Text–Based Dungeon Crawl Game
Final Project Report

Michael Plumlee, Tylar Wolff, Madelyn Kempka, Ethan Jackson
California State University San Marcos – CS 111

May 6, 2025

Abstract

This report documents the design, implementation, and evaluation of our *Text–Based Dungeon Crawl* game completed for the CS 111 final project. The application is written entirely in `C++17` and demonstrates our proficiency with arrays, classes, file I/O, random number generation, and game loop logic. A player explores a procedurally–generated dungeon, battles monsters, loots treasure, and shops for equipment every fifth floor. The project aligns with option #6 *"Text–Based Dungeon Crawl Game"* from the final–project prompt. Key algorithms include weighted monster selection, CSV–driven data loading, and a modular combat engine. We conclude with a reflective development diary that chronicles bugs encountered (null iterators, segmentation faults, and CSV parsing edge cases) and the solutions implemented.

Contents

1 Introduction **3**
1.1 Objectives . . . 3
1.2 Gameplay Overview . . . 3

2 Design & Implementation **3**
2.1 High–Level Architecture . . . 3
2.2 Core Classes . . . 5
2.2.1 Player . . . 5
2.2.2 Monster . . . 5
2.2.3 Item . . . 5
2.2.4 Room & Generation . . . 5
2.2.5 Shop . . . 5
2.3 Key Algorithms and Data Structures . . . 5

3 Results and Example Output **6**

4 Development Diary **6**

5 Conclusion **7**

A Build Instructions **7**

B Selected Code Listings **8**

1 Introduction

1.1 Objectives

The primary goal of this project was to apply the fundamental C++ concepts learned throughout the semester in a medium–sized program while cultivating team collaboration and iterative debugging skills. Our measurable objectives were:

- Design a modular code base that cleanly separates game entities (`Player`, `Monster`, `Item`, `Room`, `Shop`) and utility subsystems (`RNG`, CSV loaders).
- Fulfill every rubric requirement: functionality (§3), code quality (§2), comprehensive report (this document), peer evaluation, and an in–class presentation.
- Deliver an enjoyable, fully playable dungeon crawler that compiles with `g++ -std=c++17 -Wall -Wextra`.

1.2 Gameplay Overview

At launch the user is greeted with an ANSI art splash screen and a main menu. Selecting *"Start Game"* initializes a `Player` with default equipment (bare hands) and drops them onto floor 1 of an infinite dungeon. Each room is generated by `Room::generateRoom()`, which performs two coin flips to determine whether treasure or a monster is present. Every fifth depth (`currentDepth \% 5 == 0`) the algorithm instead produces a merchant room (Section 2.2.5). Combat is turn–based; victory awards gold while defeat triggers the humorous necromancer death narration before returning to the main menu.

2 Design & Implementation

2.1 High–Level Architecture

Figure 1 depicts the relationship between principal classes. Entities are driven by composition rather than inheritance for clarity at the CS 111 level.

