Technical Documentation

Roguelike Dungeon Game (C++ Project)

Naveen Prabakar

August 3, 2025

1 Overview

This project implements a classic ASCII roguelike dungeon crawler written in C++. It provides a turn-based, procedurally generated dungeon experience with monsters (AI behaviors), combat, inventory management, sound, and persistent dungeon state via save/load functionality.

2 Key Features

- **Procedural Dungeon Generation:** Randomly builds rooms, hallways, stairs, and places player and monsters.
- **Monster AI:** Implements 16 monster types, each with unique behaviors and pathfinding strategies.
- **Player Interaction:** Movement via numpad/vi-keys, combat, item pickup/equip, and interactive controls using ncurses.
- **Dijkstra Pathfinding:** Computes both tunneling and non-tunneling distances for AI movement.
- **Replayability:** Supports random and deterministic (saved/loaded) dungeons.
- **Inventory and Equipment Management:** Equip/unequip items, manage inventory, gain experience and stats.
- **Custom Monsters/Items:** Reads monster and item definitions from external files in the user's home directory.
- **Sound and Music:** Supports background music and sound effects during various gameplay events.
- **Save/Load:** Saves dungeon state in a custom binary file format; load functionality reconstructs dungeons.
- **User Experience:** Polished controls view, in-game help screens, ASCII art for events, and colored display using ncurses.

3 Project Architecture

3.1 Main Game Loop

- Entry point: `main()` in `hw9.cpp`

- Parses command-line to determine start mode (random, load, save, or combinations).
- Initializes ncurses, music, and seeds RNG.
- Calls `details()` function to configure and run the main game session, including dungeon generation or loading, and monster/player creation.

3.2 Map and Dungeon Generation

- **Grid Layout:** `Cell grid[21][80]`; walls/borders, rooms (.), paths (#), stairs (<,>), unexplored regions.
- `build()`: Initializes the grid, border walls, and unexplored states.
- `place_rooms()`, `connect_rooms()`, `place_stairs()`: Randomly positions and links rooms, and places stairs.
- `place_pc()`: Randomly positions the player in the dungeon.
- `assign_hardness()`: Assigns hardness for pathfinding and tunneling mechanics.

3.3 Monsters and AI

- **MonsterHeap & Monster:** Custom min-heap implementation for turn scheduling.
- 16 monster classes/behaviors (via a 4-bit type), combining psychic, telepathy, intelligence, tunneling (e.g., telepathic intelligent tunneling).
- Each type has a dedicated movement/AI function (e.g., `mon1`, `mon2`, ..., `mon16`).
- Movement choices depend on player visibility, distance maps, and monster state.

3.4 Pathfinding and Movement

- `dijkstra()` and `dijkstra_tunnel()`: Compute non-tunneling and tunneling distance fields for monsters.
- `minheap` module: Manages heap operations for pathfinding and AI turn scheduling.
- `directions()`: Helper for directional movement vectors.

3.5 Player, Combat, and Items

- `Player`: Position, HP, stats, experience, inventory, and equipment slots.
- Inventory constrained to 10 items, supports picking up, dropping, equipping, unequipping, and viewing items.
- `combat()`: Turn-based encounters with ASCII art for monster visualization, randomized damage via Dice.
- Supports leveling up speed and health through experience points.
- Items and monsters are described and loaded from external descriptor files for extensibility.

3.6 Controls/UX

- Movement: Num-pad digits or vi-keys.
- Inventory: 'i' to open, 'd' drop, 'w' equip, 'I' item description, etc.
- Equipment: 'e' to open, 't' to remove.
- Help/Controls: '?' or specialized controls menu.
- Music: Interactive jukebox with multiple tracks, mute, and switches.
- Monster Info: 'm' to view monster list, 'o' for monster description, 'L' for teleportation and monster reveal, etc.
- Exit/Restart: 'q' to quit at any time.

3.7 Persistence (Save/Load)

- `save()`: Stores grid hardness, player position, rooms, stairs, and other dungeon state into a binary file in the home directory.
- `load()`: Reconstructs the dungeon grid and dynamic elements from file (rooms, stairs, player, hardness).
- Supports save after random generation, deterministic save/load flow.

4 Code Structure – Major Modules

- **`hw9.cpp`:** Entry point, all high-level game logic and function orchestration.
- **`minheap.cpp/minheap.h`:** Generic min-heap implementation for pathfinding and monster scheduling.
- **`monsters.cpp/monsters.h`:** Monster structure, heap for monsters, AI movement strategies, and scheduling logic.
- **`Player.h`:** Player state, inventory, equipment management, and stats.
- **Auxiliary:** `Map.h`, `Room.h`, `Up.h`, `Down.h`, `File.h`, `Cell.h`, `Dice.h`, `Item.h`: Data structures for dungeon elements, items, and random dice.

5 Game Flow

1. Launch game; dungeon and player/monster state set via random or loaded map.
2. Game loop alternates monster and player turns, manages events, and updates display.
3. Monsters choose moves based on their AI and the player's location/state.
4. Player explores, fights monsters, picks up and manages inventory, and interacts with items.
5. The game ends if the player dies or upon victory (special monsters/objectives can be defined).
6. On exit, current dungeon state can be saved/loaded for continuity.

6 Extensibility

- **Monster and Item Design:** Extensible via external descriptor files; easy to add new monsters, items, or effects.
- **Room and Dungeon Shapes:** Room number, shapes, corridor logic, and dungeon size can be tuned.
- **Sound/Music:** Supports adding new tracks; music files referenced in-situ.
- **NPCs and Quests:** Possible to build upon the architecture for NPCs, quests, or advanced game logic.

7 Potential Improvements

- Enhanced monster AI with memory, tactical retreat, or grouping.
- Visual field-of-view; fog-of-war for player vision.
- Effects, spells, or ranged combat.
- Further modularization and unit testing.
- Save game versioning and cross-platform compatibility.
- Graphical tiles (beyond ASCII) and mouse support.

8 Compilation and Usage

- **Dependencies:** Requires `g++`, `ncurses`, `mpg123`, and a POSIX-compliant system.
- **To build:**

```
g++ hw9.cpp minheap.cpp monsters.cpp -lncurses -o roguelike
```

- **To run:**

```
./roguelike # Random dungeon
./roguelike --save # Random dungeon, then save
./roguelike --load # Load previous dungeon
./roguelike --load --save # Load, then re-save updated dungeon state
```

- **Note:** Monster/item descriptor files and sound files should exist in specified folders (under home directory).

9 Summary

This roguelike project demonstrates advanced procedural generation, C++ OOP, AI behaviors, and user interface design for a terminal-based game environment. Its architecture is highly modular, making it apt for extension, experimentation, or use as a teaching project.