CMPT 276 PHASE 1
TEAM 3

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

Monster's Den – Overall Plan and Game Description

Overview
Monster's Den is a 2D dungeon-maze adventure game inspired by *The Witcher* universe. The player controls **Gerald**, navigating through a grid-based dungeon filled with monsters, hostile humans, rewards, and traps. The objective is to collect all **regular rewards** and reach the **exit cell** without being defeated or letting the score fall below zero. The game balances strategy, timing, and combat decisions within a structured and replayable environment.

Core Concept
The player begins at the dungeon's **start cell** and must reach the **exit cell** after collecting all regular rewards.

- **Monsters** are moving enemies vulnerable to the **silver sword**.
- **Human Enemies** are moving enemies vulnerable to the **steel sword**.
- **Regular Rewards** increase score and are required to win.
- **Bonus Rewards** appear temporarily for extra points.
- **Punishments** reduce score and may trigger a loss if the total drops below 0.

The player can switch weapons at any time, choosing the correct sword for each encounter. Movement and combat occur in discrete **ticks**, ensuring predictable pacing and strategy.

Architecture and Design Patterns
The game is implemented in **Java** and follows modular object-oriented design.

- **Singleton (Game):** Maintains a single running game instance, managing state, score, and observers.
- **Factory Method / Abstract Factory:** `EnemyFactory` and `WeaponFactory` create enemy and sword types without exposing creation logic.
- **Strategy:** `MovementStrategy` defines flexible AI behaviors such as chasing or random movement.
- **Observer:** `GameUI` observes the game for real-time score and timer updates.
- **Command:** `MoveCommand` and `AttackCommand` encapsulate player actions for cleaner input handling and future undo support.
- **Composite:** The `Board` is composed of `Cells`, each containing entities (rewards, enemies, punishments), supporting scalable level design.

Gameplay Flow
1. **Start Game:** Loads map, spawns player and enemies, initializes UI.
2. **Move Player:** Executes movement using `MoveCommand`; triggers game tick.
3. **Enemy AI:** Each enemy uses its `MovementStrategy` to pursue the player.
4. **Combat:** Player attacks with the correct sword type; wrong weapon deals reduced damage.
5. **Collect Rewards / Punishments:** Updates score and notifies observers.
6. **Win/Loss:** Player wins by reaching the exit after all rewards or loses if caught or score < 0.

Goal
Deliver a polished, extensible dungeon game demonstrating mastery of key design patterns, strong object-oriented structure, and engaging gameplay through tactical combat and strategic navigation.