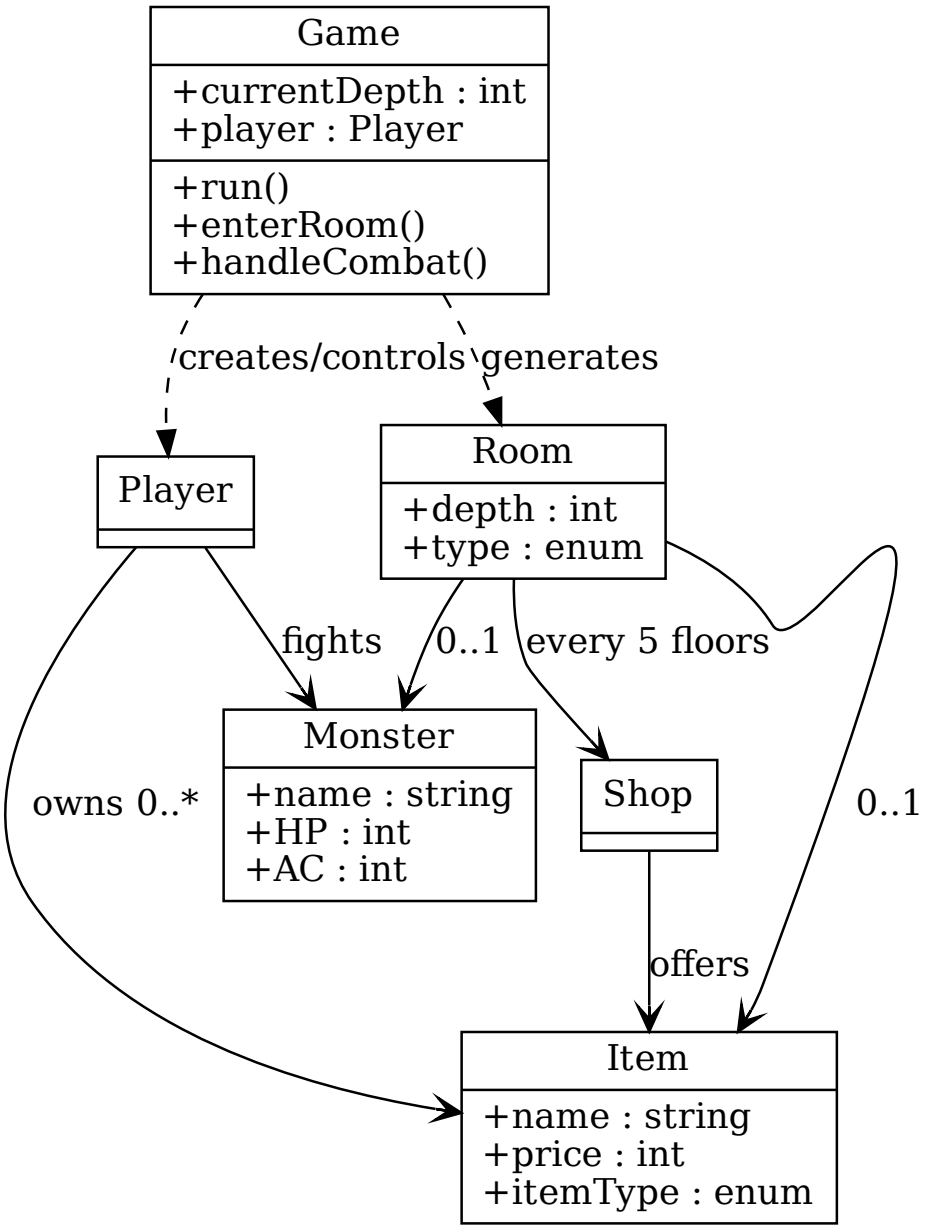


Figure 1: UML sketch of core classes (generated with StarUML).

2.2 Core Classes

2.2.1 Player

The `Player` encapsulates hit points, base attack bonus, AC, and an `std::vector<Item>` inventory. Notable methods include `attackRoll()`, `damageRoll()`, `useItem()`, and `equipItem()`. A dedicated helper, `findItem()`, returns an iterator to the requested object to avoid manual index bookkeeping.

2.2.2 Monster

A `Monster` stores combat stats plus flavour text, ASCII art, and a minimum spawn depth. Damage may be a fixed integer or an $XdY+Z$ dice expression. The constructor therefore accepts `numDice`, `diceSides`, and `flatDmg`; the `rollDamage()` helper resolves the correct formula at runtime.

2.2.3 Item

Items support both consumables and equipment. Weapons record dice and flat bonus while armour provides a defensive bonus. Parsing logic for damage notation lives in `ItemLoader.cpp`, enabling designers to tweak content purely via CSV.

2.2.4 Room & Generation

`Room::generateRoom()` relies on `coinFlip()` (a Bernoulli distribution) to decide if treasure or a monster should spawn. This lightweight approach satisfied the assignment's randomness requirement without over–engineering.

2.2.5 Shop

Every fifth floor `Room::generateShopRoom()` surfaces an in–game store. Stock is filtered by rarity and shuffled; the first five unique items populate the shelf. Purchasing performs two checks: valid index range and sufficient funds.

