CMPT 276 PHASE 2
TEAM 3

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

Monster's Den - Implementation

1. Overview of Phase 2 Implementation

Phase 2 focused on translating the Phase 1 design into a working, modular Java implementation of *Monster's Den*. The team delivered a playable build demonstrating core gameplay (movement, enemy AI, rewards/punishment, win/lose conditions) and a functional UI all while adhering to Maven's standard project layout and using the GoF design patterns taught in-class (Singleton, Strategy, Factory Method, Observer, and Command).

In this phase, our objectives were:

1. Produce clean, extensible architecture
2. Ensure the game can be build and ran with Apache Maven
3. Document key design decisions and lessons learned.

---

2. Approach and Design Rationale

Our team kept architecture close to Phase 1 but made changes during implementation to simplify integration and improve cohesion.

`Game` (Singleton) manages loop, scoring, and observers; `Board` loads maps and provides helpers; factories centralize creation of weapons/enemies; and input uses the Command layer. The UI is modular—`GameUI` (console) and `GameUIFX` (JavaFX)—both driven by Observer updates.

---

3. Division of Responsibilities

Team Member	Main Responsibilities	Key Classes / Packages
Simarjot Singh	Core Game System / Integration	`Game`, `Board`, `Cell`, `Main`
Raymond Zhou	Character and Movement Logic	`Character`, `Player`, `Enemy`, `Monster`, `HumanEnemy, UI`
Shahmeer Khan	Item and Reward System	`Sword`, `SteelSword`, `SilverSword`, `Reward`, `BonusReward`, `Punishment`

Maria Leon Campos	Design Patterns	`patterns/strategy`, `patterns/factory`, `patterns/observer`, `patterns/command`, `patterns/observer`, `patterns/strategy`,

Each member contributed through Git commits and integration testing within Maven.

---

4. Implementation

- **Build & Layout (Maven):** Follows standard structure (`src/main/java`, `src/test/java`, `src/main/resources`). The `exec-maven-plugin` runs either the console or JavaFX entry point by switching the `mainClass` in `pom.xml`.
- **Core Logic:** `Game` (Singleton) controls initialization, scoring, tick timing, and win/loss conditions. It loads maps, spawns entities, and notifies observers each update.
- **Map Loading:** `Board` reads `maps/level1.txt` from resources (with a dev fallback). Each `Cell` stores terrain and contents, with helpers for bounds and collisions.
- **Characters & Combat:** `Player` and `Enemy` share core movement through `Direction`. AI behavior is defined via **Strategy** (`MovementStrategy`, `ChaseMovementStrategy`). Weapons are type-sensitive (`SilverSword` vs `Monster`, `SteelSword` vs `HumanEnemy`), with rewards and punishments updating the score.
- **Design Patterns:**
  Singleton (`Game`), Strategy (`MovementStrategy`), Factory (`EnemyFactory`, `WeaponFactory`), Observer (`GameObserver`, `GameUI/GameUIFX`), Command (`MoveCommand`).
- **User Interface:**
  `GameUI` (console) renders a symbolic grid with live stats, while `GameUIFX` (JavaFX) displays tiles, HUD elements, and keyboard input via `MoveCommand`. Both rely on Observer notifications.

---

5. External Libraries and Tools

- **Apache Maven** – build automation, dependency management.
- **OpenJFX (JavaFX: controls, graphics, fxml)** – graphical UI rendering and input (declared in pom.xml as org.openjfx artifacts with a Java11 compatible version)
- **Exec Maven Plugin** – Runs either console or JavaFX entry point via mainClass setting.
- **Java SE 11** – toolchain target (consistent with course requirements)

6. Code Quality and Documentation

We aligned with course expectations on readability and maintainability:

- Cohesive packages and clear class responsibilities (core, characters, items, patterns/*, util).
- Javadoc headers on core classes and key methods; descriptive, action-first method summaries.
- Meaningful enums/utilities (Direction, AIState, AStarPathFinder) to keep logic focused.
- Maven-driven, IDE-agnostic configuration; resources packaged under src/main/resources.
- Consistent commit messages focused on vertical slices (UI, factory, strategy, map/board, bug fixes

7. Challenges and Lessons Learned

JavaFX on Java 11: Getting the correct OpenJFX dependencies and the JavaFX Maven plugin aligned with the local JDK caused initial runtime errors (missing modules). We resolved this by pinning *org.openjfx* versions and using the plugin's `run` goal for development, plus the `exec-maven-plugin` for CI-friendly invocation.

- **Resource Loading (maps):** Packaging and dev-path differences surfaced (resources vs filesystem). We implemented a resource-first loader with a secondary dev-path fallback and added clear exception messages so map issues can be diagnosed easily.

- **Integration Order:** Enemy AI, collision rules, and bonus timers interact across modules. We stabilized behavior by moving game-state transitions into `Game`, pushing pure rendering into observers, and routing all player input through the Command layer.

- **Time Constraints:** When the GUI became unstable during integration, the team temporarily used the console observer to validate core logic. This minimized risk and allowed JavaFX finalization to proceed later without blocking progress.

8. Next Steps (for Phase 3)

- **Map Updates:** Add at least one complete level under `resources/maps/` and make sure the game handles missing or broken map files properly. A small map generator could help with quick testing.

- **UI Improvements:** Add start and pause screens, animations, and sound effects to make gameplay feel smoother. A simple settings menu for difficulty or speed would also improve the experience.

- **Code Cleanup:** Make the code more reliable by adding error checks, limiting access to shared data, and keeping all constants together in a `GameConfig` file for easier changes.

- **Performance Testing:** Check how the game runs on larger maps with more enemies to ensure it stays responsive.

---

9. Conclusion

The team delivered a functional implementation that matches our Phase 1 intent and satisfies Phase 2 requirements: a Maven-based, pattern-driven Java game with both console and JavaFX user interfaces. The architecture is modular and extensible, enabling straightforward testing and UI enhancements in the next phase.