Final Project – Furry Fists: Tails of Combat

Matthew Luna, Gabriella Marrero, Tanuj Kancharla, Patrick Kelly

Professor Siddhartha Bhattacharyya

December 12, 2024

**Final Report**

Our team has worked towards the completion of our game: *Furry Fists: Tails of Combat.* This is a turn-based fighting game featuring combat between a selection of various animals: a cat, dog, dolphin, rat, wolf, panda, goat, and sloth, utilizing many of the common features of similar games in the field, such as Pokemon or Fire Emblem.

1. This project was selected through our group’s mutual interest in game design and development. Specifically, we determined a turn-based fighting game would be something both interesting and implementable for our team. Initially, the simplicity of a cat vs. dog fighting game proved to be appealing to our group as a manageable scope for an introductory project, while still remaining enjoyable and engaging for all members. Despite our interest in game design, our team did not have personal experience in this sort of system design process; however, all team members have a handful of experience as consumers of video games, making us well-versed in appropriate terminology, values, and general requirements of a turn-based game. This also provided a great creative opportunity for us. As consumers, we have a good understanding of features that we have liked or disliked in games we have played previously. We understood that designing a game ourselves would give us freedom to explore mechanics, characters, and visuals in a way we saw fit. By balancing the playful and cute theme of the pets with strategic gameplay, we determined our project idea could appeal to a broad audience and improve our own skills as developers in teamwork, problem-solving, coding, and design.

2. Building on the motivation to create a fun, engaging, and manageable project, we carefully considered the practicality and feasibility of our concept. The decision to develop *Furry Fists* as our group project was not just inspired by our shared interest in video games and animals, but also by how well this aligned with our learning objectives and constraints of the course. CSE2410 has three major course objectives, as outlined in the syllabus: 1. To learn the basic skills of software engineering, 2. To explore various tools and methods applied to software engineering, and 3. To apply the learnt skills of software engineering to practice. Our project offered a unique opportunity to achieve these objectives by following a structured software lifecycle, managing software changes, and utilizing conventional engineering processes. The simplicity of a two-character, turn-based setup kept the project manageable while simultaneously being challenging enough to demonstrate understanding of the software lifecycle. It provided an opportunity to apply key engineering principles and skills through means of system design, modular development, and team collaboration as well as coding, debugging, and testing in a real-world environment. Additionally, we determined early on that there is easily enough flexibility for expansion, offering room for enhancements through the inclusion of new characters, attacks, or greater visual effects. This was beneficial, as, in the end we *did* end up expanding our game to include a large handful of additional animals and features.

3. The foundation of our game began with the determination of game mechanics and features we wished to implement. Our approach to requirements gathering involved the identification of core gameplay elements, defining UI needs, and exploring edge cases and rules. First, we brainstormed the essential components of a turn-based fighting game. Both initial characters, Cat and Dog, needed predetermined attributes through their Health Points (HP) and Skill Points (SP), with the ability to take specific actions during their turn, the possible moves or attacks they could make. Actions were thought out very carefully, as there are many decisions we could make for our game characters at this point. It was important to determine what features we liked and which might add too much complexity to our game. For example, many games like our own include the ability to use items to perform tasks, such as healing the character or providing a disadvantage to the opponent. We decided against implementing this feature, as too many concerns arose from it: How will the character get the items? Do we wish to add an entire new game mechanic for this feature?, etc. Rather, we determined that the characters can instead use their own base skills to perform similar actions, with the addition of SP. So, actions were determined to include basic attacks to deal some small amount of damage without any cost to the character, special moves requiring SP or HP to deal greater damage to the opponent, and utility actions, also utilizing SP or HP, to disadvantage the opponent (such as draining the opponents SP). Next, we drafted the User Interface. A user-friendly interface was needed to display real-time updates on health, SP, and actions. HP and SP could easily be signified through a bar or numbers displayed on the screen. We also determined the necessity to include a textbox for action feedback, displaying the results of actions made (ex. Cat lost 2HP!) as well as a menu to show brief descriptions of each action, ensuring clarity for the player. Lastly, we wanted our characters to reflect the gameplay itself. With this, we determined we wanted to include three sprites for each character: one neutral for when the player is making a selection, one for when the character is attacking, and one for when the character is taking damage. By combining these visuals, we can convey the actions being made with the gameplay to our users. We were mainly inspired by the visuals of a 2D turn-based game, once again similar to Pokemon, and drew out some screen designs from this. Though we briefly considered 3D character design, the cost of learning and implementing this outweighed the benefits. While, in the end, we ended up incorporating visual aspects of the game through ASCII artwork, the presentation and ideas conveyed remained the same, just utilizing an alternative method over displaying hand-drawn images. Lastly, we discussed what were probably the most important and complex requirements: addressing edge cases and defining clear rules of the game. This was the determination of win/lose conditions as well as the acknowledgement of certain instances that can occur in-game, allowing us to avoid potential flaws or unintended behaviours in the game logic. There were many things to consider here. If a character has insufficient SP, certain actions may become invalid. Likewise, some actions require the player’s own HP to perform, enabling a player to lose by reducing their own health to zero. Critical hits could lead to overpowered outcomes, especially in the case they occur too frequently or are disproportionate to typical damage values. We established some solid rules to follow to help address these concerns:

1. Actions requiring SP cannot be performed if the character's SP is zero, it is invalid.
2. If an action would reduce a player's HP below zero it is invalid.
3. If an invalid action is performed, users will be prompted to make a different action without ending their turn.
4. Inputs will be validated before executing any action. An action must be taken before the next turn begins. (A player may not skip their turn. Turn-tracker implementation.)
5. When a character’s HP reaches zero or below, the game immediately will declare the opposing player as the winner.
6. A probability mechanic as well as a multiplier will be utilized to define critical hits. (ex. What are the chances of making a critical hit? How much higher than standard damage will this hit be?)

These rules provided a basis for the game’s structure, ensuring balanced and fair gameplay with minimal logic issues, and could be tested extensively through gameplay to identify potential oversights.

Functional Requirements:

1. Character Selection:  
   The player should be able to choose from a list of characters before game starts
2. Opponent Selection:  
   After selecting the player’s character, the user should be able to choose an opponent character from the remaining set.
3. Turn-Based Combat System:  
   The game should alternate between the player and opponent with the player making the first move.
4. Attack Options:  
   Player must be presented with these options for their move:
   * Base Attack (5 damage, 0 SP cost)
   * Secondary Attack (dice-based damage, costs 25 SP)
   * Tertiary Attack (dice-based damage, costs 50 SP)  
     The player can select one of these options, and damage is calculated accordingly.
5. SP Mechanics:  
   Certain attacks consume SP. If the SP cost is not available, the player must revert to a base attack or another valid option.
6. SP Refund on High Roll:  
   If the player (or opponent) uses a secondary or tertiary attack and rolls a 4 or higher, they regain half of the SP used.
7. HP and Victory Condition:  
   Each character starts with 100 HP. When a character’s HP is reduced to 0 or below, the other character wins and the game ends immediately.
8. SP Bonus on Significant Damage:  
   If an attack reduces the opponent’s HP below 50, the attacker gains 20 SP (capped at 100).
9. No Damage Scenario:  
   There is a 10% chance that an attack results in no damage (e.g., the attacker “slips”), and the game must display a humorous message reflecting this outcome.
10. Menu Interactions:  
    The game must allow the player to navigate menu options, adjust audio volume (1 = off, 10 = full volume), and quit at any point during their turn selection by choosing the appropriate option .

Non-Functional Requirements:

* Usability:The text based interface of the game should be clear and easy to read.
* Performance:Background tasks need to be performed fast, but with small delays (1-2 seconds) used intentionally to improve pacing rather than due to performance bottlenecks.
* Maintainability: The code should follow a logical structure, with separate classes for characters, UI, and audio, making it easier to modify or extend in the future (e.g., adding new characters or attacks).

4)

Design Approach:

The game is structured into several classes to separate concerns:

Game Class : controls starting, selecting characters, running turns, ending

Player and Opponent Classes : Store HP, SP, and character data.

UI Class : Handles input/output operations, menus, and prompts.

AudioManager Class : Controls Audio, music and sound effects.