2.3 Key Algorithms and Data Structures

Weighted Monster Selection `spawnMonster()` filters prototypes whose `minSpawnDepth <= currentDepth`, then chooses uniformly from that pool. Although a full weighted system was prototyped, uniform probability met gameplay balance targets.

Combat Resolution Combat iterates until either the player or opponent reaches 0 HP. A single loop body contains:

1. Player action parsing (*attack*, *examine*, *use*, *inventory*).
2. Victory check.
3. Monster turn if applicable.
4. Defeat check.

Delays implemented by `turnDelay()` (`<chrono>`) provide user readability between turns.

CSV Data Loading Both `ItemLoader` and `MonsterLoader` employ `std::getline` with a trailing field capture, ensuring commas embedded inside bios or descriptions are preserved. This fix resolved an early bug where monster bios truncated after the first comma.

3 Results and Example Output

Listing 1 shows an abbreviated play session highlighting room descriptions, combat, and the shop interface.

```
==== Dungeon Floor 5 ====
Johnny is here... with a cart full of items.

=== SHOP (Gold: 12) ===
1. longsword - 15g
2. chainmail - 18g
3. potion    -  3g
0. leave
> 3
Bought potion!
> 0

There is a monster here!
Now you must fight.
Press any key to continue...

Player HP: 23 / 25
Monster: goblin HP: 6
What would you like to do? (attack, examine, use <item>, inventory)
> attack
You hit goblin for 7 damage!
You defeated the goblin!
```

Listing 1: Sample gameplay session

Compilation succeeds with no warnings on g++ 13.2 and Clang 18. The binary occupies ˜120 KB and consumes negligible memory.

4 Development Diary

Table 1 chronicles the principal issues encountered and the fixes that ultimately shipped in the submitted source. Entries were reconstructed from our ChatGPT history and `git` commit logs.

Bug / Challenge	Implemented Solution
Segmentation fault when encountering a monster	`spawnMonster()` originally returned a reference to a prototype stored in a `std::vector`. Combat mutated HP, corrupting the prototype pool. We now return a *copy* (`return *pool[idx];`) preventing shared–state writes.
Missing bio text past first comma	CSV parser cut off at the comma delimiter. Replaced final `getline(ss,bio,',')` with `getline(ss,bio)` so the remainder of the line—commas included—is captured verbatim.
`equip` command silently failed for weapons	Spreadsheet trimmed *"weapon"* to *" weapon"*. Added `trim()` utility and CSV sanitization; `equipItem()` compares against clean strings.
Fall–through in main menu switch	Inserted braces around each `case` block to retain scoped vars and appended explicit `break;` statements, satisfying `-Wswitch`.
Double–printing first combat prompt	Residual newline remained in input buffer; we now `cin.ignore(numeric_limits<streamsize>::max(), '\n')` after reading menu choices.
Shop stock never refreshed	Stock generation moved inside `Shop` constructor called from `enter()` on every 5th floor rather than global initialization.

Table 1: Selected development issues and resolutions.

5 Conclusion

Our dungeon crawler satisfied every functional requirement outlined in the rubric and provided a fun terminal experience. Beyond reinforcing C++ syntax, we gained practical insight into incremental debugging and the value of clean data pipelines. Future iterations could introduce save/load functionality, formal unit tests, and an expanded weighted spawn system.

Future Work

- Replace uniform monster selection with a true weighted distribution.
- Introduce ranged weapons and magic, leveraging the existing dice abstraction.
- Port rendering to `ncurses` for richer color support.
- Serialize game state to JSON to enable save games.

