CMPT 276 PHASE 4
TEAM 3

Simarjot Singh | 301615711
Raymond Zhou | 301464055
Maria Leon Campos | 301577820
Shahmeer Khan | 301623019

1. Introduction

Phase 4 focused on finalizing features and completing our testing suite. Our goal was to integrate all modules, refine usage of design patterns, and ensure the game was functional, maintainable, and testable according to guidelines. Our artifacts for JavaDocs and our JAR file are located in the root directory.

2. Gameplay Summary

Monster's Den is a 2D dungeon style game where the player navigates a grid, collects, rewards, avoids punishments and defeats enemies using different sword types. All actions occur in ticks, providing pacing. To win, the player must collect all regular rewards and reach the exit. Otherwise they will lose when caught by an enemy, or when their score drops below zero.

For more details, check out the video demo. The final game aligns with our original Phase1 plan, with core logic maintained and smaller implementation details refined over development phases.

3. Architecture & Design

Our architecture follows modular object oriented principles.

Module overview:

- `core/` Game loop, Singleton, Game, Board, Cell
- `characters/` Player, Enemy, Monster, HumanEnemy
- `items/` Swords, rewards, punishments
- `patterns/` Strategy, Factory, Observer, Command
- `util/` Direction ENUM
- `ui/` Console/UI observers for score and timer

Design Patterns used:

- **Singleton:** Game maintains global state and tick system
- **Strategy:** Defines flexible movement behaviours (player vs. enemy AI)
- **Factory:** Creates swords and enemy types
- **Command:** Handles UI logic

These patterns reduce coupling, follow the “open for extension closed for modification” principle, and align with design principles taught in the course.

4. Testing Approach

4.1 Functional Testing

Various specification based testing were applied using equivalence partitions and boundary values. Tests covered movement/map boundaries, player-enemy collisions and combat, reward/punishment scoring, map loading, and win/lose conditions.

4.2 Structural Testing

Using JaCoCo, we measured line and branch coverage. We added tests to target Strategy branches (random vs deterministic movement), board edge conditions, and scoring logic (bonus timers, punishments).

4.3 Integration Testing

Since the game is tick-based, we tested multi-step interactions such as collision resolution (caused by players and enemies), reward collection triggering UI observer updates, and sword switching (affecting combat outcomes).

5. Tools used

- Java 17
- Maven for build + test automation + to generate JAR file + Javadocs
- JUnit 5 for testing
- JaCoCo for coverage
- JavaFX for UI

6. Conclusion

Phase 4 brought all components of our project together into a complete, functional game. By

refining our architecture, stabilizing gameplay behavior, and strengthening our test suite, we confirmed the effectiveness of our design patterns and earlier decisions. This phase reinforced key software engineering skills: modularity, maintainability, and systematic testing, while giving us practical experience with collaboration and full-cycle development. Overall, we delivered a clean and extensible final codebase that meets the learning objectives of CMPT 276.