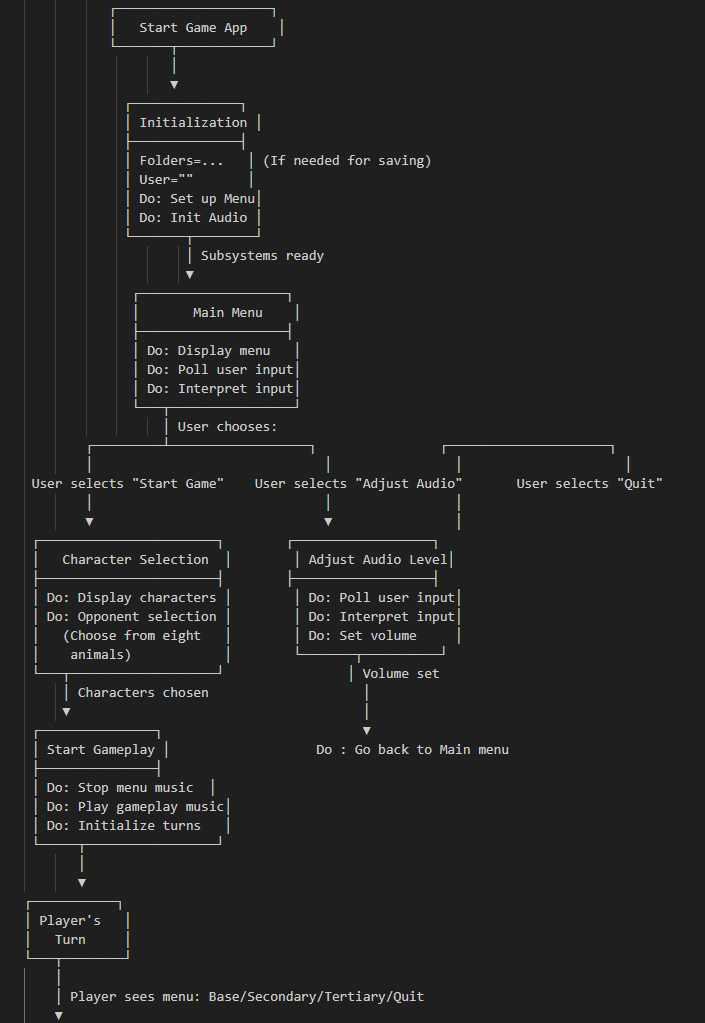
Dice Class : Provides random dice rolls.

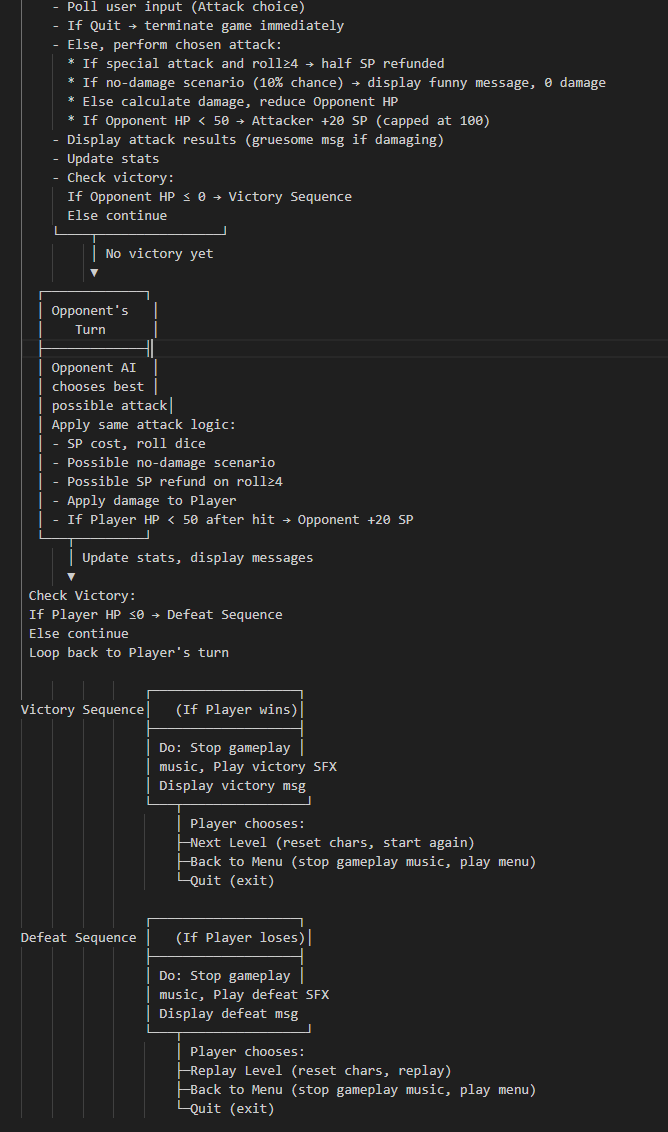
The Game class controls player and opponent turns and applies gamelogic from requirements .

The UI class focuses on user interaction, ensuring the player’s choices are obtained and displayed properly.