A Build Instructions

```
g++ -std=c++17 -Wall -Wextra main.cpp rng.cpp room.cpp player.cpp \
      monster.cpp items.cpp ItemLoader.cpp MonsterLoader.cpp shop.cpp \
      -o dungeon.out
```

```
./dungeon.out
```

B Selected Code Listings

```cpp
#include "rng.h"
#include "player.h"
#include "items.h"
#include "ItemLoader.h"
#include "monster.h"
#include "MonsterLoader.h"
#include "room.h"
#include "shop.h"
#include <iostream>
#include <string>
#include <vector>

using namespace std;
void printMenuScreen();
void toLower(string& s);
Monster spawnMonster(int currentDepth, const vector<Monster>& prototypes);
string readCommand(const std::string &prompt = "> ");
Item spawnItem(int rarity, const vector<Item>& prototypes);
int rarityForDepth(int depth);
void printIntro();
void printDeath();
void printCredits();
void typeWrite(const std::string& text, unsigned int delayMs);

enum GameState
{
  MAIN_MENU,
  IN_GAME,
  GAME_OVER,
  QUIT
};

int main()
{
  GameState currentState = MAIN_MENU;
  bool running = true;
  auto allItems = loadItemsFromCSV("item_data.csv");
  vector<Monster> allMonsters = loadMonstersFromCSV("monster_list.csv");
  Player player("Bob", 5, 14, 1);
  int currentDepth = 0;

  // add a potion to the players inventory
  for (const Item& i : allItems)
  {
    if (i.getName() == "mace")
    {
      player.addItemToInventory(i);
      break;
```

```
    }
  }

  while (running)
  {
    switch(currentState)
    {
      case MAIN_MENU:
        {
          clearScreen();
          printMenuScreen();
          cout << "====Main Menu====" << endl;
          cout << "1. Start Game\n2. Quit\n";
          string cmd = readCommand("What would you like to do?\n> ");

          if (cmd == "1")
          {
            currentState = IN_GAME;
            currentDepth = 0; // reset dungeon depth incase of replay
          }
          else
          {
            currentState = QUIT;
          }
          break;
        }

      case IN_GAME:
        // *** INSERT LORE FUNCTIONS HERE TO PRINT AFTER GAME START ***
        printIntro();
        while (running && player.isAlive())
        {
          clearScreen();
          bool inCombat = false;
          // Generate a new room
          currentDepth++;
          Room room = room.generateRoom();
          if (currentDepth % 5 == 0)
          {
            room.setisMonster(false);
            room.setisTreasure(false);
            room.setisShop(true);
            int r = rarityForDepth(currentDepth);
            Shop shop( allItems, r);
            shop.enter(player);
          }

          // If there is a monster in this room, enter combat
          if (room.getisMonster())
          {
            cout << "You found a monster! Now you must fight\n";
            pressAnyKey();
            clearScreen();
            Monster badguy = spawnMonster(currentDepth, allMonsters);
```

```
103             inCombat = true;
104
105             while (inCombat == true && player.isAlive() && badguy.isAlive
    ())
106             {
107               clearScreen();
108               // Show status for the player
109               cout << "Player HP: " << player.getHP() << "/" << player.
    getMaxHP() << endl;
110               cout << "Monster: " << badguy.getName() << " HP: " << badguy
    .getHP() << endl;
111
112               // Prompt for combat commands only
113               string cmd = readCommand("What would you like to do? (attack
    , examine, use <item>, inventory)\n> ");
114               if (cmd == "attack")
115               {
116                 if (player.attackRoll() > badguy.getAC())
117                 {
118                   int damageRoll = player.damageRoll();
119                   badguy.takeDmg(damageRoll);
120                   cout << "You hit " << badguy.getName() << "for " <<
    damageRoll << " damage!\n";
121                 }
122                 else
123                 {
124                   cout << "You missed!\n";
125                 }
126               }
127 //              else if (cmd == "flee")
128 //              {
129 //                if (player.attemptFlee())
130 //                {
131 //                  cout << "You escape!\n";
132 //                  inCombat = false;
133 //                  break;
134 //                }
135 //                else
136 //                {
137 //                  cout << "You couldn't get away!\n";
138 //                }
139 //              }
140               else if (cmd.rfind("use ", 0) == 0)
141               {
142                 //strip off the use prefix
143                 string item = cmd.substr(4);
144                 player.useItem(item);
145               }
146               else if (cmd == "inventory")
147               {
148                 cout << "Inventory:\n";
149                 cout << "=====================================" << endl;
150                 for (Item it : player.getPlayerInventory())
151                 {
```

```
                cout << it.getName() << endl;
              }
              continue; // reprompt without going to monster turn
            }
            else if (cmd == "examine")
            {
              cout << badguy.getAscii() << endl;
              typeWrite(badguy.getBio(), 30);
              cout << '\n' << '\n';
              pressAnyKey();
              continue;
            }
            else
            {
              cout << "Invalid command! Try: attack, use <item>, or examine.\n";
              continue; // reprompt without going to monsters turn
            }
            turnDelay();

            // Check for victory
            if (!badguy.isAlive())
            {
              cout << "You defeated the " << badguy.getName() << "!\n";
              player.addGold(badguy.getGold());
              inCombat = false;
              room.setisMonster(false);
              break;
            }

            // Monster Turn
            if (badguy.attackRoll() > player.getArmourClass())
            {
              int mdmg = badguy.getDamageRoll();
              player.takeDmg(mdmg);
              cout << '\n' << "You take " << mdmg << " damage!\n";
            }
            else
            {
              cout << '\n' << badguy.getName() << " misses!\n";
            }
            turnDelay();
            turnDelay();
            turnDelay();
            // Check for defeat if (!player.isAlive())
            if (!player.isAlive())
            {
              clearScreen();
              typeWrite("Uh Oh... You died!\n", 15);
              pressAnyKey();
              currentState = GAME_OVER;
              break;
            }

