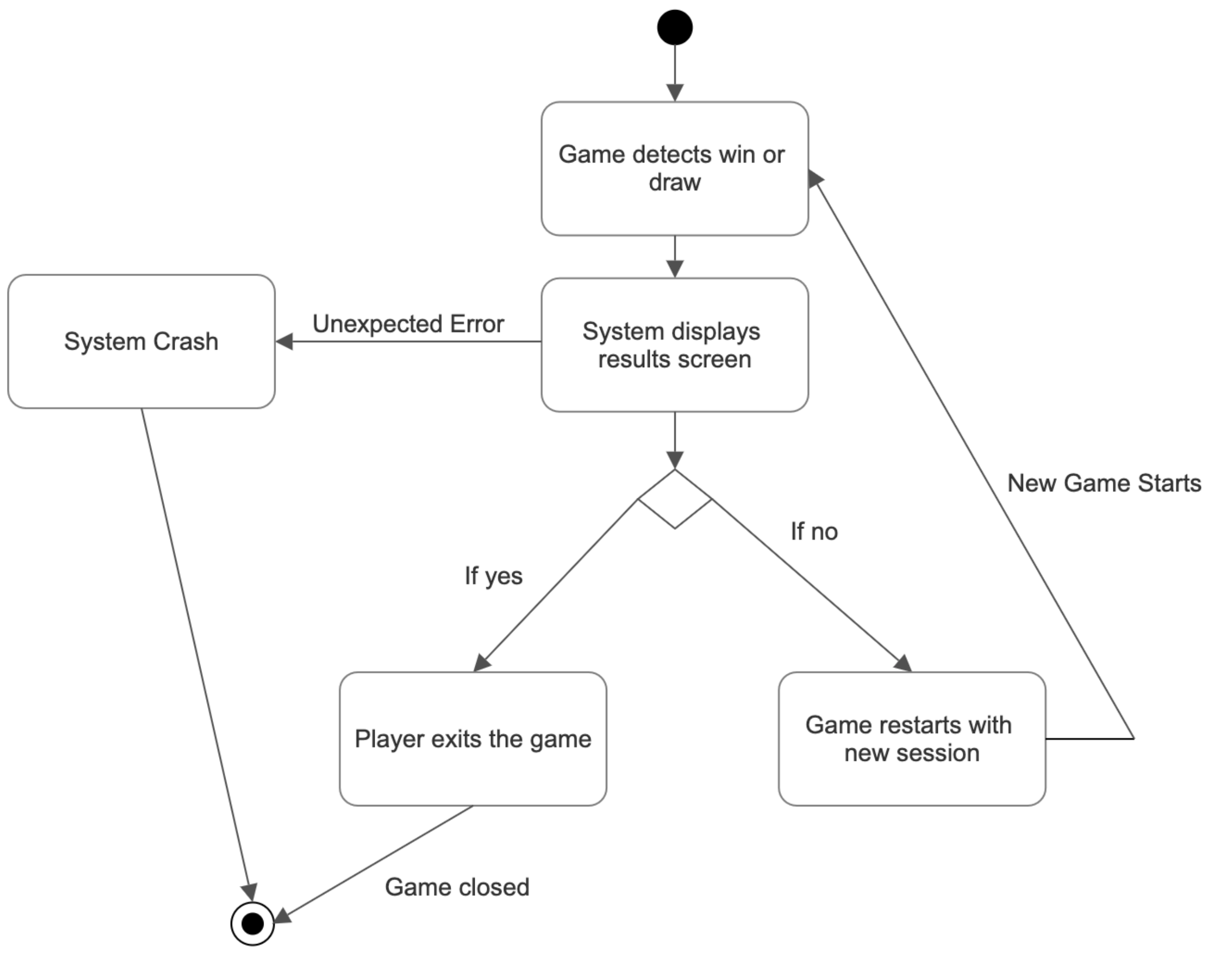


Figure 2.4.17 Activity Diagram 17: End Game and Show Results

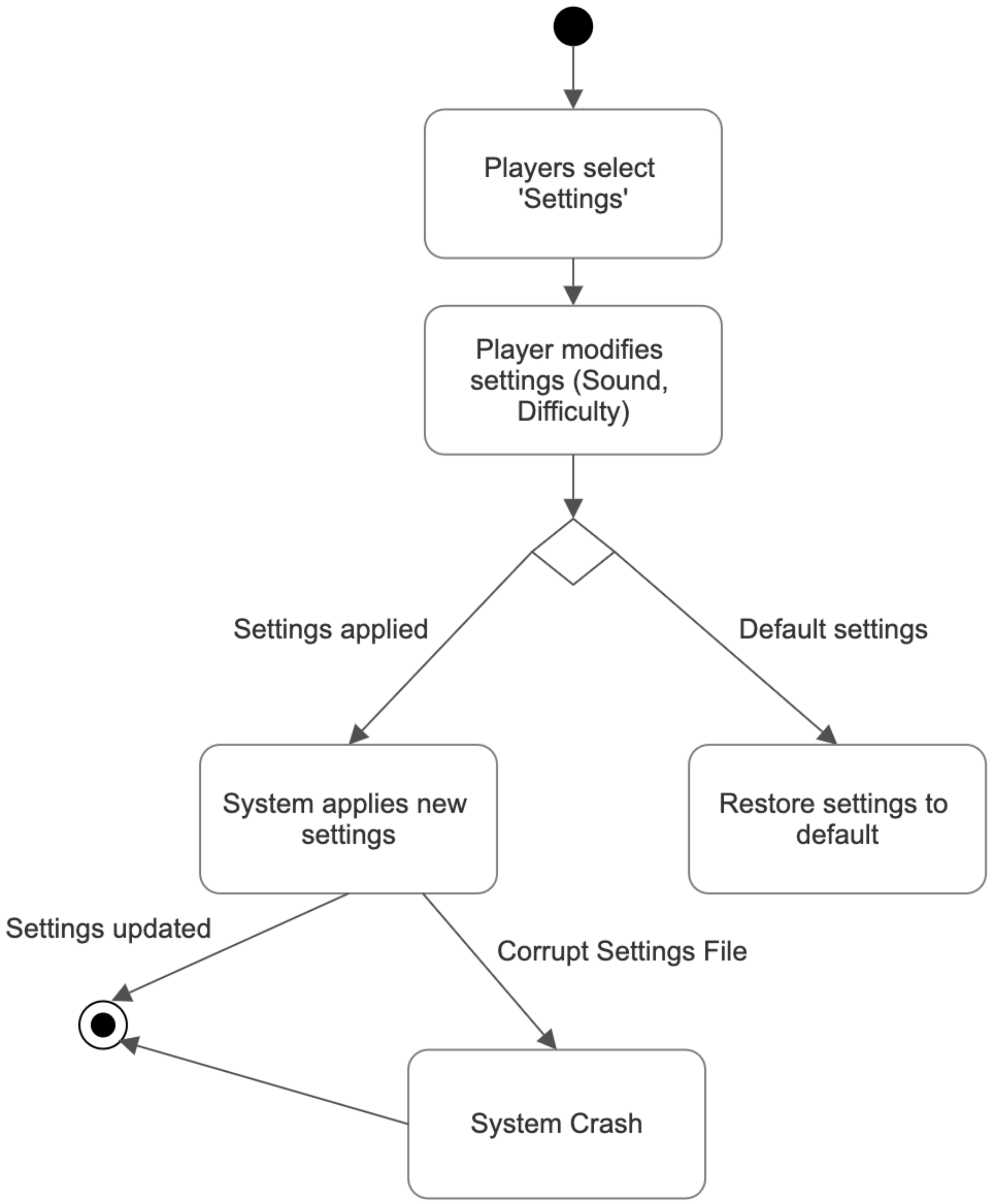

Players select 'Settings'
Player modifies settings (Sound, Difficulty)
Settings applied
Default settings
System applies new settings
Restore settings to default
Settings updated
Corrupt Settings File
System Crash