The AudioManager encapsulates all audio operations, allowing the code to remain flexible if audio resources or volume controls change.

The Character classes encapsulate stats and methods to reduce HP/SP, improving maintainability and clarity.





5)

Mapping Design to Code:

* The Game class main method starts the game, following the life cycle:
  1. Display the title screen (UI).
  2. Play menu music (AudioManager).
  3. Offer menu choices (UI).
  4. Setup game (choose characters), start gameplay music, and enter the turn-based loop.
* All the classes named in requirements are used in construction, there haven't been many changes. Code comments and structure show logical sections, ensuring that each part of the design has a corresponding implementation.

We used agile principles from the start to develop our project. We decided on building a basic prototype of the game and iterated over it expanding functionality with every iteration. This allowed us to find many problems early on and gave us time to figure out how to integrate the features without bugs.

**Testing strategies :**

**Static Testing:**

We used methods like code walkthroughs and review to verify the consistency in logic. For example we reviewed the flow of functions in the Game class to make sure it would alternate the turns between opponent and enemy after every attack and also victory conditions are checked immediately after each attack. By going over the code as a group we found ways to simplify logic and ensure critical variables(HP, SP..) are updated before being read by another method.

**Static review process caught a few early issues:**

Variable Initialization Errors: We noticed that some variables needed to be set to default when we transition from one mode to another. This also revealed some variables struggled to initialize before starting a new round in the game mode

Conditional Branching Oversights: During code reviews, we found instances where certain conditions, like the no-damage scenario, were not always checked in the correct sequence. Reordering these conditions clarified the control flow and prevented confusing outcomes during runtime.

**Dynamic Testing :**

This dynamic testing involved providing various inputs at runtime and confirming that the outputs matched our expectations.

**Process:**

* Implement a small feature (e.g., adding the tertiary attack).
* Run the program, select the tertiary attack option, and verify that SP costs and damage calculations were correct.
* If results deviated from expectations (for instance, SP was not refunded correctly when rolling ≥4), we revisited the code, examined the logic, and corrected the error.

**Common Issues and Fixes Identified During Dynamic Testing:**

SP Refund Logic Error: We initially found out that the SP refund function would sometimes refund even when the conditions were not met. The roll condition was placed incorrectly and this helped solve the issue.

Audio adjustment problem : Initially the audio adjustment slider did not work or would play the music at high volume. This was fixed with introducing a method to the AdioManager class to set the volume on an already playing clip.

No-Damage Scenario Check: Early versions of the no-damage scenario logic allowed damage to be applied before the scenario was checked, causing inconsistencies.

HP and SP Threshold Bugs: During testing, we discovered that when an attack dropped opponent HP below 50, the SP bonus was not always granted. This happened because the HP check was performed before the HP actually updated. We moved the HP update logic before the SP bonus check.

Audio Overlap Problem:During early iterations, when returning to the main menu from the audio settings or after finishing a gameplay session, the menu music would begin to play again without stopping the previous track. This resulted in two overlapping instances of the same music track, creating a jarring audio experience. We fixed it by introducing functionality to stop the main menu music playing before paling the background gameplay track and vice versa.

Iterative Development and Continuous Testing:

* Add/modify a feature.
* Perform static review (check code flow, logic consistency).
* Run dynamic tests (play through scenarios).
* Identify issues, fix them, and retest.

6. Tasks completed with group member name:

**1. Motivation for the Project**

* **Tanuj Kancharla (30%)**: Drafted the initial codebase in Java, laying the foundation for the project's functionality.
* **Matthew Luna (20%)**: Conceptualized the motivation and ensured it aligned with the team's vision.
* **Gabriella Marrero (25%)**: Wrote a clear articulation of the project's purpose in the final report.
* **Patrick Kelly (25%)**: Reviewed and refined the motivation section to ensure user-centric goals.

#### **2. Rationale for Project Selection**

* **Gabriella Marrero (40%)**: Presented compelling arguments for selecting this project, focusing on its creative and technical potential.
* **Tanuj Kancharla (25%)**: Highlighted the technical feasibility and challenges involved in the coding phase.
* **Matthew Luna (15%)**: Proposed the integration with Unity to enhance the project's scope.
* **Patrick Kelly (20%)**: Ensured the rationale was practical and user-friendly.