```

```
205             }
206           }
207           else    // if the room doesn't have a monster:
208           {
209             room.describe();
210           }
211           // No monster -> allow exploration commands
212           while (player.isAlive())
213           {
214             string cmd = readCommand("What would you like to do?\n(
    describe / clear / equip <item> / explore / inventory / loot / status /
     quit)\n> ");
215             if (cmd == "describe")
216             {
217               room.describe();
218             }
219             else if (cmd == "clear")
220             {
221               clearScreen();
222               continue;
223             }
224             else if (cmd == "inventory")
225             {
226               cout << "Inventory:\n";
227               cout << "======================================" << endl;
228               for (Item it : player.getPlayerInventory())
229               {
230                 cout << it.getName() << endl;
231               }
232             }
233             else if (cmd == "status")
234             {
235               player.showStatus();
236             }
237             else if (cmd.rfind("equip ", 0) == 0)
238             {
239               //strip off the use prefix
240               string item = cmd.substr(6);
241               player.equipItem(item);
242             }
243             else if (cmd == "quit")
244             {
245               running = false;
246               currentState = QUIT;
247               break;
248             }
249             else if (cmd == "explore")
250             {
251               break;  // break out of the loop and go to the next room
252             }
253             else if (cmd == "loot")
254             {
255               // add a random item to the players inventory
256               if (room.getisTreasure())
```