Figure 2.4.18 Activity Diagram 18: Change Game Settings

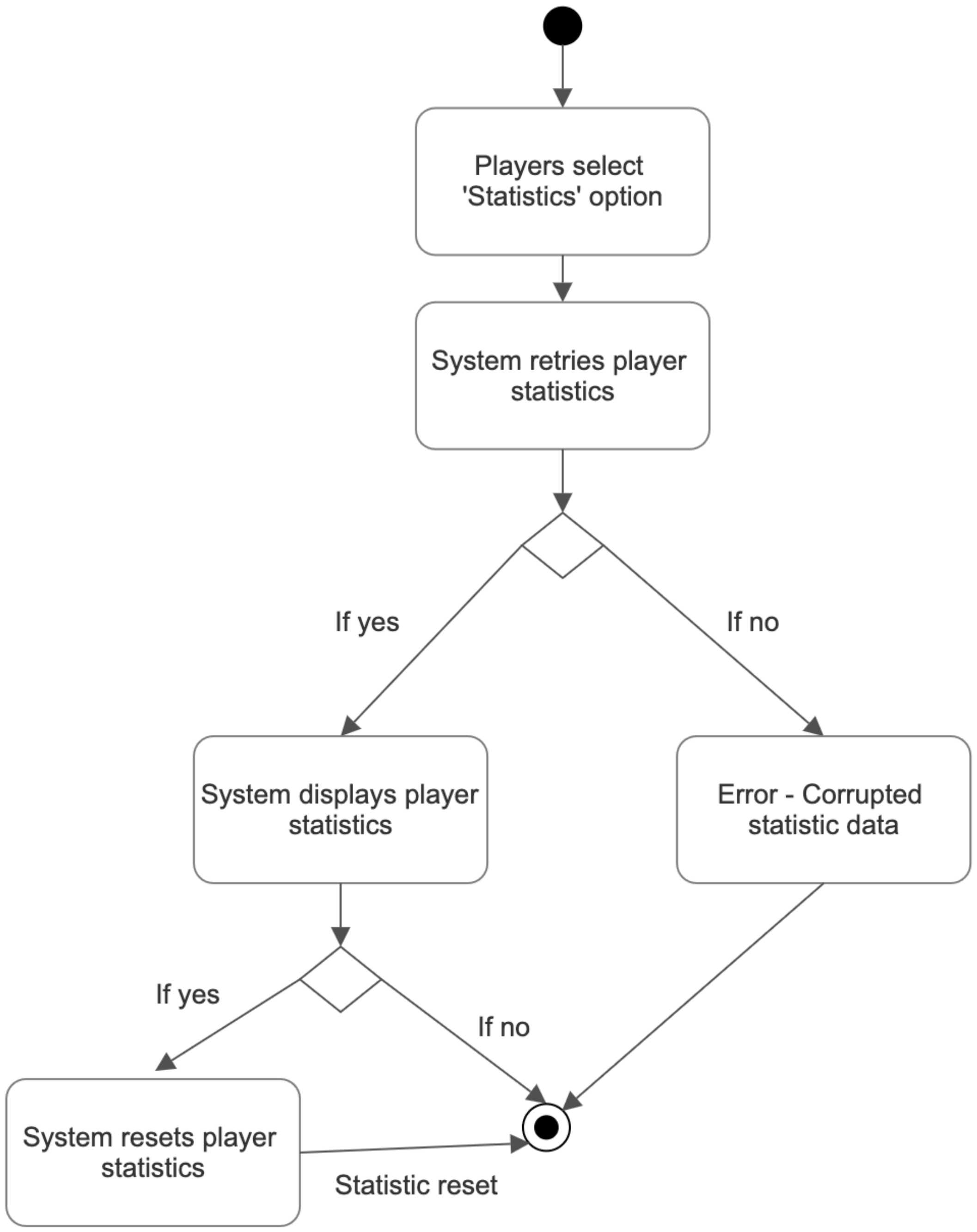

Players select 'Statistics' option
System retries player statistics
If yes
If no
System displays player statistics
Error - Corrupted statistic data
If yes
If no
System resets player statistics
Statistic reset