#### **3. Requirements Gathering**

* **Matthew Luna (30%)**: Transitioned the project into Unity, identifying gaps in the original design that required additional requirements.
* **Tanuj Kancharla (25%)**: Ensured technical feasibility and gathered coding-related requirements.
* **Gabriella Marrero (25%)**: Focused on visual and artistic requirements for an engaging user experience.
* **Patrick Kelly (20%)**: Streamlined the requirements to balance complexity and usability.

#### **4. Software Design and Modeling**

* **Tanuj Kancharla (35%)**: Developed the Java architecture, emphasizing modularity and security.
* **Patrick Kelly (30%)**: Reviewed the design for usability and logical consistency.
* **Matthew Luna (20%)**: Adapted the design for Unity, ensuring compatibility with 2D gameplay.
* **Gabriella Marrero (15%)**: Designed artistic models and UI elements to align with the software design.

#### **5. Construction and Testing**

* **Patrick Kelly (30%)**: Ensured the code was optimized and tested from a user perspective.
* **Matthew Luna (30%)**: Verified Unity integration and ran gameplay tests for seamless interaction.
* **Gabriella Marrero (20%)**: Validated the visual assets during gameplay and ensured design consistency.
* **Tanuj Kancharla (20%)**: Conducted unit tests on the Java code to identify and fix bugs early

At the end of the day we all worked very well together, communicated with each other efficiently about each report that needed to be written up and the actual design of our software. We all may have done different aspects of the project, but we all put in equal hard work to get this project completed.

7)

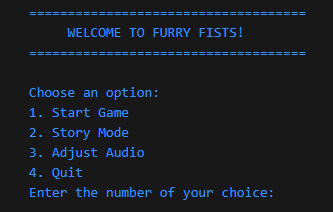
Our team successfully implemented a functional prototype of Furry Fists: Tails of Combat, a turn-based fighting game that met our project objectives. The game features dynamic combat mechanics, allowing players to select characters, execute strategic attacks, and manage resources like Health Points (HP) and Skill Points (SP). Initially planned as a simple two-character system, the project expanded to include a roster of animal characters, showcasing the flexibility and scalability of our design.

Core features, such as attacks with varying SP costs, critical hit probabilities, and SP refunds for high rolls, were implemented effectively. Rigorous testing ensured balanced gameplay and addressed edge cases like invalid moves or insufficient SP for actions. The game also includes a humorous "no damage" mechanic, adding a playful element to combat. The user interface, though ASCII-based, was designed to clearly display HP, SP, and action feedback, ensuring an intuitive experience.

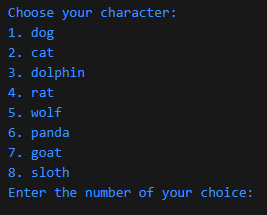
The modular design approach, with distinct classes for game logic, UI, characters, and audio, facilitated efficient debugging and feature integration. Static and dynamic testing helped resolve issues like SP refund logic errors, audio track overlaps, and initialization bugs. Iterative development allowed us to refine gameplay mechanics and incorporate feedback from playtesting, ensuring fairness and enjoyment.

Overall, Furry Fists: Tails of Combat achieved its goals, demonstrating the practical application of software engineering principles. While the game is a text-based prototype, it provides a solid foundation for future enhancements, such as graphical interfaces, AI opponents, and expanded gameplay mechanics. This project allowed our team to develop valuable skills in coding, design, and collaborative problem-solving.

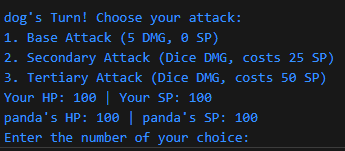
Main Menu Screen:



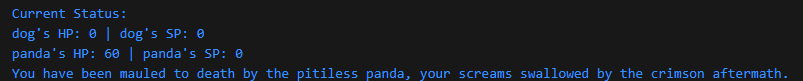
Character Selection Screen:



Combat Interface:



Game End Screen:



8)

Furry Fists: Tails of Combat began as a concept to create a simple yet enjoyable turn-based fighting game. Through structured requirements, careful design, and modular code construction, the project successfully realized a playable text-based combat system. Players navigate menus, choose characters, execute attacks, and manage resources like HP and SP while engaging with random elements that keep the gameplay fresh.

Though limited in scope compared to commercial games, this project allowed for the practical application of software engineering principles such as requirements gathering, layered design, modular construction, and volume control integration. Future enhancements could include more complex AI logic, additional attacks or items, and improved visuals or a graphical interface. We also were able to produce a fully developed version of the game but with few usecasses implemented to demonstrate how a fully realized game with proper visuals and graphics would look like.