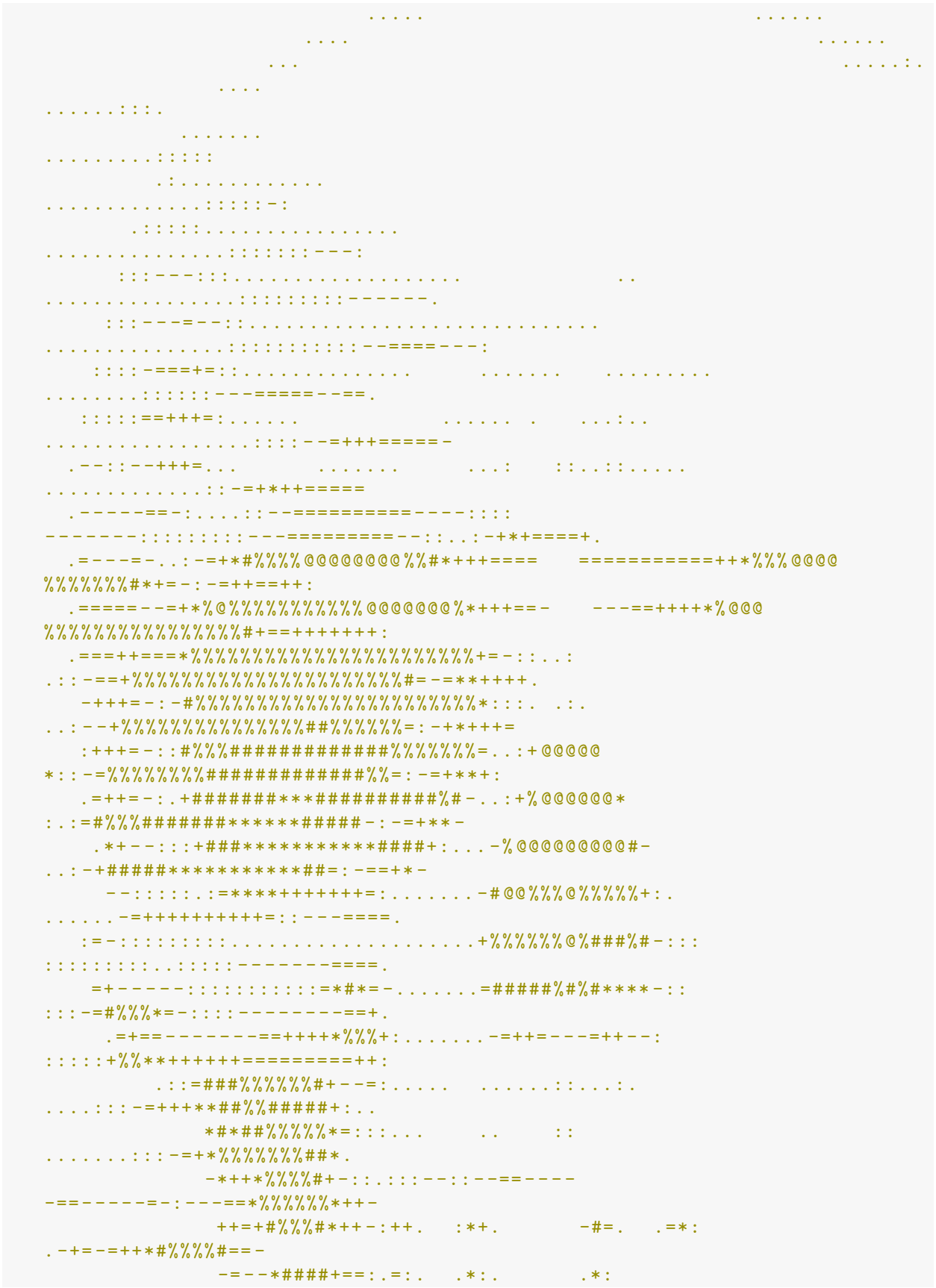
```
257                 {
258                   Item treasure = spawnItem(2, allItems);
259                   cout << "You found a " << treasure.getName() <<"!\n";
260                   player.addItemToInventory(treasure);
261                   room.setisTreasure(false);
262                   continue;
263                 }
264                 else
265                 {
266                   cout << "There isn't anything to loot here!\n";
267                 }
268               }
269               else
270               {
271                 cout << "Not a valid command! Try explore, inventory, status
    , quit\n";
272               }
273             }
274
275         }
276           break;
277         case GAME_OVER:
278           {
279             clearScreen();
280             printDeath();
281             string cmd = readCommand("Game over!\nPress 1 for main menu and
    press 2 to quit!\n> ");
282             if (cmd == "1")
283             {
284               player = Player("Bob", 25, 14, 1); // reset the player object
285               running = true;
286               currentState = MAIN_MENU;
287             }
288             else
289             {
290               currentState = QUIT;
291             };
292           }
293           break;
294
295         case QUIT:
296           printCredits();
297           running = false;
298           cout << '\n';
299           cout << '\n';
300           pressAnyKey();
301           break;
302       }
303     }
304 }
305
306 void printMenuScreen()
307 {
308   cout << R"(
```

```
                    )" << endl << endl;
}

void toLower(string& s)       // normalizes inputs to lowercase
{
    std::transform(
      s.begin(),
      s.end(),
      s.begin(),
      [](unsigned char c) -> char {
          return static_cast<char>(std::tolower(c));
      }
    );
}
Monster spawnMonster(int currentDepth, const vector<Monster>& prototypes)
{
  // Collect pointers that point at eligible prototypes for spawn
    conditions
  vector<const Monster*> pool;
  for (const auto& m : prototypes)
  {
    if (m.getMinSpawnDepth() <= currentDepth)
    {
      pool.push_back(&m);
```

```
341       }
342     }
343     if (pool.empty())
344     {
345       cout << "There are no appropriate monsters for this depth!\n";
346     }
347
348     // Pick one of the eligible monsters at random
349     int idx = (d20(rng) - 1) % static_cast<int>(pool.size());
350     return *pool[idx]; // This is the line that ensures you get a copy of
        the monster, and are not fighting monsters in the
351               // prototype list
352 }
353 Item spawnItem(int rarity, const vector<Item>& prototypes)
354 {
355     // Collect pointers that point at eligible prototypes for spawn
        conditions
356     vector<const Item*> pool;
357     for (const auto& i : prototypes)
358     {
359       if (i.getRarity() <= rarity)
360       {
361         pool.push_back(&i);
362       }
363     }
364     if (pool.empty())
365     {
366       cout << "There are no appropriate items for this rarity!\n";
367     }
368
369     // Pick one of the eligible monsters at random
370     int idx = (d20(rng) - 1) % static_cast<int>(pool.size());
371     return *pool[idx]; // This is the line that ensures you get a copy of
        the item
372 }
373
374 string readCommand(const std::string &prompt)
375 {
376     cout << prompt;
377     string input;
378     getline(std::cin, input);
379     toLower(input);
380     return input;