Figure 2.4.19 Activity Diagram 19: View Player Statistics

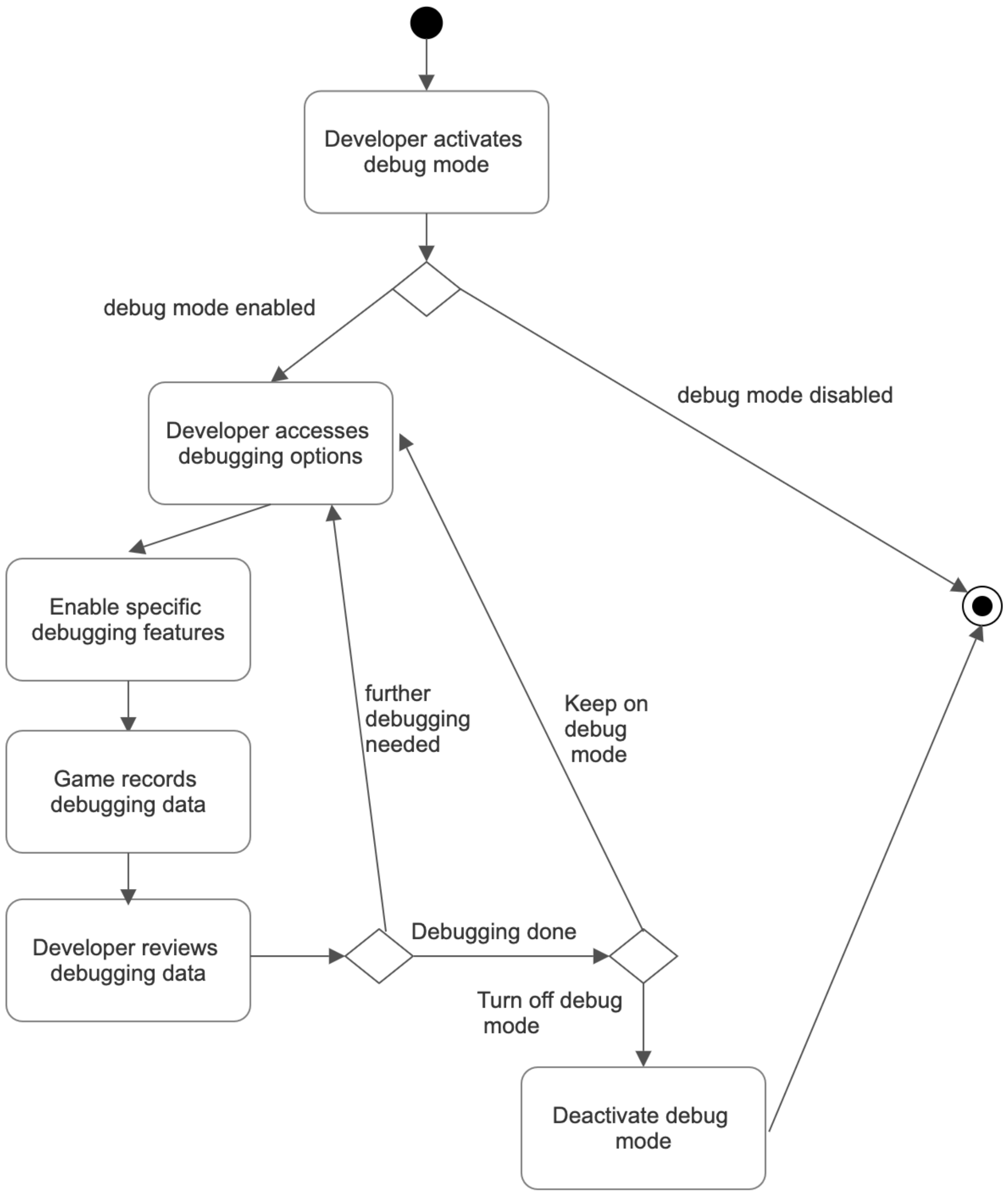

Developer activates debug mode
debug mode enabled
debug mode disabled
Developer accesses debugging options
Enable specific debugging features
Game records debugging data
Developer reviews debugging data
further debugging needed
Keep on debug mode
Debugging done
Turn off debug mode
Deactivate debug mode

