Dungeon Adventure Game: Design Report

1. Design Overview

This text-based dungeon adventure game implements object-oriented architecture with six key classes:

- Player: Manages character stats, inventory, and actions
- Room: Represents locations connected in a north-south linear structure
- Dungeon: Controls the game environment and room generation
- Enemy: Defines hostile entities with combat capabilities
- Item/Treasure: Provides collectible objects
- Weapon: Implements combat tools with damage values

The game operates on a turn-based loop offering three main actions: movement, combat, and exit. Players navigate through interconnected rooms, battle enemies, and manage limited resources (health and moves).

2. Data Structure Implementation

2.1 Doubly-Linked Room System:

This structure enables O(1) bidirectional navigation between rooms. The player's movement simply traverses pointers, with nullptr checks implementing dungeon boundaries. Each room optionally contains enemies and treasures through composition, allowing runtime removal when enemies are defeated.

2.2 Dynamic Player Inventory:

This array-based implementation provides:

- Memory-efficient storage with O(1) access time

- Dynamic item acquisition and removal

- Capacity limits for game balance

When removing items, the array is kept contiguous by left-shifting elements after the removal point, maintaining data organization.

3. Implementation Highlights

3.1 Dungeon Generation:

This algorithm creates a linear sequence of connected rooms with proper bidirectional links. The current implementation creates a simple north-south dungeon path.

3.2 Combat System:

Combat is implemented through mutual attribute reduction: the player reduces enemy health while enemies counterattack by reducing player health. This simple system creates tension through resource management.

3.3 Movement Mechanics:

Player movement leverages the doubly-linked room structure, with additional logic in the main game loop preventing movement when enemies are present in the current room.

4. Technical Challenges

4.1 Memory Management

The game uses dynamic memory for rooms and inventory, requiring proper cleanup in destructors to prevent memory leaks. The current implementation handles basic cleanup but could be enhanced with smart pointers.

4.2 Game Balance

Balance is achieved through two key limitations:

- **Moves**: Each action consumes one move from a limited pool

- **Health**: Combat reduces player health without automatic regeneration

This creates meaningful choices between exploration and combat, encouraging strategic play.

4.3 Input Validation:

Input validation ensures robust gameplay by preventing invalid actions and enforcing game rules, such as defeating enemies before leaving a room.

5. Future Improvements

The current implementation provides a functional foundation that could be enhanced with:

- Non-linear dungeon layouts using adjacency lists
- More sophisticated enemy behavior
- Expanded item and combat systems
- Save/load functionality

These improvements would maintain the core data structures while enriching gameplay complexity.