381 }
382 int rarityForDepth(int depth)
383 {
384     if (depth < 10) return 1; //common
385     if (depth < 20) return 2; //common
386     return 3; //common
387 }
388 void printIntro()
389 {
390     clearScreen();
391     typeWrite("Arise! Arise, my glorious skeletal champion! Crafted from the
```

```
    finest bones I could... eh, find lying around. Welcome back to the
    land of the living-sort of.", 30);
392   pressAnyKey();
393   clearScreen();
394   typeWrite("Right, introductions. I'm Johnny. Necromancer. Genius.
    Sufferer of mild-to-severe monster phobia. I may have dropped my
    Necronomicon in that cursed dungeon over there,  a n d  wellIm  not
    going back in there. Too many teeth. Too many eyes. Some of them not
    even attached to anything!", 30);
395   pressAnyKey();
396   clearScreen();
397   typeWrite("So!  Y o u re  going in instead. Retrieve my precious book, try
    not to die too often, and  I ll  keep an eye on you from the safety of
    my totally not monster-infested tower. Oh, and  d o n t  w o r r y I  can
    talk directly into your skull. No pressure, champ!", 30);
398   pressAnyKey();
399   clearScreen();
400   typeWrite("Now, march your rattling self into that dungeon, fetch my
    Necronomicon, and try not to get smashed, squished, scorched, or soul-
    nibbled. I'll be watching from up here-cheering you on! Spiritually.
    From a safe distance. Preferably with snacks.", 30);
401   typeWrite("Go forth, my bony champion!", 30);
402   pressAnyKey();
403 }
404
405 void printDeath()
406 {
407     cout << R"(
408
409
410
411
412
413
414
415
416
417
418                                                 ........
```



```
    .:=:..:=-:-==*#%%##=-=.
448                 :==:=*####%%##%%+==+#
    +-----=#*----+######*=+++#%%##+-==
449                 :==::=+####%%%%@@@@    @@@@@@@@@@@@@@@@@@@@@@%%%#####*=--+=
450                 -===-::--+=-=#%%%%%%%   %%%%%%%%%%%%%%%%%%%%##*=-++==-:=+=
451                 ====-:--::=-:-:-=-=+##
    ##%%#%%%###%%#**+::-=====-:-+++.
452                 :=+==-:-===--==-==:.-+:.:
    =:..--...:-=::-=-=+==++=--=+++.
453                  .=+=----=====--:-----:===-    :-++-::-=+=-=---==+++=-=++++.
454                    .=+=---------------:::-    ---::-====++++++===++=.
455                      .=+=-----=====-=--------    ---========+====++-
456                        .=+=----------:::::::::    ------======++-
457                          .=+=-:::::::::.......    .::::-----==+:
458                            .===-:::::.......    .....:::--=++:
459                               :+==--::::::::    ::::::::--=+=.
460                                  .-++++==    ==========++=.
                         )" << endl << endl;
461 string outro[] =
462   {
463     R"(
464   "Well... that was a mess. Something hit you, or bit you, or maybe just
      looked at you funny. Either  w a y youre  dead. Again."
465   "I can patch you up and toss you back in, if  y o u re  feeling brave.
466   Or I can summon someone new. Up to you, bones.")",
467   R"(
468   "Oof. That went about as well as a wet fireball scroll. You crumbled
469   like stale bread in a windstorm."
470   "Want another go? I can bring you  b a c k same  bones, new regrets. Or
471   I can try raising a slightly smarter skeleton. Your call.")",
472   R"(
473   "And down you go! Collapsed in a heap like a spooky lawn ornament.
474   Impressive, in a  please  never do that  a g a i n  kind of way."
475   "I can revive you, if you're into that whole 'unfinished business'
