1. Introduction

This project focuses on the development of a turn-based combat game prototype implemented in Python. The main purpose of the project is to produce a game template that could be used to create any polished turn-based combat game. While this project has its own unique points which are not standard for a template, main point was to have a working stable version with minimal diversions. Game development was chosen as the domain because it naturally requires structured code, state management and multidisciplinary thinking. Also as the sole developer of the project, personally I wanted to work on game development in a vastly different environment from what I was used to.

There were multiple motivations that pushed me in this direction for this project. Some of them were personal. Mainly, working in a different environment and the idea of having to deal with all rendering really piqued my interest. What I could count as not so personal maybe is that one of my previous projects was about a text-based adventure game. It had a huge scope with advanced text-based AI implementations and economic simulations, which resulted in me putting the project on a backlog so it can improve over time. And throughout development there were always the question of how “fun” would be provided to the players. And with this project, I believed I could set a good turn base combat system that could be carried over with right modifications.

2. Methodology

The project was developed using a modular and object-oriented methodology. The overall design separates game logic, rendering and data representation into independent modules as much as possible. This approach was chosen keeping in mind that it reduces coupling between components and allows individual parts of the system to be modified or extended without affecting the entire codebase. Although, there have been some problems implementing this methodology in the development process.

Object-oriented programming principles were used extensively. Core entities such as characters, skills and encounters are represented as classes, each responsible for its own data and behavior. Inheritance was avoided where unnecessary, favoring composition to keep the design simple and flexible.

During the design process, Python’s strengths and weaknesses were kept in mind. While my personal experience as a developer did not exactly align with Python’s strengths, there was an active effort in avoiding fighting against the environment in hand. Python and its library pygame provide a more rapid deployment opportunity compared to other game development tools. That was utilized throughout the development process to always test the waters with new and big ideas to adjust the scope accordingly to the recent prototype in hand.

According to my game development experience, one of the most valuable things in development process is to have an achievable short-term goal throughout the process. Because of that, main goal was to have a clean and modular structure before the development began. To ensure that, modular class structures were tested by code as project environment provides quick compilation that visualizes current progress.

The development process followed an incremental approach after the design process is done. Basic character and skill systems were implemented first, followed by rendering and encounter management. Features were tested as they were added, allowing issues to be identified early. This iterative workflow helped maintain stability while gradually increasing complexity.

3. Implementation

The system is structured into multiple core components:

Character System (character.py): Defines character attributes such as health, speed, sanity, skills and alive status. It also handles damage, death and turn-based speed calculation.

Skill System (Skill class): Each skill has damage, target rules and an implementation function. Skills include hit chance logic and interacting directly with character objects.

Encounter System (Encounter class): Each encounter has its own set of enemies and their positions in the level. Helping with modularization of targeting, collision, rendering.

Game Logic (game.py): This component is responsible with initialization and states of the game. To play the game, this program must be ran. In the initial design, this program was planned to be responsible with all the calculations and collision. That design was not followed through as inexperience in development resulted in rendered images and their colliders being in the same scope.

Render System (renderer.py): Responsible for most of the functions that are necessary to keep the game running. Skill association system, rendering system, collision system are all in renderer.py. This program was not implemented according to the design file, as modularizing rendering system and collision system proved to be very difficult and slowed down the development process to a halt. To compensate for the lost time and complication in the code management, renderer.py acted more like a singleton.

Assets: Sprites are grouped under folders such as characters, encounters, skills, UI and backgrounds to keep the project organized.

All these systems come together to run a smooth experience. Giving the player a standard story to read through and enabling the player to think of a strategy from presented playable characters to select from and make tactical plays in the combat part.

4. Results and Evaluation

The implemented system successfully achieves its core goals. The game initializes correctly, loads assets and allows the player to progress through combat encounters. Characters are rendered on screen with their associated skill and both player and enemy characters can perform actions during combat.

The turn-based combat logic functions work as intended. Skills can be selected, targets are validated based on predefined rules and damage calculations affect character health correctly. Characters transition to a defeated state when their health reaches zero, preventing further actions.

Multiple encounters with different enemy compositions were tested and the system handled these scenarios without requiring changes to the core logic. This confirms that the encounter system is mostly reusable and scalable.

From a design perspective, the modular structure proved effective. Separating character logic, rendering and game control simplified debugging and made it easier to add new characters or skills. The use of object-oriented principles reduced code duplication and improved readability.

Object-oriented approach made it very easy to add UI elements later on the development process as well. But the rendering system and the collision system were too related to each other. This has proved to be a problem when there was a bigger enemy with bigger sprite sizes. As my rendering system used a constant sprite size to decide on the image sizes, collider sizes and the UI, the solution was not very efficient or preferable, especially while trying to keep a certain level of scalability.

In terms of performance, the game runs smoothly for the intended scale of the project. Since the number of active objects on screen is limited, no performance bottlenecks were observed during testing. Asset loading and rendering remained stable throughout gameplay.

Despite working as intended, the system has some limitations. Enemy behavior relies on simple or random decision-making, which limits strategic depth. Sound effects, visual feedback and animations are minimal, making combat harder to read at times. Additionally, balancing character stats and skills was not a primary focus and would require further tuning.

5. Conclusion and Future Work

This project demonstrated how a turn-based game prototype can be built using Python with a clear modular and object-oriented structure. Through the implementation of characters, skills, encounters and rendering logic, the project successfully met its primary goal of creating a functional and extendable combat system. The development process highlighted the importance of separating responsibilities and maintaining readable code when working with interactive systems.

For future work, the first priority would be to resolve the tight coupling between the rendering system and the collision detection logic. Improving this separation would make the system easier to scale and reduce complexity when adding new features. Further modularization of these systems would help establish a stronger template structure that can be reused for future projects.

Additional improvements could include enhancing enemy AI, expanding skill mechanics, refining visual feedback (mostly with animations) and balancing gameplay elements. With improved structure for skill class, more systems could be added to future games such as a buff system. After these changes, the project could evolve from a prototype into a more complete and maintainable game framework.