Figure 2.4.20 Activity Diagram 20: Debug Mode

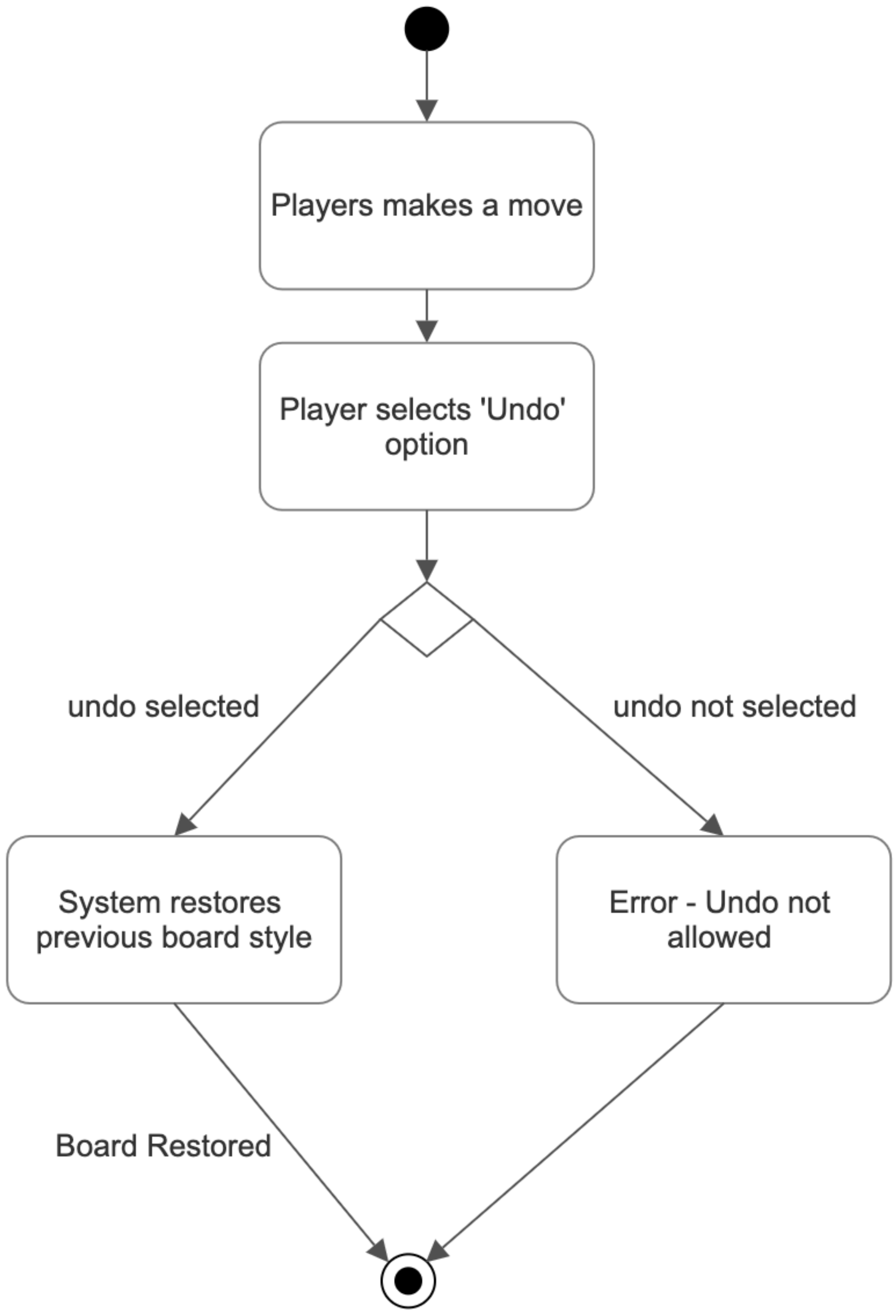


Figure 2.4.21 Activity