50.051 Programming Language Concepts

Final Report - Text-Based Adventure Game (RPG)

Name	Student ID	Email
Ethan Wong Shao Wei	1006899	ethan_wong@mymail.sutd.edu.sg
Priyanshi Saraogi	1007144	priyanshi_saraogi@mymail.sutd.edu.sg
Hareendran Kallinkeel Harsh	1007116	harsh_hareendran@mymail.sutd.edu.sg

Introduction: Project Overview and Vision

“Legacy of Yharnam: Shadows of the Forgotten” is a narrative-driven, text-based role-playing game (RPG) developed in the C programming language. Set in a cursed kingdom plagued by darkness, the game places the player in the shoes of a chosen hero tasked with vanquishing the Dark Lord and restoring balance. Inspired by classic RPG mechanics and interactive fiction, the game emphasises player choice, strategic decision-making, and immersive storytelling through a modular and technically robust C codebase.

Our primary objective was to design a modular, extensible system that incorporated dynamic gameplay, robust state management, player progression, and both textual and binary file save/load capabilities. The project was implemented with the aim of delivering an engaging gameplay experience, while satisfying the given constraints regarding functionality, modularity, and user interaction.

System Architecture and Feature Breakdown

The project follows a modular programming paradigm, with distinct functional components:

Core Gameplay Modules

- **main.c:** Entry point of the application, handles the main menu, initialisation, and game loop kickoff.

- **exploration.c:** Manages overworld traversal, forked path decisions, and branching quest logic.
- **quests.c:** Includes puzzle mechanics, crystal quests, and crafting of the Master Sword.
- **combat.c:** Implements turn-based battles including mini-bosses and the final Dark Lord confrontation.
- **player.c:** Manages player stats, experience system, leveling mechanics, and inventory state.
- **common.c:** Contains utility functions like print_pause(), input handling with input validation, and the in-game pause menu.
- **save.c:** Enables save/load functionality in both human-readable (.txt) and binary (.dat) formats, with checksum validation for file integrity.

Game Loop Design

Players must acquire two essential relics (Map Fragment and Mystic Herb), solve a riddle-based puzzle in the Old Sage Tower, collect elemental crystals from dungeon challenges, and finally craft the Master Sword to face the Dark Lord.

Constraints & How They Were Met

Constraint	**Implementation**
Modular source files	Divided into logical modules (`combat.c`, `quests.c`, `save.c`, etc.) for separation of concerns.
Save/Load system	Implemented `save_game()` and `load_game()` with both `.txt` and `.dat` formats and data validation.
Input handling	Defensive programming used in `get_int_input()` to handle invalid inputs and prevent buffer overflows.
Replayability	Player choices (fight vs stealth, which crystal dungeon to visit first, etc.) lead to replay value.
Game state management	`GameState` structure tracks location and current player stats

	throughout sessions.
Error handling and recovery	Failed inputs, save file corruption, and death scenarios handled gracefully with re-entry logic.

Gameplay Mechanics

1. Exploration:

- **Overworld navigation:** A sequence of directional choices simulates map-based movement. Wrong choices return the player to the village.
- **Forked paths:** Players choose between two critical quests: retrieving the map fragment from bandits or the mystic herb from a forest beast.
- **Replay system:** If the player fails or chooses unwisely (e.g., charging into the bandit camp), they are reset with consequences (loss of XP/items).

2. Puzzle Solving:

- **Sage Tower Puzzle:** Riddle-based selection mechanic requiring the player to deduce correct clues about the crystal locations.
- **Ice Dungeon Riddle:** A free-form string answer puzzle that requires the word “ice” or “ice crystal” to proceed.

3. Combat System

- **Turn-based fights:** Implemented as loops with branching attack options and dynamic damage calculation based on player stats.
- **Boss fights:** Scaling difficulty between mini-boss and final boss with varying dialogues and outcomes based on performance.
- **Leveling and stat progression:** Experience points earned during combat increase player level, boosting health, attack, and defense stats.

4. Inventory and Crafting

- The player must gather three crystals—Fire, Ice, Shadow—each unlocked through exploration or puzzle-solving.
- Once all are acquired, along with Rare Ore from a maze, the Master Sword is forged, unlocking the final dungeon.

5. Save and Load System

- Integrated a user-triggered pause menu that allows mid-game saving.
- Added a save-slot system with 4 slots and selectable format.
- Validates saves via a checksum algorithm to ensure integrity and detect file corruption.

Development Process

Design & Planning

The project began with narrative design and a state machine diagram for progression logic. We mapped the critical paths, from initial exploration to final boss, and defined which decisions would affect outcome or route taken.

Task Division

Each team member was responsible for distinct parts of the project, contributing to both the core functionality and the integration of the final product:

- **Ethan**: Worked on `exploration.c`, building the main game flow and connecting all game states through the storyline and navigation logic.

- **Priyanshi**: Worked on `quests.c`, implemented the load and parsing feature in `save.c`, supporting both text and binary formats, and error handling.

- **Harsh**: Developed `combat.c`, creating the battle system, and implemented the save feature in `save.c`, and contributed to the overall game logic.

Challenges Faced

- **Input Validation**: User input had to be sanitised carefully—especially in puzzles and numeric input fields.

- **State Transitions**: Designing a smooth and accurate progression between game states without unintended loops was complex.

- **Game Balance**: Combat stat scaling had to be manually tested to avoid unfair fights or overly long battles.

- **Save File Corruption**: Early versions of the save/load system didn't validate data. We later added a checksum to check for any tampered/corrupted data.

Lessons Learned

Through this project, we gained a deeper understanding of how to design and structure a modular C program. Separating the code into multiple `.c` files not only improved readability but also made it easier to develop and test each component independently. Implementing a reliable save/load system across both text and binary formats taught us the importance of careful data serialization and validation. It also reinforced our understanding of file I/O, parsing techniques, and the use of checksums to maintain data integrity. Additionally, the need for robust and user-friendly input handling pushed us to practice defensive programming, ensuring smooth gameplay even in the face of unexpected user input.

From a collaboration perspective, we learned the importance of clear task ownership and consistent communication. Dividing work by modules allowed us to work in parallel without stepping on each other's code. Frequent testing and playthroughs helped us identify design flaws or logic issues that weren't immediately visible in the code itself. Using Git effectively—committing regularly with clear messages—was crucial for managing changes, resolving merge conflicts, and keeping our development history organized. These habits made our teamwork smoother and helped ensure the final product came together successfully.

Conclusion

Legacy of Yharnam: Shadows of the Forgotten successfully delivers an immersive RPG experience while meeting all functional and structural requirements laid out in the project brief.

From combat mechanics and puzzle-solving to modular code and a functioning save/load system, the project showcases what's possible within the constraints of a systems-level language like C.

We leave the project with a stronger grasp on software architecture, collaborative workflows, and the challenges of game development—even in a purely text-based environment.