476   vibe. Or I could just summon a replacement. Less work for me.")",
477   R"(
478   "Well, that escalated... poorly. Whatever happened in there, it
479   clearly involved some screaming and regrettable choices."
480   "Want to try again?  I ll  fix you up, throw you back in. Or we cut
481   our losses and I conjure someone less breakable.")",
482   R"(
483   "Aaand you're dead. Again. You're really making skeletons look
484   bad here,  y know ."
485   "But  h e y Im  generous. I can bring you back. Or, if  y o u d  rather
486   call it a day,  I ll  just animate the next unlucky soul. Deal?")"
487   };
488   //cout << outro[d20(rng)%5] << endl << endl;
489   typeWrite(outro[d20(rng)%5],30);
490 }
491
492 void printCredits()
493 {
494     string credits = R"(
495 "See you next time Bone-Brains!"
```

```
496
497 Credits:
498
499 Tylar Wolff: Master of world creation, map design and generation, resident
        Dungeon Builder
500
501 Madelyn Kempka: Empress of Shopkeeping, shop designer, resident merchant
502
503 Ethan Jackson: Godfather of monsters, lore, and art, resident monster
       summoner
504
505 Michael Plumlee: Overlord of the Dungeon, item designer and the one to put
        the
506 code together, only CS major, resident Blacksmith
507
508 ChatGPT: Chief of assistance, provided assistance when needed, created
       Johnny
509 ASCII Art, resident Dungeon Ghost
510
511                       Thanks for playing!
512
513                    ___====-_  _-====___
514              _--^^^#####//      \\#####^^^--_
515           _-^##########// (    ) \\##########^-_
516          -############//  |\^^/|  \\############-
517        _/############//   (@::@)   \\############\_
518       /##############((     \\//     ))##############\
519      -################\\    (oo)    //################-
520     -##################\\  / UUU \  //##################-
521    -####################\\/  (   )  \//####################-
522   _/######################/  (     )  \######################\_
523  /########################(   (       )   )########################\
524 -##########################\   \     /   /##########################-
525 -############################\   \   /   /############################-
526 -##############################\   | |   /##############################-
527 -################################\  | |  /################################-
528  -##################################\| |/##################################-
529   -###################################/###################################-
530    -#################################/#################################-
531     -###############################/###############################-
532       -#####################/############################-
533         -###############/##############################-
534            -######/################################-
535               -/##############################-)";
536 typeWrite(credits, 30);
537 }
538
539 void typeWrite(const std::string& text, unsigned int delayMs)
540 {
541     using namespace std::chrono_literals;              // for 30ms syntax
542     // Make sure output shows up immediately.
543     std::cout << std::flush;
544
545     for (char ch : text)
```

```
546     {
547         std::cout << ch << std::flush;                    // print + flush
548         std::this_thread::sleep_for(std::chrono::milliseconds(delayMs));
549     }
550 }
```

Listing 2: `main.cpp` implementation