Diagram 21: Undo Last Move

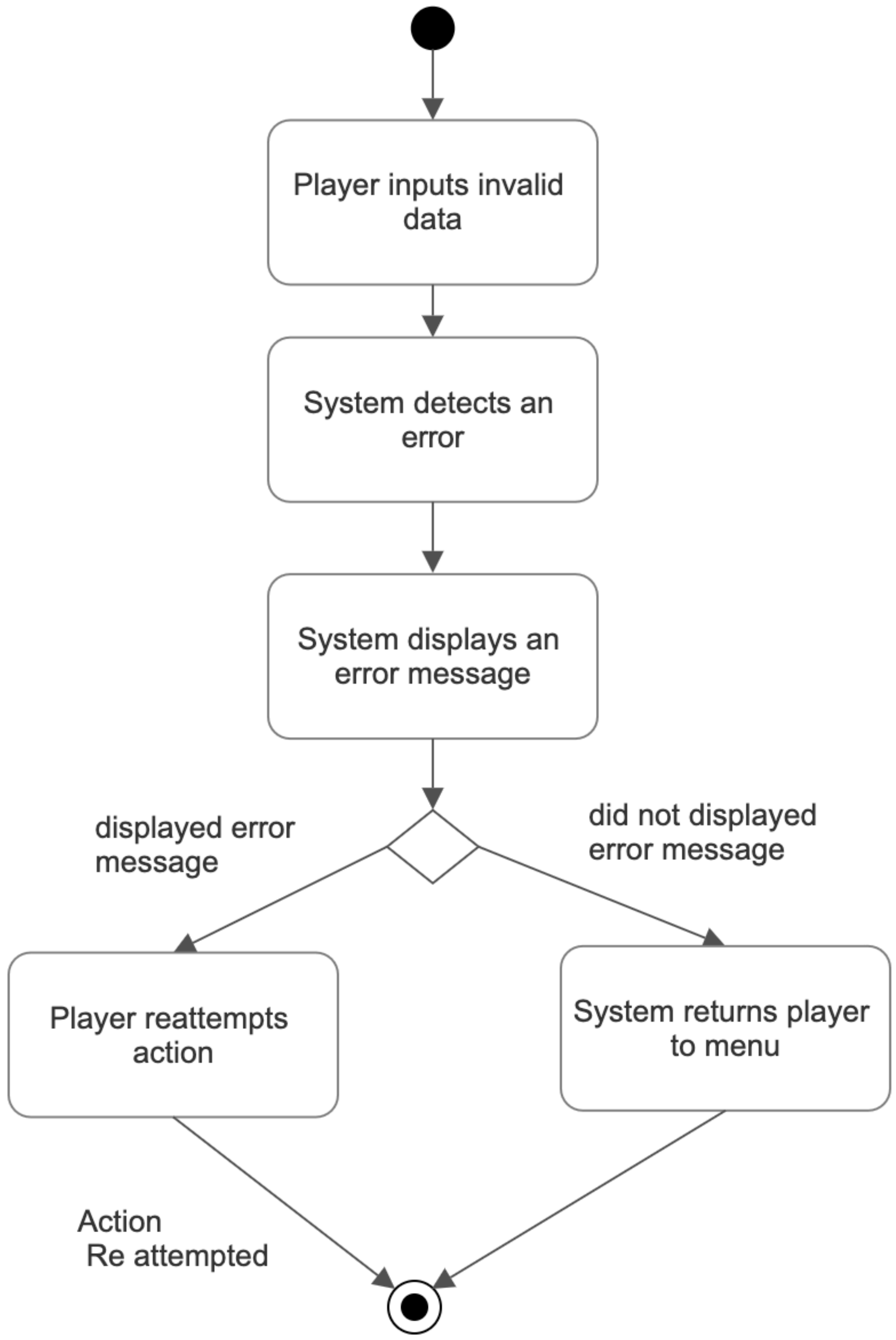
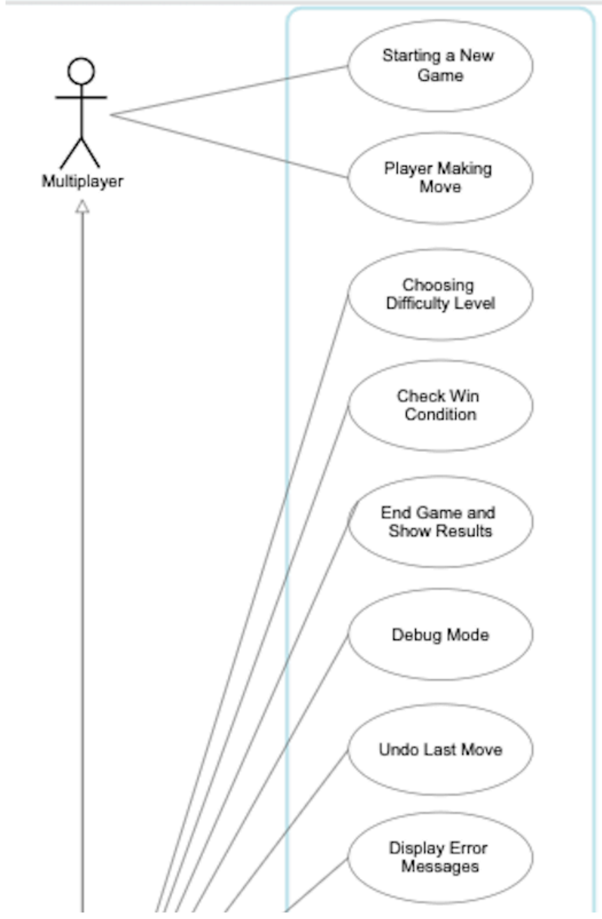

