Dungeon Adventure Game Extension — Design & Implementation Report

1. Introduction

This report documents the design, architecture, and implementation of the extended Dungeon Adventure game. Building on an initial assignment, this version showcases key Object-Oriented Programming (OOP) concepts, custom data structures, and algorithmic puzzles in C++. The game combines a quest narrative with interactive elements: a logic grid puzzle, turn-based combat, NPC interactions, and a final boss encounter. Through this extension, we demonstrate mastery of classes, encapsulation, inheritance, polymorphism, STL containers, custom containers, and recursion.

2. Gameplay Overview

2 Objectives & Flow

1. *Entrance Hall: Solve a riddle (Challenge) and fight a Goblin (Enemy).*
2. *Dark Corridor: Navigate a 4×4 N-Queens footstep puzzle—each safe tile placement acts like a queen that cannot be attacked.*
3. *Quatro's Lair: Battle the multi-tentacled Quatro-Otopus boss through iterative tentacle destruction.*

4. *Treasure Room: Interact with an NPC (Mario), collect essential items (Sword, Shield, Key), and complete a spell code puzzle.*
5. *Dragon's Den: Final boss fight requiring strategic use of sword, shield, or magic potion.*

Core Mechanics

- **Turn-Based Combat with health tracking and action choices (attack, block, potion).**
- **Inventory Management using std::vector<Treasure> for collected items.**
- **Move Limits: Each forward/backward move decreases a move counter stored in Player.**
- **Game Over Conditions: Death, move exhaustion, or dragon defeat.**

Class Hierarchy & Encapsulation

Renderable (abstract)
├── Player
└── Enemy

Puzzle (abstract)
└── Challenge
Treasure
Room
Dungeon
Stack
Queue

- *Player: encapsulates name, health, moves; manages inventory (vector<Treasure>); methods: takeDamage(), heal(), collectTreasure(), showInventory().*
- *Enemy: holds type, health, attackPower; methods: takeDamage(), isAlive(), canAttack().*
- *Treasure: simple class with name property.*
- *Challenge: inherits Puzzle; stores a question/answer and solved flag; method askQuestion() with input sanitization using std::transform.*
- *Room: composition of Challenge, Treasure, and an EnemyList (custom Queue<Enemy>); method triggerRoomEvents(Player&) orchestrates puzzle, combat, and loot.*

- *Dungeon: doubly-linked list of Room*, with backtracking Stack<Room*>; methods for room navigation and display.*

2 Inheritance & Polymorphism

- *Abstract Base Puzzle defines virtual bool askQuestion() = 0; allowing extension into new puzzle types.*
- *Renderable interface for ASCII rendering methods (future UI abstraction).*

4. Data Structures & Algorithms

1 Custom Containers

- *Stack: singly linked Node<Room*> for backtracking; methods: push(), pop(), top(), isEmpty().*
- *Queue: singly linked Node<Enemy> for Room enemy lists; methods: enqueue(), dequeue(), peek(), isEmpty().*

2 STL Utilization

- *std::vector for dynamic arrays: inventory, original enemy backup, N-Queens positions.*
- *Algorithms: std::transform for case-insensitive input; std::abs for diagonal checks.*

3 Algorithmic Challenges

- *N-Queens Footstep Puzzle: O(n) safety check per placement; uses backtracking-like logic for user-driven queen placements.*
- *Randomized Combat: rand() seeded once in main(), varying damage types and attack patterns.*

5. Detailed Implementation

1 room2_darkCorridorFootstepPuzzle

1. Initialize 4×4 grid state in vector<pair<int,int>> queens.
2. Loop 4 safe placements: display grid, prompt input, validate bounds, isSafe(), update state or apply player.takeDamage(5).
3. Visual Feedback: displayGrid() shows Q for safe, X for last invalid, . otherwise.
4. Health/Death: early return if player.isAlive() == false.

2 room3_quatroFight

1. Display ASCII tentacled creature.
2. Combat Loop: Quatro attacks (15 or 5 damage), player loses or proceeds to destroy up to 2 tentacles via cin >> input.
3. Victory: collect Treasure("Quatro Tentacle"); return true.

3 room4_npcInteraction

- Dialogue branches: collecting items, casting "dragonbegone" spell to gain potion.
- Inventory Effects: sword/shield used later in Dragon fight.

4 room5_dragonBoss

1. Key Check: ensure Key to Room 5 present.
2. Potion Shortcut: instant win if used.
3. Turn Loop: options A/S/P; varied dragon attacks (fire:10, tail:8); health updates.
4. End States: clear win/lose messages, return boolean.

6. Build & Execution

cd build
*rm -rf **
cmake ..
ninja
./game.exe