Player inputs invalid data
System detects an error
System displays an error message
displayed error message
did not displayed error message
Player reattempts action
System returns player to menu
Action Re attempted

Figure 2.4.22 Activity Diagram 22: Display Error Message

➔ Use Case Diagram



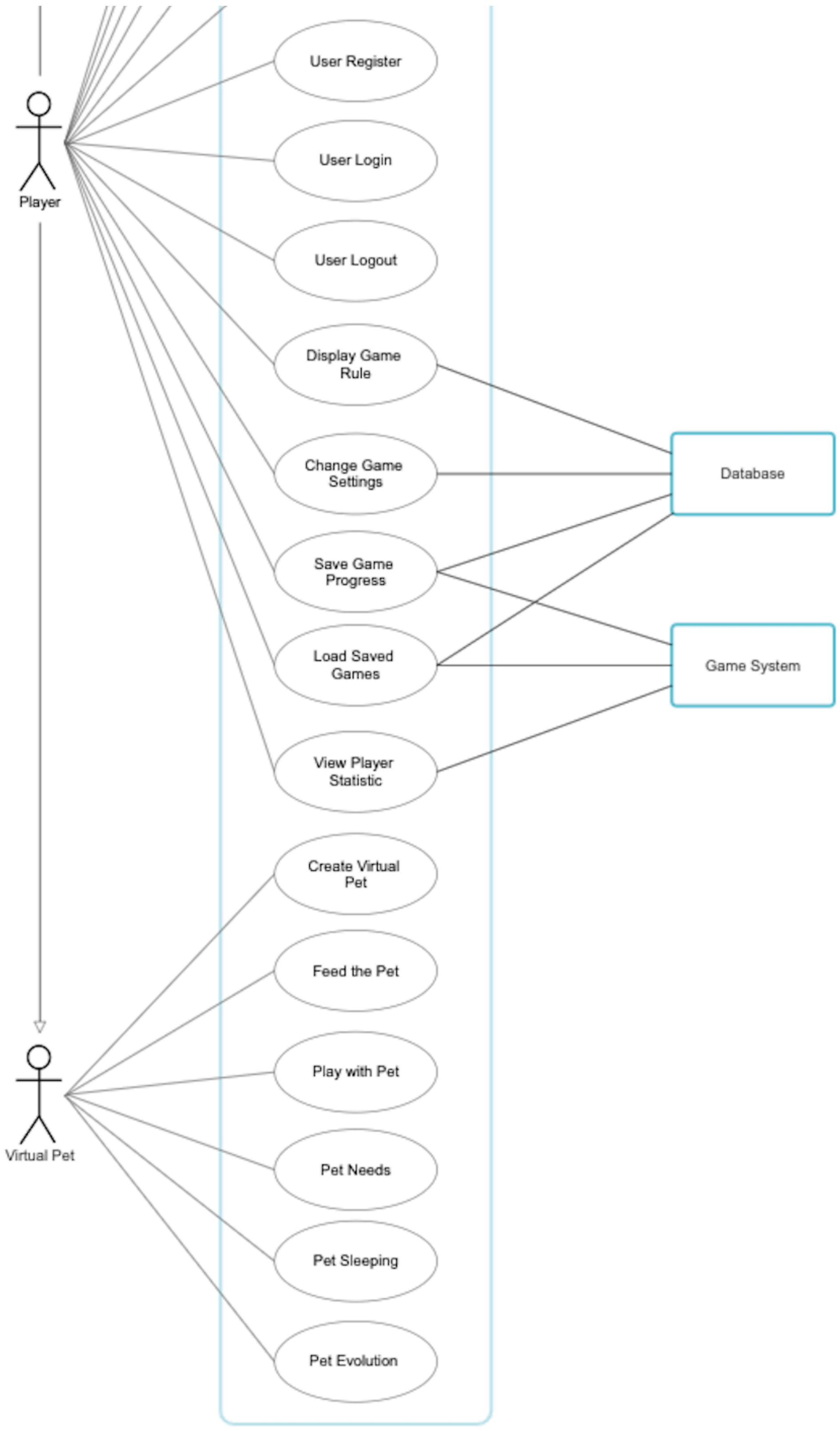


Figure 2.4.23 Use Case Diagram

2.5 Non-Functional Requirements

- **Java Version Compatibility**: The game must be developed using Java to ensure compatibility with the current software environment and optimize performance.
- **Object-Oriented Design**: The game must follow an object-oriented design approach, utilizing appropriate design patterns for scalability, maintainability, and ease of development.
- **Content Restrictions**: The game must adhere to family-friendly standards, ensuring it is suitable for players aged 5 and up. No objectionable or offensive content should be included in the game.
- **Graphical User Interface (GUI)**: The GUI should be user-friendly, visually appealing, and intuitive. It must feature clear labelling, intuitive navigation, and a consistent visual language to minimize user effort and enhance engagement. The interface should be accessible to users of all skill levels, with a low learning curve.
- **Data Storage**: The application should store all game-related data, such as pet statistics and player progress, locally on the user's device (without requiring an internet connection). Formats like JSON, XML, CSV, or TSV are going to be used for data storage, ensuring easy handling and retrieval without complex database solutions.
- **Performance Optimization**: The game must be optimized to run smoothly on typical consumer hardware, with minimal CPU or memory usage. It should have a responsive interface, ensuring that there is no lag or freezing during gameplay.
- **Error Handling and User Feedback**: The game must handle errors gracefully, providing users with clear, professional error messages when incorrect inputs or unexpected interactions occur. Feedback should be immediate and help users navigate their mistakes effectively.
- **Maintainability and Reusability**: The codebase should be modular, structured, and maintainable. It should adhere to clean coding standards, including

appropriate documentation and naming conventions. Reusable components should be developed to reduce redundancy and make future updates easier.

- **Version Control and Documentation**: All project files and code must be stored in a GitLab repository, and version control should be used throughout the development process. Design documents, diagrams, and other project artifacts should be maintained in the GitLab Wiki for transparency and collaboration.

- **Unit Testing**: Core game logic and backend functionality must be unit tested using JUnit. GUI interactions are not mandatory for testing, but backend features should be thoroughly validated to ensure reliability.

- **Self-Containment**: The application must not modify or delete files outside its installation directory, and it should not interfere with the operating system or other applications on the user's machine.

- **File Size Limitation**: The total file size of the project, including assets and code, should be under 500 MB to ensure that the game remains lightweight, easily distributable, and accessible to a wide range of users.

- **User Experience and Accessibility**: The game should be accessible to users with various abilities, including colour-blind players. The user interface must be designed with options for both keyboard and mouse navigation. Colour choices should ensure that essential elements are distinguishable by everyone.

- **Test and Balance Pet Needs Decay Rates**: The game should carefully balance pet needs decay to maintain challenge without overwhelming the player. Pets should not become too needy or passive. Properly balancing these needs will prevent frustration and ensure gameplay remains rewarding.

- **Balancing Mechanics**: The game should offer varying degrees of difficulty and time commitment, ensuring the game remains challenging yet fun. It should appeal to a wide range of players by providing a rewarding experience that encourages ongoing engagement with virtual pets.

- **Educational Components**: The game should subtly integrate aspects that teach responsibility, such as managing pet care routines. This will engage players and help them feel connected to their virtual pets while imparting useful life lessons.

- **Cross-Platform Functionality**: The game must be able to run on Windows, macOS, and Linux, ensuring compatibility with various development environments that team members have access to. The game should also function uniformly across different screen sizes and input methods (keyboard, mouse, touch).

- **Input Validation**: The game must validate user input to prevent the submission of inaccurate or harmful data, ensuring the integrity of the game experience and protecting against corruption or unauthorized changes.

- **Software Engineering Best Practices**: The overall design and implementation of the game must follow sound software engineering principles, ensuring reliability, efficiency, and maintainability over time.

- **Compliance with Project Specifications**: All functional components and specifications must be implemented as per the requirements outlined in the project documentation. Software engineering best practices should be adhered to, ensuring organized and structured development throughout.

- **Pet Progress and Data Security**: The pet's status and player progress must be accurately saved and loaded between sessions, ensuring that data is not lost. All player data should be securely preserved to prevent corruption or unauthorized access.

2.6 Summary

This document outlines the development of a Java-based virtual pet game designed to provide players with an engaging and educational experience that mirrors the responsibilities of real pet ownership. The game targets children, young teens, casual gamers, and pet lovers, emphasizing routine management, emotional bonding, and long-term engagement through interactive gameplay. Key features include pet customization, dynamic needs management, real-time progression, and a user-friendly graphical interface (GUI). The project integrates core software engineering principles, such as object-oriented design, data persistence, and performance optimization, to ensure a robust and maintainable application. The game's mechanics are designed to balance challenge and fun, with features like adaptive difficulty, reward systems, and interactive activities such as feeding, grooming, and playing. These mechanics encourage regular engagement by gradually decreasing the pet's well-being over time, fostering a sense of responsibility and care. Additionally, the game includes customization options, mini-games, achievements, and a save/load feature to enhance player retention and personalization. Beyond entertainment, the project demonstrates proficiency in Java programming and software engineering best practices. It addresses common challenges in the virtual pet game domain, such as repetitive gameplay, accessibility barriers, and balancing realism with playability. Non-functional requirements, including cross-platform compatibility, error handling, and data security, ensure a polished and reliable product.

In summary, this project combines technical expertise with creative design to create a virtual pet game that is not only entertaining but also reinforces the values of responsibility and consistency. By leveraging Java and adhering to software engineering principles, the game serves as both a functional application and an educational tool, appealing to a wide audience of players and stakeholders.

Table 2.6.1 Terms, Acronyms and notations

Term	Definition
Virtual Pet	A digital representation of a pet that requires care and interaction.
GUI (Graphical User Interface)	Graphical User Interface; the visual interface through which players interact with the game.
OOP (Object-Oriented Programming)	Object-oriented programming; is a programming paradigm that structures code into reusable objects representing real-world entities.
Pet Customization	The ability for players to personalize their pet’s name, appearance, and species.
Time-Based System	A game mechanic where the pet's well-being decreases over real-time intervals, encouraging frequent check-ins.
Mini-Games	Small, interactive games within the larger game, are designed to increase player engagement and pet happiness.
Save/Load Feature	A system that allows players to save their progress and resume later, preserving their pet’s status and overall game data.
Stat Management	The process of tracking and updating key attributes such as hunger, happiness, and health for the pet.

Progression System	A system that rewards players with unlockable items, abilities, and achievements as they care for their pets.
API	Application Programming Interface; a set of tools that allow different software systems to communicate.
JSON	JavaScript Object Notation; is a lightweight data-interchange format used for storing and transmitting data.
Unit Testing	A software testing method where individual components or functions of the game are tested for correctness.
Cross-Platform	The ability of the game to run on multiple operating systems such as Windows, macOS, and Linux without issues.
AI	Artificial Intelligence; simulates pet behavior and responses dynamically.
XML/CSV/TSV	Data storage formats are used for saving game progress and pet statistics.
JUnit	A testing framework for Java, used to validate core game logic.
Adaptive Difficulty	A feature that adjusts gameplay challenges based on player skill or progress.

Real-Time System	A game mechanic where pet needs decrease over time, requiring regular player interaction.
Data Persistence	The ability to save and load game data, ensuring continuity between sessions.
Virtual Pet	A digital representation of a pet that requires care and interaction.
Event Loop	A programming structure that continuously listens for and responds to user actions.
Java	A programming language used to develop the game, known for its portability and OOP capabilities.
Data Storage	The method of saving game-related data, such as pet statistics and player progress, locally on the user's device.
Error Handling	A system that provides clear, professional error messages when incorrect inputs or unexpected interactions occur.
Maintainability	The quality of the codebase is modular, structured, and easy to update or modify.
Version Control	A system (e.g., GitLab) used to track changes to the codebase and collaborate effectively during development.

Performance Optimization	Ensuring the game runs smoothly on typical consumer hardware with minimal CPU or memory usage.
Balancing Mechanics	Adjusting gameplay elements to maintain a balance between challenge and fun.
Educational Components	Subtly integrating aspects that teach responsibility, such as managing pet care routines.
Input Validation	Ensuring user inputs are accurate and safe, preventing errors or harmful data submissions.
Software Engineering Best Practices	Following structured, efficient, and maintainable coding standards throughout development.
Accessibility	Designing the game to be usable by players with varying abilities, including features like colorblind modes and customizable controls.